SPIAL: UMA FERRAMENTA DE APOIO AO APRENDIZADO DE MELHORIA DE PROCESSOS DE SOFTWARE

DANIELA CRISTINA CASCINI KUPSCH

SPIAL: UMA FERRAMENTA DE APOIO AO APRENDIZADO DE MELHORIA DE PROCESSOS DE SOFTWARE

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

ORIENTADOR: RODOLFO SÉRGIO FERREIRA DE RESENDE

Belo Horizonte

Setembro de 2012

DANIELA CRISTINA CASCINI KUPSCH

SPIAL: A TOOL FOR SOFTWARE PROCESS IMPROVEMENT TRAINING

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Advisor: Rodolfo Sérgio Ferreira de Resende

Belo Horizonte

September 2012

Kupsch, Daniela Cristina Cascini.
K96s SPIAL: Uma Ferramenta de Apoio ao Aprendizado de Melhoria de Processos de Software / Daniela Cristina Cascini Kupsch. — Belo Horizonte, 2012.
xxi, 201 f. : il. ; 29 cm.

Tese (doutorado) — Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Rodolfo Sérgio Ferreira de Resende.

1. Computação - Teses. 2. Engenharia de software - Teses. 3. Simulação (Computadores digitais) - Teses. I. Orientador. II. Título.

CDU 519.6*32(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

SPIAL: uma ferramenta de apoio ao aprendizado de melhoria de processos de software

DANIELA CRISTINA CASCINI KUPSCH

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. RODOLFO SÉRGIO FERREIRA DE RESENDE - Orientador
Departamento de Ciência da Computação - UFMG

PROFA. CLAUDIA MARIA LIMA WERNER
Centro de Tecnologia - UFRJ

PROF. CLARINDO ISAÍAS P. DA SILVA E PÁDUA
Departamento de Ciência da Computação - UFMG

PROF. EDUARDO MAGNO LAGES FIGUEIREDO
Departamento de Ciência da Computação - ICEx - UFMG

PROFA. FLÁVIA MARIA SANTORO
Departamento de Informática Aplicada - UNIRIO

PROF. JOSÉ LUIS BRAGA
Departamento de Informática - UFV

Belo Horizonte, 17 de setembro de 2012.

Acknowledgments

First of all, I would like to thank my supervisor, Rodolfo. He provided me excellent guidance in all these years, ensuring the quality of this thesis. Rodolfo always had confidence in my work, pushing me up further into my research. I always enjoyed our discussion about different topics of our lives. Rodolfo explained to me how the scientific community works and I really appreciate. Many thanks for all his knowledge, attention and patience.

Studying and living in Amsterdam was a completely new experience to me. I consider myself lucky in living in such a lovely city. Many thanks to Hans van Vliet who generously accepted my request to study at Vrije University. Also thanks to all colleagues who made me feel like home, Rahul Premarj, Antony Tang, Viktor Clerc, Christina Mantelli, Maryam Razavian, Adam Vanya, Qing Gu, and Patricia Lago.

I was very fortunate to work with highly qualified people: Rodrigo who implemented the FASENG framework, Soraia who helped me with the literature review, and Vitor who was always helpful with hints and feedback about my research. I am very grateful for their help. Also many thanks to the people who helped me with the Systematic Literature Review. The list is extensive and also the range of their contribution.

I would like to thank my sweet husband who encouraged me with his love and patience. Once and again, he had to put up with me. Without his support I doubt I could finish this research. Thank you Amore, *Ich liebe dich*! My lovely daughter, Lisa, who is always in a good mood. Also many thanks to my father, mother, father-in-law, mother-in-law who supported me during all these years. I love you all.

Lastly, I would like to thank all my friends and colleagues. These last few years would not have been as fun and enjoyable without you. A very special thanks to Kelly and Claudia who helped me out with Lisa.

Resumo

Software é um artefato complexo e seu desenvolvimento é ainda mais complexo. Nos últimos anos, a complexidade do software aumentou significativamente. Atualmente, software está em todas as partes e o papel desempenhado por ele é mais importante do que nunca. A sua influência na sociedade e na economia tornou-se inquestionável. Contudo, os mercados exigem redução de custos e do prazo necessário para a sua produção. Essa combinação de complexidade e restrições do mercado pode impactar significativamente na qualidade final do produto.

Uma forma de minimizar os possíveis problemas na qualidade do produto é melhorar a preparação da força de trabalho e isto leva em consideração o aprendizado de Engenharia de Software. Atualmente, os cursos de Engenharia de Software raramente abordam a prática de determinados conhecimentos. Normalmente, um curso consiste de aulas teóricas e do desenvolvimento de um projeto de software de pequeno porte. Entretanto, as metodologias utilizadas tanto nas aulas quanto nos projetos falham em prover um conhecimento mais abrangente dos processos de desenvolvimento de software necessários para a aplicação em um ambiente industrial, em particular, os que não estão diretamente relacionados aos processos de software.

Com o objetivo de proporcionar aos alunos uma experiência mais realista dos processos de desenvolvimento de software, dentro do ambiente acadêmico, nós utilizamos a simulação. A simulação pode ser uma ferramenta efetiva para a melhoria do aprendizado e entendimento de assuntos complexos. Em particular, nós acreditamos que um ambiente de simulação de processos de software traz para a Engenharia de Software os mesmos benefícios da sua utilização em outros domínios como, por exemplo, na aeronáutica com os simuladores de voos. Ao se utilizar um simulador, as dificuldades podem ser planejadas e experimentadas, sem grandes riscos.

O objetivo deste trabalho é melhorar o aprendizado de Engenharia de Software, utilizando uma simulação estruturada que permite replicar a realidade de uma organização, o que, na maioria das vezes, não é possível apresentar para os estudantes durante um projeto de uma disciplina. Nosso trabalho aborda como um jogo de simu-

lação de Engenharia de Software, especificamente, um jogo de simulação de Melhoria de Processos de Software, ensina as melhores práticas de Engenharia de Software para os estudantes de um curso introdutório de Engenharia de Software. A fim de investigar este assunto, nós desenhamos e desenvolvemos SPIAL, um jogo de simulação gráfico, interativo e personalizável. A avaliação deste jogo foi realizada através de um experimento piloto e uma inspeção utilizando o método de Inspeção Semiótica. Os aspectos educacionais abordados no experimento incluem a capacidade dos alunos em compreender, lembrar e aplicar conceitos de Engenharia de Software no contexto de uma iniciativa de Melhoria de Processos de Software baseada no CMMI.

Nossa avaliação sugere que SPIAL é uma metodologia complementar e útil para o ensino de Melhoria de Processos de Software e de conceitos de Engenharia de Software. Na opinião dos estudantes, o jogo é agradável. Eles relataram que se divertiram ao jogar. Com o objetivo de tornar a experiência mais educacionalmente efetiva, algumas pesquisas futuras, identificadas durante as avaliações, incluem a incorporação no jogo de outros fenômenos de Melhoria de Processos de Software e a melhoria do desenho da interface.

Palavras-chave: Processos de Software, Jogos de Simulação, Melhoria de Processos de Software.

Abstract

Software is a complex artifact and its development is even more complex. Comparing with the past, this complexity has increased enormously. Today software is spread out everywhere and its role is more important than ever. The economic importance of software and the society dependence on it is unquestionable. In this scenario, markets require reduced costs and short time of software production. This combination of complexity and market restrictions can cause a great amount of problems in the quality of the final product.

One way to minimize these problems is to improve the work force preparation and this leads to the basic Software Engineering education. Currently, Software Engineering courses do not support students into the practice of some skills. Typically, a course consists of theoretical lectures and a small software development project. However, the methodologies based on such lectures and projects fail to provide a broad knowledge of software development processes necessary to their application in an industrial environment, in particular, the ones not directly related to Software Engineering processes.

In order to provide students with a more realistic experience of software development processes within the academic environment, we use simulation. Simulation can be an effective tool for enhancing learning and understanding of complex subjects. In particular, we believe that a simulation environment for Software Engineering processes can bring the same benefits observed in other domains, like airlines training. By using a simulator, difficulties can be planned and experienced without great risks.

Therefore, the aim of this research is to improve the Software Engineering education in dealing with the complexity of providing to students experiences that resemble more closely those in industry. We present how a Software Engineering simulation game, specifically, a Software Process Improvement simulation game, can teach best practices of Software Engineering to students. In order to investigate this subject, we designed and developed SPIAL, a graphical, interactive, customizable, simulation game and evaluated it through a pilot experiment and an inspection using the Semiotic Inspection Method. The educational aspects addressed in the experiment included

the capability of students to understand, remember and apply Software Engineering concepts in the context of a CMMI software process improvement initiative.

Our evaluation suggests that SPIAL is a useful complementary approach to teaching SPI and Software Engineering concepts. Students found it quite enjoyable and they had fun during the game play. In order to make the experience more educationally effective, future researches have been identified during the evaluations, such as incorporating other SPI phenomena and enhancing the interface design.

Keywords: Software Process, Simulation Game, Software Process Improvement.

List of Figures

2.1 Input-Process-Outcome game model (extracted from Garris et al. [2002]). . 20

3.1 SimSE main interface. 32

4.1 SLR steps. 46
4.2 Research methodology. 51
4.3 Data analysis method. 51
4.4 Analysis methodology. 51
4.5 Data collection method. 52
4.6 Distribution of organization's size. 52
4.7 Geographical distribution of organizations. 53
4.8 Business model. 53
4.9 Approaches used by each study. 54
4.10 System Context Processes. 55
4.11 Software Specific Processes. 56
4.12 Distribution of the improvement percentage across measurements. 72
4.13 Distribution of the number of improvement percentages in each category. . 72

5.1 Educational simulation games design (adapted from Martin [2000]). 85
5.2 FASENG structural elements. 89
5.3 SPIAL introductory screen. 99
5.4 SPIAL Graphical User Interface. 100
5.5 Stakeholders communication. 101
5.6 SPIAL Analysis Tab sheet. 101
5.7 Status information. 103
5.8 Communication details between simulation engine and simulator. 104
5.9 Destroyer hierarchy and interfaces. 106
5.10 Trigger hierarchy and interfaces. 107
5.11 Rules hierarchy and interfaces. 108

5.12 First SPIAL Prototype. 109
5.13 Average score results for each knowledge level. 122
5.14 Results on Dep.6 Appropriateness (n=11). 123
5.15 Results on Dep.7 Enjoyable (n=11). 123
5.16 Results on Dep.7 Fun (n=11). 123
5.17 Results on Dep.8 subjective learning perspective (n=11). 124

D.1 Defects detected after process improvement per month. 194

List of Tables

2.1 Terminology related to Capability and Maturity Levels (extracted from [CMMI, 2010]). . . . 14
2.2 Computer-based Simulation Games. . . . 24

4.1 Research questions and motivations. . . . 42
4.2 Inclusion and exclusion criteria. . . . 45
4.3 Quality assessment criteria. . . . 47
4.4 Distribution of studies by source. . . . 50
4.5 Data analysis method according to the organization size. . . . 53
4.6 Reasons for launching an SPI initiative. . . . 69
4.7 SPI measurements. . . . 71
4.8 Quality of data collection and analysis. . . . 74

5.1 Cognitive levels of the revised version of the Bloom's Taxonomy (adapted from Gresse von Wangenheim et al. [2009]). . . . 115
5.2 Experimental Variables. . . . 116
5.3 Schedule of the experiment. . . . 118
5.4 Overview of participants' personal characteristics. . . . 120
5.5 Descriptive Analysis (Dep.3, Dep.4 and Dep.5 variables). . . . 121
5.6 Descriptive Analysis (Dep.1, Dep.2, Dep.6, Dep.7 and Dep.8 variables). . . . 121

A.1 Software Engineering Games. . . . 168

C.1 Data extraction categories. . . . 178
C.2 Selected primary studies. . . . 179
C.3 Quality scores. . . . 183
C.4 Comparison of related SLRs. . . . 184

Contents

Acknowledgments **ix**

Resumo **xi**

Abstract **xiii**

List of Figures **xv**

List of Tables **xvii**

1 Introduction **1**
- 1.1 Research Questions 4
- 1.2 Approach 5
- 1.3 Scope 6
- 1.4 Contributions 7
- 1.5 Thesis Outline 8

2 Background **11**
- 2.1 Software Process 11
- 2.2 Software Process Improvement 12
- 2.3 Software Process Simulation Model (SPSM) 14
- 2.4 Simulation Games 16
  - 2.4.1 Simulation Games Design 18
  - 2.4.2 Literature Review of Software Engineering Simulation Games . 20
- 2.5 Conclusion 25

3 Semiotic Analysis **27**
- 3.1 Background 28
- 3.2 Analyzing the Designers' Message 31
- 3.3 Communicability Evaluation 34

3.4 Triangulation . . . 35
3.5 Conclusion . . . 36

4 SPI Initiatives and Results **39**
4.1 Background . . . 40
4.2 Research Method . . . 41
4.2.1 Research Questions . . . 41
4.2.2 Case Study Roles . . . 42
4.2.3 Data Sources . . . 43
4.2.4 Search Criteria . . . 43
4.2.5 Study Selection . . . 44
4.2.6 Quality Assessment . . . 46
4.2.7 Data Extraction and Synthesis . . . 47
4.3 Literature Analysis . . . 48
4.3.1 Methodological Quality . . . 48
4.3.2 Quantitative Analysis . . . 49
4.3.3 Qualitative Analysis . . . 56
4.4 Discussion . . . 68
4.4.1 Research Questions . . . 68
4.4.2 Guidance for Reporting SPI Initiatives . . . 73
4.5 Related Work . . . 75
4.6 Limitations . . . 77
4.6.1 Construct Validity . . . 78
4.6.2 Reliability . . . 79
4.6.3 Internal Validity . . . 79
4.6.4 External Validity . . . 80
4.7 Conclusion . . . 80

5 SPIAL **83**
5.1 Background . . . 84
5.2 SPIAL Goals . . . 85
5.3 Early Decisions . . . 87
5.4 FASENG . . . 88
5.4.1 Simulation Model . . . 92
5.4.2 Simulation Engine . . . 98
5.4.3 Simulator . . . 103
5.5 Evaluation . . . 105

5.5.1 Semiotic Analysis 109
5.5.2 Pilot Experiment 114
5.5.3 Discussion 126
5.6 SPIAL x Software Engineering Simulation Games 127
5.7 Conclusion 128

6 Conclusion **131**
6.1 Research Achievements 131
6.2 Lessons Learned 133
6.3 Future Work 134

Bibliography **137**

A Software Engineering Simulation Games **167**

B Software Engineering Rules **171**

C Systematic Literature Review - Software Process Improvement **177**

D Questionnaires used for Validating SPIAL **187**

Chapter 1

Introduction

The creation of complex software products that satisfy the constraints of quality, cost and schedule requires the collaboration of many software engineers whose preparation lies primarily in Computer Science departments and their Software Engineering courses [Colomo-Palacios et al., 2012]. Typically, Software Engineering courses provide an introductory view of Software Engineering principles and techniques [Ludi and Collofello, 2001]. These courses rarely provide an opportunity to explain in details the practices and techniques related, for example, to project management, quality assurance, and client's requirements understanding. Therefore, there is a gap between learning by studying (at school) and learning by doing (at work) in this area [Moreno et al., 2012; Aasheim et al., 2010]. This gap is typical of academic courses with time and size constraints on course projects [Ludi and Collofello, 2001].

For a software engineer preparation, the Software Engineering project should bring together not just basic Software Engineering principles, but also knowledge acquired in other areas [Claypool and Claypool, 2005; Barzilay et al., 2009] such as Project Management and Human-Computer Interaction. In particular, universities need to provide to their Software Engineering students not only technology-related skills but also a basic understanding of typical phenomena occurring in industrial software projects. In addition, more emphasis is needed on interdisciplinary culture and communication skills due to the very nature of software applications.

The lack of student's preparation for dealing with real-world Software Engineering projects has been recognized by industry [McMillan and Rajaprabhakaran, 1999; Conn, 2002; Callahan and Pedigo, 2002; Aasheim et al., 2010], and in response to it, the academic community has created some practices that bring the student closer to what happens in non-academic software development organizations. These practices include, for example, requiring students to apply software quality engineering concepts

in an industry representative project [Huffman Hayes, 2002], executing programs that resemble the medical residency programs [Sampaio et al., 2005] and using simulation games [Connolly et al., 2007].

Most Software Engineering courses include a group or team project to provide undergraduate students with an opportunity to practice the concepts learned from the text books and lectures. For many Computer Science programs, this is the first opportunity that students have to work together [Coppit and Haddox-Schatz, 2005] and gain experience in some techniques of Software Engineering. The team project emphasizes the importance of the tools, processes, communication, teamwork and other techniques for software development [Olsen, 2008]. Typically, the project assignment takes the entire semester to complete. However, due to time and resources constraints this is not enough for students to experience many phenomena that occur in real-world software development processes [Navarro, 2006].

Specifically, the Software Engineering course at the Department of Computer Science at Federal University of Minas Gerais, Brazil, is taught in one semester (60 hour class). This course is designed to cover the fundamentals of Software Engineering theory and practice in an undergraduate program. The course adopts team-based projects, four to six members per team, as an approach to provide undergraduate students with hands-on learning experience. Each team is responsible for the complete specification of a small software application. The students follow the Praxis [Paula Filho, 2006, 2007, 2009] process that prescribes the use of UML for the models and Java as the programming language, using test-driven techniques [Paula Filho, 2006].

After analyzing the defects found in artifacts produced by students in the team-based project and interviewing instructors [Peixoto et al., 2010b], we observed that there is an undesirable gap between what is taught in the classroom and what the students have to do in the team project. As an introductory course, it is not possible to provide to the students all the details needed for an adequate software development. However, it seems that even some important points were not made explicit. Some qualitative impressions collected from the instructors and confirmed by the students through a survey were:

- Students have difficulties making the connection between the course-provided material (models together with the code framework) and the lectures.
- Students focus on technical artifacts and usually do not take into account the managerial ones.
- Even if they are required, students usually do not inspect their artifacts. Many

times, they filled the defect report artifact with unreal data. This can be observed because of inconsistencies among artifacts.

One would expect that complying with the prescribed process, as learned in class, is a natural behavior. In spite of that, we observed the opposite. Students usually do not follow the process during the team project development. They first elaborate the technical artifacts and then they complete the baseline with the managerial artifacts. At the end, only a few students that finish the course really understand the way and the benefits of conducting a process driven software development. Although this study was not replicated in other universities, we observed that there is a lack in the preparation of the students in general, mainly regarding bridging the gap between Software Engineering course and the industrial software development environment. In this context, some Software Engineering Education researchers argue that one feasible way to provide students with a real experience of software development processes within the academic environment is through simulation [Mandl-Striegnitz, 2001; Carrington et al., 2005; Navarro, 2006]. This is more effective as supplementary to traditional approaches rather than a stand-alone technique [Mandl-Striegnitz, 2001; Carrington et al., 2005].

Therefore, our goal is to overcome the problems identified above and contribute to Software Engineering Education improvement, using a simulation game. This simulation game allows students to practice some Software Engineering skills in a more realistic environment. Specifically, students will be able to:

- Practice concepts related to technical and managerial processes of software development.

- Reinforce the concepts learned in class, mainly, the Software Engineering lessons, carrying out a Software Process Improvement (SPI) initiative (considering, for example, that requirements deficiencies are the prime source of project failures [Glass, 1998]). Specifically, the simulation rewards correct decisions (appropriate Software Engineering practices) and penalize unjustifiable ones.

- Follow the stages of a software development process in an organization.

- Learn some of the events that can occur during a Software Process Improvement project (e.g. resistance to change, high-level management commitment).

- Observe possible results of improvement actions taken during development (e.g. the reduction of defects after performing the Requirement Engineering activities).

- Analyze the effects of following an immature software development process (e.g. more restricted process visibility, higher number of defects). Therefore, reinforcing the possible advantages of having a defined process.

1.1 Research Questions

The complexity of teaching a more realistic experience of software development processes and the applicability of simulators as a complementary educational tool are two premises that constitute the basis for this research.

In an industrial environment, the software development process is extremely complex by its nature, in which "human interactions are complicated and never very crisp and clean in their effects, but they matter more than other aspect of the work" [DeMarco and Lister, 1999]. Since software development is defined by a collection of certain and uncertain factors of diverse types that most of time negatively affects the results, teaching this subject is also intrinsically complex [Zhang, 2008].

Despite the related software complexity and uncertainty, few attempts have been made to teach an experience closer to what happens in non-academic software development organizations through simulations. In these initiatives, simulations were recommended to be used as a complementary component to the Software Engineering course [Carrington et al., 2005; Mandl-Striegnitz, 2001]. In this way, the students acquire the needed background and better learn the lessons that the simulations are designed to teach.

Given these premises, we developed a simulation game called SPIAL (*Software Process Improvement Animated Learning Environment*) and since the beginning of SPIAL development we elaborated three related research questions. The sequence of these three research questions guided the specification, development and tests of SPIAL. SPIAL is a graphical and interactive game-based simulation environment for Software Process Improvement. It is a useful approach for teaching or reinforcing the Software Engineering concepts to students. The research questions are:

- RQ 1: Can students learn Software Engineering concepts or reinforce them using SPIAL?

  Our main goal is to determine if students can learn certain Software Engineering lessons from such SPI simulator. A fundamental part of our work is to evaluate SPIAL in conjunction with actual courses. Therefore, we carried out an experiment with undergraduate Software Engineering students to verify this research question. In addition, a specialist evaluated SPIAL communicability perspective.

- RQ 2: What characteristics are required for SPIAL to be capable of supporting the learning process?

  We investigated existing Software Engineering simulation games and we carried out a systematic literature review in order to identify the concepts and requirements to be covered by an SPI simulator. Answering this question can also provide insights into the design of simulation games in general. Moreover, after carrying out the experiment, we validated whether or not these characteristics are actually employed by students.

- RQ 3: How can such a simulator be incorporated into Software Engineering courses?

  SPIAL was designed to be used in conjunction with Software Engineering courses. We provided some guidance to students and instructors of these courses. The appropriateness of this guidance was evaluated through an inspection method, informal evaluation by instructors and through the experiment. The experiment also gave us some insights about the effort required from the students in order to figure out how to play the game.

1.2 Approach

The following steps were conducted: 1) an exploratory case study 2) systematic literature reviews 3) semiotic analysis 4) methodology development and 5) validation.

1. **Exploratory case study**: we investigated the problems incurred by students during the team project development. Our investigation and its main results were published in the context of an Engineering Education conference [Peixoto et al., 2010b]. The most striking observation is the difficulty that students have to bridge the gap between the theoretical lectures and the team project, and the limited range of skills that they apply during the project development.

2. **Systematic literature reviews**: we identified and compared the main Software Engineering simulation games. Based on this analysis, we defined the essential design aspects for the development of our simulator. Our investigation and its main results were published in the context of a Software Engineering Education conference [Peixoto et al., 2011]. We also investigated the real benefits of an SPI initiative in software development organizations.

3. **Semiotic analysis**: we investigated the communicability weaknesses and strengths of a specific Software Engineering simulation game. Although the evaluation is restricted to one system, the results can be used as representative of a large category of related systems. Our investigation and its main results were published in the context of a Human Computer Interaction conference [Peixoto et al., 2010c].

4. **Methodology development**: we built a graphical and interactive simulation game for Software Process Improvement. The design issues and simulation game characteristics were published in the context of an Engineering Education conference [Peixoto et al., 2012a,b].

5. **Validation**: we evaluated SPIAL in a Software Engineering course. In addition, an inspection using the Semiotic Inspection Method was conducted by a specialist.

1.3 Scope

The research object of this work is a Software Engineering educational simulation game (defined in Section 2.4). Specifically, an SPI simulation game used as a complementary approach in an introductory Software Engineering course.

The scope of this research includes the simulation game design, process model definition (to be simulated), with its inputs and output variables, and the selection of the simulation approach, which is presented in Chapter 5. SPIAL's scope covers both a development project and an improvement project (pilot project). In SPIAL, the player acts as a manager of an SPI group. We adopted a subset of the Software Engineering knowledge area and a specific SPI reference model, CMMI-DEV version 1.3 [CMMI, 2010], to be modeled in the game. CMMI was chosen because it is the most widely known SPI reference model. The Software Engineering concepts that are taught in SPIAL were extracted from a CMMI report and also encompass 57 Software Engineering rules (Appendix B). In the first version, the player influences the improvement project and development project making improvements on some process areas: Configuration Management, Measurement and Analysis, Organizational Process Definition, Organizational Training, Project Planning, Project Monitoring and Control, Requirements Development, Requirements Management, Technical Solution, Validation and Verification. Particularly, we also selected project development aspects to be modeled at SPIAL as, for example, Waterfall development process with its artifacts,

control measures (e.g. defects, cost, time-to-market, productivity, budget, and time used) and interaction with team members.

It was not our purpose to cover all the SPI and Software Engineering body of knowledge or CMMI-DEV reference model. As presented throughout the text, the most appropriate simulation model and SPI aspects modeled depend directly on the educational objectives. We also did not focus on technical knowledge of the development process, for example, specific modeling language, or a specific development tool. In addition, we considered SPI activities able to produce effects observable during the time of one project.

Finally, a variety of simulation approaches and languages was applied to simulate software processes [Kellner and Madachy, 1999]. Since in software projects some phenomena are best modeled by continuous mechanisms while others are best modeled by discrete ones, we applied a hybrid simulation approach. It is beyond the scope of our research the adaptation of other approaches, such as qualitative and semi-quantitative software process modeling [Zhang and Kitchenham, 2006].

1.4 Contributions

The main contributions of this research lie in the development and validation process that underpinned this simulation game creation, as well as the design and application of a simulation game in Software Engineering education. This process, which has a strong focus on interaction aspects, can provide some insights for simulation game developers.

The main achievements resulting from SPIAL development can be summarized as follows:

1. Theoretical aspects:
    - Description of the main characteristics that make the adoption of simulation games successful. This information was consolidated in Possa [2011] and Peixoto et al. [2012b].
    - Specification of an SPI simulation game with:
        - definition of roles, activities, and results;
        - definition of a simulation model;
        - definition of a reusable framework.
    - Identification and characterization of the actual results of SPI initiatives.

- Identification of the issues and challenges during the design of SPIAL [Peixoto et al., 2012a].

2. Practical aspects:

   - Development of a set of components which can be reused in the development of new Software Engineering simulation games.
   - Integration of a specific SPI simulation model with the reusable components.
   - Development of the simulation game itself, along with its SPI simulation model.

3. Empirical aspects:

   - Communicability evaluation of SPIAL conducted by a specialist.
   - Practical evaluation of SPIAL carried out by students through a pilot experiment.

In sum, the innovation introduced consists of an SPI simulation game that addresses some issues of Software Engineering education, by allowing students to practice some Software Engineering skills.

1.5 Thesis Outline

Chapters 2, 3 and 4 provide the essential concepts related to the research, i.e. software process improvement, software process simulation, simulation games, and semiotic evaluation. Chapter 5 presents the core information for the simulation game design, development and validation, addressing the adoption of the proposed approach in Software Engineering practice. The details of each chapter are:

- Chapter 2 provides important concepts related to the research, i.e. software process, software process improvement, software process simulation and simulation games.

- Chapter 3 describes the communicability aspects of one specific simulation game, presenting how it can be improved. The focus is on one essential issue, the feedback [Mory, 2003], and the resulting considerations go beyond the analyzed simulation game.

- Chapter 4 describes a systematic literature review of software process improvement. The results present an analysis of the actual results of SPI initiatives. The underlying trends are also discussed. This chapter is based in one article of ours and we maintain certain aspects of the article such as the itemized contributions.
- Chapter 5 presents the modeling approach, the design decisions and the evaluation of the simulation game.
- Chapter 6 summarizes SPIAL development approach and evaluation. It ends at highlighting the contributions and suggesting future research.

Chapter 2

Background

This chapter presents the main research subjects of this work and it is structured as follows:

- Sections 2.1, 2.2 and 2.3 depict brief definitions of software process, software process improvement, and software process simulation model.
- Section 2.4 presents the analysis of design aspects related to Software Engineering educational simulation games.
- Section 2.5 concludes this chapter.

2.1 Software Process

Software process consists of a set of partially ordered activities intended to produce or enhance software products, or to provide services [Sommerville, 2011; Feiler and Humphrey, 1992]. Software process may be an undocumented ad hoc process or it may be a standardized and documented process used by various teams.

We can identify at least four process activities that are essential to Software Engineering [Sommerville, 2011]:

1. Software specification: Definition of software functionality and constraints.
2. Software design and implementation: Production of software that meets the specification.
3. Software validation: Validation of software to ensure that it does what the customer wants.

4. Software evolution: Evolution of software to meet changing customer needs.

Besides the process activities, process descriptions may include other fundamental constructs that are frequently mentioned [Feiler and Humphrey, 1992; Curtis et al., 1992]:

- Actor: An agent (human or machine) who performs a process activity.
- Role: Functional responsibility of the actors involved in the process.
- Artifact: Product created or modified by the execution of software activities.

The process is seen as the glue that ties people, technology, and procedures coherently together [O'Regan, 2011]. Software processes are important in Software Engineering, not only to provide consistence on the development activities, but also to capture past experiences to be used by others.

2.2 Software Process Improvement

Software process improvement (SPI) is the term used to refer to systematically improving software development processes in order to improve product quality and business value. Since the foundation of the Software Engineering Institute (SEI) at Carnegie Mellon University and the publishing of Watts Humphrey's book [Humphrey, 1989], a deeper understanding and a growing number of researches were reported in the so called Software Process Improvement area. A set of models and standards has been developed, such as CMM [Paulk et al., 1993, 1995], more recently CMMI [Chrissis et al., 2006], and ISO/IEC-TR-15504 [2008].

According to Sommerville [2011], SPI usually involves three main sub-processes, and we will refer to them as **SPI initiative**:

1. Process measurement: Attributes of the current project or the product are measured.
2. Process assessment: The status of the processes currently used by the organization is analyzed, and process weakness and bottlenecks are identified.
3. Process changes: Process changes that have been identified during the assessment are deployed.

In our study, the group of specialists who facilitate the definition, maintenance, and improvement of the process used by an organization is referred to as **SPI group**. This group performs the following functions:

- lead development and implementation of the improvement actions;
- define and improve software practices;
- analyze process data;
- assess periodically the software processes; and
- plan training.

Learning, primarily, happens within projects through feedback cycles that identify weaknesses, initiate and implement new efforts, as the corresponding projects evolve and deliver their results [Aaen et al., 2007] [1]. One success factor of process improvement is to have at least one software development project involved from the start of the improvement initiative in order to apply the new processes [Calvo-Manzano Villalón et al., 2002; Kauppinen et al., 2004]. The development project is called **pilot project** and it allows the identification of advantages and disadvantages of the change, before the organization-wide implementation [Kauppinen et al., 2004].

As mentioned in Chapter 1, CMMI is arguably the most famous SPI reference model. More specifically the Capability Maturity Model Integration for Development, version 1.3 (CMMI-DEV, v1.3) [CMMI, 2010] was chosen to be the base of SPIAL. CMMI-DEV is a reference model that covers activities for developing both products and services. Organizations from many industries use this reference model. Besides CMMI-DEV, there are two other models: CMMI for Services and CMMI for Acquisition. In this work we use the term CMMI to refer to CMMI-DEV. A detailed description of how CMMI is captured in SPIAL is presented in Chapter 5.

CMMI defines 22 process areas. Each **process area** consists of a set of goals and these must be implemented by a set of related practices in order to satisfy the process area. Levels in CMMI describe an evolutionary path recommended for an organization to improve the processes it uses. CMMI supports two paths (or defines two forms of representation) using levels. One path, **continuous**, enables a selection of processes (or group of process areas) by organizations in order to incrementally improve them. The other path, **staged**, enables the improvement of a set of related

[1] Here we are using the term "feedback" in the context of the cited work. In Section 2.4.1, we consider *Informative Feedback* and *Performance Feedback* according to the work of Malone [1980]. In Chapter 3, we describe a semiotic analysis considering these two types of feedback.

processes by incrementally addressing successive sets of process areas [CMMI, 2010]. With the continuous representation **capability levels** can be achieved and with the staged representation **maturity levels**. Table 2.1 shows some of the terminology related to the two types of paths used in CMMI. The Capability levels are labeled or numbered from 0 to 3 and the Maturity levels are numbered from 1 to 5. We do not discuss here the complete relationship between capability and maturity levels in CMMI, but in a simplified way, mainly for the first levels, the process areas contained in maturity level i specify that their capability must be at least at level i. Conversely if all the process areas of a certain maturity level i are assessed as being at capability level i then it is possible to say that the organization is at maturity level i.

Table 2.1. Terminology related to Capability and Maturity Levels (extracted from [CMMI, 2010]).

Level	Continuous Representation Capability Levels	Staged Representation Maturity Model
Level 0	Incomplete	-
Level 1	Performed	Initial
Level 2	Managed	Managed
Level 3	Defined	Defined
Level 4	-	Quantitatively Managed
Level 5	-	Optimizing

2.3 Software Process Simulation Model (SPSM)

A **model** is a simplified representation of a real or conceptual complex system [Kellner and Madachy, 1999]. Modeling means capturing and abstracting significant features and characteristics of the system. A **simulation model** is a computerized model which represents some dynamic features or phenomenon of the systems it represents [Kellner and Madachy, 1999; Raffo and Wakeland, 2008]. Simulations are used when the complexity of the system being modeled is beyond what static models or other techniques can usefully represent. Simulation model typically involves a set of assumptions concerning the operation of the system [Banks et al., 2009]. It is used to exercise the system model with given inputs to see its pattern of behavior. The applications of simulation are vast [Banks et al., 2009], for example, to analyze the complexity of a mail transportation network, or to balance the operating room and post-anesthesia resources. This is a viable alternative when the costs, risks or logistics of manipulating real systems are prohibitive [Kellner and Madachy, 1999].

According to Zhang et al. [2008], **Software Process Simulation Modeling** (SPSM) was introduced in the Software Engineering field by Abdel-Hamid and Madnick [1991]. Since then, it has gained increasing interest among academic researchers and practitioners alike as an approach for [Kellner and Madachy, 1999]: strategic management, planning, control and operational management, process improvement, technology adoption, understanding, training and learning. As example of its application in the Software Engineering field, specifically in the process improvement area, Christie [1999] presented how a software process simulation can support CMM in all five maturity levels. In a more recent work, Raffo and Wakeland [2008] presented a detailed report about how process simulation modeling has been implemented within industry and government organizations to improve their processes and achieve higher levels of process maturity. Via many examples, their work showed how process simulation supports CMMI Process Areas from maturity level 2 through level 5 and how some simulation results can be useful in determining financial performance measurements such as Return on Investment and Net Present Value. Raffo et al. [1999] showed that simulation can provide quantitative assessment of the risk and uncertainty associated with process change and support quantitative prediction of project level performance in terms of effort, staffing, schedule and quality. We have used process simulation to forecast the impact of applying a specific process improvement practice before each inspection in a software development organization [Peixoto et al., 2010a].

According to Banks et al. [2009], simulation models can be classified as static or dynamic, deterministic or stochastic and discrete or continuous. A static simulation model represents systems at a particular point of time and dynamic simulation represents systems as they change over time. A deterministic simulation model does not contain random variables, as it happens in a stochastic simulation model. The essential difference between discrete and continuous is how the simulation time is advanced. In discrete simulation the state variable changes only at discrete set of points in time, normally when an event occurs. Dynamic simulation does not constrain time in terms of the events' occurrence and advances time with a constant delta, as in System Dynamics [Madachy, 2008].

The most widely used techniques in SPSM are System Dynamics and Discrete-event simulation as a representative for dynamic and discrete simulation categories, respectively. As presented by Zhang et al. [2008], from 1998 to 2007, 49% of the SPSM researches apply the System Dynamics paradigm and 31% Discrete-event simulation.

A comparison of the relative strengths and weaknesses between the two are provided in Martin and Raffo [2000] and Kellner and Madachy [1999]. The Discrete-event model is efficient when the process is viewed as a sequence of activities. It is often used

to represent entities with unique values for attributes (e.g. error rates or the impact of different programmer capabilities), where values can be constant or sampled from a distribution [Martin and Raffo, 2000]. The process of changing the attributes' values is an essential characteristic of a discrete model. This model can also represent queues and the interdependence that occurs between activities in a project. Continuously changing variables cannot be modeled accurately; because they can only change their values at the event times [Kellner and Madachy, 1999] which can take days or weeks. On the other hand, the System Dynamics models represent the system as 'flows' (e.g. error generation rate) that accumulate in various 'levels' (e.g. the current number of errors). The flows can be dynamic functions or the consequence of other variables. As the time advance, in small spaced increments, the levels and the flows rates are changed [Martin and Raffo, 2000]. Its focus is not on specific individuals or events, but on patterns of behavior and on average individuals in a population [Madachy, 2008]. It accurately captures the effects of feedback [Kellner and Madachy, 1999], but it does not provide an explicit mechanism to represent sequence or to represent individual entities with their attributes, neither allows modeling uncertainty in an easy way [Martin and Raffo, 2000].

The different aspects of software development projects need both mechanisms in order to be suitably simulated. The duration and effort of activities are rarely predicted without some uncertainty, and the size of artifacts or programmers experience may differ. In addition, the dynamics of a project may vary over time, for example, the schedule pressures and programmer's fatigue. As discussed in Section 5.4, SPIAL uses a framework which provides both types of simulation. It represents the activities of a software development organization and also its dynamic environment.

2.4 Simulation Games

Over the past years it was observed a movement towards more active and experientially based learning [Madachy, 2008], especially using simulations and games. Simulation-based learning approaches are concerned with aspects of the real world [Garris et al., 2002] that could not be possible to present to the students in a course project. Important features of simulations are related to their rules and strategies, they usually represent some real-world system, and they have low cost errors compared to real-world situations, avoiding serious consequences of mistakes [Crookall and Saunders, 1989]. By contrast, games usually do not intend to represent real-world systems. They also have rules and strategies, and generally when we lose, the cost of error can be high but may

be contained within the game world [Crookall and Saunders, 1989]. Thus, simulations and games can be considered similar in some aspects. In addition, simulations can contain game features as proposed by Garris et al. [2002]: fantasy, sensory stimuli, rules/goals, challenge, mystery, and control. According to von Wangenheim and Shull [2009], games can be classified into four types: board games (e.g. Risk Management board game [Taran, 2007]), card games (e.g. Problems and Programmers [Baker et al., 2003]), quiz games (e.g. Lecture Quiz [Wang et al., 2008]), and simulation games (e.g. SimSE [Navarro, 2006]). In this section, our focus is on Software Engineering simulation games (referred to as **simulation games**), which are simulation tools with some game features.

In the Software Engineering field, we observed that it is very difficult to train students in the real situation of a software development organization, due to the very nature of software applications and the great diversity of organizational cultures. There are different ways that Software Engineering games can be incorporated in a university course. For example, games can be used instead of traditional exercises or in addition to them, they can be used within lectures to improve the participation and motivation of students, or they can be exercises by themselves (e.g. a task is given to the students to develop a simple computer game [Claypool and Claypool, 2005]). Unfortunately, there is no agreement regarding the main game aspects that support learning, the process for engaging learners and the type of learning results [Garris et al., 2002]. This is essentially related to the difficulty of designing an effective educational game system.

The design process is at the heart of the simulations and games development [Martin, 2000]. Despite the design process importance for attaining students' learning, we have noticed that it is not receiving much attention in the literature. One important step during the simulation game design is to take into account a chain of events: what will happen, what effect an event will have, and how this supports the learning process. During the design process, the designers[2] must take into account several aspects in order to design a simulation game that supports learning and motivates students.

In order to identify the differences among the Software Engineering simulation games, we discuss the main design decision involved in their development. The design issues addressed are related more to the interactional aspects of interface design than to architectural design or other kind of design concerns, which would be difficult to evaluate without additional information about the simulation game development. In the Human Computer Interaction context, system designers communicate to system users the essence of their design vision, through various types of interface signs

[2]In this research designers should be interpreted as whoever speaks for the design team.

(e.g. explanations, messages, buttons, menus, and interaction results) [de Souza, 2005], [de Souza et al., 2010]. We evaluate and compare signs used by designers to facilitate and enhance the interaction. It is not our purpose to state that one simulation game is better than other, but to show explicitly some important characteristics that each simulation game has related to its interaction design.

The results of the next section were presented at the 24th IEEE-CS Conference on Software Engineering Education and Training [Peixoto et al., 2011].

2.4.1 Simulation Games Design

The primary concern of simulation game design is to elaborate a system that is congruent with the learning goals. Designers must tell users what they mean by the simulation game they have created, and players are expected to understand how to play the simulation game and respond to it. Similarly to programs in other domains, the communication [de Souza, 2005] occurs through the interface, basically by the use of messages encoded in words, graphics, behavior, online help, and explanations.

Peters et al. [1998] propose three principles of simulation and game design: reduction (selection of elements from the real world), abstraction (simplified representation of the elements) and symbolization (elements and relations are represented into a new symbolic system, with several aspects, e.g. roles, and rules). Their work emphasizes the necessity of fidelity in the representation of real situations in games and simulations, where ideas are iteratively mapped between the conceptual content and the game process. Fripp [1993] proposes a more specific guideline for the design process that is very useful for evaluating important aspects of the simulations and games. In his work, he creates a design verification list that includes purpose, reality/complexity, timeline, feedback, decisions, participants, and roles. Moreover, Gredler [2004] proposes four important characteristics of simulations that are: (1) an "adequate" model of the complex real-world situation (the "adequate" is regarding fidelity and validity aspects); (2) a defined role for each player, with responsibilities and constraints; (3) an environment that allows players to carry out different strategies; and (4) feedback for players actions in the form of changes in the problem or situation.

Game design is an iterative process that involves dealing with event's effects and learning objectives [Martin, 2000]. Garris et al. [2002] described an input-process-outcome game model (Figure 2.1). In this model, the game cycle starts with the design of an instructional program that supports some features of the game, this triggers a cycle that include user judgment and reactions. The reactions lead to behaviors that result in system feedback on performance in the game context. Finally, the learn-

ing outcomes are checked through a careful debriefing session [Peters and Vissers, 2004]. So, the design should take into account all these steps, mainly engaging students not to leave the loop judgment-behavior-feedback. In addition, they proposed some game characteristics that were classified in six broad dimensions or categories: fantasy, rules/goals, sensory stimuli, challenge, mystery, and control.

The assessment of simulation games is an extensive activity that we decided to pursue as a research on its own and it is out of the scope of this research to carry out user tests or empirical experiments. Based on the examination of some important design characteristics collected from the literature, we decided to discuss the following aspects of the Software Engineering simulation games:

- **Clear Goals and Rules**: The simulation game should have specific rules that govern the playing context, which ultimately describe the goal structure of the game. Clear objectives are important to engage students and to motivate them [Garris et al., 2002]. One important aspect is that clear, specific goals allow players to see discrepancies related to their final target, allowing their reduction. This triggers player's attention, more effort, and consequently motivation.

- **Feedback**: Feedback is a critical component of the judgment-behavior feedback cycle, proposed by Garris et al. [2002] (see Figure 2.1). Because it permits players to interpret the software, it also supports players' decisions about interactive goals and provides information for novice players during the learning process. We identify characteristics in the simulation games related to two kinds of feedbacks: informative feedback and performance feedback [Malone, 1980]. Informative feedback presents enough information to maintain players interested in the simulation game or to maintain their curiosity, based on the information presented. Performance feedback shows the players' performance according to the established goal, i.e. how close they are to the objectives.

- **Virtual world**: The simulation game is a case study of a particular reality where players play roles of characters in a particular situation following a set of rules and interacting with other characters. The settings, players' role, players' educational profile and the learning objectives are important design aspects related to the virtual world where the game occurs.

- **Adaptability**: It plays an important role in the quality of the educational systems, allowing the learning environment to be adapted according to different levels of initial knowledge, different learning styles, and different expectations and objectives [Moreno-Ger et al., 2008]. This can be achieved basically in two

ways: one is by gradually increasing the complexity in the course of the game and the other is by providing different simulation models (related to the learning goal).

- **Game features**: As simulation games, it is important to incorporate some features that do not exist in the real world, for example score or fantasy, mainly to enhance the learning process.

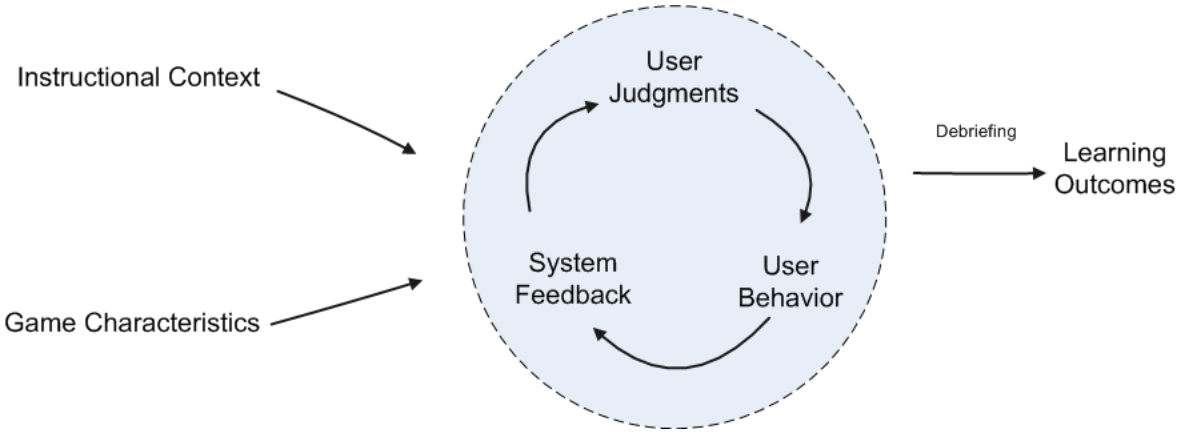

Figure 2.1. Input-Process-Outcome game model (extracted from Garris et al. [2002]).

2.4.2 Literature Review of Software Engineering Simulation Games

This section reports a literature review [Kitchenham, 2004; Kitchenham and Charters, 2007] aimed at extending and refining an existing secondary study (i.e. a study based on previously published research) that surveyed Software Engineering simulations and games in the time period from 1st January 1990 to 30th June 2008 [von Wangenheim and Shull, 2009]. The original secondary study performed its search process considering a set of seven journals and conferences, searching for simulations and games applied in Software Engineering education. In this section we presented only the main points of the systematic literature review protocol employed in this study. An example of a protocol is presented in Chapter 4.

Our review goal is more specific, we identify computer-based simulation games used for teaching Software Engineering topics having results published in the period of ten years (from 1st January 1999 to 30th December 2009). We did not find articles published before this period.

The high-level question of our research is:

How the design characteristics (described above) are promoted in the Software Engineering computer-based simulation games?

The primary studies used in this review were obtained from searching databases of peer-reviewed Software Engineering research that met two criteria:

- The database contains peer-reviewed Software Engineering journals articles and conference proceedings.
- The database was used in other Software Engineering systematic reviews.

The resulting list of databases that we searched was: ACM Digital Library, Sage Simulation&Gaming, IEEEXplore, ScienceDirect, SpringerLink and Wiley Interscience database.

We submitted the search string "software engineering" and ("simulation" or "game") on these databases [3]. The searches resulted in a large number of candidate papers: 2583 articles. We used inclusion/exclusion criteria to narrow the search to relevant papers by applying the following steps: reading the title to eliminate irrelevant papers, reading the abstract and keywords to eliminate additional irrelevant papers and reading the remainder of the paper and including only those that address the research question. As we were specifically interested in computer-based simulation games, we excluded the ones not based on computer simulation games such as card games (e.g. Problems and Programmers [Baker et al., 2003]) and Software Engineering tutorial programs that are not actual interactive simulation games (e.g. OSS [Sharp and Hall, 2000]). Using this approach, the initial results were filtered down to 16 papers (Table 2.2). Prior to conducting the systematic research, we were aware of the number of papers that are relevant (by our research interests and also by the previous literature review). As an indicator of completeness of the review, we found, among others, all papers detected in the previous literature review [von Wangenheim and Shull, 2009].

2.4.2.1 Results

Based on the search results and on the inclusion and exclusion criteria, a set of computer-based simulation games was selected. A brief description about the selected simulation games is depicted in Table 2.2 and their design characteristics are presented

[3] We experimented other search strings in order to verify that we have chosen suitable ones: "software process" and ("simulation" or "game"), "software engineering process" and ("simulation" or "game"), "software engineering simulation game", "software engineering simulation", "software engineering game"

in Appendix A. After the selection and data extraction, each simulation game was evaluated showing its main characteristics and differences regarding the design, as described below. When we conducted our evaluation, we not only read the articles, but we also played the simulation games (the ones available). The results provided a rich set of simulation games descriptions that can be used for selection and development of new applications.

Clear Goals and Rules: Basically, we can divide the simulation games in two groups: those that the final goal is to develop a software project within a certain set of constraints, and their rules are based on the Software Engineering lessons: SESAM, SimSE, SimJavaSP, MO-SEProcess; and those that the final goals and rules are specific to each simulation game (e.g. to manage a project): The Incredible Manager, SimVBSE, qGame and TREG. One important design aspect that was followed by all simulation games was the clear specification of the goal. So, at the beginning of each game, the players can understand their objectives and manage to achieve them. On the other hand, this is not true for the rules. The Software Engineering rules are not clear to the players in all games. These rules correspond to an important learning process, where players become skilled in anticipating and correcting the effects of choosing the incorrect actions. If the players learn these rules, they would eventually be in a position to achieve a good score and understanding the dynamics of a Software Engineering team.

Feedback: In the simulation games evaluated, the designers adopted a "trial and error" strategy, since the signs are limited in their message communication, mainly the signs presented at the interface. This is an expected aspect in simulation games, but in some cases the miscommunication gets in the way of the player's learning process. For example, if players do not read the manual they could not understand the purpose of some tool (e.g. Explanatory Tool, a tool that shows the rules and measures used in the game) or find a specific functionality (e.g. how to generate branches), and it is a well-known fact that players rarely read manuals before playing. Designers are challenged to find a balance between the information displayed in the interface and the information provided through other media (e.g. manual, tutorials, and so on), in a way that the game is still inviting and challenging. Regarding the type and the instant of the feedback, some simulation games present the performance feedback only at the end. The consequence of this is that good results are normally obtained only after several game plays. In contrast, there exist informative feedbacks that are presented throughout the whole game varying in signs used by designers: bubbles, graphics, tables, animated elements and so on. Only two simulation games provide a detailed feedback at the end, thus allowing the analysis of the game score and the results: SimSE

and SESAM. The importance of this technique has been emphasized in the previous literature review [von Wangenheim and Shull, 2009]. The collaboration and feedback among players is a key feature presented only in Second Life simulation games, where it can enhance teamwork skills and team member participation. Finally, the use of game-like elements to represent certain characteristics of the real world (e.g. fatigue) can improve engagement and motivation

Virtual World: As specified in the design characteristic "Clear Goals and Rules", the first group of simulation games has the educational goal to teach Software Engineering process to students. Typically, they present a metaphor of a physical software development organization with doors, walls, furniture and employees. In this virtual organization, the player assumes a project manager role that manages the team, assigns tasks, purchases tools or hardware, and monitors the project, within a reduced time frame. The other group has the goal to teach a specific subject of Software Engineering, like Value-Based Software Engineering [Jain and Boehm, 2006]. The simulation game world should be tightly coupled with the aims and objectives of the educational context, which is a close representation of the real-world system. An important question that the designers should address is: what is the appropriate balance in a particular teaching context between the real and virtual world.

Adaptability: Among all simulation games that we found, only three support different simulation models: SESAM, SimSE and The Incredible Manager; and only one allows different levels of difficulty: qGame. This restricted number is due to the complexity of design and interaction of the learning environment, when we consider the adaptability characteristic. Besides the need to adapt to meet the special needs of each student [Moreno-Ger et al., 2008], the simulation games also need to attend all stakeholders' expectations regarding their educational objectives (both students and instructors). In addition, there is the cost of building an adaptable simulation game and the cost of changing its simulation model. The modification of the simulation model is, in most case, not a trivial task, even with a friendly interface. Therefore, the designers should first investigate the educational context of their simulation game to decide which strategy they will follow (to include or not this facility). This includes investigating who are the learners, what are the learning aims, what level of study the students have, their experiences and competencies, the subject taught, and how the simulation game will fit into the existing course.

Game features: After the simulation games evaluation, we found some essential game features that can enhance motivation and in consequence the effectiveness of learning. These features are presented in the whole group of the evaluated tools, which are: not too easy nor too difficult life-like challenges (e.g. constraints on budget, de-

fects, and time); interaction; a mechanism that provides feedback from the other team members (e.g. dialogue); graphical representation; and some indication of performance (e.g. score). Not all games presented random events, but this is also an important feature to enhance challenge and in consequence motivation.

Table 2.2. Computer-based Simulation Games.

Name	Description	Ref
SESAM	SESAM (Software Engineering Simulation by Animated Models) is a single-player simulation game. The goal is to complete a project, where costs and schedule are established during a planning phase and approved by stakeholders.	[Ludewig et al., 1992; Drappa and Ludewig, 2000]
SimSE	SimSE is an educational game based on Software Engineering simulation environment. SimSE is a single-player simulation game in which the player takes the role of a project manager of a team of developers. The player goal is to develop a software project within a certain set of constraints (e.g. budget, schedule, and number of defects).	[Navarro, 2006; Navarro and van der Hoek, 2002, 2005a,b, 2007, 2009]
The Incredible Manager	The Incredible Manager is a single-player simulation game that allows a trainee to act as a project manager, being responsible for planning, executing, and controlling a software project. The simulation game focuses on project management rather than on software process.	[Barros et al., 2002, 2006]
SimJavaSP	SimJavaSP is a web-based single-player simulation game where the student act as a project manager, developing a software project within the required time and budget, and acceptable quality.	[Shaw and Dermoudy, 2005]
MO-SEProcess	MO-SEProcess (Multiplayer Online Software Engineering Process) is a multiplayer online Software Engineering process game based on SimSE. It is a collaborative simulation game developed in Second Life.	[Ye et al., 2007]
SimVBSE	SimVBSE is a single-player simulation game specifically designed to teach students the theory of Value-Based Software Engineering.	[Jain and Boehm, 2006]
qGame	qGame is a web-based single-player Software Engineering game that shows the problems resultant from the requirement flow during the project development (misunderstandings or wrong assumptions).	[Knauss et al., 2008]
TREG	TREG (Training in Requirements Engineering Game) is a multiplayer online Requirement Engineering game developed in Second Life. It is a collaborative game developed to train students in requirement elicitation.	[Vega et al., 2009b,a]

2.4.2.2 Discussion

An assessment of the simulation games using the characteristics described above reveals that, although they are, in some cases, developed with different aims (e.g. SimSE x SimVBSE), they have a considerable number of common aspects, for example, "trial and error" strategy, the required basic knowledge which includes at least the meaning of each action to be selected or activity to be assigned to the employees, goals, rules, player's role and some game features. They also have different aspects mainly regarding the feedback instant and type, and the adaptability characteristics.

Other conclusion can be draw when comparing the elements used for interaction. We observed a clear evolution in these elements. In the beginning, textual messages, and typed commands were predominant. In the more recent works, we observed collaborative simulation games applying more elaborated interfaces (e.g. Second Life style APIs). This is possible because, in the present time, there are powerful creation tools available that facilitate the simulation game development. Moreover, players are more demanding for elaborated interfaces.

Another outcome is that a few simulation games considered individual performance assessment. This was not limited to only the final score presentation, but also to the functionality that allows the evaluation of the students' mistakes, e.g. Explanatory Tool [Ludewig et al., 1992; Navarro and van der Hoek, 2005a]. This assessment is mainly centered on the rules which are the base of the performance calculation.

Finally, we believe that if designers take into account some of these aspects, the results will be more satisfactory regarding the simulation game interaction and the learning process.

2.5 Conclusion

This chapter introduced the concepts employed in this thesis, i.e. software process, software process improvement, software process simulation model and simulation games.

The simulation games mentioned in this chapter have been identified through a systematic literature review. In this review we observed that, well-designed simulation games are challenging and interesting for players, requiring the use of particular knowledge or skills. Five design characteristics that are important in meeting this requirement were examined in the domain of Software Engineering simulation games: clear goals and rules, feedback, virtual world, adaptability and game features. This leads to important observation of common aspects and differences among available educational simulation games: SESAM, SimSE, The Incredible Manager, SimJavaSP,

MO-SEProcess, SimVBSE, qGame, and TREG. Regarding the limitations of this study, one could argue that our discussion is subjective and was not validated through empirical experiments or user tests. Although this is a valid criticism, we verified our conclusions playing the games, when they were available. Furthermore, we cannot a priori assume that all the results of a study generalize beyond the specific educational environment. But we believe that many considerations can be applied to the whole educational simulation game field.

The next chapter introduces the semiotic analysis concepts that focus on how a software designer communicates with a user through the software's interface. In this research context, the semiotic analysis is an important instrument for the communicability evaluation of the simulation game interface. In addition, Chapter 3 extends the research carried out in this chapter, specifically, presenting a more detailed discussion about feedback.

Chapter 3

Semiotic Analysis

This chapter shows how semiotic concepts can be used in the analysis and generation of simulation game knowledge through the application of the Semiotic Inspection Method (SIM), a Semiotic Engineering evaluation method. It also emphasizes the importance of a communicability aspect and presents how it can be improved. The focus is on feedback[4], an essential issue which motivates players and supports their learning process. The resulting considerations go beyond the analyzed simulation game and most of them were confirmed by the literature review. We extend the previous analysis carried out by the literature review (Section 2.4) and we present a more detailed discussion about the feedback, its implications during the design and some guidance for new simulation game developments. The results guided SPIAL requirements definition (Chapter 5), reinforcing the importance of feedback to the learning process. This research was presented at 25th Annual ACM Symposium on Applied Computing [Peixoto et al., 2010c].

This chapter is structured as follows:

- Section 3.1 introduces semiotic concepts that are useful for our final assessments.
- Section 3.2 shows the core three steps of the method. Each step focuses on a single class of signs: metalinguistic, static, and dynamic.
- Section 3.3 presents a final evaluation of the inspected simulation game.
- Section 3.4 discusses the validity of the findings considering the evidence obtained with other procedures.

[4]Here we use "feedback" with a broader meaning, we also consider feedback mechanisms as discussed in the work of de Souza [2005]. Specifically, we consider "feedback" referring to the interaction mechanisms of the simulation game (e.g. help system or informative messages) and also "feedback" that supports the learning process.

- Section 3.5 concludes this chapter.

3.1 Background

Simulation games are more enjoyable and fun when they provide sufficient challenge for the player [Malone, 1980]. Games should engage student's interest and should be usable to permit results that are more clear-cut and enjoyable. Therefore, important aspects of the games must be evaluated, in particular its communicability. Communicability is a concept proposed by Semiotic Engineering Theory [de Souza, 2005] and is defined as "the property of software that efficiently and effectively conveys to the users its underlying design intent and interactive principles" [de Souza, 2005; Prates et al., 2000].

There are two distinct methods to evaluate the communicability of an interface: Semiotic Inspection Method (SIM) [de Souza, 2005; de Souza and Leitão, 2009; de Souza et al., 2006, 2010] and Communicability Evaluation Method (CEM) [de Souza, 2005; Prates et al., 2000]. Both are qualitative and interpretative methods. The use of CEM requires expensive experiments since it requires specialists to observe users in a tightly controlled environment. In our work we decided to use SIM since it is an inspection method and not so expensive. Nevertheless, it provides results comparable to CEM. As described by de Souza et al. [2010], SIM has a technical and a scientific application. Technical application of SIM focuses on how a method can improve and inform interaction design in the context of a specific system development; whereas scientific application focuses on how a method can broaden the knowledge of Human Computer Interaction (HCI). In our work, we performed a scientific application of SIM in order to evaluate the communicability of a Software Engineering educational simulation game, specifically regarding its feedback. As a result, we identified feedback aspects and issues relevant not only to the evaluated system, but also for simulation games used in educational contexts in general.

SIM has been proposed within the Semiotic Engineering theoretical framework [de Souza, 2005]. Semiotic Engineering perceives HCI as a designer to user metacommunication. That is, a twofold communicative process, because the designer to user communication is achieved through user-system communication.

The system's interface is a complete and complex message sent from designers to users [de Souza, 2005]. This message is formed by signs. According to Peirce, signs are anything that stand for something (else) in someone's perspective [Peirce, 1992]. The message conveys to users the designers' understanding of whom the users are,

what problems they want or need to solve and how to interact with the system in order to achieve their goals. This designer-to-user communication is indirect, since users understand the intended message as they interact with the interface. When users do not understand aspects of this message, a communication breakdown takes place [de Souza, 2005].

Semiotic Engineering theory classifies the signs in an interactive system into three classes: metalinguistic, static and dynamic [de Souza et al., 2006, 2010]. Metalinguistic signs refer to other interface signs. They are instructions, tips, online help, error and informative messages, warnings and system documentation. They are signs that the designer uses to explicitly communicate to users the meanings encoded in the systems and how they can be used. Static signs express and mean the system's state, they are motionless and persistent when no interaction is taking place. They can be perceived (and interpreted) in snapshots of the system's interface before or after interaction occurs. For instance, buttons, text areas and check boxes at a given moment. Dynamic signs express and mean the system behavior. Their representations unfold and transform themselves in response to an interactive turn. For example, if we click on the search button the behavior will present the results of a search. This behavior is a dynamic sign.

The SIM examines the designer's metacommunication that users are exposed to as they interact. The goal is to identify communication breakdowns that may take place during the user-system interaction and to reconstruct the designers' metamessage being conveyed by the system. To do so the evaluator analyzes the message being transmitted by signs at each level: metalinguistic, static and dynamic.

This method is carried out in five distinct steps [de Souza and Leitão, 2009; de Souza et al., 2006, 2010]: (1) an inspection of metalinguistic signs; (2) an inspection of static interface signs; (3) an inspection of dynamic interaction signs; (4) a contrastive comparison of designer-to-user metacommunications identified in steps (1), (2), (3); and, finally, (5) a conclusive appreciation of the quality of the overall designer-to-user metacommunication. At the end of steps (1), (2) and (3), the evaluator reconstructs the metamessage being conveyed by signs at that level, filling out the template of the designer to user communication. The evaluator also identifies potential communicability problems that may take place at that level. In step (4) the evaluator contrasts the metamessage generated in the previous steps and checks for inconsistencies or ambiguities among them. It is not expected that the messages generated by each sign level to be identical[5], but they should be consistent. Finally, in step (5), the inspec-

[5]Signs at each level have distinct expressive possibilities. For instance, natural text used at the help system (metalinguistic sign) and an icon at the toolbar (static sign) usually do not express the

tor reconstructs a unified metacommunication message, judging the costs and benefits of communicative strategies identified in previous steps and generates the evaluation report.

Like other methods, SIM requires a preparation step. In this step, the evaluator defines the purpose of the inspection, does an informal inspection in order to define the focus of the evaluation, navigates through the system and finally elaborates the inspection scenarios. If the method is being applied scientifically, this step should also produce a clear statement of the research goal and an examination of how it could be achieved by using SIM [de Souza et al., 2010].

The scientific application of SIM helps researchers in diagnosing the system's problematic situations and it also provides theoretical concepts that support the formulation of problems as research questions [de Souza and Leitão, 2009; de Souza et al., 2010]. In a scientific context, SIM also includes a final triangulation step to validate the results [de Souza et al., 2010]. The validation step corresponds to a triangulation of results against evidences obtained with other methods or procedures. The triangulation can be carried out with empirical evidences provided by endogenous and exogenous sources. The endogenous triangulation is obtained with reference to different aspects of the same system or systems in the same domain. In this triangulation, the evaluators search for interpretations that are consistent with their analysis. The exogenous triangulation is carried out with reference to systems in other domains.

The scientific application of SIM is recent and it was firstly applied to investigate the feedback of a system. In this study [de Souza et al., 2010], the research question addressed was: "how the system's feedback is communicated to users during interaction". The case studies were carried out with a simple editor of cascading style sheets (Simple CSS) and Google Groups. The results of the Simple CSS study were endogenously triangulated with findings of Web material (FAQ, users' opinions, etc) about CSS editors of the same kind and exogenously triangulated with an analysis of system's feedback to users' actions in creating and configuring a group in Google Groups. The results presented in de Souza et al. [2010] show that it is possible to achieve a more powerful metacommunication using different sign types (namely icons, indices and symbols) that simultaneously and consistently mean the same feedback. One important value of this scientific application of SIM is to associate feedback issues with end-users specification, designing and programming tasks.

Feedback is a critical element during interaction, because it allows players to interpret the software and the designers' message. In an educational game, it provides

---

same content equally well.

information for novice players during the learning process as well as supporting players' decisions throughout the game [Navarro, 2006]. Thus, the results presented in de Souza et al. [2010] and the importance of simulation games to educational contexts [Navarro, 2006] has motivated our investigation of feedback issues specific to this context.

It is also important to briefly present some semiotic concepts that will be used in our analysis. Namely, a semiotic classification of signs types [de Souza, 2005; Peirce, 1992] - icons, indices and symbols, and the semiotic phenomenological categories: *firstness*, *secondness* and *thirdness*. **Icons** are signs where representation evokes the *firstness* of its referent. A phenomenon of the first category brings up a unary quality experience of sensing the referent. So when we represent a cow with its picture, we evoke the visual experience we have when encountering cows in general. **Indices** are signs where the representation brings out the *secondness* of its referent. When we take a smoke as a representation of fire, this evokes a certain kind of association (smoke indicates fire). The idea of *secondness* presents some casual association between concepts. Finally, **symbols** are signs which representation evokes the *thirdness* of its representation. Symbols result from a regulating convention that relates them to their referent by virtue of an established meaning. For example,"™" conventionally refers to a registered trademark.

In order to investigate feedback issues on educational games, a Software Engineering simulation game, SimSE [Navarro, 2006], was chosen and SIM was scientifically applied to this system. SimSE was chosen because it is a free tool and, according to its authors, it is an enjoyable educational game and relatively easy to play. The goal of this analysis is to identify, in a broader context, feedback issues on educational simulation games.

3.2 Analyzing the Designers' Message

The analysis of the **metalinguistic signs** allows us to identify important elements used in the designers' discourse. The metalinguistic signs are used to provide a better understanding of the static and dynamic signs, and the intended relation among them. To analyze the metalinguistic signs the downloading site, manual, dialogs, messages and tutorials were inspected in regard to what they conveyed about the system's feedback (focus of investigation). The gist of what SimSE's designers intend to communicate to players focusing on feedback through metalinguistic signs can be summarized in 5 messages, where 'we' means 'we, the designers' and 'you' means 'you, user, or in this case, the player':

(a) If 'you' want more information about the functionalities of the game, 'you' should search in the Players Manuals. Since, there is no online help or tooltips available.

(b) 'You' can see some informative feedback through speech bubbles over the employees' head. These bubbles convey important information about the effects of the actions and also provide some guidance.

(c) 'You' can see some informative feedback of the employees and the resources (artifacts, customers, employees, projects and tools). Employees' information appears in the 'current activities' area. The status of the resources appears on the bottom table (Figure 3.1) or in a pop-up window.

(d) 'You' can only see 'your' performance feedback (score) at the end of the game.

(e) 'You' can learn how to improve 'your' score reading the rules used in the game through the Explanatory Tool. This tool allows 'you' to generate graphs of various attributes and discover insights of the rules underlying the game.

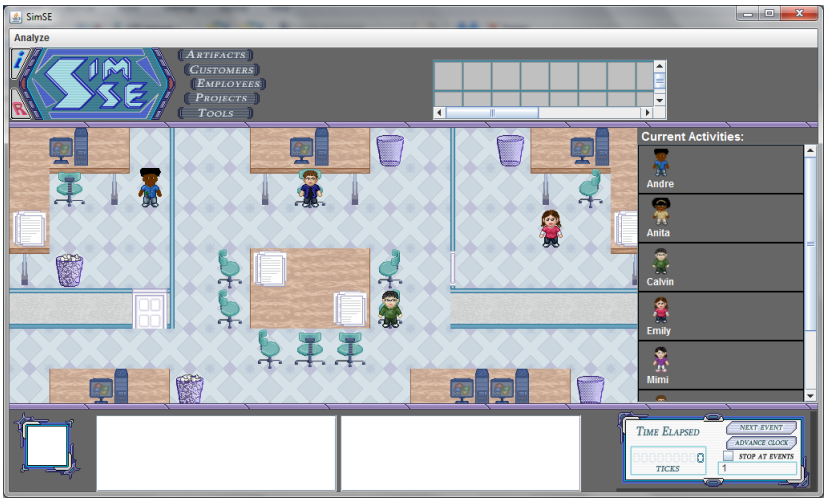

Figure 3.1. SimSE main interface.

During the analysis of the **static signs** (step 2), important signs were identified at the main interface of the game. We can organize the content expressed in the messages as:

(a) 'We' simulate an office environment, and 'you' can notice that 'you' are in an office with doors, walls, furniture and employees (avatars).

(b) 'We' provide, as in real offices, information about the current status of the resources and the employees on tables.

(c) 'We' provide some important information related to the simulation. At the main interface, 'we' provide information and action related to the simulation time as seen on the lower right corner. In the Explanatory Tool, accessed through the Analyze menu item, 'you' can generate graphs and evaluate the rules of the game.

(d) 'We' consider that 'you' will be able to infer that 'i' stands for a short narrative about the goal of the game and that 'R' stands for simulation Reset, as appears on the left side of the SimSE logo.

Static signs focus on the presentation of the metaphor of a physical office and beyond that some restricted functionality related to the simulation time control (next event, advance clock, time elapsed).

The analysis of **dynamic signs** yields more interesting and informative results for our evaluation. With this analysis, it is possible to evaluate the dynamic aspects of the game and it is also possible to see the effects of some of the simulation parameters on the attributes. The gist of what SimSE's designers intend to communicate to players focusing on feedback through dynamic signs can be summarized in:

(a) 'We' use a metaphor of a physical office to determine the interaction 'you' may have with the game.

(b) 'We' consider that 'you' have some knowledge about Software Engineering, because of the signs 'we' used (e.g. the name of the activities: 'create the system test plan').

(c) 'We' provide four important feedback mechanisms: the tables, the activity list, the bubbles over the employees head and the Explanatory Tool. The tables present information about the resources and employees. The activity list presents the actions that the employees are performing. The activities executed by each employee are also presented at the bubbles over their heads. The Explanatory Tool is accessed through the Analyze menu item.

(d) 'You' can interact with the project team directly on the main interface. The employees are static avatars that communicate through pop-up bubbles. They have the same appearance throughout the game, independently of what happens and they remain at the interface, unless they are fired (only then do they disappear).

(e) During the game 'you' can see information related to the results of 'your' actions (e.g. employees' mood and energy). 'Your' score can only be seen at the end of the game.

(f) 'We' would like 'you' to play the game immediately after reading the project goal, using a 'trial and error' strategy (exploratory interaction), since 'we' do not direct 'you' towards any other information.

In the communication described in 'c', the avatars overhead bubbles can confuse the players when acting as a project manager. When they assign more than one activity to the employees, the order of execution is not represented. And the message presented by the employees, sometimes just lists the current activity they are performing.

3.3 Communicability Evaluation

During the metalinguistic and static evaluation, we observed that the designers clearly stated that the players will play the role of a project manager. However, it is not communicated that the players must have basic knowledge of Software Engineering and this can affect their interest and motivation. Therefore, the knowledge level required to play the game should be clearly stated to the players, whatever it is.

In relation to feedback, the tool should support students in acquiring a more complete and consistent knowledge [Malone, 1980]. The informative feedback shows some status of the project, but the player can get confused with what actions would have a positive or a negative impact. If players do not read the manual they will not understand the goal of the Explanatory Tool that shows the rules of the game and it would be safe to assume that players rarely read manuals before playing. One improvement would be to allow students to access their performance indicators before the end of the game, or bring to the game a virtual consultant that could give hints (for example as a virtual software quality manager) to act as a wizard.

The informative feedback, such as the number of defects in an artifact and the money spent, is given to the player during a game session. But the players learn about their performance only at the end of the game. So, it may take several game plays to understand the impact of decisions on the overall performance and obtain good results. The performance feedback should be previously presented in the way to enhance challenge and to minimize self-esteem damage. This is in accordance with some learning theories used in the game [Navarro, 2006] as: Discovery Learning and Learning through Failure (which also reinforces the designer 'trial and error' strategy) and Constructivism (that reinforces the previous needed knowledge of the players). Otherwise, as discussed above, the game could fail to be enjoyable after a while.

At a closer examination, the problem is more complex. This game does not represent some signs of *firstness* that would be more effective to communicate and, in

consequence, give direct feedback to the players. For example, when the energy of the employees reduces, they could gradually disappear from the interface, or if they were moody they could look different (for example, with a tired face). In all, some of these characteristics should be incorporated in this simulation to turn it into a more game-like tool. The feedback represented by *secondness* can also be improved if the cause-effect relations are better represented. We observed that the consequences of some actions are not directly represented in the elements and the player can get confused. One example is when an employee goes on a break, he/she does not disappear from the interface. Another point of confusion is the definition of the order of execution of some tasks assigned to an employee. The player, as project manager, should be able to change (or at least see) the priority of the tasks, when a project is not going well, but this is not supported by the game. Regarding *thirdness*, the designer uses some known signification systems to communicate with users. For example, the names of the activities are those conventionally used for tasks in software development process. This indicates the need for players to have previous knowledge of these signification systems, since the meaning of used signs is not available at the interface. However, the student who does not know any of these concepts could be frustrated with this software. Other important symbolic signs are the rules of the game. The rules can foster an important learning process, where players become skilled in anticipating the effects of choosing an action. If the players learned these rules, they would eventually be in a position to achieve a good score and understand the dynamics of a Software Engineering team. Therefore they should be well communicated to players within the context of the game, which does not happen in SimSE.

3.4 Triangulation

To validate our conclusion about the feedback communication in SimSE, a triangulation with empirical evidence provided by endogenous sources was performed. We found concrete evidence that players have missed the informative and performance feedback and that the prerequisites for playing the game are not clear [Navarro, 2006].

Evidence 1: *"I still don't really understand what the score is based on."* and *"I'm not really sure exactly what the scoring criteria are."*

These are comments from students that do not use the Explanatory Tool and, for them, it was not possible to understand the rules that led to a certain score.

Evidence 2: *"Rules were a major help. The rules are really helpful-even if someone doesn't know anything about Software Engineering I think rules can teach you*

how to play the game." and "*SimSE's explanatory tool is a useful resource for helping players understand their score, but its value lays primarily in its rule descriptions."*

Students that used the Explanatory Tool think that rules can be useful in learning how to play the game and understanding the score. But, yet it is not possible to know how the score is calculated.

Evidence 3: *"[SimSE is] a good way of putting concepts into practice."* and *"It didn't help so much compared to what I already know."*

These comments show that the player should have some knowledge of Software Engineering to play the game.

An important result, that reinforces the first scientific application of SIM [de Souza et al., 2010], is that the consistent use of icons, indices and symbols that refer to the same feedback (redundancy) may improve the designer to user metacommunication. A comparison among the results of SimSE and those of Google Groups and Simple CSS (described in [de Souza et al., 2010]) reveals that the three evaluations have much in common. The first characteristic is the importance of the perceptible change of the iconic representation of the elements after the changing of their attributes (e.g. the employees' visual appearance). Another important characteristic is the correct visual effect of the indices (for example, after being fired the employee would disappear from the interface). In addition, an explicit and consistent symbolic representation of partial results and strategies used to calculate them could improve the feedback, considering that only through the use of icons and indices it would be virtually impossible to communicate such aspects.

3.5 Conclusion

This chapter presented the steps using SIM and the analysis of feedback in an educational simulation game. The goal was to identify issues regarding efficient and effective communicability conveyed through the game's feedback which can help the development of new simulation games.

The analysis was performed with the aim of checking issues on how SimSE deals with feedback. In this analysis we observed that designers of SimSE focused on the informative feedback and use of 'trial and error' strategy. Dynamic signs are privileged in designer-to-user communication and convey most aspects of the whole message. However, it seems that designers should explore other playability characteristics [Desurvire et al., 2004] in the game, such as varying the levels of difficulty. This would require much more elaborated signs, and it would probably provide players a more productive

experience.

We identified that some important aspects should be observed during the design of educational simulation games. These aspects were taken into consideration during SPIAL design:

- Designers should represent some signs of *firstness* that would give direct feedback to players. In some situations, this feedback can be more effective to the learning process than using tips or on-line helps. For example, in SimSE, the modification of the avatars' state after some action (they disappear, move or look different).
- *Secondness* signs that can communicate cause-effect relations are also important, since players can quickly visualize the results of the selected actions.
- Designers should use *thirdness* signs, since rules and strategies are conventions defined by designers and cannot be fully communicated through *firstness* and *secondness* signs.
- The designers should provide an efficient and effective way to present the rules used in the game and their relation to the players' performance. In educational games, this would support the learning process, and players could become skilled in anticipating the effects of an action and correcting them when necessary.
- It is important to have both informative and performance feedback during the whole game. This provides challenge for the players motivating them.

The next chapter investigates the SPI domain through a comprehensive systematic literature review over the decade from 1999 to 2009. SPI is a fundamental concept used for SPIAL creation and it is discussed in its own chapter.

Chapter 4

SPI Initiatives and Results

Through a systematic literature review, this chapter provides a view of the SPI area, which served as a guide for us in our investigations, and also provides information to managers, researchers and practitioners about SPI efforts. The high-level goal of this study is to:

Identify and characterize the actual results of SPI initiatives.

In SPIAL context, the results of this study provided the bases for the simulation model and requirements definition (Chapter 5). Mainly, helped us during the model calibration, measures and process areas definition.

This chapter is structured as follows:

- Section 4.1 presents the main contributions achieved with the systematic literature review.
- Section 4.2 details the phases carried out during the literature review: planning, conducting and reporting the review.
- Section 4.3 describes the main qualitative and quantitative findings.
- Section 4.4 discusses the review questions and the SPI reporting directives.
- Section 4.5 presents the related works.
- Section 4.6 shows the limitations of this review.
- Section 4.7 concludes this chapter.

4.1 Background

There are previous reviews of the SPI body of literature [Müller et al., 2010; Unterkalmsteiner et al., 2012; Pino et al., 2008; Sulayman and Mendes, 2009; Staples and Niazi, 2008], but these reviews do not include all the results of our analysis such as, for example, the evaluation of improvement percentages values and the SPI categories. We applied a formalized and repeatable review process [Kitchenham and Charters, 2007] and after following this process, we selected and analyzed 91 studies.

SPI is intended to enhance the software development process. Despite the expected SPI focus being on the process, it is also customary to have initiatives that call themselves "SPI initiatives" but whose scope is not the process (i.e. related to the definition or management of the process). The first contribution is to illustrate the fact that most of the SPI studies in reality do not focus on process aspects. Instead, they focus on the SPI impacts or results, e.g. the main common reason for launching SPI efforts is the reduction of the defect rates detected during development and after delivery.

Despite warnings from some authors, part of the software development community counter-intuitively believes that the results from SPI initiatives will be "dramatic" or "surprising". By "dramatic" we mean "powers of ten" increases in productivity or quality. Our second contribution is to confirm that SPI initiatives do not bring "dramatic" changes. In this aspect our contribution is in agreement with Frederick P. Brooks when he says: "There is inherently no silver bullet" [Brooks, 1987], and also with Robert L. Glass: "most software tool and technique improvements account for about a 5 to 35 percent increase in productivity and quality" [Glass, 2002]. The improvement percentage distribution of the different measurements is discussed in Section 4.4 and they reinforce the fact that there are no "dramatic" changes.

During the analysis, we were able to identify that SPI initiatives could be arranged in categories or "equivalence classes" considering aspects that are similar amongst them. We refer to these equivalence classes as "patterns". The goal of identifying these patterns is to enable a discussion referring to the category of improvement rather than the individual initiatives. These patterns provide concrete descriptions of actions, measurements and lessons learned for specific situations. Our third contribution is the identification of SPI patterns intended to ease the sharing of SPI knowledge.

In addition, we were also able to identify some shortcomings of many an SPI report. Therefore, we decided to compile some suggestions of how SPI results should be reported. Our fourth contribution is the compilation of guidelines for SPI reporting based on weaknesses observed in the selected studies.

Our last contribution consists of a set of numerical observations that can be succinctly summarized as follows:

1. A considerable number of the studies' data originated from interviews, observations or questionnaires (50%) revealing that most of the organizations have difficulties to measure and analyze quantitatively their improvements.

2. Among the SPI Initiatives, requirement engineering processes stand out (20%) and most of these initiatives have only a qualitative evaluation.

3. Large organizations represent the major part of SPI reports (38%).

4. CMM/CMMI and IDEAL [McFeeley, 1996] are the most frequently used improvement models (34%) and process improvement implementation guide (9%), respectively.

5. The major part of the SPI research originates in Scandinavia (26%) and in the rest of Europe (37%), followed by America (17%), Asia (9%) and Oceania (4%).

4.2 Research Method

A Systematic Literature Review (SLR) consists of three main phases: planning, conducting and reporting the review [Kitchenham, 2004]. In this section we summarize the review protocol that was produced during the planning phase. This protocol was constructed based on established methods for carrying out systematic literature reviews [Kitchenham and Charters, 2007; Kitchenham, 2004]. This review protocol consists of the description of the following stages: research questions specification, case study roles definition, electronic databases selection, search string definition, study selection, quality assessment, and data extraction and synthesis.

4.2.1 Research Questions

The main goal of this study is to investigate the actual SPI results described in the literature and the data that support their value. To address this research aim, we defined a set of research questions. The high-level research question of this study is:

What are the realities of SPI initiatives?

This research question was then refined into four groups of questions shown in Table 4.1, which guided the literature review.

Table 4.1. Research questions and motivations.

Research question	Motivation
Which are the most common reasons for organizations to implement SPI initiatives? (e.g. improve process visibility, increase profit or enlarge market share)	One relevant step for the analysis of these studies is to first identify the motivation for SPI initiatives. This is important to establish the connection among reasons, measurements and the analyses carried out by organizations.
Which reference/assessment models the organizations adopted? (e.g. CMMI, ISO/IEC 15504, ad hoc)	The analysis of the models, standards and approaches are essential to understand how organizations carry out their SPI initiatives, what steps they follow, and how the results are evaluated.
What are the main aspects that are measured in SPI initiatives? (e.g. productivity, quality, ROI or market share)	This question provides an underpinning aspect to evaluate the "order of magnitudes" of the improvements.
What are the lessons learned with the SPI initiatives? What are the main problems? What are the main problems during the different phases of a cycle? What are the main problems during the different cycles? What are the usual solutions?	These questions are essential to categorize improvements and bind problems and solutions in a contextualized setting.

4.2.2 Case Study Roles

We assigned specific roles to the team members in order to perform the systematic review as follows:

- SLR supervisors: They are responsible for providing the resources to the SLR Research Team, reviewing the protocol and the final results. They also ensure that the Research Team collects the required information.
- SLR Team Leader (TL): TL is responsible for constructing the SLR Protocol document.
- SLR Research Team (RT) member: RT member is responsible for executing the SLR process (identification of the primary studies and data extraction) and documenting the results.

The team was as follows:

- SLR supervisors: Three professors;
- SLR Team Leader: The author of this thesis;
- SLR Research Team members: The research team involved several students with a core group of three students.

4.2.3 Data Sources

The primary studies of this SLR were obtained from searching electronic databases that met the following criteria: (i) The databases contain peer reviewed Software Engineering journal papers and conference proceedings; (ii) the databases have a search engine with an advanced search mechanism that allowed keyword searches; (iii) the databases provide the access to full text documents; and (iv) the databases were used in other Software Engineering systematic reviews.

The resulting list of databases that we searched was:

- ACM Digital Library (http://portal.acm.org/)
- IEEEXplore (http://www.ieeexplore.ieee.org/)
- Wiley InterScience (http://www3.interscience.wiley.com/cgi-bin/home)
- Elsevier ScienceDirect (http://www.sciencedirect.com/)
- SpringerLink (http://www.springerlink.com/)

The search results were manually organized with a tool called Mendeley[6]. This tool provides a combination of a desktop and a website that allowed the sharing of information among the RT members. It also supports the automatic extraction of document details (e.g. authors, title, and journal name) from the searched databases into the tool database, which saved a lot of manual typing. During the stages of this SLR, each team member had access to the work done by his colleagues, facilitating the verification.

4.2.4 Search Criteria

Search keywords are very important for the quality of the retrieved results, so they must be chosen carefully. In our study, these keywords were based on a technique called PICO (Population, Intervention, Comparison and Outcomes) [Kitchenham and Charters, 2007]. The first Search String (SS) derived from the research questions that includes the keywords identified from the PICO criteria was: software AND "process improvement" AND empirical (SS1).

In order to validate the SS1 we conducted a pilot project in two steps. Since we were expecting to retrieve articles that were not related to our target (false-positives), our concern was related to false-negatives, i.e. articles related to our target but not

[6]http://www.mendeley.com/

retrieved with the search strings. In the first step, we carried out a manual search in two journals: *Journal of Systems and Software* and *Information and Software Technology* from 2005 to 2009, and compared these results with the automatic search results. In the second step, we evaluated the SS1 and three other search strings: software AND "process improvement" (SS2), "software process improvement" AND empirical (SS3), "software process improvement" (SS4). Our goal was to make sure that we have chosen an appropriate search string. We compared the results of automatic search among these search strings. Since valid studies [Damian et al., 2002] and [Kautz et al., 2000] were not found by the SS1, SS3 and SS4, we decided to use a broaden string SS2 that probably would not eliminate papers that are essential to our research. So, our search string is:

software AND "process improvement"

We considered publications' date ranging from 1999 to 2009. In an area where there were so many radical changes in a few years (e.g. agile methodologies), we did not want to go back too many years in our search in order to avoid factors that have been relevant before but are not relevant anymore. Therefore, our SLR dates back to 1999.

4.2.5 Study Selection

By following the steps prescribed by Kitchenham and Charters [2007], we established a multistage process consisting of four steps with different review processes as described below.

- First step: The goal of this step is to remove duplicate and irrelevant papers. It was carried out evaluating only the title of the papers. For each database, two RT members were assigned. The TL coordinated the allocation of each researcher. One RT member was responsible for the separation of the papers (included and excluded ones) based on the title selection criteria. Another RT member was responsible for the inspection of the excluded list of papers, more specifically, this RT member verified if some paper was eliminated incorrectly.

- Second step: The goal of this step is to eliminate papers for which the abstract has no relationship with any of the research questions. The papers were assigned at random to one RT member by the TL. Another RT member was responsible for the inspection of the excluded list of papers, more specifically, the RT member verified if some paper was eliminated incorrectly.

- Third step: The goal of this step is to eliminate papers by scanning through the full text in order to check whether the inclusion or exclusion criteria were met. Two RT members were assigned to each paper. All disagreements were discussed and if it persisted the TL or the supervisors were contacted to give an opinion.

- Fourth step: The goal of this step is to exclude remaining duplicated works and analyze the papers thoroughly in order to extract data from the ones that met the inclusion criteria. In this stage, a full text analysis was performed on 91 papers and the quality of the studies was further assessed. Two RT members were assigned to each paper. All disagreements were discussed and if they persisted the TL or the supervisors were contacted to give an opinion.

The inclusion and exclusion criteria shown in Table 4.2 were used to narrow the search to relevant papers. Papers that address SPI initiatives and present explicit results of their deployment were included.

Table 4.2. Inclusion and exclusion criteria.

Criterion	Description
Inclusion	Papers mainly focused on SPI initiatives which contain explicit results of the process improvement, i.e. the improvement was implemented and the results were described.
Exclusion	Publications/reports for which only an abstract or a PowerPoint slide show are available. Short papers, editorials, posters, position papers, introductions of keynote, workshop, mini-tracks, special issues, or tutorials. Studies that are based only on expert opinion, i.e. it is merely a "lessons learned" report based on expert opinion (it is not a research paper). Studies presented in languages other than English. Studies not related to any of the research questions. Studies whose findings are unclear or ambiguous (i.e. results are not supported by any evidence). Studies external to Software Engineering field. Duplicated studies of the same work. When there is more than one study related to the same work, we included the most complete version and excluded all the other ones. Studies containing unsupported claims or frequently referring to existing work without providing citations. Studies that present tool evaluation, methodology experimentation or process implementation in an organization without an SPI focus. Studies that describe an SPI initiative with no practical application in an organization. Studies that focus on an evaluation of an SPI assessment method/standard and there is not enough description of the improvement initiative (e.g. only the assessed CMMI Level before and after the improvement). Studies that do not describe explicit results of the improvement initiative.

Figure 4.1 depicts the SLR steps and the number of papers identified at each one.

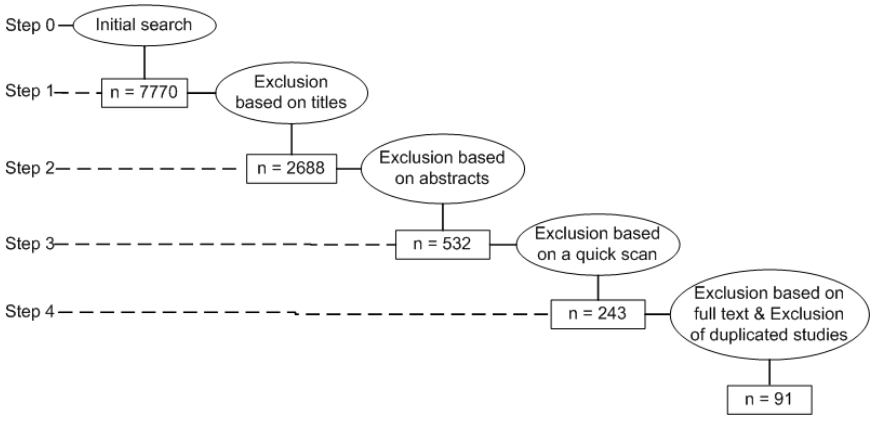

Figure 4.1. SLR steps.

4.2.6 Quality Assessment

During a systematic review, it is crucial to assess the quality of the studies in order to minimize bias and maximize internal and external validity [Kitchenham and Charters, 2007]. We created our quality assessment form based on checklists and guidelines from three other works [Dybå and Dingsøyr, 2008; Höst and Runeson, 2007; Runeson and Höst, 2009].

Two RT members assessed independently each of the 91 papers that remained after step four, following the 11 quality assessment criteria. Table 4.3 presents a summary of the quality assessment criteria applied in this SLR. These criteria, similarly to the ones in Dybå and Dingsøyr [2008], cover four aspects of quality that are critical when examining research papers.

- Reporting: It is related to the quality of reporting the goals, context, and purpose of the SPI initiative. (Questions 1-3)
- Rigor: It is related to the appropriateness of the approach applied to study the SPI initiative. (Questions 4-7)
- Credibility: It is related to the assessment of the confidence in the study's methodology for ensuring that the results were valid and meaningful. (Questions 8-10)
- Relevance: It is related to the use of the SPI findings by the software industry and the research community. (Question 11)

These 11 criteria provide a measurement that guides the interpretation of studies' findings and determines the value of their contribution to this SLR. The scoring

Table 4.3. Quality assessment criteria.

Number	Question	Issue
1	Is there a clear statement of the aims of the research?	Reporting
2	Is there an adequate description of the context in which the research was carried out?	Reporting
3	Is it clear the purpose of the SPI initiative?	Reporting
4	Was the study research design appropriate to address the aims of the research?	Rigor
5	Were threats to validity analyses addressed in a systematic way?	Rigor
6	Does the paper make explicitly a mention to the possibility of bias?	Rigor
7	Were the study cases appropriate for the study?	Rigor
8	Were the data collection procedures sufficient for the purpose (data sources, collection, storage, validation)?	Credibility
9	Were the analysis procedures sufficient for the purpose?	Credibility
10	Are the findings (positive and negative) presented?	Credibility
11	Are the results useful for another organizations or researchers?	Relevance

procedure consists of a three scale criterion: Yes (1), Partially (0.5) or No (0). We defined this scale because sometimes a simple Yes/No answer may be misleading. It is not a good practice to include study quality and reporting quality scores in a single metric [Kitchenham and Charters, 2007]. We adopted the weight of one to the reporting questions (1-3) and 1.5 to the other questions (4-11). Again, all disagreements were resolved by discussions that included all RT members, TL and supervisors.

4.2.7 Data Extraction and Synthesis

The data extraction form was used to ensure the consistency and accuracy of the information. The form contains fields derived from the research questions and from publication data. We first piloted the extraction process on the *ACM Digital Library* database. All three members of the RT reviewed the papers and extracted the required data. Then, in a consensus meeting, the RT members discussed the findings and the improvements to be incorporated into the form. In this meeting, they analyzed the extracted information making sure that the inconsistencies and doubts were solved. This step was to ensure that all RT members have the same interpretation regarding the extracted data and no additional information was required. In other two databases, *Wiley InterScience* and *Elsevier ScienceDirect*, two RT members reviewed the papers and extracted the required data. They also met in a consensus meeting to solve doubts or inconsistencies. For the *IEEEXplore* and *SpringerLink* databases, one RT member reviewed all papers and extracted data. Then, the other researcher independently reviewed and extracted data from a sample of the papers. This latter stage is consistent with the process followed by other systematic reviews [Walia and

Carver, 2009; Williams and Carver, 2010].

For each paper we collected data that were grouped into four categories:

- Publication-related data: digital library, title, year, source (e.g. journal or conference proceedings).
- Research-related data: research methodology (e.g. case study, case study and action research), analysis method (e.g. content analysis, statistics, grounded theory), data analysis (qualitative, quantitative, or both), data collection (e.g. interview, questionnaire, documentations, or observations). We followed the same terminology used in other works [Runeson and Höst, 2009; Prikladnicki and Audy, 2010; Oates, 2006].
- Organization-related data: number of organizations involved in the study, business model (Distributed or Internal), context, size, number of employees, and geography. The size of organization was classified according to a European guide [Commission, 2005]. The *Distributed* business model involves companies that work in global context and *Internal* encompasses companies where they do not have any relationship with other divisions in another country.
- SPI-related data: motivation, reference model or standard, implementation approach, list of problems, qualitative measurements, quantitative measurements, main topic of the SPI initiative, SPI project duration, list of proposed solutions, main lessons learned, benefits, and success and failures factors.

The data categories along with a description are presented in Table C.1 and the list of studies in Table C.2 (Appendix C). We synthesized the data by identifying recurrent themes emanating from the case studies reported in each paper. These identified themes gave us the patterns described in the next section. This pattern coding clusters related case studies' data into smaller number of sets, allowing a more integrated schema for analyzing the results.

4.3 Literature Analysis

4.3.1 Methodological Quality

As mentioned in Section 4.2.6, we evaluated each of the primary studies according to 11 quality criteria (Table 4.3). The quality criteria were employed in our study to investigate systematic differences between studies. Each study was rated independently

by two RT members. After checking if they had any discrepancies, the final score was defined.

The studies ranged from very well organized studies in the SPI field, to very concise ones where, in most cases, essential information for our analysis was missing. We were not interested in stating that one SPI study is better or worse than other. Instead, we focused on a set of issues that contribute to the quality of our research.

Almost all studies have some form of context description in which the research was conducted, presenting the positive and negative findings of their SPI initiative. The objectives of the research and the purpose of the SPI initiative were also described in some way in many of the studies. For 44 of 91 studies (48%), the chosen research design was not explained. As many as 70 of the 91 primary studies (77%) did not address the threats to validity. As many as 38 (42%) and 55 studies (60%), respectively, did not describe their data collection and data analysis procedures. Only 13 studies (14%) explicitly recognized the possibility of research bias. In ten studies (11%), the aims of the research were not properly described.

Therefore, we observed that the bias and validity were not adequately addressed; and the data collection, data analysis and research method were often not well explained. Three studies got a full score on the quality assessment (15 points) [Napier et al., 2009; Börjesson and Mathiassen, 2004; Anda et al., 2006] and 44 studies got a score less than 60% (nine points). The lowest number was 3.25 (see Table C.3, Appendix C).

4.3.2 Quantitative Analysis

In this section, we discuss the primary studies selected for an in-depth analysis. This analysis was divided into four categories of information: publication, research, organization and SPI. Most of the papers were case studies reporting SPI initiatives in large and small enterprises.

An important remark is that 45% of the papers present only qualitative data to support their process improvement results. A possible reason is that many of these organizations are not mature enough to have suitable quantitative data available from their practices. In addition, they were not able to set measurable goals for their process improvement effort, neither they were able to link their process improvement initiative with their business goals. We believe that an important reason for this lack of information is the one stated by van Solingen [2004]: "One frequent argument in software practice is that measuring SPI's benefits is impossible, or at least difficult". To solve this problem, van Solingen proposed some pragmatic solutions on how to calculate

cost, benefits and the ROI (Return On Investment) of an SPI initiative.

A considerable amount of papers did not report the organization-related data or the research-related data. This information may be important for other organizations or researchers interested in a specific study. The report communicates the findings of the study and it is the main source for evaluating its quality. Guidance for reporting case studies can be found in Runeson and Höst [2009].

4.3.2.1 Publication-related Data

The systematic review considered a period of 10 years (1999 to 2009). In total, we found 7770 papers as presented in Table 4.4.

Table 4.4. Distribution of studies by source.

Digital Library	Number of papers	Relevant studies	Selected studies
IEEEXplore	3508 (45%)	47 (43%)	37 (41%)
ACM	935 (12%)	7 (6%)	6 (7%)
Wiley InterScience	450 (6%)	19 (17%)	19 (21%)
Elsevier	987 (13%)	5 (5%)	4 (4%)
SpringerLink	1890 (24%)	32 (29%)	25 (27%)
Total	**7770**	**110**	**91**

After applying the exclusion criteria to the discovered papers, there were 110 relevant studies. Finally, identical studies found in several sources were removed, resulting in 91 different selected studies.

4.3.2.2 Research-related Data

The papers were classified according to the research methodology (Figure 4.2), data analysis (Figure 4.3 and Table 4.5), data collection (Figure 4.5) and analysis methodology (Figure 4.4). A large percentage of the studies (61%) used case study as a research methodology, with data collection using interviews (41%), and performing qualitative data analysis (45%). We did not find information regarding the analysis methodology in 75% of the studies, 14% of the studies applied Statistics, 4% Grounded Theory and 7% different types of analysis. This indicates that there is a demand for a more thorough guidance for the SPI analysis. From the 91 studies, 50 (55%) presented quantitative data, however in 11 of them [Salo and Abrahamsson, 2007; Larndorfer et al., 2009; Redzic and Baik, 2006; Freimut et al., 2005; Kikuchi and Kikuno, 2001; Nelson et al., 2001; Caliò et al., 2000; Karlström et al., 2005; Bibi et al., 2010; Moe et al., 2005; Motoyama, 2006] there was not enough information to make a comparison of the

improvement gains. These 50 publications refer mostly to large organizations (22, 44%) compared to Medium (5, 10%), Small (10, 20%), and Very Small (1, 2%). Surprisingly, 12 (24%) studies do not provide information about the size of the organization. This is an important information to be taken into account in an SPI initiative and should be considered when reporting it.

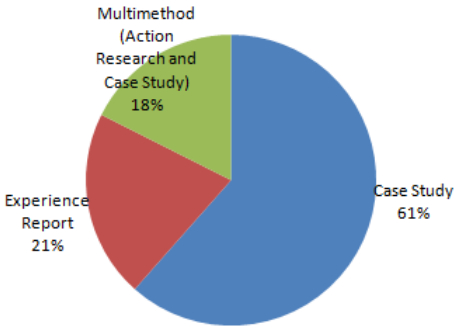

Figure 4.2. Research methodology.

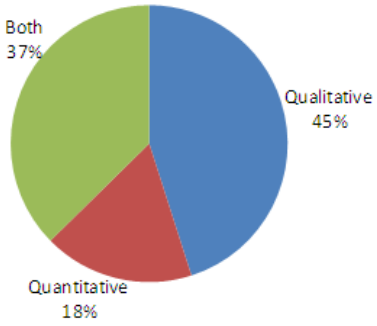

Figure 4.3. Data analysis method.

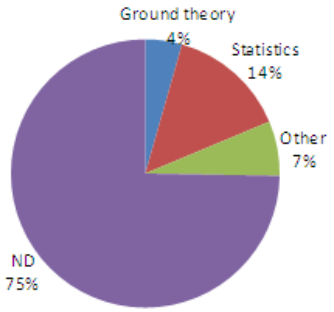

Figure 4.4. Analysis methodology.

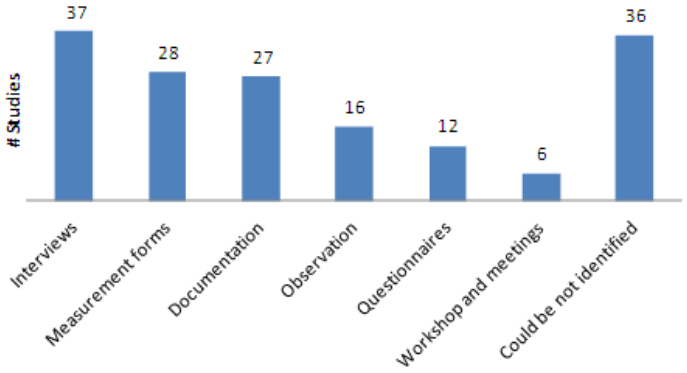

Figure 4.5. Data collection method.

4.3.2.3 Organization-related Data

We present the main results based on the organization description found in each study. We classified the organizations' size according to a European guide [Commission, 2005] where very small organizations are enterprises with fewer than 10 employees, small enterprises are organizations which have between 10 and 49, medium enterprises between 50 and 249, and large enterprises have 250 or more employees. Figure 4.6 presents the distribution according to the size classification and Figure 4.7 the geographical distribution of these companies. Figure 4.8 shows the distribution of organizations according to their business model. The organizations' descriptions provide the context for our comparative analysis and pattern definition. Most of the studies encompass large organizations from Scandinavia and Europe, applying qualitative data analysis. Observe that despite the fact that we have 91 studies, the total number of organizations is 123 (Table 4.5) since one study can report more than one organization.

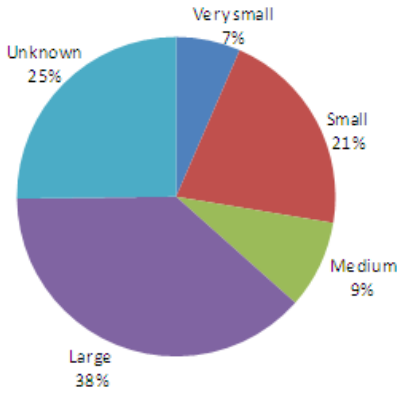

Figure 4.6. Distribution of organization's size.

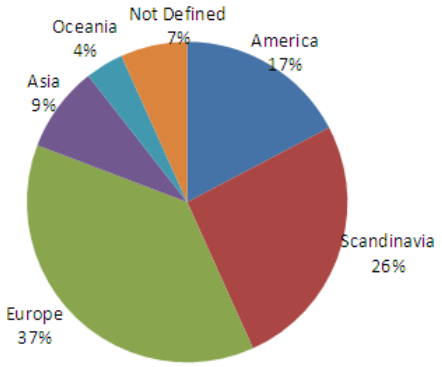

Figure 4.7. Geographical distribution of organizations.

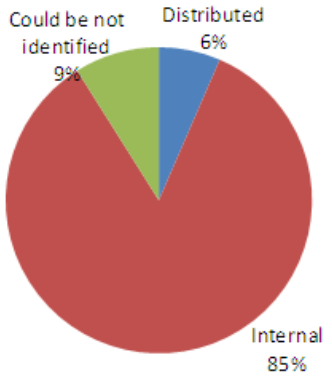

Figure 4.8. Business model.

Table 4.5. Data analysis method according to the organization size.

Organization Size	Qualitative	Quantitative	Both	Total
Very small	7	0	1	8
Small	16	2	8	26
Medium	6	1	4	11
Large	21	6	20	47
Unknown	17	7	7	31
Total	67	16	40	123

4.3.2.4 SPI-related Data

We analyze the main results of the SPI initiatives described in the primary studies. In this section we focus on the model/standards and the improved process areas. In the next section we present a detailed description of five process improvement patterns.

Different types of models and standards can be used by organizations in an improvement initiative. This may include models that guide the improvement, methods to assess the process or organization (capability or maturity), and process reference models. In the analyzed SPI initiatives, the most widely used models are the ones

from SEI. CMM and CMMI have been used by 34% of the improvement efforts studies. CMM represents 23% of the reported studies and CMMI 11%. IDEAL was used in 9% of the studies to guide improvement. Figure 4.9 shows the distribution. It is important to underline that in most of the cases the reference models were used in order to improve the process, not to obtain a certification, i.e. the certification was not the primary goal.

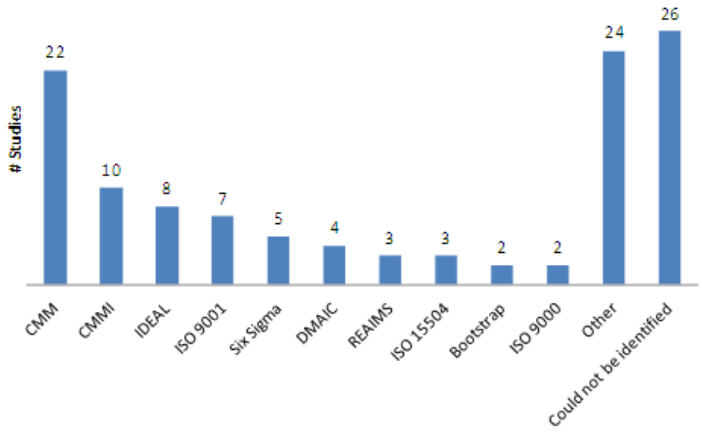

Figure 4.9. Approaches used by each study.

We searched for the information about the targeted processes in the studies. Typically, the studies did not identify the process areas to be improved using CMMI or other well-known models. Instead, they simply describe the process improvement areas in an unstructured text. We decided to map these processes areas from the textual descriptions to the process areas categorized in the ISO/IEC 12207:2008 [ISO/IEC-TR-12207, 2008]. This international standard groups the process areas in system context and software specific processes. The number of process areas derived from the studies descriptions in each group is presented in Figures 4.10 and 4.11.

The literature does not provide guidance on how to count the processes areas in an integrated context. When one study has more than one organization and the identified improvement process area is the same, we counted as one occurrence of the process area. For instance, Study S30 conveys three case studies of different organizations all of them related to Requirement Engineering and the corresponding processes areas in the ISO/IEC 12207:2008 were counted once not three times. In this specific case, the requirement engineering process improvement were interpreted as three process areas of this international standard categorization: Software Requirements Analysis Process, Stakeholder Requirements Definition Process, and System Requirements Analysis Process.

In very small organizations, 75% (6 of 8) of the improved process areas are about Software Implementation Process, Software Maintenance process, Project Processes, System Requirements Analysis Process, Stakeholder Requirements Definition Process, and Software Requirements Analysis Process. In small organizations 58% (33 of 57) of the improvements are about Software Implementation Process, Software Documentation Management Process, Project Processes, System Requirements Analysis Process, Stakeholder Requirements Definition Process, and Software Requirements Analysis Process. In medium organizations 48% (12 of 25) are about System Requirements Analysis Process, Stakeholder Requirements Definition Process, and Software Requirements Analysis Process. Finally, in large organizations 54% (37 of 69) are about Software Review Process, Quality Management process, System Requirements Analysis Process, Stakeholder Requirements Definition Process, Software Requirements Analysis Process, Project Process, and Life Cycle Management Process.

Besides the fact that the organizations' maturity in most of the studies is not explicit, we can infer from the analysis of the improved processes areas that large companies are in a better position compared to small and very small companies. In general, large companies have a defined process and some measurement methods. Only large and medium companies provide studies about the improvement of high maturity process (CMMI Level 4 and 5). A considerable amount of process definition and documentation process improvement initiatives are described in the context of small companies.

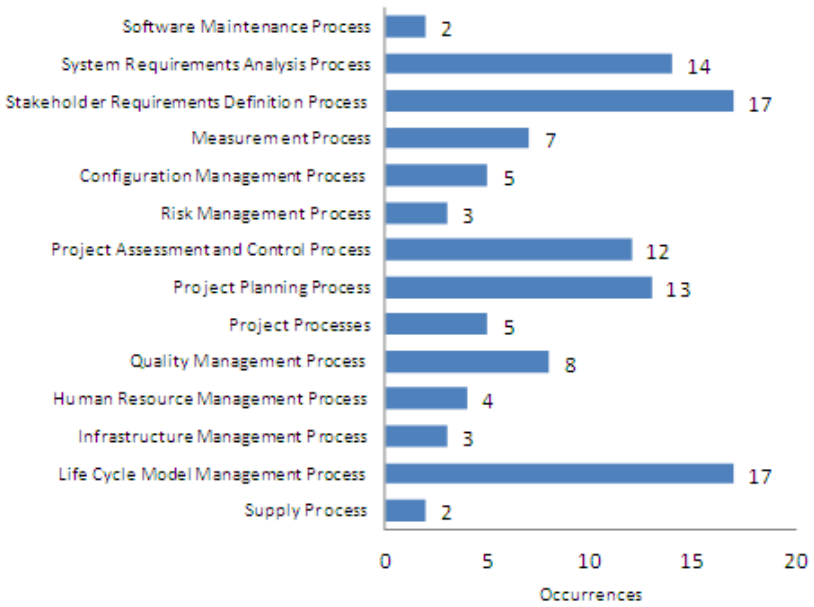

Figure 4.10. System Context Processes.

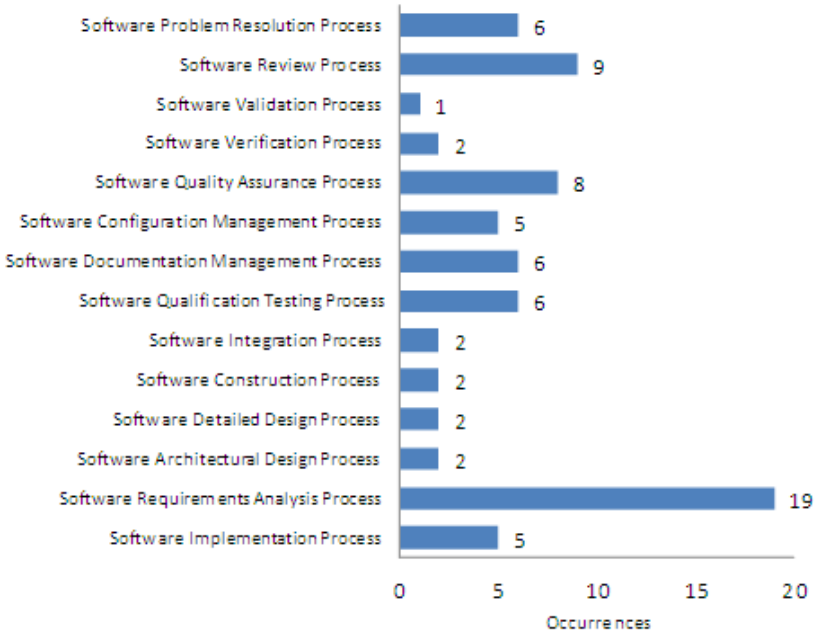

Figure 4.11. Software Specific Processes.

4.3.3 Qualitative Analysis

Defining the improvement steps of an SPI initiative requires an in-depth understanding of the processes to be improved, the organizational context and the people involved in the change. In order to share the knowledge of SPI initiatives we categorized these studies as patterns. These patterns were derived from the literature and they consist of a generalization of the usual elements of SPI initiatives carried out by organizations. It is by no means a comprehensive list of pattern sources. Indeed, emphasis is placed on the most recurrent aspects found during the literature analysis.

Software patterns became popular after the book of the "Gang of Four" [Gamma et al., 1995]. This book shows 23 design patterns with a fixed structure and a consistent representation and formatting. Various other researches have contributed to the enhancement of design patterns. Patterns are also popular in other areas, for example, the software and business process patterns [Størrle, 2001; Penker and Eriksson, 2000].

In the SPI context, patterns have been first presented by Appleton [1997] to overcome typical difficulties of SPI efforts. In Blanco et al. [2001], the authors present a consolidated view of business goals and the corresponding improved process area for a group of organizations that shared the same context. The patterns correspond to solutions (improved process areas) which address the goal. In another work, the authors present a more detailed description of software inspection patterns that can

help the improvement of the software inspection process [Harjumaa, 2005; Harjumaa et al., 2004]. These patterns come from the literature and they are intended to be a well-defined list of actions that are easy to understand and follow.

We were not able to conduct a very strict aggregation of research results, e.g. through meta-analysis or meta-ethnography [Barnett-Page and Thomas, 2009]. However, it is not our ultimate goal to compare specific techniques. In our case, more general concepts are more relevant to analyze than detailed implementation issues. We followed the subjective steps below in order to identify our patterns:

1. We identified common SPI initiatives in groups of "potential" patterns. By common, we mean SPI initiatives that address the same process area, in a similar organizational context.

2. For each group of "potential" patterns found in Step 1, we evaluated the problems and solutions adopted by those initiatives, evaluating if they had some similarities. Studies with too much diversity, for example, Project Processes improvements, were not included.

3. Whenever there were similarities in the group as mentioned in Step 2, we selected it as a pattern.

4. Studies in a group not having a suitable description of the initiative were excluded from the group. For example, a study describing two initiatives and with a complete description of only one of them, was considered only in the pattern for which it has a complete description.

In this section, we present the five identified patterns. Each pattern is described using the following template, derived from our research questions:

- An *Overview* of the pattern.
- The *Studies* that derived the pattern.
- The *Context* that describes the organizations' common characteristics.
- The *Symptoms* that present a set of usual problems and difficulties, which have been encountered before the start of the improvement initiative.
- The *SPI motivation* that presents the organizations' customary targets with the SPI initiative.

- The *Solution* that lists the common strategies adopted by organizations to solve the problems.
- The *Evaluation* that describes, in general, how the organizations evaluate the initiative.
- The *Lessons Learned* that lists the consolidate lessons learned from the improvement actions.

The SPI initiatives that were not so extensively reported in the literature or did not present similarities with others will have their characteristics discussed in the next section.

4.3.3.1 Patterns

1. Software Process Guidance

    - Overview: The new defined processes needed to be effectively guided or else only little improvement would be possible [Chrissis et al., 2006]. This guidance happened through printed process handbooks, but now organizations are shifting towards Electronic Process Guides (EPG) [Scott et al., 2002]. An EPG offers several advantages over a printed process handbook, including easy access over the web for the most up-to-date version of the process, navigation and search facilities [Kurniawati and Jeffery, 2006]. Several EPGs are now widely available such as RMC (Rational Method Composer) [7], EPF (Eclipse Process Framework) [8], and Spearmint [Becker-Kornstaedt et al., 1999]. Using one of these technologies the process guide can be created for different processes and sub-processes.
    - Studies: [Scott et al., 2002; Valtanen et al., 2009; von Wangenheim et al., 2006; Tuffley et al., 2004; Moe et al., 2005]
    - Context: This pattern was typically employed by immature and small organizations that had an ad hoc approach to development, usually never followed a defined development process, and the resources for process improvement are limited.
    - Symptoms: The assessments carried out in the studies corresponding to this pattern present the respective organizations as having a lack of internal

[7] http://www-01.ibm.com/software/awdtools/rmc/
[8] http://www.eclipse.org/epf/

procedures and a critical dependence on key resources. In some cases, even the existing procedures were not adequately documented.

- SPI motivation: Establishing a set of policies and procedures was the core motivation of the organizations in these studies. In addition, the study S57 [Tuffley et al., 2004] presented other secondary, but not less important non-process aspects, like improve efficiency within the organization, improve productivity, reduce development time and remove the dependence on key resources.

- Solution: The organizations used process standards like ISO/IEC 12207 or some other methodology to document and then personalize all phases of the development process. During this step, most of the organizations adopted an EPG tool in order to support the documentation activity of their process and to provide the usage feedback. Only the study S57 [Tuffley et al., 2004] did not specify the technology used to document the process.

- Evaluation: The measurements include the results of surveys that evaluate the EPG tool and the documented process. Furthermore, tool usage data was collected like the number of page hits versus time and time spent per week per person. Since the organizations have no historical data, they were not able to evaluate the quality of the documentation or to present some comparison. The results show that the most accessed features of the EPG tools were templates, checklists and examples. This is probably because they deal directly with the tangible outputs of projects. However, some usage trends were also observed, for example, the declining usage of the EPG caused by process learning and by access of templates or checklists from other sources (e.g. file server).

- Lessons Learned: The usage of the EPG tool brought about many more benefits than detriments. The organizations learned that the changes should be as small as possible in the first iterations in order to not produce a "too heavy" process difficult to be used. The perceived usefulness is an essential aspect to current and future usage [Moe et al., 2005]. One proposal to increase the use and benefit of an EPG is to involve all process performers in creating the process documentation [von Wangenheim et al., 2006]. In conclusion, the compatibility, ease of use, organization support and participation were the key determinants for a successful process guidance SPI initiative.

2. Defect Prevention Techniques

   - Overview: Defect prevention in the CMM and Causal Analysis and Resolution in the CMMI are focused on identifying the root cause of defects and preventing them from recurring. Actions are expected at a project level as well as at the organization level. One example of simple, low-cost approach for systematically improving the quality of software is Defect Causal Analysis developed by IBM [Card, 1998]. The principle of DCA is to collect defect reports, find frequent types of defects, discuss them with local developers, and let them suggest improved procedures.
   - Studies: [Casey and Richardson, 2004; Moritz, 2009; Li et al., 2006; Lauesen and Vinter, 2001]
   - Context: This process improvement initiative was presented by medium and large companies which defined objective goals, for instance, as delivery quality of 0.2 defects per KLOC [Li et al., 2006]. Two of these companies followed CMM or CMMI model to guide their process improvement [Casey and Richardson, 2004; Li et al., 2006], rather than to achieve a specific CMM or CMMI maturity level. A succinct description of the development process is presented in these studies. In study S10 [Casey and Richardson, 2004], the organization conducted a CMM-based process assessment of some selected process areas.
   - Symptoms: The organizations observed recurrent defects detected during the development or after delivery by customers. Examples of defects include already well-known problems of Software Engineering: incorrect bug fixes and missing requirements. In study S22 [Moritz, 2009], after the analysis of the defects detected by customers, the organization found that approximately 55% of defects were introduced by bug-fixes and not by new features in code. In study S69 [Lauesen and Vinter, 2001], the first major source of defects was missing requirements and the second one was mistaken tacit requirements (wrong solution to the requirements).
   - SPI motivation: The organizations presented business related motivation, for example, increase profitability [Casey and Richardson, 2004], better levels of quality to customers [Moritz, 2009; Li et al., 2006; Lauesen and Vinter, 2001] and development of new businesses [Casey and Richardson, 2004]. One process-related motivation was the improvement of the system test process [Moritz, 2009].

- Solution: The solution followed by the organizations was basically the one described in the Causal Analysis and Resolution process area of CMMI: analysis of defect data, classification of the defects and analysis of causes, establishment of some cost-effective prevention technique (as checklists), training, and evaluation of effects.
- Evaluation: The organizations knew correct measurement procedures, therefore the studies were able to provide quantitative measurements in order to verify the improvement results. The number of defects detected before and after the improvement was the main measurement used by organizations to evaluate the SPI initiatives (both defects detected during development and by customers). Other measurements include the cost savings and the reduction in the project delay (time-to-market). The organizations presented an expressive reduction in the number of defects varying from 35% in study S22 [Moritz, 2009] to 86% in study S26 [Li et al., 2006] and 70% in study S69 [Lauesen and Vinter, 2001]. Study S26 [Li et al., 2006] makes a comparison of the results against the business goal, and the results consisted of a reduction of 88% in the number of defects per KLOC as well as a reduction of 12 times (in days) of the delay to deliver the product. Considering these results, the SPI initiative provided expressive benefits to organizations. But in only one study it was possible to compare the effects of these changes with business goals. However, identifying and preventing recurrent errors have an expressive impact on quality (in terms of defects). Besides the fact we could not identify the amount of investment in the studies, we considered it relatively small for this type of process improvement.
- Lessons Learned: As defects were reduced and traced back to the source, better software is produced. This allows preventive and effective actions to be taken to stop systematic errors [Casey and Richardson, 2004]. Selecting the best techniques involves many factors, some of them quantifiable, others more subjective. One way to evaluate this technique is to look at the saved rework [Lauesen and Vinter, 2001].

3. Requirement Engineering Process

- Overview: Requirement Engineering (RE) involves the elicitation, definition, analysis and management of requirements. It is often cited as one of the most important and difficult phases of software development [Brooks, 1987]. Improvements to requirements activities can produce benefits such as

improving quality and preventing defects throughout software development. In our case, it comprises the Requirement Management process area of Level 2 CMMI and Requirement Definition process area of Level 3 CMMI.

- Studies: [Damian and Chisan, 2006; Holmberg et al., 2009; Kauppinen et al., 2004; Napier et al., 2009; Nikula and Sajaniemi, 2005; Kääriäinen et al., 2004; Higgins et al., 2002; Schmid et al., 2000; Caliò et al., 2000; Sommerville and Ransom, 2005; Börjesson and Mathiassen, 2004; Schneider, 2000]

- Context: This group represents the largest number of SPI initiatives described in the literature, with at least one article published every year. It shows that this is still a core subject in many SPI initiatives, mainly considering that studies of immature organizations focus on this type of initiative. The context consists of a quite broad range of organizations including very small, small, medium and large organizations. Most of them presented low performance capability with respect to Requirement Engineering (RE) processes, without an RE process documentation or with a documentation different from the practice (i.e. not appropriate for the organization) [Damian and Chisan, 2006; Kauppinen et al., 2004; Napier et al., 2009; Nikula and Sajaniemi, 2005; Sommerville and Ransom, 2005] and with a high dependence on individuals. Most of these organizations did not follow a reference model for their improvement. REAIMS [9] was selected by three organizations and CMM by one.

- Symptoms: We can observe two different organizations groups included in this pattern: one group represents immature organizations (Level 1 REAIMS or CMM), which did not have a defined RE process or with a documentation different from the practice (not appropriate for the organization)(Group 1) [Damian and Chisan, 2006; Kauppinen et al., 2004; Napier et al., 2009; Nikula and Sajaniemi, 2005; Sommerville and Ransom, 2005], the other group had an RE process but it still needed some adjustments (Group 2) [Holmberg et al., 2009; Kääriäinen et al., 2004; Higgins et al., 2002; Schmid et al., 2000; Caliò et al., 2000; Börjesson and Mathiassen, 2004; Schneider, 2000]. Group 1 includes almost all small and very small organizations of this pattern, one medium and two large organizations. None of the Group 1 organizations had a complete and documented RE process; they typically suffered from significant requirements creep, schedule and cost

[9] Requirements Engineering Adaptation and IMprovement for Safety and dependability [Sawyer et al., 1997]

overruns. They had difficulties in prioritizing task and pricing based on requirements documentation. On the other hand, SPI initiatives of Group 2 focused on specific problems of the Requirement Engineering process, for example, insufficient tool support [Holmberg et al., 2009; Kääriäinen et al., 2004; Börjesson and Mathiassen, 2004], difficulty to trace information [Schmid et al., 2000] or not well-founded methodological approach for the requirement analysis [Caliò et al., 2000].

- SPI motivation: The main motivation in this case is to define an RE process or to improve a specific activity. Other important reasons to improve the RE process include: reduce development time and effort, provide to the clients high quality business software products and services, increase professional skills, improve documents quality, and quickly and easily determine requirements.

- Solution: The solution followed by Group 1 includes the usage of models like REAIMS and CMM (4 of 5 studies, 80%) to guide the RE process definition. In Group 2 the solution was specific to each situation, but we can highlight that some articles [Holmberg et al., 2009; Kääriäinen et al., 2004; Higgins et al., 2002; Caliò et al., 2000; Börjesson and Mathiassen, 2004] describe an implementation or use of a tool support (e.g. RequisitePro or Rational Rose), while others [Schmid et al., 2000; Schneider, 2000] mainly present the improvement of a specific requirement activity.

- Evaluation: In group 1, most of the data consist of the evaluation of the maturity before and after the RE process improvement. In study S3 [Damian and Chisan, 2006], quantitative data includes reduction of the number of user product defects (45% fewer) and improvement of estimation accuracy (55% better). In the other group, the measurements include savings ([Holmberg et al., 2009]), time-to-market (reduction of the slip in approx. 50% [Higgins et al., 2002]), effort distribution and tool measurements.

- Lessons Learned: The lessons learned are mixed with the success factors in these initiatives. The important aspects are motivation, commitment, enthusiasm of personnel, involvement of senior engineers, close collaboration between software and process engineers, adoption of an evolutionary approach, and training. One important observation about the tools usage is presented in study S34 [Higgins et al., 2002]: “Tools are essential for managing requirements on an individual basis”, however “it is ineffective to

introduce a requirements management tool without a process, but on the other hand it is difficult to introduce a process without tools."

4. Measurement

   - Overview: Successful organizations implement measurement as part of their daily activities. Measurement provides the means to make informed decisions, that impact organizations' business and performance results. It is the main component to maintain an organization in a competitive place in a constantly changing environment. One sign of its growing importance is that Software Measurement has evolved into a basic practice in the Software Engineering area, as evidenced by the inclusion in Level 2 of the CMMI.

     The improvement of a measurement process provides a more accurate description of the current status of the process and products, therefore improving the visibility and the ability to assess (measure) the processes. Such visibility enables stakeholders to identify processes' performance deficiencies and to take actions to improve then.

   - Studies: [Iversen and Mathiassen, 2003; Larndorfer et al., 2009; Becker et al., 2006; Coman et al., 2009; Frederiksen and Mathiassen, 2004]
   - Context: This process improvement initiative is presented by three large organizations [Iversen and Mathiassen, 2003; Larndorfer et al., 2009; Frederiksen and Mathiassen, 2004], one small organization [Coman et al., 2009] and one with unknown size [Becker et al., 2006]. The models used are GQM, IDEAL and CMM. The process context was scarcely presented in these studies. The study S77 [Frederiksen and Mathiassen, 2004] was the only one that described how the measurement process operates in the organization, making pre and post comparisons of the measurement program.
   - Symptoms: The organizations already collected some of their project data, however they reported a lack of standards; metrics were not used to make informed decisions about future initiatives; some metrics and measurements definitions were not clear; data interpretation was not trivial, requiring more elaborated insights; and there was no managerial response based on the metrics collected.
   - SPI motivation: The primary objective of the organization in study S77 [Frederiksen and Mathiassen, 2004] was to increase quality and productivity. Therefore, it did not state the process-related reasons (if we ignore the alleged to improve in order to improve) for the process change as in the

other studies, e.g. increase process transparency [Larndorfer et al., 2009] and make the process measurable [Iversen and Mathiassen, 2003; Larndorfer et al., 2009; Coman et al., 2009]. Other reasons to implement metrics program consist of the achievement of CMM maturity Level 3 [Becker et al., 2006], improvement of the estimation accuracy [Larndorfer et al., 2009], enhance the ability to plan [Larndorfer et al., 2009], stay competitive, and provide more accurate information to all stakeholders [Larndorfer et al., 2009; Coman et al., 2009].

- Solution: The solution consists of the definition of activities to be carried out during the measurement program and the establishment of the infrastructure to support them. For instance, creation of the organizational repository for software development artifacts, usage of a tool to automate the collection, representation of the project quantitative data in a standard way, and providing support to the process analysis.
- Evaluation: The evaluation was mainly conducted through interviews and observations of the measurement results, i.e. mainly through qualitative data. The study S77 [Frederiksen and Mathiassen, 2004] was the only one that make a pre-post comparison of the program costs, showing a reduction of 57%. The other quantitative data provided by study S27 [Becker et al., 2006] consisted of secondary effects of the measurement program, for example, increasing defect removal efficiency from 82 to 93% and reducing the delivered defects density per function point from 0,21 to 0,15.
- Lessons Learned: Across the set of studies, we identified common themes that contributed to the projects' success, such as the following:
  - **Establish a project**: It is recommended to setting solid objectives and plans for the measurement program [Iversen and Mathiassen, 2003].
  - **Commitment**: It is essential to involve all stakeholders from the very beginning in the process improvement initiative to gain support from them [Larndorfer et al., 2009; Frederiksen and Mathiassen, 2004]. This can be done by ensuring data privacy, giving developers the access to information collected, giving the full control over their data, and taking into account developers' suggestions [Coman et al., 2009]. It is important that those who report the data feel safe that data will not in any way be used against them [Iversen and Mathiassen, 2003].
  - **Facilitate debate**: Establishing a clear organization-wide terminology leading to understandable measurements [Larndorfer et al., 2009]. It

is vital that the development organization be given the opportunity to provide feedback on the quality and relevance of the metrics program [Iversen and Mathiassen, 2003; Coman et al., 2009].

- **Start simple**: Simple measurements are better than complex ones. Therefore, focus should be on a few simple and high-quality measurements that are frequently consulted, rather than to produce an excessive variety of metrics that no one uses [Iversen and Mathiassen, 2003; Larndorfer et al., 2009].
- **Publish widely**: It is important that all stakeholders have access to the data, as well as informing about the importance of the data they provided and showing the results obtained from the measurement process [Iversen and Mathiassen, 2003; Coman et al., 2009].
- **Use the data**: Measurements should be useful in everyday activities [Coman et al., 2009]. "If the data are not used to gain insight and as a basis for making corrective actions the metrics program will soon degenerate into a bureaucratic procedure that merely adds to the overhead of developing software without contributing to the continuous improvement of the software operation." [Iversen and Mathiassen, 2003]
- **Verify the program**: The measurement data should be sufficiently accurate to support fruitful discussions [Iversen and Mathiassen, 2003]. Broad discussions serve to both critically evaluate the metrics program and as an important element in understanding the data and results. Numbers are not absolute truths, therefore it is vital that the data and their quality are discussed among those involved in and affected by the metrics program [Iversen and Mathiassen, 2003; Coman et al., 2009].

5. Quantitatively Managed Process

- Overview: Quantitatively managed process comprehends the concepts depicted in CMMI Level 4. In this pattern the processes are controlled using statistical or other quantitative techniques. The organization establishes quantitative objectives for quality and process performance, applying them as criteria in managing the process. The final goal is to produce desired results and achieve customer satisfaction.
- Studies: [Zhao et al., 2008; Redzic and Baik, 2006; Li, 2007; Gou et al., 2008; Russ et al., 2008; Murugappan and Keeni, 2003; Card et al., 2008]

- Context: This type of process improvement initiative is presented by medium and large companies that followed CMM/CMMI [Gou et al., 2008; Murugappan and Keeni, 2003; Card et al., 2008] or Six Sigma [Zhao et al., 2008; Redzic and Baik, 2006; Russ et al., 2008; Murugappan and Keeni, 2003] and DMAIC [Zhao et al., 2008; Redzic and Baik, 2006; Russ et al., 2008] as a guide to the process improvement implementation. While the studies S83 [Murugappan and Keeni, 2003] and S85 [Card et al., 2008] describe the process improvement steps of the organizations to high-maturity practices, other studies provide a very short description of their current organizational development processes [Redzic and Baik, 2006; Gou et al., 2008; Russ et al., 2008] or did not describe them [Zhao et al., 2008; Li, 2007].
- Symptoms: Almost all studies report problems related to the defect rates detected during development or after delivery: poor reviews efficiency [Russ et al., 2008; Murugappan and Keeni, 2003], systematic errors detected by customers [Zhao et al., 2008; Redzic and Baik, 2006], instable processes [Li, 2007], and undefined control points [Gou et al., 2008]. One study [Russ et al., 2008] focuses also on the inefficiency of knowledge retrieval.
- SPI motivation: The main motivations consist of achieving a maturity level of CMMI [Redzic and Baik, 2006; Li, 2007; Card et al., 2008], applying Six Sigma [Murugappan and Keeni, 2003], reducing the number of defects [Zhao et al., 2008; Redzic and Baik, 2006; Russ et al., 2008] and increasing retrieve effectiveness [Russ et al., 2008].
- Solution: The common solution is to apply the DMAIC [Zhao et al., 2008; Redzic and Baik, 2006; Russ et al., 2008] methodology to define problems, measure the aspects of the process, analyze the data, optimize the current process and control the process to avoid deviations from the target. The control is implemented through statistical techniques [Zhao et al., 2008; Redzic and Baik, 2006; Li, 2007; Russ et al., 2008; Murugappan and Keeni, 2003; Card et al., 2008].
- Evaluation: The studies collected a wide diversity of measurements such as cost, time-to-market, estimation accuracy, process quality, product defects, effort, knowledge retrieve efficiency and ROI, executing statistical analysis. The percentage of improvement includes a diversity of values from reducing failure costs in 80% [Murugappan and Keeni, 2003], increasing knowledge retrieval in 37% [Russ et al., 2008] and reducing defects density in approx. 99% [Card et al., 2008].

- Lessons Learned: The application of the statistical techniques improves decision making by making the results more objective, visible, repeatable and bounded [Card et al., 2008]. However, the quality of data is often not adequate to establish any firm conclusion [Russ et al., 2008].

4.4 Discussion

4.4.1 Research Questions

This section discusses what results of the SPI initiatives say about our research question:

"What are the realities of SPI initiatives?"

We summarize and discuss important findings considering the main and the refined research questions.

#1 Which are the most common reasons for organizations to implement SPI initiatives?

We had extracted text sentences containing information describing the motivation behind the launching of an SPI initiative. Then we classified these sentences in categories as shown in Table 4.6. Some of these categories were based on process characteristics [Staples and Niazi, 2008; Sommerville, 2011] and others on product quality attributes [ISO/IEC-TR-9126, 2001]. Even though this is a non-process related issue, reduction of defect rates is the main motivation for SPI initiatives. Evaluating the patterns, we also observed that, when not considering the circular SPI reasoning [Staples and Niazi, 2008], most of the patterns also did not present process-related reasons. The only pattern that presents more process related reasons is "Software Process Guidance" (the one involving process documentation). These motivations are consistent with the measurements that organizations used to verify and communicate their results (Table 4.7). Staples and Niazi [Staples and Niazi, 2008] also observed that the main motivations for adopting CMM-based SPI approaches are the product quality and project performance such as productivity, developing time and cost. Unfortunately, not all studies stated the organization business goals and needs of the respective SPI initiative. Indeed, a key success factor for SPI consists of a clearly defined initiative oriented by strategic business needs [Dybå, 2005].

#2 Which reference/assessment models the organizations adopted?

Table 4.6. Reasons for launching an SPI initiative.

Group	Motivation	Description	Frequency
Unclear	Not stated	A not defined reason.	2
Product	Defects	Reduce the number of defects detected after delivery.	16
	Service Quality	Improve the effectiveness of service delivered to the customer.	2
	Reusability	Improve the capability to use again specific functionalities.	2
	Usability	Increase the capability of the software product to be understood, learned, used and attractive to the user.	2
	Maintainability	Improve the capability to modify the software.	2
Project	Development quality	Improve the quality during development (e.g. reduce the number of defect).	14
	Time-to-market	Reduce the time needed to deliver the product.	12
	Costs	Reduce the costs required to deliver the product.	10
	Estimation Accuracy	Reduce estimation errors.	6
	Productivity	Increase the team performance.	5
	Effort	Reduce the effort required to develop the software.	5
	Communicability	Improve the capacity of communication among the team members.	3
Organization	Customer satisfaction	Increase the satisfaction in relation to the organization and the products.	9
	Market share	Enlarge market share.	5
	Competitiveness	Increase the ability and performance of the organization in a given market.	3
	Profit	Increase the profit margin.	2
Process	Process Improvement	Improve or define a specific process ("circular" reasoning).	35
	Conformance	Satisfy models (CMM, CMMI), standards (ISO 9000, ISO 9001) or other specific frameworks.	21
	Consistency	Increase process consistency in different groups of the organization.	8
	Measurability	Develop the ability to measure the process.	3
	Documentation	Create a process documentation guidance.	3
	Visibility	Improve the transparency of development process status.	2
People	Motivation	Increase individuals' enthusiasm.	3
	Knowledge	Increase professional and organizational knowledge.	2
Others	Other reasons	Other reasons that were not classified in one item above, e.g. improve knowledge retrieval and tool consistency, and reduce integration delay.	7

The CMM/CMMI were the most reported reference model adopted by organizations. They were reported in 31 studies, applied by organizations of all around the world, including eight studies in America, seven in Europe, six in Asia, two in Oceania and two in Scandinavia. The ISO 9001 and ISO 15504 standards were adopted by three organizations in America, three in Europe, one in Scandinavia and one in Australia. These results show that the application of the SEI models and ISO standards are not restricted to some location, e.g. CMM/CMMI in North America, and ISO standards in Europe.

#3 What are the main aspects that are measured in SPI initiatives?

In the studies, a restricted number of measurements was selected to monitor the results of the change. Table 4.7 depicts the main measurements performed in the quantitative case studies. This table shows that quality, translated as number of defects, is the most measured attribute, followed by effort and cost. In most of the cases, the side effects of the improvement in other measurements were not considered. Thus, the focus on a restricted measurement set may obfuscate the actual effects of the SPI in the organization's measurements. The restricted number of measurements can be interpreted in several ways. It can be interpreted, for example, that the companies are not willing to provide information that could be used by competitors, but for this specific case we noticed that the studies would use, in general, an indication of "information withheld". Our interpretation is that the companies have difficulties to measure and analyze quantitatively their improvement.

During our analyses, we observed that it was not possible to make a more accurate comparison among the studies due to the diversity of contexts of the SPI initiatives, the different measurement units, and the lack of the information about the amount of investments and benefits assigned to the SPI initiatives. In order to check the improvement percentage, we arbitrarily aggregated them in categories: less than 5%, between 5 and 35%, between 36 and 50 %, between 51% and 70%, between 71% and 99% and 100%. The measurements used in the studies are depicted according to their category in Figures 4.12 and 4.13, except for the categories "Others", "Tool measures", "Savings", "ROI" and "Profit". These last categories are not depicted due to the fact that they are not commonly used in SPI evaluations. The category "Others" are a mixture of items. The category "Tool measures" have dissimilar measures such as access rate and storage volume. Typically, the studies of category "Savings" provides only the amount of money accrued from the improvement. The category "ROI" provides the ratio of investments and only study S39 provides the improvement percentage of the "Profit" category.

We observed that the *Process quality* and *Product defects* measurements have higher improvement percentages (greater than 71%) (Figure 4.12). The frequency of these measurements and their positive benefits indicates that they are widely used to improve visibility of the SPI results to the practitioners and managers. Only in four studies we observed improvements of 100%, one in *Productivity* [Sutherland et al., 2008], two in *Process quality* [Rautiainen et al., 2003; Kenni, 2000], and one in *Estimation accuracy* [Rautiainen et al., 2003]. Although the studies used some measurements, there are doubts about their validity. Considering, for example, the *Process quality* and *Product defects*, some studies compared only the number of defects that have not been normalized by the size or LOC (lines of code) [Becker et al., 2008; Karlström and

Table 4.7. SPI measurements.

Measurement	Description	Studies	Frequency
Product defects	Represent the number of defects detected after delivery.	S3, S7, S15, S16, S19, S22, S26, S28, S29, S32, S36, S39, S45, S46, S47, S58, S65, S71, S72, S85, and S90.	21
Process quality	Indicate the quality of the development process, including the number of defects and the effectiveness of quality activities (test, review) in detecting defects.	S5, S7, S19, S23, S24, S26, S35, S48, S61, S65, S69, S71, S72, S78, S82, S86, S89, and S91.	18
Effort	Represent the effort to perform an activity.	S4, S5, S16, S24, S40, S45, S47, S48, S55, S56, S58, S65, S72, S78, S82, S91	16
Cost	Depict the monetary resources needed to complete the software development.	S5, S7, S24, S29, S36, S46, S56, S77, S83, S85, and S90.	11
Estimation accuracy	Consist of the difference between the planned and the real value.	S3, S15, S28, S46, S49, S65, S71, S83, and S91.	9
Time-to-market	Represent the length of time that the software product takes to be available to customers.	S2, S7, S26, S34, S46, S54, S71, S83, and S90.	9
Customer satisfaction	Represent the customer satisfaction in relation to the organization and the products.	S6, S7, S39, S46, S65, and S90.	6
Productivity	Indicate a measurement of the technical efficiency of the development team.	S5, S7, S24, S29, and S89.	5
Savings	Describe the amount of money that is saved during the development cycle, through for example, reducing recurrent defects.	S9, S22, S55, S61, and S83.	5
Tool measures	Include operation measurements of a particular tool (which cannot be mapped on the previous measurements), e.g. number of requirements mapped into a tool, number of access to some information stored in a tool.	S1, S18, S67, S70, and S78.	5
ROI	Represent the SPI's Return On Investment (ROI). It is calculated by dividing a financial representation of the benefits by a financial representation of the cost.	S1, S7, S12, and S72.	4
Profit	Represent the different between the purchase price of the software product and the development costs.	S7, S39, and S89.	3
Others	Group all other measurements that are not part of any of the above measurements and that are presented as a result of the SPI initiative (e.g. number of change requests, the degree of parallelism during development, and number of documents).	S2, S6, S7, S32, S47, S54, S56, S71, S78, and S83.	10

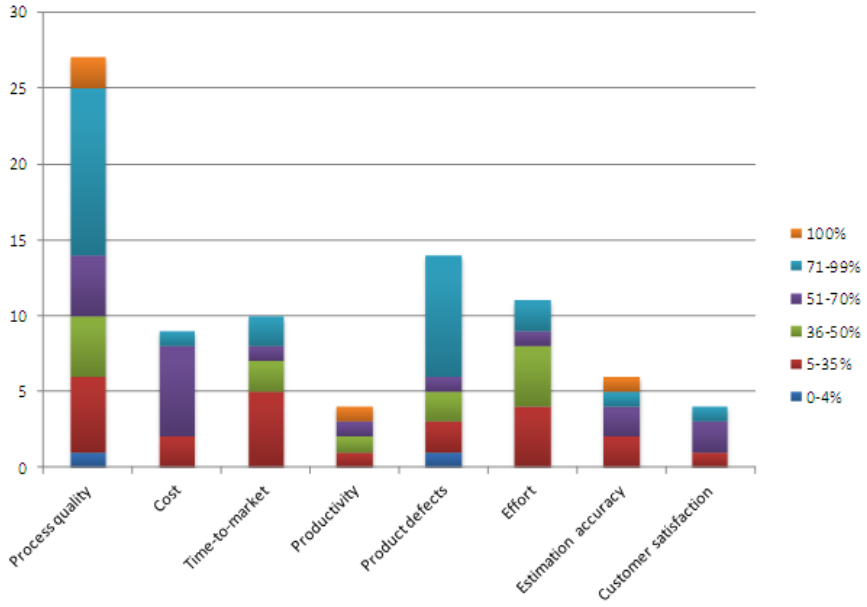

Figure 4.12. Distribution of the improvement percentage across measurements.

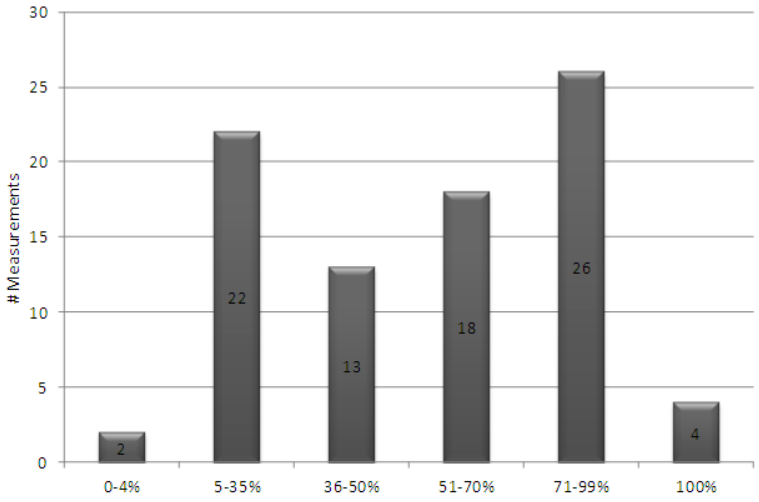

Figure 4.13. Distribution of the number of improvement percentages in each category.

Runeson, 2005] and they do not discuss the defect severity [Harter et al., 2011]. In addition, it was not possible to assign the improvement solely to the process changes [Sutherland et al., 2008] or the comparison occurs between projects that may not be considered similar. Finally, as presented in Section 4.3.1, most of the studies did not analyze possible biases and threats occurred during SPI initiatives, e.g. changes in the organization or team size.

#4 What are the lessons learned with the SPI initiatives? What are the main problems? What are the main problems during the different phases of a cycle? What are the main problems during the different cycles? What are the usual solutions?

This information is particular to the SPI initiative and organization context. As we have seen in the previous section, we were not able to generalize all the SPI problems, solutions and lessons learned. Therefore, we focused on a set of five patterns that are recurrent in the studies.

At closer examination of the results, it is possible to observe that, with exception of a few SPI studies [Kääriäinen et al., 2004], most of them reported successful initiatives. However, this issue is considerably more complex. A success or failure in SPI initiatives depends on the viewpoint of the practitioners involved in the improvement. The same process improvement can be a success from one viewpoint and a complete failure from another [Johansen and Pries-Heje, 2007]. Thus, different perspectives should be taken into account when evaluating the results. The majority of the studies analyzed the process improvements considering only one viewpoint. This indicates a shortcoming in the applied methodology. In addition, as we discussed in question #3, the SPI studies have an inclination to present only a restricted number of positive measurements, not considering important implicit relations. This confirms again the difficulty of the companies to measure and analyze quantitatively their improvement.

4.4.2 Guidance for Reporting SPI Initiatives

With respect to research papers addressed by this SLR, a relatively large number of studies did not report in a clear and defined way the business motivations, context, data collection and analysis of the SPI initiatives. There are many guidelines [Höst and Runeson, 2007; Runeson and Höst, 2009; Shull et al., 2008; Kitchenham et al., 1995, 2002] for conducting and reporting Software Engineering empirical research. However, we were not able to find general guidelines for reporting SPI initiatives. We decided to reinforce some important aspects that are not well represented in these papers, which we believe can be useful in future researches.

SPI motivations should have a business orientation: One ultimate target of any change program is to be in accordance with the business needs [Beer et al., 1990]. Studies indicate that an SPI program driven by business needs has a strong chance of success (e.g. [Dybå, 2005; Brodman and Johnson, 1995]) (as discussed in Section 4.4.1). Therefore, this is an important information that should be collected during the goals definition. In the analyzed papers, even in mature organizations, very

few studies provided such information.

The context should be clear and defined: In order to reuse and draw valid conclusions of SPI studies it is important to define the context where the initiatives happen. In our analysis, nine studies (10%) did not describe the context and 32 (35%) described partially the context where the SPI was carried out. Petersen and Wohlin [Petersen and Wohlin, 2009] structure the context for empirical industrial studies, providing a checklist to aid researchers during the selection of the context information. Besides the six different context facets proposed by them (product, processes, practices and techniques, people, organization and market), in the SPI case, it is also important to provide information about the pilot project where the change will be carried out, such as estimates of productivity, cost and schedule, making, therefore, comparisons feasible.

The overall SPI initiative description should be reported: In order to reuse or adapt the SPI experience to different contexts, we judge it essential that researches describe the following items: the improved process area, the duration of the SPI initiative, the investments in the improved process, the problems detected before and during the execution, the proposed solutions, the applied reference model (e.g. CMM or ISO), the improvement guides (e.g. IDEAL), the assessment methods (e.g. SCAMPI), the measurements, lessons learned and a comparison between the costs and benefits (e.g. ROI). Since measurement is a central point in evaluating the SPI program, we will discuss it again in the next item. With the exception of the investment information and the analysis of costs and benefits, most of these items were presented, to some extent, in the studies.

The data collection and analysis should be clearly reported: When we evaluated the studies, a considerable number either did not present or presented insufficient information about the data collection and analysis (Table 4.8).

Table 4.8. Quality of data collection and analysis.

	Not stated	Insufficient information
Data collection	38 (42%)	25 (27%)
Data analysis	55 (60%)	12 (13%)

Since the measurement is considered an important step to evaluate, communicate and assist the ongoing change [Dybå, 2005], a better definition, collection and analysis will permit a deeper understanding of the studied phenomena. Therefore, it will be possible to make more elaborate comparisons and analyses. Another problematic aspect is the quality of the raw data used to support the findings. One way to reduce the

effects of poor data quality is to execute data triangulation. The analysis of multiple data sources (triangulation) will help the data interpretation [Dybå, 2005]. Finally, the analysis of the results from different viewpoints is essential to make more general conclusions about the SPI success or failure rates [Johansen and Pries-Heje, 2007].

The validity of the study must be analyzed: The validity denotes to what extent we can trust on the results, since they can be biased by the researches' point of view [Dybå, 2005]. The studies have serious limitations in terms of validity and credibility of their findings. Very few analyzed the bias (13 studies, 14%) or addressed the threats to validity (21 studies, 23%). This problem has also been detected by other SPI studies [Unterkalmsteiner et al., 2012; Staples and Niazi, 2008].

4.5 Related Work

Researchers are aware of the importance and the challenges involved in the literature search. They have conducted SLRs in order to gain a comprehensive view of the available evidence in the SPI field, as well as in different disciplines of Software Engineering. Table C.4 (Appendix C) summarizes a comparison among five SLRs that we identified in the SPI area.

In the work of Unterkalmsteiner et al. [2012], the goal is to identify and characterize the evaluation and measurement strategies used to assess the impact of different SPI initiatives. The "pre-post comparison" (i.e. the before and after SPI implementation comparison) was the most common evaluation strategy used to assess the SPI initiatives. They also found that CMM is the most reported framework and the process quality is the most measured attribute. Their results support the findings of insufficient evaluation of potential confounding factors in SPI studies. As for the selection of primary studies, our work covered 1999-2009, a different set of the 1991-2008 time frame considered in their work. In addition, the difference between the search strings and research questions emphasizes the distinct scope of their study and ours. When comparing their selection with ours we found 23 in 110 overlapping articles. Recall that despite a final counting of 91 studies our search identified 110 relevant studies as mentioned in Section 4.3.2.1. Besides these distinctions, we found some similar results mainly regarding the measurements used, frameworks' type and the common problems found in these types of reports: incomplete context descriptions and lack of the threats of validity and bias analysis.

Pino et al. [2008] provided a literature review of SPI in small and medium software enterprises. Among the usual search databases, they also included grey literature

(Proceedings of the First International Research Workshop for Process Improvement in Small Settings). Since, we used the same guide to classify organizations' size [Commission, 2005], we can say that our work has a broader scope. "Project management" is the most improved process area and, again, CMM is the most used reference model in these companies. When comparing their selection with ours, including the papers that we considered identical (110 papers in total), we found 17 overlapping articles.

Sulayman and Mendes [2009] identified existing SPI models and techniques used by small and medium Web companies. They identified four studies, including one master thesis. The results did not suggest any specific model or technique to measure the results of the SPI in Web companies.

In Müller et al. [2010], the authors evaluated the SPI publications with a main emphasis on organizational change. They use Gareth Morgan's organizational metaphors Morgan [1996] as analytical approach: machine, organism, brain, culture, political system, psychic prison, flux and transformation and instrument of donation. The results reveal that the main SPI contributions are from Scandinavia (34%) and the Americas (32%). A large percentage of organizations (83%) was viewed as organisms, machines, flux and transformation, or brains. The other metaphors (political system, psychic prison, and culture) are more scarcely represented or not represented at all (instrument of domination). As many as 76% of the articles were mainly descriptive in nature, using theory to drive their empirical investigation as evidence (56%). In this work, they excluded articles that did not deal with organizational change. We identified only nine articles identified in their work.

Staples and Niazi [2008] investigated the motivations most frequently reported by organizations for adopting CMM-based SPI. They observed that companies are more concerned with product quality and productivity. We also observed the product quality trend when we evaluate the motivations for carrying out SPI initiatives. Staples and Niazi stated that this is an important reason for SPI implementation, but it is not the main reason for implementing SPI. According to them, two examples of common process reasons are to make a process more visible and measurable.

Other studies present an analysis of the SPI field, although they either were not conducted as a systematic review or they had no focus on the analysis of the SPI results. For example, Rainer and Hall [2001] analyzed 39 SPI publications that appear to be frequently cited by others. Two main issues emerged from their research were the organizational stability (staff stability and re-organizations of the corporate division) and process performers expertise. They conjecture that organizational stability provides a stable environment where it is possible to perform the process and to develop expertise. In another study, Niazi [Niazi, 2006] examined the impact of the SPI on the

organizations' capability and what is missing in current SPI approaches. Through a literature review, he observed that SPI can help organizations in reducing their cost, improving time-to-market, productivity and customer satisfaction. In addition, little attention has been paid to the implementation of the SPI standards and models. In Hansen et al. [Hansen et al., 2004], the authors classified 322 representative SPI contributions according to their primary goal whether they are prescriptive, descriptive or reflective. According to them, the field is dominated by CMM approach and are biased towards prescriptive contributions. Finally, a considerable number of studies evaluate the success and failures factors for conducting an SPI initiative [Dybå, 2005; Rainer and Hall, 2002, 2003; Wilson et al., 2001; Baddoo and Hall, 2002a,b, 2003; Niazi et al., 2005].

Although interesting results have been addressed by these studies, there are three other main issues that differentiate our study from them: (1) The focus of our work is on bringing insights into the current state of SPI Body of Knowledge; (2) We have a more comprehensive focus, considering the analysis of the SPI results; and (3) We perform a systematic review, allowing us to answer our questions with a certain level of confidence.

4.6 Limitations

The use of systematic procedures itself helps to avoid problems with the selection and analysis of the studies. Even though this SLR has been supported by a pre-defined study protocol and conducted in a systematic way and under the guidance of experts, it has some limitations. We will discuss these limitations considering four aspects [Kitchenham, 2004]:

- Construct Validity: to what extent the selected studies represent what we aim to investigate;

- Reliability: to what extent the data collection and analysis were conducted in a way that it can be repeated by other researchers with the same results;

- Internal Validity: to what extent the design and conduction of the study is likely to prevent systematic errors; and

- External Validity: to what extent the effects observed in the study can be generalized.

4.6.1 Construct Validity

Terminology and its validation. We discuss the (i) choice of terminology and (ii) verification of relevant studies selection [Engström et al., 2010]. Since the search for primary studies is based on the search string, each SLR is likely to miss relevant studies if this string is not properly chosen. In order to minimize this threat, we applied the following steps: (1) derive and validate a specific search string; and (2) derive, validate and compare other search strings. In the first step, we derived the search string "software process improvement" AND empirical (henceforth SS1), using the PICO structure. We carried out a manual search through all studies published from 2005 to 2009 in two specific journals, i.e. *Journal of Information and Software Technology* and *Journal of Systems and Software*. Then, we conducted a search aiming at verifying if the manually selected studies were retrieved. Only one study was not found by the search string SS1. However, after evaluating it, we decided that this study was not relevant to our research and therefore the search string was considered satisfactory. Next, in the second step, we validated the search string SS1 in *ACM Digital Library* and *IEEEXplore* in the period of 1999 and 2009 and compared the results with three other search strings, namely: software AND "process improvement"(henceforth SS2); "software process improvement" AND empirical (henceforth SS3); and "software process improvement" (henceforth SS4). Then, one valid result [Kautz et al., 2000] was not found by SS1 and SS3 and another valid result [Damian et al., 2002] was not found by the search string SS4. Thus, we decided to use a broad string SS2 that may include a considerable number of false-positive papers, but probably will not eliminate papers that are essential to our research (See Section 4.2.4).

Completeness. Our research is limited to the databases mentioned in Section 4.2.3. In other words, we cannot claim that we include all relevant studies to our research question. However, the database choice considered our knowledge of important venues. As far as we know, the chosen databases have the most relevant publications. Therefore, despite the fact that we are not able to guarantee completeness, we believe that the selected studies represent a good coverage.

Grey literature. Besides database limitations, we excluded all grey literature (e.g. technical reports, some workshop reports, and work in progress) that may present SPI results. The main reason is that we were not able to devise a good solution for assessing the quality of grey literature and therefore we decided to leave the evaluation of these studies as future research. We believe that excluding these studies did not impact the overall results obtained as they normally do not present significant SPI results.

To sum up, we expect that the countermeasures taken to minimize threats to Construct Validity were enough to maximize the number and quality of relevant studies in our research.

4.6.2 Reliability

Some countermeasures were taken to reduce the threats to Reliability. Since we followed Kitchenham's [Kitchenham, 2004] procedures (i.e. we defined a research question, the selection process, inclusion/exclusion criteria and quality criteria), we believe that the reliability threats were minimized. However, adoption of systematic procedures in itself does not guarantee reliability. Therefore, we will discuss *Reliability* in terms of classification and analysis of studies (i.e. inclusion/exclusion criteria and data extraction) and review conduction.

Classification and analysis of studies. The way in which classification and analysis of the studies are done in SLRs is a threat to reliability. Mainly because they are based on the reviewer's knowledge and experience. The steps one and two of this study suffer from this threat due to the fact that the titles and abstracts do not always have the most relevant information (i.e. relevant information is sometimes omitted) [Budgen et al., 2008] and steps three and four are subject to the perception of the reviewers. In order to minimize this threat, in all stages of this study at least two reviewers were involved, and in case of disagreement the reviewers discussed to reach a consensus.

Review conduction. The reliability of this study was improved by the fact that the first author participated in all the steps and review sections, either as a reviewer or as supporter in the decisions of inclusion, exclusion and quality criteria. The participation of the first author and also the supervisors ensured a series of countermeasures related to possible threats.

We expect that, since we reported all the procedures taken and limitations, this research has high reliability and can be replicated by other researchers.

4.6.3 Internal Validity

We will discuss Internal Validity in terms of the **member's roles**, **reviewers bias**, **publication bias**, and **pattern description**. In our study, as mentioned in Section 4.2.2, we defined **specific roles** to each member. Important roles include the supervisors and the TL who were responsible to review how the study had been conducted and guarantee that the reviewers were correctly following the established procedures.

This participation helps controlling bias in the results. **Reviewer bias** is another potential threat to Internal Validity as the extracted data has a qualitative nature. In order to mitigate this threat, the quality attributes were grouped in patterns to facilitate their further classification and comparison. The **publication bias** refers to the probability of publication of more positive results than negative ones [Kitchenham and Charters, 2007]. While we cannot fully exclude the possibility of this threat, we reduced it to the extent possible as we followed a formal search strategy (described in the protocol) to find the entire population of publications including the "negative" results. The **patterns description** uses natural language, which is not adequately precise. However, our patterns are abstraction of real-life problems, so we decided to use natural language to represent them. Mainly because we do not concentrated in many details, where pattern descriptions can become impractical.

4.6.4 External Validity

The results of this SLR were considered with respect to specific studies in the SPI domain. Therefore, the conclusions and classifications were valid only in this given context. The results of our current study were drawn from qualitative analysis and can serve as a starting point for future researches. Additional studies can be analyzed accordingly.

4.7 Conclusion

Overall, this chapter provided an up-to-date view of the SPI area, showing common characteristics and results of the SPI initiatives.

We identified 91 studies that describe how process improvement initiatives were implemented, and what their results are. As we stated in the beginning of this section, we were originally expecting that most of the studies would focus on process aspects. As discussed in Section 4.4, among all the stated reasons for launching SPI efforts, 56% is about product, project, organization and people and, when we do not consider the circular SPI reasoning, only 20% has some process discussion. So most SPI studies do not focus on processes.

As discussed in Section 4.4, and summarized in Figures 4.12 and 4.13, in most cases the improvement percentage observed is less than 70%. The studies reporting 100% improvement are special cases, where it was either not possible to assign the improvement solely to the process changes, or the adopted practices do not allow measurement comparisons. This shows that there are no "dramatic" results associated

with SPI efforts. If the studies had followed more rigorous reporting guidelines, then we would be able to synthesize more helpful information for researchers and practitioners. For the time being, we were only able to rebut the anecdotal evidence of "dramatic" improvements.

In Section 4.3.3, we arranged SPI initiatives in patterns reflecting studies considering similar aspects. These patterns enable improvement category discussions, rather than the individual initiatives. We identified five SPI initiative patterns in our study selection. During our study we were able to take advantage of the categories during some of the meetings. However, as discussed, these patterns are specific for our study selection and future work should investigate their usefulness and coverage for other study selections.

We also identified some weaknesses of the SPI reports, discussed in Section 4.4.2. From that, we compiled guidelines as to how SPI results should be reported.

Overall, SLRs with higher value to industry and academia will be possible if organizations follow more disciplined SPI procedures with the correspondent reflection in the studies.

This chapter concludes the discussion of the most important concepts related to SPIAL. The next chapter will present how these concepts were considered in SPIAL specification, design and validation.

Chapter 5

SPIAL

In this chapter, we discuss the design, development and evaluation of SPIAL (*Software Process Improvement Animated Learning Environment*), our SPI simulation game. SPIAL is a graphical and interactive simulation game, which emphasizes SPI concepts. Despite its focus on SPI, it is a teaching tool that reinforces and introduces Software Engineering concepts to students. The aim of SPIAL is to improve Software Engineering education in dealing with the complexity of what happens in selected contexts of software development organizations. Using SPIAL, we evaluated the potential of an SPI simulation game in enabling students to learn some of the best practices of Software Engineering.

We adopted an incremental and iterative approach, where each step involves different knowledge, allowing us to point out a set of important aspects that should be taken into account during the development. These aspects can guide new developers and instructors in the design and selection of educational simulation games. We also discuss the issues and challenges associated with the creation process of a Software Engineering simulation game. These results correspond to the work presented in the 42nd Frontiers in Education Conference [Peixoto et al., 2012a,b].

This chapter is structured as follows:

- Section 5.1 presents a brief overview of SPIAL.
- Section 5.2 details SPIAL goals, in order to identify and clarify the objectives.
- Section 5.3 discusses the identified SPIAL requirements. These requirements were based on: (i) other Software Engineering simulation games; (ii) SPI literature review (measures, SPI patterns, motivators and de-motivators factors); (iii) communicability evaluation (feedback aspects); and (iv) CMMI technical report.

- Section 5.4 depicts FASENG, a framework for development of Software Engineering simulation games.
- Section 5.5 presents SPIAL evaluations from two viewpoints: specialist and player.
- Section 5.6 provides an overview of the differences among SPIAL and other simulation games.
- Section 5.7 concludes this chapter.

5.1 Background

The objective of a course is not just to cover a certain set of topics, it involves educating students, facilitating students' learning and thinking [Svinick and Mckeachie, 2011]. The instructor should understand that while learning objectives help the instructor to identify elements that will contribute to an effective learning environment, a wide range of influences that are students dependent exists, all affecting the student's learning experience, such as: student's ability to comprehend a specific content, student's ability to apply that knowledge, student's attainment of critical thinking skills necessary to effectively utilize the knowledge, student's past experience, and attitude about learning as well as their expectations. All these influences impact the learning environment and the individual student's ability to make use of the learning process.

Simulation games are an appropriate complementary approach to the traditional educational techniques. Students' preparation can be improved by allowing them to practice, through a simulator, activities that are infeasible to practice during a Software Engineering course, due to restrictions of time and resources. Simulation games therefore should be available for all courses. However, the creation of a simulation game is not a straightforward activity. The primary purpose of educational simulation games development is to match the learning objectives tightly with a suitable solution [Martin, 2000]. The design and development is an iterative process, involving a number of different skills and addresses different viewpoints. During this process, designers select the conceptual content to be addressed, model the real world processes, and employ suitable game-like elements to represent these processes and concepts. The final goal is to create an environment that can motivate and engage students. These steps are illustrated in Figure 5.1 and a detailed description of SPIAL development is presented in the next sections.

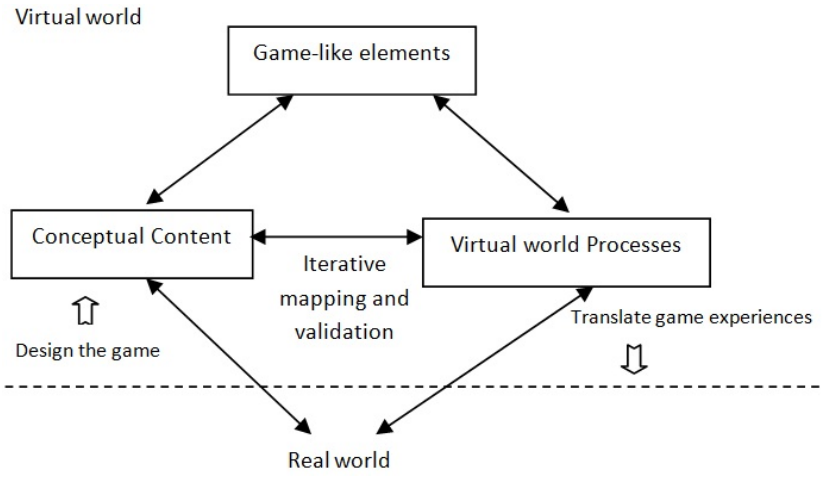

Figure 5.1. Educational simulation games design (adapted from Martin [2000]).

Most of the simulation games have as their goal to develop a software project within a certain set of constraints, and their rules are based on Software Engineering lessons (e.g. SimSE [Navarro, 2006], SESAM [Drappa and Ludewig, 2000], SimJavaSP [Shaw and Dermoudy, 2005], and MO-SEProcess [Ye et al., 2007]). Typically, they present a metaphor of a software development office, where the player assumes a project manager role. The experimental evaluations of using these tools have shown a restricted number of new concepts that students have learned after playing them [Navarro, 2006; Gresse von Wangenheim et al., 2009]. This can be a result of a not straightforward performance feedback that is presented to the students, or the way that the virtual world was designed. In order to address these aspects, we changed the context from Project Management to SPI and we integrated feedback during the whole simulation game playing.

5.2 SPIAL Goals

Identifying and clarifying objectives is of major importance for avoiding misunderstandings, for evaluating designing and for enhancing knowledge [Tchounikine, 2001]. We defined the SPIAL highest level objectives, i.e. conceptual level objectives, and their implementation, from the instructor and player viewpoint, i.e. the educational objectives of teaching and learning. In our case, teaching consists of the instructional process that gives students knowledge and learning is the process of acquiring a piece of domain knowledge, skill or some competency [Tchounikine, 2001].

Below, we present the educational objectives in SPIAL context:

- Teaching Objectives: Considering the instructor perspective, the objectives of a simulation game can be leading students to engage themselves in organizational issues of the software process, explore new perspectives, make them challenge their current knowledge, ask questions and make discoveries [Tchounikine, 2001]. Particularly, in the simulation game environment, the goal is to support students in the identification of interesting issues, definition and testing of solutions, enhancing motivation, developing positive bases for collaboration and drawing rational conclusions. In SPIAL context, the main objective is to allow students to practice Software Engineering area concepts, through an SPI initiative simulation game. This simulation game development is based on investments in CMMI process areas. CMMI conveys the Software Engineering concepts represented in SPIAL. CMMI was chosen because it is the most worldwide famous SPI reference model. In this way, SPIAL addresses issues of SPI and Software Engineering, allowing students to discover and practice concepts.

  A more concrete objective corresponds to the development and application of educational tools as simulations in order to provide more realistic experiences. This enable learners to reach an end product of learning. In SPIAL context, it consists of the development and personalization of an SPI interactive simulation game according to the teaching objectives.

- Learning Objectives: The learning objectives include the problems evaluation and the proposition of solutions, i.e. the application of knowledge in a concrete situation. Knowledge of how to play SPIAL is accumulated through observation and active participation in the gaming process. Players evaluate process improvement problems and propose solutions to them. Strategies in playing computer games included trial and error, reading instructions, relying on prior knowledge or experiences, and developing a personal game-playing strategy. Trial and error was by far the predominant strategy across all game types [Dempsey et al., 2002]. Trial and error is the core strategy envisioned for SPIAL. It involves actions and reactions to circumstances, consequences, and feedback during the game play. SPIAL helps students in the construction of knowledge because it allows them to observe in a systemic way how different conceptual aspects are articulated such as: requirements, measures, processes, team behavior, and estimates.

Section 5.3 presents a brief description of the SPIAL requirements. SPIAL was constructed based on FASENG, a Framework for Software Engineering Simulation Game (Section 5.4). This framework allows instructors to tailor a simulation model

according to their course. Both SPIAL and FASENG were created based on the objectives described above and their requirements have some interplay.

5.3 Early Decisions

SPIAL is a single-player game in which players take on the role of a manager of an SPI group in a software development organization. The player is given a process improvement task and he/she can interact with other stakeholders (high level management, project manager, team member, consultant, or customer) represented as Non-Player Characters (NPC), i.e. a character controlled by the computer. In order to complete the task, the player can make investments for improving specific process areas of a software development project (which can be considered a pilot project). A good investment strategy will result in improvement of process areas and a larger budget for further investments. SPIAL incorporates some of the concepts defined in CMMI. In the first SPIAL version, we considered an organization with maturity levels 1, 2 and 3 (incomplete), with capability levels of process areas varying between 0 and 3.

The player can visualize project estimations, indications of process areas capability level and decide in which process area to invest. During project execution, the player can visualize the effects of his/her selections on the outcomes (productivity, defect, cost, and time-to-market measures) and if needed, change his/her investments. The occurrence of events follows some probability distribution, for example, the software development team can have resistance to adopt the changes, and the player can overcome it with specific actions, such as promoting training. The final outcome is a score that represents how close the results are to the initial proposed target. During the game, the non-player characters communicate the effects of the player's actions through bubbles over their heads, for example, "Since there is dependency among process areas some of your investments may not be effective", or "Poor investment decisions result in a reduction of business value, and a reduction in the number of investment points."

For illustrative purpose, we elaborate the following initial scenario, which is a narrative description of what the user (in our case, the player) performs and experiences as he/she tries to make use of the application [Carrol, 1997]:

You are a student of a Software Engineering course and you already know some concepts of Software Engineering. Now, you want to learn more about Software Engineering, with emphasis in software process improvement using a simulation game. You need an intuitive game to help you learn how software process improvement works in the industry, the software process improvement techniques and the best practices of

*Software Engineering **[motivation]**. In this game, you will play as a manager of an SPI group **[player role]** and your responsibility is the coordination of a process improvement in a specific project. You need to analyze the current situation of the project and select in which process areas to invest. The type of improvement that is required **[improvement task]** will be stated at the beginning of the game and it can include, for example, reduce defects, improve the productivity and/or reduce costs **[goal]**. After making investments, you can verify their effect on the process area capability level (status column) and on the defect, productivity, cost or time-to-market measures. Right investments will increase the budget of investment points and wrong investment will decrease it. Other possible functionalities are: you can stop your game, you can see the Software Engineering rules applied in the game and you can receive information from the team. During the whole game you can see your performance and some tips about the available functionalities. At the end you receive a final score.*

5.4 FASENG

The knowledge of Software Engineering simulation games and their common requirements, and the lack of support to simulation games development led us to the decision to create our own framework, FASENG (*Framework for Software Engineering Simulation Games*). FASENG is composed of three components [10] (simulation model, simulator, and simulation engine) that can be reused in new simulation games creation (see Figure 5.2). These components can also be found in other Software Engineering simulation games [Drappa and Ludewig, 2000; Dantas et al., 2004b], however, in our case, they can be reused in new developments. The separation of concerns among these components reflects one of our goals which is to design components that can also be reused in the development of other simulation games.

The creation of the simulation framework started with the identification of common requirements found in the Software Engineering simulation games, with the aim to evaluate the following aspects:

1. Characteristics that promote the application of the learning theories associated with Constructivism [Possa, 2011].

2. Design decisions adopted to deploy each identified characteristic.

We evaluated the Software Engineering simulation games found in the SLR described in Section 2.4. The identified characteristics were organized as a catalog of

[10] We use the term "components" to refer to the structural elements of FASENG.

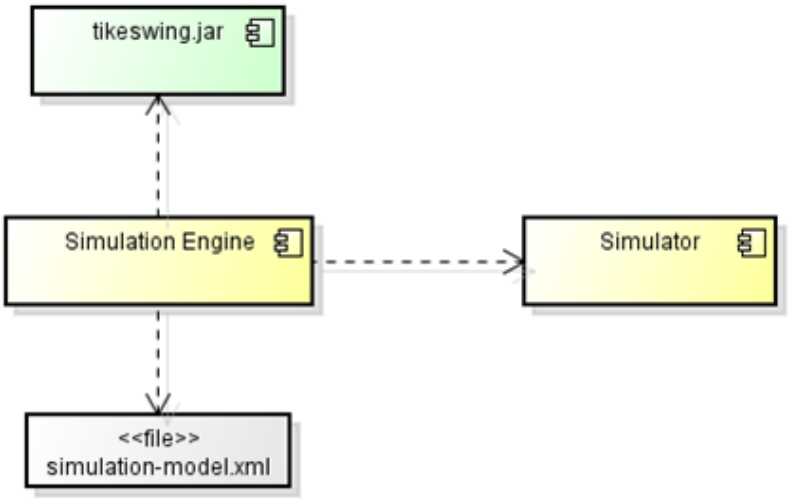

Figure 5.2. FASENG structural elements.

requirements, similar to the one used by Hoffmann et al. [2004]. They are important for the following roles:

- **Instructors** who adopt a simulation game as a complementary approach to the traditional educational methods. These requirements can assist them during the simulation game selection process.
- **Designers** who need to know which requirements are more effective for the learning process and the way they are implemented.
- **Players** who play the game to complement their knowledge in specific Software Engineering subjects.

The requirements are presented in the following and a detailed description of them can be found in Possa's dissertation [Possa, 2011] [11].

REQ1 - A simulation game should give support to the transfer of learning: It is important for players to understand the lessons of a simulation game and develop their own mental model [Alessi and Trollip, 2001]. In consequence, they will be able to transfer this knowledge to a real situation. Transfer of learning happens when the knowledge, skills, or information is transferred from one situation to another [Alessi and Trollip, 2001]. A good transfer of learning occurs when the performance in real situation is improved [Alessi and Trollip, 2001].

[11] Possa's dissertation describes his work related to simulation games requirements. After his master research, he also worked with the design and implementation of FASENG.

Two common characteristics were found in the evaluated simulation games that can enhance the transfer of learning. The first one is the players' active role during the whole game, e.g. as a project manager, they can hire or fire employees [von Wangenheim and Shull, 2009]. The second one is to allocate players to a hypothetical work or to a problematic situation, where they need to develop solutions. The most common scenario in the evaluated simulation games is students acting as project managers in a project with time, quality and budget constraints.

REQ2 - A simulation game should be interactive: One of the most beneficial aspects of simulation games is their ability to engage learners in meaningful activities through various forms of interactivity [Alessi and Trollip, 2001]. These include actions such as: making choices and decisions, manipulating objects, reacting to events, and collecting information [Alessi and Trollip, 2001]. Interactivity gives the player a sense of control over the game, improving motivation during the learning process.

The interactivity characteristic is presented in all the analyzed games with different levels of implementation. For example, SimSE implements all the actions listed above, in contrast, in qGame, players can only make choices.

REQ3 - A simulation game should reflect the complexity of the problems found in the real world: Besides simplifying the environment, it is important to support learners working in complex situations [Alessi and Trollip, 2001]. Fast and flexible simulations allow repeated experiences and different situations can be introduced and practiced [Alessi and Trollip, 2001]. The goal is that players act in specific roles, solve the problems, and observe the effects of their decisions. Simulation games can take different paths depending on the players' actions and reactions. Therefore, a simulation is a case study involving a specific reality where players play roles with responsibilities and restrictions.

All the simulation games implementation fully satisfied this requirement. For instance, the task allocation follows some of the complex aspects of the real world such as the allocated person should have the abilities required for the corresponding task.

REQ4 - A simulation game should facilitate the reflection about the learning subject: Some studies have shown that students who are encouraged to reflect on what they are doing, learn better both the declarative and procedural types of tasks [Upchurch and Sims-Knight, 1999]. Therefore, active reflection on experiences during activities carried out in a Software Engineering course (e.g. team project) promotes the acquisition of more meaningful and persistent learning [Upchurch and Sims-Knight, 1999].

In a simulation game, one example of an element with great potential for reflec-

tion support is the revisiting session, which is generally referred to as the debriefing session [Peters and Vissers, 2004]. In this session, it is possible to analyze the decisions taken during the game, draw conclusions and make the connection with the real-life situation. Unfortunately, none of the assessed simulation games provided any guidance for carrying out the debriefing sessions. Another element that can promote partial reflection is the explanatory tool or a feedback mechanism, allowing individual performance assessment.

REQ5 - A simulation game should support teamwork skills development: It is well-known that computer science students need to learn communication and collaboration skills [Lingard and Berry, 2002]. Although many programs today make team projects fundamental elements of their curricula, students receive almost no orientation about how to work effectively in a team [Lingard and Berry, 2002]. One way to meet this requirement is to build multiplayer simulation games. In these games, players can collaborate among themselves, allowing students to develop teamwork skills (e.g. MO-SEProcess). In contrast, single-player simulation games, where players can interact with other Non-Player Characters (NPC), i.e. a character controlled by the computer, can address partially the teamwork skill (e.g. qGame, SESAM, SimJavaSP, SimSE, The Incredible Manager, TREG, and SimVBSE). These games have some restrictions in representing real work situations, as communication breakdowns and conflicts between team members.

REQ6 - A simulation game should support different learning environments: The learning process may vary among students. Some students may learn better by watching and listening, others by reading, and others by doing and moving or using a hands-on environment [Zapalska and Brozik, 2006]. Adaptation of the learning environments can assist students with different learning styles, different levels of initial knowledge and different expectations and objectives [Moreno-Ger et al., 2008]. Adaptation can be static, for example, increasing the complexity in the course of the game, allowing students to choose different simulation models, or instructors to change the simulation model. Adaptation can be dynamic, for example, if players have some difficulties during the simulation game, a hint can be presented to them or the difficulty level of a task can be decreased gradually [Hunicke, 2005]. Among all simulation games that we found, no one presented dynamic adaptation features. Three of them support different simulation models: SESAM, SimSE and The Incredible Manager; and only qGame has different levels of difficulty.

In the first version of this framework, we developed components that support requirements REQ1, REQ2, and REQ3, and partially requirements REQ4, REQ5, and REQ6.

The first configurable component is the **simulation model** that can be used to represent:

- The structure of the software process, activities, interactions and artifacts;
- The system behavior and the factors that influence the environment to be simulated;
- The initial scenario of the game, i.e. the initial state of the project to be simulated.

The second component is the **simulator** which takes as input the model and interprets it, covering its equations iteratively. The simulator's final target is to calculate the behavior of each element.

The last component is the **simulation engine**, which the player interacts, receiving visual feedback from the simulation results and through which the player changes the model parameters. The next sections describe in detail each of these components.

5.4.1 Simulation Model

The simulation model represents aspects of the world to be simulated (REQ3). Its definition is represented as an XML file. The applicability of a simulation model depends on the model builder's ability to capture these aspects [Dantas et al., 2004a]. In essence, the Software Engineering processes are difficult to model, mainly because their intrinsic characteristics, mostly involving human behavior, such as, non-linear relation of cause and effect, feedback cycles, dynamic behavior and socio-cultural issues that can affect them.

5.4.1.1 Configuration file

In the XML file, instructors can tailor a simulation model according to their course (REQ6). This model allows the representation of an active role to players (REQ1), reaction to events (REQ2), feedback (REQ4) and different participants to actions (REQ 5). We used some modeling constructs similar to those defined by SimSE [Navarro, 2006]. These elements are shown below followed by a corresponding XML excerpt.

- **Object types**: The object types define the templates for all the objects. The object type consists of a name, a Boolean value indicating whether a log will be generated for each object and a set of attributes. For each attribute, its name, type (Integer, String, Double, Boolean, and List), and two Boolean values can

be defined. The Boolean values indicate the visibility of the object (i) during the game and (ii) at the end of the game. Unlike SimSE, there is no object type creation restriction.

<type name="ImprovementProject" log = "false">
<attribute name="InvestmentPoints" type="integer" visible = "true" visible-end="true"/>
</type>

- **Initial state**: The initial state contains a set of objects that are active at the beginning of the simulation. Each object is an instantiation of a given object type, containing initial values for their attributes.

<object name="IProject" type="Improvement_Project">
<attribute-value name ="InvestmentPoints">500</attribute-value>
</object>

- **Analysis**: The analysis represents types of rules that are created specifically to provide some performance feedback to the players. It contains the name of the analysis and a reference to their implementation.

<analysis-result>
<analysis-result name="Improvement_Analysis" class="br.ufmg.dcc.engsoft.spial.analysisrule.RuleImprovementAnalysis"/>
</analysis-result>

In the example above, we used Java reflection that allows an executing Java program to manipulate the respective Java class having its name.

- **Actions**: The actions represent a set of activities presented in the simulated process, for example, give training to the team or invest in some process area. Each action has a name, description, one or more rules, one or more participants, a trigger and one or more destroyers. Triggers, rules, and destroyers behavior is implemented in Java language. The advantage of this approach is a greater power of expression.

<action name="investment" description="Investing in Process Areas" >

Participant(s): The participant restricts the types of objects that can participate in the action, as well as their minimum and maximum amount.

<participant name ="ProcArea" type="Process Area" minimum-occurrence="1" maximum-occurrence="2"/>

Trigger: The trigger represents a condition to start the execution of an action. There are three distinct types [Navarro, 2006]: autonomous, player, and random. Autonomous triggers causes the action to begin automatically, without player interaction. Player triggers start actions when players carried out some operation, for example, selection of the invest operation or a menu item that appears on the right-click into the status panel. Random triggers specifies the likelihood of the action to be executed. Triggers can cause the end of the game and the calculation of a score; this is represented by the *game-ending* parameter.

<trigger name="investmentTrigger" type="player" user-operation= "Invest" description="Investment in Process Area" class="br.ufmg.dcc.engsoft.spial.trigger.TrigerPlayerInvestiment" game-ending="false"/>

Rule(s): Rules specify the effects of the execution of an action. They modify the objects attributes that participated in the action. Each rule identifies the time of its execution, which can be: trigger, destroyer, or continuous. Trigger rules will be executed at the time the action is triggered. Destroyer rules will be executed when the action is destroyed. Continuous rules, on the other hand, will be fired every clock tick, representing continuous behaviors of the model. Rules also specify the order they will be executed through the "priority" parameter.

<rule name="investmentRule" description="Investment made in some process area" timing="trigger" priority="3" class="br.ufmg.dcc.engsoft.spial.rule.RuleDiscreteInvest"/>

Destroyer(s): Opposite to the triggers, destroyers represent a condition to ending the execution of an action. It has four types: three like the triggers (autonomous, random, or player), and an additional one: timed. The timed destroyer specifies the duration of an action (time-to-live).

<destroyer name="investmentDestroyer" type="timed" time-to-live = "3" class="br.ufmg.dcc.engsoft.spial.destroyer.DestroyerTimedStandard" game-ending="false"/>

5.4.1.2 Mathematical Framework

Several techniques can be applied for the construction of a model. The SPIAL simulation model was inspired by the mathematical model proposed by Birkhölzer et al. [2004, 2005b,a]. The aim of their model is to provide an environment where users, acting as a top-level manager, can interactively change the investments in the level of a software development organization, improving their understanding of the SPI results. It was developed following a top-down approach, considering 15 process areas of CMMI as states, and deriving 27 business measures from the company strategic goals (in their study at Siemens) [Dickmann et al., 2007]. The inputs are the investments that the user can make in each process area and the outcomes are the real value of the business measures. The simulation model concentrates on the organizational level abstracting the specific aspects of software production process, which are combined in generic concepts.

The items, listed below, described the motivation to apply this mathematical framework:

- The original framework represents the overall direct and indirect effects of an SPI initiative. These effects can be very difficult to estimate and predict;
- The original framework integrates business measures that can be used as performance and informative feedback during the whole simulation game; and
- It seems that the original framework can be adapted to other SPI reference models, since the prerequisite to apply it is to have a simulation model with process areas characterized by capability levels.

We promoted some changes that turned the model more suitable to an educational environment. The reasons to adapt this model are:

- Besides the organizational-level, we simulate aspects of the project-level improvements, abstracting specific issues that have been extensively exploited by other Software Engineering simulation games (e.g. the project planning task).
- This model has ten configurable parameters that constitute a considerable burden in populating the model with adequate values for them [Birkhölzer et al., 2005a]. In its original application, these values were estimated by experts and turned out to be a tedious work. This could be a de-motivation factor for instructors and we tried to avoid this problem reducing the number of configurable parameters.

- We included Software Engineering rules and SPI motivators and de-motivators factors into the simulation model.
- We did not include advanced concepts in the initial version of SPIAL. It provides an easier way for the students to evaluate the causes and effects of their choices.

The SPIAL mathematical framework for such a model is a linear, time-discrete, state-space model (these categories are discussed by Pearson [1999]; Kitagawa and Gersch [1996]):

$$\begin{aligned} \vec{x}_{t+1} &= \vec{f}(\vec{x}_t, \vec{z}_t) \\ \vec{y}_{t+1} &= \vec{h}(\vec{x}_{t+1}) \\ t \in \mathbb{N}, \vec{x} \in \mathbb{R}^n, \vec{z} \in \mathbb{R}^m, \vec{y} &\in \mathbb{R}^p \end{aligned} \tag{5.1}$$

Where t denotes the discrete time between each player interaction, $\vec{x}_t = (x_{1,t}, \ldots, x_{n,t})$ denotes the n-tuple of the internal state variables, $\vec{z}_t = (z_{1,t}, \ldots, z_{m,t})$ denotes the m-tuple of input variables, $\vec{y}_t = (y_{1,t}, \ldots, y_{p,t})$ the p-tuple of the output variables, f and h denote functions that relate subsequent states.

In this model, the internal state variables represent the capability levels of the key process areas of CMMI (e.g. Requirements management and Technical solution) and the output variables correspond to measures that should be used by players to evaluate process improvement results. These SPI results are discussed in Section 4.4.1 and two examples are productivity and cost.

The dynamic behavior of the model is described by the following equations:

$$x_{i,t+d_i} = level(y_{i,t} \cdot (1 + p_{i,t})), i = 1, \ldots, n \tag{5.2}$$

$$p_{i,t} = \sum_j \beta_{ij} \cdot x_{j,t} + \sum_j \gamma_{ij} \cdot x_{j,t} \tag{5.3}$$

with

$$level(arg) = \begin{cases} x_{i,t} + 1 & \text{if } arg \geq c_i \cdot d_i \\ x_{i,t} & \text{if } arg < c_i \cdot d_i \\ x_{i,t} - 1 & \text{if a process area stays without investments for 100 clock ticks} \end{cases}$$

$$d_i = \alpha_i \cdot g_i$$

$$\alpha_i = \frac{\sum \mu_i \cdot \rho_i}{\sum \mu_i}$$

In equations 5.2 and 5.3 the term $p_{i,t}$ consists of the feedback from the other states. In SPIAL, 11 process areas are mapped with their dependencies (prerequisites processes areas) and impacts according to CMMI. For example, the process area *Project Monitoring and Control* (PMC) depends on process area *Project Planning* (PP), therefore, if PP is at capability level 0, investments in process area PMC will be quite ineffective. In the same way, improvements in process area Measurement and Analysis (MA) impacts positively on process area PMC, i.e., with higher capability levels of MA will ease the improvement of PMC. The effects of improvements can only be observed after a delay d_i period. We modeled the delay as a term affected by the motivators and de-motivators factors for SPI [Niazi et al., 2006; Baddoo and Hall, 2003], α_i, and by a predefined delay for each process area g_i. In α_i equation, the μ_i term represents the impact of a factor and ρ_i term is the probability of its occurrence. For values of α_i lower than zero, the state can react fast on changes unlike values greater than 0.

The level(.) function represents the capability level of each process area (from 0 to 3). Investments greater than the basic cost of improvement needed during the improvement time $t + d_i$, will increase the process area capability level. We also modeled a decrease of the capability level, after a predetermined simulation time without investments. In equation 5.3, β_{ij} and γ_{ij} serve as weights for the influence (dependence and impact) of the j-th process area on the i-th process area.

The actual value of each measure (the output) is calculated as a linear mathematical function with parameter the weighted sum of the internal states.

$$y_{i,t} = mat(\sum_j \delta_j \cdot x_{j,t}) \tag{5.4}$$

where δ_j denotes the weight of each process area on the measure value, which was derived from the Software Engineering rules. One mat(.) function was defined for each measure.

The player operates with a budget A_t of investments points, which is characterized by the equation below:

$$A_{t+1} = A_t + \sum_{i=1}^{p} \theta_i \cdot x_{i,t+1} - \sum_{i=1}^{m} z_{i,t} \tag{5.5}$$

Investment points are the amount necessary to maintain (or improve) a capability level of a process area. The budget of investment points consists of the previous balance, A_t, the previous investment, $z_{i,t}$, and the weighted sum of the internal states, $x_{i,t+1}$. Therefore, making right investments will increase the budget of investment points and wrong investment will decrease this budget.

5.4.1.3 Rules

Initially, we collected 123 Software Engineering rules from text books and other Software Engineering simulation games. These rules are used to teach the best practices of Software Engineering to students, rewarding or penalizing their actions. This set includes rules that are imprecise (do not specify values) and rules beyond the Software Engineering area, including a wider range of business processes. From this set, we selected the ones related to the SPIAL process areas, resulting in 57 Software Engineering rules (see Appendix B). Three examples of rules are:

- *Requirements deficiencies are the prime source of project failures.* [Glass, 1998]
- *Errors are most frequent during the requirements and design activities and are the more expensive the later they are removed.* [Boehm et al., 1975; Endres and Rombach, 2003]
- *A combination of different V & V methods outperforms any single method alone.* [Hetzel, 1976]

For each specific rule, we mapped the process area and measures related to it. The result was used to calibrate the δ_j parameter of the $y_{i,t}$ equation. Since most of these rules are quite imprecise, they do not allow the direct application into the simulation model. Therefore, we incorporated rules by experimenting them with different values and we selected the values that are most suitable for representing in an educational environment.

5.4.2 Simulation Engine

The simulation engine is a component with which the player interacts, receiving visual feedback from the simulation results and through which the player changes the model parameters. This component uses the MVC (Model-View-Control) design pattern, which allows the separation of domain logic from the user interface. TikeSwing[12]

[12]TikeSwing is a framework for Java Swing development providing high-level MVC architecture.

was used in the implementation of this design pattern. During the tailoring phase, instructors can change the interface according to their needs, without worrying about other aspects of the application (model and control). Since the Software Engineering simulation games can have different scopes, we did not restrict the interface design to fixed graphical elements.

When SPIAL begins execution, the player is presented with a description of the simulation game. This description includes the required knowledge to play the game, the game task, the player's role, the description of the software development company, the goals, how much time and money the player has, the final score calculation and some guidance to play the game (see Figure 5.3).

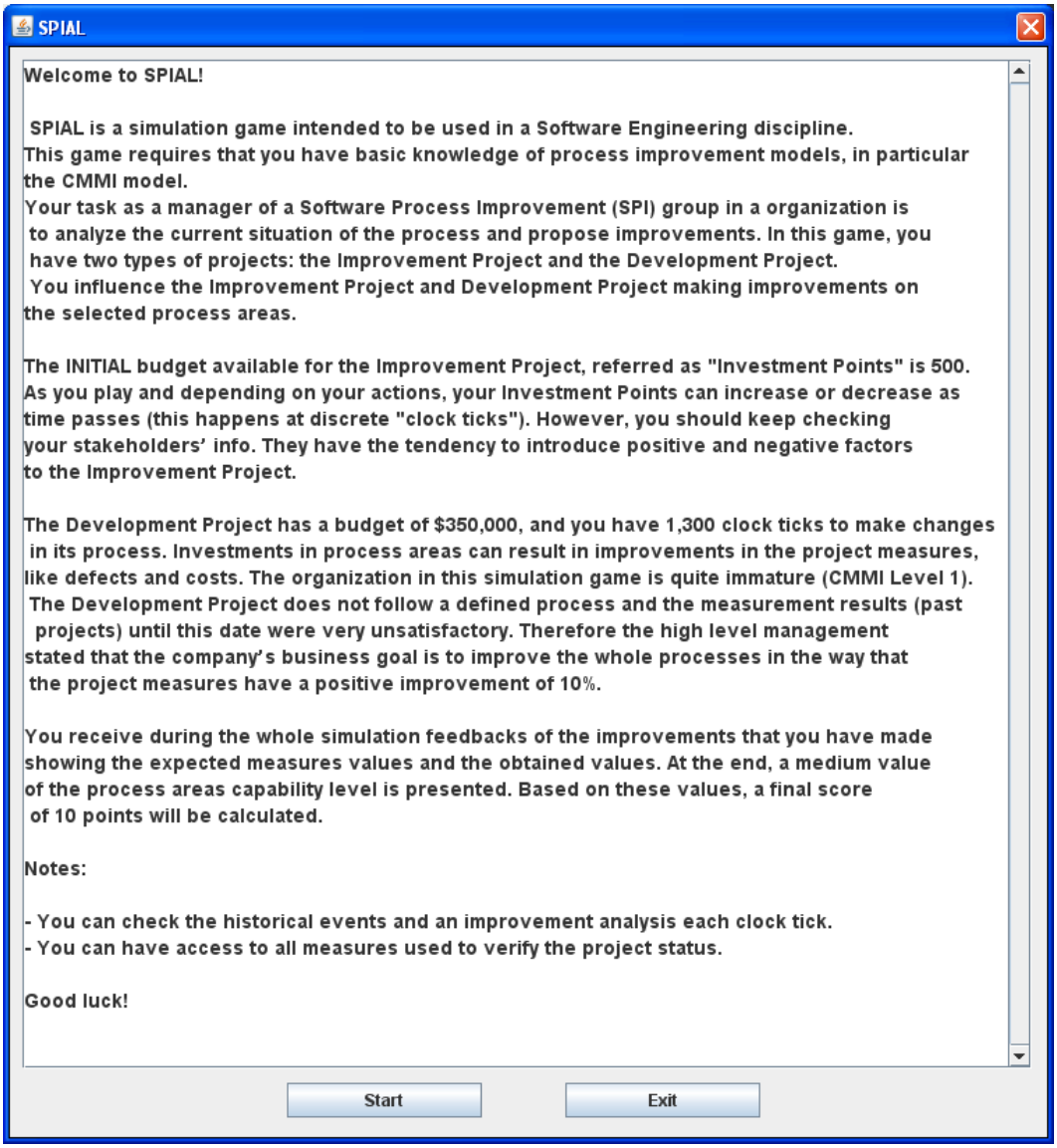

Figure 5.3. SPIAL introductory screen.

The game tab sheet (see Figure 5.4) displays the software process improvement project, the stakeholders of the software development company and some project information. The SPI project consists of 11 CMMI process areas in which players can distribute their investments points. The amount invested in each process area is registered in the *TotalAllocated* column of the *Process Areas* table. The *Status* column provides some feedback about the progress of the capability level of each process area. The stakeholders "communicate" with the player through speech bubbles over their heads (see Figure 5.5), providing valuable information that players can use during the game. The information consists of random events (de-motivator and motivator factors) that can occur during the improvement project and affects its deployment, some guidance and feedback to the player during the game. For example, the consultant could notify the player about wrong choice of investments: "Since there is dependency among process areas, some of your investments may not be effective". The project information area is a tabular view of the development project: budget, money spent, allotted time, time used, artifacts, measures and the status of each development phase. The time control mechanism, located in the lower right corner of the interface, is based on clock ticks. This mechanism allows the player to drive the simulation. The player can access the *Info* and *Restart* buttons in the lower right corner. The *Info* button shows the starting description again. The *Restart* button restarts the game.

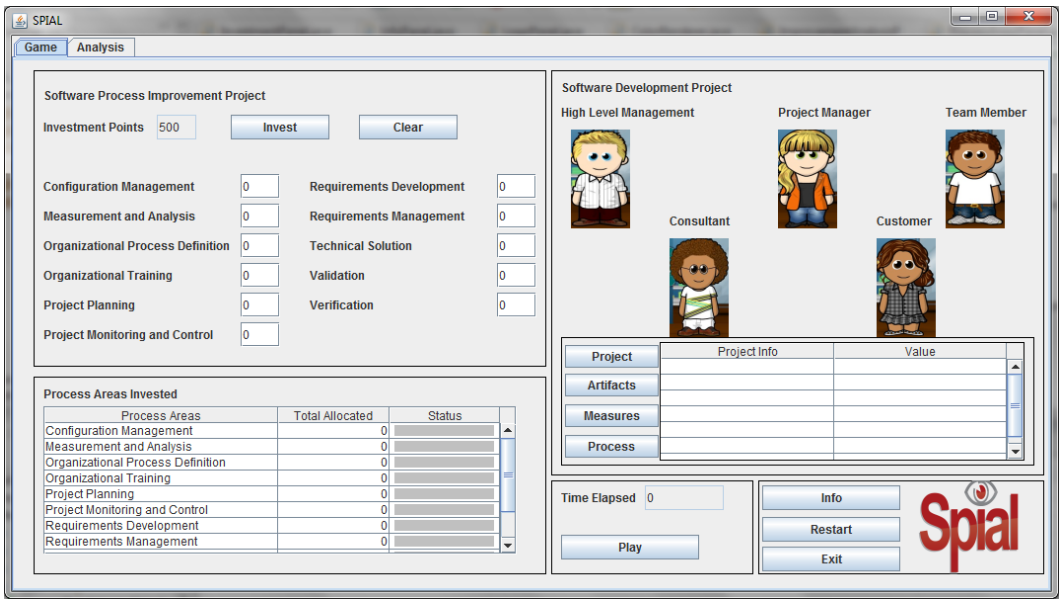

Figure 5.4. SPIAL Graphical User Interface.

The analysis tab sheet (see figure 5.6) was designed with the aim to give players a better insight of the correctness of their decisions, specifically providing graphical

Figure 5.5. Stakeholders communication.

information about the measures, the description of the rules used in the game, information about the achievement or not their improvement targets and the final score. All events occurring during the game are recorded in the *Events log* table.

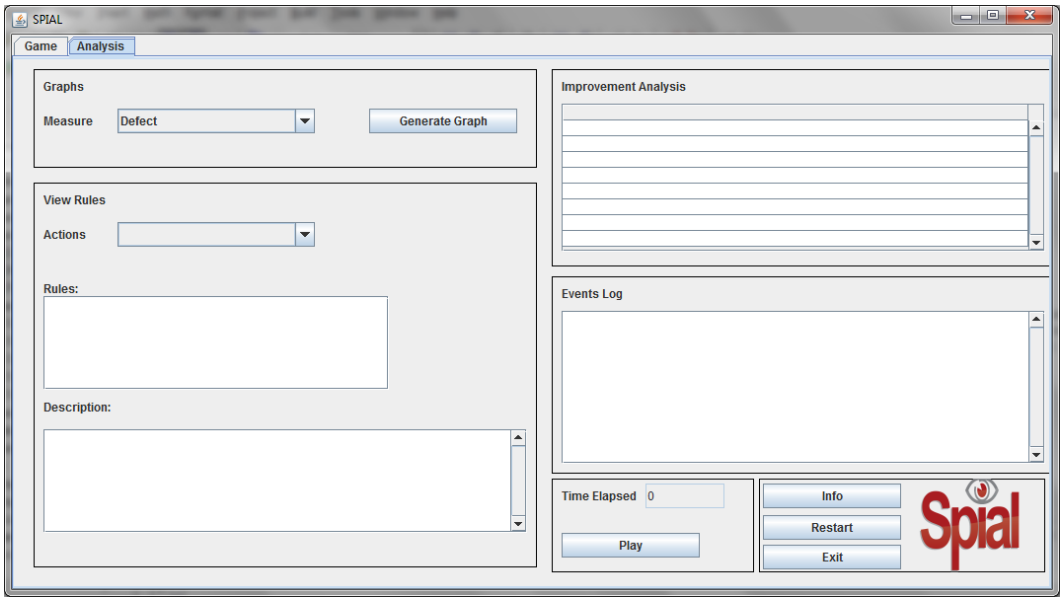

Figure 5.6. SPIAL Analysis Tab sheet.

The scenario described below is an example of how a student may use SPIAL in completing an SPI task. This example illustrates the Software Engineering rules presented in a game session and how the players' decisions can affect their game results. This scenario is based on a Waterfall development project. The initial narrative is described in Figure 5.3.

When the game begins, the player takes it through the interface elements and observes, for example, the gray Status bar, which means *no available information about*

the process area capability level. The player can verify the initial project information, the percent complete of each artifact (0%), the measures (Defect, Cost, Productivity and Time-to-market), and which development phase is active. In the analysis tab sheet, when the player tries to generate a graphical measurement representation, a message appears warning that the organization is immature and, at that time, it is not possible to have access to it. Only after improvements, the graphical information will be available. At this moment, the player can infer that immature organizations have problems to provide measures to their stakeholders.

The player decides to invest in all process areas an amount of 20 investment points. When the *Play* button is pressed, the consultant gives a message that there are dependency among process areas and some of the investments may not be effective. After 10 clock ticks, the player observes that the *Graphs* information are still not available and the measurements targets have not been achieved. The project information shows an expressive increase in the number of defects, cost, and time-to-market and a decrease in productivity. The development phase is *Requirements* and the team is working on the requirement artifact. The player decides to check the rules used by the game, and he/she observes that investments made during the upstream portion of development process will impact more on the project measures. The player decides to invest more in process areas of maturity level 2 CMMI, instead of making investments in every process areas.

After some clock ticks, the status bars change their colors (see Figure 5.7). There are some process areas where the improvement started (green) and other where nothing happened (red). It is also possible to have access to the graphical measurements representation, however the measures did not reach their targets. When the High Level Management communicates that the team is resistant to the change and the consultant gives a recommendation to make corrective actions (to give training or to give feedback of the improvement results), the player decides to invest 20 investment points in giving feedback (menu item that appears on the right-click into the Software Development Project panel), this avoided the occurrence of the negative factors during a period of time and reduced the delay of the improvement deployment. The player decides to invest more in the CMMI level 2 process areas and on the Requirements Development process area, since the current development phase is Requirements. The consultant gives a message that the strategy is suitable, the investment points start to increase, and all measures achieved the target. Then, the player decides to wait until the beginning of the next development phase (Design). Since the player stayed a long time without making investments, the process area capability level decreases, the investment points stop increasing and start decreasing. Thus, the project targets

were not reached again. After 239 clock ticks, the game scenario is similar to the initial scenario, the only difference is that the player has a different amount of investment points (234) to invest. The player decides to check with the project managers some guidance about the improvement (menu item that appears on the right-click into the Software Development Project panel). After investing 150 points in a meeting, the process activities that can be improved were shown to the player. Then, the player can infer in which process areas to invest and their dependencies. In each development phase, investments in one area will be more effective than in other. After changing the negative scenario to a positive one, the player learns to not let process areas without investments for a long period of time. Then, the player decides to invest more in process areas of level 2, than in process area of level 3, except for the current development phase.

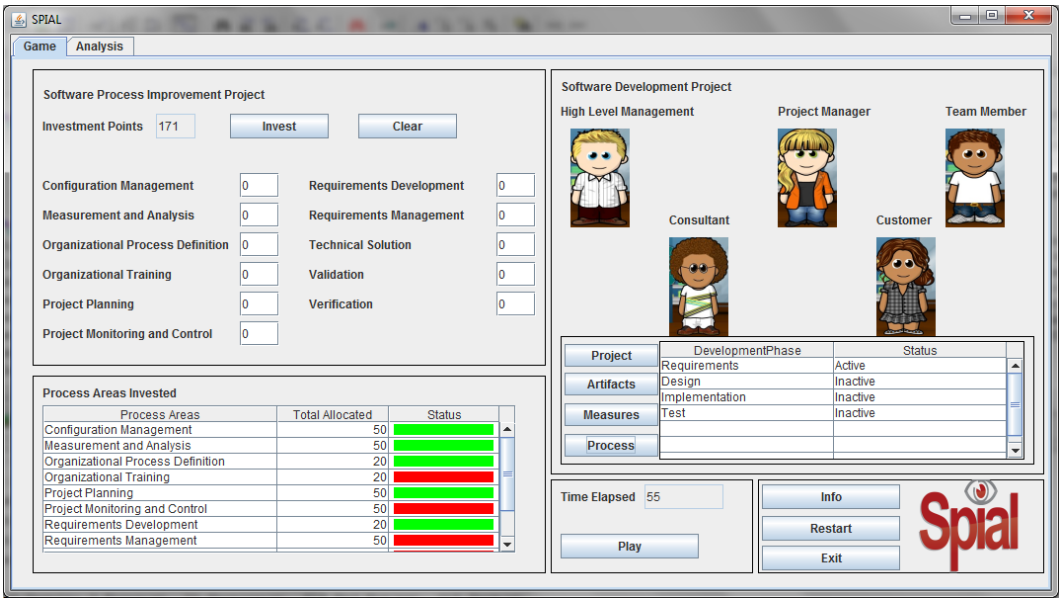

Figure 5.7. Status information.

Whether the player maintains the strategy of constant investments, respecting the dependency and trying to avoid the negative factors (that increases the investment cost and delay the improvement deployment), probably he/she will get a good final score (close to 10).

5.4.3 Simulator

A great number of simulators are either continuous or discrete. Since in software development project simulation both mechanisms are needed, we decided for a hybrid approach [Donzelli and Iazeolla, 2001] in which:

- At a higher level of abstraction, the structure of the software process, activities, interactions and artifacts are best modeled by a discrete event approach;
- At a lower level of abstraction, the behavior of the activities and the factors that influence the simulated project are best modeled by a continuous approach.

In the simulator component, aspects of discrete events are used to represent individual objects (e.g. process areas, and artifacts), actions ("give training or mentoring", "make an investment"), conditions of triggers and destroyers, and the rules executed at the start or at the end of an action. The continuous characteristics are represented through continuous rules, which are performed at each step of the simulation, whether the actions that define them are currently active.

The simulator runs in a loop. For each time step, it looks whether the conditions of triggers and destroyers were satisfied, executes continuous rules, generates log, and gives feedback to the players. If triggers or destroyers conditions are satisfied, the rules associated to the actions are checked. Continuous rules are fired every clock tick, whether their respective action is active. The log is generated for the object types defined in the configuration file. The calculations of performance feedback are defined by the instructor. In our prototype, at each time step, the simulator verifies whether the player reached the target values for the measures and presented to it an evaluation. The detailed communication between the simulation engine and the simulator is shown in Figure 5.8, using a simplified UML diagram.

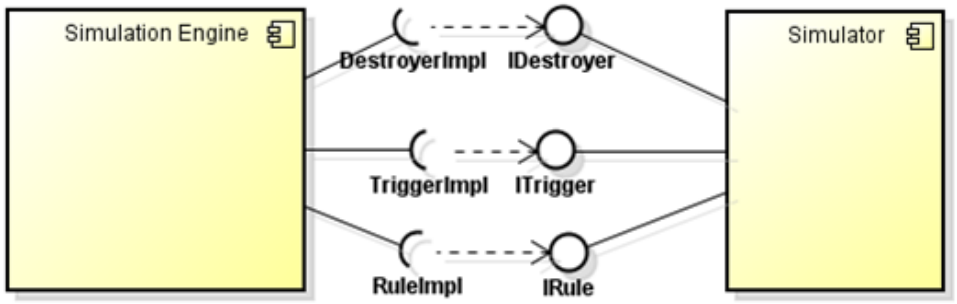

Figure 5.8. Communication details between simulation engine and simulator.

As we discussed above, the basic elements for the construction of SPIAL simulation models are basically the same used for the construction SimSE simulation models. The main difference between SPIAL and SimSE is that while the models are compiled

in SimSE, generating a new game for each model, the SPIAL models are interpreted by the simulator. The advantage of interpreting the model instead of compiling it is that the tests of the model becomes simpler, since the changes in the model is quickly viewed when running the simulator again. A disadvantage is that the execution of the simulation is slower since it is necessary to "translate" during run-time the modeling elements (e.g. actions, conditions, and rules).

The simulator has several component interfaces, representing the start (triggers) and end (destroyers) conditions for each action, as well as rules to be executed. Since there are several types of triggers, rules, and destroyers, a hierarchy of interfaces was created in the simulator component. For each condition or rule to be created in the simulator engine, an interface of the simulator should be implemented in a Java class. Then the rule class must be referenced in the XML file of the simulation model. When the simulator encounters this class name in the simulation model, it learns which class should be instantiated. The simulator was designed to handle the simulation model, defined externally in an XML file, and to generate the internal representation of it.

Figures 5.9, 5.10, 5.11 show the class hierarchy and interfaces use to implement destroyer, triggers and rules, respectively. In each step of the simulator, the destroyer, triggers and associated rules are performed. The abstract classes provide some interface methods implementation.

5.5 Evaluation

In this section we present the SPIAL evaluations carried out to answer the research questions RQ 1 and RQ3:

- RQ 1: Can students learn Software Engineering concepts or reinforce them using SPIAL?
- RQ 2: What characteristics are required for SPIAL to be capable of supporting the learning process?
- RQ 3: How can such a simulator be incorporated into Software Engineering courses?

The second research question was addressed by the analysis of other simulations games, the semiotic analysis, and the SLR that helped in the identification of SPI central issues (e.g. measures, results, motivator and de-motivators factors). The first question can be further broken down into specific questions:

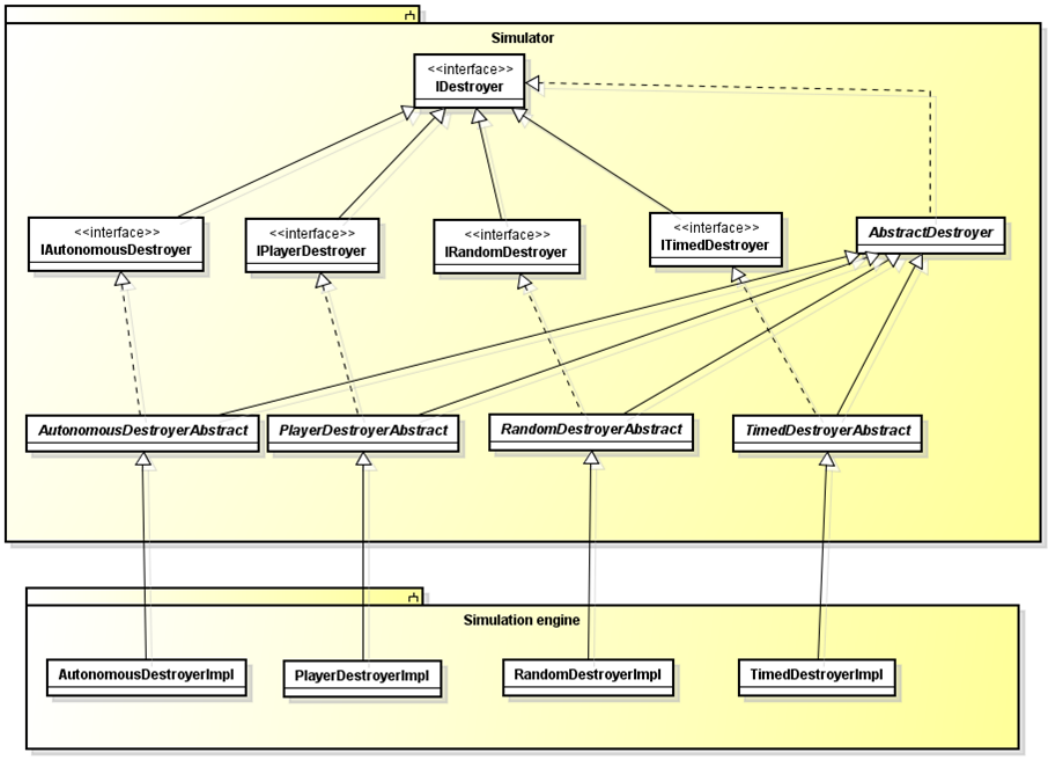

Figure 5.9. Destroyer hierarchy and interfaces.

1. What is the perception of the students about SPIAL? (e.g., it is enjoyable) What are its strengths and weaknesses? How well does SPIAL design is appropriate for their purpose?

   These characteristics are indirectly related to the learning, showing how effective is the educational environment. These questions provide a subjective evaluation of the simulation game, helping to identify flaws in the game design.

2. Is there any difficulty to play SPIAL?

   This question highlights how much guidance will be necessary before starting to play SPIAL and some suggestions for further improvements.

3. Can students actually learn software process concepts from using SPIAL, considering effects on the remembering, understanding and applying level?

   It is critical to determine if the game achieves the central goal of our research. We expect a positive learning effect on the capability of students to remember and understand software process concepts, specifically SPI concepts, and on the capability to apply the acquired knowledge.

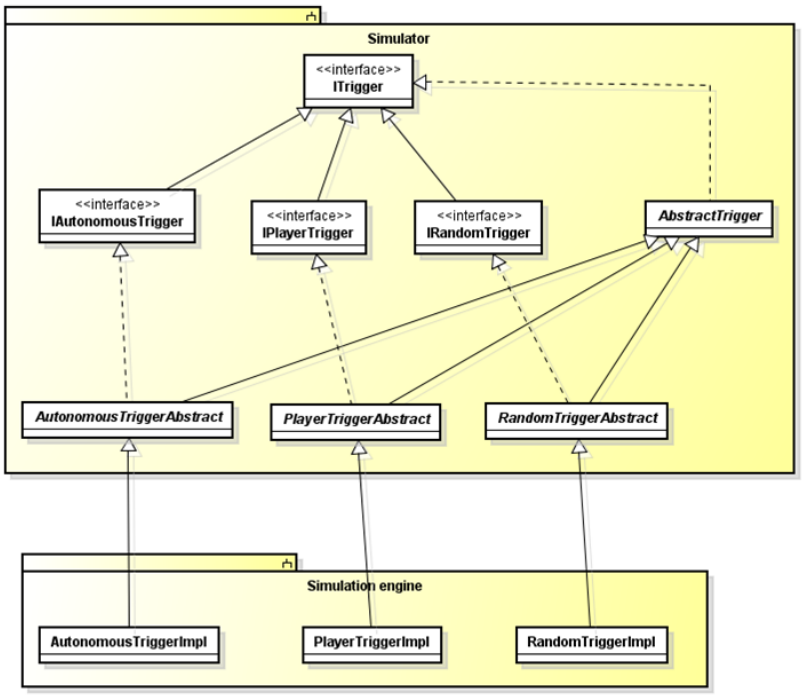

Figure 5.10. Trigger hierarchy and interfaces.

We evaluated SPIAL from two viewpoints: specialist and player. From the specialist's viewpoint, the Semiotic Inspection Method was conducted and communicability breakdowns were identified and adjusted. From the player's viewpoint, we evaluated the game carrying out a pilot experiment with undergraduate students. The core goal of all these evaluations is to verify the effectiveness of using this game as an educational tool. SPIAL is evaluated as an additional instrument to the Software Engineering class rather than a substitute to any teaching method.

Most of the simulation games were evaluated at some extent regarding their impact on Software Engineering education. One of the most extensive evaluations has been carried out by SimSE [Navarro, 2006]. The SimSE evaluation included a pilot experiment, an in-class study, a formal experiment comparing SimSE with other teaching methods (readings and lecture) and an observational study. These evaluations gave a comprehensive picture of the effectiveness of SimSE as an educational tool.

SESAM has been evaluated through a case study and an experiment [Drappa and Ludewig, 2000]. The goal was to investigate the learning effectiveness of using such a simulation model for educating Computer Science students in Software Engineering knowledge and Project Management abilities. Both in the case study and in the experiment, pre-test and post-test design were used to compare the scores.

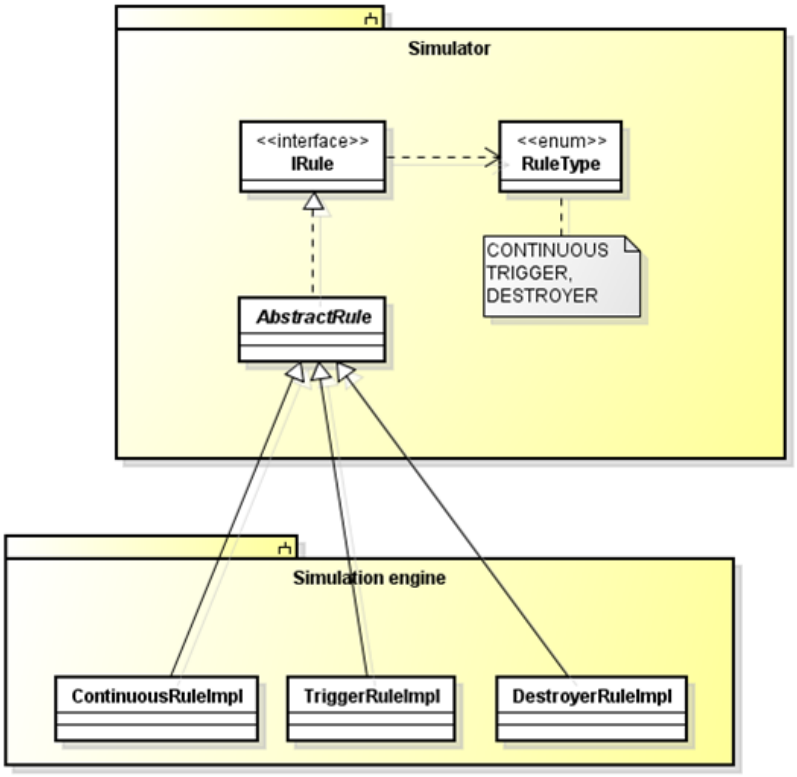

Figure 5.11. Rules hierarchy and interfaces.

Another study consists of a set of experiments carried out by Pfahl [2001] and Pfahl et al. [2004]. The goal was to evaluate the learning effectiveness of using a process simulation model for educating Computer Science students in Software Project Management. The experiment included pre-test and post-test which compared the application of a System Dynamics simulation model by the experimental group and COCOMO model [Boehm et al., 2000] by the control group. The results of each empirical study indicate that students using the simulation model gain a better understanding about typical behavior patterns of software development projects.

A study carried out by Gresse von Wangenheim et al. [2009] tried to demonstrate the learning effect of a software measurement game prototype with graduate students using pre-test and post-test experimental design. At the end of this study, the authors could not statistically demonstrate the effectiveness of this game. However, subjective evaluation indicate the potential of this game to support education. Results as those can also be observed in other related research. For example, in the context of SESAM and also with SimSE an effective learning effect could not be observed. One cause for this problem is the difficulty in making comprehensive evaluations in the educational domain [Gresse von Wangenheim et al., 2009]. Other reasons include multiple

interacting factors that make almost impossible to isolate the effects of an educational technique, and the difficulty to track the real learning effect and to get statistically significant results [Navarro and van der Hoek, 2007; Gresse von Wangenheim et al., 2009; Almstrum et al., 1996].

5.5.1 Semiotic Analysis

An inspector [13] conducted the first evaluation of SPIAL, carrying out the technical application of SIM. Like other inspection methods, SIM has a preparation step preceding its core analytical steps [de Souza et al., 2010]. The inspector reads the support material (introductory screen) and navigates through the game (see Figure 5.12). In this case, the inspector scenario is a student playing SPIAL.

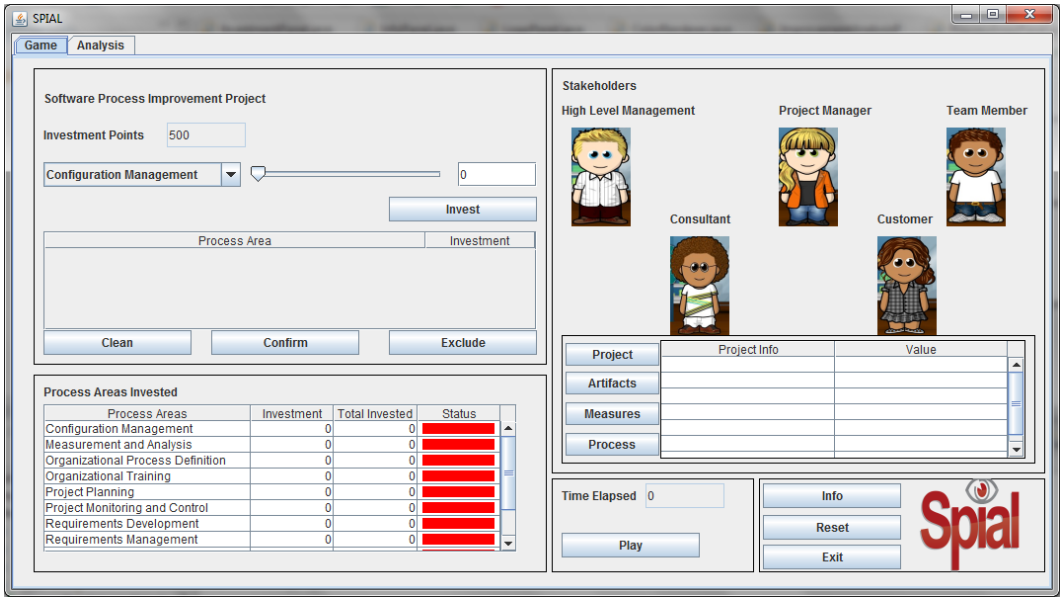

Figure 5.12. First SPIAL Prototype.

After preparation, come the core steps of the method. The first step consists of the analysis of **metaliguistic signs**. This is achieved when the inspector finishes the examination of all these types of signs encountered while running various interactive possibilities [de Souza et al., 2010]. The main evidence obtained in this phase is the introductory screen with the game description. The inspector pointed out that there is no online help function. If players click on "Info" (button) they will be told about the same game description as in the introductory screen. Very few tool tips explain

[13] Soraia de Souza Reis, a master student of DCC/UFMG who is a specialist in the Semiotic Inspection Method, conducted this evaluation.

the meaning of the buttons that the player can click on. Since there are no interactive mistakes at this level, the analysis is quickly finished, and produces the following metacommunication message:

[Here is my understanding of who you are,] *You are a student of a Software Engineering discipline. You are interested in learning more about SPI. You have basic knowledge of CMMI* [what I've learned you want or need to do, in which preferred ways, and why] *You need an intuitive game that you can monitor at the same time an improvement and a development project. You want to improve process and control the changes observing their effects on the project measures. You need some feedback during the game to verify the improvement effects.* [This is the system that I have therefore designed for you, and this is the way you can or should use it in order to fulfill a range of purposes that fall within this vision] *You play a simulation game in the role of a manager of an SPI group. You can interact with some stakeholders who can introduce positive and negative factors to the improvement. You have 500 investment points to invest in the improvement project and, depending on your actions, this improvement points can increase or decrease. The organization is CMMI level 1. You have 1300 clock ticks to make the needed changes. The development project has a budget of $350.000. During the whole game you receive feedbacks about the improvements and measures. At the end, you can see your score in a 10 points scale. You can also have access to a log of events and the rules used in this game.*

The inspector suggested the following improvements to the game description at the introductory screen:

1. A better explanation of how the project measures are affected.
2. A description explaining the investments points.
3. The list of stakeholders.
4. The characterization of the player as an employee of the organization.
5. A better description of the improvements' target.

We make the suggested changes with exception of number three, which can be verified at the main game interface.

The analysis of **static signs** was achieved when the inspector finishes the examination of all static signs encountered while playing the game. The inputs of this step are all interface elements appearing on the screens. At the end of core step 2, the metacommunication schema was filled out in the following way: text within brackets from the previous schema has been omitted; message portions that are not communicated

with static signs are crossed out; and bold text represents content that was introduced with static signs.

~~*You are a student of a Software Engineering discipline*~~. *You are interested in learning more about SPI. You have basic knowledge of CMMI. You need an intuitive game that you can monitor at the same time an improvement and a development project. You want to improve process and control the changes observing their effects on the project measures. You need some feedback during the game to verify the improvement effects.* ~~*You play a simulation game in the role of a manager of an SPI group.*~~ *You can interact with* ~~*some*~~ ***five*** *stakeholders who can introduce positive and negative factors to the improvement. You have 500 investment points to invest* ***in 11 CMMI process areas*** ~~*the improvement project, and depending on your actions this improvement points can increase or decrease*~~. ***You should select the process area to be improved from a ComboBox list and define the amount of investment points for each area. The table below the investment area informs you about the planned investments that you have made. In order to confirm the investment, you should click on "Confirm" button. You also have the options: exclude or clean the planned investments. The table "Process Areas Invested" allows you to monitor the amount invested. In the "Investment" column you can see your last investment and in the "Status" column you can observe the improvement of each specific process area.*** ~~*The organization is CMMI level 1.*~~ ***You can control the development project accessing the project, artifacts, measures and process information.*** *You have 1300 clock ticks to make the needed changes. The development project has a budget of $350.000.* ***You start the game clicking on the "Play" button. You can reset the game or exit from it.*** *During the whole game you receive feedbacks about the improvements and measures.* ***The analysis tab sheet allows you to monitor the improvement. You can generate graphs and you can analyze the impact of your actions through the Improvement Analysis table.*** *At the end you can see your score in a 10 points scale. You can also have access to a log of events and the rules used in this game.*

The inspector suggested the following improvements to the game interface:

1. The table "Process Areas Invested" can be simplified. The meaning of the "Investment" column is not clear. In addition, the "Status" column can be represented using more advanced design elements or in a textual form.

2. The action of monitoring artifacts during the game was not clear. Therefore, this information can be removed.

We changed the table "Process Areas Invested" (see Figure 5.4). Regarding the artifact information, we believe that this information is important to monitor the project evolution.

The analysis of **dynamic signs** is achieved when the inspector finishes the examination of all dynamic signs encountered while playing the game. The input for this step is all interactions supported by the interface elements. Since there is very little guidance, the inspector explores the interface to find out what the game does, and the educational benefits that it can bring to students. The inspector observed that the whole game features are only conveyed through dynamic signs. The metacommunication schema filled out at the end of this step is the following (bold text represents added content that is only communicated with dynamic signs).

~~*You are a student of a Software Engineering discipline*~~. *You are interested in learning more about SPI. You have basic knowledge of CMMI. You need an intuitive game that you can monitor at the same time an improvement and a development project. You want to improve process and control the changes observing their effects on the project measures. You need some feedback during the game to verify the improvement effects.* ~~*You play a simulation game in the role of a manager of an SPI group.*~~ *You can interact with* ~~*some*~~ *five stakeholders who can introduce positive and negative factors to the improvement* ***through bubbles over their heads****. You have 500 investment points to invest in 11 CMMI process areas* ~~*the improvement project*~~*, and depending on your actions this improvement points can increase or decrease. You should select the process area to be improved from a ComboBox list and define the amount of investment points for each area. The table below the investment area informs you about the planned investments that you have made. In order to confirm the investment, you should click on "Confirm" button. You also have the options: exclude or clean the planned investments. The table "Process Areas Invested" allows you to monitor the amount invested. In the "Investment" column you can see your last investment and in the "Status" column* ~~*you can observe the improvement of each specific process area*~~ ***you can observe if the capability level of the process area is low (red bar), increasing (green bar) or decreasing (orange bar)****.* ~~*The organization is CMMI level 1.*~~ *You can control the development project accessing the project, artifacts, measures and process information. You have 1300 clock ticks to make the needed changes. The development project has a budget of $350.000. You start the game clicking on the "Play" button. You can reset the game or exit from it. During the whole game you receive feedbacks about the improvements and measures. The analysis tab sheet allows you to monitor the improvement. You can generate graphs,* ***showing the current and estimated value****, and you can analyze the impact of your actions through the Improvement Anal-*

ysis table. ***You can verify at each moment whether you achieve the measures target****. At the end you can see your score in a 10 points scale. You can also have access to a log of events and the rules used in this game.*

In this step, the inspector pointed out the following issues about the game:

1. There is no guidance on how to invest the points. The player can invest 1, 10, 50 or 100 points in each process area.
2. It is tiresome to choose each process area and the investments points during the whole game.
3. The column "Investment" in the "Process Areas Invested" table is not clear.
4. It is difficult to understand the underlying behavior of the game. Some investment did not produce any change.
5. The total time needed in the simulation is not visible to the player. You need to access the project information.
6. Only after a period of time it was possible to observe the measure information in the analysis tab. The graph information could be placed in the game tab.
7. The rules information could be better expressed in the game (not only in the analysis tab).

Related to items 1, 4 and 7, the "Consultant" will present a message related to the game behavior, specifically, about the dependency among process areas, the amount of investment points, some Software Engineering and game rules. Regarding the interface design, we changed the interface to ease the investment procedure. We also removed column "Investment" in the "Process Areas Invested" table. However, we did not change the time information on purpose, because players should navigate through the project information (see Figure 5.4).

In the final steps conducted by inspector, an unified analysis was produced, highlighting the main communicability breakdowns. The analysis shows that getting the whole message from this game is the result of trial and error interaction. A number of feedback information were presented during the whole game, such as measurement charts and improvement analyses, and important aspects to understand the core behavior were missing, such as the reason why sometimes investments do not produce any improvement. In addition, some interface improvements can be done to allow a better interaction, moving elements from the analysis tab to the main tab. We produce a new game version, with the modifications discussed above, which provides more

guidance to players. A more elaborated interface design was let as a future work, we only make small changes in order to release, as soon as possible, a new game version to the experimental activities.

5.5.2 Pilot Experiment

In this first experiment, our aim was to gain a better understanding of the SPIAL effectiveness as an educational tool. Specifically, we observed students' understanding, remembering and application of Software Engineering concepts in the context of CMMI based SPI initiatives using SPIAL. In addition, we verified the adequacy of SPIAL in terms of its design, content, duration and student's engagement.

5.5.2.1 Context

The experiment was conducted with students enrolled in their final year of undergraduate studies. This experiment happened in the context of a Software Engineering course. The Software Engineering course at the Department of Computer Science at Federal University of Minas Gerais, Brazil, is taught in one semester (60 hour class). This course is designed to cover the fundamentals of Software Engineering theory and practice at an introductory level. The students follow Praxis [Paula Filho, 2009], a model-driven process that prescribes the use of UML for the models and Java as the programming language, using test-driven development techniques.

While the discipline was running, students were asked if they would be interested in participating in an experiment related to SPI issues, involving a simulation game. Students are expected to have a basic understanding of Software Engineering concepts, mainly the development process and the CMMI framework.

5.5.2.2 Research Questions

Our research questions are:

1. What is the perception of the students about SPIAL? (e.g., it is enjoyable) What are its strengths and weaknesses? How well does SPIAL design is appropriate for their purpose?

2. Is there any difficulty to play SPIAL?

3. Can students actually learn software process concepts from using SPIAL, considering effects on the remembering, understanding and applying level?

4. How can such a simulator be incorporated into Software Engineering courses?

The first and the second research questions evaluate practical aspects of SPIAL, such as time spent, and subjective students' perceptions, such as motivation and engagement. This can inform about the adoption of SPIAL into a Software Engineering course (fourth research question). We used a questionnaire in order to collect the reaction of the players about these aspects.

As in the Gresse von Wangenheim et al. [2009] work, our third research question is derived from knowledge levels of the revised version of Bloom's taxonomy of educational objectives (remembering, understanding, applying, analyzing, evaluating and creating) [Anderson and Krathwohl, 2001]. However, we have chosen only three levels from Bloom's taxonomy: remembering, understanding, and applying (see Table 5.1). These three levels represent what knowledge may be reasonably learned during an undergraduate education [SE2004, 2004].

Table 5.1. Cognitive levels of the revised version of the Bloom's Taxonomy (adapted from Gresse von Wangenheim et al. [2009]).

Levels	Questions	Learning Outcome
Remembering	Can students RECALL information?	Recall Software Engineering concepts presented in the CMMI framework: process areas, capacity levels, etc. Name the steps of the improvement process and business measures.
Understanding	Can students EXPLAIN ideas?	Identify important SPI aspects based on a problem description. Determine the effects of the improvement approaches on the business measures.
Applying	Can students USE a procedure?	Identify problematic areas in a software development company. Construct improvement initiatives based on the identified problems. Select adequate process areas to be improved and the measures.

5.5.2.3 Experimental design

We recruited 12 undergraduate computer science students to participate in this pilot experiment (one student gave up, leaving us with 11 students). Students had a previous training about CMMI and basic Software Engineering concepts in the undergraduate course. Since this experiment were conducted during the beginning of the semester, we decided to give a brief training session in order to reinforce these concepts. This training session took 30 minutes. Then each student took a background and a pre-test questionnaire, before playing SPIAL, and a post-test, after playing it.

1. **Experimental Variables:** During the experiment, data for a list of variables were collected. Table 5.2 lists all experimental variables, including three variables that represent potentially disturbing factors.

Table 5.2. Experimental Variables.

	List of Variable	Data Collection
Variables	Dep.1 Interest	Pre-test and Post test
	Dep.2 Competency	Pre-test and Post test
	Dep.3 Knowledge on the remembering level	Pre-test and Post test
	Dep.4 Knowledge on the understanding level	Pre-test and Post test
	Dep.5 Knowledge on the application level	Pre-test and Post test
	Dep.6 Engagement	Post-Test
	Dep.7 Appropriateness	Post-Test
	Dep.8 Learning Perspective	Post-Test
Disturbing factors	DiF.1 Personal background	Background Questionnaire
	DiF.2 Additional study	Background Questionnaire
	DiF.3 Material evaluation (personal perception)	Post-Test

Variables:

- Dep.1 Interest in SPI issues. Questions about personal interest in learning more about SPI.
- Dep.2 Competency of students in SPI. Question about personal evaluation of SPI knowledge.

- Dep. 3 Knowledge on the remembering level. Questions about typical SPI knowledge learned in class or during the game.
- Dep.4 Knowledge on the understanding level. Questions about typical characteristics of SPI initiatives. Some of these questions were based on empirical findings and lessons learned that we collected from the literature. In this case, students need to infer the meanings and explain the ideas.
- Dep.5 Knowledge on the application level. Questions about the execution of an SPI initiative. Students were asked to propose an SPI initiative based on an organizational context.
- Dep.6 Engagement. Questions that evaluate whether students had fun and enjoyed the game.
- Dep.7 Appropriateness. Questions that evaluate the students' perceptions about the appropriateness of the game. These were organized in eight dimensions [Gresse von Wangenheim et al., 2009]: duration, difficulty, content relevancy, correctness, sufficiency, sequence, teaching method, and adoption in a Software Engineering course.
- Dep.8 Learning Perspective. Questions about the student's learning perception.

Disturbing Factors: The value of three potentially disturbing factors DiF.1, DiF.2, and DiF.3 are also considered in the questionnaires that all subjects have to fill in.

- DiF.1 Personal background in terms of practical and training experience.
- DiF.2 Additional study time spent besides lectures and preferred learning style.
- DiF.3 Questions on personal judgment of the training session (subjective evaluation).

2. **Experimental Procedure:** We run the experiment with students coursing a last year discipline of a four-year Computer Science program. As part of their participation, the students earned educational credits in an extra-credit exercise. The experiment was conducted following the schedule presented in Table 5.3.

   After a short presentation of the experiment's goals and the main concepts that are prerequisites to this game (CMMI levels and basic Software Engineering concepts), the participants were asked to answer a background questionnaire about

Table 5.3. Schedule of the experiment.

Content	Duration
Introduction to the experiment and theoretical overview	30 min
Personal characteristics and background knowledge	10 min
Pre-test	40 min
Game Play	40 min
Post-Test	40 min
Game evaluation	20 min
Material evaluation	5 min

their personal experience. Questions include information about their professional experience and previous training. Then, a pre-test was conducted to establish a baseline for comparison. The pre-test was composed of 16 questions: 6 on the remembering level, 8 on the understanding level, and 2 on the application level. Following the pre-test, the students play the game. After having concluded the game, a post-test questionnaire was filled by them. This questionnaire has the same questions of the pre-test. In addition, students were asked to make subjective evaluation of the game, including assessment of its learning effects and strengths and weakness. Finally, they evaluated the training session by answering six questions.

3. **Assessment:** The learning outcome was assessed by questionnaires with multiple choice and open questions. These questions were carefully selected to cover both concepts presented in the Software Engineering course and those that were only presented in SPIAL. They allowed gaining an overall understanding of how well the game enables students to remember specific knowledge taught and how well students can infer knowledge that was not taught in their course. We followed the principles proposed by Kitchenham et al. [2002] and we adapted some of the questions based on the Lethbridge work [Lethbridge, 1998], and on other important studies [Gresse von Wangenheim et al., 2009; Navarro, 2006; Pfahl, 2001]. In the close questions, we ensured that each value on the scale has a well-defined meaning, and that the high and low values are true extremes. The questionnaires are presented in Appendix D.

4. **Data Collection Procedure:** We treated the raw data in order to carry out data analysis.

   Variables: The value for variable **Dep.1** is derived from five questions on the students' interest in SPI issues, applying a five-point Likert-type scale [Likert,

1932]. The answers are mapped to a value range R = [0,1], where "fully disagree" is encoded as "0", "disagree" as "0.25", "undecided" as "0.5", "agree" as "0.75", and "fully agree" as "1".

The value for variable **Dep.2** is derived from one question about the students' CMMI knowledge. Similarly to variable Dep.1, the answers are mapped to a value range R = [0,1].

The values for variables **Dep.3**, **Dep.4** and **Dep.5** are average scores derived from 13 questions on multiple-choice style and three open questions. The answers to these questions were evaluated according to their correctness. Thus, for multiple-choice style questions, having a binary scale with correct answers encoded as "1", and incorrect answers encoded as "0", and a value range R=[0, 1] for open questions.

The value for variable **Dep.6** is derived from eight questions on students' opinion about the game. The answers are mapped to a value range R = [0, 1], where a "strong negative" is encoded as "0", a "weak negative" as "0.25", undecided as "0.5", "positive" as "0.75", and a "strong positive" as "1".

The value for variable **Dep.7** is derived from two questions on students' opinion about the game engagement. The answers are mapped to a value range R = [0, 1], like the variable **Dep.6**.

The value for variable **Dep.8** is derived from three questions on students' opinion about the learning perspective. The answers are mapped to a value range R = [0, 1], like the variable **Dep.6**.

5.5.2.4 Results

In this first experiment, our aim was to have an overall understanding of the game. For this purpose, we asked all the students to play the game and answer the questions.

In total, 12 students participated in this experiment and 11 participants completed it. Table 5.4 summarizes the students' personal characteristics.

We used descriptive statistics to analyze the questions. The aim was to identify central tendencies and dispersion on the variables. For variables Dep.3, Dep.4 and Dep.5 we analyze the mean, median and standard deviation. For variables Dep.1, Dep.2, Dep.6, Dep.7 and Dep.8 we calculate the median and interquartile range, as they are measured on ordinal scales. Tables 5.5 and 5.6 show the raw data collected during pre-test and post-test for these variables.

Table 5.4. Overview of participants' personal characteristics.

Personal characteristics	
Number of participants	11
Gender	
Male	11
Female	0
Average age	22 years
Academic formation	
Bachelor in Computer Science	7
Bachelor in Information Systems	2
Bachelor in Electric Engineering	1
Bachelor in Airspace Engineering	1
Professional certification	0
First Software Engineering course	11
Work load	
Full-time student	7
Working 20 hours	1
Working 30 hours	2
Working 40 hours	1
Current professional position	
Software developer	4
Complementary training	
Project Management	1
Readings	
# of Books	
0	10
1-2	1
# of Papers	
0	9
1-2	1
3-5	1
CMMI acronym	
Right answer	9
Don't know	2
Learned CMMI and Software Engineering concepts	
Yes	5
No	6
Preferred learning style (multiple options possible)	
Reading of text books	7
Classroom lectures	6
Group work	2
Web-based training modules	5

Table 5.5. Descriptive Analysis (Dep.3, Dep.4 and Dep.5 variables).

	Dep.3	Dep.4	Dep.5
Pre-Test			
Mean	0,37	0,28	0,14
Median	0,33	0,22	0,14
Standard Dev.	0,21	0,29	0,04
Post-test			
Mean	0,37	0,35	0,53
Median	0,28	0,33	0,53
Standard Dev.	0,20	0,28	0,12

Table 5.6. Descriptive Analysis (Dep.1, Dep.2, Dep.6, Dep.7 and Dep.8 variables).

	Dep.1	Dep.2	Dep.6	Dep.7	Dep.8
Pre-Test					
Median	0,81	0,31	-	-	-
Interquartile range	0,24	-	-	-	-
Post-test					
Median	0,86	0,39	0,70	0,80	0,47
Interquartile range	0,15	-	0,22	-	0,23

In general, the interest in SPI improved between the pre and the post-test (Dep.1). The answers indicate that students considered important the SPI knowledge both in academic and in professional life. An interesting observation is that, after playing the game, students evaluated with high scores the importance of learning SPI in an undergraduate program (Dep.1.1 and 1.2).

The subjective auto-evaluation of the student's measured competency (Dep.2) showed that they evaluated their competency with higher values in the post-test than in the pre-test, suggesting that they have learned something after playing SPIAL.

Considering the descriptive statistics, we cannot identify a significant difference between the pre and post-test regarding the central tendency of measured knowledge on the remembering level (Dep.3) and on the understanding level (Dep.4). Only very few students presented a small improvement (see Figure 5.13). In terms of measured gain in software process knowledge, students improved somewhat in the applying level

Dep.5. This could be partially due to the fact that, playing SPIAL, students have a practical idea of what happens in a process improvement program.

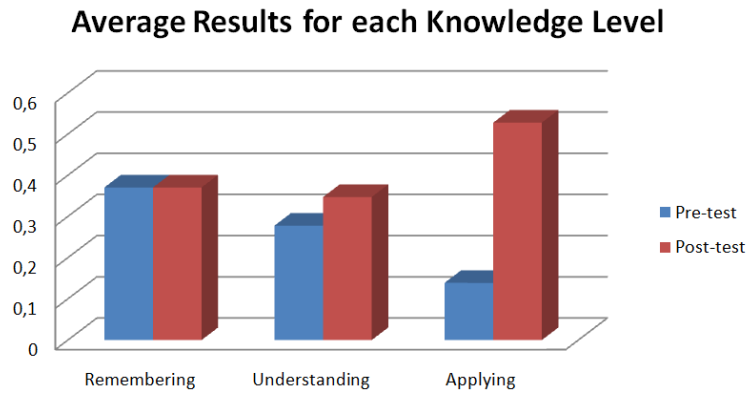

Figure 5.13. Average score results for each knowledge level.

On average, students found SPIAL quite enjoyable and they had fun during the game play (Dep.7). They also felt that the game duration was appropriate and it was relatively easy to play (Dep.6). They agreed about adopting SPIAL in a Software Engineering course as a complementary approach. The students considered the game content relevant to their learning. They relatively thought that SPIAL reflects aspects of a real Software Process initiative. They observed that this game was sufficient when considering its purpose and it had a satisfactory play sequence. All students agreed that a traditional Software Engineering class with SPIAL will be better than without it. They moderately learned new concepts and practical application of an SPI program in an organization. They felt that SPIAL was more successful in reinforcing concepts taught in Software Engineering course than teaching new concepts (Dep.8). Figures 5.14, 5.15, 5.16 and 5.17 present the results of each evaluated game aspect.

In open-ended questions, students pointed out their positive feelings about SPIAL: "Enjoyable and challenging. I like to play it." , "It was easy to play when you know what you are doing", and "The game emphasizes where the player should invest in different stages of the project. This is a concept that would be hardly fixed without some kind of practical activity". Concerning the favorite aspects, the most cited one was the feedback and the way it was presented (four citations), followed by the investments in process areas (two citations) and the events (one citation).

Although these positive responses, it was clear from this experiment that some aspects of the game needed to be improved. The confusing and least favorites aspects pointed out by the students are related to the underlining game behavior. This problem has been previously detected during the semiotic inspection method, but the corrections

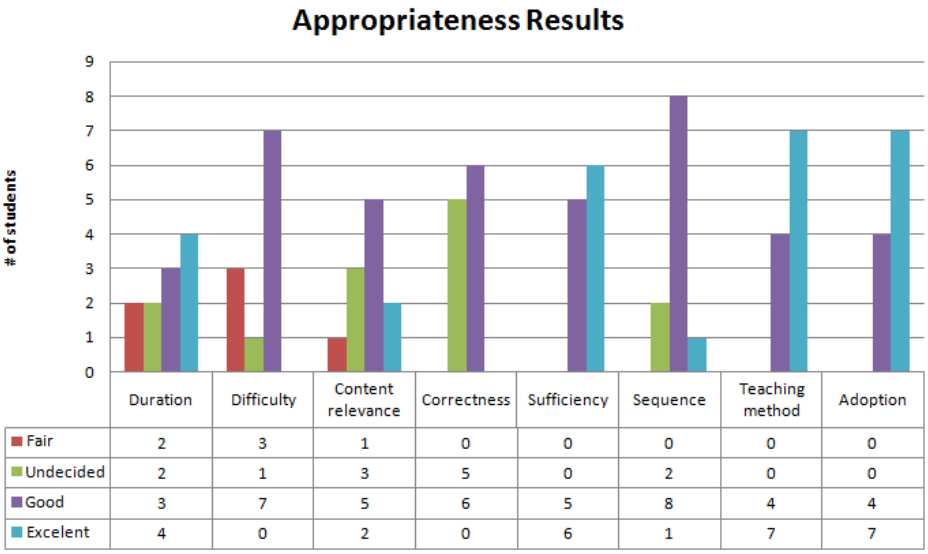

Figure 5.14. Results on Dep.6 Appropriateness (n=11).

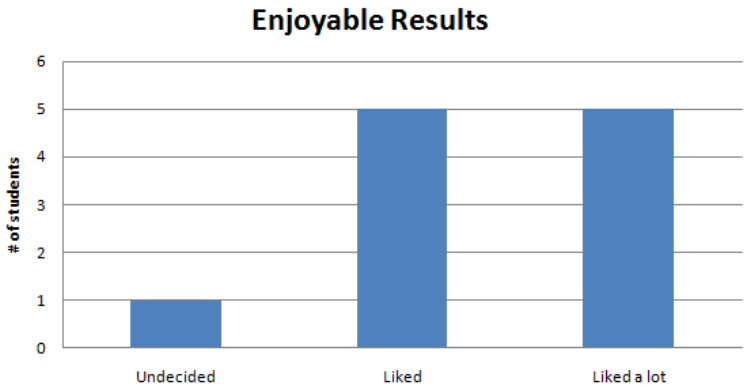

Figure 5.15. Results on Dep.7 Enjoyable (n=11).

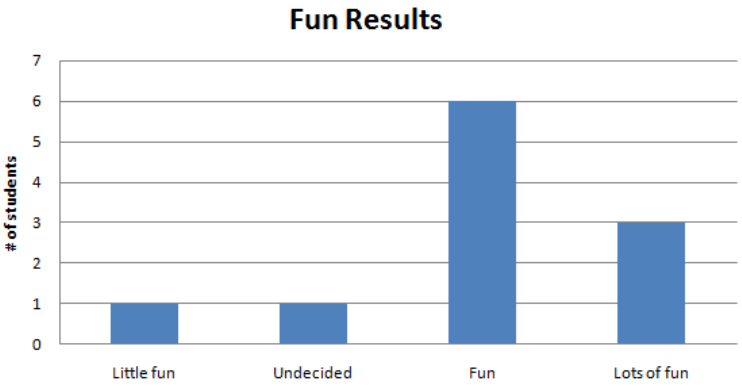

Figure 5.16. Results on Dep.7 Fun (n=11).

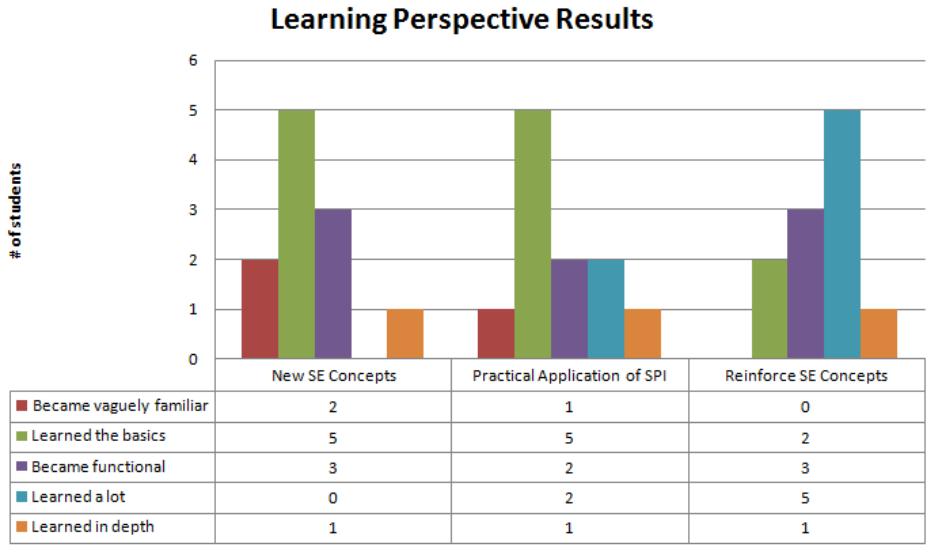

Figure 5.17. Results on Dep.8 subjective learning perspective (n=11).

were not sufficient. For instance, some students did not know how to overcome the negative feedback originated from some of the stakeholders, other did not understand the results of the process area investments and were in doubt about their investment strategy. Other issues that students were unhappy consist of technical aspects, such as the lack of the stakeholder information in the event log tab and the short duration of the messages that appear over the employees' head.

Since we wanted to guarantee anonymity to the students, we were not able to correlate the game scores with the data collected with the background questionnaire. We observed that most of the students (seven, 64%) played the game until they got the maximum score (10 points). At the first time, almost all of them got scores close to zero. We believe that the contributing factor for the low score was the lack of a suitable investment strategy. The students played over and over again in order to master the game and discover the process areas investment strategy. This is the reason why some students suggested guidance, mainly at the beginning of the game.

When we analyze the influencing factor DiF. 3, students evaluated the training session as useful, providing important information for the game play (CMMI levels and basic Software Engineering concepts).

5.5.2.5 Threats to Validity

The experiment study design applied in this research may cause a number of problems, reducing the validity of the results. An important issue is the lack of a control group as a basis to make a comparison and record differences. In this first attempt, we opted

to conduct an informal experiment in order to have an overall understanding of the students' reactions who played SPIAL. Our purpose was to make an initial judgment about its effectiveness as an educational tool. Since the experiment was run with a very small set of participants, we considered this situation a possible way to conducted SPIAL evaluation.

The experiment was conducted by one instructor who was responsible for the training sessions. This instructor also reviewed the answers and collected the data. This was done in order to reduce any bias during the tests review.

The questionnaire design is another important issue to be considered. Although students' knowledge was assessed by pre and post questionnaires, it was not possible to infer significant results. This was due to the fact that the questionnaires contain knowledge not only covered by the game. In addition, a maturation effect can have been caused, since both questionnaires have the same set of questions. We tried to minimize this effect by not giving any feedback before the post-test. We also administrated it in a different day of the pre-test. However, the repetition of the tests may have caused a loss of motivation, reducing the students' performance. Another important aspect is the number of questions. The tests score can be considered a limited measure to represent the learning effect due to the fact that it has been resulted from a relative small set of questions.

The very small sample size also represents a threat. As this is a pilot study to gain initial insights on the learning effectiveness of the educational game, we accepted these weak results as a first game evaluation attempt.

Due to the academic environment where the experiment happened, the number of subjects involved and their gender (only males), a generalization of the results may be done with caution. It can be expected that the results are at an extent representative for this class of subjects.

Although the tests were limited and may not be representative for a complete SPI initiative, we carefully prepare the tests based on a large collected empirical experience from the literature that covers, as much as possible, some important issues of a software process improvement program. A point that should be emphasized is that the research at its current stage is exploratory of nature and it is just a first step of a series of experiments that might yield more general results in the future.

Another threat is that aspects of the game evaluation were captured through subjective measures. To counteract this threat to validity, the questionnaires were developed based on existing similar studies [Gresse von Wangenheim et al., 2009; Navarro, 2006; Pfahl, 2001]. In addition, students might not be able to provide valuable feedback on the correctness, relevancy and completeness of the game. However, we did not ana-

lyze these characteristics only based on the students' opinion. During the development, experts performed informal reviews.

Finally, the application knowledge might not take place since we considered that students were quite immature to use a particular topic being taught. We confirmed this observation when we evaluated the disturbing factors. Four students work part-time in a company (DiF. 1), only one student participated in a complementary training (Project Management training)(DiF.1), three students have read additional material (DiF.2) and no student has a certification (DiF.1). However, the game play reduced drastically the effects of this threat (see Table 5.5, Dep.5).

5.5.3 Discussion

The obtained results are a first indication that SPIAL can be beneficial to reinforce and teach new Software Engineering concepts. Considering the subjective evaluations, students mentioned having fun during the game play. They expressed that playing the game was an enjoyable experience. Opportunities for improvement include the representation of other phenomena into the simulation game, improvement of the feedback and some interface design aspects.

To summarize, our first pilot experience revealed the following lessons:

- *SPIAL seems to have the potential to be a complementary tool in an introductory Software Engineering course.* Students who played SPIAL said that it is a reasonably educational tool, they were able to learn and reinforce Software Engineering concepts. In addition, they recommended its adoption in an introductory Software Engineering course.

- *SPIAL needs additional guidance in order to make the experience potentially more educationally effective.* The main complaint was the lack of guidance, resulting in disagreements about the underlining game behavior. It is clear that more help is needed in order to make the experience more positive.

- *The longer students play SPIAL, the more the score increases.* Students only learn the investment strategy after playing continually the game. They attempt a few incorrect strategies before discovering a correct one. Thus, an important factor in using SPIAL effectively consists of the number of times that students play it.

- *SPIAL's feedback mechanisms are an useful resource for helping players understand software process improvement concepts.* The feedback gives to the players

deeper insights into the correct and incorrect actions that they might have done during the game. This motivates players, supporting their learning process.

- *SPIAL's feedback mechanism allowed students to achieve the maximum score.* The students' scores on the game were quite high. Besides all the complains, this seems to suggest that the majority of students were able to learn most of the underlining game concepts.

5.6 SPIAL x Software Engineering Simulation Games

We can identify four essential differences between SPIAL and the existing Software Engineering educational simulation games:

- **Feedback and Software Engineering concepts**: most of the simulation games have the final goal to develop a software project within a certain set of constraints, and their rules are based on the Software Engineering practices. Typically, they present a metaphor of a software development office, where the player assumes a project manager role. The experimental results of using these tools have shown a restricted number of new concepts that students have learned after playing them. This can be a result of a not straightforward performance feedback that was presented to the students or the way that the virtual world was designed. In order to address these two aspects, we changed the context of our simulation game and we integrated the feedback during the whole game. In SimSE, for example, the performance feedback, presented through the Explanatory tool is an additional functionality that it is not integrated into the simulation environment. As presented in Chapter 3, we observed that students usually do not access this functionality. Since introductory Software Engineering courses typically present the best practices of Software Engineering superficially, we believe that the students will not only reinforce with SPIAL the concepts learned at the class but also will learn new concepts when playing it, which are important to understand the way that an organization works.

- **Evaluating the interaction**: As mentioned in Section 2.4, one aspect for a successful simulation game is that its interface design should take into account some basic elements to support learning and motivate students (e.g. feedback, clear goals, and game features). The existing simulation games have not been evalu-

ated considering its interface design. In our case, we conducted communicability evaluation in order to improve SPIAL interface design.

- **Verifying the results**: There are relatively few studies in the Software Engineering simulation game area that have been extensively verified through experiments. For example, SimSE has been verified through in-class and out-of-class experiments with the aim to assess it as an educational tool. Besides the communicability evaluation, we carried out a pilot experiment with undergraduate students of a Computer Science Software Engineering course.

- **Developing an adaptable game**: We know that an essential feature of a simulation game is to be easily configurable since there are many models of different processes, different process improvement frameworks, students with different learning styles and levels of initial knowledge, and instructors with different expectations and objectives. In the first version of SPIAL, we designed a configurable simulation system in order to allow distinct definitions of simulation models. In addition, we developed core simulation components needed to support the creation of other simulation games.

5.7 Conclusion

This section presented SPIAL, our SPI simulation game. SPIAL was developed using a reusable framework, which allows its adjustment to distinct learning environments. This can assist students with different learning styles, different levels of initial knowledge and instructors with different expectations and objectives.

We evaluated SPIAL from two viewpoints: specialist and player.

An inspection using the Semiotic Inspection Method was conducted by a specialist and communicability breakdowns were identified. According to the specialist, despite feedbacks given during the whole game, such as measurement charts and improvement analyses, important aspects to understand the core behavior were missing, such as the reason why sometimes investments do not produce any improvement.

We evaluated the game carrying out a pilot experiment with undergraduate students of a Computer Science Software Engineering course as players. We selected (i) questions to evaluate whether students enjoyed the game; (ii) questions that evaluate the students' perceptions about the appropriateness of the game; these were organized in eight dimensions: duration, difficulty, content relevancy, correctness, sufficiency, sequence, teaching method, and adoption in a Software Engineering course; and (iii)

questions about the learning perspective. The first two categories of question are indirectly related to the learning, showing how effective is the educational environment. On average, students found the game quite enjoyable and they had fun during the game play. They also felt that the game duration is appropriate and it is relatively easy to play. They agreed about adopting this game in a Software Engineering course as a complementary approach. They moderately learned new concepts; however, they felt this game more successful in reinforcing concepts taught in Software Engineering course than teaching new concepts.

We observed that some learning theories, such as Learning through Reflection, Elaboration Theory and Aptitude Treatment Interaction, were not appropriately addressed by the analyzed simulation games [Possa, 2011]. The identified requirements and the corresponding framework, FASENG, can support designers of new simulation games to address important learning theories during the initial development phases.

When we analyzed the questions posed at the beginning of this section, the evaluation results provided the following answers:

1. **What is the perception of the students about SPIAL? (e.g. it is enjoyable) What are its strengths and weaknesses? How well does SPIAL design is appropriate for their purpose?** Students enjoyed and had fun playing SPIAL. Students felt that it is a reasonable tool for learning Software Engineering concepts. They also felt that the game duration was appropriate and it was relatively easy to play. The students considered the game content relevant to their learning. One positive aspect is the feedback presented during the whole game. Enhancements are needed to teach some SPI behavior embodied in the simulation model. In addition, it is required a more useful graphical mechanism and more accessible rules description.

2. **Is there any difficulty to play SPIAL?** One difficult aspect of SPIAL is to understand its basic behavior in the first attempt, such as, the increasing or decreasing effects on the investment points, and in which process area to invest.

3. **Can students actually learn software process concepts from using SPIAL, considering effects on the remembering, understanding and applying level?** Students who played SPIAL seemed to capture the concepts represented in the simulation model. However, evaluating the results we cannot identify a significant difference between the pre and post-test. Thus, the learning effect could not be confirmed through this kind of evaluation.

4. **How can such a simulator be incorporated into Software Engineering courses?** Students agreed about adopting SPIAL in a Software Engineering course as a complementary approach. It is needed a proper amount of guidance and instruction of use to fulfill its educational potential.

These evaluations indicate the potential of SPIAL to support education. In addition, it provides first insights on the strengths and weaknesses, which will guide further studies. Besides answering our research questions, this experiment exposed issues that needed to be addressed in other studies, such as the improvement of the game interface and the better representation of SPI initiatives.

Regarding the limitations of FASENG, one could argue that we did not identify all requirements that can be traced back to the described learning theories. Although this is a valid criticism, we believe that for the evaluated Software Engineering simulation games they are appropriate, mainly considering an initial development of a reusable framework. A broad validation mechanism should be carried out. For example we need to create validation mechanisms to verify requirements from different stakeholders' viewpoints (students, instructors, and other developers).

We cannot *a priori* assume that the results of this study generalize beyond the specific environment of Software Engineering simulation games. We identified requirements exclusively from simulation games of the Software Engineering field. However, we believe that many considerations can be applied to the whole educational simulation game field.

As a future work, we plan to address important issues detected during the first evaluations, mainly incorporating other phenomena that happen in the real world. This will make the simulation experience resembles more closely those in industry. We will also carry out additional experiments in order to assess other aspects of SPIAL, as for example its relationship with the instructor.

Chapter 6

Conclusion

The focus of this thesis is on Software Engineering simulation games to support undergraduate students of introductory Software Engineering courses.

This last chapter is structured as follows:

- Section 6.1 summarizes the research achievements.
- Section 6.2 presents the lessons learned during SPIAL development.
- Section 6.3 outlines possible future works.

6.1 Research Achievements

The main achievement of this research lies in the design, application and validation of SPIAL, an SPI simulation game. The novelty of this work is twofold. Firstly, it presents an SPI simulator, along with its simulation model and reusable framework. Secondly, it depicts the steps needed to create the simulation game, which includes the analysis of Human-Computer Interactions issues and SPI literature.

This research provided results on theoretical, practical and empirical level. The specific achievements can be summarized as follows:

- **Identification of students' mistakes:** We investigated the common problems incurred by students in a Software Engineering course. The most striking observations are the difficulty that students have to bridge the gap between the theoretical lectures and the team project, the limited range of skills that they apply during the project development, and, usually, they do not follow the prescribed software development process during the team project development. They

first elaborate the technical artifacts and then they complete the baseline with the managerial artifacts.

- **Analysis of other simulation games:** We identified and compared eight Software Engineering simulation games. Based on this analysis, we defined the essential design aspects for the development of our simulator. The assessment of these simulation games revealed that, although they are, in some cases, developed with different aims, they have a considerable number of common aspects, for example, "trial and error" strategy, goals, rules and some game features. They also have different features mainly regarding when the feedback is available, the type of feedback, and the adaptability characteristics.

- **Investigation of the SPI and Software Engineering domains:** We investigated the domain of our simulation game. Since a simulation game should reflect what happens in the real world, we analyzed the main results reported by organizations in research papers regarding their SPI initiatives. Through a systematic literature review, we gained an up-to-date view of the SPI area, allowing us to identify and characterize the actual results of SPI initiatives. In addition, we collected 123 Software Engineering rules from text books and other Software Engineering simulation games. This set includes rules that are dependent on values that vary from one situation to another and rules beyond the Software Engineering area, including a wider range of business processes. From this set, we selected the ones related to the SPIAL process areas, resulting in 57 Software Engineering rules. These rules are used to teach the best practices of Software Engineering to students, rewarding or penalizing their actions.

- **Development of SPIAL:** Based on the steps previously carried out, we identified important requirements for our simulation game, which were used in the definition of the SPIAL behavior. We also evaluated the available components for simulation games development. Since we did not find components that were generic enough to be reused, we decided to develop our own framework, named FASENG. FASENG was developed based on a set of selected requirements related to the application of learning theories.

- **Evaluation of SPIAL:** Finally, we evaluated SPIAL from two viewpoints: specialist and player. An inspection using the Semiotic Inspection Method was conducted by a specialist and communicability breakdowns were identified. We evaluated the game carrying out a pilot experiment with undergraduate students

of a Computer Science Software Engineering course as players. On average, students found the game quite enjoyable and they had fun during the game play. They agreed about adopting this game in a Software Engineering course as a complementary approach.

6.2 Lessons Learned

Another important result corresponds to a set of issues that emerged during the simulation game design and development. These issues can guide new developers and instructors in the design and selection of educational simulation games. It was identified:

- The importance of having a core theoretical reference model, in order to assist the simulation model definition and validation. During the preparation of experiments, this model can assist in the questions definition. In our case, we used the latest version of the CMMI model, which guided the definition and validation of the mathematical framework represented in our simulation model.

- The considerable effort needed to deal with rules in general and Software Engineering related rules in particular. The use of rules is a complex and wide area.

- The importance of concepts definition scope to be covered by the simulation game. This should be done according to the students' needs and the instructors' teaching objectives.

- The influence of the virtual world processes addressed by the simulation game on the students' learning. It is interesting to include activities that trace back concepts, processes and game-like elements during the design step.

- The considerable effort needed to calibrate the simulation. In the Software Engineering area, it is difficult to find enough data in the literature (or with appropriate quality level). Beyond the instructor, we can appeal, for example, to specialists to help in the model calibration. However, we need to verify whether the results approximate the real world and whether they are suitable for an educational environment.

- The game-like elements assist the students' learning process, enhancing challenge and motivation. The designer needs to select which concepts and processes will be represented as game-like elements.

6.3 Future Work

The SPIAL evaluations have highlighted important aspects that can be improved. Possible future work includes:

1. Incorporating other phenomena that happen in the real world: More SPI phenomena need to be identified, in order to make the simulation closer to the SPI initiatives in software development organizations.

2. Developing supporting tools: New tools should be developed in order to support tasks related to the configuration and maintenance of a simulation game such as SPIAL. Examples are simulation model tailoring and configuration files management.

3. Creating guidelines: During FASENG development, we identified three roles: simulation game developer (who adapts the interface and creates the Java classes), simulation model designer (who specifies the behavior), and developer (who creates the XML file). In particular, it will be interesting to create a guideline for each of these roles.

4. Improving interface design: Interface deficiencies brought forth during the evaluations will be addressed. Graphics enhancement will make SPIAL more engaging, for example, adding animations.

5. Developing other simulation models: The modeling of other SPI reference models will permit to check the strength and adaptability of our environment.

6. Conducting more evaluations: Additional experiments and evaluations need to be conducted in order to verify SPIAL potential as a complementary educational method, including validations with different roles (e.g. professor, developers). For instance, control experiment and usability test with students, and observational experiment with developers. In addition, experiments and evaluations need to be conducted to verify how FASENG requirements are related to the requirements of other simulation games.

7. Designing debriefing sessions: Debriefing sessions allow checking how the player's performance was close to the expected performance and what is needed to bridge this gap. It is important to propose a process for planning, designing and conducting debriefing sessions after a SPIAL game section.

8. Enhancing the investments possibilities: It will be interesting to include the possibility of investments in the process areas of an organization, or in specific/generic practices of a process area.

9. Generating development process guidelines: It will be worthwhile to investigate alternative development process fitting the specific needs of simulation games, including usability activities and communicability evaluations.

10. Enhancing and validating the educational support: Educational related processes (e.g. grading process) should be added and assessed in order enhance SPIAL usability. SPIAL interface with trainers should also be enhanced and validated.

11. Evaluating motivational models: Different models for game play motivations have been proposed [Garris et al., 2002]. As part of our early decisions, we did not consider directly the questions of motivation. An interesting future work is the evaluation of different motivation perspectives which will enhance the learning process.

Bibliography

Aaen, I., Börjesson, A., and Mathiassen, L. (2007). SPI agility: How to navigate improvement projects. *Software Process: Improvement and Practice*, 12(3):267–281.

Aasheim, C. L., Li, L., and Williams, S. (2010). Knowledge and Skills Requirements for Entry-Level Information Technology Workers: A Comparison of Industry and Academia. *Journal of Information Systems Education*, 20(3):349–356.

Abdel-Hamid, T. and Madnick, S. E. (1991). *Software Project Dynamics: An Integrated Approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Alessi, S. M. and Trollip, S. R. (2001). *Multimedia for Learning: Methods and Development. Third Edition*. Allay and Bacon, Needham Heights, Massachusetts, USA.

Allison, I. and Merali, Y. (2007). Software process improvement as emergent change: A structurational analysis. *Information and Software Technology*, 49(6):668–681.

Almstrum, V. L., Dale, N., Berglund, A., Granger, M., Little, J. C., Miller, D. M., Petre, M., Schragger, P., and Springsteel, F. (1996). Evaluation: turning technology from toy to tool: Report of the working group on evaluation. In *Proceedings of the 1st Conference on Integrating Technology into Computer Science Education*, ITiCSE '96, pages 201–217, Barcelona, Spain.

Anda, B., Hansen, K., Gullesen, I., and Thorsen, H. (2006). Experiences from introducing UML-based development in a large safety-critical project. *Empirical Software Engineering*, 11:555–581.

Anderson, L. W. and Krathwohl, D. R. ((eds) 2001). *A taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Longman, New York.

Appleton, B. (1997). Patterns for Conducting Process Improvement. In *PloP'97 Conference*, Illinois, USA.

Armbrust, O., Ebell, J., Hammerschall, U., Münch, J., and Thoma, D. (2008). Experiences and results from tailoring and deploying a large process standard in a company. *Software Process: Improvement and Practice*, 13(4):301–309.

Auvinen, J., Back, R., Heidenberg, J., Hirkman, P., and Milovanov, L. (2006). Software Process Improvement with Agile Practices in a Large Telecom Company. In *Proceedings of the 7th International Conference on Product Focused Software Process Improvement*, PROFES'06, pages 79–93, Amsterdam, The Netherlands.

Baddoo, N. and Hall, T. (2002a). Motivators of Software Process Improvement: An analysis of practitioners' views. *Journal of Systems and Software*, 62(2):85–96.

Baddoo, N. and Hall, T. (2002b). Software Process Improvement Motivators: An Analysis using Multidimensional Scaling. *Empirical Software Engineering*, 7:93–114.

Baddoo, N. and Hall, T. (2003). De-motivators for software process improvement: an analysis of practitioners' views. *Journal of Systems and Software*, 66(1):23–33.

Baker, A., Navarro, E., and van der Hoek, A. (2003). Problems and Programmers: An Educational Software Engineering Card Game. In *Proceedings of the 25th International Conference on Software Engineering*, pages 614–619, Portland, Oregon, USA.

Balla, K., Bemelmans, T., Kusters, R., and Trienekens, J. (2001). Quality through Managed Improvement and Measurement (QMIM): Towards a Phased Development and Implementation of a Quality Management System for a Software Company. *Software Quality Journal*, 9:177–193.

Banks, J., Carson II, J. S., Nelson, B. L., and Nicol, D. M. (2009). *Discrete-Event Simulation (5th ed.)*. Prentice-Hall.

Barnett-Page, E. and Thomas, J. (2009). Methods for the synthesis of qualitative research: a critical review. *BMC Medical Research Methodology*, 9(59).

Barros, M. d. O., Dantas, A. R., Veronese, G. O., and Werner, C. M. L. (2006). Model-driven game development: experience and model enhancements in software project management education. *Software Process: Improvement and Practice*, 11(4):411–421.

Barros, M. d. O., Werner, C. M. L., and Travassos, G. H. (2002). A system dynamics metamodel for software process modeling. *Software Process: Improvement and Practice*, 7(3-4):161–172.

Barzilay, O., Hazzan, O., and Yehudai, A. (2009). A Multidimensional Software Engineering Course. *IEEE Transactions on Education*, 52(3):413–424.

Basili, V. and Perricone, B. T. (1993). Software errors and complexity: an empirical investigation. In *Shepperd, M., editor, Software Engineering Metrics*, pages 168–183. McGraw-Hill, Inc.

Batista, J. and Figueiredo, A. D. D. (2000). SPI in a very small team: a case with CMM. *Software Process: Improvement and Practice*, 5(4):243–250.

Becker, A. L., Prikladnicki, R., and Audy, J. L. N. (2008). Strategic alignment of software process improvement programs using QFD. In *Proceedings of the 1st International Workshop on Business Impact of Process Improvements*, BiPi'08, pages 9–14, Leipzig, Germany. ACM.

Becker, K., Ruiz, D. D., Cunha, V. S., Novello, T. C., and Souza, F. V. (2006). SPDW: A Software Development Process Performance Data Warehousing Environment. In *Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop*, SEW'06, pages 107–118, Columbia, Maryland.

Becker-Kornstaedt, U., Hamann, D., Kempkens, R., Rö, P., Verlage, M., Webby, R., and Zettel, J. (1999). Support for the Process Engineer: The Spearmint Approach to Software Process Definition and Process Guidance. In *Proceedings of the 11th Conference on Advanced Information Systems Engineering*, CAiSE'99, pages 119–133, Heidelberg, Germany. Springer Berlin / Heidelberg.

Beer, M., Eisenstat, R. A., and Spector, B. A. (1990). Why Change Programs Don't Produce Change. *Harvard Business Review*, 68(6).

Bibi, S., Stamelos, I., Gerolimos, G., and Kollias, V. (2010). BBN based approach for improving the software development process of an SME - a case study. *Journal of Software Maintenance and Evolution: Research and Practice*, 22(2).

Birkhölzer, T., Dantas, L., Dickmann, C., and Vaupel, J. (2004). Interactive Simulation of Software Producing Organization's Operations based on Concepts of CMMI and Balanced Scorecards. In *Proceedings of the 5th International Workshop on Software Process Simulation and Modeling*, ProSim'04, pages 123–132, Edinburgh.

Birkhölzer, T., Dickmann, C., Vaupel, J., and Dantas, L. (2005a). An interactive software management simulator based on the CMMI framework. *Software Process: Improvement and Practice*, 10(3):327–340.

Birkhölzer, T., Dickmann, C., Vaupel, J., and Stubenrauch, J. (2005b). Towards an Interactive Simulator for Software Process Management under Uncertainty. In *Proceedings of the 6th International Workshop on Software Process Simulation and Modeling*, ProSim'05, pages 169–174, St. Louis, Missouri, USA.

Biró, M., Ivanyos, J., and Messnarz, R. (2000). Pioneering process improvement experiment in Hungary. *Software Process: Improvement and Practice*, 5(4):213–229.

Blanco, M., Gutierrez, P., and Satriani, G. (2001). SPI Patterns: Learning from Experience. *IEEE Software*, 18(3):28–35.

Boehm, B. and Basili, V. R. (2001). Software Defect Reduction Top 10 List. *Computer*, 34:135–137.

Boehm, B. W. (1981). *Software Engineering Economics*. Upper Saddle River, NJ:Prentice Hall, Inc.

Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D., and Steece, B. (2000). *Software Cost Estimation with Cocomo II*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

Boehm, B. W., McClean, R. K., and Urfrig, D. B. (1975). Some experience with automated aids to the design of large-scale reliable software. In *Proceedings of the International Conference on Reliable Software*, pages 105–113, Los Angeles, California. ACM.

Booch, G. (1991). *Object oriented design with applications*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA.

Börjesson, A., Martinsson, F., and Timmerås, M. (2005). Crossing the Chasm in Software Process Improvement. In *Baskerville, R., Mathiassen, L., Pries-Heje, J., and DeGross, J., editors, Business Agility and Information Technology Diffusion*, volume 180 of IFIP International Federation for Information Processing, pages 111–128. Springer Boston.

Börjesson, A. and Mathiassen, L. (2004). Organisational Dynamics in the Software Process Improvement: The Agility Challenge. In *Fitzgerald, B., and Wynn, E., editors, IT Innovation for Adaptability and Competitiveness*, volume 141 of IFIP International Federation for Information Processing, pages 135–156. Springer Boston.

Brodman, J. G. and Johnson, D. L. (1995). Return on Investment (ROI) from Software Process Improvement as Measured by US Industry. *Software Process: Improvement and Practice*, pages 35–47.

Brooks, Jr., F. P. (1987). No Silver Bullet Essence and Accidents of Software Engineering. *Computer*, 20:10–19.

Budgen, D., Kitchenham, B. A., Charters, S. M., Turner, M., Brereton, P., and Linkman, S. G. (2008). Presenting software engineering results using structured abstracts: a randomised experiment. *Empirical Software Engineering*, 13:435–468.

Caliò, A., Autiero, M., and Bux, G. (2000). Software process improvement by object technology (ESSI PIE 27785 - SPOT). In *Proceedings of the 22nd International Conference on Software engineering*, ICSE'00, pages 641–647, Limerick, Ireland. ACM.

Callahan, D. and Pedigo, B. (2002). Educating experienced IT professionals by addressing industry's needs. *IEEE Software*, 19(5):57–62.

Calvo-Manzano Villalón, J. A., Cuevas Agustín, G., San Feliu Gilabert, T., De Amescua Seco, A., García Sánchez, L., and Pérez Cota, M. (2002). Experiences in the Application of Software Process Improvement in SMES. *Software Quality Control*, 10(3):261–273.

Card, D. (1998). Learning from our mistakes with defect causal analysis. *IEEE Software*, 15(1):56–63.

Card, D., Domzalski, K., and Davies, G. (2008). Making Statistics Part of Decision Making in an Engineering Organization. *IEEE Software*, 25(3):37–47.

Carrington, D., Baker, A., and van der Hoek, A. (2005). It's All in the Game: Teaching Software Process Concepts. In *Proceedings of the 35th ASEE/IEEE Frontiers in Education Conference*, FIE'05, page F4G, Indianapolis, Indiana.

Carrol, J. M. (1997). Scenario-Based Design. In *Helander, M. G. , Laundauer, T. K., and Prabhu, P. V. , editors, Handbook of Human-Computer Interaction Second, completely revised edition.* Eslevier Science BV.

Casey, V. and Richardson, I. (2004). A practical application of the IDEAL model. *Software Process: Improvement and Practice*, 9(3):123–132.

Cattaneo, F., Fuggetta, A., and Sciuto, D. (2001). Pursuing coherence in software process assessment and improvement. *Software Process: Improvement and Practice*, 6(1):3–22.

Chrissis, M. B., Konrad, M., and Shrum, S. (2006). *CMMI: Guidelines for Process Integration and Product Improvement, 2nd edition*. SEI Series in Software Engineering. Addison-Wesley, EUA.

Christie, A. M. (1999). Simulation in support of CMM-based process improvement. *Journal of Systems and Software*, 46(2-3):107–112.

Claypool, K. and Claypool, M. (2005). Teaching Software Engineering Through Game Design. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '05, pages 123–127, Caparica, Portugal. ACM.

CMMI (2010). CMMI ®for Development, Version 1.3. Technical Report CMU/SEI-2010-TR-033, Software Engineering Institute.

Cobb, R. and Mills, H. (1990). Engineering software under statistical quality control. *IEEE Software*, 7(6):45–54.

Cole, A. (1995). Runaway Projects-Cause and Effects. *Software World (UK)*, 26(3).

Colomo-Palacios, R., Casado-Lumbreras, C., Soto-Acosta, P., García-Peñalvo, F. J., and Tovar-Caro, E. (2012). Competence gaps in software personnel: A multi-organizational study. *Computers in Human Behavior*, page To appear.

Coman, I. D., Sillitti, A., and Succi, G. (2009). A Case-Study on using an Automated In-process Software Engineering Measurement and Analysis System in an Industrial Environment. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE'09, pages 89–99, Vancouver, British Columbia, Canada. IEEE Computer Society.

Commission, E. (2005). The new SME definition: User guide and model declaration. *http://ec.europa.eu/enterprise/policies/sme/files/sme_definition/sme_user_guide_en.pdf*. [Online; accessed 17th August 2011].

Conn, R. (2002). Developing Software Engineers at the C-130J Software Factory. *IEEE Software*, 19:25–29.

Connolly, T. M., Stansfield, M., and Hainey, T. (2007). An application of games-based learning within software engineering. *British Journal of Educational Technology*, 38(3):416–428.

Conradi, R. and Dingsrøyr, T. (2000). Software Experience Bases: A Consolidated Evaluation and Status Report. In *Proceedings of the Second International Conference on Product Focused Software Process Improvement*, PROFES'00, pages 391–406, Oulu, Finland.

Coppit, D. and Haddox-Schatz, J. M. (2005). Large Team Projects in Software Engineering Courses. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '05, pages 137–141, St. Louis, Missouri, USA. ACM.

Crookall, D. and Saunders, D. (1989). Towards an integration of communication and simulation. In *Crookall, D., and Saunders, D., editors, Communication and Simulation: From Two Fields to One Theme*, pages 3–32. Multilingual Matters Ltd.

Curtis, B., Kellner, M. I., and Over, J. (1992). Process modeling. *Communications of the ACM*, 35(9):75–90.

Curtis, B., Krasner, H., and Iscoe, N. (1988). A Field Study of the Software Design Process for Large Systems. *Communications of the ACM*, 31(11):1268–1287.

Dahl, O.-J. and Nygaard, K. (2002). Software Pioneers. chapter Class and subclass declarations, pages 91–107. Springer-Verlag New York, Inc.

Damian, D. and Chisan, J. (2006). An Empirical Study of the Complex Relationships between Requirements Engineering Processes and Other Processes that Lead to Payoffs in Productivity, Quality, and Risk Management. *IEEE Transactions on Software Engineering*, 32(7):433–453.

Damian, D., Zowghi, D., Vaidyanathasamy, L., and Pal, Y. (2002). An Industrial Experience in Process Improvement: An Early Assessment at the Australian Center for Unisys Software. In *Proceedings of the 2002 International Symposium on Empirical Software Engineering*, pages 111–123, Nara, Japan. IEEE Computer Society.

Dangle, K., Larsen, P., Shaw, M., and Zelkowitz, M. (2005). Software Process Improvement in Small Organizations: A Case Study. *IEEE Software*, 22(6):68–75.

Dantas, A. R., Barros, M. d. O., and Werner, C. M. L. (2004a). Treinamento Experimental com Jogos de Simulação para Gerentes de Projeto de Software. In *Simpósio Brasileiro de Engenharia de Software*, SBES 2004, pages 23–38, Brasília, Brasil.

Dantas, A. R., Barros, M. O., and Werner, C. M. L. (2004b). A Simulation-Based Game for Project Management Experiential Learning. In *Proceedings of 16th the International Conference on Software Engineering and Knowledge Engineering*, SEKE 2004, pages 19–24, Banff, Anbert, Canada.

Davis, A. M. (1993). *Software Requirements: Objects, Functions, and States*. Englewood Cliffs, NJ: Prentice-Hall.

Dawson, R. (2000). Twenty Dirty Tricks to Train Software Engineers. In *Proceedings of the 22nd International Conference on Software Engineering*, ICSE '00, pages 209–218, Limerick, Ireland. ACM.

de Souza, C. S. (2005). *The Semiotic Engineering of Human-Computer Interaction*. The MIT Press.

de Souza, C. S., Leitão, C. F., Prates, R. O., Amélia Bim, S., and da Silva, E. J. (2010). Can inspection methods generate valid new knowledge in HCI? The case of semiotic inspection. *International Journal of Human-Computer Studies*, 68:22–40.

de Souza, C. S., Leitão, C. F., Prates, R. O., and da Silva, E. J. (2006). The Semiotic Inspection Method. In *Proceedings of the VII Brazilian Symposium on Human Factors in Computing Systems*, IHC '06, pages 148–157, Natal, Brazil. ACM.

de Souza, C. S. and Leitão, C. F. (2009). Semiotic Engineering Methods for Scientific Research in HCI. *Synthesis Lectures on Human-Centered Informatics*, 2(1):1–122.

DeMarco, T. and Lister, T. (1999). *Peopleware: Productive Projects and Teams*. Dorset House.

Dempsey, J. V., Haynes, L. L., Lucassen, B. A., and Casey, M. S. (2002). Forty Simple Computer Games and What They Could Mean to Educators. *Simulation & Gaming*, 33(2):157–168.

Desurvire, H., Caplan, M., and Toth, J. A. (2004). Using heuristics to evaluate the playability of games. In *CHI '04 Extended Abstracts on Human Factors in Computing Systems*, CHI '04, pages 1509–1512, Vienna, Austria. ACM.

Dickmann, C., Klein, H., Birkhölzer, T., Fietz, W., Vaupel, J., and Meyer, L. (2007). Deriving a Valid Process Simulation from Real World Experiences. In *Proceedings of the 2007 International Conference on Software Process*, ICSP'07, pages 272–282, Minneapolis, MN, USA. Springer-Verlag.

Dijkstra, E. (1969). Notes on Structured Programming. Technical report EWD 249, Eindhoven Technical University.

Dingsøyr, T., Hanssen, G. K., Dybå, T., Anker, G., and Nygaard, J. (2006). Developing Software with Scrum in a Small Cross-Organizational Project. In *Richardson, I., Runeson, P., and Messnarz, R., editors, European Systems & Software Process Improvement and Innovation*, volume 4257 of EuroSPI'06, pages 5–15, Joensuu, Finland. Springer Berlin / Heidelberg.

Donzelli, P. and Iazeolla, G. (2001). Hybrid Simulation Modelling of the Software Process. *Journal of Systems and Software*, 59(3):227–235.

Drappa, A. and Ludewig, J. (2000). Simulation in Software Engineering Training. In *Proceedings of the 22nd International Conference on Software Engineering*, ICSE'00, pages 199–208, Limerick, Ireland.

Dybå, T. and Dingsøyr, T. (2008). Strength of Evidence in Systematic Reviews in Software Engineering. In *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM'08, pages 178–187, Kaiserslautern, Germany. ACM.

Dybå, T. (2005). An Empirical Investigation of the Key Factors for Success in Software Process Improvement. *IEEE Transactions on Software Engineering*, 31(5):410–424.

Dybå, T. and Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9-10):833–859.

Ebert, C., Liedtke, T., and Baisch, E. (1999). Improving reliability of large software systems. *Annals of Software Engineering*, 8:3–51.

Ebert, C., Parro, C. H., Suttels, R., and Kolarczyk, H. (2001). Improving Validation Activities in a Global Software Development. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE'01, pages 545–554, Toronto, Ontario, Canada. IEEE Computer Society.

Ebner, G. and Kaindl, H. (2002). Tracing all Around in Reengineering. *IEEE Software*, 19(3):70–77.

Elliott, M., Dawson, R., and Edwards, J. (2009). An evolutionary cultural-change approach to successful software process improvement. *Software Quality Journal*, 17:189–202.

Endres, A. and Rombach, D. (2003). *A Handbook of Software and System Engineering: Empirical Observations, Laws and Theories.* Addison Wesley.

Engineering, S. (1969). Software Engineering. Brussels: NATO Science Committee.

Engström, E., Runeson, P., and Skoglund, M. (2010). A systematic review on regression test selection techniques. *Information and Software Technology*, 52:14–30.

Fagan, M. E. (1986). Advances in software inspections. *IEEE Transactions on Software Engineering*, 12(7):744–751.

Feiler, P. and Humphrey, W. (1992). Software Process Development and Enactment: Concepts and Definitions. Technical Report CMU/SEI-92-TR-004, Software Engineering Institute.

Ferreiro Ferreira, A. I., Santos, G., Cerqueira, R., Montoni, M., Barreto, A., Rocha, A. R., Barreto, A. O. S., and Silva Filho, R. C. (2008). ROI of software process improvement at BL informática: SPIdex is really worth it. *Software Process: Improvement and Practice*, 13(4):311–318.

Frederiksen, H. and Mathiassen, L. (2004). Assessing Improvements of Software Metrics Practices. In *Fitzgerald, B., and Wynn, E., editors, IT Innovation for Adaptability and Competitiveness*, volume 141 of IFIP International Federation for Information Processing, pages 93–115. Springer Boston.

Freimut, B., Denger, C., and Ketterer, M. (2005). An Industrial Case Study of Implementing and Validating Defect Classification for Process Improvement and Quality Management. In *11th IEEE International Symposium on Software Metrics*, pages 10–19, Como, Italy.

Fripp, J. (1993). *Learning through Simulations: A Guide to the Design and Use of Simulations in Business and Education.* McGraw-Hill.

Galinac, T. and Car, v. (2007). Software Verification Process Improvement Proposal Using Six Sigma. In *Münch, J., and Abrahamsson, P., editors, Proceedings of the 8th International Conference on Product Focused Software Process Improvement*, volume 4589 of PROFES'07, pages 51–64, Riga, Latvia. Springer Berlin / Heidelberg.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Longman Publishing Co., Inc., Boston, Massachusetts, USA.

Garris, R., Ahlers, R., and Driskell, J. E. (2002). Games, Motivation, and Learning: A Research and Practice Model. *Simulation & Gaming*, 33(4):441–467.

Glass, R. L. (1982). *Requirements Tracing.* In Modern Programming Practices. Englewood Cliffs, NJ: Prentice-Hall.

Glass, R. L. (1992). *Building Quality Software.* Prentice-Hall, Inc.

Glass, R. L. (1998). *Software Runaways: Lessons Learned from Massive Software Project Failures.* NL: Prentice Hall.

Glass, R. L. (1999). The Realities of Software Technology Payoffs. *Communications of the ACM*, 42:74–79.

Glass, R. L. (2001). Frequently Forgotten Fundamental Facts about Software Engineering. *IEEE Software*, 18(3):110–112.

Glass, R. L. (2002). *Software Engineering: Facts and Fallacies.* Addison-Wesley Longman Publishing Co., Inc., Boston, Massachusetts, USA.

Glass, R. L. (2003). *Facts and Fallacies of Software Engineering.* Addison-Wesley.

Gou, L., Wang, Q., Yuan, J., Yang, Y., Li, M., and Jiang, N. (2008). Quantitatively Managing Defects for Iterative Projects: An Industrial Experience Report in China. In *Wang, Q., Pfahl, D., Raffo, D., editors, Proceedings of the 2008 International Conference on Software Process*, volume 5007 of ICSP'08, pages 369–380, Leipzig, Germany. Springer Berlin / Heidelberg.

Gredler, M. E. (2004). Games and Simulations and their Relationships to Learning. In *Jonassen, D. H., editor, Handbook of Research on Educational Communications and Technology, Chapter 21*, pages 571–581. Mahwah, NJ: Lawrence Erlbaum Associates.

Gresse von Wangenheim, C., Thiry, M., and Kochanski, D. (2009). Empirical evaluation of an educational game on software measurement. *Empirical Software Engineering*, 14:418–452.

Guerrero, F. and Eterovic, Y. (2004). Adopting the SW-CMM in a small IT organization. *IEEE Software*, 21(4):29–35.

Hansen, B., Rose, J., and Tjørnehøj, G. (2004). Prescription, description, reflection: the shape of the software process improvement field. *International Journal of Information Management*, 24(6):457–472.

Harjumaa, L. (2005). A pattern approach to software inspection process improvement. *Software Process: Improvement and Practice*, 10(4):455–465.

Harjumaa, L., Tervonen, I., and Vuorio, P. (2004). Improving Software Inspection Process with Patterns. In *Proceedings of the Fourth International Conference on Quality Software*, QSIC'04, pages 118–125, Braunschweig, Germany. IEEE Computer Society.

Harter, D., Kemerer, C., and Slaughter, S. (2011). Does Software Process Improvement Reduce the Severity of Defects? A Longitudinal Field Study. *IEEE Transactions on Software Engineering*, page 1.

Hetzel, W. C. (1976). *An experimental analysis of program verification methods*. PhD thesis, The University of North Carolina at Chapel Hill.

Higgins, S., de Laat, M., Gieles, P., and Geurts, E. (2002). Managing product requirements for medical IT products. In *IEEE Joint International Conference on Requirements Engineering*, pages 341–349, Essen, Germany.

Hoffmann, M., Kuhn, N., Weber, M., and Bittner, M. (2004). Requirements for requirements management tools. In *Proceedings of the 12th IEEE International Requirements Engineering Conference*, pages 301–308, Kyoto, Japan.

Hollenbach, C. and Smith, D. (2002). A portrait of a CMMI level 4 effort. *Systems Engineering*, 5(1):52–61.

Holmberg, L., Nilsson, A., Holmström Olsson, H., and Börjesson Sandberg, A. (2009). Appreciative inquiry in software process improvement. *Software Process: Improvement and Practice*, 14(2):107–125.

Höst, M. and Runeson, P. (2007). Checklists for Software Engineering Case Study Research. In *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, ESEM'07, pages 479–481, Madrid, Spain. IEEE Computer Society.

Houston, D. (2006). An experience in facilitating process improvement with an integration problem reporting process simulation. *Software Process: Improvement and Practice*, 11(4):361–371.

Huffman Hayes, J. (2002). Energizing Software Engineering Education through Real-World Projects as Experimental Studies. In *Proceedings of the 15th Conference*

on Software Engineering Education and Training, CSEE&T 2002, pages 192–206, Covington, Kentucky, USA.

Humphrey, W. S. (1989). *Managing the Software Process*. Addison-Wesley Longman Publishing Co., Inc., Boston, Massachusetts, USA.

Humphrey, W. S. (1996). Using A Defined and Measured Personal Software Process. *IEEE Software*, 13(3):77–88.

Hunicke, R. (2005). The Case for Dynamic Difficulty Adjustment in Games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACE '05, pages 429–433, Valencia, Spain. ACM.

Ikeda, K. and Akamatsu, Y. (2004). Starting SPI from Software Configuration Management: A Fast Approach for an Organization to Realize the Benefits of SPI. In *Bomarius, F., and Iida, H., editors, Proceedings of the 5th International Conference on Product Focused Software Process Improvement*, volume 3009 of PROFES'04, pages 92–104, Kansai Science City, Japan. Springer Berlin / Heidelberg.

ISO/IEC-TR-12207 (2008). Systems and software engineering - Software life cycle processes. International Standard 12207, International Organization for Standardization and International Electrotechnical Commission.

ISO/IEC-TR-15504 (2008). ISO/IEC 15504 family: Information technology - process assessment (15504-1 to 15504-7). International Standard 15504-1, 15504-2, 15504-3, 15504-4, 15504-5, 15504-6, 15504-7, International Organization for Standardization and International Electrotechnical Commission.

ISO/IEC-TR-9126 (2001). Software Quality Characteristics. International Standard 9126, International Organization for Standardization and International Electrotechnical Commission.

Iversen, J. and Mathiassen, L. (2003). Cultivation and engineering of a software metrics program. *Information Systems Journal*, 13(1):3–19.

Jain, A. and Boehm, B. (2006). SimVBSE: Developing a Game for Value-Based Software Engineering. In *Proceedings of the 19th Conference on Software Engineering Education and Training*, pages 103 –114, Oahu, Hawaii.

Jester, M., Krasner, H., and Perry, D. (2006). Software Process Definition Improvement: An Industry Report. In *Proceedings of the 32nd Euromicro Conference on*

Software Engineering and Advanced Applications, SEAA'06, pages 206–215, Cavtat/Dubrovnik, Croatia.

Johansen, J. and Pries-Heje, J. (2007). Success with improvement - requires the right roles to be enacted - in symbiosis. *Software Process: Improvement and Practice*, 12(6):529–539.

Jones, C. (1996). *Applied Software Measurement: Assuring Productivity and Quality*. McGraw-Hill.

Kääriäinen, J., Koskela, J., Abrahamsson, P., and Takalo, J. (2004). Improving Requirements Management in Extreme Programming with Tool Support - An Improvement Attempt that Failed. In *Proceedings of the 30th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA'04, pages 342–351, Rennes, France.

Karlström, D. and Runeson, P. (2005). Combining Agile Methods with Stage-Gate Project Management. *IEEE Software*, 22(3):43–49.

Karlström, D., Runeson, P., and Nordén, S. (2005). A minimal test practice framework for emerging software organizations. *Software Testing, Verification and Reliability*, 15(3):145–166.

Kauppinen, M., Vartiainen, M., Kontio, J., Kujala, S., and Sulonen, R. (2004). Implementing requirements engineering processes throughout organizations: success factors and challenges. *Information and Software Technology*, 46(14):937–953.

Kautz, K., Hansen, H. W., and Thaysen, K. (2000). Applying and adjusting a software process improvement model in practice: the use of the IDEAL model in a small software enterprise. In *Proceedings of the 22nd International Conference on Software Engineering*, ICSE'00, pages 626–633, Limerick, Ireland. ACM.

Kellner, M. I. and Madachy, Raymond J.and Raffo, D. M. (1999). Software process simulation modeling: Why? What? How? *Journal of Systems and Software*, 46(2-3):91–105.

Kenni, G. (2000). The evolution of quality processes at Tata Consultancy Services. *IEEE Software*, 17(4):79–88.

Kikuchi, N. and Kikuno, T. (2001). Improving the Testing Process by Program Static Analysis. In *Eighth Asia-Pacific Software Engineering Conference*, APSEC 2001, pages 195–201, Macau, China.

Kilpi, T. (2000). Managing the Software Process in the Middle of Rapid Growth: A Metrics Based Experiment Report from Nokia. In *Wangler, B., and Bergman, L., editors, Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, volume 1789 of CAiSE'00, pages 498–508, Stockholm, Sweden. Springer Berlin / Heidelberg.

Kitagawa, G. and Gersch, W. (1996). *Smoothness Priors Analysis of Time Series*. Lecture Notes in Statistics. Springer Springer-Verlag.

Kitchenham, B. (2004). Procedures for Performing Systematic Reviews. Technical Report TR/SE-0401, Software Engineering Group, Department of Computer Science, Keele University.

Kitchenham, B. and Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering, Version 2.3. Technical Report EBSE 2007-01, Software Engineering Group, School of Computer Science and Mathematics, Keele University and Department of Computer Science University of Durham.

Kitchenham, B., Pickard, L., and Pfleeger, S. L. (1995). Case Studies for Method and Tool Evaluation. *IEEE Software*, 12:52–62.

Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., Emam, K. E., and Rosenberg, J. (2002). Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Transactions on Software Engineering*, 28:721–734.

Knauss, E., Schneider, K., and Stapel, K. (2008). A Game for Taking Requirements Engineering More Seriously. In *First International Workshop on Multimedia Requirements Engineering*, pages 22–26, Barcelona, Catalunya, Spain. IEEE Computer Society.

Krikhaar, R. and Mermans, M. (2007). Software Development Improvement with SFIM. In *Münch, J., and Abrahamsson, P., editors, Proceedings of the 8th International Conference on Product Focused Software Process Improvement*, volume 4589 of PROFES'07, pages 65–80, Riga, Latvia. Springer Berlin / Heidelberg.

Kukkanen, J., Vakevainen, K., Kauppinen, M., and Uusitalo, E. (2009). Applying a Systematic Approach to Link Requirements and Testing: A Case Study. In *16th Asia-Pacific Software Engineering Conference*, APSEC'09, pages 482–488, Penang, Malaysia.

Kurniawati, F. and Jeffery, R. (2006). The use and effects of an electronic process guide and experience repository: a longitudinal study. *Information and Software Technology*, 48(7):566–577.

Laredo, J. and Ranjan, R. (2008). Continuous Improvement through Iterative Development in a Multi-Geography. In *IEEE International Conference on Global Software Engineering*, ICGSE 2008, pages 232–236, Bangalore, India.

Larndorfer, S., Ramler, R., and Buchwiser, C. (2009). Experiences and Results from Establishing a Software Cockpit at BMD Systemhaus. In *Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA'09, pages 188–194, Patras, Greece.

Lauesen, S. and Vinter, O. (2001). Preventing Requirement Defects: An Experiment in Process Improvement. *Requirements Engineering*, 6:37–50.

Lethbridge, T. C. (1998). A Survey of the Relevance of Computer Science and Software Engineering Education. In *Proceedings of the 11th Conference on Software Engineering Education and Training*, pages 56–66, Atlanta, Georgia, USA. IEEE Computer Society.

Leung, H. K. N. and Yuen, T. C. F. (2001). A process framework for small projects. *Software Process: Improvement and Practice*, 6(2):67–83.

Levary, R. R. and Lin, C. Y. (1991). Modelling the software development process using an expert simulation system having fuzzy logic. *Software: Practice and Experience*, 21(2):133–148.

Li, M. (2007). TRISO-Model: A New Approach to Integrated Software Process Assessment and Improvement. *Software Process: Improvement and Practice*, 12(5):387–398.

Li, M., Xiaoyuan, H., and Sontakke, A. (2006). Defect Prevention: A General Framework and Its Application. In *Sixth International Conference on Quality Software*, QSIC 2006, pages 281–286, Beijing, China.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of Psychology*, 22.

Lingard, R. and Berry, E. (2002). Teaching teamwork skills in software engineering based on an understanding of factors affecting group performance. In *32nd Annual*

Frontiers in Education, volume 3, pages S3G–1– S3G–6 vol.3, Boston, Massachusetts, USA.

Ludewig, J., Bassler, T., Deininger, M., Schneider, K., and Schwille, J. (1992). SESAM-simulating software projects. In *Proceedings of the Fourth International Conference on Software Engineering and Knowledge Engineering*, pages 608–615, Capri, Italy.

Ludi, S. and Collofello, J. (2001). An Analysis of the Gap Between the Knowledge and Skills Learned in Academic Software Engineering Course Projects and those Required in Real Projects. In *Proceedings of the 31st ASEE/IEEE Frontiers in Education Conference*, FIE'01, pages T2D–8–T2D–11vol.1, Reno, Nevada. IEEE Computer Society.

Madachy, R. J. (2008). *Software Process Dynamics*. Wiley-IEEE Press.

Malheiros, V., Paim, F. R., and Mendonça, M. (2009). Continuous process improvement at a large software organization. *Software Process: Improvement and Practice*, 14(2):65–83.

Malone, T. W. (1980). What Makes Things Fun to Learn? Heuristics for Designing Instructional Computer Games. In *Proceedings of the 3rd ACM SIGSMALL Symposium and the First SIGPC Symposium on Small Systems*, SIGSMALL '80, pages 162–169, Palo Alto, California, United States. ACM.

Mandl-Striegnitz, P. (2001). How to successfully use software project simulation for educating software project managers. In *Proceedings of the 31st ASEE/IEEE Frontiers in Education Conference*, volume 1 of FIE '01, pages T2D–19–24 vol.1, Reno, Nevada.

Martin, A. (2000). The Design and Evolution of a Simulation/Game for Teaching Information Systems Development. *Simulation & Gaming*, 31:445–463.

Martin, R. H. and Raffo, D. (2000). A model of the software development process using both continuous and discrete models. *Software Process: Improvement and Practice*, 5(2-3):147–157.

Mays, R. G., Jones, C. L., Holloway, G. J., and Studinski, D. P. (1990). Experiences with Defect Prevention. *IBM Systems Journal*, 29(1):4–32.

McCaffery, F., Burton, J., and Richardson, I. (2009). Improving Software Risk Management in a Medical Device Company. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE'09, pages 152–162, Vancouver, British Columbia, Canada.

McFeeley, B. (1996). *IDEAL: A User's Guide for Software Process Improvement*. Software Engineering Institute, Pennsylvania, USA.

McGarry, F. and Decker, B. (2002). Attaining Level 5 in CMM Process Maturity. *IEEE Software*, 19(6):87–96.

McGraw, G. (2006). *Software Security*. Addison-Wesley.

McMillan, W. W. and Rajaprabhakaran, S. (1999). What Leading Practitioners Say Should Be Emphasized in Students' Software Engineering Projects. In *Proceedings of the 12th Conference on Software Engineering Education and Training*, CSEET'99, pages 177–185, New Orleans, Louisiana. IEEE Computer Society.

Metzker, E. and Offergeld, M. (2001). An Interdisciplinary Approach for Successfully Integrating Human-Centered Design Methods into Development Processes Practiced by Industrial Software Development Organizations. In *Little, M., and Nigay, L., editors, Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction*, volume 2254 of EHCI '01, pages 19–33, Toronto, Canada. Springer Berlin / Heidelberg.

Moe, N., Dingsrøyr, T., Nilsen, K., and Villmones, N. (2005). Project Web and Electronic Process Guide as Software Process Improvement. In *Richardson, I., Abrahamsson, P., and Messnarz, R., editors, European Systems & Software Process Improvement and Innovation*, volume 3792 of EuroSPI'05, pages 175–186, Budapest, Hungary. Springer Berlin / Heidelberg.

Momoh, J. and Ruhe, G. (2006). Release planning process improvement - an industrial case study. *Software Process: Improvement and Practice*, 11(3):295–307.

Moreno, A. M., Sanchez-Segura, M.-I., Medina-Dominguez, F., and Carvajal, L. (2012). Balancing software engineering education and industrial needs. *Journal of Systems and Software*, 85(7):1607–1620.

Moreno-Ger, P., Burgos, D., Martínez-Ortiz, I., Sierra, J. L., and Fernández-Manjón, B. (2008). Educational game design for online education. *Journal Computers in Human Behavior*, 24:2530–2540.

Morgan, G. (1996). *Images of Organization*. SAGE Publications, California.

Morisio, M. (2000). Applying the PSP in industry. *IEEE Software*, 17(6):90–95.

Moritz, E. (2009). Case study: How analysis of customer found defects can be used by system test to improve quality. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE'09, pages 123–129, Vancouver, British Columbia, Canada.

Mory, E. H. (2003). Feedback Research Revisited. In *Jonassen, D. H., editor, Handbook of Research on Educational Communications and Technology*, pages 745–783.

Motoyama, T. (2006). Improving Software Development through Three Stages. *IEEE Software*, 23(5):81–87.

Müller, S. D., Mathiassen, L., and Balshøj, H. H. (2010). Software Process Improvement as organizational change: A metaphorical analysis of the literature. *Journal of Systems and Software*, 83:2128–2146.

Murugappan, M. and Keeni, G. (2003). Blending CMM and Six Sigma to meet business goals. *IEEE Software*, 20(2):42–48.

Napier, N. P., Mathiassen, L., and Johnson, R. D. (2009). Combining Perceptions and Prescriptions in Requirements Engineering Process Assessment: An Industrial Case Study. *IEEE Transactions on Software Engineering*, 35(5):593–606.

NASA (1990). *Manager's Handbook for Software Development.* NASA-Goddard.

Navarro, E. and van der Hoek, A. (2009). Multi-site evaluation of SimSE. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, SIGCSE '09, pages 326–330, Chattanooga, TN, USA. ACM.

Navarro, E. O. (2006). *SimSE: A Software Engineering Simulation Environment for Software Process Education.* PhD thesis, Donald Bren School of Information and Computer Sciences, University of California, Irvine.

Navarro, E. O. and van der Hoek, A. (2002). Towards game-based simulation as a method of teaching software engineering. In *Proceedings of the 32nd ASEE/IEEE Frontiers in Education Conference*, volume 3, pages S2G–13 vol.3, Boston, Massachusetts, USA.

Navarro, E. O. and van der Hoek, A. (2005a). Design and Evaluation of an Educational Software Process Simulation Environment and Associated Model. In *Proceedings of the 18th Conference on Software Engineering Education and Training*, pages 25–32, Ottawa, Canada. IEEE Computer Society.

Navarro, E. O. and van der Hoek, A. (2005b). Software process modeling for an educational software engineering simulation game. *Software Process: Improvement and Practice*, 10(3):311–325.

Navarro, E. O. and van der Hoek, A. (2007). Comprehensive Evaluation of an Educational Software Engineering Simulation Environment. In *Proceedings of the 20th Conference on Software Engineering Education and Training*, pages 195–202, Dublin, Ireland.

Nelson, K., Buche, M., and Nelson, H. (2001). Structural change and change advocacy: a study in becoming a software engineering organization. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, page 9 pp., Maui, Hawaii, USA.

Niazi, M. (2006). Software Process Improvement: A Road to Success. In *Münch, J., and Vierimaa, M., editors, Product-Focused Software Process Improvement*, volume 4034 of Lecture Notes in Computer Science, pages 395–401, Amsterdam, The Netherlands. Springer Berlin / Heidelberg.

Niazi, M., Wilson, D., and Zowghi, D. (2005). A maturity model for the implementation of software process improvement: an empirical study. *Journal of Systems and Software*, 74(2):155–172.

Niazi, M., Wilson, D., and Zowghi, D. (2006). Critical success factors for software process improvement implementation: an empirical study. *Software Process: Improvement and Practice*, 11(2):193–211.

Nikitina, N. and Kajko-Mattsson, M. (2009). Historical Perspective of Two Process Transitions. In *Proceedings of the Fourth International Conference on Software Engineering Advances*, ICSEA'09, pages 289–298, Porto, Portugal.

Nikula, U. and Sajaniemi, J. (2005). Tackling the Complexity of Requirements Engineering Process Improvement by Partitioning the Improvement Task. In *Proceedings of the 2005 Australian Conference on Software Engineering*, ASWEC'05, pages 48–57, Brisbane, Australia.

Oates, B. J. (2006). *Research Information Systems and Computing*. SAGE Publications.

O'Hara, F. (2000). European Experiences with Software Process Improvement. In *Proceedings of the 22nd International Conference on Software Engineering*, ICSE'00, pages 635–640, Limerick, Ireland. ACM.

Olsen, A. L. (2008). A service learning project for a software engineering course. *Journal of Computing Sciences in Colleges*, 24:130–136.

O'Regan, G. (2011). *Introduction to Software Process Improvement*. Springer, London.

Otoya, S. and Cerpa, N. (1999). An experience: A small software company attempting to improve its process. In *Proceedings of the Ninth International Workshop Software Technology and Engineering Practice*, STEP '99, pages 153–160, Pittsburgh, PA, USA.

Paula Filho, W. P. (2006). A Software Process for Time-constrained Course Projects. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE'06, pages 707–710, Shanghai, China. ACM.

Paula Filho, W. P. (2007). Using Model-Driven Development in Time-Constrained Course Projects. In *Proceedings of the 20th Conference on Software Engineering Education and Training*, pages 133–140, Dublin, Ireland. IEEE Computer Society.

Paula Filho, W. P. (2009). Praxis 3.0. `http://www.dcc.ufmg.br/~wilson/praxis/`. "[Online; accessed 10 March 2009]".

Paulk, M. C., Curtis, B., Chrissis, M. B., and Weber, C. V. (1993). Capability Maturity Model, Version 1.1. Technical report CMU/SEI-93-TR, Software Engineering Institute.

Paulk, M. C., Weber, C. V., Curtis, B., and Chrissis, M. B. (1995). *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley Professional.

Pearson, R. K. (1999). *Discrete-Time Dynamic Models*. Oxford.

Peirce, C. S. (1992). *The Essential Peirce (Vols. I and II)*. Indiana University Press, Bloomington, Edited by Nathan Houser and Christian Kloesel edition.

Peixoto, D. C. C., Batista, V. A., Resende, R. F., and Pádua, C. I. P. S. (2010a). How to Welcome Software Process Improvement and Avoid Resistance to Change. In *Proceeding of the International Conference on Software Process*, volume 6195 of ICSP'10, pages 138–149, Paderborn, Germany. Springer Berlin / Heidelberg.

Peixoto, D. C. C., Batista, V. A., Resende, R. F., and Pádua, C. I. P. S. (2010b). Learning from Students' Mistakes in Software Engineering courses. In *Proceedings of the 40th ASEE/IEEE Frontiers in Education Conference*, FIE'10, pages F1J–1–F1J–6, Northern Virginia / Washington, D.C.

Peixoto, D. C. C., Possa, R. M., Resende, R. F., and Pádua, C. I. P. S. (2012a). Challenges and Issues in the Development of a Software Engineering Simulation Game. In *Proceedings of the 42nd ASEE/IEEE Frontiers in Education Conference*, FIE'12, pages – To appear, Seattle, Washington.

Peixoto, D. C. C., Possa, R. M., Resende, R. F., and Pádua, C. I. P. S. (2012b). FASENG: A Framework for Development of Software Engineering Simulation Games. In *Proceedings of the 42nd ASEE/IEEE Frontiers in Education Conference*, FIE'12, pages – To appear, Seattle, Washington.

Peixoto, D. C. C., Possas, R. M., Resende, R. F., and Pádua, C. I. P. S. (2011). An Overview of the Main Design Characteristics of Simulation Games in Software Engineering Education. In *Proceedings of the 24th Conference on Software Engineering Education and Training*, CSEET'11, pages 101–110, Waikiki, Honolulu, Hawaii. IEEE.

Peixoto, D. C. C., Prates, R. O., and Resende, R. F. (2010c). Semiotic Inspection Method in the Context of Educational Simulation Games. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1207–1212, Sierre, Switzerland. ACM.

Penker, M. and Eriksson, H.-E. (2000). *Business Modeling With UML: Business Patterns at Work*. John Wiley.

Peters, V., Vissers, G., and Heijne, G. (1998). The Validity of Games. *Simulation & Gaming*, 29(1):20–30.

Peters, V. A. M. and Vissers, G. A. N. (2004). A Simple Classification Model for Debriefing Simulation Games. *Simulation & Gaming*, 35:70–84.

Petersen, K. and Wohlin, C. (2009). Context in Industrial Software Engineering Research. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ESEM'09, pages 401–404, Lake Buena Vista, Florida, USA. IEEE Computer Society.

Pfahl, D. (2001). *An Integrated Approach to Simulation-Based Learning in Support of Strategic and Project Management in Software Organizations*. PhD thesis, Fraunhofer-Institute for Experimental Software Engineering - Kaiserslautern.

Pfahl, D., Laitenberger, O., Ruhe, G., Dorsch, J., and Krivobokova, T. (2004). Evaluating the learning effectiveness of using simulations in software project management

education: results from a twice replicated experiment. *Information and Software Technology*, 46(2):127–147.

Pinheiro, C., Maurer, F., and Sillito, J. (2009). Improving Quality, One Process Change at a Time. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE'09, pages 81–90, Vancouver, British Columbia, Canada.

Pino, F., García, F., and Piattini, M. (2008). Software process improvement in small and medium software enterprises: a systematic review. *Software Quality Journal*, 16:237–261.

Pino, F. J., García, F., and Piattini, M. (2009). An Integrated Framework to Guide Software Process Improvement in Small Organizations. In *European Systems & Software Process Improvement and Innovation*, volume 42 of EuroSPI 2009, pages 213–224, Madrid, Spain. Springer Berlin / Heidelberg.

Pitterman, B. (2000). Telcordia Technologies: The Journey to High Maturity. *IEEE Software*, 17(4):89–96.

Possa, R. M. (2011). Um Estudo sobre os Requisitos de Jogos de Simulação usados no Ensino de Engenharia de Software. Master's thesis, Universidade Federal de Minas Gerais, Brazil.

Prates, R. O., de Souza, C. S., and Barbosa, S. D. J. (2000). Methods and Tools: A Method for Evaluating the Communicability of User Interfaces. *Interactions*, 7:31–38.

Pressman, R. S. (1992). Software Project Management: Q and A. *Cutter IT Journal*.

Prikladnicki, R. and Audy, J. L. N. (2010). Process models in the practice of distributed software development: A systematic review of the literature. *Information and Software Technology*, 52:779–791.

Raffo, D. M., Vandeville, J. V., and Martin, R. H. (1999). Software Process Simulation to Achieve Higher CMM Levels. *Journal of Systems and Software*, 46(2-3):163–172.

Raffo, D. M. and Wakeland, W. (2008). Moving Up the CMMI Capability and Maturity Levels Using Simulation. Technical Report CMU/SEI-2008-TR-002, Software Engineering Institute.

Rainer, A. and Hall, T. (2001). An analysis of some 'core studies' of software process improvement. *Software Process: Improvement and Practice*, 6(4):169–187.

Rainer, A. and Hall, T. (2002). Key success factors for implementing software process improvement: a maturity-based analysis. *Journal of Systems and Software*, 62(2):71–84.

Rainer, A. and Hall, T. (2003). A quantitative and qualitative analysis of factors affecting software processes. *Journal of Systems and Software*, 66(1):7–21.

Rautiainen, K., Vuornos, L., and Lassenius, C. (2003). An experience in combining flexibility and control in a small company's software product development process. In *International Symposium on Empirical Software Engineering*, ISESE 2003, pages 28–37, Rome, Italy.

Redzic, C. and Baik, J. (2006). Six Sigma Approach in Software Quality Improvement. In *Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications*, pages 396–406, Seattle, Washington, USA.

Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14:131–164.

Russ, R., Sperling, D., Rometsch, F., and Louis, P. (2008). Applying Six Sigma in the Field of Software Engineering. In *Dumke, R., Braungarten, R., Büren, G., Abran, A., and Cuadrado-Gallego, J., editors, Proceedings of the International Conferences on Software Process and Product Measurement*, volume 5338 of IWSM/Metrikon/Mensura '08, pages 36–47, Nara, Japan. Springer Berlin / Heidelberg.

Salo, O. and Abrahamsson, P. (2007). An iterative improvement process for agile software development. *Software Process: Improvement and Practice*, 12(1):81–100.

Sampaio, A., Albuquerque, C., Vasconcelos, J., Cruz, L., Figueiredo, L., and Cavalcante, S. (2005). Software Test Program: A Software Residency Experience. In *Proceeding of the 27th International Conference on Software Engineering*, ICSE'05, pages 611–612, St. Louis, Missouri, USA.

Sawyer, P., Sommerville, I., and Viller, S. (1997). Requirements Process Improvement through the Phased Introduction of Good Practice. Technical Report CSEG/17/1997, Cooperative Systems Engineering Group.

Schmid, K., Becker-Kornstaedt, U., Kanuber, P., and Bernauer, F. (2000). Introducing a Software Modeling Concept in a Medium-Sized Company. In *Proceedings of the 22nd International Conference on Software Engineering*, ICSE'00, pages 558–567, Limerick, Ireland.

Schneider, K. (2000). Active Probes Synergy in Experience-Based Process Improvement. In *Bomarius, F., and Oivo, M., editors, Proceedings of the Second International Conference on Product Focused Software Process Improvement*, volume 1840 of PROFES'00, pages 433–619, Oulu, Finland. Springer Berlin / Heidelberg.

Scott, L., Carvalho, L., Jeffery, R., D'Ambra, J., and Becker-Kornstaedt, U. (2002). Understanding the use of an electronic process guide. *Information and Software Technology*, 44(10):601–616.

SE2004 (2004). Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. The Joint Task Force on Computing Curricula IEEE Computer Society, Association for Computing Machinery. `http://sites.computer.org/ccse/SE2004Volume.pdf`. "[Online; accessed 12 September 2011]".

Sharp, H. and Hall, P. (2000). An Interactive Multimedia Software House Simulation for Postgraduate Software Engineers. In *Proceedings of the 22nd International Conference on Software Engineering*, pages 688–691, Limerick, Ireland.

Shaw, K. and Dermoudy, J. (2005). Engendering an Empathy for Software Engineering. In *Proceedings of the 7th Australasian conference on Computing education - Volume 42*, ACE '05, pages 135–144, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Shull, F., Singer, J., and Sjøberg, D. I. K. (2008). *Guide to Advanced Empirical Software Engineering*. Springer-Verlag, London.

Sommerville, I. (2011). *Software Engineering (9th ed)*. Addison-Wesley Longman Publishing Co., Inc., Boston, Massachusetts, USA.

Sommerville, I. and Ransom, J. (2005). An Empirical Study of Industrial Requirements Engineering Process Assessment and Improvement. *ACM Transactions on Software Engineering and Methodology*, 14:85–117.

Staples, M. and Niazi, M. (2008). Systematic review of organizational motivations for adopting CMM-based SPI. *Information and Software Technology*, 50(7-8):605–620.

Stlålhane, T. (2006). Implementing an ISO 9001 Certified Process. In *Richardson, I., Runeson, P., and Messnarz, R., editors, European Systems & Software Process Improvement and Innovation*, volume 4257 of EuroSPI'06, pages 16–27, Joensuu, Finland.

Størrle, H. (2001). Describing Process Patterns with UML. In *Ambriola, V., editor, Software Process Technology*, volume 2077 of Lecture Notes in Computer Science, pages 173–181. Springer Berlin / Heidelberg.

Sulayman, M. and Mendes, E. (2009). A Systematic Literature Review of Software Process Improvement in Small and Medium Web Companies. In *Ślęzak, D., Kim, T., Kiumi, A., Jiang, T., Verner, J., and Abrahão, S., editors, Advances in Software Engineering*, volume 59 of Communications in Computer and Information Science, pages 1–8. Springer Berlin / Heidelberg.

Sutherland, J., Ruseng Jakobsen, C., and Johnson, K. (2008). Scrum and CMMI Level 5: The Magic Potion for Code Warriors. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, page 466, Waikoloa, Big Island, Hawaii, USA.

Svinick, M. and Mckeachie, W. J. (2011). *McKeachie's Teaching Tips: Strategies, Research, and Theory for College and University Teachers, 13th Edition*. Wadsworth Cengage Learning.

Taran, G. (2007). Using Games in Software Engineering Education to Teach Risk Management. In *Proceedings of the 20th Conference on Software Engineering Education and Training*, CSEET '07, pages 211–220, Dublin, Ireland.

Tchounikine, P. (2001). *Computer Science and Educational Software Design: A Resource for Multidisciplinary Work in Technology Enhanced Learning*. Springer-Verlag.

Tosun, A., Bener, A., and Turhan, B. (2009). Implementation of a Software Quality Improvement Project in an SME: A Before and After Comparison. In *Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA'09, pages 203–209, Patras, Greece.

Tuffley, A., Grove, B., and McNair, G. (2004). SPICE for Small Organisations. *Software Process: Improvement and Practice*, 9(1):23–31.

Tvedt, J. (1996). *An Extensible Model for Evaluating the Impact of Process Improvements on Software Development Cycle Time*. PhD thesis, Arizona State University.

Unterkalmsteiner, M., Gorschek, T., Islam, A., Cheng, C., Permadi, R., and Feldt, R. (2012). Evaluation and Measurement of Software Process Improvement - A Systematic Literature Review. *IEEE Transactions on Software Engineering*, 38(2):398–424.

Upchurch, R. and Sims-Knight, J. (1999). Reflective essays in software engineering. In *Proceedings of the 29th Annual ASEE/IEEE Frontiers in Education Conference*, volume 3, pages 13A6/13 –13A6/19 vol.3, San Juan, Puerto Rico.

Valtanen, A., Ahonen, J. J., and Savolainen, P. (2009). Improving the Product Documentation Process of a Small Software Company. In *Aalst, W., Mylopoulos, J., Rosemann, M., Shaw, M. J., Szyperski, C., Bomarius, F., Oivo, M., Jaring, P., and Abrahamsson, P., editors, Proceedings of the 10th International Conference on Product Focused Software Process Improvement*, volume 32 of PROFES'09, pages 303–316, Oulu, Finland. Springer Berlin Heidelberg.

van Genuchten, M. (1991). Why is Software Late? An Empirical Study of Reasons for Delay in Software Development. *IEEE Transactions on Software Engineering*, 17:582–590.

van Latum, F. and van Uijtregt, A. (2000). Product Driven Process Improvement PROFES Experiences at Dräger. In *Bomarius, F., and Oivo, M., editors, Proceedings of the Second International Conference on Product Focused Software Process Improvement*, volume 1840 of PROFES'00, pages 7–37, Oulu, Finland. Springer Berlin / Heidelberg.

van Solingen, R. (2004). Measuring the ROI of Software Process Improvement. *IEEE Software*, 21(3):32–38.

van Solingen, R., Kusters, R. J., Trienekens, J. J. M., and van Uijtregt, A. (1999). Product-focused software process improvement (P-SPI): concepts and their application. *Quality and Reliability Engineering International*, 15(6):475–483.

Vega, K. C., Fuks, H., and de Carvalho, G. R. (2009a). Training in Requirements by Collaboration: Branching Stories in Second Life. In *Simpósio Brasilerio de Sistemas Colaborativos*, pages 116–122, Fortaleza, Ceará, Brazil. IEEE Computer Society.

Vega, K. C., Pereira, A., de Carvalho, G. R., Raposo, A., and Fuks, H. (2009b). Prototyping Games for Training and Education in Second Life: Time2Play and TREG. In *Brazilian Symposium on Games and Digital Entertainment*, pages 1–8, Rio de Janeiro, Brazil. IEEE Computer Society.

Vokác, M. and Jensen, O. (2004). Using a Reference Application with Design Patterns to Produce Industrial Software. In *Bomarius, F., and Iida, H., editors, Proceedings of the 5th International Conference on Product Focused Software Process Improvement*,

volume 3009 of PROFES'04, pages 333–347, Kansai Science City, Japan. Springer Berlin / Heidelberg.

von Wangenheim, C. and Shull, F. (2009). To Game or Not to Game? *IEEE Software*, 26(2):92–94.

von Wangenheim, C. G., Weber, S., Hauck, J. C. R., and Trentin, G. (2006). Experiences on establishing software processes in small companies. *Information and Software Technology*, 48(9):890–900.

Walia, G. S. and Carver, J. C. (2009). A systematic literature review to identify and classify software requirement errors. *Information and Software Technology*, 51:1087–1109.

Wang, A. I., Øfsdahl, T., and Mørch-Storstein, O. K. (2008). An Evaluation of a Mobile Game Concept for Lectures. In *Proceeding of the 21st Conference on Software Engineering Education and Training*, pages 197–204, Charleston, South Carolina, USA. IEEE Computer Society.

Weller, E. F. (1993). Lessons from Three Years of Inspection Data. *IEEE Software*, 10(5):38–45.

Westerheim, H. and Hanssen, G. (2005). The Introduction and Use of a Tailored Unified Process - A Case Study. In *Proceedings of the 31st Euromicro Conference on Software Engineering and Advanced Applications*, SEAA'05, pages 196–203, Porto, Portugal.

Williams, B. J. and Carver, J. C. (2010). Characterizing software architecture changes: A systematic review. *Information and Software Technology*, 52:31–51.

Wilson, D. N., Hall, T., and Baddoo, N. (2001). A framework for evaluation and prediction of software process improvement success. *Journal of Systems and Software*, 59(2):135–142.

Ye, E., Liu, C., and Polack-Wahl, J. (2007). Enhancing software engineering education using teaching aids in 3-D online virtual worlds. In *Proceedings of the 37th ASEE/IEEE Frontiers in Education Conference*, FIE '07, pages T1E–8–T1E–13, Milwaukee, Wisconsin.

Zapalska, A. and Brozik, D. (2006). Learning Styles and Online Education. *Campus-Wide Information Systems*, 23(5):325–335.

Zhang, H. (2008). *Qualitative & Semi-Quantitative Modelling and Simulation of the Software Engineering Processes*. PhD thesis, The University of New South Wales, Sydney, Australia.

Zhang, H. and Kitchenham, B. (2006). Semi-quantitative Simulation Modeling of Software Engineering Process. In *Wang, Q., Pfahl, D., Raffo, D., and Wernick, P., editors, Software Process Change*, 3966 volume of Lecture Notes in Computer Science, pages 242–253. Springer Berlin / Heidelberg.

Zhang, H., Kitchenham, B., and Pfahl, D. (2008). Reflections on 10 Years of Software Process Simulation Modeling: A Systematic Review. In *International Conference on Software Process*, volume 5007 of ICSP 2008, pages 345–356, Vancouver, Canada. Springer Berlin / Heidelberg.

Zhao, X., He, Z., Gui, F., and Zhang, S. (2008). Research on the Application of Six Sigma in Software Process Improvement. In *International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, IIHMSP'08, pages 937–940, Harbin, China.

Appendix A

Software Engineering Simulation Games

Table A.1. Software Engineering Games.

Name	Goals and Rules	Feedback	Virtual World	Adaptability	Game Features
SESAM	Develop a software project within the required time and budget, and with acceptable quality. Rules are based on Software Engineering lessons.	There are two types of interface: textual and graphical. Players must type commands in a restricted natural language. Feedback consists of text messages at the main screen: the number of defects, size of the documents, and the time and money spent. Some performance feedback is presented during the game and at the end (quality of the documents, feedback from customers and a reminder of the deadline and budget).	Setting: Company office. Player´s role: Project manager who needs to handle the project (assign tasks, manage the team and monitor the progress). Player: Software Engineering students. Goal: Teach Software Engineering process to students.	Lessons and the software process can be specified by the instructors, using a specific rule-based modeling language.	Graphics, interactivity, life-like challenges (constraints on budget, defects and time), employees' descriptions, dialogues (textual messages), clock based, and score. Score is in the form of a customer evaluation, with an overview of the project and a comparison between the real and planned values.
SimSE	Develop a software project within a certain set of constraints (e.g. budget, schedule, number of defects).Rules are based on Software Engineering lessons.	Instruments that provide feedback are tables, an activity list, bubbles over the employees' head, and the Explanatory Tool, which helps to correct the mistakes, understand the lessons and the software process, and explore different approaches. Score is presented at the end of the game.	Setting: Company office. Player´s role: Project manager who needs to handle the project (assign tasks, manage the team, purchase tools, and monitor the progress). Players: Software Engineering students. Goal: Teach Software Engineering process to students.	Lessons and the software process can be specified by the instructors, using a graphical interface. The interface elements can also be customized.	Graphics, interactivity, life-like challenges (e.g. constraints on budget, defects and time), employees' descriptions, dialogues (bubbles over the employees' head), random events, clock based, and score. Score is presented at the end of the game and it is defined in a 100 points scale, based on the defined constraints (e.g. product completeness, number of defects, schedule, and budget results).
The Incredible Manager	Complete a project where costs and schedule are established during the planning phase and approved by the stakeholders. Rules are related to costs and schedule progress calculation.	The informative feedback is provided through progress information about each activity, three types of messages from the employees (panic, tired or idle), the changing from the day to night (when employees leave the company), some complaints from the boss, and the time and schedule progress. Instead of presenting a final score, the game shows the estimated and used resources (time and money).	Setting: Company office. Player´s role: Project Manager who needs to plan, execute and control the project. Players: Students that need to be trained in software project management. Goal: Teach project management to students.	Simulation models, user interface and story line can be changed with a considerable effort in coding and designing.	Graphics, interactivity, life-like challenges (constraints on budget and time), employees' descriptions, and restricted dialogue. Score is not presented at the end of the game. The game ends when the project resources finish (time and money).

Name	Goals and Rules	Feedback	Virtual World	Adaptability	Game Features
SimJavaSP	Develop a software project within the required time and budget, and with acceptable quality. Rules are related to time, cost, and quality calculation.	The interface has a mix of graphical and textual feedback. The students can observe the employees' movements. Some information about the project is presented at the main screen. The events are texts message (similar to SESAM). The player has two control panels. The developers control panel (to control the development team) works in combination with the project control panel (to allow the player access the project management actions).	Setting: Company office. Player´s role: Project manager who needs to handle the project (assign tasks, allot time, purchase software and hardware, and monitor the progress). Players: Software Engineering students. Goal: Teach Software Engineering process to students.	Not supported.	Graphics, interactivity, life-like challenges (constraints on budget and time), employees' descriptions, dialogue (textual messages), and random events. Since the simulation game was not public, we could not evaluate if there is a score at the end of the game. The game ends when the project resources finish (time and money).
MO-SEProcess	Develop a software project within the required time and quality. Rules are based on Software Engineering lessons.	During the game, players can interact with each other through various communication means provided in Second Life (chat, Instant Message - IM). The game provides informative feedback related to their progress (scope, time, and quality). A team score (between 0 and 100) is given at the end of the game.	Setting: Company office with six cubicles. Player´s role: Software engineer team member who collaborates with other players to develop a software system. Player: Software Engineering students. Goal: Teach Software Engineering process to students.	Not supported.	Virtual World in Second Life, 3D Graphics, immediacy, social community, environment persistence, interactivity, life-like challenges, employees' descriptions, dialogues (chat, IM), clock based, and score. Score is presented at the end of the game and it is defined in a 100 points scale. It is calculated based on the product completeness, the number of defects, and the schedule.
SimVBSE	Make decisions based on different scenarios that involve reasoning about the trade-offs and different aspects of the project using stakeholders' values considerations, and identifying a win-win equilibrium that satisfies all success-critical stakeholders' primary values. It includes cause and effects rules that change the value of some parameters and controls.	The game provides some animated videos and tutorials for helping the students to understand their objectives and subjects of immediate interest. Regarding the informative feedback, there are different rooms, which provide different variables values (indirect and direct control variables). Each scenario is presented with a description and objective. Following each scenario, the performance feedback is provided through reports that discuss the strengths and weaknesses of each option that was available to the students based on their choices.	Setting: Software development organization. Player´s role: Project manager. Player: Software Engineering students who want to learn Value-Based Software Engineering. Goal: Practice the principles of value based in the field of software education.	Not supported.	Graphics, interactivity, life-like challenges, and animated videos. Quick assessments and feedback of students' actions after each scenario.

Name	Goals and Rules	Feedback	Virtual World	Adaptability	Game Features
qGame	Develop a software project within the required time and with acceptable quality of requirements. Rules are related to how to score correctly and incorrectly discovered requirements (avoiding wrong requirements and collecting the right ones).	The informative feedback presents the requirements as a metaphor of colored balls that appear in mental bags (bubbles over the team role head). The color distinguishes between the right and wrong requirements. The player can verify the requirements flow among the mental bags and check the number of right and wrong requirements collected from the stakeholders. These requirements can be materialized into artifacts (specification, design, and code). The time is also presented to the player. At the end, some measures related to the requirements are presented as well a score for competition (player's score and a ranking with the best scores).	Setting: Software development organization. Player´s role: Project manager. Player: Software Engineering students. Goal: Teach the problems related to the flow of requirements during the software development to students.	It presents three different levels of difficulty. However, the levels and models cannot be changed.	Graphics, interactivity, life-like challenges (constraints on defects and time), clock based, random elements (e.g. a requirement can be forgotten or lost), and score. Score is a customer evaluation and it is calculated based on the percentage of correct customer requirements delivered. The number of false software requirements (e.g. unwanted features) is deducted from the score.
TREG	Using the kitchen metaphor room, the trainee must get the necessary ingredients to learn about workshop techniques, a collaborative way for gathering and analyzing requirements. The selection of one ingredient takes the trainee to a different path to be followed. Rules correspond to 14 ingredients for accomplishing a successful requirements workshop.	Informative feedback is given to trainees through the use of NPC (Non Player Character) to guide them, HUD (Heads-up-Display) to control the scores of the game (main score, investment, time, mission, and technique) , and *Machinima* (animated videos) to record some problematic workshop situations, showing the consequences of choosing a specific path. The performance feedback "follows" the trainee's participation during the whole game.	Setting: Workshop for requirement elicitation. Players: Students, customers, users or software suppliers who want to be trained in requirement elicitation. Player´s role: Workshop facilitator (choose the right ingredients for carrying out the workshop). Goal: Teach Requirement Engineering techniques using a simulation game based on collaboration.	Not supported.	Virtual World in Second Life (but we were not able to play a multiplayer version), 3D Graphics, immediacy, social community, environment persistence, interactivity, life-like challenges (different paths to be followed), stakeholders' descriptions, dialogues (textual messages), clock based, and score. Score is presented during the whole game and consists of: number of phases (ingredients) already chosen, investment made, and the remaining time. Some phases present a number referring to the right selections. But in others, there are no right answers; in this case Machinima shows the consequences of a specific selection.

Appendix B

Software Engineering Rules

1. Requirements deficiencies are the prime source of project failures [Glass, 1998; Endres and Rombach, 2003].

2. Errors are most frequent during the requirements and design activities and are the more expensive the later they are removed [Boehm et al., 1975; Endres and Rombach, 2003].

3. A combination of different V & V methods outperforms any single method alone [Hetzel, 1976; Endres and Rombach, 2003].

4. Quality entails productivity [Jones, 1996; Cobb and Mills, 1990; Endres and Rombach, 2003].

5. Error prevention is better than error removal [Mays et al., 1990; Endres and Rombach, 2003].

6. Smaller changes have a higher error density than large ones [Basili and Perricone, 1993; Endres and Rombach, 2003].

7. One of the two most common causes of runaway projects is unstable requirements [Cole, 1995; van Genuchten, 1991; Glass, 2003].

8. One of the two most common causes of runaway projects is poor estimation [van Genuchten, 1991; Cole, 1995; Glass, 2003].

9. Software estimates are rarely adjusted as the project proceeds. Thus those estimates done at the wrong time by the wrong people are usually not corrected [NASA, 1990; Glass, 2003].

10. When moving from requirements to design, there is an explosion of "derived requirements" (the requirements for a particular design solution) caused by the complexity of the solution process. The list of these design requirements is often 50 times longer than the list of original requirements [Glass, 1982; Ebner and Kaindl, 2002; Glass, 2003].

11. Error removal is the most time-consuming phase of the life cycle [Glass, 1992, 2003].

12. It is nearly impossible to do a good job of error removal without tools. Debuggers are commonly used, but others, such as coverage analyzers, are not [Glass, 2003].

13. Rigorous inspections can remove up to 90 percent of errors from a software product before the first test case is run [Glass, 2003, 1999].

14. Mature processes and personal discipline enhance planning, increase productivity, and reduce errors [Humphrey, 1989, 1996; Endres and Rombach, 2003].

15. Two factors that affect productivity are work force experience level and level of project familiarity due to learning-curve effect [Abdel-Hamid and Madnick, 1991; Navarro, 2006].

16. Getting the most out of employees can be done by utilizing experts, employee training, skills assessment and job matching, and reducing turnover. Getting the most out of employees leads to increased productivity, which leads to decreased cycle time [Tvedt, 1996; Navarro, 2006].

17. The training of new employees is usually done by the "old-timers," which results in a reduced level of productivity on the "old-timer's" part. Specifically, on the average, each new employee consumes in training overhead 20% of an experienced employee's time for the duration of the training or assimilation period [Abdel-Hamid and Madnick, 1991; Navarro, 2006].

18. The average assimilation delay, the period of time it takes for a new employee to become fully productive, is 80 days [Abdel-Hamid and Madnick, 1991; Navarro, 2006].

19. Communication and coordination breakdown is a major phenomenon that greatly reduces software productivity and quality [Curtis et al., 1988; Navarro, 2006].

20. Matching the tasks to the skills and motivation of the people available increases productivity [Boehm, 1981; Navarro, 2006].

21. Employee motivation is the strongest influence of productivity [Boehm, 1981; Navarro, 2006].

22. Management complexity can be reduced by using project management planning tools and methods. Reducing management complexity reduces product complexity, which increases productivity [Tvedt, 1996; Navarro, 2006].

23. Motivation is increased through monetary incentives (profit sharing, pay for performance, merit pay, work measurement with incentives, and morale measurement), creating a positive frame of mind at work (employee involvement in wellness programs and creating fun at work), encouraging a feeling of commitment and responsibility (worker participation in decision-making, getting employees to think like owners, self-managing work teams, commitment to productivity breakthroughs, and providing an environment with more freedom and less restrictions), and increasing schedule pressure (using visible milestones and setting individual goals.) Increased motivation leads to increased productivity which reduces cycle time [Tvedt, 1996; Navarro, 2006].

24. Productivity is increased by increasing motivation, improving the work environment, getting the best people for the job, improving the process, and maximizing reuse [Tvedt, 1996; Navarro, 2006].

25. Nine ways to reduce cycle time are: increase productivity, reduce rework, maximize software reuse, reduce product complexity, eliminate or simplify tasks, maximize task concurrency, reduce undiscovered work, reduce risk, and use process models aimed at cycle time reduction [Tvedt, 1996; Navarro, 2006].

26. Software inspections find a high percentage of errors early in the development life cycle [Tvedt, 1996; Navarro, 2006].

27. The use of inspections can lead to defect prevention, because developers get early feedback with respect to the types of mistakes they are making [Tvedt, 1996; Navarro, 2006].

28. Inspections should be thought of as part of the development process, and time must be set aside accordingly. Once this is done, inspections can have a significant improvement in the development organization's ability to meet internal schedules [Tvedt, 1996; Navarro, 2006].

29. Proper use of inspections can even shorten life cycle [Tvedt, 1996; Navarro, 2006].

30. Participants in the inspection team get a high degree of product knowledge, which leads to higher productivity [Tvedt, 1996; Navarro, 2006].

31. Rework is usually due to customer requirements, product flaws, and communication breakdown between project members. Improving the process to reduce rework can be done by using prototyping and evolutionary development and by using formal specification methods, modern programming practices, and inspections. Reducing rework increases productivity [Tvedt, 1996; Navarro, 2006].

32. Decisions made in the upstream portion of the software development process (requirements and design) impact productivity, quality, and costs throughout the life cycle more than the other portions [Curtis et al., 1988; Navarro, 2006].

33. Fluctuating and conflicting requirements is a major phenomenon that greatly reduces software productivity and quality [Curtis et al., 1988; Navarro, 2006].

34. Specification mistakes often occur when designers do not have sufficient application knowledge to interpret the customer's intentions from the requirements document [Curtis et al., 1988; Navarro, 2006].

35. Sticking with a too-tight schedule increases cost due to a large work force [Abdel-Hamid and Madnick, 1991; Navarro, 2006].

36. As schedule pressure increases, quality assurance activities (especially walkthroughs and inspections) are often relaxed or suspended altogether [Abdel-Hamid and Madnick, 1991; Navarro, 2006].

37. In the absence of schedule pressure, a full-time employee allocates, on average, 60% of his working hours to the project (the rest is slack time: reading mail, personal activities, non-project related company business, etc.) [Abdel-Hamid and Madnick, 1991; Navarro, 2006].

38. Under schedule pressure, people tend to increase their percentage of working hours spent on the project by as much as 100%, due to spending less time on off-project activities, such as personal business and non-project communication, and/or working overtime [Abdel-Hamid and Madnick, 1991; Navarro, 2006].

39. The three "resource-type" variables that have the greatest impact on programmer productivity are the availability of programming tools, the availability of programming practices, and programmer experience [Abdel-Hamid and Madnick, 1991; Navarro, 2006].

40. The two "task-type" variables that have the greatest impact on programmer productivity are the programming language and the quality of external documentation [Abdel-Hamid and Madnick, 1991; Navarro, 2006].

41. Changing requirements are inevitable. Anticipating change with open architectures, adaptable designs, and flexible planning can help to mediate some of the ill effects of these changes [Dawson, 2000; Navarro, 2006]

42. Extreme time pressure leads to decreased productivity [Drappa and Ludewig, 2000; Navarro, 2006].

43. The earlier problems are discovered, the less the overall cost will be [Drappa and Ludewig, 2000; Navarro, 2006].

44. The more errors a document from a previous phase contains, the more errors will be passed on to the next document [Drappa and Ludewig, 2000; Navarro, 2006].

45. Extreme time pressure leads to a faster rate at which errors are made, which leads to a further delay in the completion date [Levary and Lin, 1991; Navarro, 2006].

46. The more bugs you find, the more buggy the rest of your program will likely be [McGraw, 2006; Navarro, 2006].

47. Tests reveal errors in the code. The better a test is prepared for, the higher amount of detected errors [Sommerville, 2011; Navarro, 2006].

48. Inspection is the most cost-effective measure of finding problems in software [Sommerville, 2011; Navarro, 2006].

49. The error detection effectiveness of reviews depends greatly on the qualifications and preparations of the reviewers and the completeness and correctness of the documents used as a reference [Weller, 1993; Navarro, 2006].

50. Most cost estimates tend to be too low [Glass, 1998, 2001; Endres and Rombach, 2003].

51. Object-oriented designs reduce errors and encourage reuse.[Booch, 1991; Endres and Rombach, 2003].

52. Well-structured programs have fewer errors and are easier to maintain [Dijkstra, 1969; Endres and Rombach, 2003].

53. Software reuse reduces cycle time and increases productivity and quality [Engineering, 1969; Endres and Rombach, 2003].

54. Object-oriented programming reduces errors and encourages reuse [Dahl and Nygaard, 2002; Endres and Rombach, 2003].

55. Inspections significantly increase productivity, quality and project stability [Fagan, 1986; Endres and Rombach, 2003].

56. Requirements errors are the most expensive to fix when found during production but the cheapest to fix early in development [Davis, 1993; Boehm and Basili, 2001; Glass, 2003].

57. Most software estimates are performed at the beginning of the life cycle. This makes sense until we realize that estimates are obtained before the requirements are defined and thus before the problem is understood. Estimation, therefore, usually occurs at the wrong time [Pressman, 1992; Glass, 2003].

Appendix C

Systematic Literature Review - Software Process Improvement

Table C.1. Data extraction categories.

	Attribute	Description
Organization-related data	Number of organizations	The number of organizations or departments in the study.
	Business Model	Business model involved in a global or local software development organization.
	Context	Primary function of the organization.
	Size	"Very Small" (<10 technical staff); "Small" (10 to 49 technical staff); "Medium" (50 to 249 technical staff); "Large" (>249 technical staff); "Unknown".
	Number of employees	The number of employees (or software developers) in the organization or department. When the *Number of organization* is greater than 1, this value should be specified for each organization.
	Geography	The location where the investigations have been conducted.
Research-related data	Research method	The approach used in the research.
	Data collection	How the data was collected.
	Data analysis	How the data was analyzed.
	Analysis method	Type of analysis carried out on the data.
SPI-related data	Motivation	The reasons for the SPI initiative.
	Reference model/standard	Models or standards used as reference for the process improvement.
	SPI implementation approach/model	Approaches or models that support the SPI implementation.
	List of problems detected	The description of the main problems.
	Qualitative measurements	The qualitative measurements used in SPI initiatives.
	Quantitative measurements	The quantitative measurements used in SPI initiatives. The values identified before and after the SPI initiative should be documented (if available).
	Main topic area of the SPI program	The process that were improved, according to ISO/IEC 12207.
	SPI project duration	The duration of the improvement initiative.
	List of proposed solutions	The list of usual solutions.
	Main lessons learned	Lessons learned with the SPI program.
	Success/Failure factors	List of success and failure factors of the SPI initiative.
	Benefits	The benefits for the organization with the SPI initiative.
Note	Note	Any other important information for this research.

Table C.2. Selected primary studies.

Study ID	Digital Library	Title	Year	Source	Reference
S1	Elsevier	Understanding the use of an electronic process guide	2002	Information and Software Technology	Scott et al. [2002]
S2	IEEEXplore	Historical Perspective of Two Process Transitions	2009	International Conference on Software Engineering Advances	Nikitina and Kajko-Mattsson [2009]
S3	IEEEXplore	An Empirical Study of the Complex Relationships between Requirements Engineering Processes and Other Processes that Lead to Payoffs in Productivity, Quality, and Risk Management	2006	IEEE Transactions on Software Engineering	Damian and Chisan [2006]
S4	Wiley InterScience	An Iterative Improvement Process for Agile Software Development	2007	Software Process: Improvement and Practice	Salo and Abrahamsson [2007]
S5	IEEEXplore	Improving validation activities in a global software development	2001	ICSE	Ebert et al. [2001]
S6	Wiley InterScience	Cultivation and engineering of a software metrics program	2003	Information Systems Journal	Iversen and Mathiassen [2003]
S7	Wiley InterScience	ROI of Software Process Improvement at BL Informática: SPIdex is Really Worth it	2008	Software Process: Improvement and Practice	Ferreiro Ferreira et al. [2008]
S8	SpringerLink	An Integrated Framework to Guide Software Process Improvement in Small Organizations	2009	EuroSPI	Pino et al. [2009]
S9	Wiley InterScience	Appreciative Inquiry in Software Process Improvement	2009	Software Process: Improvement and Practice	Holmberg et al. [2009]
S10	Wiley InterScience	A Practical Application of the IDEAL Model	2004	Software Process: Improvement and Practice	Casey and Richardson [2004]
S11	SpringerLink	Improving the Product Documentation Process of a Small Software Company	2009	International Conference on Product Focused Software Process Improvement	Valtanen et al. [2009]
S12	Elsevier	Software process improvement as emergent change: A structurational analysis	2007	Information and Software Technology	Allison and Merali [2007]
S13	Elsevier	Implementing requirements engineering processes throughout organizations: success factors and challenges	2004	Information and Software Technology	Kauppinen et al. [2004]
S14	Elsevier	Experiences on establishing software processes in small companies.	2006	Information and Software Technology	von Wangenheim et al. [2006]
S15	IEEEXplore	An Experience in Combining Flexibility and Control in a Small Company's Software Product Development Process	2003	International Symposium on Empirical Software Engineering	Rautiainen et al. [2003]
S16	IEEEXplore	Implementation of a Software Quality Improvement Project in an SME: A Before and After Comparison	2009	Euromicro Conference on Software Engineering and Advanced Applications	Tosun et al. [2009]
S17	IEEEXplore	Applying a Systematic Approach to Link Requirements and Testing: A Case Study	2009	Asia-Pacific Software Engineering Conference	Kukkanen et al. [2009]
S18	IEEEXplore	Experiences and Results from Establishing a Software Cockpit at BMD Systemhaus	2009	Euromicro Conference on Software Engineering and Advanced Applications	Larndorfer et al. [2009]
S19	IEEEXplore	Improving quality, one process change at a time	2009	ICSE	Pinheiro et al. [2009]
S20	IEEEXplore	Combining Perceptions and Prescriptions in Requirements Engineering Process Assessment: An Industrial Case Study	2009	IEEE Transactions on Software Engineering	Napier et al. [2009]
S21	IEEEXplore	Improving software Risk Management in a Medical Device Company	2009	ICSE	McCaffery et al. [2009]
S22	IEEEXplore	Case Study: How Analysis of Customer Found Defects Can Be Used by System Test to Improve Quality	2009	ICSE	Moritz [2009]

Study ID	Digital Library	Title	Year	Source	Reference
S23	IEEEXplore	Research on the Application of Six Sigma in Software Process Improvement	2008	International Conference on Intelligent Information Hiding and Multimedia Signal Processing	Zhao et al. [2008]
S24	IEEEXplore	Scrum and CMMI Level 5: The Magic Potion for Code Warriors	2008	Annual Hawaii International Conference on System Sciences	Sutherland et al. [2008]
S25	IEEEXplore	Continuous Improvement through Iterative Development in a Multi-Geography	2008	IEEE International Conference on Global Software Engineering	Laredo and Ranjan [2008]
S26	IEEEXplore	Defect Prevention: A General Framework and Its Application	2006	International Conference on Quality Software	Li et al. [2006]
S27	IEEEXplore	SPDW: A Software Development Process Performance Data Warehousing Environment	2006	IEEE/NASA Software Engineering Workshop	Becker et al. [2006]
S28	IEEEXplore	Six Sigma Approach in Software Quality Improvement	2006	International Conference on Software Engineering Research, Management and Applications	Redzic and Baik [2006]
S29	IEEEXplore	Software Process Definition Improvement: An Industry Report	2006	Euromicro Conference on Software Engineering and Advanced Applications	Jester et al. [2006]
S30	IEEEXplore	Tackling the Complexity of Requirements Engineering Process Improvement by Partitioning the Improvement Task	2005	Australian Software Engineering Conference	Nikula and Sajaniemi [2005]
S31	IEEEXplore	The Introduction and Use of a Tailored Unified Process - A Case Study	2005	Euromicro Conference on Software Engineering and Advanced Applications	Westerheim and Hanssen [2005]
S32	IEEEXplore	An Industrial Case Study of Implementing and Validating Defect Classification for Process Improvement and Quality Management	2005	International Software Metrics Symposium	Freimut et al. [2005]
S33	IEEEXplore	Improving Requirements Management in Extreme Programming with Tool Support - an Improvement Attempt that Failed	2004	Euromicro Conference on Software Engineering and Advanced Applications	Kääriäinen et al. [2004]
S34	IEEEXplore	Managing Product Requirements for Medical IT Products	2002	IEEE Joint International Conference on Requirements Engineering	Higgins et al. [2002]
S35	IEEEXplore	Improving the Testing Process by Program Static Analysis	2001	Asia-Pacific Software Engineering Conference	Kikuchi and Kikuno [2001]
S36	IEEEXplore	Structural change and change advocacy: a study in becoming a software engineering organization	2001	Hawaii International Conference on System Sciences	Nelson et al. [2001]
S37	ACM	Introducing a software modeling concept in a medium-sized company	2000	ICSE	Schmid et al. [2000]
S38	IEEEXplore	An Experience: A Small Software Company Attempting to Improve its Process	1999	International Workshop Software Technology and Engineering Practice	Otoya and Cerpa [1999]
S39	ACM	Strategic alignment of software process improvement programs using QFD	2008	International workshop on Business impact of process improvements	Becker et al. [2008]
S40	ACM	Software process improvement by object technology (ESSI PIE 27785 - SPOT)	2000	ICSE	Caliò et al. [2000]
S41	IEEEXplore	A case-study on using an Automated In-process Software Engineering Measurement and Analysis system in an industrial environment	2009	ICSE	Coman et al. [2009]
S42	ACM	Applying and adjusting a software process improvement model in practice: the use of the IDEAL model in a small software enterprise	2000	ICSE	Kautz et al. [2000]

Study ID	Digital Library	Title	Year	Source	Reference
S43	ACM	European experiences with software process improvement	2000	ICSE	O'Hara [2000]
S44	ACM	An empirical study of industrial requirements engineering process assessment and improvement	2005	Transactions on Software Engineering and Methodology (TOSEM)	Sommerville and Ransom [2005]
S45	Wiley InterScience	A minimal test practice framework for emerging software organizations	2005	Software Testing, Verification and Reliability	Karlström et al. [2005]
S46	Wiley InterScience	A Portrait of a CMMI Level 4 Effort	2002	Systems Engineering	Hollenbach and Smith [2002]
S47	Wiley InterScience	A Process Framework for Small Projects	2001	Software Process: Improvement and Practice	Leung and Yuen [2001]
S48	Wiley InterScience	An Experience in Facilitating Process Improvement with an Integration Problem Reporting Process Simulation	2006	Software Process: Improvement and Practice	Houston [2006]
S49	Wiley InterScience	BBN Based Approach for Improving the Software Development Process of an SME - A Case Study	2009	Software Process: Improvement and Practice	Bibi et al. [2010]
S50	Wiley InterScience	Continuous Process Improvement at a Large Software Organization	2009	Software Process: Improvement and Practice	Malheiros et al. [2009]
S51	Wiley InterScience	Experiences and Results from Tailoring and Deploying a Large Process Standard in a Company	2008	Software Process: Improvement and Practice	Armbrust et al. [2008]
S52	Wiley InterScience	Pioneering Process Improvement Experiment in Hungary	2000	Software Process: Improvement and Practice	Biró et al. [2000]
S53	Wiley InterScience	Product-focused software process improvement (P-SPI): concepts and their application	1999	Quality and Reliability Engineering International	van Solingen et al. [1999]
S54	Wiley InterScience	Pursuing coherence in software process assessment and improvement	2001	Software Process: Improvement and Practice	Cattaneo et al. [2001]
S55	Wiley InterScience	Release planning process improvement - an industrial case study	2006	Software Process: Improvement and Practice	Momoh and Ruhe [2006]
S56	Wiley InterScience	SPI in a Very Small Team: a Case with CMM	2000	Software Process: Improvement and Practice	Batista and Figueiredo [2000]
S57	Wiley InterScience	SPICE For Small Organisations	2004	Software Process: Improvement and Practice	Tuffley et al. [2004]
S58	Wiley InterScience	TRISO-Model: A New Approach to Integrated Software Process Assessment and Improvement	2007	Software Process: Improvement and Practice	Li [2007]
S59	SpringerLink	Using a Reference Application with Design Patterns to Produce Industrial Software	2004	International Conference on Product Focused Software Process Improvement	Vokác and Jensen [2004]
S60	SpringerLink	Starting SPI from Software Configuration Management: A Fast Approach for an Organization to Realize the Benefits of SPI	2004	International Conference on Product Focused Software Process Improvement	Ikeda and Akamatsu [2004]
S61	SpringerLink	Software Verification Process Improvement Proposal Using Six Sigma	2007	International Conference on Product Focused Software Process Improvement	Galinac and Car [2007]
S62	SpringerLink	Software Process Improvement with Agile Practices in a Large Telecom Company	2006	International Conference on Product Focused Software Process Improvement	Auvinen et al. [2006]
S63	SpringerLink	Software Experience Bases: A Consolidated Evaluation and Status Report	2000	International Conference on Product Focused Software Process Improvement	Conradi and Dingsrøyr [2000]
S64	SpringerLink	Software Development Improvement with SFIM	2007	International Conference on Product Focused Software Process Improvement	Krikhaar and Mermans [2007]

Study ID	Digital Library	Title	Year	Source	Reference
S65	SpringerLink	Quantitatively Managing Defects for Iterative Projects: An Industrial Experience Report in China	2008	International Conference on Software process	Gou et al. [2008]
S66	SpringerLink	Quality through Managed Improvement and Measurement (QMIM): Towards a Phased Development and Implementation of a Quality Management System for a Software Company	2001	Software Quality Journal	Balla et al. [2001]
S67	SpringerLink	Project Web and Electronic Process Guide as Software Process Improvement	2005	EuroSPI	Moe et al. [2005]
S68	SpringerLink	Product Driven Process Improvement PROFES Experiences at Dräger	2000	International Conference on Product Focused Software Process Improvement	van Latum and van Uijtregt [2000]
S69	SpringerLink	Preventing Requirement Defects: An Experiment in Process Improvement	2001	Requirement Engineering	Lauesen and Vinter [2001]
S70	SpringerLink	Organisational Dynamics in the Software Process Improvement: The Agility Challenge	2004	IFIP International Federation for Information Processing	Börjesson and Mathiassen [2004]
S71	SpringerLink	Managing the Software Process in the Middle of Rapid Growth: A Metrics Based Experiment Report from Nokia	2000	International Conference on Advanced Information Systems Engineering	Kilpi [2000]
S72	SpringerLink	Improving reliability of large software systems	1999	Annals of Software Engineering	Ebert et al. [1999]
S73	SpringerLink	Implementing an ISO 9001 Certified Process	2006	EuroSPI	Stlålhane [2006]
S74	SpringerLink	Experiences from introducing UML-based development in a large safety-critical project	2006	Empirical Software Engineering	Anda et al. [2006]
S75	SpringerLink	Developing Software with Scrum in a Small Cross-Organizational Project	2006	EuroSPI	Dingsøyr et al. [2006]
S76	SpringerLink	Crossing the Chasm in Software Process Improvement	2005	IFIP International Federation for Information Processing	Börjesson et al. [2005]
S77	SpringerLink	Assessing Improvements of Software Metrics Practices	2004	IFIP International Federation for Information Processing	Frederiksen and Mathiassen [2004]
S78	SpringerLink	Applying Six Sigma in the Field of Software Engineering	2008	International Conferences on Software Process and Product Measurement	Russ et al. [2008]
S79	SpringerLink	An Interdisciplinary Approach for Successfully Integrating Human-Centered Design Methods into Development Processes Practiced by Industrial Software Development Organizations	2001	IFIP International Conference on Engineering for Human-Computer Interaction	Metzker and Offergeld [2001]
S80	SpringerLink	An evolutionary cultural-change approach to successful software process improvement	2009	Software Quality Journal	Elliott et al. [2009]
S81	SpringerLink	Active Probes Synergy in Experience-Based Process Improvement	2000	International Conference on Product Focused Software Process Improvement	Schneider [2000]
S82	IEEEXplore	Improving Software Development through Three Stages	2006	IEEE Software	Motoyama [2006]
S83	IEEEXplore	Blending CMM and Six Sigma to meet business goals	2003	IEEE Software	Murugappan and Keeni [2003]
S84	IEEEXplore	Applying the PSP in industry	2000	IEEE Software	Morisio [2000]
S85	IEEEXplore	Making Statistics Part of Decision Making in an Engineering Organization	2008	IEEE Software	Card et al. [2008]
S86	IEEEXplore	Combining agile methods with stage-gate project management	2005	IEEE Software	Karlström and Runeson [2005]
S87	IEEEXplore	Software process improvement in small organizations: a case study	2005	IEEE Software	Dangle et al. [2005]

Study ID	Digital Library	Title	Year	Source	Reference
S88	IEEEXplore	Adopting the SW-CMM in a small IT organization	2004	IEEE Software	Guerrero and Eterovic [2004]
S89	IEEEXplore	Attaining Level 5 in CMM process maturity	2002	IEEE Software	McGarry and Decker [2002]
S90	IEEEXplore	Telcordia Technologies: the journey to high maturity	2000	IEEE Software	Pitterman [2000]
S91	IEEEXplore	The evolution of quality processes at Tata Consultancy Services	2000	IEEE Software	Kenni [2000]

Table C.3. Quality scores.

Score	Study ID	Frequency
<5	S2, S18, S23, S33, S42, S59, S73, S81, S90, and S91.	10
$5 \leq and < 10$	S5, S7, S10, S11, S14, S17, S19, S21, S24, S25, S26, S27, S29, S30, S31, S32, S34, S35, S37, S38, S39, S40, S41, S43, S46, S48, S50, S51, S52, S53, S54, S55, S57, S58, S60, S64, S66, S71, S72, S78, S79, S80, S82, S83, S84, S86, S87, S88, and S89.	49
$10 \leq and < 15$	S1, S3, S4, S6, S8, S9, S12, S13, S15, S16, S22, S28, S36, S44, S45, S47, S49, S56, S61, S62, S63, S65, S67, S68, S69, S75, S76, S77, and S85.	29
15	S20, S70, and S74.	3

Table C.4. Comparison of related SLRs.

Title	# Studies	Period	Research Questions	Answer
Evaluation and Measurement of Software Process Improvement - A Systematic Literature Review [Unterkalmsteiner et al., 2012]	148	1991 to 2008	What types of evaluation strategies are used to evaluate SPI initiatives?	The most common evaluation strategy is the "Pre-Post Comparison" applied in 49% of the inspected papers. Others include: statistical analysis, "Pre-Post comparison & survey" and Statistical Process control.
			What are the reported metrics for evaluating the SPI initiatives?	Quality (Process Quality, Product Quality and other quality attributes) was the most measured attribute in 62% of the studies, followed by Cost (Effort and Cost) in 41% of the studies and Time-to-market in 18% of the studies.
			What measurement perspectives are used in the evaluation and to what extent are they associated with the identified SPI initiatives?	"Project" perspective represents the majority with 66%, followed by "Project and Product" with 20% and "Project, Product and organization" with 5%. The CMM, CMMI, ISO/IEC 15504 and PSP were conducted mostly in the "Project" perspective.
			What are the confounding factors in relation to the identified evaluation strategies?	Most of the articles do not consider the potential confounding factors. A list of identified confound factors was provided in this SLR.
Software process improvement in small and medium software enterprises: a systematic review [Pino et al., 2008]	45	January 1996 - March 2006	What approaches concerning SPI exist, which focus both on SMEs and report on a case study?	The current state of the practice of SPI efforts in SMEs is: 47% of the companies involved in the improvement effort were small companies, 33% were very small and 20% were medium companies. The most frequently used improvement models in the SMEs are CMM (25%) as a process reference model, ISO/IEC 15504 (56%) as a process assessment method and IDEAL (20%) as the model for guiding improvement. 71% of the improvement effort focused on the guidance of an improvement project and the priority of improvement implementation, along with the use of existing improvement models, adjusting them to the needs. The most improved process is the "Project Management" process (according to the ISO/IEC 12207:2004 processes classification).

Title	# Studies	Period	Research Questions	Answer
A systematic literature review of software process improvement in small and medium web companies [Sulayman and Mendes, 2009]	4	no time period specified	Which software process improvement models/techniques are applied by small and medium Web development organizations?	All the suggested SPI models used an iterative approach influenced by CMM/CMMI, and showed a strong tendency to use the IDEAL model.
			Which software process improvement models/techniques were successful to small and medium Web development organizations and how success is being measured?	Post project analysis and process management & measurement were the techniques that have been used in all four studies. The measurements of success for small and medium Web companies included development team and client satisfaction, increase in productivity, reduced development time, compliance with standards and overall operational excellence.
			Are there any software process improvement models that have been specifically made to measure for small and medium Web companies?	Since none of the studies were specific to the SPI of small and medium Web development companies, they did not find any specific model.
			What are the important characteristics of small and medium Web organizations that pursue software process improvement activities and practices?	None of the selected studies provided data relevant to the specific measurement models for small and medium Web companies.
			What constitutes a small or medium Web organization for the studies investigated?	These companies operate under tight budget constraints and with short deadlines; their strategies do not tend to be risk-based; they always demand quick results, using a "quick-to-market" approach.
Software Process Improvement as organizational change: A metaphorical analysis of the literature [Müller et al., 2010]	71	no time period specified	What perspectives do the literature offer on SPI as organizational change and how is this knowledge presented and published?	The literature as a whole is firmly grounded in both theory and practice, it appropriately targets both practitioner and academic audiences, and Scandinavian and American researchers are the main contributors. The current literature offers important insights into organizational change in SPI when viewed through machine, organism, flux and transformation, and brain metaphors. In contrast, the other metaphors (the impact of culture, dominance, psychic prison, and politics) has only received scarce attention.
Systematic review of organizational motivations for adopting CMM-based SPI [Staples and Niazi, 2008]	43	no time period specified	Why do organizations embark on CMM-based SPI initiatives?	The results show that organizations adopted CMM-based SPI mainly to improve their product quality and project performance (e.g. development time, development cost, and productivity). Satisfying customers was not a very common reason for adopting CMM-based SPI, and organizations rarely adopted it in order to improve their employees' capability, motivation, or work environment.

Appendix D

Questionnaires used for Validating SPIAL

DiF. 1 - Influencing Factors - Background questionnaire (Before Pre-test)

Start time: _____

Name (optional): _____

Gender : ○ male ○ female

Age: ____

English level: ○ Basic
○ Intermediate
○ Advanced

University Education:

Course: ○ Bachelor in Computer Science
○ Bachelor in Information Systems
○ Bachelor in Electric Engineering
○ Specialization in Software Engineering
○ Other (specify): _____

Is your first Software Engineering course? ○ yes ○ no

Practical Software Engineer Experience:

Do you have any Professional certification?

- ☐ MPS.BR - Brazilian Model for Software Process Improvement
- ☐ CMMI
- ☐ PMP - Project Management Professional
- ☐ Other (specify): _____
- ☐ No

Full-time student? ○ yes ○ no

If you answered **NO** (answer the questions below):

Position: _____

(e.g. trainee, project manager, software quality member, software developer, software analyst)

Team: _____

(Process, Requirement Engineering, Implementation, Test, Usability, Quality, Administration)

Roles: _____

(Requirements Analyst, Software Architect, Designer, Integrator, Implementer, Test Designer)

(Tester, Project Manager, Process Engineer, Quality Analyst, Configuration Manager)

How many hours do you work per week: _____ h/week

Did you take any other complementary Software Engineering training (for example, a project management training)?

○ yes ○ no

If **YES**, subject taught: _____

Duration: _____ hours

DiF. 2 - Software Process Improvement literature

How many items containing information about Software Process Improvement have you read?

Books: ○ 0 ○ 1-2 ○ 3-5 ○ more than 5

Papers: ○ 0 ○ 1-2 ○ 3-5 ○ more than 5

Other items (specify): _____ ○ 0 ○ 1-2 ○ 3-5 ○ more than 5

What does the acronym CMMI stand for (in Software Engineering)?

○ Constructive Maturity Model Interactive

○ Capability Measurement Model Integration

○ Capability Maturity Model Integration

○ Co-operative Maturity Model Interactive

○ I don´t know

Have you already learned the basic concepts of the CMMI and Software Engineering (e.g. disciplines)?

○ yes ○ no

DiF. 2 - Learning Style <more than one answer is possible> (adapted from Pfahl [2001])

What is your preferred learning style?

○ reading of text books (with exercises)

○ classroom lectures (with exercises)

○ group work (interaction with peers and instructor)

○ web-based training modules (with computer interaction / including examples and exercises)

End time: _____

Pre-Test/Post-Test

Start time: _____

Name(optional): _____

Dep. 1 - Interest in Software Process Improvement (adapted from Pfahl [2001])

Below you will find a number of opposing adjectives on both sides of each line. You can react to the statements by checking the appropriate number as the next example:

disagree ● 1 ○ 2 ○ 3 ○ 4 ○ 5 agree

Scale: 1:fully disagree 2:disagree 3:undecided 4:agree 5:fully agree

I consider it very important for computer science students to know as much as possible about Software Process Improvement:

disagree ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 agree

I would like to get more information on Software Process Improvement in my Software Engineering lectures at the university:

disagree ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 agree

I would like to participate in a seminar (either at university or in the form of a training course) about Software Process Improvement:

disagree ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 agree

I consider it very important for Software Engineers to know as much as possible about Software Process Improvement:

disagree ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 agree

I would like to learn more about the Software Process Improvement:

disagree ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 agree

Dep. 2 - Software Process Improvement Competency <exactly ONE answer per question> (adapted from [Gresse von Wangenheim et al., 2009])

What is your knowledge about CMMI:

○ I know nothing

○ I have a vague notion

○ I know the basics

○ I can apply the concepts in practice with assistance (identify problems and propose solutions)

○ I can apply the concepts in practice without assistance (identify problems and propose solutions)

Dep. 3 - Knowledge on the remembering level

1- List three metrics that can be used to monitor a Software Process Improvement initiative.

_____

_____

_____

_____

2- Which CMMI representation allows the organization to choose different process areas to be improved with different progress rate:

○ Single process

○ Continuous

○ Staged

○ Multi-level

3- The CMMI has FIVE maturity levels. Which one is **NOT** a maturity level:

○ Incomplete: processes are not performed or are partially performed.

○ Managed: processes are planned and executed in accordance with policy.

○ Defined: processes are well characterized and understood.

○ Optimizing: processes are continually improved based on an quantitative understanding.

○ Initial: processes are ad hoc.

4- What are typical improvement initiatives carried out by immature organizations (more than one answer can be selected):

☐ Establish Performance Baselines and Models

☐ Quantitatively Manage the Project

☐ Manage Requirements

☐ Develop a Project Plan

☐ Determine Causes of Defects

☐ Track and Control Change

5- With respect to the structure of CMMI, check the CORRECT answer.

○ The maturity levels consist of a predefined set of process areas, which have specific and generic goals. The sequence of levels in order are: initial, optimized and quantitatively managed.

○ In the maturity level 2, projects establish their processes by tailoring the set of organizational standard processes following the tailoring organizational instructions.

○ In the maturity level 4, subprocesses are selected to be controlled using predominantly qualitative methods and techniques.

○ A maturity level is a defined evolutionary plateau for organizational process improvement. It provides the way to characterize organization's performance.

6- Which steps are essential (in the correct sequence) to the execution of a Software Process Improvement initiative:

○ Define business goals, derive and select improvements, change process, measure the results, communicate results.

○ Define business goals, implement actions, communicate results, derive and select new improvement, give feedback.

○ Define project specific goals, derive and select improvements, communicate results, change process, measure the results.

○ Define project specific goals, measure process, give feedback, change process, implement actions, communicate results.

Dep. 4 - Knowledge on the understanding level

1- Assume you are responsible for a process improvement in a small software organization. Assume you don't have any additional information about the project that the company is starting to develop. You already noted that the company has not defined internal procedures (ad-hoc process) and there is a critical dependence on key resources. Due to new customer requirements, the reliability level of the software has to be "very high" (i.e. without major defects). Without changing the project planning, which process you would invest first in order to achieve the increased reliability level?

○ Organizational Process Performance
○ Organizational Process Definition
○ Measurement and Analysis
○ Technical Solution
○ Causal Analysis and Resolution
○ all areas (above) should be intensified equally

2- If the problems identified above persist. What should be a suitable explanation?

○ The previous experience was suitable.
○ The team presented resistance to change.
○ The involved people were the process performers.
○ The key resources cooperated with the initiative.

3- Which rule does **NOT** apply during an SPI initiative:

○ Inspection is the most cost-effective measure of finding problems in software.
○ Errors are most frequent during the requirements and design activities and are the more expensive the later they are removed.
○ Problems in requirements is a major phenomenon that greatly reduces software productivity and quality.
○ Investments in Measurement and Analysis depends on the capability level of other process areas.
○ Mature process and personal discipline enhance planning, increase productivity and reduce errors.

4- Assume you are working in a high level maturity organization, and the top level management decided as business goal to increase profitability and provide better level of quality to customers. The organization observed recurrent defects detected during the development and after delivery. Example of defects included incorrect bug fixes and missing requirements. All managers are engaged in the improvement effort, which is in agreement with the organization's business goals. The investment in which area will result in a reduction of recurrent defects:

○ Measurement and Analysis
○ Technical Solution
○ Requirement Management
○ Defect Causal Analysis
○ Process and Product Quality Assurance
○ Configuration Management

5- After investing heavily in the Requirement Management and Requirement Development Processes area (CMMI Level 2 and 3) before the beginning of a development project, which of the patterns presented below described the typical defect variation on this development project most appropriate (sum of defects detected):

○ Figure 1 ○ Figure 2 ○ Figure 3 ○ Figure 4

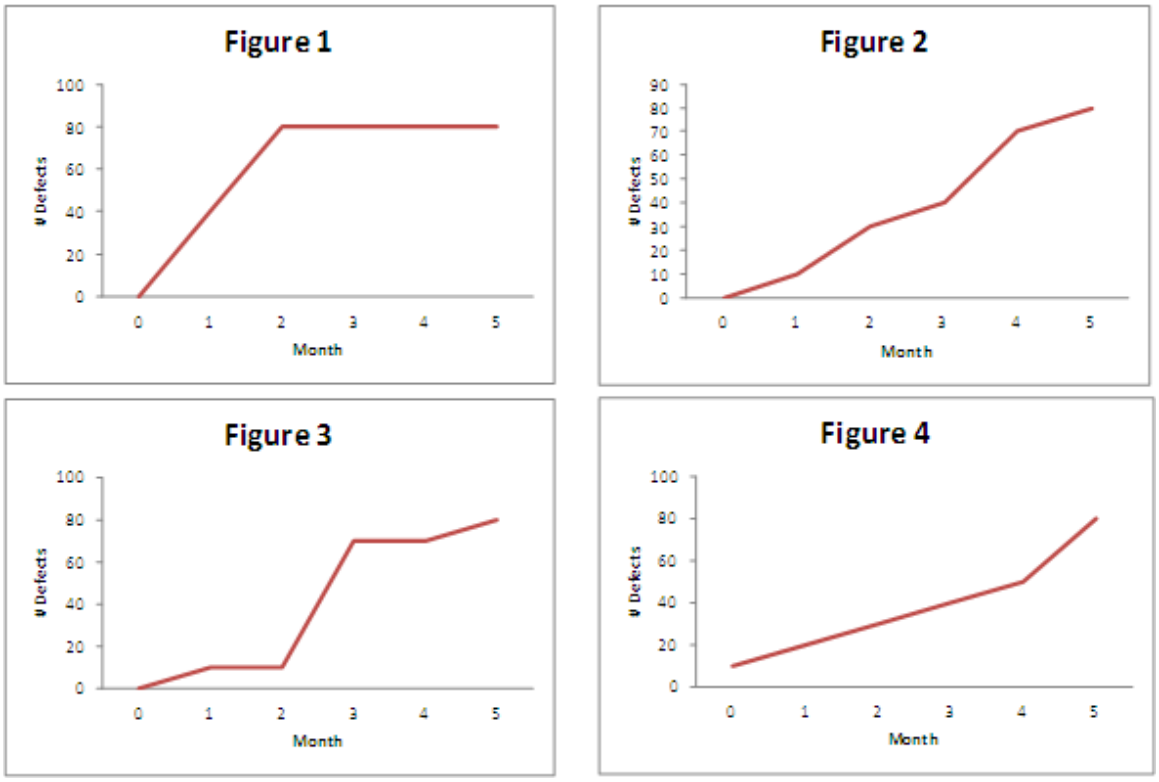

Figure D.1. Defects detected after process improvement per month.

6- In the same scenario described in question 5, which improvement percentage can most appropriately describe the typical defect reduction:

○ 5-35%
○ 0-5%
○ greater than 100% less than 200%
○ greater than 1000%

7- Which de-motivators factor (improvement difficulties) is **NOT** essential to a process improvement initiative?

○ Resistance to change
○ Imposition
○ Budget constraints
○ Time pressure/constraints
○ Lack of commercial pressures

8- Assume you are working in a high level maturity organization, and the top level management decided as business goal to reduce the number of defects in order to control the processes, and achieve the quality and process performance objectives. Which area should be invested in order to allow the achievement of business goal:

- ○ Measurement and Analysis
- ○ Technical Solution
- ○ Quantitative Project Management
- ○ Organizational Innovation and Deployment
- ○ Process and Product Quality Assurance
- ○ Configuration Management

Dep. 5 - Knowledge on the application level

1 - You have just been named the manager of the Software Engineering process group of an established software development company. In this company, it is already collected some of the project data. However, the data does not follow a standard, some metrics were not used to make informed decisions, the interpretation of the metrics was not trivial and there is no managerial responses based on the metrics collected. Now, the senior management wants to observe trends on the quality and productivity. What would be an adequate process improvement program in this context?

Objective:

Process areas to invest:

Measures to evaluate the program:

2 - You have just been named the responsible for the process improvement of a small software development company. This company has a lack of internal procedures and a critical dependence on key resources. Even the existing procedures were not adequately documented. What would be an adequate process improvement program in this context?

Objective:

Process areas to invest:

Measures to evaluate the program:

End time: _____

Game-play subjective evaluation:

Start time: _____

Below you will find a number of opposing adjectives on both sides of each line. You can react to the statements by checking the appropriate number as the next example:
dislike ● 1 ○ 2 ○ 3 ○ 4 ○ 5 liked
Scale: 1:disliked 2:liked little 3:undecided 4:liked 5:liked a lot

Engagement (adapted from [Gresse von Wangenheim et al., 2009])

How enjoyable is playing this game?

dislike ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 liked

Do you have fun during the game play?

boring ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 lots of fun

Appropriateness (adapted from [Gresse von Wangenheim et al., 2009])

Below you will find a number of opposing adjectives on both sides of each line. You can react to the statements by checking the appropriate number as the next example:
unsatisfactory ● 1 ○ 2 ○ 3 ○ 4 ○ 5 excellent
Scale: 1:Unsatisfactory 2:fair 3:undecided 4:good 5: excellent

What do you think about the game duration?

Unsatisfactory ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 excellent

If you select unsatisfactory or fair, please specify the problem (game lasts too long or too short) and Why?

_____

_____

_____

What is your opinion about the difficulty of the game:

Unsatisfactory ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 excellent

What do you think about the content?

Unsatisfactory ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 excellent

Do you think that the game reflects aspects of a real Software Process Improvement initiative?

Unsatisfactory ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 excellent

Is the game sufficient for its purpose?

Unsatisfactory ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 excellent

Is the play sequence satisfactory?

Unsatisfactory ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 excellent

Is this game an adequate **complementary** teaching method?

Unsatisfactory ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 excellent

What is your opinion about adopting this game in a Software Engineering class?

Unsatisfactory ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 excellent

Do you think a class using this game will be better than a traditional class (without this game)?

○ yes ○ no

Learning Perspective (adapted from [Lethbridge, 1998])

Below you will find a number of opposing adjectives on both sides of each line. You can react to the statements by checking the appropriate number as the next example:

learning nothing ● 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 learning in depth

Scale: 0:Learned nothing at all 1:Became vaguely familiar 2:Learned the basics 3:Became functional (moderate working knowledge) 4:Learned a lot 5: Learned in depth; became expert (learned almost everything)

How much did you learn about **new** Software Engineering concepts (including CMMI, SPI measures) playing this game?

Learned nothing at all ○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 Learned in depth; became expert (learned almost everything)

How much did you learn about the practical application of an SPI program in an organization?

Learned nothing at all ○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 Learned in depth; became expert (learned almost everything)

How useful has this game been to you to **reinforce** Software Engineering concepts that were presented in class?

Scale: 0 Completely useless 1 Almost never useful 2 Occasionally useful 3 Moderately useful 4 Very useful 5 Essential

Completely useless ○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 Essential

Open questions (adapted from [Navarro, 2006])

List and explain your most favorite aspects found in the game?

List and explain your least favorite aspects found in the game?

Is there anything confusing about the game? Please, list the confusing parts.

What changes do you suggest to improve the game?

What was your score? _____

End time: _____

DiF. 3 - Influencing Factors(adapted from [Pfahl, 2001])

Below you will find a number of opposing adjectives on both sides of each line. You can react to the statements by checking the appropriate number as the next example:

useless ● 1 ○ 2 ○ 3 ○ 4 ○ 5 useful

Scale: 1:Very useless 2:Useless 3:Undecided 4:Useful 5: Very useful

I consider the explanations/information provided by the instructor in general?

useless ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 useful

boring ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 interesting

difficult ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 easy

confusing ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 clear

I would like to make the following comment(s)/ suggestion(s):

Did you have enough time to complete the whole test?
○ yes ○ no