Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação

MARCELO PROCOPIO GUIMARÃES SANTOS

O PLANEJAMENTO E A EMERGÊNCIA DA ARQUITETURA DOS SOFTWARES DESENVOLVIDOS COM OS MÉTODOS DOS AGILISTAS

Belo Horizonte
2016

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação
Especialização em Informática: Ênfase: Engenharia de Software

O PLANEJAMENTO E A EMERGÊNCIA DA ARQUITETURA DOS SOFTWARES DESENVOLVIDOS COM OS MÉTODOS DOS AGILISTAS

por

MARCELO PROCOPIO GUIMARÃES SANTOS

Monografia de Final de Curso
CEI-ES-0xxx-T20-2016-01

Prof. Rodolfo Sérgio Ferreira Resende
Orientador

Belo Horizonte
2016

MARCELO PROCOPIO GUIMARÃES SANTOS

O PLANEJAMENTO E A EMERGÊNCIA DA ARQUITETURA DOS SOFTWARES DESENVOLVIDOS COM OS MÉTODOS DOS AGILISTAS

Monografia apresentada ao Curso de Especialização em Informática do Departamento de Ciências Exatas da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Especialista em Informática.

Área de concentração: Engenharia de Software

Orientador(a): Prof. Rodolfo Sérgio Ferreira Resende

Belo Horizonte
2016

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Santos, Marcelo Procópio Guimarães.

S58m O planejamento e a emergência da arquitetura dos softwares desenvolvidos com os métodos dos agilistas . / Marcelo Procópio Guimarães Santos. Belo Horizonte, 2016.
viii, 38 f.: il.; 29 cm.

Monografia (especialização) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Rodolfo Sérgio Ferreira Resende.

1. Computação 2. Engenharia de software. 3. Software - Desenvolvimento. Orientador. II. Título.

CDU 519.6*32(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS

INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
ESPECIALIZAÇÃO EM INFORMÁTICA: ÁREA DE CONCENTRAÇÃO ENGENHARIA DE SOFTWARE

O PLANEJAMENTO E A EMERGÊNCIA DA ARQUITETURA DOS SOFTWARES DESENVOLVIDOS COM OS MÉTODOS DOS AGILISTAS

MARCELO PROCÓPIO GUIMARÃES SANTOS

Monografia apresentada aos Senhores:

Prof. Rodolfo Sérgio Ferreira de Resende
Orientador
DCC - ICEx - UFMG

Profa. Gisele Lobo Pappa
DCC - ICEx - UFMG

Belo Horizonte, 29 de fevereiro de 2016

À Deus,
aos professores,
aos colegas de curso e
aos meus familiares,
dedico este trabalho.

AGRADECIMENTOS

Inicialmente quero agradeço a Deus, pelos dons recebidos.

Agradeço aos meus pais, pelo amor incondicional.

Aos meus professores, pelos conhecimentos adquiridos.

E finalmente aos colegas de curso pela convivência e trocas.

"Viva como se fosse morrer amanhã.
Aprenda como se fosse viver para sempre”
Mahatma Gandhi

RESUMO

O principal objetivo do presente trabalho é apresentar um estudo exploratório a respeito de aspectos da emergência da arquitetura de produtos durante o desenvolvimento de software, empregando equipes que adotam modelos de processo de desenvolvimento ágil. Nos métodos tradicionais, os requisitos devem ser obtidos de forma extensiva e exaustiva e, a arquitetura deve ser definida e especificada de forma a atender a este conjunto de requisitos de forma abrangente e consolidada. Nos métodos dos agilistas não é possível incorporar a forma tradicional de definição do desenho da arquitetura de software, pois não existe a pretensão de obter os requisitos de forma extensiva e exaustiva. Os requisitos são considerados em função dos desejos e expectativas julgados mais urgentes e importantes. Então, desenha-se a arquitetura de maneira incremental na medida em que os requisitos vão sendo considerados, com a crença de que a arquitetura irá "emergir" a partir dos diferentes conjuntos de requisitos considerados. Neste trabalho são apresentados: i) formas de desenhar a arquitetura de software plena antes de iniciar a etapa de codificação; ii) princípios e trabalhos relacionados a desenvolvimento de software quanto ao trabalho de arquitetura, empregando equipes que adotam modelo de processo de desenvolvimento agil; iii) práticas dos métodos de desenvolvimento ágil quando empregadas em projetos de desenvolvimento de software; iv) como os princípios dos métodos de desenvolvimento ágil podem influenciar no desenho da arquitetura e os riscos de se trabalhar com uma arquitetura “emergente”; e, v) práticas de gestão de projeto de software em função da emergência da arquitetura em projetos de software quando executados por equipes que empregam métodos ágeis.

Palavras-chave: Processo de desenvolvimento de software, Agilidade, Emergência Arquitetural.

ABSTRACT

The main objective of this document is to present an exploratory study regarding aspects of the architecture emergence that occurs along the software development process among teams that adopt agile development process models. In traditional methods, the requirements are obtained from extensive and exhaustive work, and the architecture design is defined and specified to meet this set of comprehensive and consolidated requirements. In the agile methods, it´s not possible to incorporate the traditional way of the software architecture design work, since there is no plan to collect a full set of requirements. The requirements are considered in rounds according to the stakeholders' wishes and main needs. So, the architecture design is defined while the requirements are considered with the belief that the architecture design will "emerge" from the different sets of requirements that are considered to develop. Along those lines, this document presents: i) ways to design full software architecture up front; ii) how agile software development teams work out the architecture design; iii) what practices agile software development teams use when they work in software development projects; iv) how the agile principles can influence the architectural design and how the risks of working with an "emerging" architecture can be managed; and v) software project management practices considering architecture emergence in software projects and teams employing agile methods.

Keywords: Software development process, Agility, Architectural Emergency.

LISTA DE FIGURAS

Figura 1: modelo 4+1 (KRUCHTEN, 1995. p. 2)..... 16

Figura 2: Atividades arquitetura de software (HOFMEISTER et al., 2005. p. 3)..... 18

Figura 3: fatores que influenciam na emergência da arquitetura desejada ao longo do desenvolvimento de software aplicando refatoramento contínuo (CHEN e BABAR, 2014. p. 144)..... 23

Figura 4: comparação entre as estratégias e posturas frente a acumulação de dívida técnica e completude do refatoramento (MARTINI, BOSH, CHAUDRON, 2015. p. 245) 33

SUMÁRIO

1. INTRODUÇÃO 9

2. VISÃO GERAL 12

3. ATIVIDADES DA ARQUITETURA TRADICIONAL E DOS AGILISTAS 15

4. A EMERGÊNCIA DA ARQUITETURA 22

4.1. FATORES DO PROJETO 24

4.2. FATORES DA EQUIPE DE DESENVOLVIMENTO 25

4.3. FATORES PRÁTICOS 26

4.4. FATORES ORGANIZACIONAIS 27

5. O PROBLEMA DA DÍVIDA TÉCNICA E REFATORAMENTO 29

6. CONCLUSÃO 34

REFERÊNCIAS 36

1. Introdução

Dado o contexto em que necessidades humanas serão atendidas por meio de um sistema em que a solução exige o desenvolvimento de um software, uma estratégia é dividir o problema fim em problemas menores e designar a responsabilidade do desenvolvimento de cada subproblema a equipes diferentes que adotam modelos de processo de desenvolvimento de *software* que seguem os princípios conforme descrito no manifesto de desenvolvedores que chamaremos aqui de "agilistas":

> Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através deste trabalho, passamos a valorizar:
>
> Indivíduos e interações mais que processos e ferramentas;
>
> Software em funcionamento mais que documentação abrangente;
>
> Colaboração com o cliente mais que negociação de contratos;
>
> Responder a mudanças mais que seguir um plano;
>
> Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.
>
> (BECK et al., 2001)

O que pode ser observado é que os agilistas explicitam certos aspectos que são considerados de maior ou menor valor. Um aspecto que não é explicitado acima e que discutiremos mais à frente é a chamada Arquitetura de Software. A arquitetura corresponde à estrutura e comportamento do software. A arquitetura de software é um elemento tanto do "software em funcionamento" quanto da chamada "documentação abrangente".

Em função dos agilistas valorizarem o “software em funcionamento” mais do que “documentação abrangente” evidencia-se um confronto potencial: a arquitetura de software, a cada entrega, deve levar em conta apenas os valores visíveis pelos usuários e clientes ou deve considerar outros elementos? A partir desse cenário, o seguinte aspecto deve ser considerado: deixar que a arquitetura apareça em tempo de desenvolvimento no decorrer das entregas de software, deixando-a “emergir” durante o ciclo de vida do projeto de software? Ou planejar e definir a arquitetura tendo em vista aspectos que vão além do contexto de uma entrega?

Esse dilema, nos extremos, é definido pelas expressões (i) *YAGNI (you ain´t gone need it)*, ou seja, use a arquitetura mínima que atenda a entrega atual e (ii) *BDUF (big design up front)*, ou seja, use a arquitetura que leve em conta tudo que possa ser considerado. Para analisar essa questão, elaboramos um estudo, relatando os argumentos de vários autores, compilando e confrontando as suas opiniões, descobertas e visões a respeito de aspectos do trabalho de planejamento e da emergência da arquitetura de software durante o processo de desenvolvimento, empregando equipes que adotam modelos de processo de desenvolvimento dos agilistas.

Especificamente, este texto:

- expõe formas de desenhar a arquitetura de software plena antes de iniciar a etapa de codificação;
- apresenta conceitos, princípios e trabalhos relacionados a desenvolvimento de software quanto ao trabalho de arquitetura, empregando equipes que adotam modelo de processo de desenvolvimento dos agilistas;
- relaciona práticas dos métodos de desenvolvimento ágil quando empregadas em projetos de desenvolvimento de software;
- apresenta como os princípios dos métodos de desenvolvimento ágil podem influenciar no desenho da arquitetura e os riscos de se trabalhar com uma arquitetura “emergente”;
- descreve práticas de gestão de projeto de software em função da emergência da arquitetura em projetos de software quando executados por equipes que empregam métodos ágeis.

A motivação para a realização do trabalho, de forma geral, partiu do questionamento de quanto esforço é necessário para detalhar o desenho da arquitetura no início de um projeto de desenvolvimento de software quando há a utilização de métodos ágeis (WATERMAN, NOBLE e ALLAN, 2015), uma vez que, dada a instabilidade dos requisitos, entre outros fatores definidos por este autor, a arquitetura poderá mudar em maior ou menor extensão com o passar do tempo. Este estudo contribui, portanto, para esclarecer e orientar as partes interessadas no projeto de software sobre esforços que são necessários no início do projeto, mas também, durante a execução do mesmo e quando estiver em sua fase de evolução.

O método de trabalho para o desenvolvimento dessa monografia foi um estudo exploratório de trabalhos científicos, relatórios técnicos, trabalhos apresentados em conferências e artigos publicados em revistas especializadas. O critério para a escolha dos artigos foi baseada na análise do conteúdo e fundamentos dos modelos de processos ágeis e tradicionais, observando aspectos da arquitetura emergente e a convergência, tangibilidade e aplicabilidade na rotina de equipes de desenvolvimento de software.

2. Visão Geral

A engenharia de sistemas inclui uma abordagem interdisciplinar com a finalidade de possibilitar a produção de produtos e serviços envolvendo o desenvolvimento de software de sucesso. Quando sistemas dependem de softwares que são desenvolvidos por equipes que adotam modelos de processo de desenvolvimento ágil, há desafios em como coordenar o legado da abordagem tradicional com aquelas adotadas pelos agilistas (ROSSER et al., 2014).

Sommerville discute três modelos de processos de software: (i) modelo em cascata, que é um modelo referido como tradicional; (ii) desenvolvimento incremental; e, (iii) desenvolvimento de software orientado a reuso. Todos eles compartilham as seguintes atividades fundamentais: especificação; desenho e codificação; validação; e, evolução (SOMMERVILLE, 2011).

Nos modelos de processo tradicionais, as etapas do projeto são bem definidas e uma inicia após o término da etapa imediatamente anterior. Nessa abordagem, se for detectado algum erro na atividade corrente, é necessário rever as decisões tomadas anteriormente, revisar e corrigir os artefatos produzidos. Esses eventos podem causar atraso na entrega do produto e, consequentemente, no atendimento das necessidades das pessoas (WATERMAN, NOBLE e ALLAN, 2015).

Nos modelos de processo dos agilistas, há a valorização da interatividade, necessidades atendidas, a facilidade de reagir a mudanças e, quando aparecem requisitos previamente não considerados, que impactam na arquitetura do software, pode exigir que boa parte do que já está pronto seja refeito, causando aumento de custo não previsto no orçamento (WATERMAN, NOBLE e ALLAN, 2015).

Em relação a essas duas vertentes, os agilistas diriam que a arquitetura de software é "emergente" e que, não necessariamente, deve ser investido muito esforço no planejamento da arquitetura logo de início do projeto e que muito planejamento poderá ser um desperdício de esforço (COPLIEN, 2015). Contudo, a proposta desse autor é incluir no processo de desenvolvimento ágil os princípios do método *LEAN*, desenvolvido pela TOYOTA, que consiste em priorizar o planejamento da arquitetura do software logo de início do projeto. Mas existe a consideração sobre o quanto de esforço gastar nesse planejamento inicial, levando em consideração fatores que influenciam no dia a dia das equipes, como por exemplo, a instabilidade dos requisitos e as respectivas respostas às mudanças, conforme descrito por Waterman, Noble e Allan (2015).

Seguindo a mesma linha de Jim Coplien, a fim de tentar propor uma abordagem que coloca os métodos tradicionais em sinergia com os métodos ágeis, Grundy et al. (2011) cita a possibilidade de um meio termo, como por exemplo, incluir nos requisitos de software (considerados na forma de "histórias de usuários"), os atributos de qualidade que serão atendidos à medida que o software evolui.

Pode-se também levar em consideração as Leis de Lehman e as respectivas práticas sugeridas por Lehman e Ramil (2001), como por exemplo, a rotina de manter uma documentação atualizada com relação às decisões tomadas em resposta à primeira lei que fala que o software tem que se adaptar às mudanças do contexto em que o mesmo está inserido a fim de evitar que se torne obsoleto.

Assim, cada equipe de desenvolvimento, seguindo uma arquitetura predefinida, adequada e sujeita a mudanças, terá condições de produzir módulos integráveis e, após a integração desses módulos, obter como resultado um todo melhor gerenciável a fim de minimizar os esforços de evolução do software. Outro aspecto a ser considerado no desenvolvimento e manutenção do software é relacionado aos problemas de perda de boas características ou inclusão de aspectos indesejáveis, a chamada "erosão arquitetural". Para tornar este trabalho menos complexo e viável, optamos por não incluir tais discussões.

Sob a perspectiva de dividir o problema em problemas menores e mais simples, este estudo exploratório está organizado de forma a apresentar as práticas de desenho da arquitetura de software e do modelo de processo de desenvolvimento ágil relacionando conceitos, princípios e trabalhos relacionados a estes assuntos.

3. Atividades da arquitetura tradicional e dos agilistas

A definição de arquitetura de software pode ser resumida, a partir da ISO/IEC 42010, como a estrutura de componentes do software, o relacionamento entre estes componentes e com o ambiente e, os princípios que servem de orientações de como o software vai evoluir (ELONTRA, 2015).

O trabalho de arquitetura de software compreende, segundo esse mesmo autor, em:

- Desenho e documentação da arquitetura de software;
- Avaliação da arquitetura de software;
- Gerenciamento do conhecimento da arquitetura de software;
- Decisões arquiteturais: decidir e documentar escolhas por determinadas soluções e não por outras.

Os objetivos desse trabalho de arquitetura de software é prover processos, procedimentos e ferramentas para satisfazer requisitos de qualidade de software como, por exemplo, escalabilidade, extensibilidade ou desempenho, através de decisões, bem como relembrar e comunicar às partes interessadas essas decisões (ELONTRA, 2015). Menascé, Almeida e Dowdy (2004) destacam, além desses atributos de qualidade, a disponibilidade, confiabilidade e segurança.

Ao longo do ciclo de vida do software, é criado o conhecimento arquitetural que consiste no registro das decisões arquiteturais e do desenho arquitetural resultante (ELONTRA, 2015). Esse conhecimento poderá estar em acordo com a arquitetura desenvolvida no código fonte ou não. Quando há inconformidades é necessário atuar de forma a minimizar essa diferença e até mesmo eliminá-la antes que isso impacte negativamente na qualidade do projeto. Por exemplo, a velocidade com a qual uma equipe que segue os princípios dos agilistas gera valor ou responde às mudanças não

pode diminuir ao longo do ciclo de vida do projeto, conforme afirmam Martini, Bosch e Chaudron (2015).

Há algumas abordagens para concepção da arquitetura de software:

- Modelo 4+1 (KRUCHTEN, 1995): este modelo descreve como documentar a arquitetura de software apresentando informação similar, mas, sob pontos de vistas diferentes, dependendo dos interesses de cada grupo de partes interessadas.

  A partir de um cenário ou de um caso de uso, organizam-se estruturas nas seguintes perspectivas ou visões para registrar e comunicar as decisões arquiteturais: i) visão lógica; ii) visão de desenvolvimento; iii) visão de processo; iv) visão física.

  Este modelo está representado na Figura 1.

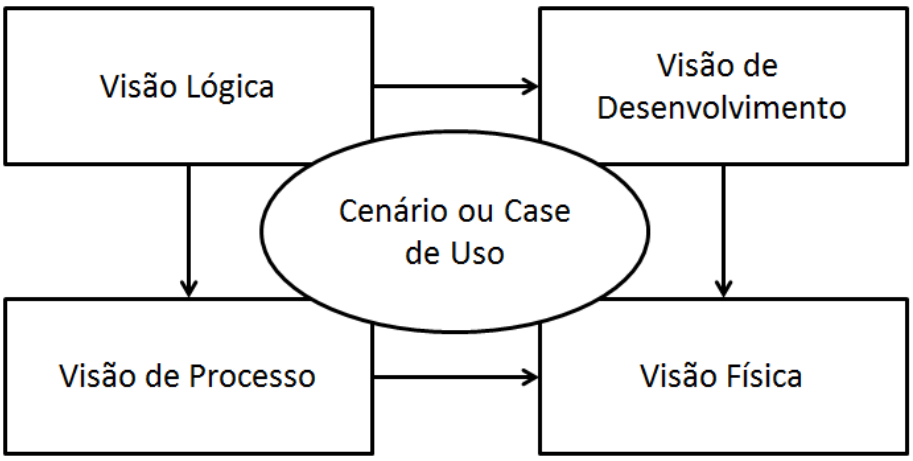

Figura 1: modelo 4+1 (KRUCHTEN, 1995. p. 2)

Visão Logica: modela os requisitos funcionais, comportamentos e estrutura de dados, sendo uma perspectiva necessária para comunicação entre as partes interessadas, gerentes de projetos e equipes de desenvolvimento a respeito do que o software irá fazer e o que ele não irá fazer.

Visão de desenvolvimento: nesta visão são representados os módulos, pacotes, bibliotecas e subsistemas, o relacionamento entre cada uma dessas estruturas e como estão organizadas no ambiente de desenvolvimento. Representa a visão do cenário ou do caso de uso de forma a dar suporte à equipe de desenvolvimento.

Visão de Processo: o software é dividido em unidades e cada uma delas executa um conjunto de tarefas que, quando executadas, irão operacionalizar o respectivo processo ao qual dá suporte. Esta visão comunica também como cada processo interage com os outros.

Visão Física: a partir dos interesses não funcionais como disponibilidade, tolerância a falhas, desempenho e escalabilidade, a visão física é importante para auxiliar no trabalho de desenvolvimento, testes e integração de sistemas.

- Gabarito para a documentação das decisões arquiteturais (VAN VLIET, 2008): enquanto o modelo 4+1 de Kruchten (1995) representa a solução escolhida para o desenho da arquitetura, Vliet (2008) propõe que seja feito o registro das decisões arquiteturais, ou seja, por que foi escolhida uma alternativa e não outra. Estes documentos podem ser organizados da seguinte forma:

    - Tópico de decisão;
    - Decisão;
    - Pressupostos;
    - Alternativas;
    - Implicações;
    - Decisões Relacionadas.

- Processo para planejar, registrar e avaliar o desenho de arquitetura de software (HOFMEISTER et al., 2005): esta é uma compilação de outros de processos e procedimentos usados na indústria de software cujo foco é o detalhamento, desenvolvimento, registro da avaliação da arquitetura de software a partir de requisitos arquiteturais, ou seja, aqueles requisitos que influenciam nas decisões arquiteturais. Esse modelo de Hofmeister (2005) está representado na Figura 2.

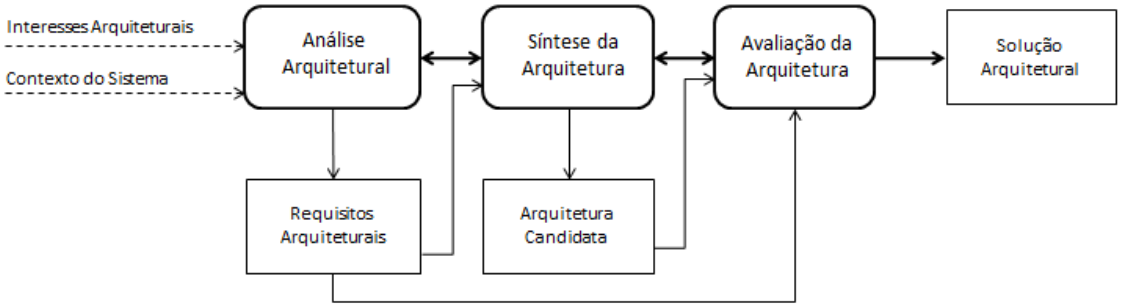

Figura 2: Atividades arquitetura de software (HOFMEISTER et al., 2005. p. 3)

Yang, Liang e Avgeriou (2015) afirmam, baseado em um estudo de mapeamento sistemático de conhecimento sobre a combinação entre práticas de trabalho de arquitetura e as praticas agilistas (YANG, LIANG e AVGERIOU, 2015), que há as seguintes etapas no trabalho de arquitetura, além dessas mostradas da Figura 2:

- Implementação arquitetural;
- Manutenção e evolução da arquitetura;
- Recuperação arquitetural;
- Entendimento arquitetural;
- Análise de impacto arquitetural;
- Reuso arquitetural;
- Refatoramento arquitetural.

Para Babar (2009) essas abordagens tradicionais no trabalho de arquitetura são atividades onerosas e, para Martini, Bosch e Chaudron (2015), podem causar menos agilidade ao longo do ciclo de vida do produto de software.

A proposta agilista é que ocorra a entrega contínua, com geração de valor para as partes interessadas o mais cedo possível, adaptação a mudanças e preferência em ter software funcionando ao invés de documentação abrangente. Com a presença do usuário final, personificada pelo proprietário do produto (*product owner)*, no caso da metodologia SCRUM, é possível elucidar os requisitos em tempo de desenvolvimento e obter um retorno a respeito do que já foi entregue. Neste retorno, é possível que ocorra alteração, adição, remoção de requisitos. As entregas e a consequente integração desses novos módulos ao todo são garantidas estar em conformidade com os interesses e requisitos do software através de atividades de testes, preferencialmente, automatizados e planejados antes do desenvolvimento, em concomitância com as atividades de desenvolvimento (WATERMAN, NOBLE e ALLAN, 2015).

Percebe-se, portanto, que quando um software é desenvolvido, aplicando princípios de desenvolvimento ágil, o fenômeno da emergência se entrelaça com atividades de planejamento da arquitetura e, para os agilistas isso pode ocorrer de várias formas. A seguir são apresentadas práticas agilistas no desenvolvimento de software e como que elas abordam o trabalho de planejamento e desenho da arquitetura de software.

A abordagem agilista vem tomando preferência em muitas empresas a fim de possibilitar redução de custo e aumentar a capacidade das equipes de desenvolvimento de absorverem às mudanças de requisitos e de interesses (BABAR, 2009). Contudo, essas mudanças de requisitos, que impactam na arquitetura, podem ser realizadas com menor esforço se executadas logo no início do projeto. Ou seja, ao longo do ciclo de vida do software as mudanças arquiteturais aumentam de custo (ELONTRA, 2015).

Abaixo são itemizadas algumas das práticas dos agilistas (BABAR, 2009):

- Iteração;
- Planejamento da iteração;
- Revisão da Iteração;
- Reunião diária

- Refatoramento;
- Desenho simples;
- Padrões de codificação;
- Propriedade e responsabilidade coletiva do código fonte;
- Programação em pares;
- Desenvolvimento dirigido a teste;
- Reuniões pós-planejamento (*Post Game Sessions*);
- 40 horas semanais de trabalho;
- Disponibilidade e presença do cliente sempre que o desenvolvedor necessitar.

Elontra (2015) afirma que o planejamento e registro do desenho da arquitetura de software podem ocorrer seguindo quatro padrões. Segundo este autor, a equipe de desenvolvimento, que emprega o modelo SCRUM, pode adotar uma das seguintes abordagens quanto ao trabalho de arquitetura:

- *Big Up-Front Architecture (BUFA):* a partir dos requisitos de software do produto, planejar e desenhar a sua respectiva arquitetura de forma completa, definindo diretrizes arquiteturais que irão conduzir as decisões durante o desenvolvimento;
- *Sprint Zero (SPR0)*: reservar as iterações iniciais para o planejamento e registro do desenho da arquitetura;
- Desenhar a arquitetura à medida que é necessário com *sprints* exclusivos para isso e executado pela equipe de desenvolvimento (SPR*);
- Semelhante à abordagem anterior, há o SEPA, que é um processo de arquitetura separado, executado por uma equipe diferente da equipe de desenvolvimento.

Observam-se, contudo, questões entre as práticas agilistas do SCRUM e esses padrões de atividades do trabalho de arquitetura. Se a arquitetura for concebida de forma completa no início do projeto de software ou por uma equipe diferente da equipe de desenvolvimento, é preciso mapear como que esse conhecimento irá passar de uma equipe para outra (ELONTRA, 2015). Contudo, conforme afirma Waterman, Bosch e Chaudron (2015), estratégias como planejamento de mitigação de risco e o desenho arquitetural definido de forma completa, logo no início do projeto de software, minimizam prejuízos causados caso eventos indesejados ocorram. Observa-se que as práticas SPR0, SPR* e SEPA exigem menos esforço logo de início do projeto de software e são aderentes às estratégias de adaptação a mudanças, entrega de valor mais cedo e arquitetura emergente, segundo Waterman, Bosch e Chaudron (2015).

4. A emergência da arquitetura

Arquitetura emergente produz uma arquitetura na qual a equipe de desenvolvimento realiza o mínimo de decisões logo de início do projeto, como por exemplo, selecionar as tecnologias, estilos e padrões arquiteturais que serão utilizados. A equipe de desenvolvimento considera apenas os requisitos necessários para o desenho da arquitetura, ignorando, mesmo em alto nível, aqueles que serão desenvolvidos no longo prazo. Utilizar-se dessa estratégia ajuda a garantir desenho simples, conforme os agilistas pregam, e o produto que está sendo desenvolvido pode ser distribuído no mercado e começar a gerar valor o mais rápido possível. Além disso, trabalhar com arquitetura emergente é tipicamente usado quando se adota a prática de desenvolver o mínimo do produto, mas suficiente para ele ser lançado no mercado (WATERMAN, NOBLE e ALLAN, 2015).

Contudo, esses autores, também afirmam que, se o produto de software demandar uma solução arquitetural mais complexa para atender requisitos arquiteturalmente significativos, envolvendo vários pontos de integração entre sistemas legados, componentes de terceiros, sistemas distribuídos, entre outros, será necessário mais esforço no desenho da arquitetura logo de início do projeto a fim de criar planos de contingências para tratar as incertezas caso elas ocorram.

Chen e Babar (2014) afirmam que uma característica comum entre SCRUM, Crystal Clear e XP é que a arquitetura, desenvolvida logo no início do projeto para dar suporte às funcionalidades, pode ser refatorada para que ela atinja o nível desejado de forma incremental.

Aqueles que advogam pela aplicação das abordagens de desenvolvimento de software ágil defendem que a arquitetura desejada do software emerge da prática de

refatoramento constante (CHEN e BABAR, 2014). Uma vez que não há produção de software funcionando e gerando valor quando se gasta muito esforço logo de início do projeto em atividades de planejamento e de desenho da arquitetura, desenvolver o mínimo viável e ajustar periodicamente ao longo do desenvolvimento é uma alternativa.

A partir da prática de refatoramento constante durante o ciclo de vida do projeto, para que a arquitetura desejada apareça ao longo do desenvolvimento, depende de vários fatores, que estão categorizados conforme representado na Figura 3 e, descritos a seguir, conforme abordado por Chen e Babar (2014).

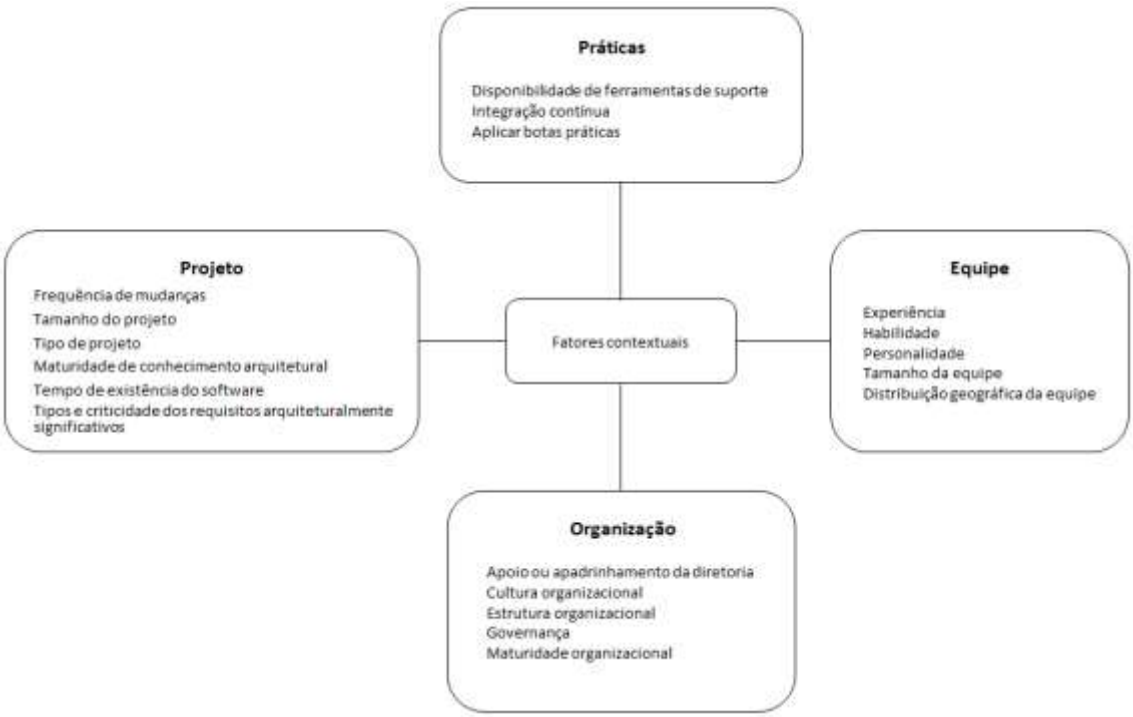

Figura 3: Fatores que influenciam na emergência da arquitetura desejada ao longo do desenvolvimento de software aplicando refatoramento contínuo (CHEN e BABAR, 2014. p. 144) [tradução nossa]

4.1. Fatores do projeto

Os fatores relacionados ao projeto, segundo Chen e Babar (2014), são características do projeto que influenciam se o aparecimento da arquitetura desejada irá ocorrer de forma satisfatória ou não. Se a taxa de mudanças dos requisitos arquiteturais é muito alta ou se muitas das funcionalidades foram desenvolvidas antes de capturar e elucidar esses requisitos arquiteturais, dificilmente a arquitetura desejada será obtida através de atividades de refatoramento constante. Por outro lado, se a taxa de mudança desses requisitos arquiteturais for pequena, atividades de refatoramento constante podem gerar trabalho extra desnecessário para atingir a arquitetura desejada.

Dependendo do tamanho do projeto ou da quantidade mínima requerida de funcionalidades desenvolvidas para que se tenha um produto viável de ser lançado no mercado, torna-se mais difícil ocorrer o "aparecimento" da arquitetura durante o desenvolvimento do software, aplicando refatoramento constante. Projetos pequenos favorecem o "aparecimento" da arquitetura desejada durante o desenvolvimento do produto de software através de atividades de refatoramento constante.

Uma maior maturidade e experiência da equipe de desenvolvimento em relação ao conhecimento de arquitetura de software, como padrões e estilos arquiteturais, permitem que se tenha ideia de forma intuitiva de como iniciar e qual o caminho deve ser trilhado para que se obtenha a arquitetura desejada ao longo do desenvolvimento aplicando atividades de refatoramento constante. O fator da tecnologia utilizada ser mais atual influenciará positivamente na obtenção da arquitetura desejada, utilizando-se de refatoramento.

Em relação aos tipos de requisitos arquiteturais, há duas tendências: requisitos relacionados com segurança precisam de maior esforço logo para planejar e desenhar a arquitetura logo de início. Por outro lado, se o requisito condutor for grau de manutenibilidade, por exemplo, é um facilitador para que a arquitetura desejada "apareça" ao longo do desenvolvimento aplicando atividades de refatoramento constantemente.

Pode-se concluir a partir das afirmações de Chen e Babar (2014), que em projetos com as características listadas a seguir, o aparecimento da arquitetura desejada é favorecido com a prática de refatoramento contínuo:

- pequenos projetos com pequeno número de funcionalidades que devem ser desenvolvidas logo de início para que o produto de software seja viável para gerar valor;
- executados por uma equipe de desenvolvimento com maior maturidade e experiência em relação ao conhecimento de arquitetura de software como, padrões e estilos arquiteturais;
- utilizar-se de tecnologia mais atualizada;
- e, cujo requisito condutor da arquitetura seja a manutenibilidade.

Outro fator relacionado a projeto é a taxa de mudança de requisitos. Chen e Babar (2014) também afirmam que, se os requisitos são muitos instáveis, dificilmente a arquitetura desejada será obtida através de atividades de refatoramento constante. Contudo, Diaz et al. apud. Yang, Liang e Avgeriou (2015) propõem que seja feita uma análise de impacto das mudanças para permitir que a equipe de desenvolvimento possa fazer com que a arquitetura evolua para o desejado de forma iterativa e incremental, seguindo os princípios de flexibilidade e de adaptabilidade do agilismo.

4.2. Fatores da equipe de desenvolvimento

Chen e Babar (2014) descrevem como fatores relacionados à equipe de desenvolvimento, o conhecimento tácito adquirido pela equipe na prática de refatoramento, a personalidade dos indivíduos, a quantidade e a distribuição geográfica das pessoas que integram a equipe do projeto.

A senioridade da equipe de desenvolvimento em alguma tecnologia e a familiaridade com o domínio do negócio colabora para a equipe entender o problema, fazendo com que uma arquitetura adequada “apareça” ao longo do desenvolvimento do software.

Quanto à personalidade dos integrantes da equipe, pode-se observar que as pessoas que tendem a ser mais colaborativas são mais aptas a fazer com que a arquitetura desejada emerja em tempo de desenvolvimento. O mesmo é válido para equipes menores e que trabalham próximas.

4.3. Fatores práticos

Os fatores práticos, segundo Chen e Babar (2014), compreendem em ter disponível ferramental para diminuir esforço como testes automatizados para garantir que foi realizado um ajuste ou adequação no desenho da arquitetura codificada e que não houve impacto no comportamento ou introdução de inconformidades com os requisitos já desenvolvidos. A integração contínua e o uso de boas práticas de desenho de arquitetura também fazem parte desse grupo.

Ao refatorar, deve-se alterar a estrutura do software de forma a diminuir o desvio entre o que está implementado e o desejado. Mas, não deve alterar o comportamento do software. Para garantir que se tenha realizado o refatoramento e não tenha introduzido inconformidades com os requisitos funcionais ou mesmo quebra de compatibilidade entre os componentes, é necessário testes de unidade e de integração. Como as atividades de testes podem atingir altos custos, o teste automatizado é um recurso que aumenta as chances de sucesso do projeto.

A integração contínua e, o teste de integração, são chaves para que se obtenha uma arquitetura emergente através de refatoramento contínuo.

Chen e Babar (2014) também afirmam que boas práticas de desenho de arquitetura são formas de ajudar a obter a arquitetura desejada, sendo que o uso de boas práticas e testes de unidade são atividades que ser complementam. Como exposto anteriormente, o teste de unidade verifica se é obtido algo com um comportamento diferente do esperado quando se faz uma intervenção na estrutura. O teste de integração, a cada refatoramento, segundo esses autores, também minimiza o trabalho de reparos causados por quebra de compatibilidade entre os sistemas.

4.4. Fatores organizacionais

Por fim, os fatores organizacionais compreendem em priorizar tanto a agilidade quanto o trabalho de arquitetura, criar canais de comunicação eficientes, encorajar o comprometimento e o sentimento de propriedade do código por parte da equipe de desenvolvimento, evitar que as pessoas se sentem constrangidas com os erros e a estrutura organizacional (CHEN e BABAR, 2014).

O apoio da gerência em priorizar as atividades relacionadas a melhorar a arquitetura do software é fundamental para que se obtenha a arquitetura desejada. O que ocorre é que há a pressão de se desenvolver e disponibilizar funcionalidades em detrimento do trabalho para fazer a arquitetura evoluir; isso fará com que a equipe não se sinta confortável em reservar esforço para refatorar o código de forma a melhorá-lo.

A cultura organizacional é um fator que, para aumentar a probabilidade da arquitetura desejada aparecer em tempo de desenvolvimento, compreende em dispor-se de bons canais de comunicação, incentivar a propriedade coletiva do código e não fazer as pessoas se sentirem constrangidas em função dos potenciais erros.

A estrutura organizacional não pode ser muito engessada de forma a inibir as práticas agilistas. Muito pelo contrário, ela deve deixar fluir abertamente essas práticas para facilitar que a arquitetura desejada “apareça” em tempo de desenvolvimento.

Governança e maturidade organizacional em desenvolvimento de software faz com que sejam definidos padrões que devem ser seguidos pelas equipes de desenvolvimento. Por exemplo, Chen e Babar (2014) constataram que se a gestão de uma organização não priorizar uma transformação na arquitetura, preterindo as atividades de refatoramento em função de cumprir prazos estabelecidos para a entrega de funcionalidades, a arquitetura desejada não irá emergir durante o desenvolvimento do software. Em consequência disso, Maiti e Mitropouls (2015) afirmam que “negligenciar requisitos não funcionais tem impacto negativo no produto de software por resultar um produto de baixa qualidade e em um custo maior para corrigir problemas em etapas posteriores do projeto” (MAITI e MITROPOULUS, 2015. p. 1) [tradução nossa]. Muitas vezes, segundo Maiti e Mitropoulus (2015) as equipes de desenvolvimento deixa para adequar

a arquitetura no final do projeto quando a diferença entre arquitetura planejada e a desejada é muito grande, gerando um estado de crise no projeto, aumentando o risco do projeto fracassar, devido ao acumulo de inconformidades arquiteturais.

Como se pode observar, no estudo de Chen e Babar (2015), a arquitetura desejada pode "aparecer" ao longo do ciclo de vida de um projeto de software e, dependendo de alguns fatores, esse fenômeno pode ocorrer de forma satisfatória ou não.

5. O problema da dívida técnica e refatoramento

A dívida técnica pode ser definida como a diferença entre a arquitetura projetada e a aquela codificada, podendo causar impactos negativos em relação ao cumprimento das estimativas de tempo e de custo. Se o acúmulo da dívida técnica não for controlado poderá causar problemas no projeto de software, como por exemplo, iterações mais longas ou com menos entrega de funcionalidades ao longo do ciclo de vida do projeto (MARTINI, BOSH e CHAUDRON, 2015).

A captura, elucidação, alteração e surgimento de requisitos e, até mesmo necessidades que deixam de existir, criará demanda de alteração no desenho da arquitetura software, uma vez que o software tem que evoluir a medida que o ambiente de negócio evolui para que ele não perca a sua utilidade (LEHMAM, 2001).

Maiti (2015) afirma que, quanto menos esforço empregado para capturar, elucidar requisitos e, desenhar e avaliar a arquitetura logo de início, maior a chance dos requisitos não funcionais serem identificados completamente e nem de forma consistente ou até mesmo subestimados, o que impactará negativamente no software em função de se produzir uma arquitetura mais distante da desejada.

Além disso, Waterman, Bosch e Chaudron (2015) afirmam que, pelo princípio agilista de entregas rápidas e frequentes de software, existe a possibilidade de menor esforço ser empregado no trabalho de desenho da arquitetura. E quanto menos esforço for empregado neste trabalho, maior a chance de produzir uma arquitetura inadequada em função de requisitos não atendidos.

Além do software produzido com mecanismos criados de forma deficiente para atender certos requisitos, em particular os requisitos não funcionais, há outros exemplos de

dívidas arquiteturais, conforme afirmam Martini, Bosh e Chaudron (2015). Segundo esses autores, há dívida técnica também quando:

- há violação de dependências de componentes de diferentes níveis;
- não conformidade com padrões e políticas definidos no desenho arquitetural;
- duplicação de código;
- propriedades temporais de recursos interdependentes;
- e, demais práticas que são consideradas proibidas.

Martini, Bosch e Chaudron (2015) apresentam fatores causadores de acumulação de dívida técnica, que compreendem em fatores do negócio, como a sua evolução, pouca elucidação dos casos de uso ou história de usuários logo de início do projeto; pressão para entregar soluções de software cada vez mais rápido; priorizar os requisitos funcionais em detrimento dos requisitos não funcionais, desenvolvendo funcionalidades e, adiando as atividades de refatoramento para o final do projeto com o software já em ambiente de uso pelos usuários contribuem para o surgimento da dívida técnica.

No primeiro fator, por exemplo, a evolução do negócio faz com que se decida a desenvolver um novo recurso. Neste caso, faz com que haja acumulação de dívida técnica (MARTINI, BOSCH e CHAUDRON, 2015). Outros fatores, que não estão ligados diretamente com o negócio e que também são citados por esses autores, pode se destacar como:

- escassez de documentação com ênfase em detalhar apenas os requisitos arquiteturais mais críticos, apesar de Elontra (2015) afirmar que o detalhamento desses requisitos, ao longo das iterações, ser uma prática empregada pelo agilistas;
- reutilização de software legado, de terceiros ou de código aberto, apesar de que há equipes que adotam soluções de terceiros para minimizar o tempo de planejamento e desenho da arquitetura, logo de início, como afirmam Waterman,

Bosch e Chaudron (2015) , que é considerado por estes autores um risco técnico, demandando requisitos arquiteturais significantes, como desempenho, segurança e disponibilidade, pelo fato de haver vários pontos de integração;

- equipes desenvolvendo software paralelamente, uma vez que cada equipe terá a responsabilidade de detalhar os requisitos em tempo de desenvolvimento com uso de diferentes padrões para solucionar problemas semelhantes, ainda mais se empregarem a prática de desenhar a arquitetura da forma SEPA, conforme descrita por Elontra (2015), uma vez que a arquitetura do software é desenhada em um processo separado do desenvolvimento das funcionalidades por uma equipe diferente da equipe de desenvolvimento e, portanto, parte do conhecimento arquitetural poderá ser comunicada entre as equipes de forma ineficiente;
- o impacto arquitetural em outras áreas do software, causado por uma intervenção a fim de eliminar a dívida técnica em determinada área bem conhecida, é muito difícil de ser capturado completamente e, com isso, mesmo que elimine o desvio arquitetural em determinadas partes, ainda haverá dívidas técnicas desconhecidas;
- o refatoramento incompleto, postura relacionada com a pressão dos fatores do negócio, garantia de compatibilidade dos componentes não alterados e os efeitos desconhecidos causados por refatoramente em uma área do software, faz com que sempre haja dívidas técnicas remanescentes e com que novas apareçam no software como se pode observar na figura 3;
- a evolução tecnológica de software e hardware faz com que coexistam componentes que empregam novas e antigas tecnologias, gerando algo aquém do ponto ótimo;
- a escassez de conhecimento pode se manifestar como inexperiência e pouco conhecimento do domínio da aplicação da equipe de arquitetos, desenvolvedores e demais analistas e, esses profissionais são mais suscetíveis a criar dívidas técnicas do que os que possuem mais experiência e conhecimento do negócio e, mesmo que haja a gestão do conhecimento arquitetura, a falta de conhecimento e

o descuido em relação aos padrões e políticas definidas pela organização causa a acumulação de dívida técnica.

Equipes de desenvolvimento de software que adotam os princípios agilistas e que adotam a prática de refatoramento faz com que a arquitetura desenvolvida fique mais próxima e consistente com o planejado ao longo do ciclo de vida do projeto de software. Essa estratégia e a sua relação com a gestão da arquitetura dos produtos de softwares produzidos por equipes que adotam métodos que seguem os princípios dos agilistas será abordada a seguir.

As posturas de refatoramento, descritas por Martini, Bosch e Chaudron (2015), com o objetivo de eliminar as inconformidades entre o desenho arquitetural desejado e o que realmente está desenvolvido, são usadas como forma de lidar com a acumulação da dívida técnica. O desenvolvimento de novas funcionalidades do software e os ajustes na arquitetura em algumas partes do código causam o desvio entre arquitetura desejada e a concretizada. Pode-se adotar, portanto, segundo esses autores, as seguintes posturas:

- deixar a acumulação de dívida técnica ocorrer sem realizar refatoramento algum até o momento em que os impactos nos atributos de qualidade começam a ficar perceptíveis.

  Essa postura é uma característica de equipes que adotam o gerenciamento de dívida técnica orientado a crise. Quando a acumulação da dívida técnica começa a inviabilizar o negócio, decide-se em trocar o software ou propor, como um projeto de evolução do software, um refatoramento total.

- Para minimizar o impacto nas metas das iterações seguintes, faz-se um refatoramento parcial ou total antes de iniciar a próxima iteração. Essa estratégia consiste em intercalar atividades de desenvolvimento de funcionalidades e atividades de refatoramento, realizando-as, concomitantemente. Dessa forma, a cada refatoramento, o desenho da arquitetura vai em direção ao desejado.

A Figura 4 compara essas posturas e o impacto na acumulação da dívida técnica em função do tempo.

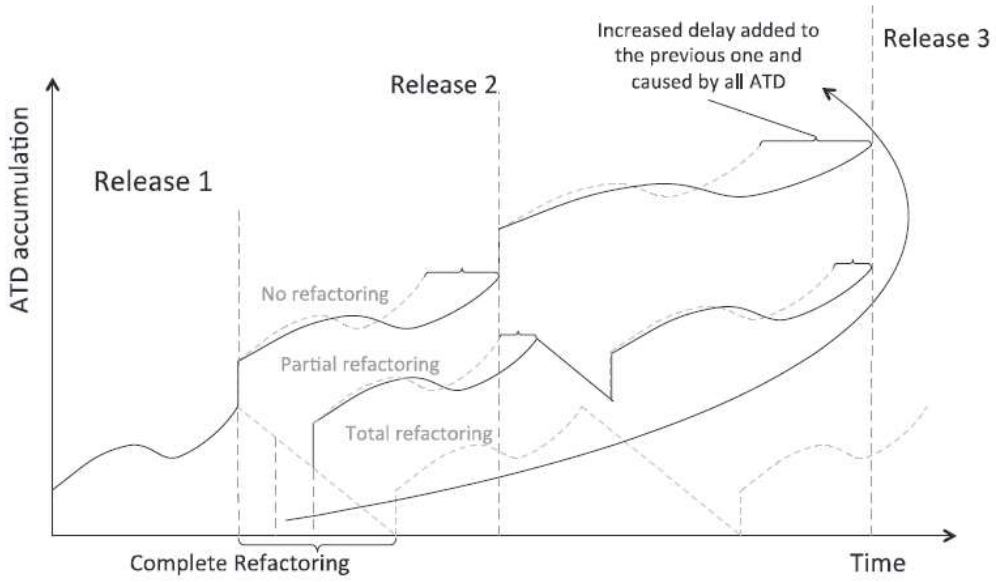

Figura 4: comparação entre as estratégias e posturas frente a acumulação de dívida técnica e completude do refatoramento (MARTINI, BOSH, CHAUDRON, 2015. p. 245)

Podemos sintetizar o problema da acumulação de dívida técnica e o uso do refatoramento como estratégia para eliminar ou diminuir o desvio arquitetural, afirmando que há causas da acumulação de dívida técnica em função de fatores internos e externos a equipe de desenvolvimento, como a pressão para resposta e adaptação rápida ao contexto mercadológico em que a organização ou empresa de software está inserida. E, uma vez sabendo que este fenômeno é intrínseco ao ciclo de vida do projeto de software, pode-se decidir em eliminar parcial ou completamente a dívida técnica ou mesmo decidir em tratar o desvio arquitetural quando o impacto negativo das mesmas começarem a ser observados e já estiverem criando situações de crise como atrasos e aumento de esforço. Por fim, comparando os resultados dessas estratégias aplicadas ao longo do tempo, percebe-se que a prática de eliminar completamente a dívida técnica, de forma frequente, causará na produção de algo mais próximo do desejado, apesar de não ser possível conhecer em sua completude todas as ocorrências de dívidas técnicas no produto de software.

6. Conclusão

Percebe-se que nos princípios dos agilistas há uma valorização maior nos indivíduos, em software em funcionamento, gerando valor, colaborando com o cliente, que se adapta às mudanças em contraposição de processos e ferramentas, documentação abrangente, negociação de contratos e seguir um plano. Isso não quer dizer que há a ausência de processos, ferramentas, documentação, contratos e plano a ser seguido. Sem respostas prontas, sempre dependendo do contexto, como foram discutidos em alguns aspectos, há um espectro onde os polos são compostos de um lado por práticas que seguem os princípios agilistas, por outro, pelas práticas que seguem os princípios dos métodos tradicionais.

Em um primeiro momento foi discutido o processo, procedimentos e artefatos produzidos pelo trabalho de arquitetura. Em seguida, foi levantado como os desenvolvedores, que adotam os princípios agilistas, planejam o trabalho de arquitetura. Esse trabalho pode ter um esforço maior no início do projeto do software, mas, pode também ser diluído ao longo do ciclo de vida do projeto, fazendo com que a arquitetura "apareça" junto com as funcionalidades ao invés de ser produzida em uma etapa separada, como nos métodos tradicionais.

Sob o aspecto em ser mais ou menos agilista, foi apresentado o estudo de Chen e Babar (2015). O fato da arquitetura desejada "aparecer", de forma satisfatória ao longo do ciclo de vida do projeto com a prática de refatoramento contínuo, depende de fatores externos e internos a equipe de projeto do software e podem ser categorizados no nível prático, organizacional, da equipe e do projeto. Dependendo do contexto, será necessário dispender mais esforço logo de início no planejamento e desenho da arquitetura.

Por fim, com o objetivo de manter a conformidade entre a arquitetura desenvolvida e a desejada, foi discutido o problema da acumulação da dívida técnica e como planejar atividades de refatoramento para diminuir ou até mesmo eliminar essas disparidades, dependendo da postura da equipe de decidir realizar o refatoramento no último momento ou realizar essas atividades contínuamente com eliminação parcial ou total da dívida técnica.

Os artigos considerados nesta monografia não cobrem todos os aspectos e questões que podemos observar, mas, discutem uma boa parte do espectro que deve ser considerado. A contribuição deste estudo foi apresentar, portanto, algumas das formas de considerar o planejamento e a emergência da arquitetura quando se desenvolve software seguindo métodos dos agilistas.

REFERÊNCIAS

BABAR, M. A. *An exploratory study of architectural practices and challenges in using agile software development approaches*. In: Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on. Disponível em <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6827119>. Acesso em 31 jan. 2016

BECK, K. et al. *Manifesto for agile software development*. 2001. Disponível em: <http://www.agilemanifesto.org/>. Acesso em 30 set. 2015.

CHEN, L. BABAR, M. A. *Towards an evidence-based understanding of emergence of architecture through continuous refactoring in agile software development*. In: Conference on Software Architecture (WICSA), 2014 IEEE/IFIP. Disponível em <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6827119>. Acesso em 31 jan. 2016

COPLIEN, J. *Why Architecture is needed even in Agile?* : entrevista. Business901. 24 jan. 2011. Disponível em <http://business901.com/blog1/why-architecture-is-needed-even-in-agile/>. 2011. Acesso em 29 set. 2015.

ELONTRA, V. *Techiniques and Practices for Software Architecture Work in Agile Software Development*. 2015. 153 f. Tese (Doutorado)-Tampereen Teknillinen Yliopisto, Tampere University of Techinology, Tampere, FINLAND, 2015. Disponível em <https://dspace.cc.tut.fi/dpub/handle/123456789/22918>. Acesso em 29 set. 2015.

GRUNDY, J. AVGERIOU, P. HALL, J. LAGO , P. MISTRIK, I. *Emerging Issues in Relating Software Requirements and Architecture*. In: _____ Relating Software Requirements and Architectures. Berlim: Springer Berlin Heidelberg, 2011. cap. 17, p. 303-306.

HOFMEISTER, C. KRUTCHEN, P. NORD, R.L. OBBINK, H. RAN, A. AMERICA, P. *Generalizing a Model of Software Architecture Design from Five Industrial Approaches*. In: Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on. Disponível em <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1620093>. Acesso em 08 dez. 2015.

KITCHENHAM, B. CHARTERS , S. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*. EBSE TR-2007-01, Departments of Computer Science, Keele University and University of Durham, UK. 2007. Disponível em <http://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf>. Acesso em 29 set. 2015.

KRUCHTEN, P. *Architectural Blueprints—The "4+1" View Model of Software Architecture*. In: Software, IEEE. vol: 12, issue: 6. nov. 1995. Disponível em <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=469759>. Acesso em 29 nov. 2015.

LEHMAN, M. M. RAMIL, F. J. *Rules and Tools for Software Evolution Planning and Management.* In: Annals of Software Engineering. Vol. 11, 2001.

MAITI, R. R. MITROPOULOS, F. J. *Capturing, Eliciting, Predicting and Prioritizing (CEPP) Non-Functional Requirements Metadata During The Early Stages of Agile Software Development*. SoutheastCon 2015, Fort Lauderdale, FL. Disponível em <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7133007>. Acesso em 20 jan. 2016.

MARTINI, A. BOSCH, J. CHAUDRON, M. *Investigating architectural technical debt accumulation and refactoring over time: a multiple-case study*. Information and Software Technology. 2015. Disponível em <http://www.sciencedirect.com/science/article/pii/S0950584915001287>. Acesso em 31 jan. 2016.

PAULA FILHO, W. P. *Engenharia de software: fundamentos, métodos e padrões.* 3. ed. Rio de Janeiro: LTC, 2009. 1248 p.

ROSSER, L. MARBACH, P. OSVALDS, G. LEMPIA, D. *System Engineering for Software Intensive Projects Using Agile Methods.* In: SEDEC 2014 CONFERENCE. Washington. Disponível em <http://www.sedcconference.org/systems-engineering-for-software-intensive-projects-using-agile-methods/>. Acesso em 13 set. 2015.

SOMMERVILLE, I. *Engenharia de software.* 9. ed. São Paulo: Person Addison-Wesley, 2011. 529 p.

VAN VLIET, H. *Software Architecture Knowledge Management*. In: Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on. Disponível em <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4483186>. Acesso em 29 nov. 2015.

WATERMAN, M. NOBLE, J. ALLAN, G. *How much up-front? A grounded theory of agile architeture.* Vitoria University Of Wellington, School of Engineering and Computer Science. Disponível em <http://ecs.victoria.ac.nz/foswiki/pub/Main/TechnicalReportSeries/ECSTR15-01.pdf>. Acesso em 25 jun. 2015.

YANG, C. LIANG, P. AVGERIOU, P. *A systematic mapping study on the combination of software architecture and agile development*. Journal of Systems and Software. 2015. Disponível em <http://www.sciencedirect.com/science/article/pii/S0164121215002290>. Acesso em 31 jan. 2016.