Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Bernardo Kaucher Darmstadter de Pádua

Uma técnica de priorização de testes em linhas de produtos de software

São Paulo
2013

Bernardo Kaucher Darmstadter de Pádua

Uma técnica de priorização de testes em linhas de produtos de software

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, como parte dos requisitos para a obtenção do título de Mestre em Engenharia de Software

Data da aprovação ____/____/_____

_____
Prof. Dr. Reginaldo Arakaki (Orientador)
IPT – Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Membros da Banca Examinadora:

Prof. Dr. Reginaldo Arakaki (Orientador)
IPT – Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Profa. Dra. Gabriela Maria Cabel Barbarán
Universidade de São Paulo

Prof. Dr. Mario Yoshikazu Miyake
IPT – Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Bernardo Kaucher Darmstadter de Pádua

Uma técnica de priorização de testes em linhas de produtos de software

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, como parte dos requisitos para a obtenção do título de Mestre em Engenharia de Software

Área de Concentração: Engenharia de Software

Orientador: Prof. Dr. Reginaldo Arakaki

São Paulo
Maio/2013

Ficha Catalográfica
Elaborada pelo Departamento de Acervo e Informação Tecnológica – DAIT
do Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT

P125u **Pádua, Bernardo Kaucher Darmstadter de**
Uma técnica de priorização de testes em linhas de produtos de software. / Bernardo Kaucher Darmstadter de Pádua. São Paulo, 2013.
85p.

Dissertação (Mestrado em Engenharia de Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Área de concentração: Engenharia de Software.

Orientador: Prof. Dr. Reginaldo Arakaki

1. Linha de produto 2. Produto de software 3. ATAM (Architectural Trade Off Analysis Method) 4. Teste de regressão 5. Garantia da qualidade 6. Qualidade de software 7. Arquitetura de software 8. Tese I. Arakaki, Reginaldo, orient. II. IPT. Coordenadoria de Ensino Tecnológico III. Título

13-45 CDU 004.415.53(043)

“Ciência é o que entendemos bem o suficiente para explicar a um computador. Todo o resto que fazemos é arte”.

Donald Knuth, 1996

Dedico este trabalho ao meu pai, cujo carinho e exemplo foram minha principal inspiração.

AGRADECIMENTOS

Ao Prof. Dr. Reginaldo Arakaki pela paciência, incentivo e conselhos prestados, durante todo o desenvolvimento deste trabalho.

Aos membros da banca examinadora, pelos conselhos e orientação compartilhados durante o exame de qualificação.

À empresa em que trabalho, pelo apoio financeiro, pelo incentivo e por permitir que dados fossem coletados em seus projetos

Aos colegas de trabalho, cujo empenho e dedicação tornaram possível a elaboração deste trabalho.

À minha noiva Viviane, cujo apoio e dedicação na forma de conselhos, carinho e revisões foram essenciais para a conclusão deste trabalho.

Finalmente, a todos aqueles que participaram direta ou indiretamente na elaboração deste trabalho.

RESUMO

Para atingir os benefícios que a produção de linhas de produtos de software traz, tais como um menor tempo de colocação no mercado e a alta reutilização de componentes, é necessário um alto nível de qualidade nos componentes produzidos. Em nenhuma das estratégias de teste em linhas de produtos de software revisadas por este trabalho, a arquitetura do núcleo é considerada o foco principal para a seleção de testes de regressão. Este trabalho propõe um método de priorização de testes nos componentes finais de uma linha de produtos de software, tal que as decisões arquiteturais tomadas durante o desenvolvimento do núcleo sejam utilizadas como critério para a seleção das principais atividades de teste. Para atingir o objetivo, este trabalho propõe a instanciação da técnica ATAM de análise da arquitetura para avaliar o núcleo de uma linha de produtos. Os artefatos obtidos pela aplicação da técnica serão utilizados para obter um conjunto priorizado de testes de regressão, que servirão de base para uma verificação da cobertura de teste de um conjunto de produtos finais desta mesma linha de produtos de software. Ao final da aplicação do método proposto neste trabalho espera-se determinar um conjunto de testes que auxilie futuras versões dos produtos finais avaliados a garantir a qualidade do seu conjunto de requisitos não funcionais. A principal contribuição deste trabalho será uma técnica de garantia da qualidade de software em linhas de produtos que possa ser aplicada a um ambiente onde o processo de engenharia de software utilize pouca documentação, ao mesmo tempo em que os principais requisitos não funcionais da arquitetura do núcleo são avaliados no produto final.

Palavras-chave: Teste de regressão, ATAM, linhas de produtos, qualidade de software, arquitetura de software

ABSTRACT

A test prioritization technique for software product lines

In order to achieve the benefits that the production of software product lines brings, such as time-to-market and high component reuse, a high level of quality in the produced components is necessary. However, in none of the strategies for software product lines analyzed in this work, the architecture of the core is considered the main focus for selecting regression test cases. This work proposes a test prioritization technique at the final components of a software product line, such that the architectural decisions taken during the development of the core are utilized as the selection criteria for the main test activities. To achieve this objective, this work proposes the instantiation of the ATAM technique of architectural analysis to evaluate the core of the product line. The artifacts produced by the technique will be used to produce a prioritized set of regression tests, which will serve as basis for a verification of the test coverage of a group of end products in this same software product line. At the end of the proposed method's application, it is expected to produce a set of tests that will help future versions of the end products in ensuring the quality of the non-functional requirements. The main contribution for this work is a quality assurance technique in software product lines that can be applied in an environment where the software engineering process uses minimal documentation while at the same time the main non functional requirements of the core architecture are evaluated at the end product.

Key words: Regression testing, ATAM, product lines, software quality, software architecture

Lista de ilustrações

Lista de tabelas

Lista de Abreviaturas e Siglas

ATAM	*Architecture Tradeoff Analysis Method*
ISO	*International Standard Organization*
SEI	*Software Engineering Institute*
UML	*Unified Modeling Language*
CFAG	*Component Function Access Graph*
OVM	*Orthogonal Variability Model*
J2EE	*Java 2 Platform, Enterprise Edition*
REST	*Representational state transfer*

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

O *Software Engineering Institute* (SEI) define uma linha de produtos de software como um grupo de produtos que partilha um conjunto comum e gerido de características, que satisfaz às necessidades específicas de uma missão ou mercado selecionado e que é desenvolvida a partir de um núcleo comum de recursos, de uma maneira prescrita (SEI, 2011).

Um dos recursos mais importantes desse núcleo comum é a arquitetura. A arquitetura incorpora as primeiras decisões de projeto de software. Essas decisões permitem ou impedem que seja atingida a qualidade desejada de um sistema, baseada em atributos como confiabilidade, modificabilidade ou interoperabilidade. (CLEMENTS; BERGEY; MASON, 2005).

Visando atingir os benefícios de uma linha de produtos de software, tais como a redução do prazo para colocação no mercado ou no custo de desenvolvimento, um alto nível de qualidade nos recursos reutilizados se faz necessário, tornando a garantia da qualidade uma parte crucial em qualquer esforço em linhas de produtos (GANESAN; KOLB; HAURY, 2007).

Para Ganesan, Kolb e Haury (2007), existem duas estratégias principais de garantia da qualidade para linhas de produtos de software: a garantia da qualidade com ênfase no produto, a qual tem como ênfase o produto que está sendo construído a partir do núcleo e é feita apenas como parte do processo de desenvolvimento do produto durante sua construção; e a garantia da qualidade com ênfase no núcleo, a qual tem como foco os recursos reutilizáveis do núcleo da linha de produtos.

Na primeira estratégia, a principal desvantagem é que um mesmo componente precisa ser testado mais de uma vez em uma grande quantidade de produtos. Outra desvantagem é que os problemas de qualidade nos componentes são encontrados tardiamente no processo de desenvolvimento, quando o componente já foi utilizado em diversos produtos (GANESAN; KOLB; HAURY, 2007).

Neto et al (2011b) inclui mais três estratégias de teste para linha de produtos de software:

- Teste incremental da linha de produtos, no qual o primeiro produto é testado individualmente, e técnicas de teste de regressão são usadas em produtos subsequentes.
- Reuso oportunista dos recursos de teste, quando os recursos produzidos para uma linha de produtos de software são reutilizados nas aplicações derivadas.
- Divisão de responsabilidades: Níveis específicos de teste são adequados a cada processo de linhas de produtos de software. Os recursos devem, em geral, ser verificados usando teste de unidade durante a construção do domínio, enquanto teste de integração, de sistema e de aceitação ocorrem durante sua instanciação, na construção da aplicação.

A adoção de uma técnica de seleção de casos de teste de regressão é útil em alguns cenários e domínios. Resumidamente, ela seleciona um conjunto de casos de teste a partir de suítes existentes para testar a versão original do produto, evitando a execução de todos os casos de teste. Na indústria aeronáutica, por exemplo, a redução de um caso de teste sem perdas na cobertura, pode gerar uma grande economia em recursos de teste (Harrold, 2001 apud Neto et al, 2010).

Em nenhuma das estratégias de teste em linhas de produtos de software analisadas por Neto et al (2011b) em sua revisão sistemática da literatura, a arquitetura do núcleo é considerada o foco principal para a seleção de testes de regressão em linhas de produtos de software. A seleção de testes de regressão sob a luz da arquitetura poderia garantir que as decisões tomadas durante a construção do núcleo sejam validadas durante o ciclo de testes do produto, podendo conseqüentemente gerar uma melhoria nas métricas de qualidade diretamente relacionadas a cada uma das abordagens arquiteturais. Pode-se afirmar, portanto que os trabalhos analisados carecem de técnicas de teste para os produtos finais em linhas de produtos que utilizem requisitos não funcionais provenientes da arquitetura do núcleo como critério de seleção.

1.2 Objetivo

O objetivo deste trabalho é propor um método de priorização de testes em produtos finais de uma família de produtos, tal que as decisões arquiteturais tomadas durante o desenvolvimento do núcleo sejam utilizadas como critério para a seleção das principais atividades de testes, tais como integração, regressão, unidade e aceitação. Espera-se que este critério de priorização leve a uma melhora das métricas de qualidade em um projeto onde hoje não se adota nenhum critério formal para a seleção de casos de teste.

1.3 Contribuições

Neste trabalho, a contribuição será a proposição de um método de priorização de casos de teste em famílias de produtos que utilize requisitos não funcionais da arquitetura do núcleo como critério de seleção. Este método difere de trabalhos anteriores que tratam da atividade de testes em linhas de produtos de software, tal como proposto por Neto et al.(2010), que utiliza grafos que descrevem modificações na arquitetura do produto como critério para seleção, ou tal como proposto por Jezequel, Le Traon e Nebut (2006), que utiliza geração automatizada dos testes a partir de um conjunto de requisitos descritos em um modelo UML.

Ao contrário dos métodos anteriores, nos quais a aplicação é adequada a grandes projetos que possuem requisitos descritos em uma linguagem pré-determinada tal como grafos ou UML, o método proposto por este trabalho pretende se adequar à reutilização por ciclos de manutenção em produtos finais de uma mesma família, em que o processo de desenvolvimento não prevê o levantamento de requisitos de qualidade nem a garantia destes requisitos.

1.4 Método de trabalho

O exemplo prático da aplicação do método proposto neste trabalho será elaborado em um laboratório de desenvolvimento de software de uma empresa multinacional da indústria de tecnologia da informação. O conjunto de projetos no qual este exemplo está inserido tem como objetivo a elaboração novas versões de diferentes produtos finais para uma mesma linha de produtos de software cujo núcleo tem como objetivo principal a gestão de ativos.

Cada projeto final avaliado tem como objetivo entregar uma nova versão de manutenção de seu respectivo produto final, construído a partir da ultima versão de manutenção do produto núcleo. Esta versão de manutenção contém alterações de requisitos solicitadas pelos clientes, bem como correções de defeitos internos e reportados externamente. A fim de garantir que nenhum novo efeito colateral seja inserido, testes de regressão e integração são executados a cada entrega. Entretanto, não existe um método ou processo formalmente definidos para a execução de testes que verifiquem os objetivos de qualidade da arquitetura do núcleo, ficando a cargo da experiência da equipe de desenvolvimento e testes decidir a seleção e cobertura dos testes executados.

- Levantamento de referências e detalhamento da técnica proposta

Inicialmente, é necessário levantar trabalhos disponíveis na literatura sobre a instanciação do ATAM bem como as técnicas para teste em famílias de produtos. Uma vez encontradas, estas referências devem ser compiladas e organizadas no contexto da técnica proposta, provendo sua fundamentação teórica. Juntamente ao detalhamento da técnica proposta, se inicia dos pré-requisitos necessários para a condução do exemplo prático, tais como definição de equipe avaliadora, escolha da linha de produtos e do produto final.

- Levantamento inicial da documentação arquitetural do produto

Como pré-requisito para os três passos iniciais do ATAM, deve ser levantada toda a documentação de alto nível previamente disponível para o produto núcleo avaliado, tais como requisitos de negócio, definição da arquitetura e requisitos detalhados do núcleo. O produto avaliado em questão é a ultima versão principal do núcleo de uma família de produtos de software, sendo este desenvolvido originalmente para gestão de ativos.

- Análise do ambiente onde o método proposto será aplicado

Nesta etapa, deve ser definida a equipe avaliadora, bem como o cronograma de reuniões e documentado quais as diferenças desta instanciação do ATAM quando comparado a sua definição formal pelo SEI. Em sua descrição original, o ATAM espera que a equipe avaliadora seja composta por inicialmente poucos indivíduos chave, e então em uma segunda etapa, a equipe avaliadora é ampliada para incluir

todos os envolvidos no projeto avaliado. Uma vez que o projeto avaliado se trata de núcleo de uma família de produtos de software, a equipe avaliadora será composta por cinco ou seis pessoas que trabalhem com diversos produtos que utilizam o mesmo núcleo, e na segunda fase do ATAM será ampliada para incluir a equipe de garantia da qualidade de um dos produtos finais.

- Instanciar o ATAM

Cada reunião tem como objetivo produzir parcialmente ou totalmente os artefatos descritos no método de análise arquitetural, sendo eles a lista de abordagens arquiteturais, a lista de cenários arquiteturais e a árvore de qualidade, que define a ligação entre os principais atributos de qualidade do núcleo avaliado, suas respectivas abordagens arquiteturais e para cada abordagem, o conjunto de cenários arquiteturais relacionados. Para atingir este objetivo, as reuniões devem seguir a ordem cronológica descrita nos passos 1 a 8 do método de análise da arquitetura.

- Analisar a documentação de teste disponível

Uma vez de posse dos resultados obtidos pela instanciação ATAM, o objetivo deste passo é selecionar um conjunto de produtos finais cujos resultados do ultimo projeto de manutenção serão avaliados. Para atingir tal objetivo, a documentação de teste disponível para cada produto final (documentos de projeto arquitetônico, casos de teste e baterias de teste regressão) deve ser levantada para análise posterior de cobertura.

- Análise de critério de cobertura e elaboração de casos de teste

Neste passo, o objetivo principal é produzir um conjunto de casos de teste que possibilite a avaliação da cobertura de testes em diversos produtos finais de uma mesma linha de produtos. O ATAM prevê que os cenários arquiteturais devem ser priorizados pela equipe avaliadora. Esta lista de cenários arquiteturais priorizados deve ser utilizada como fonte de definição de prioridade para a atividade de testes, uma vez que cada cenário está ligado a um conjunto de casos de teste. A partir do critério de cobertura escolhido, a lista de cenários deve ser transformada em um conjunto de casos de teste, de tal forma a possibilitar a avaliação da cobertura nos projetos de produtos finais. Esta lista de casos de teste tem como objetivo avaliar a

qualidade da cobertura de testes do produto final sob a ótica dos requisitos de qualidade da arquitetura do núcleo.

- Avaliação da cobertura de teste nos produtos finais

A partir da lista de casos de teste obtidos no passo anterior, a documentação de teste disponível nos projetos de desenvolvimento em cada um dos produtos finais escolhidos deve ser avaliada, analisando para cada um dos casos de teste obtidos dos cenários arquiteturais se estes estão cobertos por algum teste disponível neste projeto. Ao final da análise, deve ser preparado para cada projeto de produto final um relatório contendo quais casos de teste foram satisfeitos, bem como quais não foram e suas respectivas prioridades, de acordo com a priorização dos cenários arquiteturais. Os relatórios de cobertura e a lista de requisitos de teste poderão ser reaproveitados em projetos posteriores, servindo de critério para a especificação de novos testes.

1.5 Organização do trabalho

O restante da dissertação está organizado da seguinte forma:

O capítulo 2, Revisão bibliográfica, discute a técnica ATAM de análise da arquitetura, bem como o padrão de qualidade ISO-9126-1. Discute também técnicas de teste de software, bem como as principais técnicas de teste em linhas de produtos de software.

O capítulo 3, Técnica de priorização de testes em linhas de produtos, tem como objetivo detalhar a técnica apresentada neste trabalho, descrevendo a preparação para a instanciação do ATAM em uma família de produtos, o critério que foi utilizado para a priorização dos cenários arquiteturais, detalhando em seguida o método para a avaliação da cobertura de teste do produto final quando comparada aos resultados produzidos pela avaliação da arquitetura.

O capítulo 4, Aplicação do exemplo prático, é dividido em duas subseções. A primeira seção detalha o ambiente onde o exemplo prático será desenvolvido, contextualizando as características do projeto que será avaliado pelo método proposto no capítulo anterior. A segunda parte deste capítulo descreve a

implantação do exemplo prático propriamente dito, descrevendo quais adaptações foram necessárias para as técnicas utilizadas, a seqüência de atividades bem como os resultados obtidos. Na terceira parte deste capítulo é feita a análise dos resultados, detalhando os fatores que levaram a tomada das decisões descritas nas seções anteriores e as conseqüências dos resultados obtidos.

O capítulo 5, Conclusão. Apresenta a conclusão dos resultados obtidos na aplicação da técnica de priorização de testes em família de produtos durante um ciclo de manutenção.

2 REVISÃO BIBLIOGRÁFICA

2.1 Introdução

Neste capítulo serão apresentados os conceitos relevantes para o trabalho a ser desenvolvido, como o ATAM de análise da arquitetura do produto de software, o padrão de qualidade ISO-9126-1, teste de software em geral, linhas de produtos de software e suas técnicas de teste e garantia da qualidade.

O capítulo atual será organizado como se segue. Na seção 2.2 é introduzido o conceito de linha de produtos de software. Na seção 2.3, serão apresentadas as atividades essenciais do ATAM e na seção 2.3.1 é abordando o padrão de qualidade de software a ser utilizado neste trabalho durante a implantação do método. Na seção 2.4 são apresentados os conceitos e técnicas utilizadas em testes e na garantia da qualidade de software em geral. Na seção 2.5 são destacadas as peculiaridades do teste de software em linhas de produtos, bem como as diferenças entre as diversas técnicas disponíveis. Na seção 2.6 faz-se uma conclusão sobre os tópicos apresentados no capítulo.

2.2 Linhas de produtos

Muitas empresas estão chegando à conclusão que a prática de construir uma linha de produtos de software traz vários benefícios quantitativos em produtividade, tempo de colocação no mercado, qualidade do produto e satisfação do cliente. A prática de construir conjuntos de sistemas relacionados a partir de um mesmo núcleo compartilhado pode também satisfazer a forte demanda por personalização do software (Northtrop, 2002).

Northtrop (2002) afirma que existem nove áreas práticas de engenharia de software que devem ser dominadas para o desenvolvimento bem sucedido de linhas de produtos de software. Estas áreas são: definição da arquitetura, avaliação da arquitetura, desenvolvimento de componentes, compra e utilização de componentes disponíveis no mercado, prospecção de artefatos existentes, engenharia de requisitos, integração de sistemas de software, compreensão de domínios relevantes e teste. A figura 1 ilustra como estas áreas se relacionam.

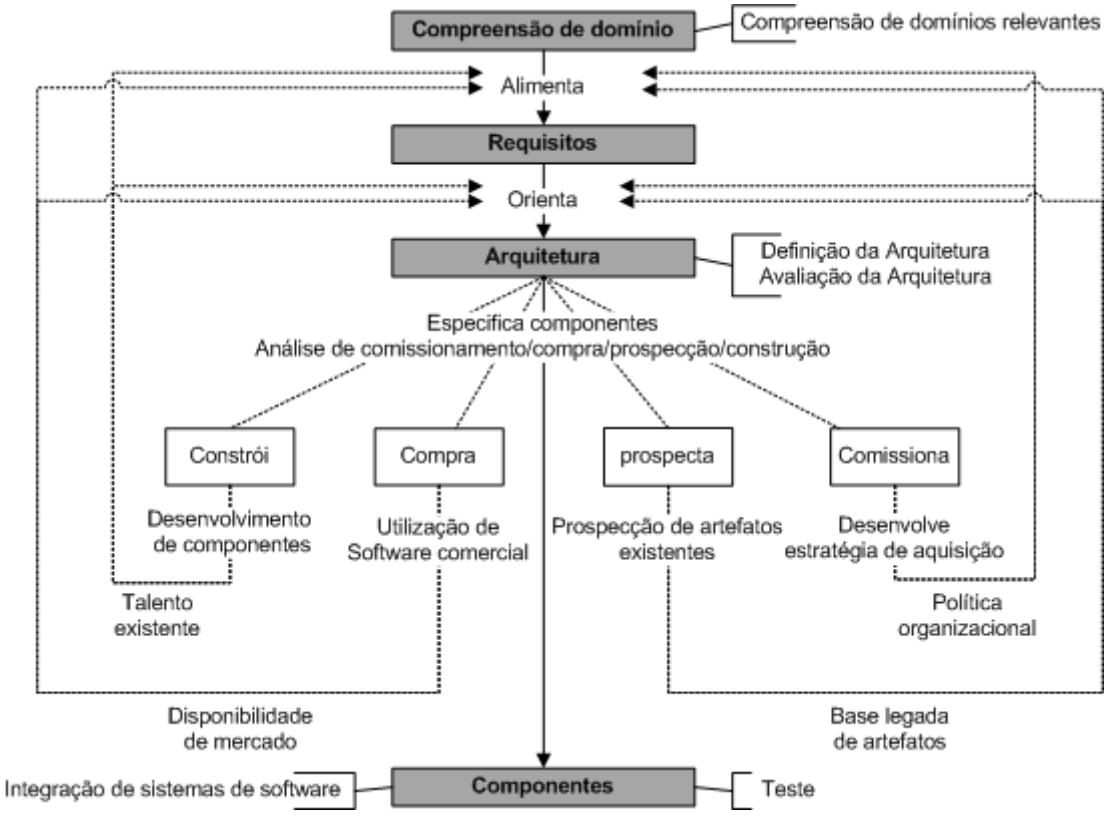

Figura 1 - Relacionamento entre áreas práticas da engenharia de software
Fonte: Northtrop (2002)

A compreensão de domínio alimenta os requisitos, que por sua vez orientam a arquitetura, que especifica componentes. Componentes podem ser feitos internamente, comprados no mercado, prospectados a partir de artefatos existentes ou comissionados por contrato. Esta escolha depende na disponibilidade de talentos internos, recursos, mercado, base pré-existente e prestadores de serviço. Sua existência ou não pode afetar os requisitos e a arquitetura da linha de produtos. Uma vez disponível, o componente deve ser integrado e, juntamente com o sistema, testado (Northtrop, 2002).

A arquitetura de software de um programa ou sistema computacional é a estrutura ou estruturas do sistema, que compreendem elementos de software, as propriedades visíveis externamente destes elementos e o relacionamento entre eles (BASS; CLEMENTS; KAZMAN, 2003).

SEI (2011) afirma que a arquitetura de uma linha de produtos de software vai além da dicotomia entre projeto e código. Esta arquitetura deve se preocupar em identificar e prover mecanismos para atingir um conjunto de variações explicitamente permitidas, uma vez que estas variações podem se tornar produtos. Escolher o

mecanismo apropriado de variação pode estar entre uma das tarefas mais importantes para um arquiteto de linha de produtos.

A arquitetura é um membro precoce e proeminente no conjunto de artefatos do núcleo. Espera-se que a arquitetura persista por todo o clico de vida da linha de produtos mudando relativamente pouco e lentamente. A arquitetura especifica um conjunto de componentes de software que povoam a base de artefatos do núcleo. A arquitetura, em conjunto com o plano de produção proporciona a descrição de como produtos são construídos a partir dos artefatos do núcleo. A arquitetura é usada, então, para criar arquiteturas para cada novo produto final de acordo com o processo de desenvolvimento da linha de produtos (SEI 2011).

2.3 ATAM

O ATAM é um método de avaliação arquitetural baseado em cenários, com ênfase nos objetivos de qualidade do produto. A entrada do método consiste na arquitetura, os requisitos de negócio de uma determinada linha de produtos de software e as perspectivas das partes interessadas envolvidas. O ATAM tem como um de seus objetivos a compreensão de como uma determinada abordagem arquitetural é utilizada para atingir um determinado objetivo de qualidade, bem como as consequências desta abordagem. (SEI, 2011)

O ATAM tem como principais saídas:

- O conjunto de cenários que as partes interessadas julgarem ser prioridade mais alta para atingir os objetivos de uso e qualidade do sistema e sua arquitetura
- A árvore de atributos que designa cenários específicos para os atributos de qualidade que se aplicam ao sistema avaliado
- Resultados específicos da análise, incluindo a identificação explícita de pontos sensíveis e pontos onde há interação entre as decisões tomadas durante o desenvolvimento da arquitetura.

O ATAM tem como propósito avaliar as consequências das decisões tomadas no desenvolvimento da arquitetura de um produto de software, tendo como referência

os requisitos não funcionais do produto (atributos de qualidade). Os principais objetivos do método podem ser definidos como: levantar e refinar uma definição dos requisitos de qualidade do produto de software, levantar e refinar afirmações a respeito das decisões tomadas no desenvolvimento da arquitetura e avaliar e determinar se as decisões arquiteturais tomadas atendem aos requisitos de qualidade satisfatoriamente (CLEMENTS; BERGEY; MASON, 2005)..

O ATAM não tem como objetivo prever precisamente métricas e comportamentos para os atributos de qualidade, uma vez que não existem dados suficientes para fazer tal predição durante um estágio inicial de desenvolvimento. Além de levantar a influência que as decisões arquiteturais exercem sobre os requisitos de qualidade, o método também tem como resultado a identificação de riscos, pontos sensíveis e pontos onde há interação entre as decisões tomadas durante o desenvolvimento da arquitetura (KAZMAN; KLEIN; CLEMENTS, 2000).

Kazman, Klein e Clements (2000) seguem definindo pontos sensíveis como pontos na arquitetura onde há uma grande correlação com um atributo de qualidade mensurável. Um exemplo é o número de requisições que um servidor atende, identificado como sendo dependente do processamento de uma única classe, influenciando diretamente a disponibilidade do produto como um todo. Define também como pontos de interação quando o parâmetro de uma abordagem arquitetural tem influencia direta sobre um atributo de qualidade ligado a outro ponto na arquitetura. Por exemplo, um aumento no número de requisições atendidas do exemplo anterior pode levar a uma redução na confiabilidade. Neste caso, a velocidade de comunicação é um ponto de interação.

Utilizando-se uma definição precisa do conceito de qualidade, o método é definido em (KAZMAN; KLEIN; CLEMENTS, 2000) como uma sucessão de nove passos, agrupados em quatro áreas: apresentação, investigação e análise, teste e relatório.

Os passos são então refinados e detalhados por Clements, Bergey e Mason (2005) que propõem a divisão do método em duas fases. Durante a primeira fase, o time avaliador é composto pelos principais tomadores de decisão do produto de software: clientes, líderes, arquitetos e gerentes. Na segunda fase, o grupo avaliador é expandido para incluir mais membros da equipe de desenvolvimento (programadores, testadores, administradores de sistema) bem como usuários.

Fase 1:

Apresentação

1. **Apresentar o ATAM**: Neste passo, os avaliadores explicam para as partes interessadas no produto de software, tipicamente representantes dos usuários, clientes e membros da equipe de desenvolvimento o ATAM.
2. **Apresentar objetivos de negócio**: Os representantes da equipe de desenvolvimento apresentam os objetivos de negócio que motivam o projeto, seu contexto e as motivações de qualidade, que servirão de ponto de partida para a identificação dos objetivos arquiteturais.
3. **Apresentar a arquitetura**: A equipe de desenvolvimento apresenta a arquitetura proposta, concentrando-se em como atender os objetivos de negócio.

Investigação e análise

4. **Identificar abordagens arquiteturais**: O time avaliador captura abordagens arquiteturais de alto nível e os atributos de qualidade específicos que são diretamente influenciados por estas abordagens. São adicionadas também à lista quaisquer abordagens que tenham sido identificadas durante o passo 3, ou durante o pré-exercício de revisão da documentação.
5. **Geração da árvore de atributos de qualidade**: Os participantes constroem uma árvore, partindo dos atributos de qualidade mais importantes para a arquitetura. Em seguida, são refinados na forma de cenários que descrevem situações de uso do produto e então priorizados.
6. **Analisar abordagens arquiteturais**: A partir dos atributos de qualidade de alta prioridade identificados no passo anterior, cada uma das abordagens arquiteturais é analisada, classificada e ligada a um atributo de qualidade. Neste ponto também são identificados os riscos para a arquitetura, pontos sensíveis e pontos de interação.

Fase 2:

Teste

7. **Levantar e priorizar cenários**: O time avaliador expandido produz mais cenários para atributos específicos de qualidade. Em seguida, todo o grupo prioriza os cenários, escolhendo os mais importantes.
8. **Analisar abordagens arquiteturais**: Este passo repete o processo utilizado no passo 6, mas para os cenários identificados no passo 7. Estes cenários são considerados casos de teste que podem confirmar análise feita até o momento.

Relatório

9. **Apresentar resultados**: Baseando-se nas informações coletadas pelo ATAM (arvore de qualidade, abordagens, cenários, pontos de interação e riscos), uma apresentação é feita com estes resultados e apresentada a todas as partes interessadas.

No passo 7 do ATAM, Kazman, Klein e Clements (2000) definem a técnica de priorização de cenários da seguinte maneira: cada participante recebe um número de votos equivalente a 30% do total de cenários. Estes votos podem ser organizados da forma que o avaliador desejar, votando tanto uma vez para cada cenário quanto utilizando todos os seus votos em apenas um cenário.

Ainda em Kazman, Klein e Clements (2000), cada cenário é priorizado também em relação a duas dimensões: a importância que cada nó tem para o sucesso do sistema e o grau de risco percebido representado pelo cumprimento deste cenário. Cada dimensão pode ser classificada em um de três valores: alto, médio e baixo.

Finalmente, apesar do ATAM produzir um conjunto de documentos que podem ser aplicados na melhoria do produto, o método proposto por Kazman, Klein e Clements (2000) não aborda como estes documentos devem ser utilizados, ficando a cargo da equipe de desenvolvimento utilizar estes artefatos no contexto do processo de engenharia de software instanciado para o desenvolvimento do produto avaliado.

Em linhas de produtos de software, um mesmo componente compartilhado por diversos produtos finais deve ser testado a cada instância, uma vez que a interação com diversos componentes diferentes pode levar a comportamentos variados (JEZEQUEL; LE TRAON; NEBUT, 2006). Apesar de não definir explicitamente como os documentos produzidos devem ser utilizados, o ATAM, quando aplicado ao núcleo da linha de produtos, gera artefatos que podem ser aplicados no projeto de testes dos componentes compartilhados pelos produtos finais.

2.3.1. Padrão de qualidade

Kazman, Klein e Clements (2000), na definição do ATAM, apresentam uma caracterização própria do conjunto de atributos de qualidade, listando 3 atributos: desempenho, modificabilidade e disponibilidade. Por se tratar de um trabalho anterior a criação da norma, esta caracterização não apresenta o mesmo escopo e padronização do conjunto propostos na norma ISO-9126-1, Qualidade de Produto.

Desharnais, Abran e Suryn (2011) descrevem a norma ISO-9126-1 como um modelo de qualidade para produtos de software, bem conhecida no meio acadêmico e na indústria de software, se dividindo em três sub-modelos de qualidade (externa, interna e em uso), dez características de qualidade, 27 sub-características e mais de 250 métricas.

Conforme definido na norma NBR ISO/IEC 9126-1 (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2003), os características de qualidade e suas sub-características são:

Modelos de qualidade interna e externa:

- Funcionalidade: adequação, acurácia, interoperabilidade, segurança de acesso e conformidade relacionada à funcionalidade.
- Confiabilidade: maturidade, tolerância a falhas, recuperabilidade e conformidade relacionada à confiabilidade.
- Usabilidade: inteligibilidade, apreensibilidade, operacionalidade, atratividade, conformidade relacionada à usabilidade.
- Eficiência: comportamento em relação ao tempo, utilização de recursos, conformidade relacionada à eficiência.
- Manutenibilidade: analisabilidade, modificabilidade, estabilidade, testabilidade, conformidade relacionada à manutenibilidade.

- Portabilidade: adaptabilidade, capacidade para ser instalado, coexistência, capacidade para substituir, conformidade relacionada à portabilidade.

Modelo de qualidade em uso: eficácia, produtividade, segurança e satisfação.

A norma de qualidade NBR ISO/IEC 9126, em suas partes 2, 3 e 4, propõe mais de 250 métricas para as características e sub-características, mas não definem um processo para a coleta destas métricas.

Para a instanciação do ATAM, é necessária a adoção de uma definição para o conceito de qualidade de software, uma vez que as abordagens arquiteturais estão diretamente ligadas a um ou mais requisitos de qualidade. Para este trabalho, o padrão adotado será NBR ISO/IEC 9126, visto que este padrão é amplamente difundido no meio acadêmico e indústria.

2.4 Testes

Uma das distinções mais importantes a se fazer em testes de software é entre validação e verificação. O padrão IEEE 610.12 (1990) as define:

- Verificação: Processo de avaliar um sistema ou componente durante ou ao fim do processo de desenvolvimento a fim de determinar se satisfaz determinados requisitos.
- Validação: Processo de avaliar se os requisitos de um sistema ou componente estão corretos e satisfazem as condições impostas durante o começo do desenvolvimento

A verificação é normalmente uma atividade técnica que utiliza o conhecimento de artefatos individuais. Já a validação usualmente depende do conhecimento do domínio da aplicação para qual o software está sendo desenvolvido. Por exemplo: um software para aviões depende do conhecimento de engenheiros aeronáuticos e pilotos (AMMANN; OFFUTT 2008).

O teste é utilizado de duas formas fundamentais em um produto: durante as fases do desenvolvimento para verificar se um produto está correto e também para validar o produto em relação aos seus requisitos (SEI 2011)

Ammann e Offutt (2008) definem que testes podem ser derivados de requisitos e especificações, artefatos de projeto ou até mesmo o código fonte. Um nível diferente de teste acompanha cada uma das atividades de desenvolvimento:

- Teste de aceitação: avalia o software em relação aos requisitos

- Teste de sistema: avalia o software em relação ao projeto arquitetônico
- Teste de integração: avalia o software em relação ao projeto de subsistemas
- Teste de modulo: avalia o software em relação ao projeto detalhado
- Teste de unidade: avalia o software em relação à implementação

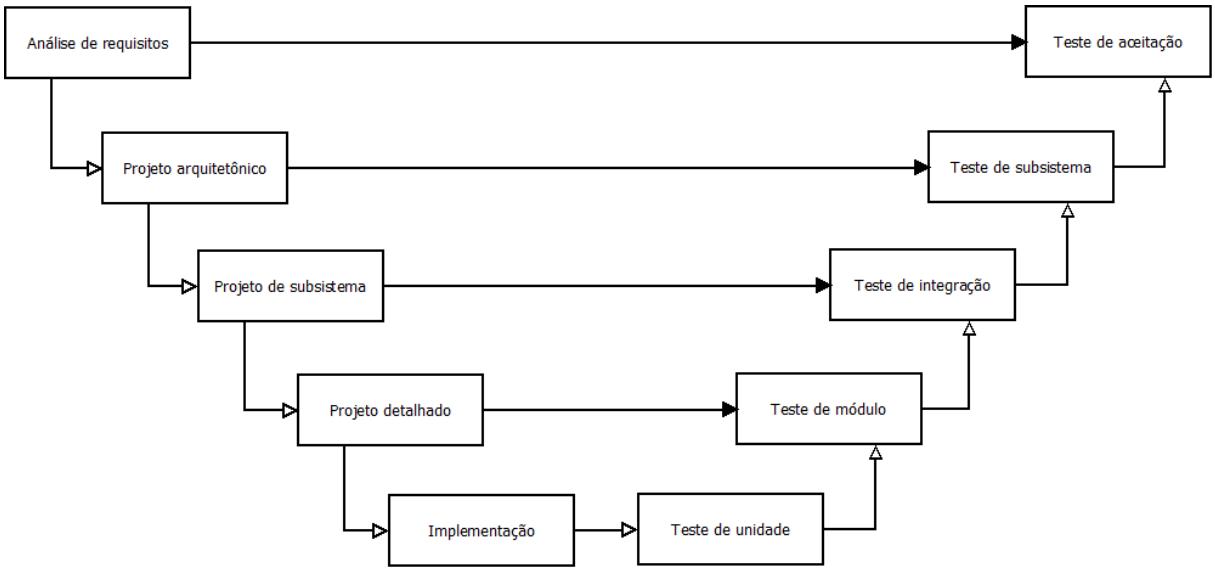

Figura 2 - Atividades de desenvolvimento de software e seus respectivos níveis de teste – o modelo-V

Fonte: Ammann e Offutt (2008)

O processo de teste envolve quatro atividades principais, que são executadas em cada um dos níveis. O planejamento, que envolve a coordenação de pessoal, gestão de recursos e equipamento. O projeto do teste, que é baseado no nível a ser executado e a técnica adotada em particular. A execução abrange os passos necessários para realizar os testes e scripts criados na atividade anterior. A Geração de relatório, atividade em que todos os resultados da execução são avaliados para determinar se o teste gerou a saída esperada. Esta atividade é importante para calibrar o teste, identificando problemas durante as outras atividades (Neto et al., 2011b, p 21).

Uma das principais limitações do processo de teste de software é a sua incapacidade de evidenciar a ausência de falhas (apenas a presença de falhas é indicada). Esta é uma limitação fundamental e teórica, uma vez que o problema de encontrar todas as falhas em um determinado programa é indecidível (AMMANN; OFFUTT 2008).

Duas estratégias básicas de teste são usadas para contribuir com o projeto de casos de teste: caixa-preta (ou funcional) e caixa-branca (limpo, ou caixa de vidro). Ambas as estratégias devem ser utilizadas para se atingir um software de alta qualidade, uma vez que se complementam (Neto, 2011b, p 25).

Usando a abordagem de caixa-preta, o engenheiro de teste considera o software testado como uma caixa opaca, não possuindo nenhum conhecimento sobre sua estrutura interna. A descrição do comportamento, ou funcionalidade para o software pode vir de alguma documentação formal ou um conjunto bem definido de pré e pós-condições (Burnstein, 2010).

Já a abordagem de caixa branca enfatiza a estrutura interna do software a ser testado. Para o projeto de casos de teste usando esta estratégia, o engenheiro deve possuir um grande conhecimento desta estrutura. O código-fonte, ou alguma representação apropriada em pseudocódigo deve estar disponível. O testador seleciona casos de teste para exercitar elementos específicos da estrutura interna e determinar se eles estão funcionando corretamente. Uma vez que o projeto, execução e análise dos resultados de um teste caixa-branca são extremamente demorados, esta estratégia é geralmente aplicada a partes menores do software, tal como um módulo ou função (Burnstein, 2010).

2.5 Critério de cobertura de teste

Dado um trecho de código e o conjunto de todas as entradas possíveis para este trecho, é inviável a execução exaustiva de todos os testes resultantes desta combinação. A escolha de um critério adequado é feita tendo como objetivo maximizar a cobertura de teste e conseqüentemente a qualidade do software, dado um esforço limitado para a execução.

Bertolino e Marré (1996) afirmam que um critério de cobertura de testes na prática define um conjunto de requisitos a serem cumpridos. Em particular, para um critério de cobertura estrutural estes requisitos estão mapeados a um conjunto de entidades no programa que devem ser cobertos quando os testes são executados. Estas entidades podem ser derivadas da estrutura de fluxo de controle ou de dados.

Ammann e Offutt (2008) definem formalmente o critério de cobertura como uma coleção de regras que impõe requisitos de teste em um conjunto de testes, enquanto

requisitos de teste são elementos específicos de um software que um caso de teste deve satisfazer.

O modelo utilizado para avaliação de cobertura de testes descrita inicialmente em Bertolino e Marré (1996) e então detalhado em Ammann e Offutt (2008) consiste em modelar os requisitos de teste na forma de grafos, sendo os casos de teste equivalentes a cada caminho percorrido neste grafo.

Bertolino e Marré (1996) afirmam que o fluxo de controle de um programa pode ser mapeado em um grafo direcionado de diferentes maneiras. Em seu trabalho foi adotado o modelo que é chamado grafodd (grafo decisão-para-decisão). Neste modelo, um nó está associado a uma decisão, um ponto onde o fluxo de controle diverge ou uma junção, um ponto onde o fluxo de controle converge. Uma aresta é associada a um segmento, ou seja, uma sequencia contínua de declarações. Um caminho completo em um grafodd para um determinado grafo G é o caminho percorrido entre a aresta de entrada e a aresta de saída.

Ammann e Offutt (2008) aperfeiçoam o modelo apresentado por Bertolino e Marré (1996) ao utilizar diretamente o conceito de grafos direcionados no lugar do grafodd, bem como ao expandir a aplicação deste modelo não só ao apresentar critérios de cobertura adicionais para testes de caixa branca, mas também ao propor uma técnica para modelar a especificação de casos de uso e inferir casos de teste a partir deste modelo. Esta técnica propõe modelar o fluxo principal de um caso de uso e suas alternativas na forma um único diagrama de atividade, onde os casos de teste podem ser inferidos a partir de caminhos percorridos no diagrama.

Dentre os critérios estruturais de cobertura apresentados em Bertolino e Marré (1996) e Ammann e Offutt (2008), podemos destacar:

- Alternativa: Cada alternativa do programa (todas as divergências possíveis do fluxo de controle) devem ser percorridas pelo menos uma vez por algum teste
- Declaração: Forma mais simples de teste do fluxo de controle. Todas as declarações do programa devem ser percorridas pelo menos uma vez por algum teste
- Cobertura de nó: Para todos os nós de um determinado grafo G, no conjunto de testes deve existir ao menos um caminho que percorra cada um dos nós.

- Cobertura de caminhos especificados: Os testes deve conter um conjunto S de caminhos de teste, onde S é passado como parâmetro.

Enquanto as técnicas descritas anteriormente se destinam principalmente a testes de unidade ou caixa branca, ambos Burnstein (2010) e Myers, Sandler e Badgett (2011) fazem uma revisão das metodologias disponíveis para testes de caixa preta. Dentre estes métodos podemos destacar:

- Particionamento de classe de equivalência: Consiste em selecionar um subconjunto do conjunto de todas as possibilidades de entrada em um determinado programa, de forma tal que maximize a probabilidade de encontrar erros. Cada subconjunto representa uma classe de entradas equivalentes, e o testador pode escolher um teste que seja representativo desta classe.
- Análise de valor limite: Enquanto no particionamento de classe de equivalência qualquer teste da classe de equivalência pode ser selecionado, a análise de valor limite exige que sejam selecionados aqueles cujas entradas estejam perto do valor limite de forma tal que o conjunto de testes resultante cubra as bordas superiores e inferiores da classe de equivalência.
- Grafos de causa e efeito: Trata-se de uma técnica que modela a especificação na forma de um grafo que representa um circuito digital simples, utilizando uma notação simplificada. Uma causa é uma entrada distinta ou classe de equivalência de condições de entrada. Um efeito é uma condição de saída, ou uma transformação no sistema. Uma vez identificadas causa e efeito, o conteúdo semântico da especificação deve ser analisado e transformado em um grafo booleano, ligando-as. Ao metodicamente rastrear as condições de estado, o grafo é convertido em uma tabela de decisão. Cada coluna na tabela representa um caso de teste.
- Adivinhação de erros: Esta técnica é em grande parte um processo intuitivo e ad hoc. Consiste em dado um programa em particular, supor tanto por intuição quanto por experiência certos tipos de erros prováveis e então escrever casos de teste que expõe estes erros.

Dentre outros trabalhos que abordam o problema do critério de cobertura de teste, podemos destacar Gao, Espinoza e Jingsha (2005), que apresentam um conjunto de critérios de cobertura para testes de caixa preta em componentes que utilizam o modelo CFAG (Component Function Access Graph). A partir deste trabalho, Hashim, Ramakrishnan e Schidt (2007) apresentam um modelo probabilístico de cobertura de teste arquitetural em testes de integração baseado em componentes, onde alguns critérios de cobertura são adaptados para a utilização em testes de caixa branca. Cohen, Dwyer e Shi (2006) discutem os problemas da cobertura de teste em linhas de produtos de software e apresenta um critério de cobertura para família de produtos que utilizam a modelagem OVM (*Orthogonal Variability Model*).

2.6 Esquema de integridade baseado em risco

Apresentando o conceito de impacto e frequência que uma falha pode ter sobre a operação de um software, a norma IEEE 1012-2012 sugere a utilização de um esquema de criticidade em duas dimensões: consequência, que descreve o possível impacto que o erro pode ter sobre o sistema e frequência que descreve a probabilidade de ocorrência de um estado que possa contribuir para o erro do sistema.

Existem sobreposições entre os níveis de integridade para permitir a interpretação individual de risco aceitável dependendo da aplicação. (IEEE Std 1012, 2012)

A tabela 2 ilustra o esquema de integridade mostrado na tabela 1. Cada célula na tabela atribui um nível de integridade baseado na combinação entre a consequencia de um erro e a propabilidade de ocorrência de um estado operacional que contribua para o erro. Algumas células refletem mais de um nível de integridade, indicando que a atribuição final pode ser selecionada para levar em consideração o sistema e as recomendações de mitigação de risco. (IEE Std 1012, 2012)

Tabela 1 - Definição de consequências

Consequência	Definição
Catastrófica	Perda de vida humana, falha completa da missão, perda da segurança do sistema, perda financeira ou social extensa.
Crítica	Lesão ou doença grave e permanente, perda parcial da missão, dano grave ao sistema, perda financeira ou social grave.
Ocasional	Lesão ou doença grave, degradação de missão secundária, alguma perda financeira ou social.
Insignificante	Pequena lesão ou doença, baixo impacto no desempenho do sistema ou inconveniência para o operador.

Fonte: Norma IEEE 1012-2012

Tabela 2 - Mapa de atribuição de níveis de integridade

Erro	Probabilidade de ocorrência de um estado que contribua para o erro			
Consequência	**Razoável**	**Provável**	**Ocasional**	**Infrequente**
Catastrófica	4	4	4 ou 3	3
Crítica	4	4 ou 3	3	2 ou 1
Marginal	3	3 ou 2	2 ou 1	1
Insignificante	2	2 ou 1	1	1

Fonte: Norma IEEE 1012-2012

2.7 Estado da arte de testes em linhas de produtos de software

A atividade de testes em uma linha de produtos examina primeiramente o produto núcleo, em seguida os componentes específicos de um produto e suas interações e por ultimo o produto completo. As responsabilidades para testar uma linha de produtos podem ser distribuídas diferentemente de um produto de software comum. Ao contrário de sistemas únicos, esta é uma atividade cujos artefatos produzidos são utilizados em produtos diferentes (SEI, 2011).

SEI (2011) segue definindo alguns aspectos fundamentais do teste em linhas de produtos, sendo dentre eles:

- Testar todos os artefatos o mais cedo possível durante o ciclo de vida de desenvolvimento. A interação entre os produtos pode levar a uma explosão combinatória de casos de teste
- Os testes devem ser estruturados de forma a acomodar a grande variação entre os diferentes produtos em uma mesma linha

SEI (2011) faz a distinção da atividade de teste em uma linha de produtos em duas partes: teste dos componentes do núcleo e teste durante o desenvolvimento de produtos.

Em um esforço de desenvolvimento de linha de produtos, uma das atividades principais é a montagem de produtos a partir do núcleo. Grande parte da documentação do produto será gerada de forma genérica. Durante o teste, um conjunto completo de testes funcionais deve ser especificado também de forma genérica (SEI, 2011).

Já durante o desenvolvimento de produtos, o conjunto de testes deve ser definido a partir da documentação genérica criada no desenvolvimento do núcleo. Mesmo que um componente em específico tenha sido testado exaustivamente, a integração a outros produtos e componentes pode levar a novas falhas. Os testes criados anteriormente podem ser derivados em testes de interação que devem garantir que as novas adições ao produto final não gerem falhas (SEI, 2011).

Tevanlinna, Taina e Kauppinen (2004) afirmam que o processo de teste em linhas de produtos de software se diferencia do processo tradicional de teste pela presença inerente do reuso. Ao testar apenas um produto é possível projetar e implementar diretamente os ativos de teste. Na abordagem em famílias de produtos de software pode se criar e gerenciar ativos de teste reutilizáveis que podem ser aplicados a diversos produtos de forma a reduzir custos.

Northtrop (2002) afirma que apesar do reuso como estratégia para diminuir o custo de produção de software não ser uma idéia nova, na abordagem de linhas de produtos de software a reutilização de artefatos é planejada e obrigatória, uma vez que a base reutilizada inclui artefatos de software que seriam muito caros para se desenvolver sozinhos.

Segundo Burnstein (2010), Depois que os componentes em um sistema tenham sido testados individualmente através de testes de unidade e testes de integração, a equipe de desenvolvimento inicia a verificação do sistema como um todo. O objetivo desta fase de teste é garantir que o sistema funciona de acordo com os seus

requisitos quando todos os seus módulos e componentes são montados. Testes de sistema avaliam tanto o comportamento funcional do produto, quanto seus requisitos não funcionais (qualidade).

Para Ammann e Offutt (2008), o teste de sistema é projetado para determinar se o sistema montado atinge a sua especificação. Ele assume que as peças funcionam individualmente, enquanto questiona o sistema como um todo. Este nível de teste normalmente procura por problemas de arquitetura e especificação de requisitos.

Em uma linha de produtos de software que utiliza a divisão de responsabilidades como estratégia de teste, onde os níveis de teste do modelo-V são distribuídos entre os diversos componentes, as atividades de teste de integração, sistema e aceitação podem ser delegadas aos produtos finais. (TEVANLINNA; TAINA; KAUPPINEN, 2004)

Ganesan, Kolb e Haury (2007) compararam duas estratégias diferentes para o problema de teste em linhas de produtos: garantia da qualidade com ênfase no produto e a garantia da qualidade com ênfase na infraestrutura.

Na estratégia de garantia da qualidade com ênfase no produto, a maior parte do esforço é direcionada ao produto final construído a partir do núcleo e é executada apenas como parte do processo de desenvolvimento do produto final. A principal desvantagem dessa estratégia é que um mesmo componente pode ser utilizado em vários produtos finais diferentes e defeitos nos componentes são detectados apenas muito tarde no processo de desenvolvimento, quando já se espalharam por diversos produtos finais. Além disso, muito esforço é gasto em atividades de garantia da qualidade redundantes (GANESAN; KOLB; HAURY, 2007).

Na estratégia de garantia da qualidade com ênfase no núcleo, o esforço de teste é direcionado na reutilização de artefatos da linha de produtos. Os componentes destinados a reutilização são rigorosamente testados. Em casos extremos, os produtos são construídos apenas a partir de componentes que foram extensivamente assegurados e qualificados e, portanto é assumido que o produto funcione corretamente. A genericidade dos componentes, entretanto, torna a garantia da qualidade problemática, uma vez que é economicamente impraticável testar todas as combinações e interações de componentes entre os diversos produtos finais em uma mesma linha de produtos de software (GANESAN; KOLB; HAURY, 2007).

A problemática da explosão combinatória de testes na interação entre componentes mostra que o direcionamento do esforço de teste para o núcleo ao adotar a estratégia de garantia da qualidade com ênfase no núcleo não exime os produto finais da atividade de teste da interação entre os componentes r

Ambos os estudos SEI (2011) e Ganesan, Kolb e Haury (2007) afirmam que métodos de verificação da arquitetura do núcleo podem ser usados como meio de garantir a consistência entre os produtos finais, o núcleo e seus componentes.

Muccini, Inverardi e Bertolino (2004) abordam a questão das estratégias de teste orientadas a arquitetura, onde usa modelos que descrevem a dinâmica da arquitetura de software para identificar interações entre componentes do sistema e então selecionar testes relevantes ao comportamento da arquitetura, sem abordar, entretanto, as peculiaridades de uma arquitetura para linhas de produtos de software.

Em uma revisão sistemática da literatura, Neto (2011a, p.45) expande a lista de estratégias de teste para linhas de produtos. Estas estratégias são: teste incremental de produto por produto, reutilização oportunista dos casos de teste e a divisão de responsabilidades.

Na estratégia de teste incremental, o primeiro produto é testado individualmente, enquanto os produtos consecutivos são testados utilizando-se técnicas de regressão, de forma a garantir que os componentes do produto ainda funcionem conforme especificado (Neto 2011a, p.45).

Já na estratégia de reuso oportunista dos casos de teste, os artefatos produzidos para o primeiro produto final desenvolvido a partir do núcleo são derivados e reutilizados nos produtos consecutivos. Esta forma de reuso não é utilizada sistematicamente e, portanto não existe técnica que auxilie na seleção de casos de teste (Reuys et al, 2006 apud Neto 2011a, p.45).

A estratégia de divisão de responsabilidades refere-se à seleção de diferentes níveis de teste aplicados dependendo do objetivo de cada fase. Esta divisão pode ser vista claramente quando os artefatos são submetidos a testes de unidade no desenvolvimento do núcleo e ao instanciar o desenvolvimento de um produto final, testes de integração, sistema e aceitação são executados (TEVANLINNA; TAINA; KAUPPINEN. 2004).

Ao testar uma funcionalidade específica proveniente do núcleo, casos de teste específicos podem ter que ser reescritos para cada produto final. Como resultado,

escrever os casos de teste para todos os produtos é impraticável (JEZEQUEL; LE TRAON; NEBUT, 2006). O ATAM (*Architecture Tradeoff Analysis Method*) para análise da arquitetura do núcleo se apresenta como um método capaz de capturar e assegurar que aspectos relevantes da arquitetura do núcleo da família de produtos sejam reutilizados em projetos que aplicam a estratégia de divisão de responsabilidades como forma de reduzir custos, garantindo que as atividades de teste executadas posteriormente nos produtos finais levem em consideração os requisitos arquiteturais do núcleo.

2.8 Conclusão

Neste capítulo foram apresentados os conceitos relevantes ao projeto a ser desenvolvido, como linhas de produtos de software, a técnica ATAM de avaliação da arquitetura, o padrão de qualidade NBR ISO/IEC-9126-1, bem como alguns conceitos de teste.

Inicialmente apresentando uma técnica de verificação da arquitetura de um produto de software, indicando ao final desta subseção que está técnica pode ser utilizada em linhas de produtos para garantir a consistência entre o núcleo e seus componentes, servindo de ponto de partida para o trabalho a ser desenvolvido.

Nas subseções seguintes é apresentado o modelo de qualidade que será utilizado durante a aplicação do ATAM e em seguida é feita uma breve revisão dos conceitos mais importantes em testes de software, e finalmente uma revisão do estado da arte em testes de linhas de produtos de software.

3 TÉCNICA DE PRIORIZAÇÃO DE TESTES EM LINHAS DE PRODUTOS

Um dilema constante na disciplina de teste de software é encontrar o equilíbrio entre o orçamento disponível em um projeto e atingir o nível desejado de qualidade no produto. Uma vez que na maioria dos casos é economicamente inviável um teste exaustivo, é necessário priorizar o conjunto de testes de acordo com um determinado critério.

Linhas de produtos de software se baseiam na reutilização de componentes e arquitetura entre vários produtos, para desta forma diminuir os custos de desenvolvimento. Isto cria novos problemas para a disciplina de testes, onde a interação entre diversas combinações de componentes devem ser verificadas.

Nas diversas estratégias de teste em linhas de produtos de software analisadas por este trabalho, o objetivo principal é garantir uma divisão de responsabilidades de teste entre o núcleo e os componentes de tal forma que o benefício da reutilização de componentes não seja perdido. Em nenhuma destas estratégias foi apresentada uma técnica de priorização de testes dentro dos componentes de forma a validar os requisitos de qualidade da arquitetura do núcleo diretamente nos produtos finais, garantindo assim a manutenção dos objetivos de qualidade da arquitetura.

3.1 Introdução

A proposta apresentada neste trabalho é uma técnica para o levantamento e priorização de testes não funcionais em uma linha de produtos de software. Para tanto, propõe-se utilizar o ATAM de análise arquitetural para levantar as metas de qualidade do núcleo de uma linha de produtos, e seu consequente impacto nos requisitos não funcionais dos produtos finais em nesta mesma linha de produtos de software.

A técnica se organiza em três passos principais (figura 3): a instanciação do ATAM em uma linha de produtos, mais especificamente seus passos 1 a 7; a priorização dos cenários arquiteturais, utilizando uma personalização do critério de priorização dos cenários, diferentemente do critério adotado pelo ATAM em sua definição original; e finalmente a análise da cobertura de testes, onde a partir do conjunto de cenários priorizados, é verificado se o conjunto de testes disponíveis

atende aos requisitos não funcionais de onde se originaram os cenários arquiteturais. Para aqueles requisitos onde de acordo com o critério de cobertura adotado houver deficiência, novos testes devem ser especificados.

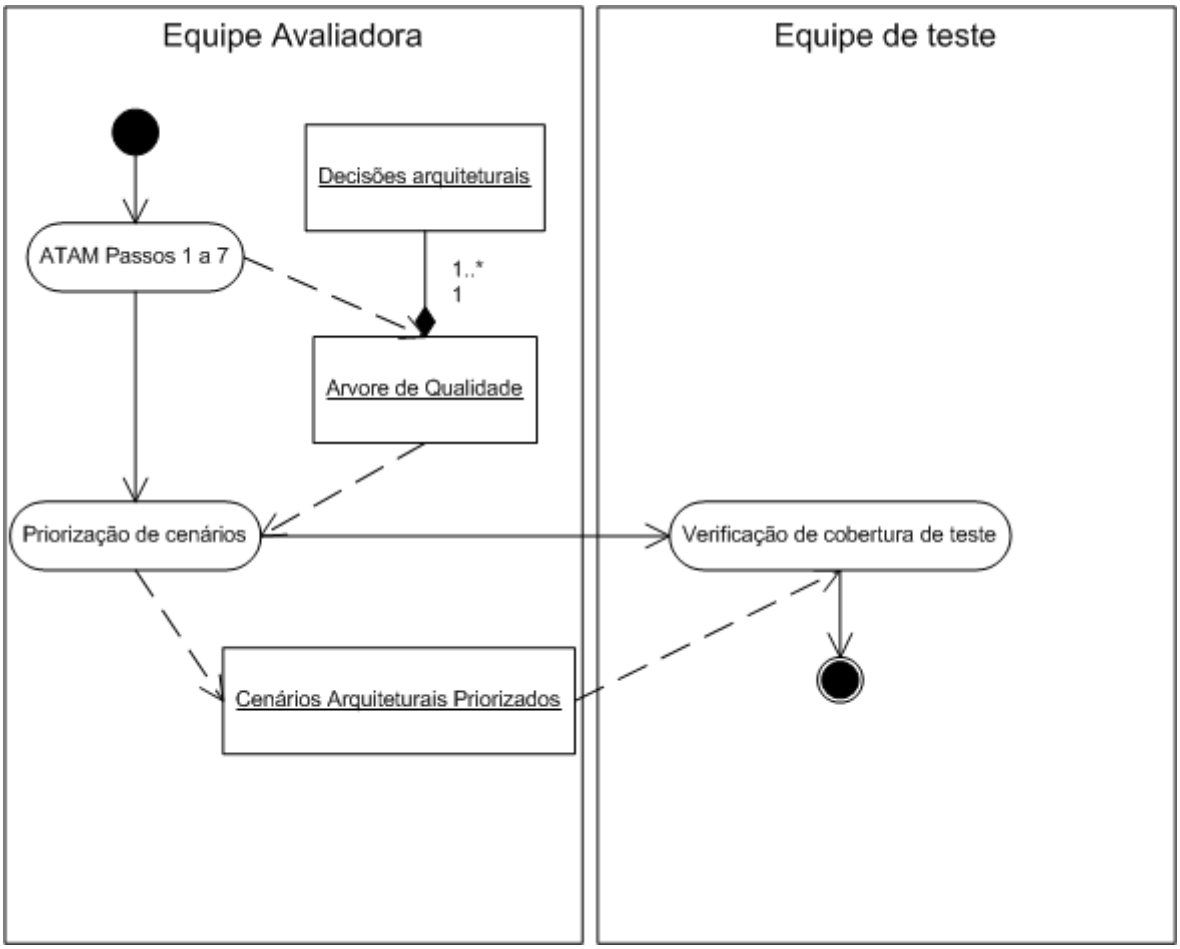

Figura 3 - Diagrama de atividades da técnica proposta

3.2 Avaliação arquitetural do núcleo em uma linha de produtos

SEI (2011) afirma que para uma arquitetura ser bem sucedida, as suas restrições devem ser bem conhecidas. Ao contrário dos modelos de engenharia de software em cascata, as restrições de uma arquitetura em linhas de produtos vão muito além da implementação de um determinado requisito funcional.

SEI (2011) considera a arquitetura do núcleo como um dos artefatos mais importantes de uma linha de produtos de software, afirmando ser a chave para o

sucesso de qualquer projeto, uma vez que os atributos de qualidade de um sistema (tais como desempenho, modificabilidade e disponibilidade) são em grande parte, permitidas ou impedidas por esta arquitetura.

Nesta etapa da técnica proposta por este trabalho, o objetivo principal é a identificação dos dois primeiros itens listados como saídas principais do ATAM: o conjunto de cenários arquiteturais e a árvore de atributos de qualidade.

Um cenário é uma afirmação curta que descreve a interação de uma das partes interessadas com o sistema. Um usuário descreveria um cenário utilizando o sistema para desempenhar uma determinada tarefa. Para um mantenedor do sistema, um cenário seria descrito como fazer uma alteração, por exemplo, atualizar o sistema operacional ou adicionar uma função específica. O cenário descrito por um desenvolvedor pode explicar como utilizar a arquitetura do sistema para sua construção ou mesmo prever seu desempenho. O cenário de um cliente pode descrever como o sistema será utilizado para um segundo produto em uma linha de produtos. (KAZMAN; KLEIN; CLEMENTS, 2000). Exemplo:

1. O usuário precisa integrar sistema de abertura de ticket por telefone com o produto de software
2. O usuário precisa utilizar o sistema em duas línguas diferentes e ser capaz de alternar entre elas
3. O usuário precisa ser capaz de fazer a atualização incremental do produto sem que haja perda de dados

As árvores de atributos de qualidade provêem um mecanismo para eficientemente traduzir os objetivos de negócio encontrados no segundo passo em um conjunto concreto de cenários de uso. As árvores ajudam a concretizar e priorizar estes requisitos.

Em sua definição original, o ATAM prevê que os atributos de qualidade que devem compor a raiz da árvore devem ser identificados durante os passos iniciais de sua aplicação, mas sem definir um conjunto que represente o conceito de qualidade. Entretanto, neste trabalho o conjunto de atributos utilizado são os atributos de qualidade interna, externa e em uso definidos norma ISO/NBR 9126-1, Qualidade de Produto, por se tratar de uma norma amplamente reconhecida no meio acadêmico e indústria.

O modelo da árvore de atributos de qualidade como é apresentado originalmente por Kazman, Klein e Clements (2000) como possuindo nos nós intermediários subfatores de qualidade específicos para o atributo no nível acima. Em Clements, Bergey e Mason (2005), estes subfatores são substituídos por “preocupações de atributos”, derivados dos estímulos e respostas específicos para cada atributo de qualidade que a arquitetura do produto deve lidar. A Tabela 3 apresenta o modelo de árvore de atributos de qualidade utilizado por Clemens (2005), incluindo o conceito de “preocupações de atributos”. Neste modelo, para cada atributo de qualidade uma sequencia de preocupações de atributos é listado na tabela. De maneira similar, para cada preocupação de atributo uma sequencia de cenários arquiteturais pode ser listada. Cada cenário é então classificado, neste exemplo, com pares de valores Alto, Médio e Baixo que representam primeiro a importância deste cenário para o sistema e em segundo a dificuldade da arquitetura em atingir este objetivo.

Tabela 3 - Exemplo de uma árvore de atributos de qualidade

Atributo de qualidade	I. Modularidade	
Preocupações de atributo	A. Substituir um componente arquitetural	
Cenários	*1. Substituir DBMS por uma nova versão durante a manutenção e conseguir esta substituição em uma pessoa/mês*	(M,B)
Preocupações de atributo	B. Baixo acoplamento/Alta coesão	
Preocupações de atributo	C. Implementável separadamente	
Cenários	*2. Um empreiteiro em um local implementa um serviço que utiliza outro serviço desenvolvido em um local diferente. O empreiteiro implementa o serviço sabendo apenas a definição da interface*	(A,A)

Fonte: Clements, Bergey e Mason (2005)

Baseado no modelo apresentado na tabela 3, este trabalho optou por criar um novo modelo para a árvore de atributos de qualidade (tabela 4). Neste modelo, a

prioridade do cenário não é representada, uma vez que isso é abordado no passo seguinte do método proposto. As decisões arquiteturais, chamadas de preocupações de atributo no modelo da tabela 3, são subdivididas em três elementos: Problema, ou a motivação que levou a implementação desta decisão arquitetural, Solução, que descreve a decisão encontrada e conseqüência, onde se pode descrever uma possível interação com outras decisões ou atributos de qualidade.

Tabela 4 - Modelo de árvore de atributos de qualidade

Atributo de qualidade	Funcionalidade
Acurácia	**Problema**: Excesso de validação em campos para manter a consistência de dados, travando a interface de usuário enquanto a informação não era obtida do banco de dados **Solução**: validação assíncrona, validando dados na tela enquanto o usuário pode preencher outros campos **Conseqüência**: abre muitas conexões com o servidor de banco
	Cenário: Usuário cria um novo registro em qualquer aplicação e preenche um campo com validação (ex: Data)
	Cenário: Usuário precisa fazer filtro por campos que não aparecem na aba de listagem

Para produzir a árvore de atributos de qualidade já com uma lista inicial de cenários arquiteturais, é necessário que ao menos a primeira fase do ATAM seja instanciada (passos 1 a 6). Entretanto, esta lista inicial de cenários representa apenas a visão de algumas poucas partes interessadas sobre o projeto avaliado.

Em sua apresentação original do ATAM, a segunda fase se inicia ao final do sexto passo. Esta fase inclui os três passos finais (7, 8 e 9), onde a equipe avaliadora é expandida para incluir o maior número possível de partes interessadas no projeto.

Durante o passo 7, a árvore de atributos de qualidade deve ser finalizada, com cada um dos diversos cenários arquiteturais encontrados ligados a uma decisão arquitetural, e estas decisões por sua vez ligadas a atributos de qualidade que os

impactam diretamente. Como produzir uma análise detalhada das abordagens arquiteturais não faz parte do objetivo principal, neste trabalho o ATAM é instanciado apenas dos passos 1 a 7, sendo os passos 8 e 9 opcionais.

3.3 Priorização dos cenários arquiteturais

A utilização dos dois critérios de classificação de cenários sugeridos em Kazman, Klein e Clements (2000) leva em consideração principalmente a relevância de cada cenário para o sucesso do sistema bem como a sua dificuldade de implementação sob a ótica da arquitetura avaliada.

Uma vez que este trabalho tem como objetivo a obtenção de um conjunto priorizado de testes, é necessário que os cenários arquiteturais sejam ordenados tendo como objetivo principal indicar o impacto que sua falha pode proporcionar sob a ótica das práticas de verificação e validação. Os critérios de priorização apresentados na definição original do ATAM não tem como objetivo representar o risco de falha de um requisito, tornando necessária a adoção de um conjunto diferente de critérios para a classificação destes cenários.

Uma única definição será apresentada a todos os avaliadores para uniformizar a percepção de falha de software. Neste sentido, será adotado inteiramente, sem personalizações, o esquema de integridade baseado em risco descrito no anexo B da norma IEEE 1012-2012, apresentado na seção 2.6 deste trabalho.

Cada nível de integridade resulta de uma combinação de sua possível consequência com um dos quatro valores para frequência: infrequente, ocasional, provável e razoável.

Todos os avaliadores envolvidos na elaboração da árvore de atributos de qualidade devem, uma vez que a árvore esteja completa, classificar cada um dos cenários levantados com um valor para frequência e outro valor para a consequência. Estes valores são convertidos em um número, de 1 a 4 de acordo com o mapa de níveis de integridade (tabela 2). A soma das notas de todos os avaliadores para cada um dos cenários deve resultar na priorização dos cenários arquiteturais.

3.4 Verificação da cobertura de teste em produtos finais

A atividade de verificação da cobertura de teste em produtos finais pode ser dividida em quatro passos:

1. Definição do produto final onde o conjunto priorizado de testes será executado
2. Escolha do critério de cobertura utilizado durante a verificação da cobertura
3. Comparar o conjunto de testes pré-existentes de acordo com os cenários arquiteturais e critério de cobertura
4. Especificar novos testes em áreas com cobertura deficiente

No primeiro passo, um dos projetos de desenvolvimento pertencentes à família de produtos avaliada nos passos anteriores deve ser escolhido. Qualquer projeto de desenvolvimento que inclua uma etapa de execução de testes de regressão é elegível para esta atividade de verificação da cobertura. Embora esta etapa precise ser refeita uma vez para cada novo projeto, entre iterações de um mesmo projeto não há necessidade de refazer toda a verificação, já que a arquitetura básica do núcleo não sofreu alterações.

No passo seguinte, deve-se escolher qual o critério de cobertura será utilizado como parâmetro durante a verificação do conjunto de casos de teste. A escolha do critério de cobertura é essencial para os passos subseqüentes desta atividade, uma vez que é de acordo com o critério escolhido que pode ser determinado se é necessário e em caso afirmativo, qual o critério de parada para a especificação de novos casos de teste.

Kazman, Klein e Clements (2000) definem os cenários arquiteturais durante a instanciação do ATAM como um mecanismo para a concretização dos requisitos não funcionais (objetivos de qualidade) do sistema avaliado. É a partir dos cenários obtidos durante a aplicação do método de avaliação arquitetural de onde deve ser derivado o conjunto de requisitos de teste e, por conseguinte o critério de cobertura.

Nos projetos avaliados por este trabalho, os artefatos de teste são em sua grande maioria especificações de testes funcionais ou de sistema, inicialmente manuais com um subconjunto podendo ser automatizado posteriormente, mas todos eles são classificados como teste de caixa preta. Ao determinar qual critério de

cobertura será utilizado, pode-se escolher apenas um ou uma combinação dos critérios de cobertura para testes de caixa preta apresentados na seção 2.5, com exceção do critério Grafos de causa e efeito, uma vez que o grande volume de artefatos de teste disponíveis torna impraticável a sua modelagem no escopo deste trabalho.

Portanto, o critério de cobertura utilizado nesta etapa foi o particionamento de classe de equivalência. Burnstein (2010) lista duas vantagens principais deste método:

1. Eliminar a necessidade de teste exaustivo, que é pouco factível
2. Guia o testador a selecionar um subconjunto de entradas com maior probabilidade de detectar defeitos

No terceiro passo, é feita verificação do conjunto existente de testes de regressão do projeto avaliado. Neste ponto, para cada cenário arquitetural devem ser identificados quais casos de teste, automáticos ou manuais, validam a abordagem arquitetural diretamente ligada ao cenário arquitetural, de acordo com a hierarquia apresentada pela árvore de atributos qualidade.

Finalmente no quarto passo, para aqueles cenários arquiteturais que foram identificados como possuidores de uma quantidade deficiente de casos de teste de acordo com o critério de qualidade escolhido no segundo passo, devem ter novos testes especificados até que o critério de cobertura seja satisfeito. Em casos onde durante o terceiro passo for determinado que o conjunto existente de testes já atenda a todos os cenários arquiteturais, o quarto passo torna-se opcional.

Ao final destes quatro passos, tem-se como resultado um conjunto de testes organizados de acordo com a priorização dos cenários arquiteturais obtidas durante a aplicação do ATAM. Um possível quinto passo é a execução deste conjunto de testes. Entretanto, não será abordado neste trabalho, uma vez que esta execução é dependente do processo de desenvolvimento instanciado no projeto avaliado, bem como do esforço disponível para teste no momento da execução.

3.5 Conclusão

Foi apresentada neste capítulo a proposta de uma técnica de priorização de testes em linhas de produtos tal que as decisões arquiteturais tomadas na construção do núcleo sejam utilizadas como fonte de requisitos de teste.

Para atingir esse objetivo, na primeira parte foram apresentados os aspectos mais relevantes da aplicação do ATAM sob a ótica do núcleo de uma linha de produtos de software, bem como os artefatos produzidos pelo método que são mais relevantes para a etapa de priorização de testes.

Na seção seguinte, é apresentada a formalização do conceito de criticidade de falha utilizado durante a etapa de priorização dos cenários arquiteturais, através da utilização do esquema de duas dimensões apresentado no apêndice A da norma IEEE 1012-2012.

Por ultimo, foi discutida a utilização dos cenários arquiteturais priorizados na definição de um critério para a avaliação da cobertura, bem como para a especificação de novos casos de teste em locais onde a cobertura for avaliada como deficiente.

Não faz parte da técnica proposta neste trabalho a definição de uma forma de utilizar o conjunto de testes priorizados, ou a definição de um plano de testes, uma vez que ambos só podem ser definidos a partir do processo de engenharia de software utilizando pela equipe avaliadora.

Os conceitos apresentados neste capítulo serão utilizados na realização do exemplo a ser descrito no capítulo 4.

4 APLICAÇÃO DO EXEMPLO PRÁTICO

Neste capítulo são apresentados os detalhes do desenvolvimento do exemplo prático onde a técnica proposta foi validada.

Este exemplo foi desenvolvido em uma linha de produtos para gestão de ativos de uma grande multinacional da indústria de tecnologia da informação, sendo que o desenvolvimento desta linha de produtos é distribuído geograficamente entre laboratórios localizados na Carolina do Norte e Texas, nos EUA, na Irlanda e em São Paulo.

4.1 Características do ambiente de desenvolvimento

Ao todo, cinco produtos foram avaliados no decorrer do desenvolvimento deste trabalho. O núcleo da linha de produtos de software, bem quatro produtos finais que estendem e modificam os componentes providos pelo núcleo.

O núcleo da linha de produtos consiste em um conjunto de serviços e interfaces, disponibilizados por um conjunto reduzido de aplicações e componentes, projetados para executar sobre um servidor de aplicação J2EE. A plataforma de execução do núcleo inclui no mínimo uma aplicação J2EE, um servidor de aplicação, um banco de dados e um navegador, sendo possível que todos estes coexistam em um mesmo ambiente ou em ambientes separados. O núcleo provê o meio para a extensão ou criação de novas aplicações e serviços. A figura 4 representa a divisão de componentes entre o núcleo e seus respectivos produtos finais.

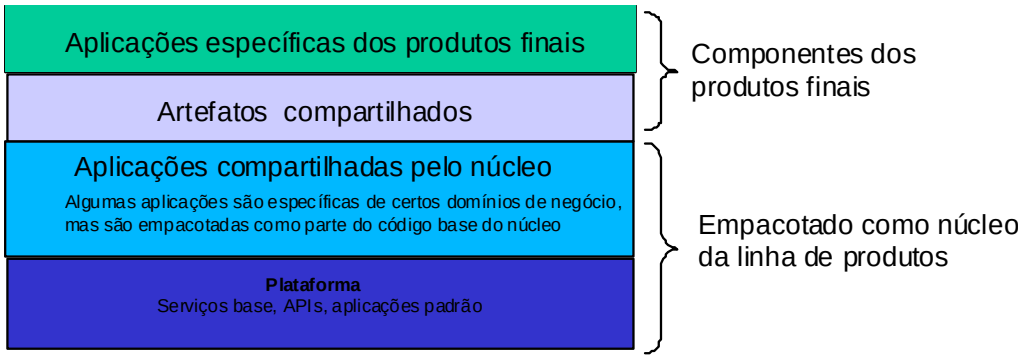

Figura 4 - Blocos de construção da linha de produtos

Na etapa de verificação da cobertura de teste, foram escolhidos quatro projetos diferentes de desenvolvimento de produtos finais. Para manter a confidencialidade dos dados envolvidos no desenvolvimento deste trabalho, os produtos finais envolvidos serão identificados como Projeto 1, Projeto 2, Projeto 3 e Projeto 4.

Toda a equipe de desenvolvimento dos produtos finais, tanto programadores quanto analistas de qualidade, se encontra dividida entre diversas localizações geográficas, se concentrando em três centros de desenvolvimento: Raleigh, Carolina do Norte e Austin, Texas, ambas nos EUA, São Paulo e Cork, Irlanda, com a gerência destes projetos se dividindo entre duas localidades nos Estados Unidos.

Todo o desenvolvimento do núcleo da linha de produtos se concentra em uma única localidade nos Estados Unidos. Entretanto, a garantia da qualidade do núcleo adota a estratégia descrita por Tevanlinna, Taina e Kauppinen (2004) como divisão de responsabilidades. Esta equipe executa a bateria de teste de unidade e um conjunto reduzido de testes automatizados de integração nas aplicações providas pelo núcleo. A execução dos demais testes, tais como integração mais aprofundada e de sistema, é de responsabilidade dos projetos de desenvolvimento dos produtos finais, sendo que cada projeto garante a qualidade das respectivas aplicações estendidas do núcleo.

4.2 Aplicação da técnica proposta

4.2.1. ATAM

Equipe avaliadora

O passo inicial para a aplicação do ATAM na linha de produtos escolhida foi a definição da equipe avaliadora. Na definição da equipe avaliadora, por se tratar de software de prateleira, não foi possível obter representantes dos usuários. Foram escolhidos avaliadores provenientes de diversos projetos envolvidos no desenvolvimento de produtos, onde desempenham papéis variados. Ao todo, a equipe avaliadora inicial continha 6 participantes:

- Dois analistas de qualidade, que desempenham o papel de líder de teste de seus respectivos projetos
- Um líder de suporte ao cliente

- Dois desenvolvedores seniores, que desempenham o papel de líder de desenvolvimento em seus respectivos projetos
- Um arquiteto de teste, responsável pela estratégia de teste em vários projetos da linha de produtos

Ao longo da aplicação do exemplo prático deste trabalho, a equipe avaliadora foi alterada apenas durante a aplicação do passo 7 do ATAM, quando a equipe foi ampliada para incluir mais partes interessadas, conforme recomendado em Kazman, Klein e Clements (2000).

ATAM: Passo 1

Neste passo inicial, foi apresentado à equipe avaliadora o método ATAM, conforme definido em Kazman, Klein e Clements (2000). Foram necessárias duas reuniões de aproximadamente uma hora cada:

1ª reunião:

Duração: 1h

Tempo de preparação do material: 1h 30min

Material utilizado: Apresentação em *PowerPoint* introduzindo o ATAM.

Detalhes: Esta reunião correspondeu ao 1º passo do ATAM, onde uma descrição geral do método foi apresentada ao time. Ao longo da apresentação, o grupo demonstrou desconhecer a definição de qualidade de software utilizada neste trabalho (NBR ISO/IEC 9126-1).

2ª reunião:

Duração: 1h

Tempo de preparação do material: 1h

Material utilizado: Apresentação em PowerPoint introduzindo a norma ISO-9126. Sistema de conferência remota, software para compartilhar apresentação remotamente.

Detalhes: Embora não prevista no processo, o desconhecimento do time diante da definição formal de qualidade utilizada neste trabalho mostrou que era necessário introduzir ao time a norma de qualidade. O conceito foi bem recebido, gerando comentários sobre outras situações do processo de desenvolvimento onde esta definição poderia ser aplicada.

ATAM: Passos 2 e 3

Apenas uma reunião foi necessária para cobrir os passos 2 e 3 do ATAM:

3ª reunião:

Duração: 1h 30min

Tempo de preparação do material: 2h, tempo equivalente ao total gasto com pesquisa nos diversos repositórios de documentação e comunicação com time responsável nos EUA

Material utilizado: Documentação arquitetural do produto: Um documento extensivo contendo detalhes finos da arquitetura do produto, e duas apresentações contendo uma visão geral da arquitetura.

Detalhes: A arquitetura foi apresentada utilizando-se a documentação pré-existente. Nenhum dos integrantes conhecia diretamente a documentação utilizada. Durante uma sessão que durou metade da reunião, o time chegou a um consenso do que seria um conjunto inicial de objetivos de negócio, que desempenham o papel de requisitos de alto nível, ao responder a pergunta: “Por que alguém compraria o seu software?”.

Na segunda metade da reunião, a documentação arquitetural encontrada foi apresentada ao time. Na discussão que se seguiu, os desenvolvedores seniores clarificaram alguns pontos da arquitetura ao restante do time. A lista de objetivos de negócio elaborados nesta reunião pode ser vista abaixo:

1. O Núcleo deve permitir controlar e manter todo o ciclo de vida (Aquisição, operação, manutenção e decomissionamento) de um ativo (maquinário, subsistema, equipamento...)
2. Deve permitir a integração e troca de informação com outros sistemas externos (financeiro, RH, aquisição)
3. Gerenciamento de requisição de serviços, incidentes
4. Permitir a configuração de segurança do produto através da aplicação, definição de grupos de usuários e restrição de visualização.
5. Proporcionar capacidade de auditoria de transações e assinatura eletrônica
6. Permitir mudanças em seu comportamento de forma a adequar as regras de negócio do cliente

ATAM: Passos 4, 5 e 6

Esta reunião tinha como objetivo estender a discussão da arquitetura (passo 3 do ATAM) e iniciar a elicitação de abordagens arquiteturais (passo 4).

4ª reunião:

Duração: 1h

Tempo de preparação do material: Não houve. O material utilizado já estava disponível.

Material utilizado: Documentação arquitetural do produto, lista de objetivos de negócio gerada na reunião anterior e definição dos atributos de qualidade da norma ISO-9126

Detalhes: Logo no início da reunião, ficou evidente que sem um ponto de partida guiando as discussões, o modelo de *brainstorm* conforme proposto em Kazman, Klein e Clements (2000) não funcionaria para esta equipe avaliadora levantar as abordagens arquiteturais. O moderador optou então por utilizar os atributos de qualidade da norma como ponto de partida, enquanto cada atributo era questionado, levando a equipe avaliadora a identificar abordagens que tivessem impacto direto naquele atributo. Cada abordagem foi levantada contendo 3 partes: um problema que motivou esta decisão na arquitetura, a descrição da abordagem e algum possível efeito colateral ou conseqüência. Não foi possível concluir todos os atributos de qualidade nesta reunião, sendo finalizados na reunião seguinte.

5ª reunião:

Duração: 1h

Tempo de preparação do material: 30min (organização do material obtido da reunião anterior)

Material utilizado: Documentação arquitetural do produto, lista de objetivos de negócio, definição dos atributos de qualidade da norma ISO-9126, lista inicial de abordagens arquiteturais gerada na reunião anterior

Detalhes: Esta reunião terminou de percorrer todos os atributos de qualidade, levantando as respectivas abordagens arquiteturais relevantes. Continuou a utilizar a abordagem da reunião anterior, onde para cada atributo de qualidade as abordagens eram levantadas utilizando-se a definição Problema (motivação) /Solução/Conseqüência

Nestas duas reuniões não foram seguidos seqüencialmente os passos 4, 5 e 6 conforme a definição original do ATAM. Ao invés disso, optou-se por em cada reunião expandir uma parte da árvore de abordagens arquiteturais, dando ao time um objetivo claro a ser atingido gradualmente, um atributo de qualidade por vez. A estratégia adotada consistia em já iniciar a elaboração da arvore de atributos de qualidade (passo 5), ao mesmo tempo que estas abordagens identificadas (passo 4), e então analisadas (passo 6). Os resultados obtidos são equivalentes aos passos 4, 5 e 6, embora não completos, deixando para a próxima reunião a identificação dos cenários arquiteturais.

ATAM: Passo 7

6ª reunião:

Duração: 1h 30m

Tempo de preparação do material: 10 min (organização do material gerado na reunião anterior)

Material utilizado: definição dos atributos de qualidade da norma ISO-9126, árvore de atributos de qualidade da reunião anterior

Detalhes: O objetivo desta reunião foi iniciar a ultima etapa de confecção da árvore de atributos de qualidade, levantando os cenários arquiteturais. Em concordância com a fase 2 do ATAM, o grupo foi expandido para incluir mais dois analistas de qualidade e um desenvolvedor, provenientes de diferentes projetos finais da mesma linha de produtos. Foram identificados cenários para 7 dos 13 atributos de qualidade que tiveram decisões arquiteturais encontradas. Mais uma vez optou-se por direcionar a identificação dos cenários arquiteturais percorrendo as abordagens uma a uma, ao invés da técnica de *brainstorm* sugerida na definição original do ATAM.

7ª reunião:

Duração: 1h

Tempo de preparação do material: 10 min (organização do material gerado na reunião anterior)

Material utilizado: definição dos atributos de qualidade da norma ISO-9126, árvore de atributos de qualidade expandida na reunião anterior,

Detalhes: Esta reunião terminou os últimos passos na criação da árvore de atributos de qualidade. Foram levantados os cenários arquiteturais para os últimos oito atributos.

A tabela 5 apresenta os resultados obtidos nas reuniões 4 a 7. A árvore de atributos de qualidade é dividida entre seis atributos de qualidade. Em cada seção, são listados os sub-atributos que tiveram abordagens arquiteturais identificadas. Dos 27 sub- atributos de qualidade externa e interna descritos na norma NBR ISO/IEC 9126, foram encontradas decisões arquiteturais para 13. Estas abordagens podem ser identificadas pelo conjunto Problema/Solução/Conseqüência. Para cada abordagem, são listados um a um os cenários relacionados com a abordagem imediatamente acima.

Tabela 5 - Árvore de atributos de qualidade

Atributo de qualidade	Funcionalidade
Acurácia	**Problema**: Excesso de validação em campos para manter a consistência de dados, travando a interface de usuário enquanto a informação não era obtida do banco de dados **Solução**: validação assíncrona, validando dados na tela enquanto o usuário pode preencher outros campos **Conseqüência**: abre muitas conexões com o servidor de banco
	Cenário: Usuário cria um novo registro em qualquer aplicação e preenche um campo com validação (ex: Data)
	Cenário: Usuário precisa fazer filtro por campos que não aparecem na aba de listagem
Interoperabilidade	**Problema**: comunicar e integrar o sistema com diversos produtos complementares durante um fluxo de negócio **Solução:** framework de integração via XML, síncrona e assíncrona e integração via REST (*Representational State Transfer*). **Conseqüência**: Impacto direto no desempenho, tanto entrada quanto saída
	Cenário: Usuário precisa integrar sistema de abertura de ticket por telefone com o produto
	Cenário: Usuário integra dados com fonte ERP Oracle ou SAP e precisa enviar para o produto
	Cenário: Usuário precisa migrar dados em planilhas ou XML para o produto

Atributo de qualidade	Confiabilidade
Maturidade	**Problema**: Garantir a consistência dos dados durante utilização simultânea por vários usuários **Solução**: Utilização em paralelo: Travamento otimista dos dados, prevalecendo o usuário que salvar primeiro a atualização do registro **Conseqüência**: Usuário só é informado do conflito ao tentar salvar o registro. As alterações feitas por este usuário são perdidas.
	Cenário: Uma *escalation*/*Cron Task* executa com intervalos curtos, alterando o estado de registros atualizados por usuários
	Cenário: Usuários simultâneos interagindo no mesmo registro
Recuperabilidade	**Problema**: Recuperar de erros de sistema, interrupções do serviço **Solução**: Soluções de tolerância a falhas são delegadas para o servidor de aplicação e ao banco de dados. **Conseqüência:** Aumento de complexidade e tempo de resolução no suporte ao cliente
	Cenário: Usuário entra com dados no produto, quando uma ocorre uma interrupção repentina na conexão com o banco de dados
Atributo de qualidade	Usabilidade
Inteligibilidade	**Problema**: Universalização da utilização em diversas organizações em várias partes do mundo **Solução**: Produto é traduzido em mais de 20 idiomas. **Conseqüência**: Aumento da complexidade de manutenção
	Cenário: Usuário precisa usar o produto em dois idiomas diferentes. Ex: inglês e francês
	Cenário: Usuário pesquisa por dados em um ambiente com dois idiomas instalados
Operacionalidade	**Problema**: Prover uma forma fácil e consistente de recuperar um registro e preencher um campo com este registro **Solução**: Interface padronizada para todos os produtos da linha de produtos de software **Conseqüência**: Uso excessivo do mouse, solução inadequada para preenchimento rápido
	Cenário: Usuário digita texto em um campo associado a uma lista
Atributo de qualidade	Eficiência
Comportamento em relação ao tempo	**Problema**: Garantir um tempo de resposta aceitável **Solução**: Tamanho configurável da página de resultados para

	consultas ao banco de dados **Conseqüência**: Requer a utilização de relatórios para obter dados completos
	Cenário: Usuário carrega todos os registros em uma aplicação
	Cenário: Ao entrar no produto, os *portlets* na página inicial são carregados com os dados existentes no banco
Atributo de qualidade	Manutenibilidade
Analisabilidade	**Problema**: Diagnosticar falhas e suas causas **Solução**: Nível de detalhe do log configurável (INFO, DEBUG, WARN, ERROR, FATAL) **Conseqüência**: O desempenho diminui conforme o nível de detalhe do log aumenta
	Cenário: Avaliação e recuperação de falhas durante personalização e configuração do produto
Modificabilidade	**Problema**: Atualizar o sistema e entregar ao cliente alterações na estrutura do banco ou nos meta-dados do produto **Solução**: Alterações no banco de dados organizadas em arquivos XML com regra de nomenclatura identificando o nível de atualização **Conseqüência**: Aumento do acoplamento entre aplicação e banco de dados
	Cenário: Atualização incremental do produto (entrega de novos requisitos) e correção de erros
	Problema: prover a coexistência entre componentes **Solução**: Atualização da hierarquia de classes através da manipulação do bytecode **Conseqüência**: Rastreabilidade de defeitos é prejudicada, aumento da complexidade da interação entre componentes
	Cenário: Desenvolvimento em paralelo de alterações em mesma aplicação por times diferentes
	Cenário: Coexistência entre vários produtos finais que atualizam um mesmo componente
	Cenário: Entregar correções de um produto específico em um ambiente onde vários produtos finais coexistem
Testabilidade	**Problema**: Verificar múltiplas traduções da interface **Solução**: Simula ambiente localizado utilizando um conjunto de arquivos que define os rótulos dos campos e mensagens **Conseqüência**: -
	Cenário: Instalação do produto utilizando traduções falsas e simular a execução do produto em línguas diferentes da utilizada no ambiente de desenvolvimento
Atributo de qualidade	Portabilidade
Adaptabilidade	**Problema**: Adequar o produto a diferentes processos de negócio de diversos clientes **Solução**: A camada de interface de usuário é construída a partir de um formato próprio de XML, configurável pelo usuário **Conseqüência**: aumento da complexidade e imprevisibilidade durante a manutenção
	Cenário: Uma mesma aplicação pode ser personalizada para diferentes situações e clientes
	Problema: Adequar o produto a diferentes processos de

	diversos clientes **Solução**: Fluxo de dados e ciclo de vida de objetos pode ser personalizado através da aplicação *workflow* **Conseqüência**: Não há realimentação para o usuário sobre o que acontece com os dados
	Cenário: Parametrização do processo de aprovação, designação e ciclo de vida de trabalho
Atributo de qualidade	Portabilidade
Capacidade para ser instalado	**Problema**: Prover a entrega e instalação de um produto WEB **Solução**: Instalador executável automatiza a atualização/instalação no banco de dados e a disponibilização no servidor de aplicação **Conseqüência**: -
	Cenário: O usuário instala o produto e seus componentes são disponibilizados automaticamente no servidor de aplicação
Capacidade para substituir	**Problema**: Migrar dados para dentro e pra fora do produto **Solução**: Camada de integração para importação exportação de dados em formato XML **Conseqüência**:
	Cenário: Migração de objetos de produtos concorrentes para a última versão do produto
	Cenário: Propagação (sincronização) de dados entre produtos concorrentes
	Cenário: Povoamento de dados para execução de testes que requerem grande massa de dados

4.2.2. Priorização dos cenários arquiteturais

Uma vez produzidos os cenários arquiteturais e a árvore de qualidade, a lista de cenários foi compilada em um único modelo de documento, apresentado individualmente a diversos membros da equipe avaliadora responsável pela instanciação do ATAM. Uma ultima reunião foi realizada para apresentar aos avaliadores o conceito de nível de integridade utilizado neste trabalho.

8ª reunião:

Duração: 1h

Tempo de preparação do material: 10 min (compilação dos cenários levantados na árvore arquitetural em um formulário)

Material utilizado: Norma IEEE-1012/2012, árvore de atributos de qualidade, lista de cenários arquiteturais

Detalhes: Nesta reunião foi apresentado aos avaliadores o esquema de criticidade presente no anexo B da norma IEEE-1012/2012. Cada membro atribuiu um valor de

um a quatro para a freqüência e a criticidade de um possível erro para cada um dos cenários arquiteturais, de acordo o anexo B da norma IEEE 1012-2012 e anotaram seus resultados em um formulário (Apêndice A).

Após os participantes preencherem seus respectivos formulários de classificação, um valor médio foi calculado para a conseqüência e a freqüência, para que então cada cenário fosse rotulado conforme o mapa de atribuição de níveis de integridade. O modelo de formulário enviado aos participantes, as respostas individuais, bem como os valores médios podem ser vistos no apêndice A deste trabalho.

A classificação resultante dos cenários arquiteturais pode ser visto na tabela 6, ordenados de acordo com o nível de integridade. Este nível de integridade foi em seguida utilizado como critério para ordenar e priorizar os cenários arquiteturais.

Tabela 6 - Classificação final dos cenários arquiteturais

Cenário	**Nível de integridade**
1. Uma *escalation*/*Cron Task* executa com intervalos curtos, alterando o estado de registros atualizados por usuários	4
2. Teste de coexistência entre vários produtos finais que atualizam um mesmo componente	4
3. Entregar correções de um componente de um produto específico em um ambiente onde vários produtos coexistem	4
4. Usuário integra dados com fonte ERP Oracle ou SAP e precisa enviar para o produto	3
5. Usuários simultâneos interagindo no mesmo registro	3
6. Usuário entra com dados no produto, quando uma ocorre uma interrupção repentina na conexão com o banco de dados	3
7. Usuário precisa usar o Produto em duas línguas diferentes. Ex: inglês e francês	3
8. Usuário carrega todos os registros em uma aplicação	3
9. Ao entrar no produto, os *portlets* na página inicial são carregados com os dados existentes no banco	3
10. Atualização incremental do produto (entrega de novos requisitos) e correção de erros	3
11. Desenvolvimento em paralelo de uma mesma aplicação por times diferentes	3
12. Propagação (sincronização) de dados entre produtos concorrentes	3
13. Usuário cria um novo registro em qualquer aplicação e preenche um campo com validação (ex: Data)	2
14. Usuário precisa integrar sistema de abertura de ticket por telefone com o produto	2
15. Usuário precisa migrar dados em planilhas Excel/XML para o produto	2
16. Usuário pesquisa por dados em um ambiente com duas línguas instaladas	2

Cenário	Nível de integridade
17. Usuário digita texto em um campo associado a uma lista	2
18. Avaliação e recuperação de falhas durante personalização e configuração do produto	2
19. Instalação do produto utilizando traduções falsas ou *mock testing* e simular a execução do produto em línguas diferentes da utilizada no ambiente de desenvolvimento	2
20. Uma mesma aplicação pode ser personalizada para diferentes situações e clientes	2
21. Parametrização do processo de aprovação, designação e ciclo de vida de trabalho	2
22. O usuário instala o produto e seus componentes são disponibilizados automaticamente no servidor de aplicação	2
23. Migração de objetos de produtos concorrentes para a ultima versão do Produto	2
24. Povoamento de dados para execução de testes que requerem grande massa de dados	2
25. Usuário precisa fazer filtro por campos que não aparecem na aba de listagem	1

4.2.3. Verificação da cobertura de teste

Partindo da lista priorizada de cenários arquiteturais obtidas na etapa anterior, o próximo passo foi escolher um critério de cobertura para testes de caixa preta. Em seguida, utilizando o critério escolhido, foi desenvolvido um conjunto de casos de teste que foi apresentado em um formulário de cobertura de teste (apêndice C). Este formulário foi enviado aos líderes de teste de cada um dos quatro projetos de desenvolvimento de produtos finais escolhidos.

A estratégia de teste adotada na linha de produtos avaliada era a divisão de responsabilidades e não fazia parte do escopo deste trabalho propor modificações estratégicas na linha de produtos. Portanto, os testes elaborados nesta etapa de verificação da cobertura deveriam ter como objetivo garantir o funcionamento da instancia da arquitetura nos produtos finais, deixando a cargo do projeto de desenvolvimento responsável pelo núcleo da linha de produtos o seu teste exaustivo. Portanto, o critério de cobertura utilizado nesta etapa foi o particionamento de classe de equivalência.

A aplicação deste critério de cobertura para testes de caixa preta resultou em um total de 27 casos de teste, sendo numerados primeiramente de acordo com o cenário arquitetural que o originou e em seguida seqüencialmente. A lista de casos

de teste pode ser vista na tabela 7 abaixo, juntamente com as respectivas prioridades herdadas do nível de integridade dos cenários.

Tabela 7 - Casos de teste resultantes da aplicação do critério de cobertura

Casos de teste	Prioridade
Caso de teste 1.1: Criar um novo objeto *escalation*/*Cron Task* de forma a alterar o estado de um registro enquanto é atualizado por um usuário	4
Caso de teste 2.1: Instalar um novo produto que atualize um objeto já utilizado por um produto previamente instalado	4
Caso de teste 3.1: Executar a instalação sucessiva de correções provenientes de produtos diferentes, atualizando sempre um mesmo objeto	4
Caso de teste 4.1: Importar dados provenientes de ERP ou SAP formatados em XML que pode ser interpretado pelo framework de integração do máximo	3
Caso de teste 4.2: Importar dados provenientes de ERP ou SAP formatados em XML que não pode ser interpretado pelo framework de integração do máximo	3
Caso de teste 5.1: Alterar simultaneamente em dois clientes ao mesmo tempo um mesmo registro	3
Caso de teste 6.1: Durante a operação do Produto, interromper a conexão com o banco de dados por um tempo especificado X, tal que seja menor que o limite configurável	3
Caso de teste 6.2: Durante a operação do Produto, interromper a conexão com o banco de dados por um tempo especificado X, tal que seja maior que o limite configurável	3
Caso de teste 7.1: Usuário altera um mesmo registro em dois momentos distintos, utilizando primeiramente a língua base do sistema e em seguida a língua adicional	3
Caso de teste 8.1: Em uma aplicação que possui um grande número de registros (10.000 entradas ou mais), verificar o tempo de resposta ao listar todos os registros na aba de listagem para um determinado tamanho de conjunto de resultados	3
Caso de teste 9.1: Em uma aplicação que possui um grande número de registros (10.000 entradas ou mais), exercitar diferentes combinações de *portlets* diferentes e conjuntos de resultados	3
Caso de teste 10.1: Instalação incremental de uma nova versão do produto em um ambiente onde já tenha uma versão anterior instalada	3
Caso de teste 10.2: Tentar instalar uma versão anterior a versão instalada no ambiente de teste	3
Caso de teste 12.1: Importação de dados XML em formato válido pelo MIF	3
Caso de teste 12.2: Importação de dados XML em formato inválido pelo MIF	3
Caso de teste 12.3: Exportação de dados XML em formato válido pelo MIF	3
Caso de teste 13.1: Preencher consecutivamente dois campos com validação (ex: Data) em um tempo menor que o necessário para a validação do primeiro campo preenchido	2

Casos de teste	Prioridade
Caso de teste 12.2: Importação de dados XML em formato inválido pelo MIF	3
Caso de teste 15.1: A partir de um conjunto de dados em planilha Excel, executar importação bem sucedida no Produto	2
Caso de teste 17.1: Em um campo relacionado a uma lista de objetos, digitar um valor válido	2
Caso de teste 17.2: Em um campo relacionado a uma lista de objetos, digitar um valor inválido	2
Caso de teste 18.1: Executar uma personalização do produto, tal como alteração na estrutura de tabela no banco de dados e verificar se estas mudanças são indicadas no log	2
Caso de teste 19.1: Executar o mesmo teste funcional em ambiente *mock* e posteriormente em ambiente com a língua base do *mock*, verificando se o comportamento do produto se mantém o mesmo	2
Caso de teste 20.1: Fazer uma personalização de instalação do máximo, adicionando novos campos na interface de usuário e novas estruturas no banco de dados	2
Caso de teste 21.1: Criar um novo *workflow* para um determinado objeto Produto e exercitar a sua execução	2
Caso de teste 22.1: Executar a instalação de um produto via MADT com diferentes estruturas de servidor de aplicação, por exemplo, o servidor de aplicação e banco de dados localizados em ambientes separados da estação cliente.	2
Caso de teste 25.1: Filtrar uma massa de dados de um determinado objeto por um ou mais campos que não estejam presentes na aba de listagem	1

Os passos intermediários da aplicação do critério de cobertura, tais como as condições externas, classes de equivalência válidas e inválidas podem ser vistas em detalhes no APÊNDICE B – Projeto de casos de teste.

Cada projeto de desenvolvimento de produtos finais avaliado possuía uma metodologia e um conjunto de ferramentas diferentes para armazenar seus artefatos de teste. Optou-se então por transformar o conjunto obtido na etapa anterior (tabela 7) em um formulário de cobertura (anexo C). Para cada item era questionado se este já era coberto por algum outro teste ou atividade de desenvolvimento pré-existente.

O formulário foi então respondido pelos líderes de teste de cada projeto, uma vez que o líder é responsável por distribuir atividades de verificação e validação para todo o seu time, tendo uma visão geral sobre todo o conjunto de artefatos de seu projeto. As respostas, assim como o nível de cobertura resultante podem ser vistos na tabela 8 abaixo.

Tabela 8 - Avaliação de cobertura de teste dos produtos finais

		Coberto por teste existente?			
Caso de teste	Prioridade	Proj 1	Proj 2	Proj 3	Proj 4
1.1	4	Sim	Sim	Sim	Não
2.1	4	Não	Sim	Sim	Não
3.1	4	Sim	Não	Sim	Sim
4.1	3	Não	Não	Não	Não
4.2	3	Não	Não	Não	Não
5.1	3	Não	Não	Sim	Não
6.1	3	Sim	Não	Não	Não
6.2	3	Sim	Não	Não	Não
7.1	3	Não	Sim	Sim	Sim
8.1	3	Sim	Não	Sim	Não
9.1	3	Sim	Não	Sim	Não
10.1	3	Sim	Sim	Sim	Sim
10.2	3	Não	Não	Sim	Não
12.1	3	Não	Sim	Sim	Não
12.2	3	Não	Sim	Sim	Não
12.3	3	Não	Sim	Sim	Não
13.1	2	Não	Não	Sim	Sim
14.1	2	Sim	Não	Não	Não
15.1	2	Não	Não	Não	Não
17.1	2	Sim	Não	Sim	Sim
17.2	2	Sim	Não	Sim	Não
18.1	2	Não	Não	Não	Não
19.1	2	Sim	Sim	Não	Não
20.1	2	Não	Não	Não	Sim
21.1	2	Não	Não	Sim	Não
22.1	2	Sim	Não	Sim	Sim
25.1	1	Sim	Não	Sim	Não
Cobertura total (%)		48,15	29,63	66,67	25,93

Proj - Projeto

4.3 Análise dos resultados

Na técnica proposta neste trabalho, enquanto os passos 1 e 2, ou instanciação do ATAM e priorização de cenários devem necessariamente ser desenvolvidos no núcleo da linha de produtos, no terceiro passo qualquer projeto da linha de produtos pode ser avaliado, incluindo-se projetos de desenvolvimento de produtos finais ou mesmo novas versões do núcleo.

4.3.1. ATAM

Em Kazman, Klein e Clements (2000) propõem que o ATAM seja executado durante dois dias inteiros, sendo o primeiro dia dedicado inteiramente a execução da primeira fase e o segundo dia finalizando a instanciação do método executando a segunda fase. Recomendam também que a equipe seja formada desde cinco pessoas a até quarenta ou cinqüenta.

Logo no início da instanciação do ATAM neste trabalho ficou evidente a impossibilidade de se mobilizar durante dois dias inteiros até mesmo cinco ou seis partes interessadas que apresentassem o perfil necessário, uma vez que isso causaria um impacto indesejável no progresso dos respectivos projetos de desenvolvimento. A solução foi utilizada foi fracionar a execução do método em reuniões que variavam entre uma e duas horas de duração, com cada passo do ATAM sendo executado em uma ou mais reuniões ao longo de um ano.

O tempo total gasto na instanciação dos passos um a sete foi de 8,5 horas para as reuniões com a equipe avaliadora e 5 horas na preparação dos materiais utilizados em cada uma das reuniões. Além disso foram considerados os seguintes fatores:

- Foram instanciados apenas os passos da primeira fase e o início da segunda (passos um a sete). O objetivo principal deste trabalho era a produção dos cenários arquiteturais, tornando desnecessários os passos oito e nove.
- A equipe avaliadora era composta pelo número mínimo de participantes recomendado em Kazman, Klein e Clements (2000), o que justifica o menor tempo gasto nos passos seis e sete.

- Foi necessário adotar e apresentar ao grupo um padrão que formalizasse o conceito de qualidade de software, etapa que não estava prevista na definição do ATAM.
- Não foi possível encontrar para compor a equipe avaliadora uma parte interessada que se envolvesse ativamente na formulação dos requisitos do produto

Estes resultados desviaram pouco do que é esperado pela definição original do ATAM.

Os produtos finais nesta linha têm por contrato um tempo de vida no mercado de aproximadamente cinco anos apenas para uma mesma versão. Pode-se concluir então que o investimento total de horas nesta avaliação pode ser considerado baixo, uma vez que este resultado é aplicável à versão atual de todos os produtos finais desta mesma linha.

4.3.2. Priorização dos cenários arquiteturais

Após a priorização dos cenários arquiteturais pela equipe avaliadora, do total de cenários (tabela 7) temos que:

- Três foram classificados com nível de integridade 4
- Nove foram classificados com nível de integridade 3
- Doze foram classificados com nível de integridade 2
- Um cenário foi classificado com nível de integridade 1

Uma fração significativa dos cenários se concentrou nos níveis de integridade 2 ou 3. Uma análise dos valores médios para a conseqüência e freqüência destes vinte e um cenários obteve 2,28 e 2,5 para a conseqüência e a freqüência respectivamente, o que arredondando para valores inteiros equivalem a 2 e 3.

Os valores encontrados indicam que na maioria dos cenários levantados durante a aplicação dos passos seis e sete do ATAM, a equipe avaliadora julgou que a conseqüência de uma falha hipotética é marginal, o que de acordo com a norma IEEE 1012-2012 significa lesão ou doença grave, degradação da missão secundária ou alguma perda financeira. Já a freqüência ou probabilidade de ocorrência de uma falha, foi considerada provável.

Embora os resultados obtidos pela priorização dos cenários arquiteturais sejam utilizados preferencialmente na priorização de testes de sistema neste trabalho, os

dados obtidos também podem ser utilizados em outras áreas, como o planejamento de estórias de usuário, priorização das baterias de teste automatizado ou testes de unidade durante a integração contínua.

4.3.3. Verificação da cobertura de teste

Os resultados da cobertura total indicados na tabela 8 mostram que dos quatro projetos de desenvolvimento de produtos finais avaliados, três apresentaram um índice de cobertura de testes sobre a arquitetura do núcleo abaixo de 50%, ou seja, dos testes especificados durante a instanciação do critério de cobertura escolhido neste trabalho, menos da metade já era previamente executado por algum outro teste ou atividade do projeto avaliado.

Em algumas situações, os líderes de teste responsáveis pela avaliação da cobertura argumentaram que a execução de certos casos de teste era de responsabilidade apenas ao núcleo da linha de produtos. Entretanto, como afirmado por SEI (2011), cada produto final possui a sua própria arquitetura, uma instancia da arquitetura do núcleo onde são exercitados os mecanismos de variação, justificando assim a execução destes testes também no escopo dos produtos finais.

Quando analisados conjuntamente as respostas de todos os projetos, pode ser visto que quatro casos de teste não são cobertos por nenhum teste em nenhum projeto. Trata-se dos casos de teste 4.1, 4.2, 15.1 e 18.1.

Utilizando a árvore de atributos de qualidade é possível listar qual atributo cada caso de teste verifica. Ao somar o peso de todos os cenários e comparar com a quantidade de casos de teste cobertos em cada projeto (tabela 9), é possível notar algumas discrepâncias na cobertura.

Tabela 9 - Comparação da cobertura total de atributos de qualidade por projeto

			Total de testes cobertos			
	Peso total dos cenários	Número de testes	Proj 1	Proj 2	Proj 3	Proj 4
Manutenibilidade	18	6	3	3	4	3
Portabilidade	14	6	2	3	6	1
Confiabilidade	13	4	3	1	2	0
Funcionalidade	12	5	1	0	1	1
Usabilidade	9	4	2	1	3	2
Eficiência	6	2	2	0	2	0

Dentre as discrepâncias na cobertura de teste encontradas por este resultado, podemos destacar:

- No projeto 3 o segundo atributo, portabilidade, apesar de ter um peso menor, tem mais testes especificados do que o primeiro atributo, manutenibilidade.
- Raciocínio semelhante se aplica ao projeto 1, com o atributo Confiabilidade possuindo três testes, enquanto portabilidade possui apenas dois.

A cobertura de testes dos projetos avaliados não corresponde ao peso dos cenários encontrados pela equipe avaliadora durante o segundo passo da aplicação deste exemplo prático. Isto pode indicar que a prioridade para a especificação de testes nos projetos avaliados é influenciada por outros fatores que não são percebidos diretamente pela equipe de desenvolvimento ou usuários finais, tais decisões de mercado.

4.4 Conclusão

Neste capítulo foram apresentados os resultados obtidos durante o desenvolvimento do exemplo prático do método proposto no capítulo 3. Inicialmente, foi necessário contextualizar onde este exemplo seria desenvolvido, descrevendo a linha de produtos e o ambiente de desenvolvimento na primeira seção deste capítulo.

Na segunda seção, foram apresentados integralmente os dados resultantes da aplicação da técnica proposta neste trabalho, tais como modelos utilizados, adaptações feitas durante a instanciação do ATAM, modelos de documentos elaborados e atas das reuniões. Esta seção é dividida em três subseções, uma para cada passo da técnica proposta.

Na terceira sessão são discutidos os resultados listados na seção anterior, bem como suas as consequências. Assim como a seção anterior, esta seção é dividida em três subseções correspondendo aos passos da técnica proposta.

5 CONCLUSÃO

Linhas de produtos de software trazem como principal beneficio a reutilização de componentes. A partir da necessidade de verificação das variações resultantes da instanciação da arquitetura do núcleo no desenvolvimento dos produtos finais, este trabalho mostrou os resultados de um exemplo prático que teve como objetivo validar a possibilidade de utilizar o ATAM para levantar os requisitos não funcionais do núcleo e assim priorizar a verificação destes requisitos nos produtos finais em um ambiente onde o processo de desenvolvimento não previa este levantamento.

Em uma revisão da literatura disponível para estratégias de teste em linhas de produtos de software, não foram encontradas técnicas que avaliassem a instanciação da arquitetura do núcleo nos produtos finais. A partir dos cenários arquiteturais obtidos durante a instanciação dos passos um a sete do ATAM, a técnica proposta neste trabalho sugeriu um critério de priorização baseado na norma IEEE-1012-2012, para em seguida utilizar estes cenários priorizados na elaboração de um conjunto de casos de teste da arquitetura, com o fim de verificar a cobertura nos produtos finais da linha avaliada.

O exemplo prático desenvolvido neste trabalho mostra que é possível atingir o objetivo de verificar a cobertura, bem como propor uma priorização de testes da arquitetura para futuras versões dos produtos finais, sem que seja necessário modificar o processo de desenvolvimento de software já estabelecido no ambiente avaliado, onde não é prevista a elicitação de requisitos não funcionais.

Uma limitação da técnica proposta neste trabalho é o impacto que o perfil da equipe avaliadora pode exercer sobre o resultado. Uma equipe avaliadora formada por indivíduos com pouco conhecimento ou poder de decisão no desenvolvimento da linha de produtos pode implicar na produção de resultados menos significativos, enquanto a inclusão na equipe avaliadora de membros essenciais tais como arquitetos ou representantes dos usuários podem ser mais difíceis tendo em vista o tempo utilizado pela instanciação do ATAM.

5.1 Contribuições

Diferentes técnicas para dividir tarefas entre componentes e reutilizar artefatos de teste têm sido sugeridas para linhas de produtos de software. A técnica proposta neste trabalho se aplica a ambientes de desenvolvimento onde o processo utilizado não prevê o levantamento de requisitos não funcionais da arquitetura do núcleo, bem como sua priorização durante a verificação nos produtos finais.

O desenvolvimento do exemplo prático neste trabalho permitiu concretizar na forma de atributos de qualidade e decisões arquiteturais as características e metas de qualidade do núcleo e definir com sucesso um critério para a sua verificação nos produtos finais.

Na empresa envolvida no desenvolvimento deste exemplo prático, os resultados obtidos na priorização dos cenários arquiteturais e na verificação da cobertura de teste permitiram o aperfeiçoamento do plano de testes que será utilizado nas próximas versões de manutenção dos produtos finais avaliados, levando a uma melhor compreensão e cobertura dos requisitos de qualidade da arquitetura da linha de produtos. Os resultados obtidos com a instanciação do ATAM serão utilizados também na criação e priorização de estórias de usuário com a finalidade de implementar melhorias que impactem positivamente nos atributos de qualidade

5.2 Trabalhos futuros

A verificação da cobertura de testes desenvolvida no exemplo prático deste trabalho permitiu apontar deficiências nas atividades de execução de testes manuais de sistema. Trabalhos futuros podem ser feitos para avaliar a utilização da técnica proposta em outras atividades de testes do modelo-V, tais como unitário, módulo e integração.

Uma característica que precisa ser estudada é o impacto que a formação da equipe de avaliação da arquitetura, durante a instanciação do ATAM, exerce sobre os resultados obtidos durante a etapa de verificação da cobertura de teste.

Outra proposta para trabalhos futuros consiste em determinar a influência da utilização de diferentes critérios de cobertura de teste na efetividade da verificação da cobertura nos produtos finais.

Por ultimo, ainda que o esforço investido no desenvolvimento do exemplo prático deste trabalho seja relativamente baixo quando comparado à vida útil da versão corrente de toda a linha de produtos de software, uma sugestão de trabalho futuro é a quantificação da melhoria na qualidade, através da aplicação da técnica proposta durante todo o ciclo de desenvolvimento de uma nova versão de um produto final e a aferição de métricas para os atributos de qualidade.

REFERÊNCIAS

AMMANN, P. OFFUTT, J. **Introduction to Software Testing**, 1. ed. New York: Cambridge University Press, 2008. 344p.

BURNSTEIN, I. **Practical Software Testing: A Process-Oriented Approach**, 1. Ed. Springer Publishing Company, 2010. 714p.

BASS, L. CLEMENTS, P. KAZMAN, R. **Software Architecture in Practice**, 2. Ed. Addison-Wesley, 2003. 560p.

BERTOLINO, A. MARRÉ, M. Reducing and estimating the cost of test coverage criteria. In: **Proceedings of the 18th international conference on software engineering**, 1996, Berlin, Germany.

CLEMENTS, P., BERGEY, J., MASON, D. Using the SEI Architecture Tradeoff Analysis Method to Evaluate WIN-T: A Case Study. **Technical Note CMU/SEI-2005-TN-027,** Software Engineering Institute, 2005. Disponível em: <http://www.sei.cmu.edu/library/abstracts/reports/05tn027.cfm>. Acesso em: 04 fev 2012.

COHEN, M. B., DWYER, M. B., Shi, J. Coverage and adequacy in software product line testing. In: **Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis**, 2006, New York, US.

DESHARNAIS, J. M. ABRAN, A. SURYN, W. Identification and analysis of attributes and base measures within ISO 9126. **Software Quality Journal**, Hingham, p.447-460, jun. 2011.

GANESAN, J. K. D., KOLB, R., HAURY, G. M. U., Comparing Costs and Benefits of Different Test Strategies for a Software Product Line: A Study from Testo AG. In: **Proceedings of the 11th International Software Product Line Conference**, 2007, Kyoto, Japan.

GAO, J. ESPINOZA, R. JINGSHA HE, Testing coverage analysis for software component validation. In: **Proceedings of the 29th Annual International Computer Software and Applications Conference**, 2005, Edinburgh, UK.

HASHIM, N.L., RAMAKRISHNAN, S., SCHMIDT, H.W., Architectural Test Coverage for Component-Based Integration Testing. In: **Proceedings of the 7th International Conference on Quality Software**, 2007, Portland, US.

IEEE STD 610.12. IEEE Standard Glossary of Software Engineering Terminology, **Institute of Electrical and Electronics Engineers**, 1990

IEEE STD 1012. IEEE Standard for System and Software Verification and Validation, **Institute of Electrical and Electronics Engineers**, 2012

JEZEQUEL, J. M. LE TRAON, Y. NEBUT, C. **System Testing of Product Lines: From Requirements to Test Cases**. In: Käkölä, T. Dueñas, J. C. **Software Product Lines: Research issues in Engineering and Management**. Secaucus: Springer-Verlag, 2006. p. 447-478

KAZMAN, R., KLEIN, M., & CLEMENTS, P. ATAM method for architecture evaluation. **Technical Report CMU/SEI-2000-TR-004**, Software Engineering Institute, 2000. Disponível em: <http://www.sei.cmu.edu/library/abstracts/reports/00tr004.cfm>. Acesso em: 04 fev 2012.

MUCCINI, H., INVERARDI, P., BERTOLINO, A. Using software architecture for code testing. **IEEE Transactions on Software Engineering**, Los Alamitos, V. 30, n. 3, p.160-171, março 2004.

MYERS, G. J. SANDLER, C. BADGETT, T. **The Art of Software Testing**, 3. Ed. Wiley Publishing, 2011. 240p.

NBR ISO/IEC 9126-1. “Engenharia de software – Qualidade de produto – Modelo de qualidade”, **Associação Brasileira de Normas Técnicas**, 2003

NETO, P. A. M. S. et al. A Regression Testing Approach for Software Product Lines Architectures. In: **Proceedings of the Fourth Brazilian Symposium on Software Components, Architectures and Reuse**, 2010, Bahia, Brazil.

NETO, P. A. M. S**. A Regression Testing Approach for Software Product Lines Architectures.** Recife, 2011a. 182 f. Dissertação (Mestrado) - Centro de informática, Universidade Federal de Pernambuco, Pernambuco, 2011.

NETO, P. A. M. S. et al. Testing Software Product Lines. **IEEE Software**, Los Alamitos, p.16-20, september/october. 2011b.

NORTHTROP, L. SEI's Software Product Line Tenets. **IEEE Software**, Los Alamitos, p.32-40, July/August. 2002.

OBJECT MANAGEMENT GROUP. **Software & Systems Process Engineering Meta-Model Specification, Version 2.0**. Abril, 2012.
Diponível em: < http://www.omg.org/spec/SPEM/2.0/>. Acesso em: 03 ago. 2012.

SOFTWARE ENGINEERING INSTITUTE. **Framework for Software Product Line Practice, Version 5.0**. Pittsburgh: Carnegie Mellon University, 2011.
Diponível em: < http://www.sei.cmu.edu/productlines/frame_report/index.html >.
Acesso em: 04 fev. 2012.

TEVANLINNA, A. TAINA, J. KAUPPINEN, R., Product Family Testing – a Survey. **ACM SIGSOFT Software Engineering Notes**, New York, v.29, n.2, Mar. 2004

APÊNDICE A – Classificação de cenários em níveis de integridade

Tabela 10 - Formulário de classificação de nível de integridade em cenários arquiteturais

Conseqüência	Freqüência	Cenário
		Usuário cria um novo registro em qualquer aplicação e preenche um campo com validação (ex: Data)
		Usuário precisa fazer filtro por campos que não aparecem na aba de listagem
		Usuário precisa integrar sistema de abertura de ticket por telefone com o produto
		Usuário integra dados com fonte ERP Oracle ou SAP e precisa enviar para o produto
		Usuário precisa migrar dados em planilhas Excel/XML para o produto
		Uma *escalation*/*Cron Task* executa com intervalos curtos, alterando o estado de registros atualizados por usuários
		Usuários simultâneos interagindo no mesmo registro
		Usuário entra com dados no produto, quando uma ocorre uma interrupção repentina na conexão com o banco de dados
		Usuário precisa usar o Produto em duas línguas diferentes. Ex: inglês e francês
		Usuário pesquisa por dados em um ambiente com duas línguas instaladas
		Usuário digita texto em um campo associado a uma lista
		Usuário carrega todos os registros em uma aplicação
		Ao entrar no produto, os *portlets* na página inicial são carregados com os dados existentes no banco
		Avaliação e recuperação de falhas durante personalização e configuração do produto
		Atualização incremental do produto (entrega de novos requisitos) e correção de erros
		Desenvolvimento em paralelo de uma mesma aplicação por times diferentes
		Teste de coexistência entre vários produtos finais que atualizam um mesmo componente
		Entregar correções de um componente de um produto específico em um ambiente onde vários produtos coexistem
		Instalação do produto utilizando traduções falsas ou *mock testing* e simular a execução do produto em línguas diferentes da utilizada no ambiente de desenvolvimento
		Uma mesma aplicação pode ser personalizada para diferentes situações e clientes
		Parametrização do processo de aprovação, designação e ciclo de vida de trabalho
		O usuário instala o produto e seus componentes são disponibilizados automaticamente no servidor de aplicação
		Migração de objetos de produtos concorrentes para a ultima versão do Produto
		Propagação (sincronização) de dados entre produtos concorrentes
		Povoamento de dados para execução de testes que requerem grande massa de dados

Tabela 11 - Resultados da classificação de cenários em níveis de integridade pela equipe avaliadora

	Arq. Teste		Desenv.		Eng. Teste		Lider Suporte		Arq. Desen.		Média		
	cons.	freq.	cons.	freq.	cons.	freq.	cons.	freq.	cons.	freq.	cons.	freq.	Nivel de Int.
Usuário cria um novo registro em qualquer aplicação e preenche um campo com validação (ex: Data)	2	3	2	1	2	4	3	1	1	1	2	2	2
Usuário precisa fazer filtro por campos que não aparecem na aba de listagem	4	1	2	2	2	2	3	1	1	1	2	1	1
Usuário precisa integrar sistema de abertura de ticket por telefone com o produto	2	4	3	3	1	1	2	2	2	2	2	2	2
Usuário integra dados com fonte ERP Oracle ou SAP e precisa enviar para o produto	2	4	3	3	3	3	3	2	2	2	3	3	3
Usuário precisa migrar dados em planilhas Excel/XML para o produto	2	4	2	1	3	3	2	2	1	1	2	2	2
Uma *escalation*/*Cron Task* executa com intervalos curtos, alterando o estado de registros atualizados por usuários	2	4	3	4	3	3	2	3	3	3	3	3	4
Usuários simultâneos interagindo no mesmo registro	1	4	3	2	3	4	1	3	3	3	2	3	3
Usuário entra com dados no produto, quando uma ocorre uma interrupção repentina na conexão com o banco de dados	1	4	2	1	4	4	2	1	3	3	2	3	3
Usuário precisa usar o Produto em duas línguas diferentes. Ex: inglês e francês	2	4	3	4	2	2	3	4	2	2	2	3	3
Usuário pesquisa por dados em um ambiente com duas línguas instaladas	2	4	2	2	2	2	2	2	1	1	2	2	2
Usuário digita texto em um campo associado a uma lista	3	3	2	1	1	4	1	4	1	1	2	3	2
Usuário carrega todos os registros em uma aplicação	1	4	3	3	3	4	3	2	1	1	2	3	3
Ao entrar no produto, os *portlets* na página inicial são carregados com os dados existentes no banco	3	3	1	2	3	3	2	4	1	1	2	3	3
Avaliação e recuperação de falhas durante personalização e configuração do produto	2	3	3	3	3	3	2	2	1	1	2	2	2

	Arq. Teste		Desenv.		Eng. Teste		Lider Suporte		Arq. Desen.		Média		
	cons.	freq.	cons.	freq.	cons.	freq.	cons.	freq.	cons.	freq.	cons.	freq.	Nivel de Int.
Atualização incremental do produto (entrega de novos requisitos) e correção de erros	4	3	3	3	4	2	2	2	2	1	3	2	3
Desenvolvimento em paralelo de uma mesma aplicação por times diferentes	3	4	2	3	4	3	3	1	1	2	3	3	3
Teste de coexistência entre vários produtos finais que atualizam um mesmo componente	4	4	3	3	3	4	2	3	2	2	3	3	4
Entregar correções de um componente de um produto específico em um ambiente onde vários produtos coexistem	4	4	3	3	3	4	3	3	2	2	3	3	4
Instalação do produto utilizando traduções falsas ou *mock testing* e simular a execução do produto em línguas diferentes da utilizada no ambiente de desenvolvimento	2	3	2	2	2	2	2	1	1	1	2	2	2
Uma mesma aplicação pode ser personalizada para diferentes situações e clientes	3	2	3	2	3	3	2	4	1	1	2	2	2
Parametrização do processo de aprovação, designação e ciclo de vida de trabalho	2	4	2	2	2	3	2	2	1	1	2	2	2
O usuário instala o produto e seus componentes são disponibilizados automaticamente no servidor de aplicação	2	4	2	2	3	3	2	1	1	1	2	2	2
Migração de objetos de produtos concorrentes para a ultima versão do Produto	2	3	2	2	4	3	3	2	1	1	2	2	2
Propagação (sincronização) de dados entre produtos concorrentes	2	3	3	2	4	3	3	2	2	2	3	2	3
Povoamento de dados para execução de testes que requerem grande massa de dados	3	3	2	2	1	2	2	1	2	2	2	2	2

APÊNDICE B – Projeto de casos de teste

Cenário 1: Uma *escalation*/*Cron Task* executa com intervalos curtos, alterando o estado de registros atualizados por usuários

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Um novo objeto *escalation*/*Cron Task*	Qualquer objeto do produto que seja atualizado por usuário	

Caso de teste 1.1: Criar um novo objeto *escalation*/*Cron Task* de forma a alterar o estado de um registro enquanto é atualizado por um usuário

Cenário 2: Coexistência entre vários produtos finais que atualizam um mesmo componente

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Instalação de um novo produto	O produto instalado atualiza um mesmo objeto do núcleo que outro produto já previamente instalado	

Caso de teste 2.1: Instalar um novo produto que atualize um objeto já utilizado por um produto previamente instalado

Cenário 3: Entregar correções de um componente de um produto específico em um ambiente onde vários produtos coexistem

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Entrega de correções	Várias correções em um mesmo objeto	
Produtos corrigidos	Uma correção em um objeto utilizado por vários produtos	

Caso de teste 3.1: Executar a instalação sucessiva de correções provenientes de produtos diferentes, atualizando sempre um mesmo objeto

Cenário 4: Usuário integra dados com fonte ERP Oracle ou SAP e precisa enviar para o produto

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Dados provenientes de uma fonte externa (ERP ou SAP)	Dados formatados em XML que pode ser interpretado pelo framework de integração do produto	Dados formatados em XML que não pode ser interpretado pelo framework de integração do produto

Caso de teste 4.1: Importar dados provenientes de ERP ou SAP formatados em XML que pode ser interpretado pelo framework de integração do produto

Caso de teste 4.2: Importar dados provenientes de ERP ou SAP formatados em XML que não pode ser interpretado pelo framework de integração do produto

Cenário 5: Usuários simultâneos interagindo no mesmo registro

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Número de usuários	Dois ou mais	

Caso de teste 5.1: Alterar simultaneamente em dois clientes ao mesmo tempo um mesmo registro

Cenário 6: Usuário entra com dados no produto, quando uma ocorre uma interrupção repentina na conexão com o banco de dados

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Duração da interrupção da conexão	Interrupção menor que o limite configurável (Time Out)	Interrupção maior que o limite configurável (Time Out)

Caso de teste 6.1: Durante a operação do produto, interromper a conexão com o banco de dados por um tempo especificado X, tal que seja menor que o limite configurável
Caso de teste 6.2: Durante a operação do produto, interromper a conexão com o banco de dados por um tempo especificado X, tal que seja maior que o limite configurável

Cenário 7: Usuário precisa usar o produto em duas línguas diferentes. Ex: inglês e francês

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Língua base	Qualquer língua suportada pelo produto	
Língua adicional	Qualquer língua suportada pelo produto	

Caso de teste 7.1: Usuário altera um mesmo registro em dois momentos distintos, utilizando primeiramente a língua base do sistema e em seguida a língua adicional

Cenário 8: Usuário carrega todos os registros em uma aplicação

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Tamanho do conjunto de resultados	De 1 ao tamanho da tabela no banco	

Caso de teste 8.1: Em uma aplicação que possui um grande número de registros (10.000 entradas ou mais), verificar o tempo de resposta ao listar todos os registros na aba de listagem para um determinado tamanho de conjunto de resultados

Cenário 9: Ao entrar no produto, os *portlets* na página inicial são carregados com os dados existentes no banco

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Número de *portlets*	Não existe limite estabelecido	
Conjunto de resultados dos *portlets*	De 1 ao tamanho da tabela no banco	

Caso de teste 9.1: Em uma aplicação que possui um grande número de registros (10.000 entradas ou mais), exercitar diferentes combinações de *portlets* diferentes e conjuntos de resultados

Cenário 10: Atualização incremental do produto (entrega de novos requisitos) e correção de erros

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Novas versões de produtos	Instalação de versões posteriores a instalada	Instalação de versões anteriores a instalada

Caso de teste 10.1: Instalação incremental de uma nova versão do produto em um ambiente onde já tenha uma versão anterior instalada

Caso de teste 10.2: Tentar instalar uma versão anterior a versão instalada no ambiente de teste

Cenário 11: Desenvolvimento em paralelo de uma mesma aplicação por times diferentes

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Alterações em código fonte de uma mesma aplicação	0 ou mais times alterando a mesma classe	

Caso de teste 11.1: coberto pelo caso de teste 3.1

Cenário 12: Propagação (sincronização) de dados entre produtos concorrentes

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Importação de dados MIF	Dados no formato XML apropriados	Dados em formato inválido
Exportação de dados MIF	Dados no formato XML apropriados	Dados em formato inválido

Caso de teste 12.1: Importação de dados XML em formato válido pelo MIF
Caso de teste 12.2: Importação de dados XML em formato inválido pelo MIF
Caso de teste 12.3: Exportação de dados XML em formato válido pelo MIF

Cenário 13: Usuário cria um novo registro em qualquer aplicação e preenche um campo com validação (ex: Data)

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Campo com validação	Preenchimento de 2 ou mais campos com validação em curto período de tempo	

Caso de teste 13.1: Preencher consecutivamente dois campos com validação (ex: Data) em um tempo menor que o necessário para a validação do primeiro campo preenchido

Cenário 14: Usuário precisa integrar sistema de abertura de ticket por telefone com o produto

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Chamadas por telefone	Abertura de tickets	

Caso de teste 14.1: Utilizando-se das ferramentas de integração do máximo, executar a abertura de um ticket via telefone

Cenário 15: Usuário precisa migrar dados em planilhas Excel/XML para o Produto

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Importação de dados de planilha para o Produto	Conjunto de dados em planilha Excel	

Caso de teste 15.1: A partir de um conjunto de dados em planilha Excel, executar importação bem sucedida no Produto

Cenário 16: Usuário pesquisa por dados em um ambiente com duas línguas instaladas

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Línguas instaladas	Conjunto de línguas suportadas pelo Produto	
Dados pesquisados	Qualquer objeto Produto	

Caso de teste 16.1: Coberto pelo caso de teste 7.1

Cenário 17: Usuário digita texto em um campo associado a uma lista

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Lista de objetos de um campo	Conjunto de objetos presentes na tabela relacionada a este campo	Qualquer texto ou dado que não esteja na tabela relacionada a este campo

Caso de teste 17.1: Em um campo relacionado a uma lista de objetos, digitar um valor válido
Caso de teste 17.2: Em um campo relacionado a uma lista de objetos, digitar um valor inválido

Cenário 18: Avaliação e recuperação de falhas durante personalização e configuração do produto

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Nível de log do produto	INFO, DEBUG, WARN, ERROR, FATAL	

Caso de teste 18.1: Executar uma personalização do produto, tal como alteração na estrutura de tabela no banco de dados e verificar se estas mudanças são indicadas no log

Cenário 19: Instalação do produto utilizando traduções falsas ou *mock testing* e simular a execução do produto em línguas diferentes da utilizada no ambiente de desenvolvimento

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Língua base da tradução falsa	Todos os formatos suportados de língua *mock*	

Caso de teste 19.1: Executar o mesmo teste funcional em ambiente *mock* e posteriormente em ambiente com a língua base do *mock*, verificando se o comportamento do produto se mantém o mesmo

Cenário 20: Uma mesma aplicação pode ser personalizada para diferentes situações e clientes

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Arquivo “presentation.xml” que define interface de aplicação do produto	Qualquer campo suportado pelo framework de personalização do PRODUTO	
Estrutura de tabelas no banco de dados relacionadas a aplicação do produto	Qualquer estrutura de tabela que seja suportada pelo banco de dados do PRODUTO	

Caso de teste 20.1: Fazer uma personalização de instalação do máximo, adicionando novos campos na interface de usuário e novas estruturas no banco de dados

Cenário 21: Parametrização do processo de aprovação, designação e ciclo de vida de trabalho

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
workflow, ciclo de vida de um determinado objeto Produto	Fluxo de ciclo de vida especificado na aplicação *Workflow Designer*	

Caso de teste 21.1: Criar um novo *workflow* para um determinado objeto Produto e exercitar a sua execução

Cenário 22: O usuário instala o produto e seus componentes são disponibilizados automaticamente no servidor de aplicação

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Estrutura da arquitetura de servidores de aplicação	Estruturas de servidor de aplicação suportadas pelo *Middleware* utilizado	

Caso de teste 22.1: Executar a instalação de um produto via MADT com diferentes estruturas de servidor de aplicação, por exemplo, o servidor de aplicação e banco de dados localizados em ambientes separados da estação cliente.

Cenário 23: Migração de objetos de produtos concorrentes para a última versão do Produto

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Produtos concorrentes	Dados exportados por produtos concorrentes	

Caso de teste 23.1: Cobertos pelos casos de teste 12.1, 12.2 e 12.3

Cenário 24: Povoamento de dados para execução de testes que requerem grande massa de dados

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Camada de integração	Massa de dados	

Caso de teste 24.1: Coberto pelo caso de teste 8.1

Cenário 25: Usuário precisa fazer filtro por campos que não aparecem na aba de listagem

Condição externa	Classes de equivalência válidas	Classes de equivalência inválidas
Campos de um objeto Produto	Campos que não estão presentes na aba de listagem deste objeto	

Caso de teste 25.1: Filtrar uma massa de dados de um determinado objeto por um ou mais campos que não estejam presentes na aba de listagem

APÊNDICE C – Formulário de cobertura de testes

Casos de teste	Prioridade
Caso de teste 1.1: Criar um novo objeto *escalation*/*Cron Task* de forma a alterar o estado de um registro enquanto é atualizado por um usuário	4
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 2.1: Instalar um novo produto que atualize um objeto já utilizado por um produto previamente instalado	4
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 3.1: Executar a instalação sucessiva de correções provenientes de produtos diferentes, atualizando sempre um mesmo objeto	4
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 4.1: Importar dados provenientes de ERP ou SAP formatados em XML que pode ser interpretado pelo framework de integração do máximo	3
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 4.2: Importar dados provenientes de ERP ou SAP formatados em XML que não pode ser interpretado pelo framework de integração do máximo	3
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 5.1: Alterar simultaneamente em dois clientes ao mesmo tempo um mesmo registro	3
É coberto por um caso de teste ou atividade? Qual?	

Caso de teste 6.1: Durante a operação do Maximo, interromper a conexão com o banco de dados por um tempo especificado X, tal que seja menor que o limite configurável	3
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 6.2: Durante a operação do Maximo, interromper a conexão com o banco de dados por um tempo especificado X, tal que seja maior que o limite configurável	3
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 7.1: Usuário altera um mesmo registro em dois momentos distintos, utilizando primeiramente a língua base do sistema e em seguida a língua adicional	3
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 8.1: Em uma aplicação que possui um grande número de registros (10.000 entradas ou mais), verificar o tempo de resposta ao listar todos os registros na aba de listagem para um determinado tamanho de conjunto de resultados	3
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 9.1: Em uma aplicação que possui um grande número de registros (10.000 entradas ou mais), exercitar diferentes combinações de *portlets* diferentes e conjuntos de resultados	3
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 10.1: Instalação incremental de uma nova versão do produto em um ambiente onde já tenha uma versão anterior instalada	3
É coberto por um caso de teste ou atividade? Qual?	

Caso de teste 10.2: Tentar instalar uma versão anterior a versão instalada no ambiente de teste	3
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 12.1: Importação de dados XML em formato válido pelo MIF	3
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 12.2: Importação de dados XML em formato inválido pelo MIF	3
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 12.3: Exportação de dados XML em formato válido pelo MIF	3
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 13.1: Preencher consecutivamente dois campos com validação (ex: Data) em um tempo menor que o necessário para a validação do primeiro campo preenchido	2
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 14.1: Utilizando-se das ferramentas de integração do máximo, executar a abertura de um ticket via telefone	2
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 15.1: A partir de um conjunto de dados em planilha Excel, executar importação bem sucedida no Maximo	2
É coberto por um caso de teste ou atividade? Qual?	

Caso de teste 17.1: Em um campo relacionado a uma lista de objetos, digitar um valor válido	2
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 17.2: Em um campo relacionado a uma lista de objetos, digitar um valor inválido	2
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 18.1: Executar uma personalização do produto, tal como alteração na estrutura de tabela no banco de dados e verificar se estas mudanças são indicadas no log	2
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 19.1: Executar o mesmo teste funcional em ambiente *mock* e posteriormente em ambiente com a língua base do *mock*, verificando se o comportamento do produto se mantém o mesmo	2
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 20.1: Fazer uma personalização de instalação do máximo, adicionando novos campos na interface de usuário e novas estruturas no banco de dados	2
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 21.1: Criar um novo *workflow* para um determinado objeto Maximo e exercitar a sua execução	2
É coberto por um caso de teste ou atividade? Qual?	

Caso de teste 22.1: Executar a instalação de um produto via MADT com diferentes estruturas de servidor de aplicação, por exemplo, o servidor de aplicação e banco de dados localizados em ambientes separados da estação cliente.	2
É coberto por um caso de teste ou atividade? Qual?	
Caso de teste 25.1: Filtrar uma massa de dados de um determinado objeto por um ou mais campos que não estejam presentes na aba de listagem	1
É coberto por um caso de teste ou atividade? Qual?	