

FATEsC - Uma Ferramenta de apoio ao teste estrutural de componentes

Vânia Somaio Teixeira^{1,2}, Marcio Eduardo Delamaro¹,
Auri Marcelo Rizzo Vincenzi³

¹Programa de Pós-graduação em Ciência da Computação (PPGCC)
Centro Universitário Eurípides de Marília (UNIVEM) – Marília, SP

²Faculdade Gennari&Peartree (FGP) – Pederneiras, SP

vania@fgp.com.br, delamaro@pesquisador.cnpq.br

³Universidade Católica de Santos (UNISANTOS) – Santos, SP

auri@unisantos.br

Abstract. *The use of components in software development brought many benefits, but also problems for the testing activity, which can be seen under two perspectives: the developer's and the user's. The problem in both of them is the lack of information; for the developer who doesn't know all the contexts in which the component will be used and for the user who doesn't know details about the component development or test. The FATEsC Tool allows the developer to generate testing metadata and to associate them to the component, permitting the user of the component to manipulate such information in order to evaluate her or his own test set, considering the coverage of the component's code.*

Resumo. *O uso de componentes em desenvolvimento de software trouxe muitos benefícios, mas também problemas para a atividade de teste, que pode ser vista sob duas perspectivas: a do desenvolvedor e a do usuário do componente. O problema, em ambas, é a falta de informação, do desenvolvedor por não conhecer todos os contextos de utilização do componente e do usuário pela falta de documentação adequada sobre o componente e sobre seu desenvolvimento e teste. A Ferramenta FATEsC permite que o desenvolvedor gere metadados de teste e os associe ao componente, viabilizando ao usuário do componente consultar tais informações para avaliar seus conjuntos de teste, levando em conta também, a cobertura do código do componente.*

1. Introdução

O objetivo primordial da Engenharia de Software é a construção de software com qualidade e com baixo custo. Para tanto, diversos métodos, técnicas e ferramentas têm sido propostos e utilizados buscando o aumento da produtividade no desenvolvimento de software. Uma das formas para tal foi o reaproveitamento de artefatos de software, assim, em vez de iniciar-se sempre um projeto de software “do zero”, são reutilizadas soluções e experiências adquiridas em projetos anteriores. É o que se costuma chamar de reúso de software. Uma forma de reúso de software, embora não seja uma prática

recente, é o desenvolvimento baseado em componentes, que é a reutilização de software por meio do encapsulamento de funcionalidades em unidades que são independentes e podem ser acopladas a um sistema por meio de uma interface conhecida.

Beydeda e Gruhn (2003) afirmam que os desenvolvimentos baseados em componentes tendem a trazer grandes benefícios na produção de software, mas, por outro lado, adicionam dificuldades que precisam ser transpostas, principalmente na fase de testes. Dificuldades essas, como a falta de informação sentida tanto pelo desenvolvedor como pelo usuário do componente, para que sejam realizados os testes de forma sistemática e com qualidade. O desenvolvedor do componente não conhece todos os contextos de utilização do componente e, portanto, o componente pode ter um comportamento em determinados ambientes conforme o esperado e, em outros, não. O usuário sofre com a falta de documentação adequada sobre o componente, agravada muitas vezes, pelo fato de o código fonte não estar disponível. A troca de informações entre eles, desenvolvedor e usuário do componente, é fundamental para o sucesso da utilização do componente, mas nem sempre isso ocorre.

A atividade de teste é importante em todo processo de desenvolvimento de software, pois tem como objetivo principal revelar defeitos em produtos que estão sendo testados. Apesar dos vários aspectos do desenvolvimento de software baseado em componente, Vinzenzi et al (2005) afirmam que as técnicas tradicionais de teste podem ser utilizadas nos testes de software baseado em componentes. Dentre as técnicas tradicionais de testes destaca-se neste trabalho a técnica estrutural, que tem como objetivo garantir que determinadas estruturas do código da implementação sejam exercitadas. Tais estruturas, como comandos, desvios ou pontos do programa onde as variáveis são utilizadas, constituem os requisitos de teste a serem satisfeitos pelos casos de teste. A porcentagem de tais requisitos executados – ou cobertos – pelos casos de teste, representa a cobertura desse conjunto em relação ao teste estrutural.

Neste contexto, foi desenvolvida a Ferramenta FATEsC – Ferramenta de Apoio ao Teste Estrutural de Componentes – com o objetivo de contribuir com a proposição de uma estratégia para o teste de componentes utilizando a técnica de teste estrutural. Foi implementada como uma extensão da ferramenta JaBUTi (VINZENZI et al, 2006) para gerar e disponibilizar metadados encapsulados com o código do componente, fornecendo dados sobre a cobertura dos testes realizados pelo desenvolvedor do componente ao usuário. A ferramenta JaBUTi permite a automação de determinados processos da fase de teste estrutural para programas escritos na linguagem de programação Java, necessitando apenas do bytecode para suas atividades. Essa característica deve permitir que os dados de cobertura de teste do componente, alcançados durante o teste efetuado pelo desenvolvedor, possam ser reutilizados pelo usuário do componente.

Na próxima seção é apresentada a Ferramenta FATEsC e o processo de sua utilização através de um exemplo. A Seção 3 contém as considerações finais deste artigo e propostas de trabalhos futuros.

2. FATEsC – Ferramenta de Apoio ao Teste Estrutural de Componentes

FATEsC é uma ferramenta *DeskTop*, desenvolvida na linguagem de programação Java e é composta por duas partes: uma para ser utilizada pelo desenvolvedor do componente e outra para ser utilizada pelo usuário do componente. Basicamente tem as mesmas funcionalidades, exceto pela comparação das coberturas obtidas pelos casos de teste do desenvolvedor e do usuário do componente, que aparece apenas na versão do usuário.

Na FATEsC, perspectiva do desenvolvedor do componente, é possível criar, manipular e gerenciar casos de teste para cada método público do código, além de definir para cada caso de teste uma descrição informal de suas características. Após definidos os casos de teste, a ferramenta auxilia o desenvolvedor a avaliar a cobertura de cada um deles usando a ferramenta JaBUTi e associa essa informação a cada um dos métodos públicos testados. Essa informação é anexada ao componente (arquivo jar) por meio de metadados no formato XML, tornando-se disponível aos usuários do componente.

Na perspectiva do usuário do componente, é possível criar e avaliar a adequação dos casos de teste e comparar medidas de cobertura obtidas, com as fornecidas nos metadados. Por exemplo, suponha-se um método **m**, da interface pública do componente. Se na execução dos casos de teste do usuário obteve-se a mesma cobertura, em relação ao código de **m** que os casos de teste do desenvolvedor, isso pode significar que realmente os casos de teste estão adequados. Caso contrário pode-se analisar os casos de testes descritos nos metadados e verificar se há a necessidade de criar novos casos de teste ou se para a aplicação em questão, não é possível obter a mesma cobertura. É importante notar que para a aplicação desta estratégia é essencial que o usuário possa avaliar a cobertura obtida sobre o código do componente, sem que o programa fonte lhe esteja disponível. Por essa razão optou-se pela utilização da ferramenta JaBUTi, que utiliza apenas o código objeto (bytecode Java) para realizar a análise de cobertura.

2.1. Processo de utilização da Ferramenta

Para melhor compreensão da proposta desse trabalho e a utilização da ferramenta propõe-se um exemplo simples considerando como componente a classe **Fat** que faz o cálculo do fatorial de um número e uma aplicação, que utiliza o componente, a classe **Agrupamentos** que faz o cálculo de combinação simples. As Figuras 1 e 2 mostram o código fonte para o exemplo proposto.

```
public class Fat
{

    /* calcula o fatorial de um
    número inteiro.
    O valor passado deve ser >= 0
    e <= 20.
    O valor retornado é um long
    */

    public long fat(int x)
    {
        if ( x < 0 || x > 20 )
            return -1;
        if ( x == 0 )
            return 1;
        if ( x <= 2 )
            return x;
        long s = 1;
        while ( x > 1 )
        {
            s *= x;
            x--;
        }
    }
}
```

Figura 1. Código fonte do componente proposto como exemplo.

```

public class Agrupamentos {

    /* Calcula o número de combinações de k
    elementos n a n.
    Os valores passados devem ser >=1 e de
    deve ser >= n
    */

    public long combinacao(int k, int n)
    {
        Fat f = new Fat();
        if (k < 1 || n < 1 || k < n )
            throw new IllegalArgumentException("Argumento invalido");
        long s = f.fat(k);
        s /= f.fat(n);
        s /= f.fat(k-n);
        return s;
    }
}
    
```

Figura 2. Código fonte da aplicação que utiliza o componente (Fat) proposto como exemplo.

A partir de um arquivo jar contendo o bytecode do componente, cuja estrutura hierárquica é apresentada na parte esquerda (1) da tela da Figura 3, o desenvolvedor do componente pode criar casos de teste para cada interface pública do componente. Os casos de teste ficam listados na parte direita da tela (3), podendo a qualquer tempo ser selecionado e o código fonte de cada caso de teste, no formato JUnit, bem como a descrição deles podem ser vistos e editados na parte central da tela (2).

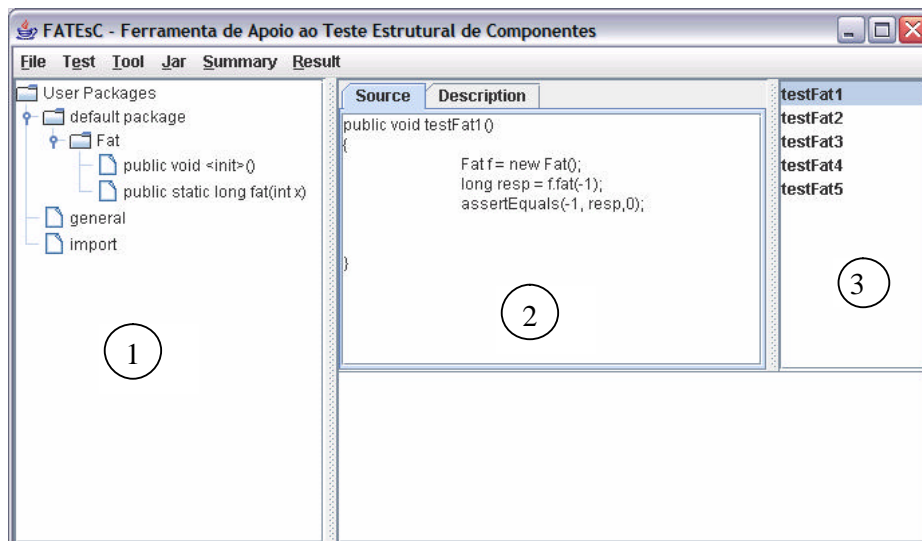


Figura 3. Tela principal da FATEsC: (1) hierarquia do componente, (3) lista de casos de teste e (2) código fonte do caso de teste testFat1.

Após a geração dos casos de teste, a FATEsC disponibiliza a funcionalidade de juntar todos os casos de testes em um arquivo único e compilá-lo. Os casos de teste

devem estar no formato JUnit (framework para desenvolvimento de teste de unidade em Java) (MASSOL e HUSTED, 2005) e nesse momento pode-se executá-los na Ferramenta JaBUTi.

As informações de cobertura dos testes, realizados na Ferramenta JaBUTi, são capturadas pela Ferramenta FATEsC e disponibilizadas no formato XML, ou seja, para cada caso de teste utilizado para testar uma determinada interface pública do componente, obtém-se os requisitos cobertos separados por critério de teste. Um requisito coberto pode ser um comando, um desvio, uma associação definição-uso ou outro elemento estrutural, definido pelo critério de teste que está sendo usado pelo desenvolvedor.

Os dados de cobertura dos casos de testes e as descrições dos casos de testes são disponibilizados junto ao componente, o que significa adicionar metadados XML gerados pela FATEsC ao arquivo contendo o bytecode do componente no formato .jar, que será usado pelo usuário do componente.

O usuário do componente, por sua vez, também utiliza a FATEsC para criar os casos de testes para a sua aplicação que utiliza o componente, o processo é o mesmo percorrido pelo desenvolvedor do componente, conforme Figura 4. Após a obtenção dos dados de cobertura dos seus casos de testes executados na Ferramenta JaBUTi, o usuário do componente, poderá fazer a análise da adequação de seus casos de testes, comparando-os com os dados de cobertura obtidos pelo desenvolvedor do componente.

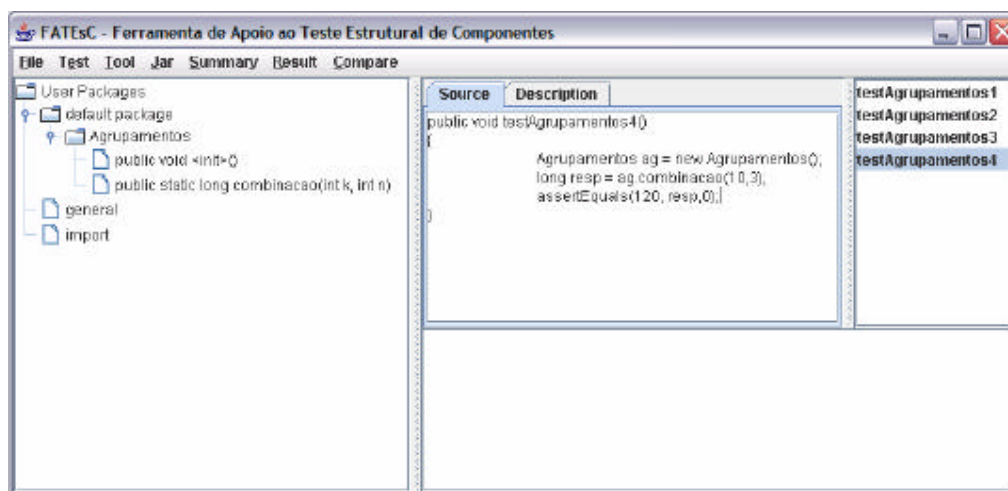
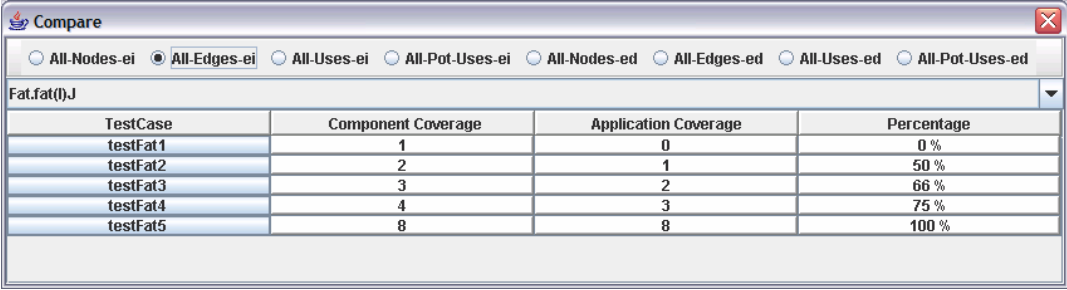


Figura 4. Tela principal da FATEsC: hierarquia da aplicação, lista de casos de teste e código fonte do caso de teste testAgrupamentos4.

O usuário deverá selecionar o critério e a interface pública do componente para o qual os resultados serão comparados, conforme apresentado na Figura 5. Na primeira coluna são listados os casos de testes gerados pelo desenvolvedor do componente. Na segunda coluna, são apresentados os dados de cobertura do componente, ou seja, quantidade de requisitos cobertos pelo caso de teste. Na terceira coluna são apresentadas as quantidades de requisitos cobertos pelos casos de teste do usuário em relação à cobertura obtida pelo desenvolvedor do componente e na quarta e última coluna o cálculo da porcentagem de cobertura obtida pelo usuário do componente em relação à cobertura obtida pelo desenvolvedor do componente. No exemplo, o caso de teste **testFat5** cobriu oito requisitos e os casos de teste do usuário também cobriram os

mesmos requisitos, portanto, a cobertura foi de 100%. Os requisitos cobertos, neste exemplo, referem-se a desvios do programa, visto que o critério escolhido é Todos-arcos.

Já nos outros casos de testes, a cobertura não foi a mesma, ou seja, os casos de testes do usuário do componente, não conseguiram cobrir os mesmos requisitos cobertos pelos casos de testes do desenvolvedor do componente, nesse caso é necessário que se faça uma análise com o objetivo de verificar se os casos de teste do usuário não estão adequados, ou se os casos de teste do componente, não podem ser reproduzidos no contexto da aplicação.



TestCase	Component Coverage	Application Coverage	Percentage
testFat1	1	0	0 %
testFat2	2	1	50 %
testFat3	3	2	66 %
testFat4	4	3	75 %
testFat5	8	8	100 %

Figura 5. Apresentação dos dados de cobertura obtidos pelo usuário e pelo desenvolvedor do componente.

Para ajudar o usuário do componente em sua avaliação, é possível visualizar as descrições dos casos de testes disponibilizadas pelo desenvolvedor, bastando para isso selecionar o caso de teste desejado. Para o caso de teste **testFat1**, conforme apresentado na Figura 6, a descrição diz que foi passado um argumento negativo para o método **fat** e como a aplicação do usuário não faz acesso à interface pública do componente para parâmetros $k < 1$, $n < 1$ e $k < n$ conclui-se que esse caso de teste não pode ser reproduzido para a aplicação. Nesse caso marca-se “infeasible” na tela de apresentação da descrição do caso de teste e o caso de teste é apresentado com uma cor diferente (cinza) representando a avaliação feita, conforme mostrado na Figura 7.

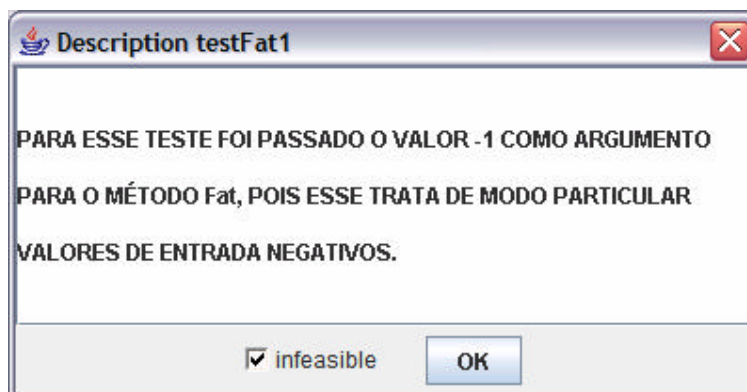


Figura 6. Apresentação da descrição do caso de teste testFat1.

TestCase	Component Coverage	Application Coverage	Percentage
testFat1	1	0	0 %
testFat2	2	1	50 %
testFat3	3	2	66 %
testFat4	4	3	75 %
testFat5	8	8	100 %

Figura 7. Aparência de um caso de teste (testFat1) que não poderá ser reproduzido na aplicação do usuário do componente.

Ao analisar o caso de teste **testFat4**, verificando a descrição do caso de teste apresentado na Figura 8 o usuário do componente perceberá que poderá alcançar a mesma cobertura que o caso de teste do componente, necessitando apenas criar um novo caso de teste que passe como parâmetros $k = 10$ e $n = 2$.

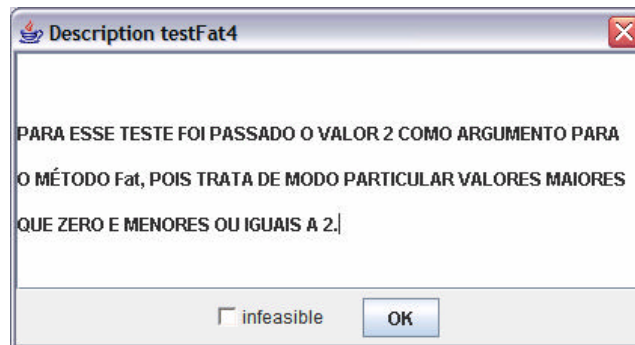


Figura 8. Apresentação da descrição do caso de teste testFat4.

Ao analisar o caso de teste **testFat2**, verificando a descrição do caso de teste apresentado na Figura 9, o usuário do componente perceberá que o cálculo do fatorial de 25 não é -1, então poderá se lembrar que o componente calcula o fatorial até o número 20 e que esta limitação deverá ser tratada em sua aplicação, isso mostra como a estratégia adotada por esse trabalho ajuda a revelar defeitos. Assim, o usuário do componente deveria continuar suas análises até que as coberturas se iguallassem ou se concluísse que alguns casos de testes não poderiam ser reproduzidos para a aplicação.

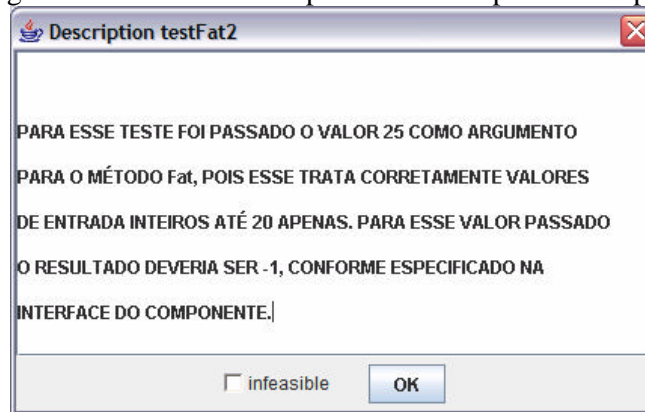


Figura 9. Apresentação da descrição do caso de teste testFat2.

3. Conclusão

Apesar de existirem algumas propostas na literatura abordando formas de solucionar efetivamente o problema da falta de informação, poucas tratam objetivamente de quais seriam essas informações e como as mesmas poderiam ser utilizadas. Por meio da utilização conjunta da Ferramenta FATEsC e JaBUTi o desenvolvedor do componente pode gerar casos de teste e executá-los, encapsulando ao componente os dados de coberturas obtidos, bem como as descrições dos casos de teste. O usuário do componente por sua vez, também pode gerar casos de teste para sua aplicação que utiliza o componente, executá-los e fazer uma avaliação da adequação dos casos de teste tendo como base para comparação os dados da cobertura dos testes obtidos pelo desenvolvedor do componente. Esta é uma forma de suprir a falta de informação, beneficiando o usuário do componente ao realizar a atividade de teste em sua aplicação.

Como forma de evolução desse trabalho é importante a condução de estudos de casos em ambientes de desenvolvimento de software baseado em componentes para avaliar a ferramenta, identificado os benefícios obtidos e possíveis adequações necessárias.

Referências

- BEYDEDA, S. and GRUHN, V. (2003) "State of the art in testing components", In: International Conference on Quality Software (QSIC), IEE Computer Society Press.
- MASSOL, V. and HUSTED, T. (2005) "JUnit Em Ação". Editora Ciência Moderna.
- VINCENZI, A., DELAMARO, M.E., WONG, E., MALDONADO, J.C. and SPOTO, E.S (2005) "Software baseado em componentes: uma revisão sobre teste". In: Desenvolvimento baseado em components: conceitos e técnicas. Editora Ciência Moderna.
- VINCENZI, A., DELAMARO, M.E., WONG, E. and MALDONADO, J.C. (2006) "Establishing Structural Testing Criteria for Java Bytecode". Software, Practice & Experience, v. 36, p. 1513-1541.