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

FABRÍCIO BATISTA DE OLIVEIRA

APLICAÇÕES DE ALGORITMOS EVOLUCIONÁRIOS EM ENGENHARIA DE SOFTWARE

Belo Horizonte
2010

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

APLICAÇÕES DE ALGORITMOS EVOLUCIONÁRIOS EM ENGENHARIA DE SOFTWARE

por

FABRÍCIO BATISTA DE OLIVEIRA

Monografia de Final de Curso

Profa. Dra. Gisele Lobo Pappa
Orientadora

Belo Horizonte
2010

FABRÍCIO BATISTA DE OLIVEIRA

APLICAÇÕES DE ALGORITMOS EVOLUCIONÁRIOS EM ENGENHARIA DE SOFTWARE

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

Área de concentração: *Engenharia de Software*

Orientador (a): *Profa. Gisele Lobo Pappa*

Belo Horizonte
2010

RESUMO

O principal objetivo do presente trabalho é apresentar estudos e aplicações baseados na teoria dos Algoritmos Evolucionários (AE). Esta abordagem tem se mostrado muito eficiente para resolver problemas no mundo computacional. A utilização desta técnica vem obtendo bastante sucesso em várias áreas da computação. No caso deste trabalho, o uso desta abordagem será dirigido a analisar resultados de estudos direcionados no sentido de resolver problemas relacionados à disciplina de teste de software. A característica da abordagem dos AEs que tentam imitar a natureza, no sentido de como ela se adapta a determinadas situações para resolver seus problemas, é ideal para resolver os problemas computacionais de alto nível de dificuldade de resolução. No decorrer do trabalho, são feitas algumas definições sobre algumas variações dos AEs. Como os AEs são usados no meio acadêmico e no cenário comercial, as espectativas de aperfeiçoamento e disseminação de seu uso. Um exemplo de utilização dos AEs no mercado de desenvolvimento de software é dado, no final do texto, através da descrição do trabalho de uma empresa criadora de um *framework* para criar cenários de casos de testes.

Palavras-chave: Algoritmos Evolucionários, Programação Genética, Teoria da Evolução das Espécies, Teste Evolucionário.

ABSTRACT

The main objective of this work is to present studies and applications based on the theory of Evolutionary Algorithms (EA). This approach has proved very efficient for solving computational problems in the world. This technique has achieved considerable success in many areas of computing. In this project, using this approach will be taken to analyze the results of studies directed towards solving problems related to the discipline of software testing. The feature of the approach of EAs who try to imitate nature in the sense of how it fits in certain situations to solve its problems is ideal for solving computational problems of high difficulty level of resolution. Throughout his work, some definitions are made on some variations of EAs; Because EAs are used in academic environment and the business scenario, the expectations for improvement and dissemination of its use. An example of using EAs in the market for software development is given at the end of the text in the description of the work of an organization creates a framework for creating scenarios for testing.

Keywords: Evolutionary Algorithms, Genetic Programming, Evolution of Species, Evolutionary Testing.

LISTA DE FIGURAS

LISTA DE SIGLAS

AE	Algoritmos Evolucionários
AG	Algoritmos Genéticos
AP	Aprendizado de Máquina
CST	Classe Sob Teste
CT	Cluster de Teste
EA	*Evolutionary Algorithm*
ETF	*Evolutionary Testing Framework*
IDE	*Integrated Development Environment*
MST	Método Sob Teste
OO	Orientação a Objetos
PG	Programação Genética
SST	Sistema Sob Teste
TE	Teste Evolucionário
TEE	Teoria da Evolução das Espécies
UST	Unidade Sob Teste

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Fornecer software de qualidade para o mercado não é mais considerado uma vantagem competitiva para uma empresa desenvolvedora de software. Empresas precisam ter como premissa básica a obrigatoriedade de manter um processo que garanta a qualidade de seus produtos. A disciplina de Engenharia de Software, desde o final da "crise do software", vem se preocupando com a criação e aperfeiçoamento de abordagens de automatização de teste software para este fim. Ferramentas criadas sob estas abordagens contribuíram e contribuem bastante para que os softwares atendam a mínima qualidade exigida. Abordagens tradicionais contribuem no processo de qualidade de software. No entanto, apresentam limitações que impossibilitam colher melhores resultados, tanto no tempo de criação, quanto na qualidade de criação de seus casos de testes.

Alguns projetos desenvolvidos em grandes instituições de pesquisa como, por exemplo, a NASA, foram baseados em tecnologias inovadoras que permitiram a alcance de resultados muito superiores em relação às antigas abordagens. A NASA, em um de seus projetos, desenvolve estudos sobre a criação de micro-antenas com o objetivo de formar uma grande rede para monitorar a mesosfera da terra por certo período de tempo. O problema em questão demandaria a criação de uma sequência de protótipos para serem avaliados, até que fosse encontrado um que satisfizesse uma série de especificações. Se esta tarefa fosse feita por uma abordagem tradicional, o profissional executor, obrigatoriamente, deveria ter um conhecimento enorme do domínio da aplicação. Além disso, o trabalho empenhado e o tempo gasto para desenvolver estes protótipos seriam enormes. A solução encontrada para o problema foi utilizar a abordagem dos Algoritmos Evolucionários (AE). Os resultados obtidos, neste projeto da NASA, foram muito satisfatórios. Conseguiu-se, em uma quantidade menor tempo, a criação de uma boa quantidade de protótipos que foram avaliados pelo próprio AE.

Os AEs se adaptam muito bem a uma grande variedade problemas computacionais (MITCHEL, 1996). Assim, este trabalho pretende mostrar como os AEs podem contribuir para geração de caso de teste que consigam produzir resultados superiores aos fornecidos pelas abordagens tradicionais.

Baseado nesta idéia, este presente trabalho apresentará a definição dos AEs e suas variações; alguns trabalhos de desenvolvimento de técnicas de teste de software baseados nos AEs; os AEs no cenário comercial; e como eles podem contribuir para criação, para a evolução do teste de software. O texto é finalizado com o caso de uma empresa que desenvolveu e publicou seu trabalho baseado nos AEs.

1.1 Objetivo

Nesta Seção serão apresentados os objetivos gerais e específicos desse trabalho.

1.1.1 Geral

Estudar novos algoritmos e técnicas evolucionárias que possam ser utilizados no desenvolvimento de abordagens de testes de software, visando aprimorar o processo de testes de software.

1.1.2 Específicos

- Estudar os fundamentos essenciais que baseiam a teoria dos Algoritmos Evolucionários;
- Fazer um levantamento de aplicações existentes que utilizam Algoritmos Evolucionários para testes

    Contrastar os resultados obtidos com testes utilizando métodos convencionais e aqueles baseados em Algoritmos evolucionários.

1.2 Justificativa

A motivação para elaboração deste tema de estudo é, especialmente, o fato de que uma das fases mais importantes no projeto de software e que pode garantir que este seja considerado um produto de qualidade é a fase de testes. Em um ambiente de desenvolvimento é percebido que métodos tradicionais não atendem de forma completa a necessidade de cobrir todas as possibilidades de falhas em um software. Este trabalho focará na pesquisa de trabalhos voltados em desenvolver recursos inovadores para o meio corporativo, para o desenvolvimento e aplicação de métodos de testes que atendam a esta necessidade. O resultado obtido neste trabalho poderá ajudar no processo de produção de software de qualidade.

1.3 Estrutura do Trabalho

O presente trabalho se divide em seis seções onde são discutidos, gradativamente, os principais assuntos que compões a idéia central deste estudo.

- A apresentação do trabalho se inicia com uma introdução que faz um apanhado geral do que foi desenvolvido no trabalho;

- A Seção 2 “Algoritmos Evolucionários” apresenta a definição dos algoritmos evolucionários e alguns trabalhos desenvolvidos utilizando esta abordagem;

- A Seção 3, “Teste de software”, descreve algumas das técnicas de teste software disponíveis no mercado de desenvolvimento de software e as novas abordagens que estão surgindo. Ela fala também da historia da crise do software e o que a causou;

- Na Seção 4, “Algoritmos Evolucionários e Teste de software” é abordada a questão dos algoritmos evolucionários e sua utilização na disciplina de teste de

software. Ela descreve as vantagens e desvantagens de algoritmos evolucionários em relação às abordagens tradicionais de testes de software;

- As necessidades, vantagens e dificuldades em se usar AE no cenário comercial são relatadas na Seção 5, “Algoritmos Evolucionários em Ambientes Comerciais”. Nesta seção é exemplificada, por meio de uma empresa, a criação e uso de um *framework* baseado na teoria dos AEs;

- Por fim, na Seção 6, é desenvolvida a conclusão deste trabalho.

2 ALGORITMOS EVOLUCIONÁRIOS

O desenvolvimento desta técnica computacional foi inspirado na "Teoria da Evolução das Espécies"[1] -TEE- proposta originalmente por Charles Darwin, naturalista Britânico do século 19 (RUSE, 1975). Segundo MITCHELL, a determinação teórica dos Algoritmos Evolucionários é baseada na forma biológica de desenvolvimento da vida na natureza. Algoritmos evolucionários simulam, de forma simplificada, as etapas do processo de evolução descritas na teoria Darwiniana, e trabalham sobre operações primárias nativas da TEE (KOZA, 1992),

Seleção: operação responsável por selecionar os cromossomos para reprodução. Quanto mais apto for o cromossomo, mais vezes este poderá ser selecionado para reprodução.

Cruzamento (Crossover): operação responsável por relacionar dois cromossomos de uma população para gerarem descendentes.

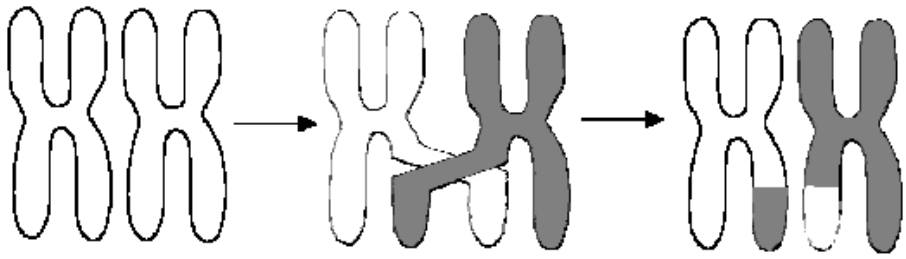


Fig. 1 - Ilustra a troca de material genético entre dois cromossomos
Fonte: (PAPPA, 2010).

Mutação (Mutation): operação responsável por mudar arbitrariamente parte de um cromossomo. Causando anomalias como perda de partes do cromossomo (Deletion) e inversão de partes do cromossomo (Inversion), a **Figura 2** ilustra a operação:

1 - Processo da evolução proposto por Charles Darwin e aceito pelo *mainstream* da comunidade científica como a melhor explicação para a adaptação e especialização dos seres vivos como evidenciado pelo registro fóssil [RUSE].

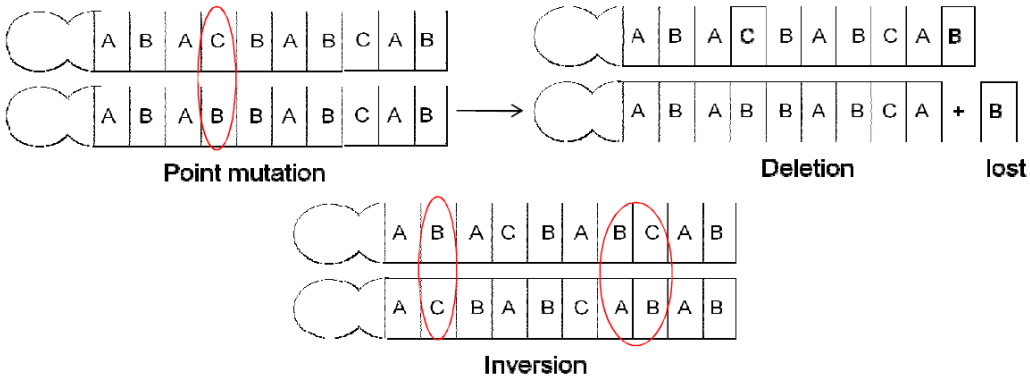


Fig. 2 - Ilustra a operação que efetua a mutação dos cromossomos

Fonte: (PAPPA, 2010).

Tendo como base a **Figura 2** e - em substituindo letras por bits (1s e 0s) para representar os alelos -; poder-se-ia produzir um exemplo de mutação, a fim que a string (0)0010000 possa ser transformada em sua primeira posição para produzir (1)1010000. Este tipo de operação pode ocorrer com certa probabilidade, geralmente muito pequena, em cada posição de bits em uma string (MITCHELL).

Existem vários tipos de algoritmos evolucionários, incluindo algoritmos genéticos, programação genética, estratégias evolucionárias e programação evolucionária. A principal diferença entre esses métodos está na forma que as soluções para os problemas são representadas. No caso de testes de software, cada solução pode representar um possível teste. Porém, enquanto em algoritmos evolucionários elas serão um vetor de bits, em programação genética poderão ser uma árvore.

A **Figura 3** ilustra o fluxo de um AE,

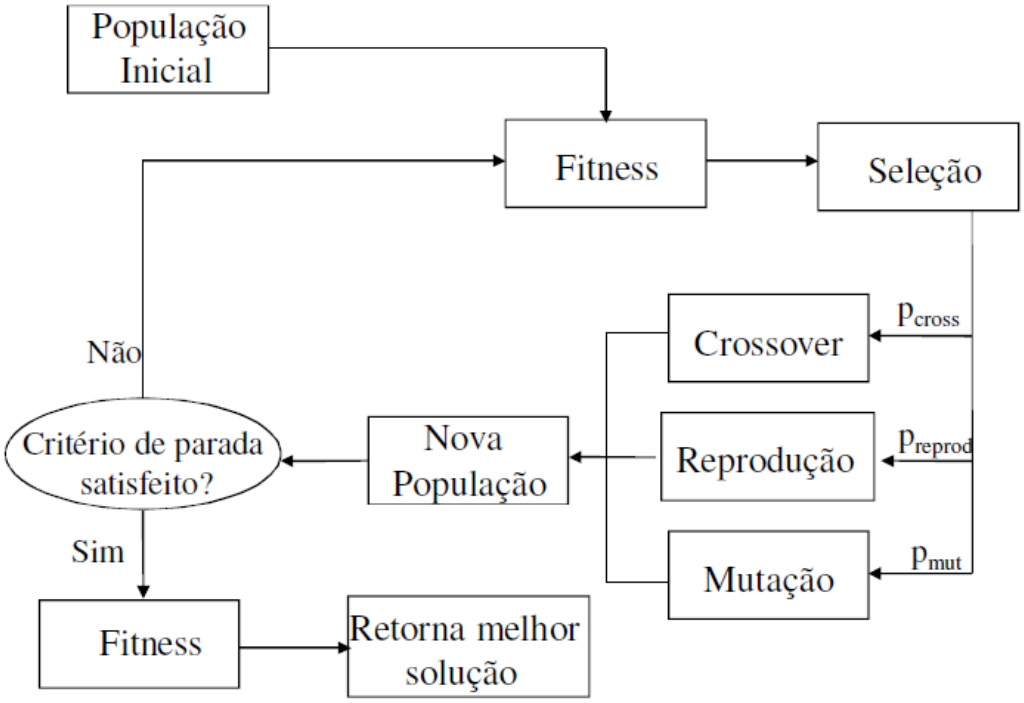


Fig. 3 - Fluxo básico de um AE para evoluir uma população inicial
Fonte: (PAPPA, 2010).

A **Figura 3** ilustra a afirmação de (PAPPA, 2010) que diz que um "AE é um procedimento iterativo que evolui uma população de indivíduos em que cada indivíduo representa uma solução candidata para um dado problema [...]".

Na tentativa de aumentar a confiabilidade e, consequentemente, a qualidade dos softwares utilizando ferramentas de testes, empresas produtoras de software esbarram no problema de como alcançar este objetivo, já que tais ferramentas não conseguem "descobrir" todas as falhas ("bugs") que um software pode esconder. Algoritmos Evolucionários se encaixam perfeitamente no estudo para desenvolvimento de soluções para este tipo de problema – de alta complexidade. Em estudos feitos por MITCHELL, um Algoritmo Evolucionário pode ser adaptar ao cenário de um problema e, a partir de uma série de "evoluções", o algoritmo consegue gerar uma solução, senão a ótima, a solução que mais se aproxima da melhor solução para o problema.

Para se ter uma idéia da complexidade de se desenvolver uma solução para testar um software, uma passagem que está no **Capítulo 9** da **Disciplina de Testes** em PÁDUA FILHO (pág. 349), e que chama bastante a atenção, é o fato de que mesmo para testar um "sistema simples que utilize como entrada um texto de 10 caracteres é necessário 26^{10} combinações". Por esta afirmação, fica evidente que, em testes feitos de forma

convencional, fica praticamente impossível de se detectar vários erros que serão descobertos mais tarde, até mesmo quando o software estiver em faze de produção. Esta é uma situação gravíssima. Quanto mais tarde um erro for descoberto em um software, mais caro se torna sua correção. A situação ainda pode se agravar, como por exemplo, se a quantidade de caracteres usado no texto de entrada for maior que 10. O número de combinações aumenta drasticamente, aumentando, na mesma proporção, a probabilidade de grandes quantidades de erros não serem encontrados. Alguns pesquisadores estão utilizando Algoritmos Genéticos a fim de resolver problemas de alta complexidade similares aos dos testes. Este trabalho será orientado em seguir por este caminho.

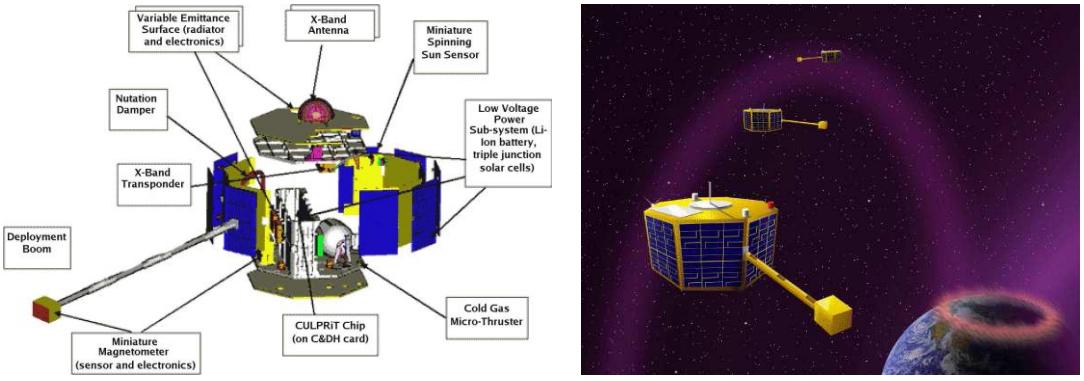
MITCHELL cita em seu artigo "*Genetic Algorithms: An Overview*" algumas áreas em que os Algoritmos Genéticos (e algoritmos evolucionários em geral) são utilizados:

> **Otimização:** AGs tem sido utilizados em uma grande variedade de tarefas de otimização, otimização numérica assim como otimização de problemas combinatoriais tal como formato de circuitos e escalonamento de serviços.
>
> **Programação automática:** AGs tem sido utilizados para desenvolver programas de computador em tarefas específicas, desenhar outras estruturas computacionais como autômatos celulares e pesquisas em redes.
>
> **Aprendizado de máquina:** AGs tem sido utilizados para muitas aplicações de máquinas de aprendizado, incluindo tarefas de classificação e predição, como predição do tempo/clima ou estrutura de proteínas. AGs também tem sido utilizados para envolver sistemas particulares de aprendizado, como por exemplo, largura de uma rede neural. Regras para sistemas classificadores de aprendizagem ou sistemas de produção simbólica e sensores para robôs.
>
> **Modelos econômicos:** AGs tem sido utilizados para modelar processos de inovação, desenvolvimento de estratégias relacionadas e surgimento de mercados econômicos.

Modelo de Sistema imunológico: AGs tem sido utilizados para modelar vários aspectos de sistemas naturais de imunidade, incluindo mutação somática durante o tempo de vida de um indivíduo, descobrindo famílias de multi-gene durante o tempo de evolução.

Modelos ecológicos: AGs tem sido utilizados para modelar fenômenos como a "corrida armamentista biológica", hospedeiros de parasitas, co-evolução, simbioses e fluxos de recursos dentro das ecologias.

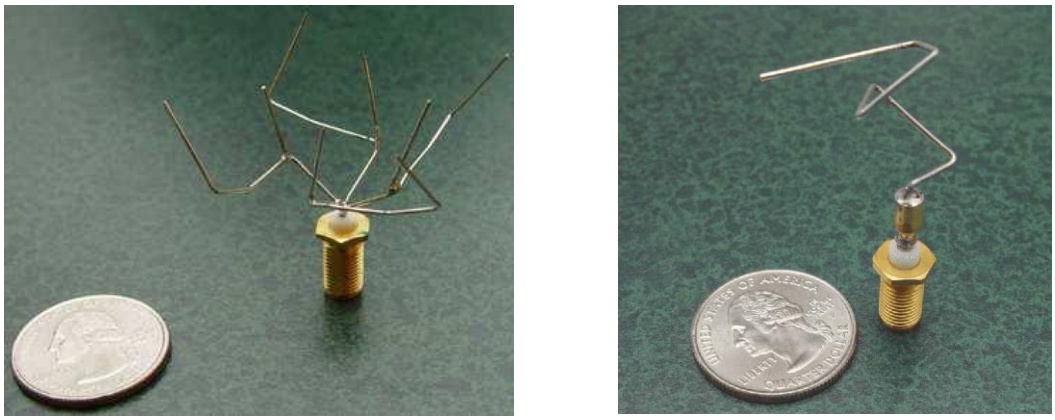
Para se perceber a importância e o poder tecnológico proporcionado pelo uso dos Algoritmos Evolucionários nas várias áreas de conhecimento, outro exemplo pode visto no artigo "*Automated Antenna Design with Evolutionary Algorithms*". Neste artigo, HORNBY e sua equipe, relatam o estudo e a evolução do desenvolvimento de dois modelos de antena para duas missões de exploração espacial pela NASA. A primeira missão, nomeada como ST5, "consiste em três satélites em formato miniatura, chamados de pequenas estrelas, encarregados de medir os efeitos das atividades solares na magnetosfera terrestre por um período de três meses". Veja imagens artísticas do projeto:



(a) (b)

Fig. 4 - Ilustração: (a) Modelo de satélite em miniatura com seus diferentes componentes. (b) Missão ST5 com seus três satélites em formato de corrente de perolas em orbita. Fonte: (HORNBY, 2006).

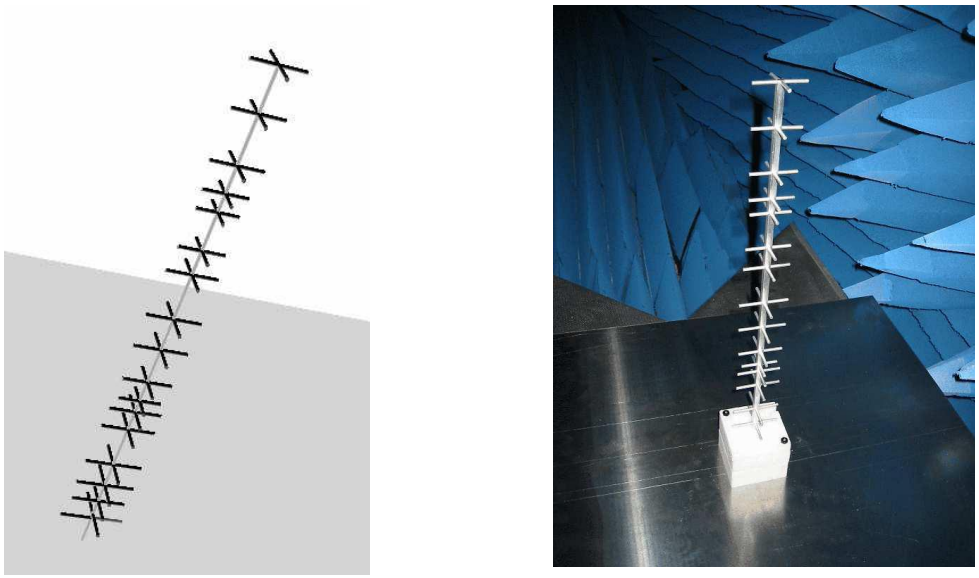
A seguir as imagens dos protótipos desenvolvidos durante os trabalhos.



(a) (b)

Fig. 5 - Protótipo das antenas envolvidas: (a) melhor antena envolvida para o requisito inicial de padrão de ganho; (b) melhor antena envolvida para as especificações revisadas. Fonte: (HORNBY, 2006).

A segunda missão chamada TDRS-C que reúne os requisitos de comunicação exigidos pela NASA está adiantanda para ser lançada na próxima década. O projeto original é composto de dois elementos: o primeiro elemento será incumbido apenas de receber sinais; o outro elemento será responsável por transmitir e receber os sinais. Abaixo segue exemplo da antena:



(a) (b)

Fig. 6 - Melhor resultado do modelo TDRS-C: (a) modelo (b) antena fabricada

Fonte: (HORNBY, 2006).

De acordo com HORNBY as vantagens foram inúmeras em utilizar Algoritmos Evolucionários na construção de antenas para os dois projetos – **TS5, TDRS-C**. Alguns exemplos citados no artigo se direcionam para o ganho de tempo de criação de novos

modelos, testes de estruturas, testes de desempenho. Nos dois últimos casos fica evidente a superioridade do método de criação de antenas utilizando EAs. EAs geram vários modelos de antenas as quais poderão ser, já dentro de processo de "evolução", avaliados se serão ou não modelos aptos a garantir os requisitos do projeto, antes mesmo de serem prototipadas fisicamente. Certamente, um problema bem maior para um especialista em projetar antenas nos moldes convencionais, segundo o autor.

Os resultados dos trabalhos vistos até aqui desenvolvidos com base na teoria dos Algoritmos Evolucionários, tiveram seus resultados melhorados a cada nova etapa do projeto que se seguia, com uma velocidade muito maior do que se observaria em um trabalho que utilizasse métodos tradicionais. Alcançar estes resultados de alto nível de produtividade e assertividade foi possível pelo fato de que, a computação vem desenvolvendo técnicas avançadas como a Programação Genética (PG), que pode ser vista como uma especialização do Algoritmo Genético, onde a codificação dos indivíduos passa a ser capaz de representar programas de computador (VASCONCELOS, 2010). Esta variação dos AGs foi aplicada pelos integrantes dos projetos do satélite ST5 e da antena TDRS-C para que pudesse ser alcançada uma solução exata ou pelo menos aproximada para resolver o problema de achar o melhor formato para antena TDRS-C e a melhor formação para os t mini-satélites da missão ST5 cobrir o espaço se estudado da melhor forma possível.

Estes tipos de projeto que vem sendo bem sucedidos estão respondendo a uma das questões mais importantes da ciência da computação, discutida por SAMUEL,

> "Como os computadores podem aprender a resolver problemas, sem serem explicitamente programados para tal?"

John KOZA relata que um dos principais desafios da ciência da computação é conseguir construir um computador para fazer o que precisa ser feito, sem contá-lo como fazê-lo. A Programação Genética (PG) é definida em GENETIC-PROGRAMMING como um método automatizado para a criação de um programa de computador para trabalhar a partir de uma declaração de um problema de grande dificuldade de resolução. POLI descreve em seu livro "*A Field Guide to Genetic Programming*" que PG é uma computação evolutiva que, em suma, é uma técnica

computacional que produz soluções automaticamente, sem exigir que o computador conheça a especificação antecipadamente da forma ou a estrutura da solução para um problema específico. BANZHAF, em sua definição diz que PG é parte do grande corpo de pesquisa chamada aprendizado de máquina (AP). KOZA trata a programação genética como um método para criar automaticamente um programa de computador para trabalhar a partir de um enunciado de problema de alto nível de dificuldade de resolução.

Todas as definições sobre as variações de AE (PG e AGs) dadas até aqui são muito importante para conhecimento inicial destas abordagens. Entretanto, para que não haja confusão no entendimento da representação destes métodos é importante destacar que, mesmo que a PG e os AGs sejam um tipo de AE, eles não são iguais. A representação de um algoritmo genético é dada por formato de strings, onde cada string representa um cromossomo. No caso da programação genética, sua representação é dada em formato de árvore, onde cada arvore representa, em teoria, um programa (PAPPA, 2010).

A seguir, a **Tabela 1** e a **Figura 7,** respectivamente, exibem exemplos de representações de AGs e PG. Considerando o cruzamento a partir da quinta posição do cruzamento dos strings (fluxos de caracteres) que representam os cromossomos envolvidos na operação.

Cromossomos pai	***Descendentes***
010(00001)	010(11101)
101(11101)	101(00001)

Tab. 1 - Representação dos AGs

Fonte: próprio autor.

Exemplo de representação da PG,

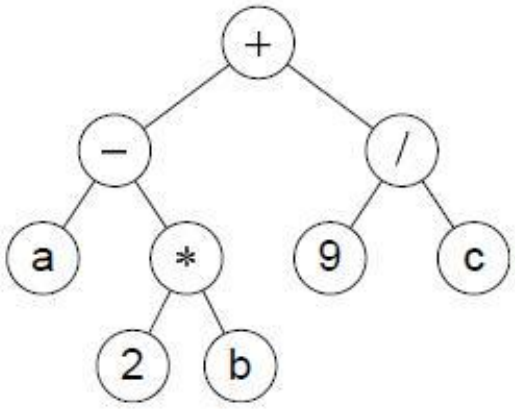


Fig. 7 - Representação gráfica da GP.
Fonte: adaptação de (WAPPLERa, 2006).

Nesta seção foram descritas algumas definições sobre as variações dos AE - algoritmos genéticos e programação genética. Alguns exemplos de trabalhos desenvolvidos sob estas tecnologias também foram discutidos. Na próxima seção serão descritas algumas técnicas de teste de software.

3 TESTE DE SOFTWARE

Testar software é fundamental para elevar a qualidade do mesmo. No final da década de 60 e início da década de 70 houve uma grande crise chamada "Software crisis" – crise do software. Este termo foi apresentado, pela primeira vez, por BAUER[2] em "*The 1968/69 NATO Software Engineering Reports*" - Conferência sobre Engenharia de Software da OTAN - no final da década de 60. Tal expressão descreve a crescente complexidade dos problemas de software enfrentados pelas entidades desenvolvedoras de software, diretamente proporcionais ao crescimento computacional na época. A causa destes problemas foi associada à complexidade global do hardware e do processo de desenvolvimento de software. Nas palavras do holandês, Edsger W. Dijkstra[3],

> "A principal causa da crise de software é que as máquinas tornaram-se várias ordens de grandeza mais poderosas! Para ser mais claro: enquanto havia máquinas com poder computacional a quase zero, a programação não causava tantos problemas; Quando tivemos alguns computadores com poder computacional leves, tivemos problemas leves. Agora, que temos computadores gigantescos, a programação tornou-se um problema igualmente gigantesco."
>
> – Edsger Dijkstra, The Humble Programmer (EWD340), Communications of the ACM.

Naquela época, teste de software era uma tarefa arcaica, sem muita eficácia. Verdadeiramente, não existia uma distinção clara entre teste de software e depuração do código fonte. A situação começou a mudar a partir do final da década de 70. MYERS, no ano de 1979, introduziu o conceito de teste de software, separadamente da depuração de software. Entretanto, ainda existia uma visão limitada do que a disciplina de teste representa nos dias de hoje. MYERS afirmava que "um teste bem sucedido era um teste que encontrava um bug". Por esta afirmação, pode-se perceber a grande necessidade de separar as tarefas específicas de desenvolvimento de software das tarefas realmente relacionadas ao teste software. Tarefas como verificação de funcionalidades de software em acordo com requisitos, depuração de código fonte, entre outras, eram executadas

2 - Friedrich Ludwig Bauer é um cientista da computação alemão e professor emérito na Universidade Munique de Tecnologia.
3 - Edsger Wybe Dijkstra foi um cientista da computação holandês. Ele recebeu o Prêmio Turing 1972 por contribuições fundamentais para o desenvolvimento de linguagens de programação. dente da Schlumberger Centenário de Ciências da Computação na Universidade do Texas em Austin, de 1984 até 2000.

pela mesma pessoa – o que espanta é que mesmo depois da evolução do teste de software e do aparecimento de ferramentas de automação de testes de software, desde o final da década de 60, ainda há esta prática em algumas empresas. Pesquisadores da área de teste de software como os doutores, Dave Gelperin[4] e William C. Hetzel[5] classificaram, em 1988, as cinco fases pelas quais a disciplina de teste de software passou desde a metade da década de 50 até os anos 2000.

A **Tabela 2** mostra, na visão de GELPERIN, as fases de evolução do teste de software.

Fase	*Período*		*Descrição*
1	até 1956	-	Orientado a depuração
2	1957 - 1978	-	Orientado a demonstração
3	1979 - 1982	-	Orientado a destruição
4	1983 - 1987	-	Orientado a avaliação
5	1988 - 2000	-	Orientado a prevenção

Tab. 2 - Fazes de evolução da disciplina de teste de software

Fonte: (GELPERIN, 1988).

LUO, em seu artigo "*Software Testing Techniques.Technology, Maturation and Research Strategy*" explica as fases descritas na tabela acima:

Fase 1 - **Orientado a depuração:**

Teste não era separado da depuração do código;

Fase 2 - **Orientado a demonstração:**

Teste era realizado para certificação de que o software satisfazia os requisitos de sua especificação;

Fase 3 - **Orientado a destruição:**

Teste era realizado para detectar falhas de implementação;

Fase 4 - Orientado a avaliação:

Teste era realizado para detectar falhas nos requisitos, desenho de software e implementação;

4 - David Gelperin é CTO e presidente da LiveSpecs Software em Minneapolis, MN. Experiente na área de engenharia de software com ênfase em qualidade de software, verificação e testes (SQVT) e engenharia de processos de software (STICKYMIND).

Fase 5 - Orientado a prevenção:

Teste era realizado para prevenir falhas nos requisitos, desenho de software e implementação.

A falta de um processo bem definido para efetuar o processo de teste de software causou enormes prejuízos, tanto para as entidades contratantes, tanto para as entidades desenvolvedoras de software. A **Figura 8** satiriza a situação caótica vivida na época da crise do software,

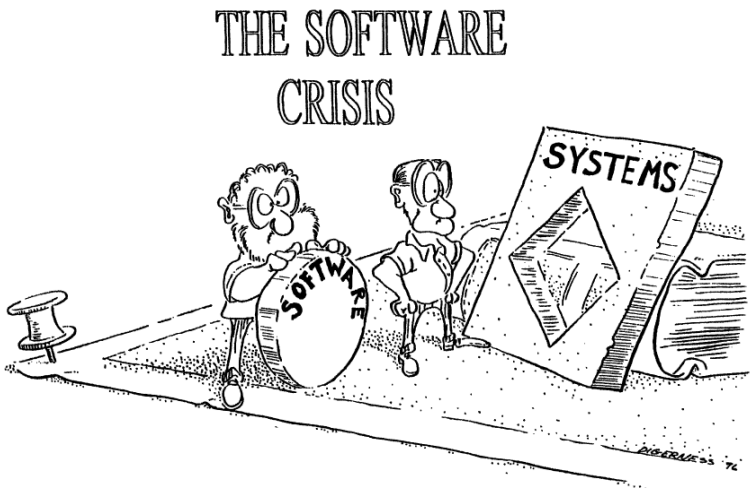


Fig. 8 - Satirização da crise do software
Fonte: (SHAW, 1994).

Mesmo com grande perda financeira causada pela crise e o aumento da desconfiança nos produtos desenvolvidos, a "Crise do Software" deixou evidente alguns fatores chave que estimularam a Engenharia de Software a identificar muitos dos problemas de desenvolvimento de software. Isso possibilitou, ao longo dos anos, uma melhora significativa na qualidade do software que é fabricado atualmente (SHAW, 1994).

5 - Doutor William C. Hetzel escreveu em 1991 a grande referencia em teste de software: o livro "O Guia Completo" para Teste de Software, que continua a ser o centro das atenções na cultura do teste de software e continua a ser uma fonte constante de referência.

3.1 Abordagens para Geração de Casos de Testes

Existem algumas técnicas de testes que são aplicadas a um produto de software para garantir que o nível da qualidade do mesmo melhore. O que não pode se garantir é que um software funcionará corretamente, por um longo período de tempo, sem apresentar algum tipo de erro (MYERS, 2004, p.8). Esta conclusão é válida no sentido de que é praticamente impossível conseguir testar todas as possibilidades de estados funcionais que um software pode apresentar, simplesmente pela complexidade variável de seus algoritmos e suas entradas de dados. Para complicar um pouco mais, anomalias podem se confirmar como erro devido à combinação de uma série de fatores destrutivos, não somente de um fato isolado. Fatores que estão, ou não, diretamente relacionados ao software, como erro nos requisitos, requisitos impossíveis de implementar, limitações de hardware, e, até mesmo, erro de implementação (humanos, inevitavelmente, inserem erros em um software), podem contribuir para a inserção de erros em um software (PAULA FILHO, 2009). As técnicas de teste de software que são usadas atualmente, para os sistemas orientados a objeto, são herdadas da técnica de testes de sistemas construídos na teoria da programação estruturada, mas com a mesma finalidade: detectar erros de software. Algumas destas técnicas serão discutidas no decorrer deste tópico.

3.1.1 Teste de Caixa-branca

A técnica chamada “Teste caixa-branca” é um teste dirigido exclusivamente ao código fonte do software, ou seja, esta técnica visa testar diretamente características básicas de funcionamento do software, mas, no nível de codificação do produto. Esta técnica de teste avalia o fluxo de dados, lógica dos algoritmos, partes do código que nunca serão executadas, eventuais ciclos de execução de um bloco de código. Esta técnica é também chamada de teste estrutural ou orientada à lógica (JURISTO, 1983).

Ao contrário do teste de caixa-preta, o software é observado como uma caixa branca uma vez que a estrutura e o fluxo do software sob teste são visíveis para o testador. Os planos de teste são desenvolvidos de acordo com os detalhes da implementação do

software, como estilos de programação, linguagem utilizada, lógica utilizada nas regras de negócio, dentre outras características. Os casos de teste são derivados da estrutura do programa.

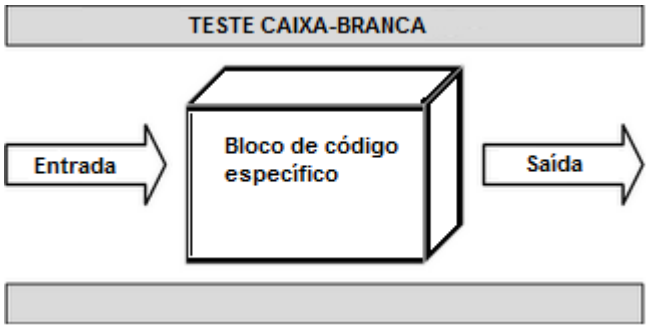


Fig. 9 - Exibição gráfica de um teste de caixa-branca
Fonte: adaptado de "*Software Testing Fundamentals*", 2010.

No teste caixa-branca tem se a intenção de esgotar alguns aspectos do software. Tal grau de exaustão pode ser alcançado, segundo PARRINGTON, utilizando alguns princípios, como a execução de cada linha de código, o que é descrito com cobertura de instrução; percorrer todos os ramos de declarações, descrito como cobertura ramo; ou cobrir todas as combinações possíveis de predicados: condição que resultaram em verdadeiro ou falso. Este último princípio é descrito como cobertura múltipla de condição. Os casos de teste são cuidadosamente selecionados com base no critério de que todos os caminhos serão cobertos pelo menos uma vez. Utilizar esta técnica pode facilitar a descoberta de código sem utilidade, códigos que provavelmente nunca serão executados. Descobertas que não poderiam se realizadas por meio de testes funcionais ou de caixa-preta.

3.1.2 Teste de Caixa-Preta

Na concepção de Jiantao PAN, a abordagem caixa-preta é um método de experimento em que os dados de teste são derivados dos requisitos funcionais, sem levar em conta a estrutura do programa final. Estes testes podem ser baseados em dados de entrada/ saída, ou em requisitos de teste. Esta situação se afirma pelo motivo de que só a funcionalidade do módulo de software é alvo para esta modalidade de teste. O testador trata o software sob teste como uma caixa preta, ou seja, só as entradas, saídas e as

especificações são visíveis. A funcionalidade do software é determinada observando as saídas geradas, levando-se em consideração as entradas fornecidas. Neste tipo de teste, entradas são submetidas - respeitando alguma regra, e as saídas são comparadas com a especificação para validar se o teste foi bem sucedido, isto é, se o resultado obtido foi o esperado. Todos os casos de teste são derivados da especificação. Neste tipo de teste, detalhes de implementação do código não são levados em consideração.

Fica evidente que, quanto mais combinações de entrada são testadas, mais problemas serão encontrados, elevando a qualidade do software. O ideal seria testar exaustivamente um software. Entretanto, testar massivamente todas as combinações válidas de entradas para um software é praticamente impossível. Quando se considera entradas inválidas, relacionadas a tempo de execução, sequência configurada para a sequência de entrada pode ocorrer o que é chamado de explosão combinatória.

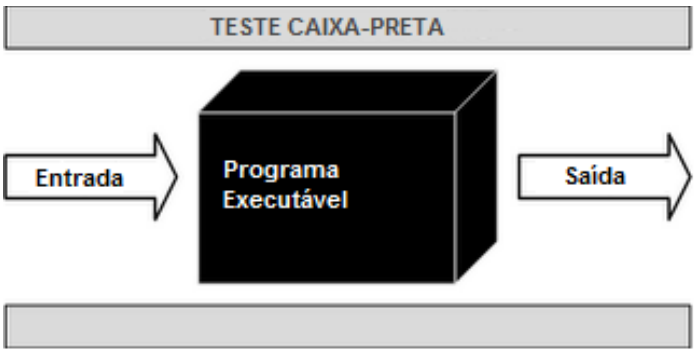


Fig. 10 - Exibição gráfica de um teste de caixa-preta
Fonte: adaptado de "Software *Testing Fundamentals*", 2010.

Este efeito é o principal obstáculo no teste funcional. Nunca se tem a certeza que a especificação é completa ou correta. Devido às limitações da linguagem - geralmente linguagem natural, idiomática - usada nas especificações, combinadas com níveis diferentes de experiência dos profissionais; problemas de interpretação podem surgir. Um exemplo clássico é a ambiguidade de informação que é frequentemente observada, causando distorções no entendimento da especificação dos testes. Mesmo que seja usado algum tipo de linguagem formal, pode-se deixar de anotar todos os casos possíveis na especificação. Não é incomum que a especificação se torne um problema. Pessoas raramente podem especificar com exatidão o que desejam. Normalmente, descrevem um esboço destorcido da realidade desejada – aquela velha história: "sabe o que quer, mas não sabe explicar". Segundo BEIZER, o produto final quase sempre não

corresponde às expectativas, depois da tarefa concluída. Segundo o mesmo autor, os problemas de especificação contribuem com uma média de 30% de todos os erros no software.

3.1.3 Teste de Caixa-cinza

O teste de caixa-cinza é a combinação dos métodos de testes de caixa-preta e caixa-branca, formando uma poderosa forma de testar software. Testes de caixa-cinza são gerados com base nas informações, como modelos baseados no estado ou diagramas de arquitetura do sistema sob teste (YUNUS, 2010). O método de caixa-cinza permite ao testador definir o estado das entradas e o resultado esperado, em termos de requisitos de sistema. COULTER, em seu artigo "*Graybox Software Testing Methodology*" afirma que o engenheiro de software pode definir os requisitos de desempenho, juntamente com os requisitos funcionais e estruturais. Assim, poderá ser formulado um teste completo, aprofundado do sistema em questão, de acordo com o nível do teste que deverá ser executado.

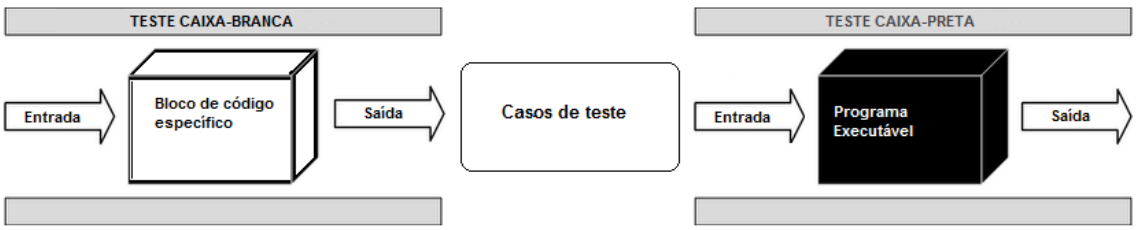


Fig. 11 - Exibição gráfica de um teste de caixa-cinza
Fonte: adaptado de "Software *Testing Fundamentals*", 2010.

A metodologia caixa-preta inicia com a atividade de identificar todos os requisitos de entradas e saídas designadas para um determinado software. Tais informações são extraídas da própria documentação de especificação de requisitos do software, ou seja, o teste é realizado confrontando a sua especificação. Porém, este tipo de teste exige que se tenha o mínimo possível de conhecimento do funcionamento interno do software.

Um exemplo de teste de caixa-cinza seria quando os códigos para duas unidades/ módulos são estudados - metodologia da caixa-branca - para projetar casos de teste reais

que são realizados utilizando as interfaces expostas do software - metodologia da caixa-preta (COULTER, 2001).

Para uma leitura mais aprofundada das técnicas de teste de software citadas neste texto, uma ampla abordagem pode ser encontrada nas obras de MYERS, PAULA FILHO, GELPERIN, KANER, BEIZER. Existem outras técnicas para testar software que não serão tratadas neste texto, mas, podem ser encontradas nas referências acima.

Na próxima seção serão discutidas outras abordagens mais recentes para testes de software, principalmente, teste de software para sistemas OO. Estas novas abordagens utilizam a abordagem dos AE para desenvolver os casos de testes, os chamados testes evolucionários TE.

3.2 Abordagens para Teste de Software Orientado a Objeto

Os sistemas que são construídos na atualidade são praticamente todos desenvolvidos utilizando OO (WAPPLERa, et al., 2006). Testes nessa área dão ênfase a testes de unidade.

O teste de unidade é usado nos estágios iniciais do desenvolvimento com a finalidade de detectar erros no software com foco em suas unidades funcionais, ou seja, uma Classe contendo suas funções e métodos codificados e testados um a um, observando a saída, em relação à entrada fornecida. Um TE no contexto de OO considera uma Classe ‘X’ como uma unidade sob teste (UST). A lógica por traz deste procedimento está no desejo de encontrar erros resultantes de comportamento inadequados, em cenários específicos, inseridos por programação não coerente.
Nem todos os cenários são passíveis de se observar uma UST. Para avaliar esta situação, um critério de validação, como uma “Função de Cobertura”, é aplicado para saber se uma Classe poderá se posta em um estado UST. O interessante desta avaliação é que o critério usado serve, indiretamente, no processo de geração de casos de teste. Se as

entradas dos casos de teste gerados satisfazem o critério de adequação não mais será necessário criar outros casos de testes.

Naturalmente, haverá casos em que uma classe sob teste dependerá de outras classes para que possa ser testada. Neste caso, a classe ou as classes utilizadas pela classe testada deverão existir e estarem funcionando. Este conjunto de classes requeridas é chamado de *cluster* de teste (CT) para CST (WAPPLERb, 2006).

Na prática, um teste de unidade, para software OO consiste em uma sequência de chamada a métodos, no sentido de uma ou mais "assertions" serem declaradas para avaliação da saída produzida por um método. A sequência de chamada dos métodos, também chamada de programa de teste, tem por objetivo representar um cenário de teste específico.

Neste cenário são criadas as instâncias dos *clusters* de classes de teste - os objetos das classes sob teste - necessárias para que os testes sejam criados e colocados em um estado específico, chamando várias instancias de métodos destes objetos. Ao final da execução do programa de teste as "assertions" fazem as verificações necessárias para verificar se os resultados obtidos são os esperados.

Um fator importante é que para testar um sistema OO usando a abordagem de adequação de critérios orientada a cobertura, exige-se que casos de testes sejam gerados de tal forma que satisfaçam os critérios de cada método de uma classe em particular. O que pode se concluir que um caso de teste é focado em um método por vez, chamado de método sob teste (MST). Para que estes critérios sejam atendidos é necessário que haja um conjunto de Classes que forneçam as funcionalidades específicas (ou métodos) a serem testadas. A este conjunto é nomeado com *Cluster* de teste.

A **Figura 12** exemplifica um *cluster* de teste para a classe ser testada, "**Controlador**",

```
class Controlador
{
    public Controlador(Configurador cfg)
    public void reconfigurar(Configurador cfg)
    public Configurador getConfig()
    public void conectar()
    public int recuperar(int sinal)
    public void disconectar()
}
class Configurador
{
    public Configurador()
    public Configurador(int port, int count)
    public int getPorta()
    public int getContadorSinal()
}
```

Fig. 12 - Cluster de teste para a classe Controlador

Fonte: adaptação de (WAPPLERa, 2006).

Nesta figura, percebe-se que a classe "Controlador" usa objetos da classe "Configurador" que, pela definição de cluster de teste, faz parte do cluster de teste porque a classe "Controlador" precisa das instâncias da classe "Configurador" para que seja executado seu caso de teste. Um exemplo de uso desta prática é mostrado na **Figura 13**,

```
public class CenarioDeTese {

    public Configurador cfg1;
    public Configurador cfg2;
    public Controlador ctl;

    public CenarioDeTese()
    {
        cfg1 = new Configurador( 0x0A, 5);
        cfg2 = new Configurador( 0x0B, 2);
        ctl = new Controlador( cfg1 );
        ctl.reconfigurar( cfg2 );
    }

    public void avaliacaoMetodoReconfigurar()
    {
        assert(ctl.getConfig().getPorta() == cfg2.getPorta());
        assert(ctl.getConfig().getContadorSinal() == cfg2.getContadorSinal());
    }
}
```

Fig. 13 - Exemplo de caso de teste para o método Controlador.reconfigurar()

Fonte: adaptação de (WAPPLERa).

onde o cenário de teste é criado tendo como parâmetros a criação de duas instâncias da classe “Configurador”, passando dois parâmetros, criando uma instância da classe “Controlador” utilizando como parâmetro uma das instâncias do tipo “Configurador”. Neste exemplo foi usando a instância Configurador.cfg1. A intenção do teste é descobrir possíveis erros na instância passada para o método “reconfigurar()” da Classe “Controlador”. Se não encontrar erros, o teste é avaliado como sucesso, caso contrário, o teste falhou.

Na tentativa de melhorar os resultados de teste software, AE são utilizados para desenvolver soluções para este fim. Com o estudo e aperfeiçoamento da técnica, será possível, no futuro, que as abordagens de automação de testes convencionais sejam substituídas por TE.

Na seção 4 “Algoritmos Evolucionários e Teste de software” será discutida a motivação do desenvolvimento das novas técnicas para criação de casos de teste de software.

4 ALGORITMOS EVOLUCIONÁRIOS E TESTE DE SOFTWARE

Desde o estouro da "crise do software", descrita na Seção 3, o teste de software tornou-se uma disciplina de grande importância no processo de produção de produtos de software. Naquela época, o software era caro e pouco confiável, devido a vários equívocos cometidos ao longo do processo de seu desenvolvimento. Com a separação efetiva do teste de software da depuração de código, e o aparecimento de ferramentas de automação de teste, o software começou a sair da "linha de produção" com uma qualidade consideravelmente maior em comparações aos tempos anteriores.
Atualmente, existem várias ferramentas disponíveis no mercado que permitem gerar casos de teste que são executados automaticamente, um grande avanço desde aquela época. Entretanto, existe ainda um problema a ser resolvido. Métodos de testes convencionais não conseguem cobrir boa parte do código e fluxos lógicos que, possivelmente, escondem erros que poderão aparecer quando menos se esperar.

A fim de resolver este problema, estudiosos estão empenhados em trabalhos direcionados na aplicação dos mais diversos algoritmos para a área de teste de software, incluindo algoritmos evolucionários. Estes algoritmos visam gerar casos de teste de forma a tentar capturar o maior número de erros em uma determinada unidade de software. Os resultados obtidos ao decorrer destes estudos tem se mostrado bem-sucedidos em testes estruturais – programação estruturada (WAPPLERb, 2006).
No entanto, pesquisas recentes estão se concentrado no desenvolvimento de testes evolucionários para sistemas desenvolvidos utilizando o paradigma de orientação a objetos (OO). A geração automática de casos de teste para este tipo de sistema favorecerá o aumento de eficiência, efetividade e custos reduzidos de desenvolvimento de software.

Além das abordagens tradicionais de automação de testes de software conhecidas no mercado, estão surgindo o que pode ser considerado como a evolução dos testes de software. Os primeiros avanços surgiram com a possibilidade de gerar casos de testes de software baseado em AEs para sistemas desenvolvidos sobre o paradigma da programação estruturada. Os resultados obtidos foram considerados sucesso. Mesmo com obtenção de ótimos resultados em TE para sistemas desenvolvidos com

programação estruturada, no futuro próximo, esta abordagem não mais será útil. Os sistemas que são construídos na atualidade são praticamente todos desenvolvidos utilizando OO (WAPPLERa, et al., 2006). Existem poucos trabalhos desenvolvidos no sentido de fornecer recursos de TE específicos para sistemas OO, como o *framework* "TestFul", desenvolvido por (MIRAZ, et al., GECCO 2009). Pesquisadores como WAPPLERc, TONELLA e LIU estão envolvidos no aperfeiçoamento de suas abordagens para TEs baseados em busca orientada para sistemas OO, com ênfase em testes de unidade.

A Seção 4.1 discutirá as vantagens e desvantagens em usar Algoritmos Evolucionários.

4.1 Vantagens e Desvantagens do uso de Algoritmos Evolucionários

Algoritmos evolucionários se mostraram bastante produtivos em alguns casos em que abordagens de testes de software convencionais não conseguiriam alcançar, naturalmente por suas limitações. Um exemplo, bem intuitivo, descrito por POLI, seria a não possibilidade de alcançar uma solução ótima para um determinado problema, a solução aproximada seria o possível de se obter. O autor levanta está questão, que não poderia deixar de ser relacionada à natureza: "Um coelho não precisa ser o animal mais rápido da natureza, ele precisar ser rápido o suficiente para escapar de uma raposa específica." O autor exemplifica claramente a finalidade de um TE que busca encontrar a melhor forma para cobrir os espaços passíveis de serem testados em um software, não a forma ótima, que é praticamente impossível. Encontrar a forma ótima seria o mesmo que afirmar que um TE descobriria todos os erros existentes em um software. Afirmação que, por enquanto, não pode ser feita, nem por TEs e nem por testes convencionais. Isto seria o ponto em comum entre as duas abordagens.

TEs tem várias vantagens sobre outros métodos. A principal delas seria a de, por natureza, os algoritmos evolucionários possuírem grande adaptabilidade e que são bastante flexíveis aos problemas reais (ARAKI, 2009). Neste caso específico, o problema seria a criação dos melhores casos de teste possíveis para um software. Testes automatizados tradicionais, principalmente os testes de unidades, necessitam que

pessoas criem manualmente os dados de entrada para cobrir uma especificação. Essa é visivelmente uma tarefa maçante, muito cara em termos de tempo e gastos com contratação de pessoal especializado.

Metodologias tradicionais exigem que algumas versões sejam construídas e avaliadas até que se encontre a versão ideal a ser usada. TEs também utilizam este principio, mas com a vantagem de produzir um número muito maior de versões automaticamente e avaliá-las para encontrar a melhor solução possível, gastando uma quantidade de tempo muito menor, em comparação ás técnicas disponíveis no mercado (HORNBY, 2006). Algoritmos evolucionários apresentam grandes vantagens, não apenas para testes de unidade CST, mas também pode ser aplicado para outros objetivos de teste (BAARS, et al., 2010).

É notável que a utilização de algoritmos evolucionários para geração de casos de testes tem grandes vantagens sobre os métodos usados atualmente no mercado de software em geral. Eles podem se adaptar facilmente a um dado problema. Praticamente, quase todas as características deste tipo de algoritmo são passíveis de serem customizadas (HEITZINGER, 2003).
No entanto, a abordagem dos EAs é considerada recente e ainda passa por estágios de estudo e compreensão. Em algumas situações os AEs se adaptam bem, em outras situações eles apresentam algumas deficiências. O risco de desenvolver TEs baseado em AE que não sejam bem formulados é grande.

Esta situação pode ser entendida como uma dificuldade natural de definir a melhor forma de explorar os espaços a serem cobertos em busca de erros. A maior desvantagem, caso esta situação ocorra, seria a obtenção de resultados inesperados. Ao contrário de se ter uma execução com resultados satisfatórios (espera se grande desempenho dos TEs), em termos de tempo e qualidade de cobertura de código, obter-se-ia resultados indesejáveis, tendendo para gastos excessivos de recursos de computação – memória e processamento (WHITLEY, et al., 1995). Entretanto, no final das contas, o resultado de sua execução poderia ser mais rápido que o programador (PAPPA, 2010). Outros estudos sobre o assunto descrevem que tentativas de se desenvolver soluções nas abordagens dos AE podem gerar resultados que tendem a uma

complexidade muito alta, em termos computacionais. A probabilidade de se encontrar maneiras de simplificar o desenvolvimento é bastante pequena (LIM, et al., 2003).

Como puderam ser observadas, as duas abordagens tem características com propriedades favoráveis e algumas limitações. A divergência maior entre as duas abordagem é, evidentemente, o nível de conhecimento que se tem sobre elas no mercado. Atualmente, no mercado de desenvolvimento de software, as abordagens de testes convencionais são mais conhecidas e, por este motivo, existem mais profissionais disponíveis para executá-las. Por enquanto, esta situação pode ser considerada uma grande vantagem sobre os TEs.

Estudos que se iniciam para desenvolver técnicas para resolver problemas que já são resolvidos por técnicas antigas tem a obrigação de apresentar possibilidades que resolvam tais problemas de maneira mais eficiente. Este é o resultado que se deseja obter com os AE para a área de teste de software. Qualquer técnica tem suas vantagens e desvantagens. O objetivo do estudo dos AEs é justamente desenvolver técnicas que gerem vantagens superiores as geradas pelas técnicas convencionais de teste de software.

Os desafios destas novas abordagens serão discutidos na Seção 5 “Algoritmos Evolucionários em Ambientes Comerciais”.

5 ALGORITMOS EVOLUCIONÁRIOS EM AMBIENTES COMERCIAIS

Há evidências de que algoritmos são frequentemente usados em controle de processos industriais. O problema é que empresários não se preocupam (ou evitam) em tornar público o resultado do trabalho desenvolvido nos domínios de suas empresas. Esta situação ocorre por alguns motivos não muito claros. Dentre estes motivos, poderia ser a preservação de segredo industrial ou, simplesmente, os detentores de empresas que utilizam a abordagem de TE não julgam importante separa uma fração de tempo para publicar os documentos de registro do trabalho.

Existem poucos relatos de uso efetivo de TE no cenário comercial. Devido ao pouco conhecimento neste ramo, é natural que isto ocorra. No mundo acadêmico, onde praticamente todas as novas tecnologias nascem existem vários trabalhos sobre o assunto. Muitos resultados já foram colhidos com execução do método a titulo de teste de conceito. No entanto, existe um trabalho publicado pela empresa Rila Solutions[6] que mostra o resultado da produção de um *framework* baseado em computação evolucionária "*Evolutionary Testing Framework*" (ETF), cuja *IDE* é uma extensão do Eclipse (Ferramenta de desenvolvimento em Java). O ETF tem por objetivo criar cenários de teste que facilmente poderão simular erros causados por memória corrompida e falhas causadas por arquivos corrompidos. Tais falhas, segundo documenta a empresa, são presenciadas apenas pelos usuários finais dos produtos. Segundo Arthur BAARS, do Departamento de Sistemas da Informação e Computação da Universidade Politécnica de Valência, membro do projeto *"EvoTest"*, que é o construtor do ETF, "em práticas industriais, desenvolvedores são frequentemente incapazes de reproduzir falhas que são encontradas por usuários finais ou testadores. Reproduzir o erro é importante para investigar a causa raiz da falha. Sem saber a causa da falha, um desenvolvedor, obviamente, não pode reparar-la."

O *framework* pode ser customizado para um teste específico. Para que os testes fossem realizados, um pocket PC da HP foi utilizado como SST. A **Figura 14** mostra o ChatPC usado com Sistema Sob Teste (SST),

6 - Empresa búlgara, especializada em Serviços de TI e soluções customizadas.

Fig. 1 - Exemplo de uso do ChatPC em sua versão móvel
Fonte: (BAARS, et al., 2010).

Para que a realização do teste seja possível, é necessário que sejam definidos três parâmetros: (i) Especificação para cada item a ser testado; (ii) Piloto dos testes (responsável por manter a conexão entre o framework e SST, no caso o ChatPC); (iii) Função de avaliação de resultados. O piloto de testes (*Test Driver*) monitora o SST em execução, os resultados são passados para o *framework* ETF. Através do ETF, a função de avaliação de resultados, gradativamente, consegue calcular a adequação do teste, indicando melhorias para o caso de teste executado.

A idéia de simulação de uso do pocket PC, pelo ETF seria a transmissão de comandos específicos de execução pontual ou disparando um conjunto de ações a serem realizadas pelo SST. Funções diversas podem ser disparadas. Entre elas pode-se citar a navegação em aplicativos ou páginas web, partindo para a gravação e reprodução de áudio e vídeo. A intenção de todas estas atividades é a simulação do uso prolongado do aparelho por usuários comuns em seu dia de trabalho ou lazer. Toda esta transação é constantemente monitorada e avaliada pelo ETF.

A empresa Rila Solutions é dos poucos casos de empresas que mostraram interesse em estudar, de forma mais aprofundada, os AEs, a fim de desenvolver um framework para

gerar casos de testes - TEs. O resultado do projeto “EvoTest” foi publicado no evento *“Third International Conference on Software Testing, Verification, and Validation Workshops”* com o título *“Using Evolutionary Testing to find Test Scenarios for Hard to Reproduce Faults”* em Abril de 2010.

O caso da empresa Rila Solutions é raro no cenário comercial de desenvolvimento software, mesmo que seja um produto de uso específico de empresa. Uma das maiores dificuldades de se desenvolver um produto neste perfil para o mercado seria o problema em respeitar os requisitos que exigem que esta ferramenta seja robusta, flexível, eficiente e fácil de usar. Esta ferramenta deve atender aos testadores experientes e, por outro lado, também deve atender as necessidades dos profissionais menos experientes. Este produto de geração de TEs deve ter estas características para que tenha aceitação no mercado. Este é considerado o ponto crítico em um projeto com estas características, segundo (BAARS, et al., 2010).

6 CONCLUSÃO

Ao longo deste texto foram discutidos alguns trabalhos que puderam ser realizados usando a abordagem dos Algoritmos Evolucionários. Entidades de renome mundial como a NASA já desenvolveram projetos baseados nesses métodos. Entre eles, destacamos o projeto Missão ST5, que tinha a finalidade construir micro satélites a fim de medir os efeitos das atividades solares na magnetosfera terrestre por um período determinado de tempo.

Na área comercial, empresas também estão utilizando os AE para construir ferramentas para solucionar problemas enfrentados por usuários de seus produtos. Como exemplo do uso dos AE na área comercial, mostrou-se o caso da empresa Rila Solutions. Esta empresa desenvolveu um *framework,* chamado ETF, para criar cenários de testes para simular erros que eram presenciados apenas por usuários finais de seus produtos. Um *Pocket PC* da HP foi utilizado como alvo de testes.

Vários estudos são desenvolvidos atualmente sobre os AEs. Na área cientifica o número de pesquisas é bem maior, em comparação ao cenário comercial. Esta situação pode ser percebida pela quantidade de trabalhos publicados por pesquisadores do meio acadêmico. Poucos trabalhos são conhecidos por parte de publicações de empresas que atual no mercado de desenvolvimento de software. Este fato decorre das empresas não se preocuparem em publicar o resultados de seus trabalhos ou, simplesmente, não querem publicar. Em outras palavras, o segredo da tecnologia garante que tal empresa, que tenha este conhecimento, exerça certa vantagem sobre as que não dominam tal tecnologia. O que seria admissível, em se tratando de se manter em vantagem na concorrência de mercado.

Dificuldades são e serão encontradas durante os períodos de desenvolvimento dos TEs. O desenvolvimento dos algoritmos é bastante complexo. A execução de seus algoritmos demanda um considerável poder computacional. Os resultados podem ser imprevisíveis, em algumas situações. No entanto, as vantagens em se usar AE para se construir teste de software são animadoras, em relação às abordagens convencionais. Os resultados obtidos são, na maioria dos casos, superiores em relação às abordagens convencionas.

Esta situação pode ser exemplificada nos resultados obtidos nos projetos da NASA. Os executores dos projetos conseguiram simular uma quantidade maior de testes antes para conseguir o melhor formato da antena. Esse resultado foi obtido em tempo reduzido em comparação as abordagens convencionais. Outra vantagem significativa, observada no projeto, é a flexibilidade de adaptação a mudanças que tiveram que se realizadas no projeto. O tempo gasto para adaptar a modelagem do algoritmo para se adaptar às mudanças no projeto foi da casa de poucas semanas. Segundo HORNBY, se o projeto fosse desenvolvido baseado nas abordagens convencionais, certamente o tempo para adaptação seria bem maior.

Sem dúvida, o teste de software, utilizado no mercado é de fácil acesso e de boa aceitação. Isto é natural, pois são mais antigos e vêm sendo aperfeiçoadas desde o final da “crise do software”, no final da década de 60. AEs passaram a ser estudados com mais atenção a partir da década de 90. Na atualidade, algumas vantagens dos testes convencionais sobre os TE, são o maior número de ferramentas disponíveis no mercado, o domínio da tecnologia e quantidade de pessoal especializado.

Certamente, com mais alguns anos de pesquisa e aperfeiçoamento das técnicas baseadas nos AE, novas ferramentas de geração e automatização de testes estarão disponíveis no mercado.

REFERÊNCIAS

AL SALAMI ,Nada M. A. Genetic Programming under Theoretical Definition. International Journal of Software Engineering and Its Applications Vol. 3, No.4, October 2009.
Disponível em:
< http://www.sersc.org/journals/IJSEIA/vol3_no4_2009/4.pdf>.
Acesso em: 20 Set. 2010.

ARAKI, Lucilia Yoshie. Um Algoritmo Evolutivo para Geração de Dados de Testes para Satisfazer Critérios Baseados em Código Objeto Java. Dissertação apresentada como requisito parcial para obtenção do Grau de Mestre. Programa de Pós-Graduação em Informática. Universidade do Paraná.
Disponível em:
< http://dspace.c3sl.ufpr.br/dspace/bitstream/1884/20935/1/Dissertacao.pdf >.
Acesso em: 19 Out. 2010.

BAARS, Arthur I.; Vos, Tanja E. J.; Dimitrov, Dimitar M. Using Evolutionary Testing to find Test Scenarios for Hard to Reproduce Faults. Third International Conference on Software Testing, Verification, and Validation Workshops.
Disponível em:
< http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=05463646&tag=1>.
Acesso em: 21 Out. 2010.

BANZHAF, W.; Nordin, P.; Keller, Robert .E.; Francone, Frank. D. Motivation
In: _____Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications. San Francisco, California: Morgan Kaufmann Publishes, Inc, 1998. P 4- 10.

BAUER. The 1968/69 NATO Software Engineering Reports. Dagstuhl-Seminar 9635: "History of Software Engineering" Schloss Dagstuhl, August 26 - 30, 1996.
Disponível em:
< http://homepages.cs.ncl.ac.uk/brian.randell/NATO/NATOReports/index.html>.
Acesso em: 21 Set. 2010.

BEIZER, Boris, Black-box Testing: techniques for functional testing of software and systems. Publication info: New York : Wiley, c1995. ISBN: 0471120944 Physical description: xxv, 294 p.: ill. ; 23 cm.

COULTER, André C. Graybox Software Testing in Real-Time in the Real World. Graybox Software Testing Methodology. Lockheed Martin Missiles and Fire Control – Orlando, v. 8, n. 1, p. 2 – 9, 2001.
Disponível em:
<http://legacy.cleanscape.net/docs_lib/paper_graybox.pdf> Acesso em: 06 Out. 2010.

GELPERIN, D.; B. Hetzel (1988). "The Growth of Software Testing". CACM , v 31, n. 6. ISSN 0001-0782.

HEITZINGER, Clemens. Advantages and Disadvantages of Evolutionary Algorithm Optimizers (Pág. Web do autor).
Disponível em: <http://www.iue.tuwien.ac.at/phd/heitzinger/node35.html>
Acesso em: 22 Out. 2010.

HORNBY , Gregory S., et al. Automated Antenna Design with Evolutionary Algorithms. University of California Santa Cruz, Mailtop 269-3, NASA Ames Research Center, Mo_ett Field, CA.

JURISTO, Natalia; Vegas, Sira. Functional Testing, Structural Testing and Code Reading: What Fault Type do they Each Detect? Facultad de Informática. Universidad Politécnica de Madrid Campus de Montegancedo, 28660, Boadilla del Monte, Madrid, Spain.
Disponível em:
< http://www.grise.upm.es/docs/ESERNET_Functional_Testing.pdf >.
Acessado em: 04 Out. 2010.

KANER, Cem; James Bach, Bret Pettichord. Lessons Learned in Software Testing: A Context-Driven Approach. Wiley, 2001, p. 4. ISBN 0-471-08112-4.

KOZA, Jhon .R.. Introduction to Genetic Algorithms In: _____Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, 1992, Bradford Book. P 17 - 62.

LIM, H. S.; Rao, M. V. C.; Tan, Alan W. C.; Chuah, H. T. Multiuser Detection for DS-CDMA Systems Using Evolutionary Programming. IEEE COMMUNICATIONS LETTERS, VOL. 7, NO. 3, MARCH 2003 101.
Disponível em:
<http://www-ee.ccny.cuny.edu/www/web/ysun/References/MUD/Lim_03_IEEE_COMLETTER(1).pdf>. Acesso: 25 Out. 2010.

LIU, X.; Wang, B.; Liu, H. Evolutionary search in the context of object-oriented programs. In MIC2005: The Sixth Metaheuristics International Conference, September 2005.

LUO, Lu. Software Testing Techniques.Technology Maturation and Research Strategy. Institute for Software Research International Carnegie Mellon University Pittsburgh, PA15232 USA.
Disponível em: <www.cs.cmu.edu/~luluo/Courses/17939Report.pdf>. Acesso em: 16 de Nov. 2010.

MANTERE,Timo Jarmo T. Alander. Evolutionary software engineering, a review. Science Direct Applied Soft Computing 5 (2005) 315–331: N. pag. 315.

MIRAZ, Matteo; Lanzi, Pier Luca; Baresi, Luciano. TestFul: using a Hybrid Evolutionary Algorithm for Testing Stateful Systems. Dipartimento di Elettronica e Informazione – Politecnico di Milano piazza Leonardo da Vinci, 32 20133 - Milano, Italy. GECCO 2009.
Disponível em:
< http://home.dei.polimi.it/miraz/pubs/GECCO09.pdf>. Acesso em: 15 Out. 2010.

MITCHELL, Melanie. Genetic Algorithms: An Overview. Santa Fe Institute 1399 Hyde Park Road, NM 87501 Complexity, 1 (1) 31 - 39. (1995): N. pag. 17.

MYERS, Glenford J.. A Self Assessment Test In: _____The Art of Software Testing. John Wiley & Sons, Inc., Hoboken, New Jersey, 2004. Cap. 1, p. 8 – 10. ISBN 0-471-04328-1.

PAN, Jiantao. Software Testing. Carnegie Mellon University, Spring 1999.
Disponível em:
< http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/>. Acesso em: 04 Out. 2010.

PAPPA, Gisele L. Computação Evolucionária. Notas de Aula. Universidade Federal de Minas Gerais – UFMG, 2010.

PARRINGTON, Norman; Roper, Marc. Understanding Software Testing. John Willey & Sons, 1989. ISBN:0-7458-0533-7; 0-470-21462-7.

PAULA FILHO, Wilson de Pádua. Engenharia de Software: Fundamentos, Métodos e Padrões / Wilson de Paula Pádua Filho. – 3o ed. – Rio de Janeiro: LTC, 2009.

POLI, Riccardo, et al. A Field Guide to Genetic Programming. University of Essex – UK. March 2008.
Disponível em:
<http://www.lulu.com/items/volume_63/2167000/2167025/2/print/book.pdf>.
Acesso em: 15 Set. 2010.

RUSE, Michael. Charles Darwin And Artificial Selection. Journal of the History of Ideas. University of Pennsylvania Press. v. 36, pp. 339-350 ,No. 2 Abr. - Jun., 1975.
Disponível em:
<http://www.jstor.org/stable/2708932>. Acesso em: 21 Set. 2010.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. IBM Journal of Research and Development, v. 44, n. 1 de 2, p. 206, 2000.

SHAW ,Mary. Issues - The Software Crisis. Chapter 1, p. 1 -2.
Disponível em:
< http://media.wiley.com/product_data/excerpt/94/08186760/0818676094.pdf >.
Acesso em: 30 Set. 2010.

Software Testing Fundamentals. Black Box Testing Definition, Example, Application, Techniques, Advantages and Disadvantages. (Pág. Web).
Disponível em: < http://www.softwaretestingfundamentals.com/2009/03/black-box-testing.html> Acesso em: 22 Out. 2010.

TONELLA, P. Evolutionary testing of classes. In ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis, pages 119–128, New York, NY, USA, 2004. ACM Press.
VASCONCELOS, Dr. João Antônio de. Notas de aula. Programação Genética.
Disponível em:
<http://www.cpdee.ufmg.br/~rdmaia/2010/01/DIVERSOS/AulaPG.pdf>.
2010.

WAPPLERa,S; Wegener ,J.. Evolutionary unit testing of object-oriented software using strongly-typed genetic programming. In GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, volume 2, pages 1925–1932, Seattle, Washington, USA, 8-12 July 2006. ACM Press.

WAPPLERb ,Stefan; Wegener , Joachim. Evolutionary Unit Testing Of Object-Oriented Software Using A Hybrid Evolutionary Algorithm. 2006 IEEE Congress on Evolutionary Computation Sheraton Vancouver Wall Centre Hotel, ancouver, BC, Canada July. P. 16-21, 2006.

WAPPLERc, S., Lammermann, F. Using evolutionary algorithms for the unit testing of object-oriented software. In GECCO ’05: Proceedings of the 2005 conference on Genetic and evolutionary computation, pages 1053–1060, New York, NY, USA, 2005. ACM Press.

WHITLEY, Darrell; Ranaa, Soraya; Dzubera, John; Mathias, Keith E. Evaluating evolutionary algorithms. Computer Science Department, Colorado State University, Fort Collins, CO 80523, USA Philips Research Labs, Amherst, MA 01003, USA. Received January 1995; revised October 1995.
Disponível em: <http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B6TYF-3VVCKK9-F-1&_cdi=5617&_user=10&_pii=0004370295001247&_origin=search&_coverDate=08/31/1996&_sk=999149998&view=c&wchp=dGLbVlb-zSkzV&md5=086312b1c1e9b1c9ad8860652ab5d6b0&ie=/sdarticle.pdf>. Acesso: 25 Out. 2010.

YUNUS ,Mamoon; Mallal, Rizwan.SOA Testing using Black, White and Gray Box Techniques.
Disponível em: <http://www.crosschecknet.com/resources/white_papers/SOA_Testing_Techniques.pdf>. Acesso em: 05 Out. 2010.