Transparência e Testabilidade para a Geração Automatizada de Casos de Teste de Software

Edgar Sarmiento Calisaya and Julio Cesar Sampaio do Prado Leite

Pontifícia Universidade Católica do Rio de Janeiro, PUC - Rio, Brasil

{ecalisaya,julio}@inf.puc-rio.br

Resumo. Transparência de Software é um conceito relacionado à disponibilidade de informação; e, o software é considerado transparente quando suas operações são visíveis e se tem acesso ao processo que o gerou. Durante o processo de construção do grafo de interdependências de transparência, percebeu-se que os requisitos não funcionais relacionados à transparência também estavam relacionadas à Testabilidade; ou seja, transparência e testabilidade têm contribuições positivas. Testabilidade é uma característica de software que expõe o grau em que um artefato de software facilita o processo de testes. Este trabalho descreve os relacionamentos entre a testabilidade e a transparência: *como a testabilidade melhora a transparência, e como a transparência facilita a testabilidade?* A testabilidade e a transparência podem facilitar à automação de testes, e este trabalho propõe uma abordagem automatizada para a geração de casos de teste a partir de descrições de requisitos. O entendimento e a transparência dos requisitos possibilitam a construção de software mais transparente; as linguagens de Cenário e Léxico serão usadas para representar as descrições de requisitos.

Palavras-chave. transparência de software, testabilidade, requisitos, cenários, teste de software, casos de teste.

1 Introdução

Segundo [2], o direito de ser informado e de ter acesso à informação tem sido uma consideração importante nas sociedades modernas; entretanto, como o software permeia vários aspectos da sociedade, em algum ponto do futuro, engenheiros de software terão que dar conta de mais uma demanda: transparência.

O grupo de engenharia de requisitos da PUC-Rio (Grupo ER) [10] classificou a transparência como um conjunto de requisitos não funcionais (metas flexíveis), organizados hierarquicamente através de um grafo de interdependências (SIG – NFR Framework [13]). Este grafo representa as diferentes contribuições e correlações necessárias para satisfazer este novo requisito não funcional, e mostrado parcialmente na Fig. 1. Este grafo é composto por 28 nós folha e cinco grupos. Cada nó é uma característica ou meta flexível necessária para alcançarmos a transparência.

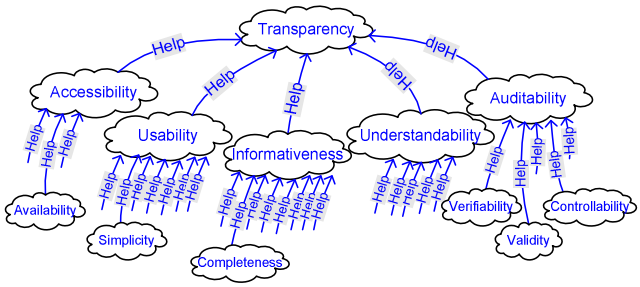


Fig. 1. Transparência como um requisito não funcional [2]

Considera-se a transparência como um conceito em desenvolvimento, onde alguns aspectos ainda estão sendo pesquisados; pois fazer software transparente tem algumas implicações, ou seja, os NFRs do SIG de transparência podem contribuir ou ter correlações sobre outros requisitos não funcionais (Fig. 2). Por exemplo:

- ***Segurança e Privacidade***: software transparente pode comprometer a segurança.
- ***Custo***: software transparente e de qualidade pode ser muito custoso.
- ***Testabilidade***: software transparente pode facilitar a testabilidade, e, software testável pode impactar positivamente na transparência.

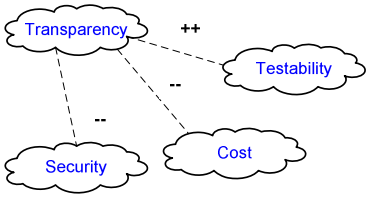


Fig. 2. Possíveis Correlações entre NFRs [10]

Durante o processo de construção do SIG de transparência, percebeu-se que as NFRs identificadas também contribuem positivamente na testabilidade; ou seja, transparência e testabilidade têm contribuições positivas. Para isto foi necessário construir o SIG da testabilidade. Na Fig. 3 é possível observar que os NFRs que contribuem na testabilidade já fazem parte do SIG de transparência.

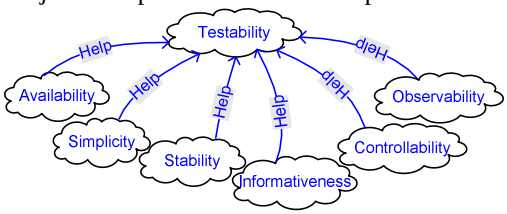


Fig. 3. Testabilidade como um requisito não funcional (Adaptado de [14])

Controlabilide e observabilidade são as características chave para a testabilidade [14]; ou seja, para testar um artefato de software é necessário controlar as entradas

(*controlabilidade*) e observar as saídas (*observabilidade*). Sem estas características é impossível melhorar a testabilidade do software; assim como o é a *acessibilidade* na transparência.

A Tabela 1 mostra os requisitos não funcionais presentes na transparência e que contribuem na testabilidade. Em alguns casos foi necessário identificar algumas equivalências (Observabilidade ≡ Validade e Verificabilidade).

Table 1. Relacionamentos e equivalências entre transparência e testabilidade.

Transparency	**Testability**
Availability	Availability
Simplicity	Simplicity
Completeness	Stability
Informativeness	Informativeness
Controllability	Controllability
Verifiability & Validity	Observability

Diversas pesquisas demonstraram que o processo de desenvolvimento de software deveria ser dirigido pelos requisitos [2,12]. Isto principalmente porque os conceitos usados para definir os requisitos de um sistema são (e devem ser) usados no restante das etapas do processo de desenvolvimento (projeto e implementação). Alem disso, demonstrou-se que o caminho para construir software transparente precisa de métodos de desenvolvimento de Software dirigidos por requisitos [2]. Estes requisitos devem ser entendíveis e transparentes para todos os items envolvidos (cidadãos, usuários, clientes e desenvolvedores).

A transição de requisitos para outros artefatos de software tem sido um dos principais desafios no desenvolvimento de software e diversas abordagens que visam ajudar neste processo têm sido propostas (*Model-driven Development*). Neste contexto, a transição de modelos de requisitos para modelos de teste (*Model-driven Testing*) sempre foi um tópico desafiante, pois as tarefas de testes são as mais custosas, demoradas e obscuras. Então, a especificação transparente de requisitos pode facilitar a transformação automatizada de descrições de requisitos para casos de teste.

Uma abordagem de especificação de requisitos baseada em cenário e léxico [1] pode ajudar tanto na compreensão do sistema que será desenvolvido quanto na validação do conhecimento do engenheiro e pelo cliente. Isto se dá pelo fato de que os cenários são descritos em linguagem natural e modelam situações do sistema, fazendo com que clientes se sintam mais à vontade e entendam melhor o processo de descobrimento dos requisitos.

Desta forma, nossa motivação principal é facilitar a transição entre requisitos e casos de teste de uma maneira a seguir os princípios de transparência de software e paralelamente melhorar a testabilidade do produto de software; as linguagens de Cenário e Léxico serão utilizadas para representar as descrições de requisitos.

2 Objetivos da Pesquisa

Os testes de software continuam sendo uma das técnicas de verificação e validação mais comumente utilizadas. No entanto, as atividades associadas aos testes são bastante custosas, obscuras e usualmente executadas manualmente. Para reduzir o seu custo e aumentar sua eficácia é desejável relacionar os requisitos ao processo de testes o mais cedo possível e automatizar as atividades relacionadas a este [3, 4, 9]. Então, é sempre um tópico desafiante reduzir o custo destas tarefas: *fazer a tarefa de testes mais fácil e mais efetiva*. Em outras palavras, melhorar a testabilidade do software.

A Testabilidade e a Transparência são satisfeitas através da operacionalização das características de qualidade relacionadas a estas; este fato possibilita um melhor entendimento, manipulação e uma melhor visualização e interpretação dos resultados da execução do software. Estas operacionalizações têm um impacto positivo sobre o produto e o processo de software. Neste trabalho, testabilidade e transparência facilitam a automação de testes.

O objetivo desta pesquisa é mostrar os relacionamentos entre a testabilidade e a transparência: *como a testabilidade melhora a transparência e como a transparência facilita a testabilidade?* Ou seja, este trabalho mostrará como a Testabilidade e a Transparência facilitam à automação de testes, e este propõe uma abordagem automatizada para a geração de casos de teste a partir de descrições de requisitos.

3 Contribuições Esperadas

A contribuição principal deste trabalho é uma abordagem automatizada para a geração de casos de teste de aplicações complexas, a partir de descrições de requisitos descritas nas linguagens de Cenário e Léxico [1]. Neste processo, para cada cenário é derivado um grafo direcionado e este grafo é representado como um diagrama de atividades (UML). Este diagrama é usado para a derivação de cenários de teste usando estratégias de *graph-search* e *path-combination* [6, 7, 8]. Um diagrama de atividades pode mostrar o comportamento visual de um cenário porque este representa o fluxo de controle entre atividades, este também pode ser usado para facilitar a transformação a outros modelos (*Model-driven Development - MDD*). A Fig. 4 mostra este processo.

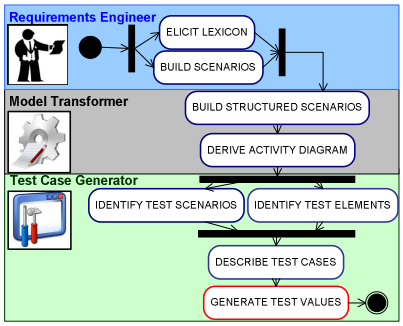


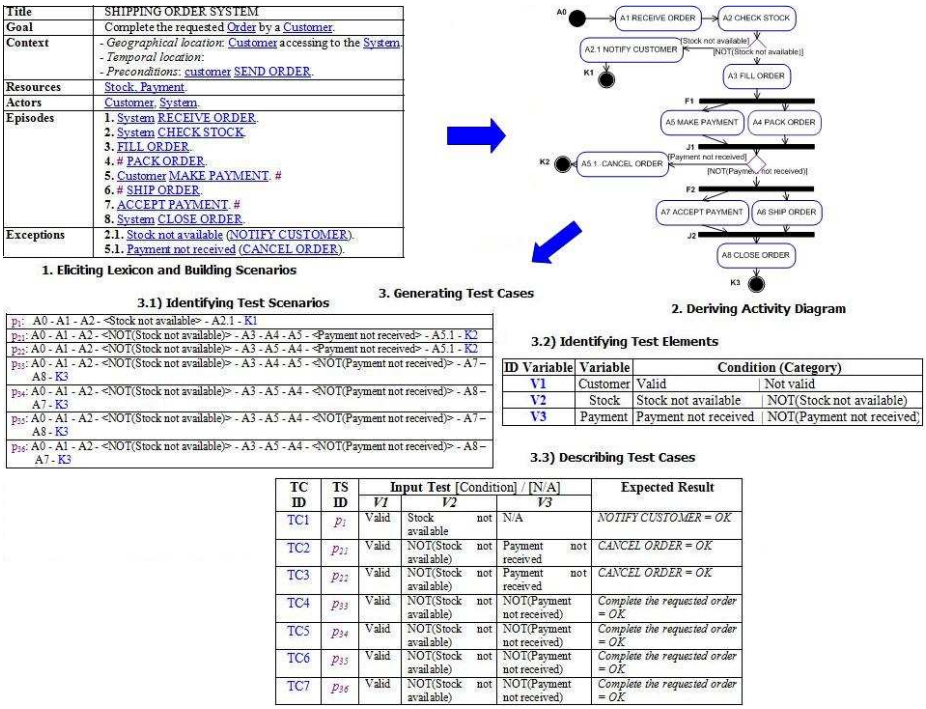
Fig. 4. Processo de Derivação de Casos de Teste

Cenários [1] descrevem situações especificas da aplicação, e estas descrições referenciam as palavras ou frases relevantes do domínio da aplicação (Léxico). Este relacionamento: (1) explicita as principais entidades da aplicação desde descrições iniciais de requisitos, (2) explicita as interações de tarefas concorrentes descritas no cenário, e (3) pode ser usado para reduzir o número de cenários de teste derivados. Este fato também possibilita as descrições de aspectos internos e externos da aplicação. Casos de uso [5] podem ser vistos como um modelo particular de cenários; entretanto, caso de uso é usado para descrever interações entre o usuário e o sistema através de uma interface (aspectos externos).

4 Resultados já Alcançados

Para demonstrar a viabilidade desta abordagem, os algoritmos de transformação de descrições de requisitos para grafos de atividades, e de geração de cenários de teste a partir destes grafos estão sendo implementados junto com uma ferramenta de apoio a este processo. Os algoritmos propostos são capazes de lidar com as descrições de requisitos que contém estruturas complexas como a concorrência. O desenvolvimento esta sendo baseado na ferramenta C&L (edição de cenários e léxicos) [11].

A Fig. 5 mostra através de um exemplo as principais tarefas da nossa abordagem. O exemplo ilustrado é a derivação de casos de teste para o "*Shipping Order System*".



Title	SHIPPING ORDER SYSTEM.
Goal	Complete the requested Order by a Customer.
Context	- *Geographical location*: Customer accessing to the System. - *Temporal location*: - *Preconditions*: customer SEND ORDER.
Resources	Stock, Payment.
Actors	Customer, System.
Episodes	1. System RECEIVE ORDER. 2. System CHECK STOCK. 3. FILL ORDER. 4. # PACK ORDER. 5. Customer MAKE PAYMENT. # 6. # SHIP ORDER. 7. ACCEPT PAYMENT. # 8. System CLOSE ORDER.
Exceptions	2.1. Stock not available (NOTIFY CUSTOMER). 5.1. Payment not received (CANCEL ORDER).

TC ID	TS ID	Input Test [Condition] / [N/A]			Expected Result
		V1	*V2*	*V3*	
TC1	p_1	Valid	Stock not available	N/A	*NOTIFY CUSTOMER = OK*
TC2	p_{21}	Valid	NOT(Stock not available)	Payment not received	*CANCEL ORDER = OK*
TC3	p_{22}	Valid	NOT(Stock not available)	Payment not received	*CANCEL ORDER = OK*
TC4	p_{33}	Valid	NOT(Stock not available)	NOT(Payment not received)	*Complete the requested order = OK*
TC5	p_{34}	Valid	NOT(Stock not available)	NOT(Payment not received)	*Complete the requested order = OK*
TC6	p_{35}	Valid	NOT(Stock not available)	NOT(Payment not received)	*Complete the requested order = OK*
TC7	p_{36}	Valid	NOT(Stock not available)	NOT(Payment not received)	*Complete the requested order = OK*

Fig. 5. Ilustração do Processo de Geração de Casos de Teste

5 Conclusão

A nossa abordagem fornecerá contribuições, devido às seguintes razões: (1) a geração de casos de teste baseado em cenários ajuda os desenvolvedores a melhorar a qualidade do produto e, conseqüentemente, reduzir: custos de falhas em uso, custos de manutenção depois de entregue o produto final, o tempo de desenvolvimento do software. (2) a abordagem será capaz de detectar as interações, erros de comunicação e bloqueios entre processos concorrentes de forma mais compreensiva do que as abordagens já existentes. (3) a abordagem será capaz de derivar casos de teste para aplicações concorrentes a partir de descrições de requisitos baseados numa linguagem natural semi-estruturada, as abordagens existentes são baseadas em modelos visuais semi-formais. (4) a abordagem será capaz de reduzir o número de cenários de teste gerados para aplicações concorrentes.

Cenário [1] facilita testes unitários e testes de integração, pois descreve as interações entre objetos ou módulos do sistema. Os módulos são descritos como cenários de um macro-cenário (Sistema), e os módulos podem ser descritos por sub-cenários que descrevem situações específicas (procedimentos).

6 Referências

1. Leite, J. C. S. P., Hadad, G., Doorn, J., Kaplan, G.: A scenario construction process, Requirements. Engineering Journal, Springer-Verlag London Limited, 5, 1, pp. 38-61 (2000)
2. Leite, J. C. S. P., Cappelli, C.: Software Transparency, *Business & Information Systems Engineering*: Vol. 2: Iss. 3, 127-139 (2010)
3. Heumann, J.: Generating test cases from use cases, IBM (2001)
4. Binder, R. V.: Testing object-oriented systems. Addison Wesley (2000)
5. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley, Reading, MA (2001)
6. Katayama, T., Itoh, E., Furukawa, Z.: Test-case generation for concurrent programs with the testing criteria using interaction sequences. 6th Asia-Pacific Software Engineering Conference (1999)
7. Yan, J., Li, Z., Yuan, Y., Sun, W., Zhang, J.: BPEL4WS Unit Testing: Test Case Generation Using a Concurrent Path Analysis Approach, In Proceedings of ISSRE'06, pp.75-84 (2006)
8. Shirole, M., Kumar, R.: Testing for concurrency in UML diagrams. SIGSOFT Softw. Eng. Notes, vol. 37, no. 5 (2012)
9. Denger, C., Medina, M.: Test Case Derived from Requirement Specifications. Fraunhofer IESE Report (2003)
10. Grupo de Pesquisas em Engenharia de Requisitos da PUC-Rio. Disponível em: http://transparencia.inf.puc-rio.br/wiki/index.php/Integrantes.
11. C&L – Cenários e Léxicos – Disponível em: http://pes.inf.puc-rio.br/cel.
12. Van Lamsweerde, A. From System Goals to Software Architecture. Formal Methods for Software Architectures (2003)
13. Chung L., Nixon B.A., Yu E., Mylopoulos J. Non- Functional Requirements in Software Engineering. Kluwer Academic Publishing. Boston Hardbound (2000)
14. Bach, J.: Heuristics of Software Testability (2003). Disponível em: http://www.satisfice.com/tools/testable.pdf Acessado em 20/07/2013