Geração de Testes de Aceitação em Fit a partir de Especificações em B

Thiago C. de Sousa[1], Claudia de O. Melo[2]

[1]Departamento de Engenharia de Computação – Escola Politécnica
Universidade de São Paulo (USP)

[2]Departamento de Ciência da Computação – Instituto de Matemática e Estatística
Universidade de São Paulo (USP)

thiago.carvalho@poli.usp.br, claudia@ime.usp.br

Abstract. *The use of automated acceptance tests contribute effectively improving software quality for better understanding of the requirements, and also helping catch defects early in the development cycle. However, for critical systems, which there is the need for extensive testing, their preparation requires much time effort since they are written one by one. In this paper we propose the use of specifications described in B as a lightweight formal method for rapid and massive generation of acceptance tests into Fit tables format, increasing the coverage of them and helping the agile team in the development of critical systems.*

Resumo. *O uso de testes de aceitação automatizados contribuem efetivamente para o aumento da qualidade de desenvolvimento de um software pois melhoram o entendimento dos requisitos, além de facilitarem a descoberta de erros logo no inicio do processo. No entanto, para sistemas críticos, onde há necessidade desses testes serem exaustivos, a sua elaboração demanda muito esforço de tempo, uma vez que são escritos um a um. Nesse artigo propomos a utilização de especificações descritas em B como um método formal leve para a geração rápida e maciça de testes de aceitação no formato de tabelas Fit, aumentando a cobertura destes e ajudando a equipe ágil no desenvolvimento de sistemas críticos.*

1. Introdução

Um problema tradicional da engenharia de software é que as funções implementadas pelos sistemas não conseguem satisfazer as necessidades do cliente. A definição de testes de aceitação no início do projeto ajuda a equipe de desenvolvimento a assegurar que as funções desejadas serão implementadas conforme o exigido. Beck (2002) defende que esses testes, além de feitos no início de cada iteração do projeto, sejam definidos pelo cliente em conjunto com a equipe. Um dos *frameworks* mais populares que segue esse princípio é o Fit (*Framework for Integrated Test*), desenvolvido por Cunningham (2005), no qual o cliente, com o apoio da equipe, escreve testes de aceitação em tabelas pré-formatadas de fácil compreensão e as salva em formato HTML ou até mesmo em planilhas. Segundo Ricca et al. (2009), é altamente desejável que um teste de aceitação seja também o mais automatizado possível. No entanto, pode se tornar moroso para cliente e equipe especificarem testes de aceitação em Fit para sistemas com

requisitos críticos, pois em geral eles envolvem muitas restrições e mais casos de teste. Por isso, times ágeis que desejam desenvolver sistemas críticos podem encontrar problemas para gerenciar a qualidade e o tempo ao longo do desenvolvimento.

O B é um método formal que vem crescendo nos últimos anos na indústria de sistemas críticos por garantir a geração de código a partir de especificações formais descritas em uma notação matemática. Essa notação se baseia na lógica de primeira ordem, na teoria dos conjuntos e na aritmética inteira, complementados pelo cálculo de substituições generalizadas, que permite que as especificações tenham uma aparência de programação imperativa, facilitando o seu uso. Uma das mais usadas ferramentas que dão suporte à essa notação é o ProB, criada por Leuschel e Butler (2003), que se utiliza de técnicas de satisfação de restrições (CSP – *Constraint Satisfaction Problem*) para encontrar automaticamente valores para as variáveis de tal maneira que as restrições impostas sejam satisfeitas. O B pode ser empregado como um método leve, uma vez que é possível usá-lo parcialmente dentro de qualquer processo de desenvolvimento de software, principalmente para verificação de consistência de modelos UML e de requisitos controlados, segundo Lano et al. (2004) e Sousa et al. (2010).

Nesse trabalho, propõe-se a utilização de especificações B apenas no seu nível mais abstrato como ferramenta auxiliar para geração automática de testes de aceitação dentro de um processo de desenvolvimento rápido de aplicações. Mais especificamente, proporemos um mapeamento da notação B para as principais tabelas Fit e mostraremos alguns exemplos de casos de testes que podem ser gerados pelo ProB. Dessa forma, esperamos contribuir para a complementação dos testes de aceitação de times ágeis por meio do uso de métodos formais.

O restante do artigo está organizado da seguinte forma: trabalhos relacionados na Seção 2, uma breve explicação sobre testes de aceitação e Fit na Seção 3, uma introdução ao método B e ao funcionamento do ProB na Seção 4, apresentação da proposta de mapeamento de especificações formais B em tabelas Fit na Seção 5 e, por fim, reservamos a Seção 6 para discussões e enumeração de trabalhos futuros.

2. Trabalhos Relacionados

Segundo Bowen et al. (2002), métodos formais e testes sempre serão técnicas complementares para a redução de erros em um sistema. Os autores ressaltam a aplicabilidade de métodos formais para a geração de casos de teste. Já Bacchelli et al. (2008) relatam como a adoção de ferramentas de geração de testes pode melhorar o *trade-off* entre restrições de tempo e qualidade em um projeto.

O trabalho de Stotts et al. (2002) apresenta um método para a criação sistemática de testes de unidade completos e consistentes baseado em especificação algébrica. Resultados iniciais apontaram que os testes gerados pelo método encontraram mais erros que testes de unidade *ad-hoc*. Porém, o uso de especificação algébrica requer um conhecimento matemático maior que o utilizado pelo método B.

Uma abordagem próxima à nossa foi apresentada por Souza (2009), onde casos de testes em JUnit são gerados a partir de especificações B através do ProB usando técnicas de particionamento de equivalência e análise do valor limite. Entretanto, essa proposta não discute uma integração com contextos ágeis, além de pouco focada na usabilidade, uma vez que não há um software para automatizar todo o processo.

Gupta e Bhatia (2010) propõem um método para geração de casos de testes a partir de especificações formais em B, com o uso da ferramenta ProB. Para aumentar a cobertura dos requisitos, uma matriz com as combinações de ações é gerada. No entanto, os casos de teste gerados não são compatíveis com as atuais ferramentas de testes de aceitação automatizados.

3. Testes de aceitação automatizados e o *framework* Fit

De acordo com IEEE (1986), testes de aceitação são testes formais conduzidos pelo cliente para determinar se o sistema satisfaz ou não seus critérios de aceitação, determinando também se o sistema deve ou não ser aceito. Reppert (2004) define automatização de testes de aceitação como uma abordagem para a especificação de requisitos de forma legível e executável, que ajuda o time a produzir e entregar o sistema. A comunidade ágil prega que os testes de aceitação automatizados devem ser feitos no início de cada iteração de desenvolvimento, o que é uma extrapolação do conceito de TDD (*Test Driven Development*).

Diversas ferramentas foram propostas para auxiliar a especificação, organização e execução de testes de aceitação. Segundo Melnilk et al. (2006), uma das mais populares na comunidade ágil é a Fit, uma ferramenta livre que permite a especificação de testes de aceitação em forma tabular. As tabelas podem ser escritas em diversos formatos, como Word, Excel e Html. Uma tabela Fit especifica as entradas e saídas esperadas de um dado caso de teste. Existem três tipos de tabelas para a criação de casos de teste: *Column, Row* e *Action*. As tabelas *Column* são usadas para representar cada cenário em uma linha. As tabelas *Row* servem para validar objetos produzidos após uma consulta, enquanto as tabelas *Action* permitem a definição de uma sequência de comandos usados, em geral, para testar interfaces de usuário e workflows. Após criar os casos de teste nas tabelas Fit, o desenvolvedor deve criar *Fixtures*, código que faz a ligação entre as tabelas Fit e o sistema em desenvolvimento, tornando os testes de aceitação executáveis.

4. O método B

B é um método formal criado por Abrial (1996), baseado em transição de estados e usado para especificar, refinar e implementar sistemas críticos. A ideia principal do B é iniciar o desenvolvimento com um modelo bem abstrato do software e ir adicionando gradualmente mais detalhes, gerando modelos cada vez mais refinados até se alcançar o nível de código. A corretude e a consistência desse refinamento são garantidas via obrigações de provas matemáticas.

O B usa uma notação baseada em máquinas abstratas de estados conhecida como ASM (*Abstract Machine Notation*) para especificar os sistemas. Uma máquina encapsula um estado e as operações sob esse estado. Um estado é descrito através de um conjunto de variáveis. Uma máquina abstrata (MACHINE) contém informações em conjuntos (SETS) que representam as estruturas de dados do sistema. Uma máquina também possui variáveis (VARIABLES), seus valores iniciais (INITIALIZATION) e invariantes (INVARIANT), que explicitam propriedades do sistema e restrições sobre as variáveis. As variáveis são passíveis de modificação ao serem usadas como parâmetros as operações (OPERATIONS), que descrevem os procedimentos

(verificando as pré-condições e provendo pós-condições) que uma máquina deve ser capaz de executar. As operações são classificadas em regulares e consultas.

Na programação por restrições, os problemas são especificados usando um conjunto de restrições ao invés de um algoritmo. Logo, os desenvolvedores não precisam descrever como resolver as restrições, somente precisam especificá-las para um solucionador, que encontrará as soluções possíveis para o problema realizando buscas em todas as combinações de valores de acordo com as restrições. Seguindo essa abordagem, Leuschel e Butler (2003) desenvolveram uma ferramenta chamada ProB, que tem sido bastante utilizada para animação e verificação de modelos escritos em B. O ProB realiza tanto animações com um número randômico de operações quanto com um número definido pelo usuário. A ferramenta fornece combinações válidas de dados de entrada ao usuário, que então pode escolher um subconjunto desses dados e assim animar a máquina B e verificar os resultados obtidos.

Na Figura 1 apresentamos no ProB uma animação randômica para 10 operações de uma especificação de um sistema de registro de notas (cuja a invariante é uma função parcial que mapeia aluno com uma nota de 0 a 10) com uma operação para inclusão (registrar, cuja pré-condição é verificar se "cc" faz parte do conjunto de alunos, mas não possui nota ainda e se a nota "nn" atribuída se encontra entre 0 e 10) e outra para consulta (nota). Na coluna da esquerda podemos visualizar o estado corrente do banco de notas. Na coluna do meio estão as operações aplicáveis a partir desse estado. E na coluna da direita, podemos verificar as operações que já foram aplicadas aleatoriamente.

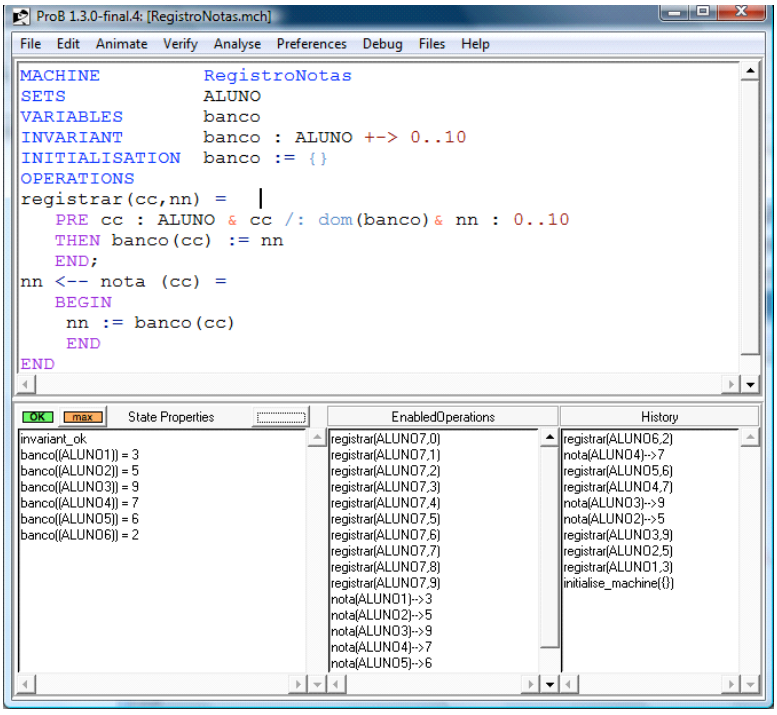

Figura 1. Especificação da Máquina RegistroNotas no ProB

5. Mapeamento do B para o Fit

Nessa seção, apresentamos a visão mais detalhada de nossa ideia explicando o fluxo ideal do processo proposto, bem como a inserção de nossa ferramenta no mesmo.

Posteriormente, discutiremos padrões de especificação em B e exibiremos exemplos simples a fim de elucidar a nossa proposta.

5.1. Visão Geral da Proposta

O mecanismo de execução da proposta é bem simples: uma especificação em B padronizada para cada uma das três principais tabelas Fit (*ColumnFixture*, *RowFixture* e *ActionFixture*) é editada no ProB, que por sua vez gera um arquivo texto com os dados das simulações. Esses dados, juntamente com os modelos das tabelas Fit em formato HTML, são usados como entrada para o aplicativo que estamos desenvolvendo chamado ***B2Fit***. Este, por sua vez, encaixa automaticamente os dados no modelo de tabela Fit mais apropriado, produzindo tabelas Fit já preenchidas. A perspectiva da arquitetura proposta pode ser vista na Figura 2.

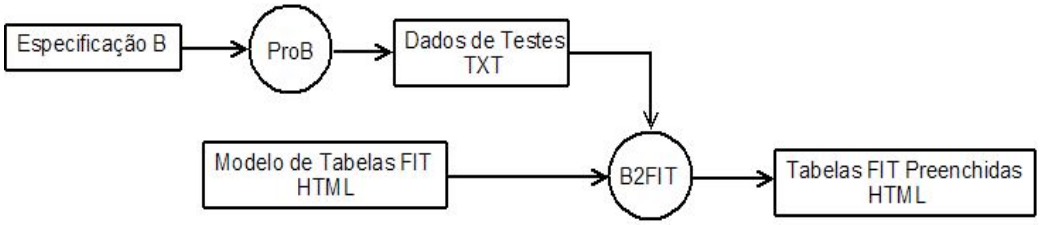

Figura 2. Visão da Solução Proposta

5.2. Padrões para Especificações em B

Os padrões visam facilitar a reutilização de soluções já conhecidas para determinados problemas. Com a intenção de evitar a perda de tempo no aprendizado da notação B, criamos padrões a serem seguidos quando se deseja escrever testes de aceitação para cada uma das três mais usadas tabelas Fit.

5.2.1. Padrão para *ColumnFixture*

A tabela *ColumnFixture* é sem sombra de dúvida a mais conhecida dos usuários do Fit, sendo usada quando se deseja verificar se uma dada função produz corretamente uma saída de acordo com as entradas fornecidas.

Como não existe a necessidade de se armazenar dados, a máquina B não precisa ter um estado definido, podendo-se descrever apenas a operação a ser verificada que o solucionador de restrições do ProB gera os valores de entradas possíveis. Assim sendo, o padrão de especificação em B para esse tipo de tabela seria como mostrado abaixo.

```
MACHINE OPFUN1

OPERATIONS OPFUN1

END
```

Suponha que o cliente precise especificar um teste de aceitação para verificar a corretude de uma função que deve calcular a subtração de dois números. Na Figura 3 apresentamos tal especificação com as combinações encontradas (coluna do meio no ProB, "*EnabledOperations*"), bem como a tabela Fit que deve ser gerada.

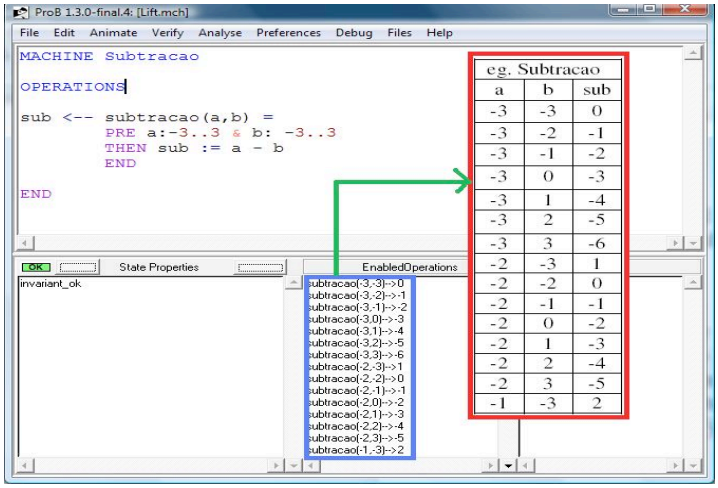

Figura 3. Exemplo para Geração de ColumnFixture

5.2.2. Padrão para *RowFixture*

Já a tabela *RowFixture* é bastante utilizada quando se deseja testar consultas que devolvem um conjunto com um número exato de elementos, como por exemplo uma função que encontra os cinco primeiros números primos. Nesse caso já existe a necessidade de se armazenar dados e, portanto, a máquina B precisa ter um estado com todos os elementos definidos, incluindo a operação que se almeja analisar. Assim sendo, o padrão de especificação em B para esse tipo de tabela seria como mostrado abaixo.

```
MACHINE OPQUERY1

VARIABLES  XX

INVARIANT  YY

INITIALISATION  ZZ

OPERATIONS OPQUERY1

END
```

Suponha agora que o cliente necessite especificar um teste de aceitação para analisar se uma função devolve o conjunto dos 10 primeiros números de Fibonacci. Podemos também especificar em B tal função e simulá-la para valores em um determinado intervalo. Na Figura 4, apresentamos tal especificação com o resultado da simulação (coluna da direita no ProB, "*History*"), bem como a tabela Fit gerada.

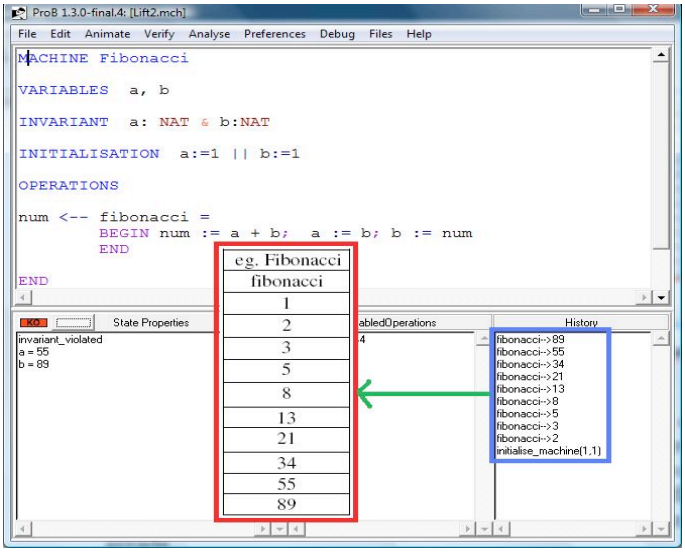

Figura 4. Exemplo para Geração de RowFixture

5.2.3. Padrão para *ActionFixture*

Por fim, a tabela *ActionFixture* é muito usada quando se precisa simular eventos ocasionados pelos usuários na interface do sistema. Nesse caso existe também a necessidade de se armazenar dados e, portanto, a máquina B precisa ter um estado com todos os elementos definidos. A diferença em relação ao padrão anterior é a inclusão de varias operações, cada uma representando um possível evento do usuário. Assim sendo, o padrão de especificação em B para esse tipo de tabela seria como mostrado abaixo.

```
MACHINE OPQUERY1

VARIABLES  XX

INVARIANT  YY

INITIALISATION  ZZ

OPERATIONS OPEVENT1 END; OPEVENT2 END; .....OPEVENTN END

END
```

Imagine que o cliente precisa testar a execução de um contador sobre uma pagina web onde ele pode incrementar o contador pressionando um botão, atribuir um valor ao contador e verificar o seu valor atual. Podemos também especificar em B tais operações e simulá-las para um determinado número aleatório de eventos. Na Figura 5, apresentamos tal especificação com o resultado da simulação (coluna da direita no ProB, "*History*"), bem como a tabela Fit que deve ser gerada.

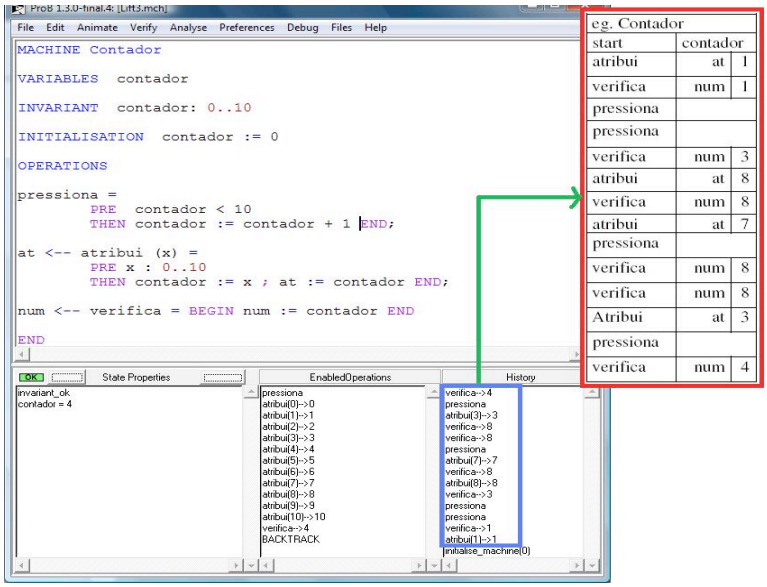

Figura 5. Exemplo para Geração de *ActionFixture*

6. Discussões e Trabalhos Futuros

Segundo Cohen e Money (2008) e Black et al. (2009), a integração de métodos ágeis e métodos formais é recente e promissora e pode trazer muitos benefícios para as duas comunidades. A proposta apresentada contribui para mostrar que essa integração é viável. Nesse artigo foi exposta uma ideia de utilização do método formal B para geração rápida e maciça de testes em formato de tabelas Fit, que são facilmente entendidas pela equipe e pelo cliente, como forma de ajudar equipes ágeis que lidam com desenvolvimento de sistemas críticos e que gastam muito tempo para criar

manualmente os testes de aceitação. Outra vantagem da abordagem proposta é a redução do tempo de aprendizagem da notação matemática presente no B devido ao uso de padrões de mapeamento para cada uma das tabela Fit, facilitando assim a sua adoção por desenvolvedores que não tenham um bom conhecimento matemático.

O projeto ***B2Fit*** ainda está em fase inicial e no momento ainda estamos desenvolvendo o primeiro protótipo. Quando finalizado, pretende-se realizar um estudo de caso em uma empresa especializada em desenvolvimento de sistemas críticos a fim de analisar a viabilidade do seu uso em exemplos mais interessantes e reais.

Entre as melhorias que devem ser incorporadas ao longo do projeto, pode-se citar: a integração com algum *framework* de testes de aceitação que use o Fit, como o Fitnesse; e a inclusão de particionamento de equivalência e análise do valor limite para evitar testes redundantes em sistemas não críticos.

Referências

Abrial, J-R. (1996) The B-Book: assigning programs to meaning. C.U.P.

Bacchelli, A., Ciancarini, P., and Rossi, D. (2008) On the Effectiveness of Manual and Automatic Unit Test Generation. In *Proceedings of the 2008 the Third international Conference on Software Engineering Advances*. ICSEA. IEEE Computer Society, Washington, DC, 252-257.

Beck, K. (2002) Test Driven Development: by Example. Addison-Wesley Longman Publishing Co.

Black, S., Boca, P. P., Bowen, J. P., Gorman, J., and Hinchey, M. (2009) Formal Versus Agile: Survival of the Fittest. *Computer* 42, 9, 37-45.

Bowen, J., Bogdanov, K., Clark, J., Harman, M., Hierons R. and Krause, P. (2002) FORTEST: Formal Methods and Testing. In *Proc of 26th Annual International Computer Software and Applications Conference*, p. 91

Cohen, S. J. and Money, W. H. (2008) Bridge Methods: Complementary Steps Integrating Agile Development Tools and Methods with Formal Process Methodologies. In *Proceedings of the Proceedings of the 41st Annual Hawaii international Conference on System Science*. HICSS. IEEE Computer Society, Washington, DC, 460.

Cunningham, W. (2005) "Framework for Integrated Test", http://fit.c2.com/, April.

de Sousa, T. C., Almeida, J. R., Viana, S., and Pavón, J. (2010). Automatic analysis of requirements consistency with the B method. *SIGSOFT Softw. Eng. Notes* 35, 2, 1-4.

Gupta, A. and Bhatia, R. (2010) Testing functional requirements using B model specifications. *SIGSOFT Software Engineering Notes* 35, 2 , 1-7.

IEEE Std 1012 (1986) IEEE Standard for Software Verification and Validation Plans.

Lano, K and Clark, D and Androutsopolous, K (2004) UML to B: formal verification of object-oriented models. In: Integrated Formal Methods: 4th International Conference, IFM.

Leuschel, M. and Butler, M. (2003) ProB: A model checker for B. In FME 2003: Formal Methods, Springer-Verlag LNCS 2805, pp 855–874.

Melnik, G., Maurer, F., and Chiasson, M. (2006) Executable Acceptance Tests for Communicating Business Requirements: Customer Perspective. In *Proceedings of the Conference on AGILE 2006*. IEEE Computer Society, Washington, DC, p. 35-46.

Reppert, T. (2004) "Don't Just Break Software, Make Software: How Story-Test-Driven-Development is Changing the Way QA, Customers, and Developers Work". *Better Software*, 6(6): 18–23.

Ricca, F., Torchiano, M., Di Penta, M., Ceccato, M., and Tonella, P. (2009) Using acceptance tests as a support for clarifying requirements: A series of experiments. *Inf. Softw. Technol.* 51, 2 , 270-283.

Souza, F. (2009) Geração de Casos de Teste a partir de Especificações B. Dissertação de Mestrado. UFRN.

Stotts, P. D., Lindsey, M., and Antley, A. (2002) An Informal Formal Method for Systematic JUnit Test Case Generation. In *Proc of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods - Xp/Agile Universe 2002*. Lecture Notes In Computer Science, vol. 2418. Springer-Verlag, London, 131-143.