engenharia de software magazine

Requisitos
Desenvolvimento de software dirigido por casos de uso – Parte 2

Planejamento
Conheça abordagens e modelos que apóiam a gerência de riscos

DevMedia group Ano I - Edição 03

Melhoria de Processos de Software com o uso de Análise Causal de Defeitos

Planejamento
Plano de Projeto: Um 'Mapa' Essencial à Gestão de Projetos de Software

Requisitos
Entenda o que são requisitos não funcionais e como eles podem impactar a arquitetura de seu sistema

Projeto
Saiba como identificar e especificar componentes de negócio usando como base casos de uso e diagramas UML

Metodologias Ágeis
A importância dos testes automatizados

Verificação, Validação & Teste
Ferramentas Open Source e melhores práticas na gestão de testes.

ISSN 1983127-7

Aulas desta edição:

- Introdução ao MS Project - Parte 01
- Introdução ao MS Project - Parte 02
- Introdução à Engenharia de Requisitos - Parte 07
- Introdução à Engenharia de Requisitos - Parte 08
- Introdução à Engenharia de Requisitos - Parte 09
- Coleta e análise de métricas com Metrics for Eclipse
- Teste Unitário com JUnit
- Teste de Cobertura com Eclemma
- Teste Funcional com Selenium-IDE

engenharia de software magazine

Ano 1 - 3ª Edição 2008 - - Impresso no Brasil

Corpo Editorial

Colaboradores
Rodrigo Oliveira Spínola
rodrigo@sqlmagazine.com.br

Marco Antônio Pereira Araújo
Eduardo Oliveira Spínola

Editor de Arte
Vinicius O. Andrade
viniciusoandrade@gmail.com

Diagramação
Compublix – Cia. de Publicações Especiais

Revisão
Gregory Monteiro
gregory@clubedelphi.net

Na Web
www.devmedia.com.br/esmag

Apoio

Parceiros:

Rodrigo Oliveira Spínola
rodrigo@sqlmagazine.com.br
Doutorando em Engenharia de Sistemas e Computação (COPPE/UFRJ). Mestre em Engenharia de Software (COPPE/UFRJ, 2004). Bacharel em Ciências da Computação (UNIFACS, 2001). Colaborador da Kali Software (www.kalisoftware.com), tendo ministrado cursos na área de Qualidade de Produtos e Processos de Software, Requisitos e Desenvolvimento Orientado a Objetos. Consultor para implementação do MPS. BR. Atua como Gerente de Projeto e Analista de Requisitos em projetos de consultoria na COPPE/UFRJ. É Colaborador da Engenharia de Software Magazine.

Marco Antônio Pereira Araújo
(maraujo@devmedia.com.br)
É Doutorando e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ – Linha de Pesquisa em Engenharia de Software, Especialista em Métodos Estatísticos Computacionais e Bacharel em Matemática com Habilitação em Informática pela UFJF, Professor dos Cursos de Bacharelado em Sistemas de Informação do Centro de Ensino Superior de Juiz de Fora e da Faculdade Metodista Granbery, Analista de Sistemas da Prefeitura de Juiz de Fora. É editor da Engenharia de Software Magazine.

Eduardo Oliveira Spínola
(eduspinola@gmail.com)
É Editor das revistas Engenharia de Software Magazine, SQL Magazine, WebMobile. É bacharel em Ciências da Computação pela Universidade Salvador (UNIFACS) onde atualmente cursa o mestrado em Sistemas e Computação na linha de Engenharia de Software, sendo membro do GESA (Grupo de Engenharia de Software e Aplicações).

EDITORIAL

"Análise causal é considerada na maioria das abordagens de melhoria de processo de software, tais como MPS, CMMI, Six Sigma, e Lean. A análise causal de defeitos visa identificar as causas dos defeitos para que ações possam ser tomadas para prevenir sua ocorrência em iterações ou projetos futuros. Embora a identificação das causas seja realizada em projetos específicos, estas causas podem revelar oportunidades de melhoria dos ativos de processo organizacionais.

Uma das principais características de qualidade do produto de software são os defeitos encontrados em seus artefatos. Desta forma, a análise causal de defeitos fornece uma oportunidade eficiente de melhoria de processos de software baseada em produto."

Neste contexto, a Engenharia de Software Magazine destaca nesta edição uma matéria cujo propósito é fornecer uma visão geral de como a análise causal de defeitos pode ser aplicada para obter melhoria de processo baseada no produto. A base teórica a respeito de análise causal de defeitos é descrita, bem como maneiras eficientes de implementar a mesma em organizações de software.

Gostaríamos de destacar também uma matéria sobre a importância dos testes automatizados. Este artigo se inspira na filosofia dos Métodos Ágeis de Desenvolvimento de Software e em práticas recomendadas pela Programação eXtrema (XP), em especial nos Testes Automatizados, que é uma técnica voltada principalmente para a melhoria da qualidade dos sistemas de software.

Além destas duas matérias, esta terceira edição traz mais seis artigos:

- Plano de Projeto: Um 'Mapa' Essencial à Gestão de Projetos de Software;
- Requisitos Não Funcionais: Critérios para análise arquitetural;
- Modelos e Abordagens para Gerenciamento de Riscos de Projetos de Software;
- Gestão de testes: Ferramentas Open Source e melhores práticas na gestão de testes;
- Identificação e especificação de componentes de negócio;
- Desenvolvimento de Software Dirigido por Caso de Uso Parte II: Especificando caso de uso;

Desejamos uma ótima leitura!

Equipe Editorial Engenharia de Software Magazine

Caro Leitor,
Para esta terceira edição, temos um conjunto de 9 vídeo aulas totalizando mais de 2 horas de conteúdo. Estas vídeo aulas estão disponíveis para download no Portal da Engenharia de Software Magazine e certamente trarão uma significativa contribuição para seu aprendizado. Os assuntos discutidos são: engenharia de requisitos, planejamento e gerência de projetos e, orientação a objetos. A lista de aulas publicadas pode ser vista abaixo:

01 – Engenharia de Requisitos
Título: Introdução à Engenharia de Requisitos - Parte 07
Mini-Resumo: Esta vídeo aula é parte de um curso de introdução à engenharia de requisitos. Nesta sétima parte os requisitos serão categorizados em requisitos funcionais, não funcionais e de domínio. Complementando, aborda-se inicialmente os requisitos funcionais.

02 – Engenharia de Requisitos
Título: Introdução à Engenharia de Requisitos - Parte 08
Mini-Resumo: Esta vídeo aula é parte de um curso de introdução à engenharia de requisitos. Nesta oitava parte são discutidos conceitos relacionados aos requisitos não funcionais.

03 – Engenharia de Requisitos
Título: Introdução à Engenharia de Requisitos - Parte 09
Mini-Resumo: Esta vídeo aula é parte de um curso de introdução à engenharia de requisitos. Nesta nona parte o assunto requisito não funcional é finalizado, destacando os tipos mais comuns de requisitos não funcionais.

04 – Planejamento e Gerência de Projetos
Título: Introdução ao MS Project - Parte 01
Mini-Resumo: Esta vídeo aula apresenta as funcionalidades básicas do MS Project. São discutidos conceitos introdutórios sobre a elaboração de cronogramas.

05 – Planejamento e Gerência de Projetos
Título: Introdução ao MS Project - Parte 02
Mini-Resumo: Esta vídeo aula apresenta as funcionalidades básicas do MS Project. São discutidos conceitos introdutórios sobre a elaboração de cronogramas.

06 – Métricas
Título: Coleta e análise de métricas com Metrics for Eclipse
Mini-Resumo: Esta vídeo aula apresenta a coleta e análise de métricas como complexidade ciclomática, complexidade em software orientado a objetos e linhas de código, bem como a análise destas medições no sentido de auxiliar na qualidade de aplicações desenvolvidas em Java utilizando o plug-in Metrics for Eclipse.

07 – Testes
Título: Teste Unitário com JUnit
Mini-Resumo: Esta vídeo aula apresenta o planejamento e a implementação de teste unitário utilizando a ferramenta JUnit, apoiada por métricas de complexidade ciclomática e análise do valor limite.

08 – Testes
Título: Teste de Cobertura com Eclemma
Mini-Resumo: Esta vídeo aula apresenta a utilização de teste de cobertura em conjunto com testes unitários no sentido de garantir a abrangência dos casos de teste planejados utilizando o plug-in Eclemma para o ambiente Eclipse.

09 – Testes
Título: Teste Funcional com Selenium-IDE
Mini-Resumo: Esta vídeo aula apresenta o planejamento e implementação de teste funcional para aplicações Web utilizando a ferramenta Selenium-IDE

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Carmelita Mulin – Atendimento ao Leitor
www.devmedia.com.br/central/default.asp
(21) 2220-5375

Kaline Dolabella
Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 2220-5375

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site SQL Magazine, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:
Rodrigo Oliveira Spínola - Colaborador
editor@sqlmagazine.com.br

ÍNDICE

Requisitos Não Funcionais

Critérios para análise arquitetural

Antonio Mendes da Silva Filho
antoniom.silvafilho@gmail.com
Professor e consultor em área de tecnologia da informação e comunicação com mais de 20 anos de experiência profissional, é autor dos livros Arquitetura de Software e Programando com XML, ambos pela Editora Campus/Elsevier, tem mais de 30 artigos publicados em eventos nacionais e internacionais, colunista para Ciência e Tecnologia pela Revista Espaço Acadêmico com mais de 60 artigos publicados, tendo feito palestras em eventos nacionais e no exterior. Foi Professor Visitante da University of Texas at Dallas e da University of Ottawa. Formado em Engenharia Elétrica pela Universidade de Pernambuco, com Mestrado em Engenharia Elétrica pela Universidade Federal da Paraíba (Campina Grande), Mestrado em Engenharia da Computação pela University of Waterloo e Doutor em Ciência da Computação pela Univesidade Federal de Pernambuco.

A organização das funcionalidades de software de um sistema é explicitada através da arquitetura de software. Há uma grande quantidade de estilos arquiteturais que trazem características peculiares a cada estilo, e estes, por sua vez, podem ser combinados gerando novos estilos. A heterogeneidade de estilos arquiteturais é salutar. Na verdade, isto ocorre devido à necessidade da arquitetura prover suporte a um conjunto de requisitos, às vezes conflitantes, dentre eles os requisitos não funcionais, tema abordado neste artigo.

Requisitos Não Funcionais e Arquitetura de Software

O projeto da arquitetura de software é uma etapa essencial no desenvolvimento de sistemas de software de grande porte e complexos. Dentro deste contexto, a arquitetura de software é fundamental para o desenvolvimento de linhas de produção de software onde se tem um conjunto de funcionalidades concebidas e implementadas a partir da mesma arquitetura (de software) base. Entretanto, antecedendo a etapa de projeto da arquitetura de software, há a necessidade de fazer o levantamento dos requisitos do sistema.

De um modo geral, o conjunto de requisitos de um sistema é definido durante as fases iniciais do processo de desenvolvimento. Tal conjunto de requisitos é visto como uma especificação do que deveria ser implementado. Os requisitos são descrições de como o sistema deveria se comportar, e contêm informações do domínio da aplicação e restrições sobre a operação do sistema.

Durante a fase de elicitação de requisitos, um projetista ou arquiteto de software faz uso de sua experiência a fim levantar os requisitos, buscando identificar características do sistema a ser desenvolvido. Além disso, informações do domínio juntamente com informações de estilos arquiteturais

existentes podem ser utilizados como fontes de dados que auxiliam na identificação dos requisitos.

Outro recurso que pode ser usado pelo projetista é construir cenários. Os cenários de uso oferecem suporte a requisitos específicos e visam tanto a elicitação quanto a análise de requisitos. Uma vez que o conjunto de requisitos tenha sido obtido, o projetista/arquiteto de software estará em condições de iniciar o projeto da arquitetura de software, conforme ilustrado na Figura 1. Este processo de levantamento e análise de requisitos, em conjunto com o uso de cenários, é usado no suporte da definição da arquitetura de software, como é discutido ao longo do artigo. É importante observar que a etapa de projeto arquitetural pode precisar fazer uso de cenários de uso ou mesmo uma re-análise a fim de refinar a arquitetura a ser empregada no sistema a ser desenvolvido.

O processo de desenvolvimento baseado na arquitetura considera a arquitetura de software como fator orientador do processo. Isto acarreta em colocarmos os requisitos não funcionais associados à arquitetura como principais aspectos do processo de desenvolvimento. Note que o desenvolvimento de um sistema de software centrado na arquitetura inicia-se com um arquiteto de software, de posse de um conjunto de requisitos do sistema. Nesse momento, busca-se identificar qual estilo ou combinação destes oferece suporte mais adequado a esses requisitos e, portanto, derivar uma arquitetura de software que atenda às características do sistema a ser desenvolvido. Vale ressaltar que a complexidade de um sistema de software é determinada tanto por seus requisitos funcionais – o que ele faz – quanto requisitos não funcionais – como ele faz. Tal distinção é baseada nas seguintes definições [Thayer 1990]:

Requisito funcional – um requisito de sistema de software que especifica uma função que o sistema ou componente deve ser capaz de realizar. Estes são requisitos de software que definem o comportamento do sistema, ou seja, o processo ou transformação que componentes de software ou hardware efetuam sobre as entradas para gerar as saídas. Esses requisitos capturam as funcionalidade sob o ponto de vista do usuário.

Requisito não funcional – em engenharia de sistemas de software, um requisito não funcional de software é aquele que descreve não o que o sistema fará, mas como ele fará. Assim, por exemplo, têm-se requisitos de desempenho, requisitos da interface externa do sistema, restrições de projeto e atributos da qualidade. A avaliação dos requisitos não funcionais é feita, em parte, por meio de testes, enquanto que outra parte é avaliada de maneira subjetiva.

Perceba que tanto os requisitos funcionais quanto os não funcionais possuem importância no desenvolvimento de um sistema de software. Entretanto, os requisitos não funcionais, também denominados de atributos de qualidade, têm um papel relevante durante o desenvolvimento de um sistema, atuando como critérios na seleção e/ou composição de uma arquitetura de software, dentre as várias alternativas de projeto.

Cabe salientar que à medida que os sistemas tornam-se maiores e mais complexos, o suporte a requisitos não funcionais depende cada vez mais de decisões tomadas no projeto da arquitetura de software. Trata-se de uma visão compartilhada pelos profissionais da área e, especificamente, pela comunidade de arquitetura de software.

Requisitos Não Funcionais

Requisitos não funcionais são aqueles que não estão diretamente relacionados à funcionalidade de um sistema. O termo requisitos não funcionais é também chamado de atributos de qualidade. Os requisitos não funcionais têm um papel de suma importância durante o desenvolvimento de um sistema, podendo ser usados como critérios de seleção na escolha de alternativas de projeto, estilo arquitetural e forma de implementação. Desconsiderar ou não considerar adequadamente tais requisitos é dispendioso, pois torna difícil a correção uma vez que o sistema tenha sido implementado. Suponha, por exemplo, que uma decisão tenha sido feita de modularizar a arquitetura de um sistema de modo a facilitar sua manutenção e adição de novas funcionalidades. Entretanto, modularizar um sistema adicionando uma camada a mais pode comprometer um outro requisito, o de desempenho. Portanto, faz-se necessário definir logo cedo quais requisitos não funcionais serão priorizados na definição de uma arquitetura.

Os requisitos não funcionais abordam aspectos de qualidade importantes em sistemas de software. Se tais requisitos não são levados em consideração, então o sistema de software poderá ser inconsistente e de baixa qualidade, conforme discutido acima. Para tanto, quanto mais cedo forem definidos os critérios arquiteturais, mais cedo o projetista pode identificar o estilo, ou combinação de estilos, mais apropriado ao sistema considerado.

Ao desenvolver um novo sistema de software bem como sua arquitetura, os projetistas ou engenheiros de software apresentam um conjunto de atributos de qualidade ou requisitos não funcionais

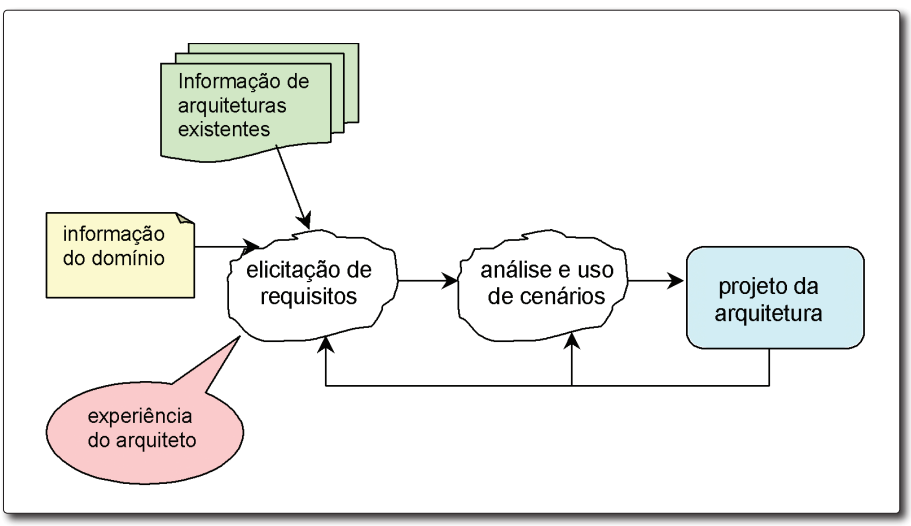


Figura 1. Elicitação de requisitos

que o sistema deveria suportar. Exemplo destes requisitos são desempenho, portabilidade, manutenibilidade e escalabilidade. A arquitetura de software deveria oferecer suporte a tais requisitos. Isto resulta da associação existente entre arquitetura de software e requisitos não funcionais. Importante observar que cada estilo arquitetural (isto é, a forma na qual o código do sistema é organizado) suporta requisitos não funcionais específicos. A estruturação de um sistema é determinante no suporte oferecido a um requisito não funcional. Por exemplo, o uso de camadas permite melhor separar as funcionalidades de um sistema, tornando-o mais modular e facilitando sua manutenção.

Considere, por exemplo, o padrão IEEE-Std 830-1993 [IEEE 1993] que lista um conjunto de 13 requisitos não funcionais a serem considerados no documento de especificação de requisitos de software. Este padrão inclui, dentre outros, requisitos de desempenho, confiabilidade, portabilidade e segurança.

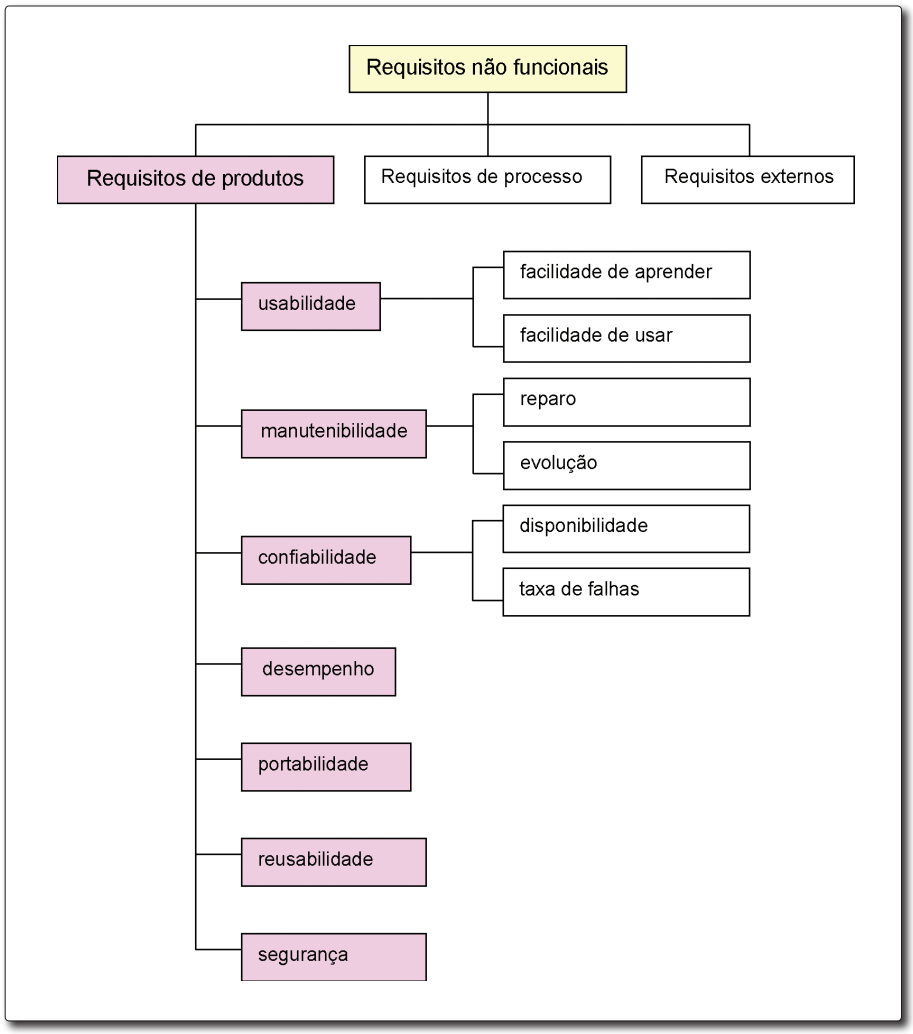
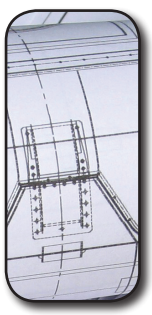


Figura 2. Tipos de requisitos não funcionais



1. Tempo para realizar um tarefa.
2. Percentual de tarefa concluído.
3. Percentual de tarefa concluído por unidade de tempo.
4. Taxa de sucessos/falhas.
5. Tempo consumido com erros.
6. Percentual de erros.
7. Número de comandos utilizados.
8. Número de comandos disponíveis não utilizados.
9. Freqüência de uso de *ajuda* (help) ou documentação.
10. Número de vezes que o usuário expressa satisfação ou frustação.

Figura 3. Critérios de medição de usabilidade

Embora haja um conjunto de propostas, consideradas como complementares, concentraremos nossas atenções num conjunto de requisitos diretamente associados a um sistema de software e, especificamente, à arquitetura de software. Este conjunto é baseado numa classificação apresentada por Sommerville, onde é feita a distinção entre requisitos externos, de produto, e de processo [Sommerville 2007].

A Figura 2 é uma adaptação da proposta de Sommerville, onde consideramos os requisitos de produto, associados à arquitetura de software, bem como adicionamos outros não presentes na proposta original de Sommerville. É importante observar que a Figura 2 mostra um subconjunto de requisitos não funcionais, denominados de requisitos de produtos os quais estão associados à arquitetura de um sistema de software. Note que a classificação apresentada em [Sommerville 2007] ainda considera os requisitos de processo e requisitos externos como sendo requisitos não funcionais, além dos requisitos de produtos. A figura exibe um conjunto de 7 requisitos não funcionais, sendo alguns destes ainda decompostos.

Usabilidade

Usabilidade é um dos atributos de qualidade ou requisitos não funcionais de qualquer sistema interativo, ou seja, no qual ocorre interação entre o sistema e seres humanos. A noção de usabilidade vem do fato que qualquer sistema projetado para ser utilizado pelas pessoas deveria ser fácil de aprender e fácil de usar, tornando assim fácil e agradável a realização de qualquer tarefa.

Requisitos de usabilidade especificam tanto o nível de desempenho quanto a satisfação do usuário no uso do sistema. Dessa forma, a usabilidade pode ser expressa em termos de:

- **Facilidade de aprender** – Associado ao tempo e esforço mínimo exigido para alcançar um determinado nível de desempenho no uso do sistema.
- **Facilidade de uso** – Relacionado à velocidade de execução de tarefas e à redução de erros no uso do sistema.

Os requisitos de usabilidade são coletados juntamente com outros requisitos (de dados e funcionais) usando algumas

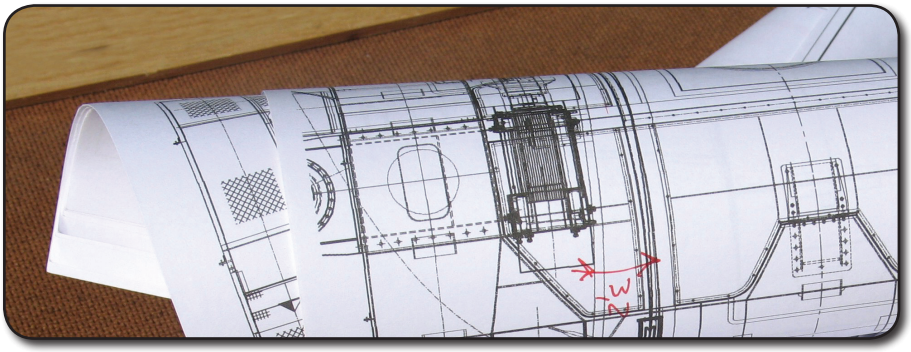
das técnicas de elicitação de requisitos como entrevistas ou observação. A coleta desses dados pode ocorrer, por exemplo, verificando o log de ações do usuário quando do uso de funcionalidade do sistema.

Esses requisitos de usabilidade podem ser expressos através de métricas de usabilidade, expressas em termos de medidas de desempenho. Tyldesley apresentou um conjunto de critérios que podem ser utilizados durante a medição de usabilidade [Tyldesley 1988]. A seleção de critérios a serem usados para mensurar a usabilidade depende do tipo de sistema. Exemplos de critérios de medição de usabilidade são apresentados na Figura 3.

A definição das metas de usabilidade através de métricas é parte do processo denominado de engenharia de usabilidade. Neste processo, faz-se necessário também estabelecer os níveis desejados de usabilidade. Se, por exemplo, o usuário tem dificuldade em encontrar a funcionalidade desejada no sistema e, conseqüentemente, precisa recorrer à ajuda (Help) ou expressa insatisfação, então se observa que dois dos critérios da Figura 3 são considerados. A quantidade de vezes que essas ocorrências são observadas serve de indicador do suporte oferecido à usabilidade pelo sistema.

A usabilidade é um dos atributos de qualidade de um sistema e tem sido cada vez mais levada em consideração durante o desenvolvimento de software. A usabilidade pode ser afetada pelos componentes funcional (ou de aplicação) e de apresentação de um sistema. Mesmo que esses componentes sejam bem projetados, ainda assim a usabilidade poderá ficar comprometida se a arquitetura do sistema não levar em consideração a facilidade de efetuar uma modificação.

É importante acrescentar que a simples separação arquitetural entre componente de aplicação e apresentação em sistemas interativos não é suficiente para assegurar a usabilidade do mesmo. Assim, para determinar se uma arquitetura de software satisfaz a aspectos de usabilidade, torna-se necessário desenvolver cenários de uso do sistema. Em tais cenários, busca-se assegurar que



informações corretas estejam disponíveis ao usuário no momento adequado, bem como encaminhar corretamente as instruções/comandos dos usuários a componentes apropriados do sistema. Note que a arquitetura de software do sistema tem um papel de suma importância visto que tais informações serão trocadas entre componentes do sistema e entre estes e o usuário, fluindo através de conectores da arquitetura.

Manutenibilidade

O termo manutenção de software é geralmente empregado quando nos referimos às modificações feitas após o sistema de software ter sido disponibilizado para uso. Na realidade, o termo manutenibilidade é um tanto abrangente já que ele envolve tanto a atividade de reparo (de algum defeito existente no sistema de software) quanto a atividade de alteração/evolução de características existentes ou adição de novas funcionalidades não previstas ou capturadas no projeto inicial.

O reparo de um sistema de software ocorre quando defeitos são detectados, fazendo-se necessária a correção deles. A capacidade de efetuar um reparo depende do número de componentes do sistema. Por exemplo, se o sistema é monolítico, ou seja, constituído de um único componente, então tornar-se mais difícil efetuar o reparo se este sistema de software é de grande porte.

No entanto, se o sistema de software é modularizado, então tende a ser mais fácil analisar e reparar o existente. Podemos dizer que a modularidade favorece a capacidade de fazer reparo, permitindo que defeitos fiquem confinados a poucos módulos, considerando-se que se tenha a funcionalidade adequadamente separada. Uma observação importante é que a necessidade de fazer reparo é minimizada à medida que a confiabilidade do sistema aumenta.

Similarmente a outros sistemas, os sistemas de software evoluem ao longo do tempo, seja com a adição de novas funcionalidades ou com a modificação das existentes. Esta capacidade de evolução de um sistema de software é também influenciada pela modularidade do sistema. Note que se a arquitetura do sistema não considerar sua possibilidade de evolução por meio da adição e/ou alteração de funcionalidades do sistema, modificações tornar-se-ão cada vez mais difíceis à medida que o software evolui.

De um modo geral, a manutenibilidade é um dos requisitos mais relacionados com a arquitetura de um sistema de software. A facilidade de fazer alteração no sistema existente, seja adicionando ou modificando alguma funcionalidade, depende muito da arquitetura deste. Se considerarmos a necessidade de incrementar a quantidade de componentes encarregados do processamento de uma ligação telefônica (numa central telefônica) ou de transações eletrônicas, então esta adição de novos componentes deve ocorrer sem a necessidade de modificar a arquitetura existente e ainda comprometer pouco (ou nada) o desempenho atual do sistema. Tal suporte à manutenibilidade (seja para correção ou evolução do sistema) deve ser facilmente acomodada pela arquitetura de software.

É importante lembrar que uma arquitetura de software define os componentes e conexões entre estes e, portanto, também definem sob que circunstâncias componentes ou conectores podem ser alterados. Dessa forma, uma

arquitetura ou estilo arquitetural de um sistema de software deveria efetivamente acomodar as modificações que precisarem ser feitas tanto durante seu desenvolvimento quanto após o sistema entrar em operação.

Confiabilidade

Confiabilidade de software é a probabilidade de o software não causar uma falha num sistema durante um determinado período de tempo sob condições especificadas. A probabilidade é uma função da existência de defeitos no software. Assim, os estímulos recebidos por um sistema determinam a existência ou não de algum defeito. Em outras palavras, a confiabilidade de software, geralmente definida em termos de comportamento estatístico, é a probabilidade de que o software irá operar como desejado num intervalo de tempo conhecido. Também, a confiabilidade caracteriza-se um atributo de qualidade de software o qual implica que um sistema executará suas funções como esperado.

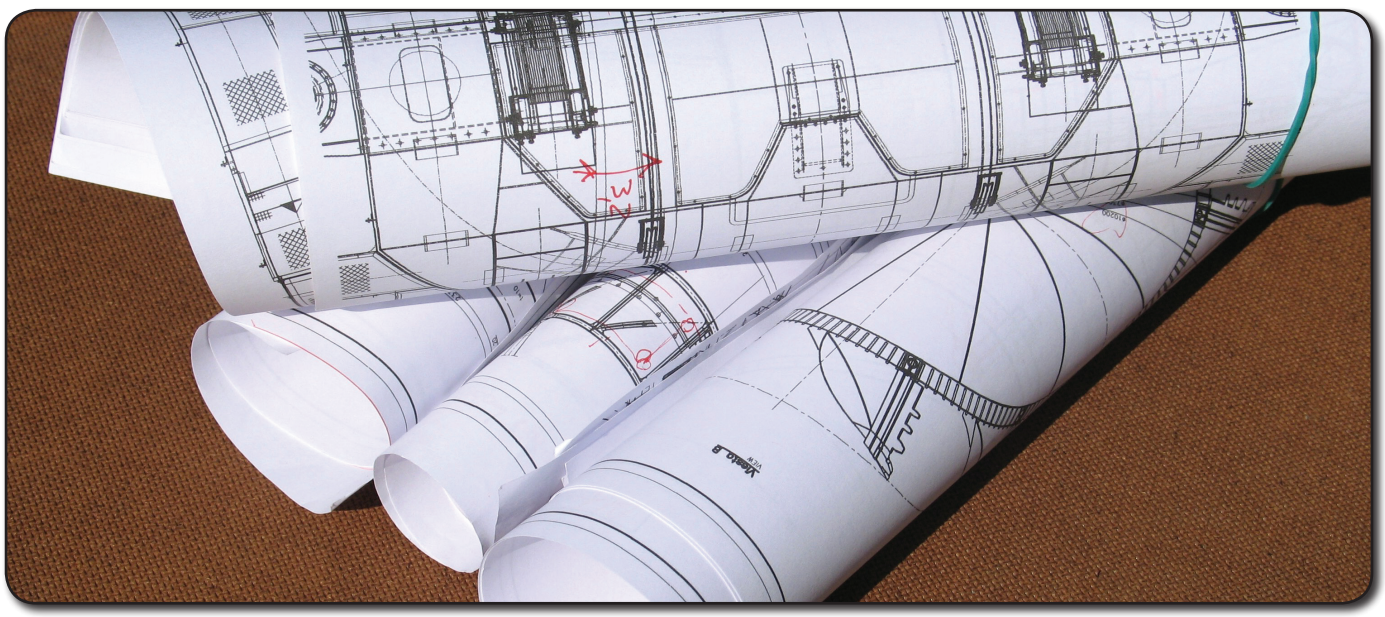
Os requisitos de confiabilidade compreendem restrições sobre o comportamento do sistema de software em tempo de execução. Na realidade, tem-se um conjunto de métricas de confiabilidade de software associadas a esses requisitos. Geralmente, as falhas de um componente de software são de natureza transitória, ou seja, elas ocorrem apenas para algumas entradas (estímulos) enquanto o sistema poderá continuar operando normalmente em outras circunstâncias. Isto distingue o software do hardware já que as falhas no segundo são de natureza permanente. Vale ressaltar que falha é o que se observa pelos usuários (externamente) enquanto que os defeitos, de origem interna ao sistema, são os motivadores das falhas.

Não é tão simples relacionar a disponibilidade de um sistema de software a uma falha existente, pois isto depende de vários fatores, tais como grau de corrupção de dados em decorrência de algum defeito de software e tempo de re-inicialização, dentre outros. Exemplos de métricas utilizadas para avaliar a confiabilidade de software compreendem:

- Disponibilidade – Esta é uma medida de quão disponível o sistema estaria para uso, isto é, quão disponível o sistema estaria para efetuar um serviço solicitado por algum usuário. Por exemplo, um serviço de um sistema de software terá uma disponibilidade de 999/1.000. Isto significa que dentre um conjunto de 1.000 solicitações de serviço, 999 deverão ser atendidas. Esta métrica é muito importante em sistemas de telecomunicações, por exemplo.
- Taxa de ocorrência de falha – Esta é uma medida da freqüência na qual o sistema falha em prover um serviço como esperado pelo usuário, ou seja, a freqüência na qual um comportamento inesperado é provável de ser observado. Por exemplo, se temos uma taxa de ocorrência de falha de 2/1.000, isto significa que 2 falhas são prováveis de acontecerem para cada 1.000 unidades de tempo.
- Probabilidade de falha durante fase operacional – Esta é uma medida da probabilidade que o sistema irá comporta-se de maneira inesperada quando em operação. Esta métrica é de suma importância em sistemas críticos que requerem uma operação contínua.
- Tempo médio até a ocorrência de falha ou Mean Time To Failure (MTTF) – Esta é uma medida do tempo entre falhas observadas. Note que esta métrica oferece um indicativo de quanto tempo o sistema permanecerá operacional antes que uma falha aconteça.

Qualquer métrica que venha a ser utilizada para avaliar a confiabilidade de um sistema dependerá da forma como o sistema é usado. Note ainda que o tempo é um fator considerado nas métricas. Ele é escolhido de acordo com a aplicação. Há sistemas de software que operam de forma contínua, enquanto outros operam de maneira periódica.

Por exemplo, considere um caixa eletrônico de banco. Este é um exemplo de sistema que opera periodicamente, isto



é, um caixa eletrônico encontra-se parte do tempo em operação, enquanto no restante do tempo fica ocioso (embora disponível para uso por algum cliente do banco). No exemplo de um caixa eletrônico de banco, uma unidade de tempo mais adequada é o número de transações. Assim, um exemplo de falha seria a perda de dados entrados por um usuário. Neste caso, a especificação de confiabilidade poderia ser a ocorrência de uma falha dessa natureza a cada 10.000 transações.

A arquitetura de software influenciará na confiabilidade de um sistema. O tempo médio até a ocorrência de uma falha ou MTTF poderá ser reduzido se houver replicação de componentes críticos. A perda de um desses componentes incorre em falha.

Uma forma de evitar isto ou contornar a perda de um componente é provendo uma arquitetura tolerante a falha onde uma réplica de um componente assume o processamento do componente com falha, evitando assim qualquer interrupção na operação de um sistema. Outra alternativa é degradar o desempenho do sistema sobrecarregando um componente com um número maior de requisições do que ele foi projetado. Dessa forma, a qualidade do sistema é degradada, mas ainda assim o sistema continua a funcionar (embora de modo precário até que uma ação corretiva seja tomada). A medida de confiabilidade pode ser definida em termos do tempo médio entre a ocorrência de falhas, ou Mean Time Between Failure (MTBF). Esta medida é dada por:

$$MTBF = MTTF + MTTR$$ (MTTR ou Mean Time To Repair é o tempo médio de reparo)

A medida de disponibilidade também pode ser descrita em termo do MTTF e é definida como:

$$\text{Disponibilidade} = \frac{MTTF}{(MTTF + MTTR)} \times 100\%$$

Se considerarmos essas medidas, é importante observar que se o MTTR é reduzido, então a disponibilidade e confiabilidade do sistema serão maiores. Isto pode ser obtido arquiteturalmente se a separação de interesses for considerada durante o projeto. Perceba que quão menor é o tempo para proceder o reparo da falha, mais rapidamente o sistema voltará a ficar operacional e, portanto, a ficar disponível. Este atributo de projeto leva a uma maior integração, bem como maior facilidade nas modificações necessárias no sistema.

Note que adicionar componentes redundantes a um sistema de software implicará numa maior confiabilidade. Esta redundância é acrescentada na forma de verificações adicionais realizadas a fim de detectar erros antes que eles ocasionem falhas no sistema. Todavia, o uso de componentes redundantes acarreta numa redução de desempenho do sistema como discutido a seguir.

Desempenho

Desempenho é um atributo de qualidade importante para sistemas de software. Considere, por exemplo, um sistema de uma administradora de cartões de crédito. Em tal sistema, um projetista ou engenheiro de software poderia considerar os requisitos de desempenho para obter uma resposta de tempo para autorização de compras por cartão.

Note que os requisitos de desempenho têm impacto mais global sobre o sistema e, por essa razão, estão entre os requisitos não funcionais mais importantes. Contudo, é geralmente difícil lidar com os requisitos de desempenho e com outros requisitos não funcionais uma vez que eles estão em conflito, conforme discutido acima. No início da atividade de projeto da arquitetura de software torna-se necessário definir quais requisitos não funcionais serão priorizados, dada a possibilidade de conflito entre eles.

Adicionalmente, desempenho é importante porque afeta a usabilidade de um sistema. Se um sistema de software é lento, ele certamente reduz a produtividade de seus usuários ao ponto de não atender às suas necessidades. Também, se o sistema de software requer muito espaço em disco para armazenamento de informações, pode ser oneroso utilizá-lo. Por exemplo, se um sistema de software exige muita memória para ser executado, ele pode afetar outras aplicações que são executadas no mesmo ambiente. Além disso, ele pode executar tão lentamente que o sistema operacional tenta balancear o uso de memória entre as diversas aplicações. De um modo geral, o requisito de desempenho pode ser decomposto em termos de tempo e espaço, como ilustrado na Figura 4.

O requisito de desempenho restringe a velocidade de operação de um sistema de software. Isto pode ser visto em termos de:

- Requisitos de resposta – Especificam o tempo de resposta de um sistema de software aceitável para usuários. Neste caso, um projetista

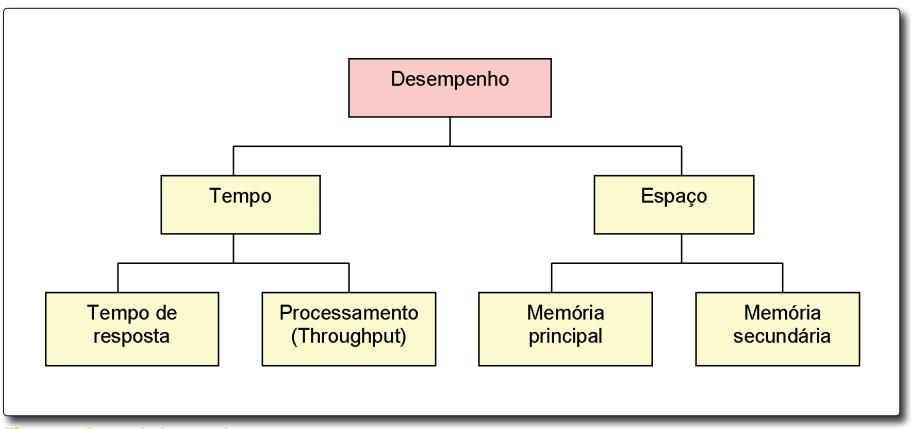
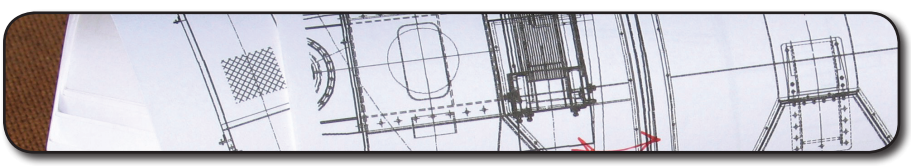


Figura 4. Fatores de desempenho



poderia especificar que o sistema deveria responder à solicitação de um serviço específico de um usuário dentro de um intervalo de 2 segundos. Por exemplo, num caixa eletrônico, após o usuário inserir o cartão magnético do banco no local apropriado (leitor do equipamento), o sistema deveria exibir uma nova tela, num intervalo de 2 segundos, requerendo que o usuário digite sua senha de conta corrente. Numa outra situação, o usuário pode ser solicitado a digitar sua senha e não o faz dentro de um período de 20 segundos, quando então um timeout ocorre e o sistema retorna à tela inicial.

- Requisitos de processamento (throughput) – Estes requisitos especificam a quantidade de dados que deveria ser processada num determinado período de tempo. Um exemplo seria exigir que o sistema de software possa processar, no mínimo, 6 transações por segundo.
- Requisitos de temporização – Este tipo de requisito especifica quão rapidamente o sistema deveria coletar dados de entrada de sensores antes que outras leituras de dados de entrada, feitas posteriormente, sobrescrevam os dados anteriores. Assim, por exemplo, poderia ser especificado que o sistema deveria efetuar leitura de dados 5 vezes por segundo, como condição mínima.
- Requisitos de espaço – Em alguns casos, os requisitos de espaço podem ser considerados. Aqui, podemos nos referir à memória principal ou secundária. Por exemplo, a memória principal para executar uma aplicação poderia ser considerada como um requisito de desempenho uma vez que ela está relacionada ao comportamento do sistema em tempo de execução.

É importante observar que o desempenho depende da interação existente entre os componentes de um sistema de software e, portanto, está muito associado à arquitetura. Neste caso, os mecanismos de comunicação utilizados pelos componentes de um sistema têm influência sobre o desempenho obtido. Conforme vimos anteriormente, desempenho está relacionado a outros requisitos não funcionais. Por exemplo, a confiabilidade melhora com o uso de componentes redundantes. Todavia, o desempenho fica bastante comprometido, implicando na sua redução.

Portabilidade

Portabilidade pode ser definida como a facilidade na qual o software pode ser transferido de um sistema computacional ou ambiente para outro. Em outras palavras, o software é dito portável se ele pode ser executado em ambientes distintos. Note que o termo ambiente pode referir-se tanto à plataforma de hardware quanto a um ambiente de software como, por exemplo, um sistema operacional específico.

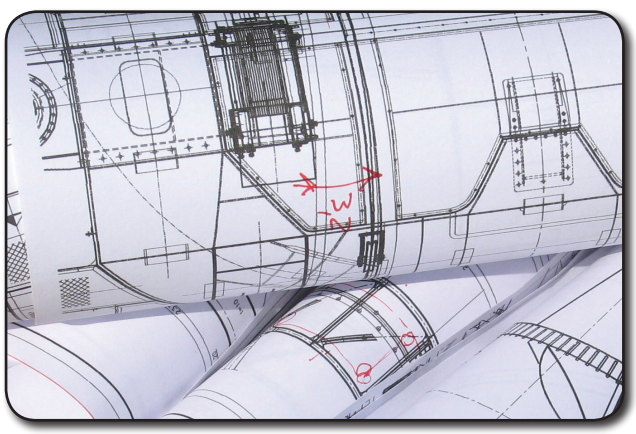
De um modo geral, a portabilidade refere-se à habilidade de executar um sistema em diferentes plataformas. É importante observar que à medida que aumenta a razão de custos entre software e hardware, a portabilidade torna-se cada vez mais importante. Adicionalmente, podemos ter a portabilidade de componentes e a portabilidade de sistemas. Esta última situação pode ser vista como caso especial de reusabilidade. O reuso de software ocorre quando todo o sistema de software é reutilizado, implementando-o em diferentes sistemas computacionais.

A portabilidade de um componente ou sistema de software é proporcional à quantidade de esforço necessário para que funcione num novo ambiente. Se uma quantidade menor de esforço é exigida quando comparada ao trabalho de desenvolvimento, então o sistema é dito portável.

Dois aspectos relevantes na portabilidade de programas são a transferência e adaptação. A transferência é o movimento de componente (código de um programa e dados associados) de um ambiente para outro. A adaptação engloba as modificações exigidas para fazer com que o programa seja executado em um novo ambiente.

Cabe salientar que o ambiente no qual um sistema de software opera é normalmente composto de hardware, sistema operacional, sistema de entrada e saída (E/S), bem como a aplicação. Uma abordagem geral que poderia ser adotada para obter um sistema portável tentaria separar as partes do sistema que dependem do ambiente externo numa camada ou interface de portabilidade, conforme ilustrado na Figura 5.

A interface de portabilidade mostrada na figura acima poderia ser vislumbrada e projetada como um conjunto de tipos de dados abstratos ou objetos, os quais iriam encapsular os elementos não portáveis procurando esconder características do software da aplicação. Assim,



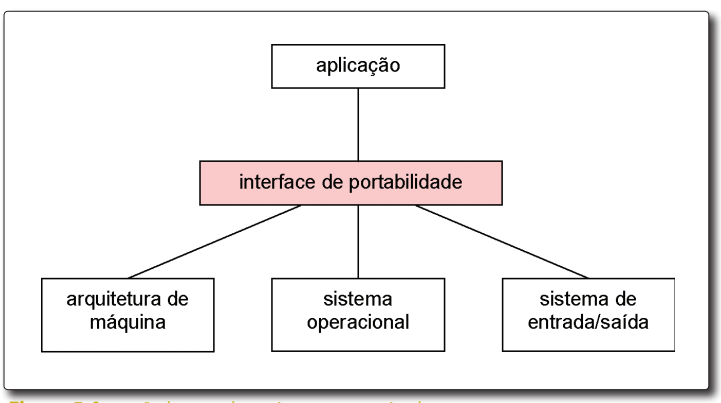


Figura 5. Separação de partes de um sistema computacional

quando o sistema de software muda de hardware ou sistema operacional, apenas a interface de portabilidade precisaria ser alterada. Alguns problemas de portabilidade surgem, geralmente, devido à adoção de diferentes convenções para representar informação em arquiteturas de máquinas distintas, ou seja, hardware diferente.

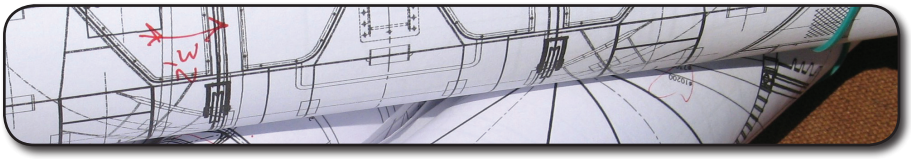
Reusabilidade

Uma característica das engenharias é fazer uso de projetos existentes a fim de reutilizar componentes já desenvolvidos, objetivando minimizar o esforço em novos projetos. Dessa forma, componentes que já tenham sido desenvolvidos e testados podem ser reutilizados. Considere os elevados níveis de reusabilidade que encontramos tanto na indústria de automóveis quanto de aparelhos eletrônicos. Na indústria de automóveis, por exemplo, um motor é geralmente reutilizado de um modelo de carro para outro.

Em Engenharia de Software, à medida que aumenta a pressão para reduzir custos de desenvolvimento e manutenção de sistemas de software, bem como pela obtenção de sistemas com qualidade elevada, torna-se necessário considerar a reusabilidade como requisito não funcional no desenvolvimento de novos sistemas.

O reuso pode ser visto sob diferentes perspectivas. Ele pode ser orientado a componentes, orientado a processos ou orientado ao conhecimento específico de um domínio. Note que ainda poderíamos considerar o reuso de requisitos. Entretanto, aqui, iremos concentrar nossa atenção no reuso orientado a componentes. Exemplos desse tipo de reuso são:

- Aplicação – Toda a aplicação poderia ser reutilizada.
- Subsistemas – Os principais subsistemas de uma aplicação poderiam ser reutilizados.
- Objetos ou módulos – Componentes de um sistema, englobando um conjunto de funções, podem ser reutilizados.
- Funções – Componentes de software que implementam uma única função (como uma função matemática) podem ser reutilizados.



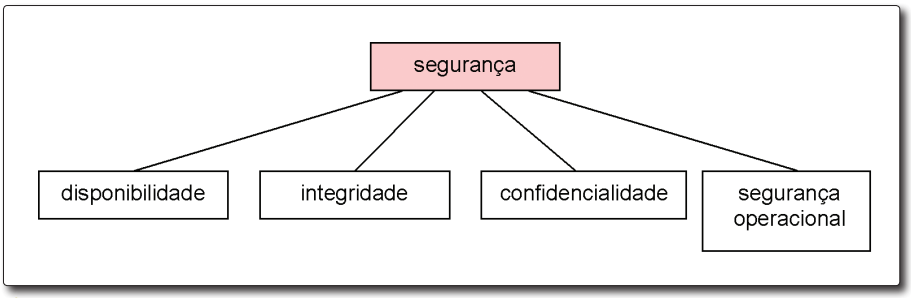


Figura 6. Tipos de segurança

Dois tipos de reuso que estamos mais interessados são o reuso de subsistemas e objetos ou componentes, que chamaremos, simplesmente, de reuso de componentes. Este tipo de reuso não envolve apenas o código, mas também engloba a arquitetura e projeto associados.

É importante observar que podemos obter ganhos se reutilizarmos tanto projetos quanto arquiteturas. Isto minimiza esforços de desenvolvimento e requer menos alterações ou adaptações. Na realidade, quando se tem em mente que é necessário prover suporte à fácil modificação, indiretamente, obtemos componentes reutilizáveis.

Assim, o requisito reusabilidade pode envolver a arquitetura de um sistema de software ou componentes deste. O que determinará quão fácil será conseguir componentes reutilizáveis é a interdependência ou acoplamento entre os componentes. Por exemplo, uma biblioteca de componentes que oferece um conjunto de funcionalidades já implementadas e testadas oferece ao usuário (programador) um recurso valioso, já que basta incorporar esses componentes ao seu código e acessar suas funcionalidades através de sua interface.

Segurança

Em um sistema de software, este requisito não funcional caracteriza a segurança de que acessos não autorizados ao sistema e dados associados não serão permitidos. Portanto, é assegurada a integridade do sistema quanto a ataques intencionais ou acidentes. Dessa forma, a segurança é vista como a probabilidade de que a ameaça de algum tipo será repelida.

Adicionalmente, à medida que os sistemas de software tornam-se distribuídos e conectados a redes externas, os requisitos de segurança vão se tornando cada vez mais importantes. Exemplos de requisitos de segurança são:

- Apenas pessoas que tenham sido autenticadas por um componente de controle acesso e autenticação poderão visualizar informações dado que a confidencialidade permite esse tipo de acesso apenas às pessoas autorizadas.
- As permissões de acesso ao sistema podem ser alteradas apenas pelo administrador de sistemas.
- Deve ser feito cópias (backup) de todos os dados do sistema a cada 24 horas e estas cópias devem ser guardadas em um local seguro, sendo preferencialmente num local diferente de onde se encontra o sistema.
- Todas as comunicações externas entre o servidor de dados do sistema e clientes devem ser criptografadas.

Requisitos de segurança são essenciais em sistemas críticos, tais como sistemas de controle de vôo de aeronaves, uma vez que é impossível confiar num sistema deste tipo se ele não for seguro. Assim, pode-se dizer que todos os sistemas críticos possuem associados a eles requisitos de segurança. Os requisitos não funcionais de segurança envolvem diferentes aspectos. A Figura 6 mostra tipos de segurança que podemos encontrar.

Também é importante considerar as diferentes ênfases encontradas para segurança:

- Disponibilidade – Refere-se a assegurar o sistema contra qualquer interrupção de serviço.

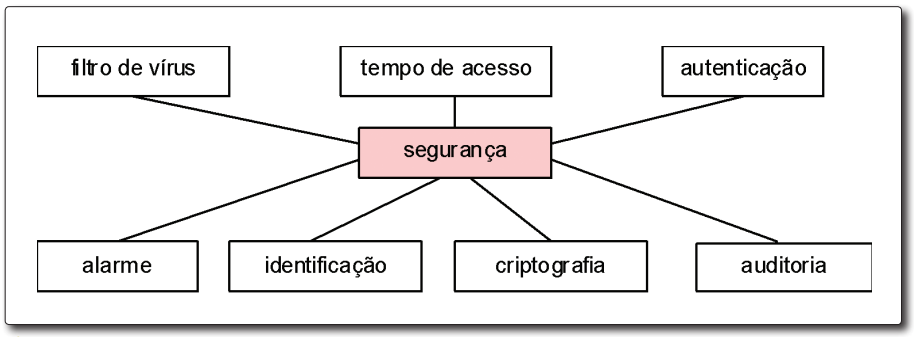


Figura 7. Métodos usados para prover segurança

- Integridade – O foco na integridade ocorre principalmente em sistemas comerciais, onde se busca assegurar que acesso ou atualizações não autorizadas ocorram.
- Confidencialidade – A ênfase aqui é a de não permitir a revelação não autorizada de informações.
- Segurança operacional – Refere-se à fase considerada para o sistema em uso.

Note que para satisfazer ao requisito de qualidade não funcional de segurança em um sistema de software, alguns métodos podem ser empregados. Esses métodos podem ser vistos como um refinamento da meta de prover segurança a um sistema de software. Como exemplos desses métodos, podemos considerar:

1. Identificação – Identifica o nome do usuário, informando ao sistema quem está utilizando-o.
2. Autenticação – Visa assegurar que os usuários são, de fato, quem afirmam ser. Para tanto, fazem um teste de identidade. Este método envolve alguns aspectos, tais como:
   - Tipo de protocolo usado – isto requer operação de senha.
   - Quantidade de autenticações – pode requerer uma única senha ou múltiplas senhas ou procedimentos. Por exemplo, alguns bancos já fazem uso de múltiplas senhas durante operação de autenticação.
   - Partes envolvidas – isto pode envolver a autenticação de uma parte envolvida (cliente) ou de ambas as partes envolvidas (cliente e sistema).
3. Tempo de acesso – Busca limitar o tempo de acesso ao sistema a fim de reduzir qualquer tipo de ameaça.
4. Auditoria de segurança – Objetiva habilitar pessoal autorizado a monitorar o sistema e, seletivamente, rastrear eventos importantes.
5. Alarme – Esta operação visa prevenir acessos, potencialmente suspeitos às informações vitais ou dados do sistema, notificando esses acessos à supervisão de segurança do sistema ou devidas autoridades.

A Figura 7 apresenta alguns métodos que podem ser empregados em requisitos de segurança.

É importante observar que a arquitetura de um sistema de software deve levar em consideração esses aspectos a fim de atender aos requisitos de segurança. Entretanto, o tipo de sistema determinará quais fatores precisarão ser levados em conta. Assim, poderá haver a inserção de componentes específicos de segurança bem como a conexão destes com outros componentes funcionais.

Conclusão

A elicitação de requisitos funcionais e não funcionais é etapa fundamental no desenvolvimento de sistemas de software. Elementos de entrada desse processo de elicitação compreendem os principais influenciadores do sistema, envolvendo projetista, arquiteto e usuários.

Aliado a isso, a experiência do arquiteto de software é de grande importância. Como saída deste processo, tem-se um conjunto de requisitos funcionais oferecendo suporte às funcionalidades do sistema e uma lista de requisitos não funcionais oferecendo suporte à arquitetura de software. Cabe destacar que, quando da análise de arquiteturas candidatas para um sistema de software, um arquiteto ou engenheiro de software considera os requisitos não funcionais como um dos principais critérios para sua análise.

Por fim, é importante notar que uma cobertura completa de todos os possíveis requisitos não funcionais está fora do objetivo deste artigo. Para tanto, o leitor pode consultar o livro intitulado Non-Functional Requirements in Software Engineering de L. Chung, E. Yu, and J. Mylopoulos. Todavia, um subconjunto dos mais proeminentes requisitos não funcionais foi apresentado. Esses requisitos servem, via de regra, como critério para análise arquitetural objetivando a definição da arquitetura de software de um sistema.

Links

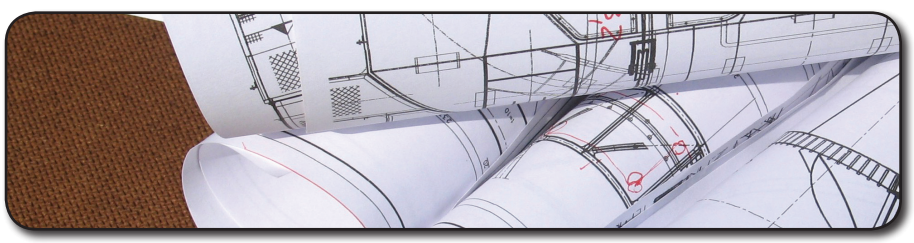
Non-Functional Requirements in Software Engineering
http://www.utdallas.edu/~chung/BOOK/book.html

Are All Quality Goals Created Equal? Functional vs. Non-Functional
www.sei.cmu.edu/architecture/saturn/2005/quality_steven.pdf

Acquisition Practices: Good and Bad
www.sei.cmu.edu/programs/acquisition-support/conf/2003-presentations/oberndorf.pdf

SEI's Software Architecture Technology Initiative
www.sei.cmu.edu/architecture/sat_init.html

The Software Architecture Portal
http://www.softwarearchitectureportal.org/



Já pensou em hospedar todos os seus sites em um único plano de hospedagem?

PLANO PROFISSIONAL 1

- 3 domínios independentes
- 50 GB de transferência
- 30 GB de e-mails com SSL
- 1 GB de espaço
- Painel de controle
- ASP, ASP.NET 3.5 E PHP 5
- Access ilimitado
- 3 bancos MYSQL 5
- 1 banco SQL Server 2005 Express

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe!

RVSTMEDIA

Tudo isso por apenas R$ **25,90** por mês

Na Hospedix você ainda ganha registro de domínio (.com .net .org ou .info) grátis e isento de taxas anuais enquanto for cliente.

Acesse agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

hospedix.com.br

HOSPEDIX
HOSPEDAGEM PROFISSIONAL

Desenvolvimento de Software Dirigido por Caso de Uso

Parte II: Especificando Caso de Uso

Vinicius Lourenço de Sousa
viniciuslsousa@gmail.com
Atua no ramo de desenvolvimento de software há mais de 10 anos, é autor de diversos artigos publicados pelas revistas ClubeDelphi e SQL Magazine. É Graduado em Tecnologia da Informação pela ABEU Faculdades Integradas e Pós-Graduado em Análise, Projeto e Gerência de Sistemas pela PUC-RJ, IBM Certified: Especialista Rational Unified Process e instrutor de UML, Análise OO e Java. Atualmente trabalha na CPM Braxis como Especialista nas áreas de arquitetura, especificação de requisitos, levantamento e modelagem de processo de negócio e projetista de software em soluções com componentes de negócio e SOA.

No artigo anterior explicamos a importância do desenvolvimento de software utilizando como artefato principal o caso de uso (ver Figura 1) para dirigir todo o esforço de desenvolvimento. Foram também explicados os conceitos do caso de uso e as características e regras para construirmos um modelo de caso de uso (diagrama UML) correto e coeso em relação ao negócio do cliente. Também foram mostrados os benefícios que a utilização do caso de uso traz para o desenvolvimento de software de forma que nos possibilite a rastreabilidade de requisitos que sofrem impactos devido a mudanças nas regras de negócio. Por fim, também foi visto que os impactos decorrentes de alterações durante o ciclo de vida do desenvolvimento de sistemas são minimizados com o caso de uso.

Neste artigo daremos continuidade ao desenvolvimento de software dirigido por caso de uso, mostrando boas práticas sobre como especificar um caso de uso. Com isso, toda a equipe de desenvolvimento poderá executar suas atividades e gerar os artefatos necessários, assim como criar o software de maneira mais robusta, compreensível, com maior facilidade e mantendo o mesmo nível de comunicação entre os membros da equipe.

Especificação de Caso de Uso

Antes de começarmos a descrever as boas práticas de como especificar casos de uso, temos que nos atentar para algumas considerações. Não existe um padrão formal para se criar uma especificação de caso de uso, ou seja, não existe um modelo único para que um Analista de Requisitos possa usar quando ele for descrever como um caso de uso deve funcionar. O que existe são alguns modelos pré-existentes e a escolha de sua utilização fica a critério da equipe de desenvolvimento ou é feita uma adaptação de um desses modelos para as necessidades de um projeto(s) específico(s) ou cliente(s) específico(s):

- **Descrição Contínua** – Nesse modelo a especificação do caso de uso será uma descrição textual livre, explicando todo o funcionamento do caso de uso e seus "cenários". Esse modelo foi introduzido por Ivar Jacobson e seus colaboradores e é muito bom quando queremos descrever como funciona o caso de uso no nível de processo de negócio, entendendo como o negócio do cliente funciona. Esse tipo de especificação funciona muito bem para casos de uso de negócios em que o objetivo é entender o processo de negócio do cliente, mas também pode ser usado em um caso de uso sistêmico como uma versão preliminar. Detalharemos casos de uso de negócios em outra parte do artigo. Veja o exemplo no Quadro 1;
- **Tabela de Narrativa Particionada** – Nesse modelo colocamos uma tabela na especificação do caso de uso contendo duas colunas, sendo a primeira coluna a ação do ator e a segunda coluna a reação do sistema. A leitura de um caso de uso estruturado nesse modelo pode ser feita em ziguezague. Esse modelo foi proposto por Rebecca Wirfs-Brock e outros. Veja o exemplo no Quadro 2;
- **Descrição Numerada** – Nesse modelo a especificação do caso de uso é feita através de passos do ator e do sistema, ou seja, são criados tópicos de numeração onde em uma linha o ator executa seus passos e em outra o sistema também executa seus passos. Esse é o modelo mais usado que eu tenho visto em desenvolvimento de software e o que torna a leitura mais fácil para o cliente que irá validar o caso de uso. Tomaremos como base para o artigo esse tipo de modelo. Veja o exemplo no Quadro 3.

Seções da Especificação do Caso de Uso

Agora que conhecemos os modelos de especificação de casos de uso, iremos descrever as seções que compõem uma especificação de caso de uso. Essas seções são baseadas em um template de

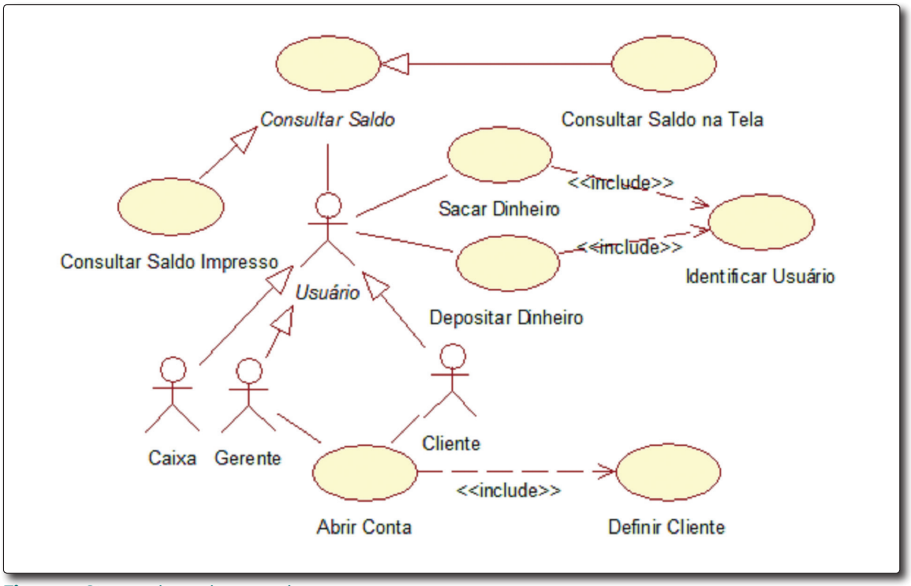


Figura 1. Diagrama de caso de uso completo

Quadro 1. Especificação de caso de uso com descrição contínua

Caso de Uso: Abrir Conta
Ator: Gerente e Cliente
Descrição: O Gerente ao receber a solicitação de abertura de conta pelo cliente, inicia o cadastramento do mesmo informando para o sistema os dados pessoais, o tipo da conta (corrente ou poupança) e se a conta é em conjunto. Em seguida o Gerente solicita que o cliente informe a senha da conta no sistema para que o processo possa continuar. Caso o cliente já esteja cadastrado no sistema, não é necessário o cadastramento do cliente. Ao final do procedimento, o sistema informa o número da nova conta e o Gerente repassa a informação ao cliente.

Quadro 2. Especificação de caso de uso com narrativa particionada

Caso de Uso: Sacar Dinheiro

Usuário	Sistema
Insere cartão no caixa eletrônico.	Apresenta solicitação de senha.
Informa senha.	Apresenta menu de operações.
Solicita realização de saque.	Solicita quantia a ser sacada.
Informa valor que deseja sacar.	Fornece a quantia informada.
Retira a quantia.	

Quadro 3. Especificação de caso de uso com descrição numerada

Caso de Uso: Sacar Dinheiro
Ator: Usuário

1) Usuário insere cartão no caixa eletrônico.
2) Sistema apresenta solicitação de senha.
3) Usuário informa senha.
4) Sistema apresenta menu de operações.
5) Usuário solicita realização de saque.
6) Sistema solicita quantia a ser sacada.
7) Usuário informa valor que deseja sacar.
8) Sistema fornece a quantia informada.
9) Usuário retira a quantia.
10) Caso de uso é encerrado.

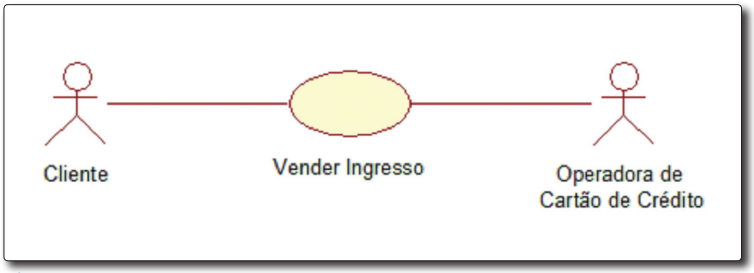


Figura 2. Caso de uso com mais de um ator interagindo

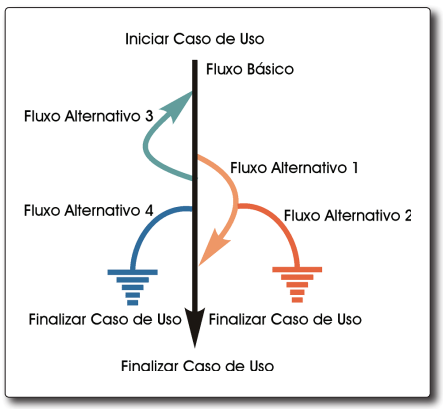


Figura 3. Fluxos de Eventos de um caso de uso

caso de uso que utilizamos em todos os projetos que participo e que foi retirado do Rational Unified Process.

Descrição

Nesta seção inserimos uma breve descrição de como o caso de uso funciona, deixando de forma clara e coesa qual o seu objetivo de forma que não seja preciso ler todo o caso de uso para entender seu objetivo.

Atores

Nesta seção colocamos os atores que iniciam e interagem com o caso de uso. Lembre-se que explicamos na primeira parte do artigo que podemos ter mais de um ator associado a casos de uso. Para o caso de existir mais de um ator no caso de uso, devemos criar duas subseções chamadas de: Ator Principal para o ator que inicia o caso de uso e Ator(es) Secundário(s) para os demais atores (aqueles que recebem ou disponibilizam informações para o caso de uso poder executar seu processo de solução - veja exemplo na Figura 2).

Pré-Condição

Esta seção é de grande importância no caso de uso, pois aqui colocaremos a pré-condição para que o caso de uso seja executado. A pré-condição é bastante utilizada no desenvolvimento de sistemas em métodos, funções e serviços de interfaces. A pré-condição é um pré-requisito para que a funcionalidade que for chamada possa ser executada sem nenhum problema e possa atender ao seu propósito.

Como exemplo, vamos imaginar um método de uma classe calculadora que recebe dois números e retorna o resultado da divisão do primeiro número pelo segundo. Vamos dizer que a pré-condição desse método seja que o segundo número não pode ter valor zero para não ter divisão por zero. O que acontecerá é que o desenvolvedor antes de fazer a chamada ao método deverá validar se o segundo número é maior que zero e caso não seja, deverá fazer o tratamento adequado. Com isso o método passou a ser mais coeso e não precisa ficar fazendo críticas que não lhes dizem respeito.

Então, como aplicaremos a pré-condição no caso de uso, já que não existem métodos nele? De maneira geral a pré-condição de um caso de uso funciona do mesmo jeito que uma pré-condição de um método, ou seja, é um pré-requisito. Mas a pré-condição do caso de uso está inserida em um contexto mais amplo no negócio do cliente. Por exemplo, vamos pegar o caso de uso Depositar Dinheiro, onde o ator Cliente quer fazer um depósito em uma conta que pode ser dele ou não. Como pré-condição, poderíamos colocar o seguinte: o Cliente que receberá o depósito (cliente destino) deverá estar definido (criado) no sistema e deverá existir uma conta aberta associada ao cliente em questão na agência que for informada.

Perceba que a pré-condição do caso de uso está ligada diretamente ao negócio do cliente, ao contrário da pré-condição do método que está ligada diretamente ao código. A pré-condição também nos indica a responsabilidade que cada caso de uso tem em sua execução, pois como no exemplo acima, percebemos que a criação de um cliente e a abertura de uma conta e sua associação com o cliente é feito por outro caso de uso ou outros casos de uso. Podem existir pré-condições para cada fluxo de evento ou apenas uma para o caso de uso inteiro.

Fluxo de Eventos

Esta é uma das seções principais do caso de uso. É onde descrevemos os passos entre o ator e o sistema. Cada fluxo de evento representa um cenário dentro do contexto de negócio no caso de uso e para cada fluxo de evento existirão os passos entre o ator e o sistema. Em um caso de uso existirá pelo menos um fluxo de evento que é chamado de fluxo de eventos básico (ou principal) e os demais fluxos caso venham a existir são chamados de fluxo de eventos alternativo (Figura 3).

A criação dos fluxos de eventos nos casos de uso deve levar em consideração algumas regras:

- Todo cenário deve ter uma pequena descrição para que possamos entender em que parte da solução do negócio o caso de uso está executando;
- Fluxo básico é o cenário que mais acontece no caso de uso e/ou o mais importante. Na verdade, quem decide qual cenário é o fluxo básico é o cliente juntamente com os Analistas de Sistemas;
- Fluxo alternativo é o caminho alternativo tomado pelo caso de uso a partir do fluxo básico, ou seja, dada uma condição de negócio o caso de uso seguirá por outro cenário que não o básico caso essa condição seja verdadeira. É muito importante ficar atento na hora de criar fluxos alternativos, pois não é qualquer condição que irá se transformar em fluxo alternativo. Como mencionei, o fluxo alternativo existe para uma condição de negócio que tenha importância e sentido para o negócio. Veja a seguir dois exemplos, sendo um correto e outro errado;
- Todo fluxo alternativo deve dizer qual a condição que será testada para o fluxo ser executado;
- Fluxos alternativos podem conter outros fluxos alternativos que são chamados de sub-fluxos.

Exemplo 1:
Fluxo de Evento Básico: Abertura de Conta não conjunta para um novo cliente.
Fluxo de Evento Alternativo: Abertura de Conta não conjunta para cliente já existente.

Exemplo 2:
Fluxo de Evento Básico: Abertura de Conta não conjunta para um novo cliente.
Fluxo de Evento Alternativo: Não informado Tipo da Conta

Os dois exemplos são do caso de uso Abrir Conta e cada um deles possui um fluxo de evento alternativo. O primeiro exemplo está correto, pois o fluxo alternativo está ligado diretamente ao negócio do cliente, ou seja, uma conta pode ser aberta para um novo cliente ou para um cliente já existente. Com isso serão adicionados novos passos e/ou poderá ter passos removidos entre o ator e o sistema nesse cenário alternativo.

Já no segundo exemplo, o fluxo alternativo está errado, pois ele tem uma condição não ligada ao negócio e sim a uma crítica/validação de dados. Sabemos que na abertura de uma conta o tipo da conta é uma informação obrigatória (Corrente ou Poupança) e que o usuário do sistema deverá informá-lo, mas caso não seja informada não devemos criar um fluxo alternativo só para tratar da validação. Lembre-se que um caso de uso controla os passos entre o ator e o sistema e o máximo que irá acontecer em um caso como esse do exemplo 2 é emitir uma mensagem para o usuário que a informação é obrigatória.

Validações de obrigatoriedade são feitas na seção de Exceções das regras do caso de uso, que está mais adiante. Veja no Quadro 4 como ficam os fluxos de eventos para esse caso de uso.

Vamos entender alguns pontos importantes nos fluxos de eventos do Quadro 4:

- No fluxo de evento básico temos o passo 6, onde é feita a chamada ao caso de uso Definir Cliente. Nesse momento todos os passos do caso de uso Definir Cliente serão executados e após seu término a execução do caso de uso Abrir Conta continuará. Mas sabemos que um caso de uso pode ter vários fluxos e no passo em que um caso de uso é executado, podemos indicar qual o

Quadro 4. Fluxo de Eventos do caso de uso Abrir Conta

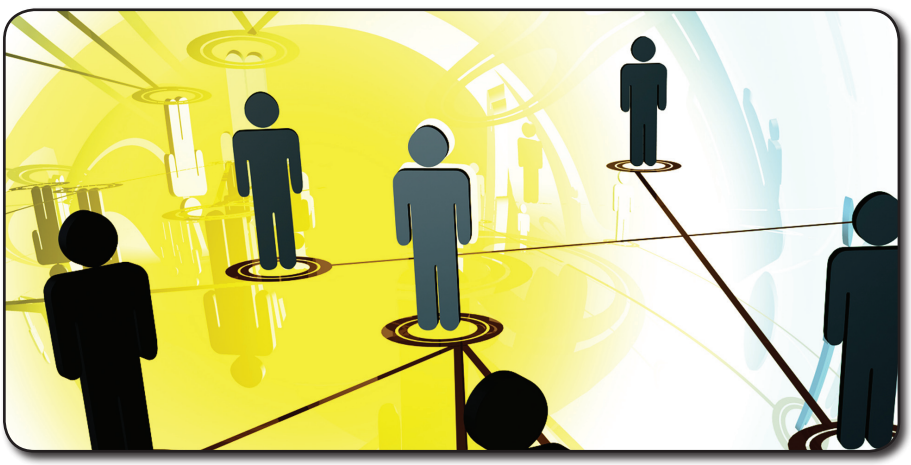
Caso de Uso: Abrir Conta
Ator Primário: Gerente
Ator(es) Secundário(s): Cliente

Fluxo de Evento Básico: Abertura de conta não conjunta para um novo cliente.
1. Gerente solicita abertura de conta.
2. Sistema solicita as seguintes informações: tipo da conta e se conta é conjunta.
3. Gerente informa os dados.
4. Sistema solicita o cpf do cliente.
5. Gerente informa o cpf.
6. Caso de uso Definir Cliente é executado.
7. Sistema solicita se cliente usará cheque.
8. Gerente informa os dados.
9. Gerente solicita confirmação da abertura de conta.
10. Sistema solicita a senha da conta.
11. Cliente informa a senha.
12. Sistema apresenta o número da conta.
13. Gerente solicita re-confirmação da abertura de conta.
14. Sistema emite mensagem M0.
15. Caso de uso é encerrado.

Fluxo de Evento Alternativo 1: Abertura de conta não conjunta para cliente já existente
[Condição: Cliente que está abrindo a conta já está definido no sistema e conta não é conjunta]
1. Após o passo 5 do fluxo de evento básico, o sistema apresenta os dados do cliente.
2. Gerente confirma que é o cliente correto.
3. Retorna para o passo 7 do fluxo de evento básico.

Fluxo de Evento Alternativo 2: Abertura de conta conjunta para novos clientes
[Condição: Clientes não estão definidos no sistema e estão abrindo uma conta em conjunto]
1. Após o passo 6 do fluxo de evento básico, o sistema solicita o cpf do segundo cliente.
2. O Gerente informa o cpf.
3. Caso de uso Definir Cliente é executado.
4. Retorna para o passo 7 do fluxo de evento básico.

Sub-fluxo Alternativo 1.1: Abertura de conta conjunta para clientes já existentes
[Condição: Clientes que estão abrindo a conta em conjunto já estão definidos no sistema]
1. Após o passo 2 do fluxo alternativo 1, o sistema solicita o cpf do segundo cliente.
2. Gerente informa o cpf.
3. Sistema apresenta os dados do segundo cliente.
4. Gerente confirma que é o cliente correto.
5. Retorna para o passo 3 do fluxo alternativo 1.



fluxo de evento será executado. Se não indicarmos nada, será o fluxo de evento básico por padrão;

- No passo 14 é dito que o sistema emite a mensagem M0. Colocar um código de mensagem ao invés da própria mensagem é uma boa prática, pois poderemos reutilizar a mesma mensagem em várias partes do caso de uso. As mensagens ficam na seção de mensagens;
- O passo 1 do fluxo alternativo 1 é iniciado com a frase "Após o passo 5 do fluxo de evento básico" e no passo 3 é retornado para o passo 7 do fluxo de evento básico. Isso significa que após o caso de uso executar o passo 5 do fluxo básico, a execução será direcionada para o fluxo alternativo e ao final voltará para o passo 7, deixando de executar o passo 6 do fluxo básico. Um fluxo alternativo pode ser iniciado substituindo um passo do fluxo básico ou de um fluxo alternativo, onde nesse caso colocamos a frase "no passo x do fluxo de evento" e pode ser encerrado de duas maneiras. A primeira é retornando para algum passo do fluxo que o chamou e a segunda é encerrando o caso de uso;
- Perceba que no fluxo alternativo 1 existem apenas três passos e no artigo anterior explicamos que casos de uso com poucos passos podem ter o problema de não serem casos de uso por tenderem a não mostrar a interação entre o ator e o sistema de forma que possamos entender o processo de negócio. Também explicamos que sempre a ponderação do que é importante para o negócio e para o sistema deve predominar nesse momento. É o que acontece neste exemplo, pois no fluxo alternativo 1 existe o passo do sistema retornando dados do cliente existente. Esse é um passo de extrema importância para o negócio do cliente e existe o passo onde o gerente confirma ser o cliente correto, ou seja, os dados informados pelo sistema são do mesmo cliente que está abrindo a conta;
- Outro detalhe importante é que no passo 3 do fluxo alternativo 2 o caso de uso Definir Cliente também é executado. Na figura 1 vemos que o caso de uso é chamado por um include, pois ele sempre será executado a partir do fluxo de evento básico. Mas em certas situações como neste exemplo, podemos chamar um caso de uso que está como include no diagrama de caso de uso dentro de um fluxo alternativo;
- E por final, existem passos repetidos em alguns fluxos. Neste caso podemos tomar duas decisões. A primeira é criar um caso de uso com esses passos e fazer um include ou um extend dependendo de qual fluxo está fazendo a chamada. Mas, neste caso, devemos sempre verificar se realmente vale a pena criar mais um caso de uso só para reusar certos passos ou se dentro do contexto do negócio esses passos são de grande importância. O outro caminho e que foi escolhido neste exemplo é simplesmente repetirmos os passos dentro dos fluxos, pois esses passos são apenas passos simples de interação entre o sistema e o ator.

Pós-Condição

Diferente da pré-condição que é o pré-requisito para o caso de uso ser executado, a pós-condição é a garantia de que tudo ocorrerá sem nenhum problema caso a pré-condição seja respeitada. Com isso, o resultado esperado do caso de uso será obtido. A pós-condição de um caso de uso também está ligada diretamente ao negócio do cliente e não ao código como uma pós-condição de método. Mas para uma pós-condição ser considerada válida, devemos nos atentar para as seguintes considerações:

- Deverá existir uma pós-condição para cada fluxo de evento do caso de uso, pois o caso de uso pode terminar de várias formas (Figura 4);
- É uma ótima prática escrever a pós-condição sempre no passado. Por exemplo, podemos colocar o seguinte texto como pós-condição do fluxo de evento básico do caso de uso Abrir Conta: Uma nova conta foi aberta e associada ao novo cliente que foi definido;
- É muito importante considerar todos os objetos de negócio criados, removidos, alterados e relacionados com outros objetos. Isso representa uma fotografia do estado que o sistema ficou depois da execução do caso de uso;
- Quando usamos pós-condições junto com relacionamentos de extensão, deve-se tomar muito cuidado para que o caso de uso que estende não introduza um subfluxo que viole a pós-condição no caso de uso base.

Regras

A seção de regras também é uma das mais importantes seções existentes no caso de uso. Nesta seção descrevemos todas as regras funcionais que o caso de uso deve cumprir durante sua execução. É uma boa prática que toda regra inserida nesta seção deva possuir um código para podermos criar um link com outras seções do caso de uso que venham a fazer menção a essa regra. Por exemplo: R1 – CPF é obrigatório.

Também é uma boa prática criarmos subseções para dividirmos as regras funcionais em tipos de regras. Veja o exemplo dos tipos de regras existentes:

- **RFDN – Regra Funcional do Domínio do Negócio** – São regras inerentes ao negócio do cliente, ou seja, são as principais regras que devem ser cumpridas para que o negócio do cliente seja atendido. As RFDN's são regras que devemos ter muito cuidado quando estamos fazendo a análise e projeto de um caso de uso, pois essas são regras corporativas que devem estar sempre sendo reaproveitadas em outras aplicações, evitando assim a duplicação das regras em termos de código no momento da implementação;
- **RFDA – Regra Funcional do Domínio da Aplicação** – São regras inerentes à aplicação que existem devido a uma imposição da aplicação, na maioria das vezes sendo uma imposição visual. São regras que caso não existissem, a aplicação em termos de negócio continuaria funcionando. Como exemplo dessa regra, posso citar um campo que será preenchido em função da informação de outro campo. Esse tipo de regra é estritamente visual, ou seja, a forma da tela funcionar impôs que essa regra existisse;

- **RFDP – Regra Funcional do Domínio do Processo** – São regras inerentes ao processo de negócio (workflow), ou seja, são regras que devem existir porque o processo impôs. Esses tipos de regras existem quando o nosso sistema está envolvido com gerenciamento de processo de negócio (BPM). Neste artigo não detalharei sobre essas regras, pois falar sobre processo de negócio é muito extenso e exigiria um artigo só para isso.

Fluxos de Exceção

Nesta seção, colocamos todas as exceções das regras do caso de uso da seção Regras e as ações que se deve tomar em caso das regras não serem respeitadas. Veja no Quadro 5 um exemplo de exceção de regras.

No Quadro 5 vemos que as exceções também possuem um código que facilita muito sua identificação dentro do caso de uso. No exemplo, as ações tomadas para as exceções são que o sistema emitirá uma mensagem. Como já foi explicado, as mensagens são referenciadas através de um código para reaproveitamento das mesmas e em caso de alteração do texto da mensagem não precisarmos alterá-las por todo o caso de uso. Mas podem existir situações em que ao invés do sistema emitir uma mensagem, um caso de uso ser executado quando uma regra não for satisfeita.

Mensagens do Sistema

Nesta seção colocamos as descrições de todas as mensagens que o sistema deve emitir para o usuário. Veja exemplo:

- M0 – Abertura de conta efetuada com sucesso;
- M1 – O campo CPF é obrigatório;
- M2 – O campo Tipo da conta é obrigatório;
- M3 – A senha é obrigatória;
- M4 – Cliente já possui conta nesta agência do mesmo tipo informado;

Requisitos Especiais

Nesta seção colocamos todos os requisitos não funcionais que o caso de uso possui. Em alguns casos, podemos nos deparar com situações que o sistema deve tratar regras não funcionais, e que são de muita importância de modo que sua documentação se faz necessária. Esses requisitos não só servem para dizer que o sistema deve tratar essas restrições, mas eles são de extrema importância para o Arquiteto de Software, pois esses requisitos ajudam a escolher os casos de uso arquiteturalmente mais significantes para a construção da arquitetura do sistema que formará toda a base de desenvolvimento. Alguns exemplos de requisitos especiais para os casos de uso são:

- Performance;
- Padrões visuais do sistema;
- Tratamento de logs.

Pontos de Extensão

Nesta seção, colocamos todos os casos de uso que estendem o caso de uso base e em quais pontos eles são chamados dentro dos fluxos de eventos. Além de termos uma listagem dos casos de uso chamados e o ponto em que eles são chamados, essa seção também serve para o caso de omitirmos a chamada do caso de uso de extensão dentro dos passos dos fluxos de eventos do caso de uso e fazermos esse link nesta seção. Mas não acho muito prático colocar o link somente nesta seção, pois a leitura dos passos do caso de uso pode ficar confusa. O ideal é mantermos a chamada aos casos de uso de extensão nos próprios passos e nesta seção colocar o nome dos casos de uso como um link para a

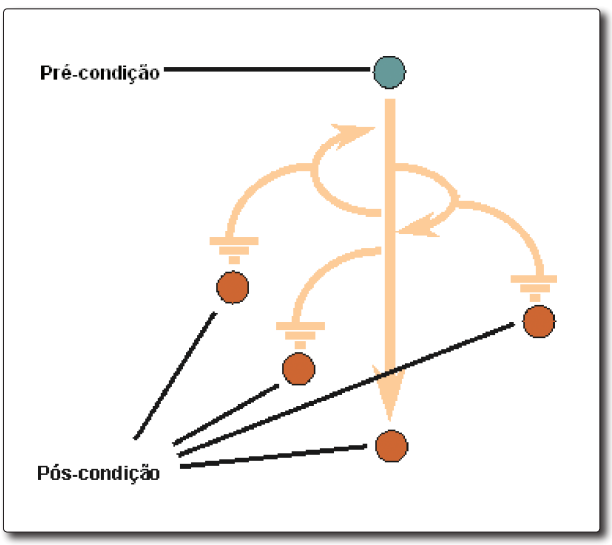


Figura 4. Pós-condições para cada fluxo de evento de um caso de uso

Quadro 5. Fluxo de Exceções do caso de uso Abrir Conta

Regras

R1 – CPF é obrigatório.
R2 – Tipo da conta é obrigatório.
R3 – Senha é obrigatória.
R4 – Caso o cliente já possua uma conta do mesmo tipo na agência, não poderá abri-la.

Fluxo de Exceções

E1 – CPF não foi informado.
Regra: R1
Ação: Sistema emite mensagem M1.

E2 – Tipo da conta não foi informado.
Regra: R2
Ação: Sistema emite mensagem M2.

E3 – Senha não foi informada.
Regra: R3
Ação: Sistema emite mensagem M3.

E4 – Cliente solicita abertura de conta de mesmo tipo que ele já possui na agência.
Regra: R4
Ação: Sistema emite mensagem M4.

especificação dos mesmos e informar o ponto que eles são chamados. Veja o exemplo do caso de uso Abrir Conta no Quadro 6.

Diagramas de Caso de Uso

Nesta seção colocamos a parte do diagrama de caso de uso que faz parte do escopo do caso de uso que estamos especificando. Com isso, tanto a equipe de desenvolvimento quanto o cliente que estiver lendo ou validando o caso de uso poderá ter uma visão mais abstrata de como funciona o caso de uso, seus atores envolvidos e os demais casos de uso envolvidos. Veja na Figura 5 a parte do diagrama de caso de uso do caso de uso Abrir Conta.

Outros Diagramas

Nesta seção colocamos outros diagramas da UML, caso seja necessário para o maior entendimento do caso de uso. Na prática colocamos aqui um diagrama de atividade que serve para entendermos a parte do processo de negócio que o caso de uso controla. O diagrama de atividade facilita o entendimento quando o caso de uso tem muitos fluxos alternativos. Com o diagrama de atividade podemos ver de forma visual todas as ramificações que o caso de uso possui. Veja na Figura 6 o diagrama de atividade do caso de uso Abrir Conta.

Quadro 6. Ponto de Extensão do caso de uso Abrir Conta

Caso de uso: Definir Cliente
Ponto de Extensão:
Fluxo de Evento: Básico
Passo: 6

Fluxo de Evento: Alternativo 2
Passo: 3

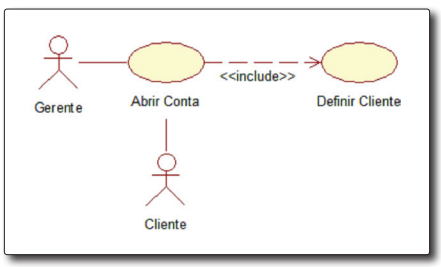


Figura 5. Diagrama do caso de uso Abrir Conta

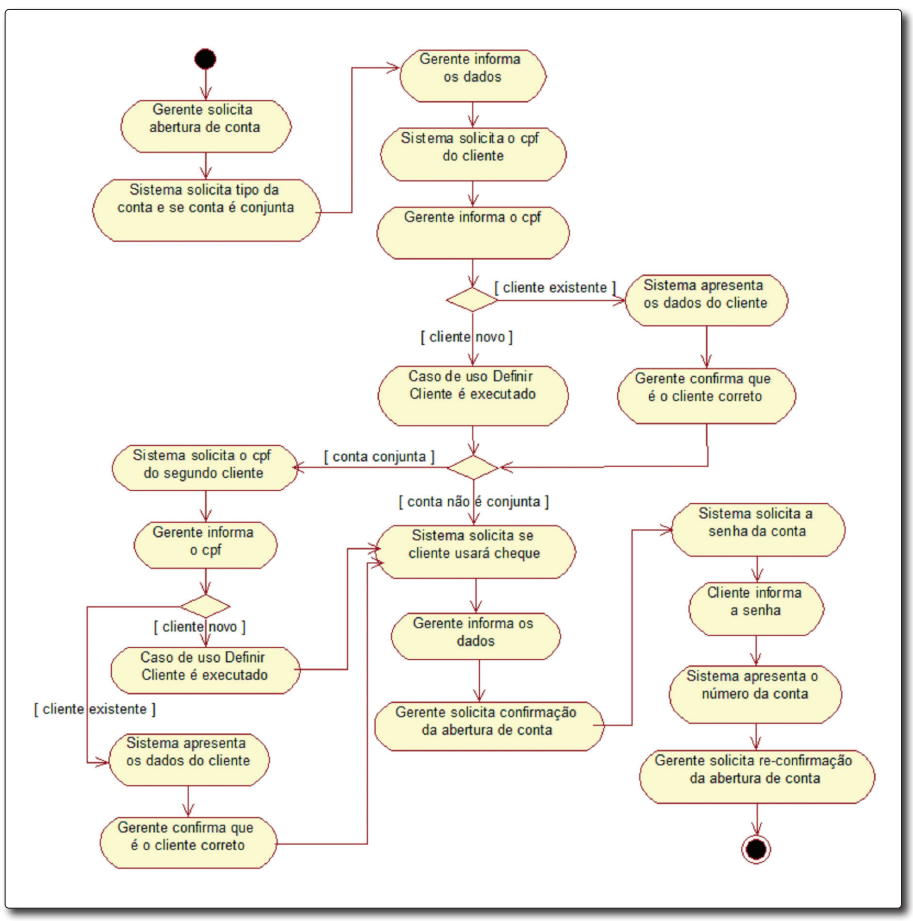


Figura 6. Diagrama de atividade do caso de uso Abrir Conta

Especificação Suplementar

Como foi explicado, na seção de regras do caso de uso colocamos as regras funcionais que devem ser validadas na execução do caso de uso. E na seção de requisitos especiais colocamos as regras não-funcionais referentes apenas ao caso de uso em questão. Mas existem regras não-funcionais que pertencem a mais de um caso de uso. Quando isso ocorre, nós colocamos essas regras em um documento separado do caso de uso, chamado de Especificação Suplementar. Como o nome já diz, a especificação suplementar visa ser um suplemento para os casos de uso com regras que não diz respeito ao negócio do cliente e sim ao sistema em si. Na Tabela 1 estão os tipos de especificações que podem existir dentro da especificação suplementar.

Conclusão

Nesta segunda parte vimos como especificar um caso de uso e as boas práticas de como criarmos essa especificação. Também foi mostrada a especificação suplementar que é um complemento para as regras dos casos de uso com as regras não funcionais. Com uma boa especificação, o caso de uso além de ser entendido pelo cliente que validará se os requisitos foram capturados e entendidos, também será entendido por toda equipe de desenvolvimento, desde os analistas, projetistas e implementadores até os gerentes de projetos. Com o caso de uso é possível fazer uma análise dos requisitos e um projeto físico de forma mais eficaz e termos um projeto centrado em arquitetura que formará a base de todo o desenvolvimento. Na próxima parte do artigo falaremos sobre os casos de uso de negócios. ●

Links

OMG: The Object Management Group
www.omg.org

RUP: Rational Unified Process 7.0
http://www-306.ibm.com/software/awdtools/rup/

Bibliografia

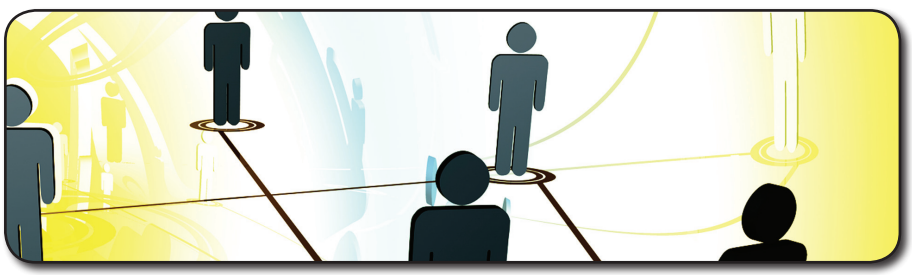
UML Essencial – 3ª Edição – Martin Fowler

Utilizando UML e Padrões – Larman, Craig

Princípios de Análise e Projeto de Sistemas com UML - 2ª Edição – Eduardo Bezerra

Especificação	Descrição
Funcionalidade	Especificação referente às funcionalidades do sistema, ou seja, regras não-funcionais que o sistema deve tratar em mais de um caso de uso. Exemplo: O sistema não pode permitir data futura.
Usabilidade	Especificação sobre a usabilidade do sistema, como teclas de atalho, navegabilidade das telas e etc.
Confiabilidade	Especificação que delimita a disponibilidade do sistema em porcentagem de horas para seu uso, a taxa máxima de erros ou defeitos do sistema, a taxa de erros ou defeitos e outros itens para demonstrar a confiabilidade do sistema.
Desempenho	Especificação que delimita os tempos de respostas que o sistema deve ter em determinados processos como, o tempo de resposta para o início de uma impressão do saldo de uma conta corrente.
Manutenibilidade	Especificação que indica todos os requisitos que aprimorarão a manutenibilidade do sistema que está sendo criado, incluindo padrões de codificação, convenções de nomeação, bibliotecas de classes, acesso à manutenção e utilitários de manutenção. Como diz respeito a todos os requisitos do sistema necessários para suportar o aplicativo, poderão estar incluídos as plataformas de rede e os sistemas operacionais suportados, configurações, memória, periféricos e software fornecido. Na prática fazemos um link para o documento de arquitetura de software.
Restrições de Design	Esta especificação deve indicar todas as restrições de design referentes ao sistema que está sendo criado. As restrições de design representam decisões de design que foram impostas e devem ser obedecidas. Entre os exemplos desse tipo de restrição estão linguagens de programação, requisitos de processo de software, uso prescrito de ferramentas de desenvolvimento, restrições de design e de arquitetura, componentes comprados, bibliotecas de classes, etc. Na prática fazemos um link para o documento de arquitetura de software.
Restrições Técnicas	Esta especificação deve indicar todas as restrições técnicas referentes ao sistema que está sendo criado. As restrições técnicas representam decisões de ferramenta e tecnologias que foram impostas e devem ser obedecidas. Entre os exemplos desse tipo de restrição estão as definições de uso de DBMS Oracle 8i, Servidor de Aplicação Oracle 9ias v.9.0.3, interface WEB, etc.
Requisitos de Sistema de Ajuda e Documentação	Especificação que indica se o sistema tem manual do usuário e em qual formato está e se possui help on-line e como será ativado dentro do sistema.
Componentes Adquiridos	Esta especificação descreve todos os componentes adquiridos a serem usados no sistema, restrições de utilização ou de licenciamentos aplicáveis e quaisquer padrões associados de compatibilidade/interoperabilidade ou de interface.
Interfaces	Esta especificação define as interfaces que devem ser suportadas pelo aplicativo. Ela deve conter especificidades, protocolos, portas e endereços lógicos adequados, entre outros, para que o software possa ser desenvolvido e verificado em relação aos requisitos de interface. Por exemplo: interfaces do usuário, de hardware e etc.
Requisitos de Licenciamento	Esta especificação define todos os requisitos de imposição de licenciamento ou outros requisitos de restrição de utilização que devem ser exibidos pelo software.
Observações Legais, de Copyright e Outras	Esta seção descreve todos os avisos legais necessários, garantias, observações sobre direitos autorais, observações sobre patentes, logomarcas, marcas comerciais ou problemas de conformidade com logotipos referentes ao software.
Padrões Aplicáveis	Esta seção descreve, por meio de referências, todos os padrões aplicáveis e as seções específicas desses padrões que se aplicam ao sistema que está sendo descrito. Entre esses padrões estão incluídos, por exemplo, padrões legais, de qualidade e reguladores, padrões de indústria referentes à usabilidade, interoperabilidade, internacionalização, compatibilidade com o sistema operacional, etc.
Requisitos de Manutenção de Banco de Dados	Esta seção descreve requisitos que definem qual a política de backup e limpeza da base de dados, bem como as responsabilidades sobre essas atividades. Deixar claro que atividades ficam a cargo do sistema e quais devem ser de alguma forma realizada por não fazerem parte do escopo do sistema. Especificar também, para cada atividade relacionada a backup e limpeza da base de dados, qual a sua periodicidade.
Requisitos Ambientais	Esta seção descreve os requisitos ambientais, conforme necessário. Para sistemas baseados em hardware, as questões ambientais poderão incluir temperatura, choques, umidade, radiação, etc. Para aplicativos de software, os fatores ambientais podem incluir condições de uso, ambiente do usuário, disponibilidade de características, problemas de manutenção e recuperação e tratamento de erros.

Tabela 1. Tipos de especificações do documento Especificação Suplementar



Plano de Projeto

Um 'Mapa' Essencial à Gestão de Projetos de Software

Antonio Mendes da Silva Filho
antoniom.silvafilho@gmail.com
Professor e consultor em área de tecnologia da informação e comunicação com mais de 20 anos de experiência profissional, é autor dos livros Arquitetura de Software e Programando com XML, ambos pela Editora Campus/Elsevier, tem mais de 30 artigos publicados em eventos nacionais e internacionais, colunista para Ciência e Tecnologia pela Revista Espaço Acadêmico com mais de 60 artigos publicados, tendo feito palestras em eventos nacionais e no exterior. Foi Professor Visitante da University of Texas at Dallas e da University of Ottawa. Formado em Engenharia Elétrica pela Universidade de Pernambuco, com Mestrado em Engenharia Elétrica pela Universidade Federal da Paraíba (Campina Grande), Mestrado em Engenharia da Computação pela University of Waterloo e Doutor em Ciência da Computação pela Univesidade Federal de Pernambuco.

Imagine você desejar sair da cidade onde você reside e ir até Maringá (localizada na região noroeste do estado do Paraná), supondo obviamente você não residir lá. Sem um mapa, um plano ou qualquer outra fonte de informação para saber por quais cidades, menor percurso, melhores estradas, dentre outras informações, você não terá certeza de alcançar o seu objetivo (i.e. chegar em Maringá).

Agora, trazendo esta meta para o contexto de um projeto de software, você necessitará de um mapa de quais atividades devem ser realizadas, sem o qual você ficará perdido. Aqui, também, um plano torna-se essencial para compreender riscos, compromissos e decisões de projeto.

Precisamos de um 'mapa' ou 'guia' que ofereça uma base sistemática de como conduzir o projeto e quaisquer modificações necessárias além de servir como eficiente mecanismo para comunicação entre os principais interessados no projeto (isto é stakeholders) que inclui cliente, usuário final, gerente projeto, dentre outros. Esse 'mapa' existe e é conhecido como plano de projeto. Trata-se de um dos documentos produzidos durante a realização de projeto. O plano de projeto é essencial e determinante no sucesso para uma boa condução de qualquer projeto.

A gestão de projetos define quem, o que, quando e o porquê dos projetos. Ela faz uso de processos e ferramentas de gestão os quais servem para ajudar o gerente de projetos e equipe a organizar, documentar, rastrear e relatar as atividades e progresso de um projeto. Dentro desse contexto, o plano de projeto compreende:

- Escopo de projeto bem definido;
- Um roadmap dos artefatos a serem entregues;
- Documentação de papéis e responsabilidades dos participantes;
- Uma linguagem 'comum' para comunicação das atividades do projeto, bem como a rastreabilidade e relatórios dessas atividades;
- Mecanismos de resolução de conflitos e mitigação ou atenuação de riscos.

Você pode está se questionando porque tudo isso é necessário. E a resposta para tal indagação vem da necessidade de gerenciar adequadamente os recursos (geralmente, restritos) existentes, além dos custos, tempo e qualidade, a fim de atingir os objetivos do projeto. De um modo geral, podemos entender o modelo de desenvolvimento de um projeto como ilustrado na Figura 1.

Perceba a necessidade de uma abordagem sistemática e consistente para conduzir o projeto. Isto é conseguido com a elaboração de um plano de projeto e seu uso ao longo de todo o projeto. O plano de projeto é essencial para o sucesso de um projeto e o gerente de projeto não se separa dele até o encerramento do mesmo.

Plano de Projeto

O plano de projeto é um dos documentos produzidos na condução de um projeto. Ele funciona como:

- Um 'integrador' entre diversas ações do projeto;
- Mecanismo de comunicação para os stakeholders (isto é, as partes interessadas do projeto);
- Captura e documenta a evolução do projeto à medida que ele vai sendo executado e novas informações vão sendo disponibilizadas.

A gerência da execução do plano de projeto tem o objetivo de realizar o trabalho definido na descrição do escopo do projeto. Durante a execução do plano de projeto, o gerente de projeto se apóia nesse documento para tomar ações corretivas visando alcançar o conjunto de metas planejadas em concordância com o que foi definido no plano. Nesse sentido, o plano de projeto deve conter:

- Como os processos de gerência serão utilizados;
- Como as mudanças serão monitoradas e controladas;
- Milestones com datas de pontos estratégicos para avaliação do projeto;
- Baselines para cronograma, custo e qualidade;
- Calendário para recursos utilizados;
- Mecanismos de comunicação para os stakeholders;
- Definição de revisões para resolução de pontos em aberto e/ou pendentes;
- Planos de outras áreas de conhecimento (como, comunicação e qualidade).

Itens de um Plano de Projeto	Conteúdo
1. Introdução	Contém uma descrição dos objetivos do documento, o público ao qual ele se destina e em linhas gerais o propósito do projeto a ser desenvolvido. Pode adicionalmente conter termos e abreviações usadas, além de informar como o plano deve evoluir.
2. Escopo do projeto	Esta seção descreve em linhas gerais o projeto a ser desenvolvido, comunicando o propósito do mesmo, e a importância do projeto para todas as partes envolvidas. O escopo do projeto que será executado é apresentado com uma descrição dos requisitos técnicos (isto é, os requisitos do produto a ser desenvolvido) que podem ser funcionais, não funcionais (desempenho, usabilidade, portabilidade, confiabilidade, etc.) e tecnológicos (tecnologia a ser utilizada). Também, apresentam-se requisitos não técnicos (como, por exemplo, treinamento) e o escopo não contemplado (que descreve quais funcionalidades não fazem parte do escopo do projeto).
3. Organização do projeto	Apresenta-se uma descrição da estrutura organizacional do projeto, incluindo organograma e a definição de papéis e responsabilidades.
4. Equipe e infra-estrutura	Contém descrição da equipe e da infra-estrutura utilizada para o desenvolvimento do projeto, incluindo: pessoal, equipamentos, ferramentas, software de apoio, materiais, dentre outros. Isto visa garantir uma estrutura adequada para a execução das atividades previstas no plano. Nesta seção também é apresentada o planejamento da alocação de pessoal no projeto.
5. Acompanhamento do projeto	Esta seção do plano de projeto relaciona os momentos para realização das atividades de verificação do projeto, as quais poderão ser feitas pela equipe técnica das instituições envolvidas (desenvolvedora e cliente), e também a forma como estas atividades serão realizadas. Estas atividades incluem a realização de reuniões e geração de relatórios descrevendo informações sobre o progresso do projeto.
6. Marcos do projeto	Contém uma descrição de marcos (milestones) importantes do projeto (incluindo as datas de início e fim do projeto), bem como os artefatos que serão entregues pela empresa desenvolvedora nestes marcos, quando aplicável. Apenas marcos relevantes devem ser listados, ou seja, aqueles que contribuirão para a medição do desempenho do projeto. Por exemplo: reuniões de revisão, apresentação de protótipos ou realização de testes de aceitação. Note que é possível inserir uma visão do cronograma do projeto neste item, destacando apenas os marcos importantes e suas datas alvo.
7. Gerência de riscos	Os riscos identificados para o projeto estão detalhados e monitorados nos relatórios de progresso. Exemplos de riscos compreendem: risco de pessoal, risco tecnológico e de escopo, dentre outros. Um caso de risco de escopo é a falta de clareza na definição do escopo de projeto, que pode resultar em inúmeras solicitações de mudança de escopo.
8. Qualidade do produto (ou sistema)	Informa-se a metodologia de desenvolvimento adotada no projeto. Caso, por exemplo, alguma ferramenta específica de desenvolvimento venha a ser utilizada no projeto, isso deve ser descrito neste item. Adicionalmente, informam-se como os artefatos serão gerados por este projeto, os padrões adotados, formatos dos arquivos e templates a serem empregados. Também, neste item, costuma-se informar os critérios de aceitação do projeto.
9. Testes do produto (ou sistema)	Este item apresenta uma descrição do projeto de testes do projeto, incluindo detalhamento da estratégia de implementação dos testes, com estágios e tipos de testes a serem realizados para garantir a conformidade do produto com as especificações de requisitos funcionais, não funcionais e requisitos de aceitação do projeto.
10. Referências	Apresenta-se uma relação dos documentos pertinentes ao projeto.

Tabela 1 – Relação de itens de um plano de projeto.

É importante perceber a importância do plano de projeto como determinante para o sucesso de um projeto. Ele identifica quais artefatos deverão ser entregues e quando e, igualmente importante, informa os recursos necessários para realizar as entregas (de artefatos) indicando as dependências existentes para essas entregas. A seção seguinte apresenta um exemplo de um plano de projeto ilustrando e complementando os pontos destacados.

Exemplificando o Plano de Projeto

O plano de projeto contém um conjunto de informações que permite o gerente de projeto não apenas executar o projeto, mas também monitorar seu progresso e verificar se o executado está em conformidade com o planejado. A Tabela 1 apresenta uma relação dos itens considerados imprescindíveis em um plano de projeto. A relação de itens destacados na Tabela 1 não pressupõe a intenção de ser completo, mas de apontar os itens considerados como obrigatórios num plano de projeto de empresa.

O conteúdo exato das seções que compõem um plano de projeto, geralmente, difere de empresa para empresa. Entretanto, os itens apontados na Tabela 1 normalmente compõem as seções do documento de plano de projeto. As subseções, destacadas nos Quadros 1 a 12, ilustram o conteúdo que compõe um plano de projeto.

Note que a Tabela 2 identifica um subconjunto de termos que pode caracterizar um projeto. Poderíamos, por exemplo, adicionar o termo MS para se referir a um produto ou solução da Microsoft. Todo e qualquer termo, convenção adotada ou abreviações deveriam ser apresentadas nesta tabela a fim de comunicar às partes envolvidas e interessadas (i.e. os stakeholders) o seu significado. Isto visa prover os stakeholders com as denominações corretas empregadas no projeto.

A seção seguinte apresenta uma visão geral do projeto trazendo objetivos, participantes e mecanismos de evolução do plano de projeto e aceitação. Isto é exemplificado no Quadro 2.

Quadro 1

1. Introdução

Este documento apresenta o planejamento do projeto do sistema Exemplo o qual será utilizado como base às atividades de acompanhamento, revisão, verificação e validação do projeto desde seu início até sua conclusão, a fim de garantir a análise comparativa do desempenho real versus planejado. Desta forma, ações corretivas e preventivas poderão ser tomadas, sempre que resultados ou desempenhos reais desviarem significativamente do planejado. Sua elaboração é derivada das informações contidas no Plano de Trabalho e convênio assinado com o cliente.

1.1 Termos e acrônimos

Esta seção explica o conceito de um subconjunto de termos importantes que serão mencionados no decorrer deste documento. Estes termos são descritos na Tabela 2, estando apresentados por ordem alfabética.

Termo	Descrição
Artefato	Tudo que é produzido e documentado em qualquer atividade de qualquer fluxo do projeto. Por exemplo: documento de requisitos, diagrama de casos de usos e glossário.
Milestone	Ponto de checagem; marco que indica a conclusão de uma fase ou etapa.
NA	Não Aplicável
Patrocinador	Representante da empresa cliente ou contratada responsável pelo sucesso do projeto em instância superior, garantindo o cumprimento de responsabilidades estabelecidas.
Revisão	Apresentação de produtos de software para os interessados visando comentário e aprovação dos mesmos.
SQA	Software Quality Assurance, profissional ou grupo responsável por garantir a qualidade do produto de software e processo de desenvolvimento.

Tabela 2 – Termos e acrônimos do projeto.

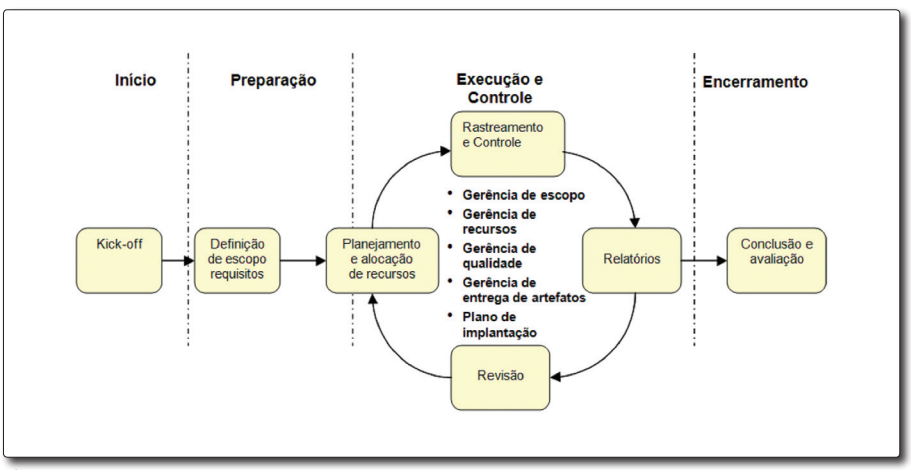


Figura 1. Perspectivas da gestão de projetos.

Quadro 2

2. Visão Geral do Projeto

Analisando-se os aspectos técnicos (métodos, processos e ferramentas) e não técnicos (gerenciamento, planejamento e questões econômicas) de produção de aplicativos de software baseados em componentes, este projeto propõe construir a infra-estrutura necessária para o desenvolvimento de componentes e aplicativos, fazendo uso da plataforma tecnológica orientada a serviços (web services).

Nesse sentido, os componentes orientados a serviço a serem desenvolvidos pela Empresa AM Ltda (desenvolvedora do projeto) servirão de base para construção de aplicações pelo cliente direcionadas à área de Turismo. Como resultado, isto permitirá elevar a produtividade e competitividade, promovendo a posição do cliente no mercado com uso de soluções tecnologicamente avançadas. Este projeto propõe ainda realizar pesquisa e desenvolvimento da infra-estrutura para o desenvolvimento de componentes e aplicativos orientado a serviços a serem usados pelo cliente.

2.1 Participantes

Esta seção lista o conjunto de participantes e parceiros envolvidos no desenvolvimento do projeto que serão mencionados no decorrer deste documento. Esta lista é apresentada por ordem alfabética.

- Empresa AM Ltda no papel de Executor
- Organização BrasilTur no papel de Cliente / Financiador

2.2 Objetivos Específicos

De acordo com o plano de trabalho assinado com o Cliente, os objetivos desse projeto compreendem:

- Análise de plataformas tecnológicas;
- Definição de um modelo de desenvolvimento de web services;
- Desenvolvimento de web services para monitoração e controle de acesso;
- Desenvolvimento de web services para controle de qualidade de serviço;
- Levantamento e avaliação de requisitos de componentes e aplicações de negócio para a área de Turismo;
- Desenvolvimento de protótipo: aplicação de web services para área de Turismo;

2.3 Critérios de Aceitação do Projeto

A aceitação final do projeto está condicionada a:

- Todos os artefatos e indicadores físicos de execução descritos na seção 9.1 devem ter sido aprovados pela Divisão de Qualidade da empresa;
- Todos os objetivos listados na seção 2.2 devem ter sido atingidos.

2.4 Mecanismos de Evolução do Plano de Projeto

O plano do projeto deve ser mantido atualizado para refletir a situação corrente do projeto. Dessa forma, as seguintes situações representam os gatilhos para atualização deste documento:

- Alterações de rubricas junto ao patrocinador (Cliente);
- Mudanças nos critérios de aceitação do projeto;
- Alterações dos objetivos previstos no Plano de Trabalho aprovado pelo cliente;
- Mudanças na gerência do projeto da empresa ou do cliente.

Perceba que os objetivos do projeto são peculiares a cada projeto. Além disso, os critérios de aceitação final do projeto é resultado de acordo entre as parte envolvidas (i.e. empresa desenvolvedora e cliente). O Quadro 3 caracteriza a seção dos requisitos do sistema.

A seção seguinte apresenta uma visão geral do projeto trazendo objetivos, participantes e mecanismos de evolução do plano de projeto e aceitação. Isto é exemplificado no Quadro 3.

A motivação da seção anterior é caracterizar as principais funcionalidades a serem implementadas (as quais são detalhadas no documento de requisitos), além de informar o que não faz parte do escopo do projeto. A próxima seção (Quadro 4) apresenta uma visão organizacional do projeto e os principais atores dessa estrutura.

O Quadro 5 destaca o quantitativo da equipe e sua respectiva alocação. Também, informações e específicas de cada um dos membros da equipe e possíveis ferramentas utilizadas no projeto são apresentadas. Os Quadros de 6 a 9 abordam, respectivamente, Treinamentos do Projeto, Acompanhamento do Projeto, Controle de Mudanças do Escopo de Projeto e Cronograma.

Quadro 3

3. Requisitos do Sistema

Esta seção apresenta os requisitos do sistema que servirão de base ao seu planejamento, bem como do escopo não contemplado (ou escopo negativo). Mudanças nestes requisitos devem ser submetidas ao controle de mudanças estabelecido para o projeto, descrito no plano de gerencia de configuração.

3.1 Requisitos Técnicos

Os requisitos a seguir representam uma visão dos produtos a serem desenvolvidos nesse projeto. Estes requisitos serão descritos em detalhes no Documento de Requisitos do Projeto, que será complementado e refinado no decorrer do ciclo de vida do projeto.

3.1.1 Requisitos Funcionais

Os requisitos funcionais considerados compreendem:

- A infra-estrutura de desenvolvimento de componentes e aplicativos será baseada na plataforma orientada a serviços (web services) e compreenderá:
- Web services de controle de acesso;
- Web services de controle de qualidade de serviço;
- Web services para área de Turismo;
- Desenvolvimento de web services para uma subárea de aplicação de Turismo, bem como disponibilizar seus respectivos componentes de serviços no repositório utilizado. A definição da subárea de Turismo será realizada na etapa inicial deste projeto.

3.1.2 Requisitos Não Funcionais

Os requisitos não funcionais considerados neste projeto compreendem:

- O modelo de desenvolvimento de web services deverá considerar uma infra-estrutura que permita o desenvolvimento rápido de componentes e aplicativos baseado na plataforma orientada a serviços. O conjunto de componentes orientados a serviços resultantes deste projeto servirá de base para a construção de várias aplicações para o setor de turismo.
- O processo de validação de web services se encerrará com a entrega da versão final dos web services.

3.4 Mecanismos de Evolução do Plano de Projeto

O plano do projeto deve ser mantido atualizado para refletir a situação corrente do projeto. Abaixo, é apresentado um conjunto de situações que requer a atualização deste documento:

- Alterações de rubricas junto ao cliente;
- Mudanças nos critérios de aceitação do projeto;
- Alterações dos objetivos previstos no Plano de Trabalho aprovado pelo cliente;
- Mudanças na gerência do projeto da empresa ou do cliente.

3.2 Requisitos Não Técnicos

Os requisitos não técnicos compreendem:

- Todo e qualquer material de divulgação resultante da execução deste projeto deverá conter informação do suporte financeiro do cliente e, especialmente, no caso de:
  - Seminários e eventos científicos e tecnológicos;
  - Publicações técnicas e científicas em revistas especializadas;
  - Relatórios técnicos publicados ou divulgados em qualquer mídia.

3.3 Escopo Não Contemplado

O escopo do projeto não contempla a realização das seguintes atividades:

- Elaborar e/ou realizar treinamento no uso dos web services desenvolvidos no projeto;
- Correção de bugs, identificados em qualquer um dos produtos, após a duração prevista do projeto perante o cliente.

Quaisquer outros artefatos não previstos na seção de produtos deste plano também são considerados como não contemplado no escopo do projeto.

Quadro 4

4. Organização do Projeto

Esta seção apresenta o organograma utilizado no projeto juntamente com seus papéis e responsabilidades.

4.1 Organograma

Esta seção apresenta o organograma do projeto, incluindo os papéis exigidos para realização do projeto e a relação entre os mesmos. O organograma do projeto é mostrado na Figura 2.

4.2 Papéis e Responsabilidades

A Tabela 3 descreve um conjunto de papéis e respectivas responsabilidades.

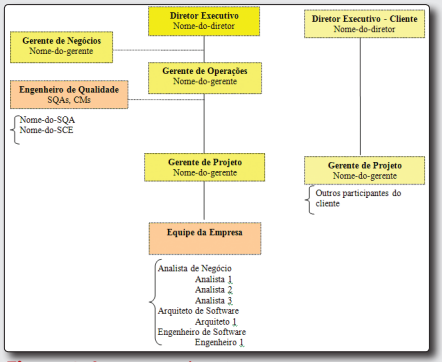


Figura 2. Organograma do projeto.

Papel	Responsabilidades
Diretor Executivo	• Divulgar as diretrizes estratégicas; • Tomar decisões estratégicas; • Garantir o cumprimento de responsabilidades estabelecidas entre as partes, possibilitando o sucesso do projeto; • Apoiar as decisões da equipe do projeto.
Gerente de Operações	• Prover todos os recursos necessários para a execução do projeto (capital humano, hardware, software, treinamento, etc.); • Realizar acompanhamento técnico, financeiro, de escopo, riscos e cronograma do projeto, conjuntamente com o gerente de projeto; • Assumir a responsabilidade sobre todo o ciclo de vida do serviço; • Posicionar o cliente sobre o andamento dos serviços junto com o gerente do projeto; • Negociar junto ao gerente de qualidade a resolução de questões de qualidade não solucionadas no âmbito do projeto.
Gerente de Negócios	• Elaborar propostas comerciais; • Conduzir negociação com o cliente quando houver mudanças no custo do projeto e que impactam o cliente.
Gerente de Projeto	• Realizar planejamento do projeto; • Gerenciar a equipe do projeto; • Gerenciar o orçamento do projeto; • Garantir o andamento adequado do projeto com relação ao planejado, gerenciando riscos e tomando ações preventivas e corretivas; • Posicionar o cliente sobre o andamento dos serviços; • Elaborar relatório de acompanhamento e conclusão do projeto; • Coordenar a interação da equipe com o cliente.
Analista de Negócio	• Realizar modelagem do negócio, quando apropriado; • Elicitar requisitos e realizar análise e projeto do sistema, elaborando modelos associados; • Elaborar projeto de testes e conduzir testes de sistema; • Elaborar documentação técnica necessária, por exemplo, ajuda (help), guia de usuário, material de treinamento; • Acompanhar atividades dos engenheiros de software, assegurando integridade com requisitos e casos de uso especificados; • Conduzir implantação do sistema.
Arquiteto de Software	• Definir a arquitetura do sistema; • Liderar e coordenar as atividades de engenharia de software do projeto; • Suportar o uso de ferramentas no âmbito do projeto; • Acompanhar os engenheiros de software, esclarecendo dúvidas técnicas; • Participar dos testes integrados do sistema; • Integrar os diversos componentes de software produzidos, gerando versão do sistema para implantação.
Desenvolvedor	• Implementar componentes do sistema; • Realizar testes unitários dos componentes de software, de acordo com os padrões adotados pelo projeto; • Participar da fase de projeto, quando apropriado; • Participar dos testes integrados do sistema.
Engenheiro de Qualidade	• Documentar e configurar o processo de software a ser utilizado; • Auditar o uso do processo; • Participar de revisões quando adequado; • Apoiar uso processo.

Tabela 3 - Papéis e Responsabilidades do Projeto

Quadro 5

5. Equipe e Infra-Estrutura do Projeto

Esta seção define a composição da equipe e lista a relação de ferramentas (Tabela 6) necessárias ao ambiente de desenvolvimento do projeto com o objetivo de garantir uma estrutura adequada para a execução das atividades previstas neste plano.

5.1 Planejamento da Alocação de Pessoal

A Tabela 4 apresenta o planejamento de alocação de pessoal do projeto. A quantidade de funções em relação ao cronograma foi reduzida no sentido de simplificação.

5.2 Equipe de Projeto

A Tabela 5 lista a relação de participantes do projeto e informações de período de participação e formas de contato. Novamente, o número de linhas nesta tabela foi reduzido para efeitos de simplificação, em comparação com o quantitativo da tabela anterior.

Função	Quantidade	% Alocação
Gerente de projeto	1	100
Analista de negócio	3	100
Arquiteto de software	1	50
Engenheiro de software	5	100
Web designer	1	50
Engenheiro de Qualidade	1	20
Engenheiro de Configuração	1	20
Administrador de Banco de Dados	1	10
Administrador de Sistemas	1	10

Tabela 4 – Planejamento de Alocação de Pessoal.

Nome	Papel	Período	E-mail	Telefone
Nome do gerente	Gerente de Projetos	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9990
Nome do analista	Analista de Negócio	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9991
Nome do analista	Analista de Negócio	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9992
Nome do arquiteto	Arquiteto de Software	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9993
Nome do engenheiro	Engenheiro de Software	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9994
Nome do engenheiro	Engenheiro de Software	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9995
Nome do engenheiro	Engenheiro de Software	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9996
A definir	Engenheiro de configuração	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9997
A definir	Engenheiro de qualidade	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9998
Engenheiro	Web designer	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9999
Engenheiro	Administrado de dados	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9900
A definir	Administrador de Sistemas	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9901

Tabela 5 - Equipe da empresa.

5.3 Ferramentas do Projeto

Funcionalidade	Ferramenta	Nº Licenças Necessárias
Gerência de projetos	MSProject	1
Gerência de configuração	CVS	0
Análise e projeto	Rational Rose	6
Implementação	JBuilder	6
Testes	A ser definido	-
SGBD	A ser definido	-

Tabela 6 - Ferramentas do projeto.

Quadro 6

6. Treinamentos do Projeto

Esta seção apresenta treinamentos previstos no projeto e a maneira pela qual eles serão realizados.

O treinamento da equipe técnica envolvida no desenvolvimento do projeto já faz parte do planejamento corporativo de capacitação de pessoal, de acordo com o plano de cargos em vigor na empresa, o qual descreve os perfis necessários para a execução de diferentes papéis. Desta forma, os profissionais selecionados já possuem perfil necessário para o cumprimento dos compromissos firmados. Uma vez que o projeto não incorpora nenhuma tecnologia desconhecida pela equipe técnica, não serão necessários treinamentos adicionais. Entretanto, qualquer treinamento necessário à equipe será relatado no primeiro relatório de acompanhamento de projeto.

Quadro 7

7. Acompanhamento do Projeto

Esta seção apresenta as atividades de acompanhamento e verificação do projeto, envolvendo a equipe do projeto, gerente de projeto da empresa, gerente de operações e representante do cliente. Estas atividades incluem a realização de reuniões e geração de relatórios descrevendo informações sobre o progresso do projeto, questões não resolvidas, dentre outras. A Tabela 7 contempla as atividades de acompanhamento planejadas para o projeto.

As reuniões e relatórios apresentados possuem o seguinte objetivo:

- Reunião de acompanhamento de atividades: tem por objetivo coletar periodicamente informações junto à equipe, cliente e demais áreas envolvidas, além de tomar ações corretivas quando forem identificados desvios do planejado.
- Relatório de progresso do projeto: tem por objetivo comunicar mensalmente os principais interessados sobre o andamento do projeto.
- Revisão formal do projeto: visa principalmente comunicar a gerência da empresa e representantes do cliente sobre status das atividades do projeto. Adicionalmente, deve-se verificar se o trabalho essencial da etapa anterior foi completado com sucesso (planejado vs. realizado), determinando pré-condições para o sucesso da próxima etapa, resolver questões do projeto, reafirmar compromissos e reavaliar riscos. Estas reuniões deverão ser realizadas a cada semestre e servem como prestação de contas para o cliente.
- Reunião de fechamento do projeto com a equipe: visa comunicar o feedback do cliente à equipe, além de discutir as lições aprendidas com o projeto e avaliar o feedback da equipe.
- Reunião de fechamento do projeto com o cliente: tem por objetivo avaliar a realização dos compromissos firmados entre as partes e obter o aceite formal do projeto pelo cliente. Esta formalização deve ser realizada através do formulário de aceitação do produto/serviço.

Reunião / Relatório	Realização	Participantes / Interessados
Reunião de acompanhamento de atividades	Semanal	• Gerente de projeto da empresa; • Equipe de projeto da empresa.
Relatório de progresso do projeto	Mensal	• Gerente de Operações; • Gerente de projeto da empresa; • Representante do cliente, se apropriado.
Revisão formal do projeto	Semestral	• Gerente de projeto da empresa; • Representante do cliente, se apropriado.
Reunião de fechamento do projeto com a equipe	Ao final do projeto	• Gerente de projeto da empresa; • Equipe do projeto da empresa; • Engenheiro de qualidade da empresa.
Reunião de fechamento do projeto com o cliente	Ao final do projeto	• Gerente de projeto da empresa; • Representante(s) do cliente.

Tabela 7 – Reuniões e Relatórios de Acompanhamento do Projeto

Quadro 8

8. Controle de Mudanças do Escopo de Projeto

O controle de mudanças de escopo do projeto considera solicitações referentes a alterações nas especificações funcionais ou técnicas, adição de novos requisitos, serviços adicionais de consultoria ou apoio técnico, alterações de cronograma e/ou na administração do projeto como um todo. Tais solicitações podem ser propostas pelo cliente ou pela equipe de projeto da empresa, sendo passíveis de um novo dimensionamento do esforço e custo necessários a sua implementação. As solicitações de mudança devem ser registradas na ferramenta de controle de mudanças (e.g. CVS).

O controle de mudanças permite ainda a realização de acordo entre as partes, considerando o impacto sobre o projeto, sobre os acordos e compromissos previamente estabelecidos, e sobre os cronogramas físico e financeiro do projeto. Os custos associados à mudança deverão ser apoiados pelo cliente.

Quadro 9

9. Cronograma

O cronograma do projeto contempla as atividades, milestones, dependências e recursos humanos alocados. Para obter detalhes sobre o mesmo, o documento do Cronograma do Projeto deve ser consultado.

9.1 Marcos Significativos do Projeto

A Tabela 8 apresenta os marcos significativos do projeto, com datas fictícias, bem como os artefatos importantes que serão entregues ao cliente nestes marcos, quando aplicável. Mudanças acordadas nas datas alvo serão acompanhadas e registradas, através das reuniões de acompanhamento do projeto.

Marco / Meta Física	Artefatos/ Indicadores Físicos de Execução	Responsável	Data Alvo
Estudo e análise de soluções tecnológicas para infra-estrutura orientada a serviços (web services).	Relatório técnico de estudo e análise	Empresa	02/10/2007
Definição de um modelo de desenvolvimento de web services.	Relatório técnico de modelo	Empresa	27/10/2007
Desenvolvimento de web services para controle de acesso	Documento de requisitos de web services de controle de acesso; Documento de análise e projeto; Código fonte implementado; Documento do plano de testes e dos resultados de testes.	Empresa	25/01/2008
Desenvolvimento de web services para a área de Turismo	Documento de requisitos de web services para a área de Turismo; Documento de análise e projeto; Código fonte implementado; Documento do plano de testes e dos resultados de testes.	Empresa	25/03/2008

Tabela 8 - Marcos Significativos do Projeto

Quadro 10

10. Gerência de Riscos

Os riscos identificados para o projeto serão detalhados nos relatórios de acompanhamento, bem como na planilha de acompanhamento do projeto. Estes documentos contêm a lista de riscos identificados, seus impactos e informações relevantes para definir estratégia de controle e atenuação (ou mitigação) do risco. Todo o acompanhamento dos riscos do projeto (riscos previamente identificados e riscos surgidos no decorrer do andamento do projeto) será registrado nos documentos supracitados.

Quadro 11

11. Gerência de Configuração

A gerência de configuração do projeto será detalhada no Plano de Gerência de Configuração do Projeto. A gerência de configuração visa estabelecer e manter a integridade dos produtos de um projeto de software durante o seu ciclo de vida. Suas atividades envolvem identificar a configuração do software, manter sua integridade durante o projeto e controlar sistematicamente as mudanças.

Os artefatos do projeto deverão ser disponibilizados no repositório do projeto. Além disso, os documentos de interesse do cliente e dos gestores sênior ou diretores da empresa serão disponibilizados no site do projeto, cujo acesso será restrito aos principais envolvidos.

Perceba que o cronograma apresentado no Quadro 9 destaca apenas as principais atividades e elas se encontram agrupadas, caracterizando os principais marcos do projeto. Não há, contudo, a intenção aqui em ser completo, mas a de ressaltar como as informações podem ser apresentadas no plano de projeto. Os Quadros seguintes, de 10 a 12, tratam de informações que são detalhadas em outros documentos, como indicado.

O conjunto de seções apresentados servem para ilustrar pontos importantes num plano de projeto. Não houve aqui a intenção de ser completo, mas de informar quais itens deveriam compor o plano de projeto, bem como a de ilustrar o conteúdo que pode ser encontrado nesse documento.

Conclusão

Um projeto compreende um conjunto de atividades inter-relacionadas com datas de início e fim, além de metas específicas. Também, define-se o que será entrada e saída para cada atividade. Tudo isso precisa estar muito bem 'orquestrado' num documento (que é o plano de projeto) para que o projeto possa ser conduzido de maneira adequada, alcançar seus objetivos e atender a metas de qualidade e cronograma. Sem esse 'mapa' torna-se muito difícil realizar com sucesso um projeto. ●

Dê seu feedback sobre esta edição!

A Engenharia de Software Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/esmag/feedback

Links

The Project Management Institute
www.pmi.org

PMBOK – Project Management Body of Knowledge
www.projectsmart.co.uk/pmbok.html

The Project Management Forum
www.pmforum.org

Checklist para Gerentes de Projeto
www.gantthead.com

The Chaos Report (by Standish Group) – Uma Análise sobre Sucesso e Falha de Projetos
www.projectsmart.co.uk/docs/chaos-report.pdf

Quadro 12

12. Testes do Projeto

Os testes são aplicáveis apenas para a implementação dos web services. Os testes de desempenho, carga, estresse e segurança e controle de acesso só se aplicam, quando pertinente, aos web services.

12.1 Estágios de Testes

A Tabela 9 apresenta os estágios de testes previstos para o projeto e seus objetivos. O planejamento e o controle dos testes é apresentado em outro documento (Plano de Testes).

Estágio de Testes	Objetivo
Teste Unitário	Visa validar individualmente os menores componentes (classes básicas e componentes) que serão utilizados na implementação das funcionalidades do sistema.
Teste de Integração	Objetiva validar a integração entre componentes e dos diversos pacotes na implementação das funcionalidades.
Teste de Sistema	Objetiva validar se todos os elementos do sistema foram adequadamente integrados e se realizam corretamente as funções especificadas.
Teste no Ambiente de Aceitação	Visa assegurar que tudo está realmente pronto para ser utilizado pelo usuário. Estes testes são realizados pela equipe de projeto da empresa antes da entrega do sistema ao cliente. Deve ser realizado em um ambiente o mais próximo possível do ambiente de produção.
Teste de Aceitação	Teste realizado pelo cliente objetivando aceitar ou homologar o sistema. Depois de realizado este teste com sucesso, o sistema estará pronto para ser implantado no ambiente de produção.

Tabela 9 - Estágios de Testes do Projeto

Melhorando Processos de Software através de Análise Causal de Defeitos

Marcos Kalinowski
Doutorando e mestre em Engenharia de Software da COPPE/UFRJ. Bacharel em Ciência da Computação pela UFRJ. Diretor executivo da Kali Software (www.kalisoftware.com), empresa de consultoria e treinamento em Engenharia de Software. Professor e pesquisador do curso de Ciência da Computação do Centro Universitário Metodista Bennett. Professor da pós-graduação e-IS Expert da UFRJ. Consultor de implementação, instrutor, avaliador e membro da equipe técnica do MPS.BR.

Marcelo Nascimento Costa
Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ. Bacharel em Ciência da Computação pela UFPA. Consultor da Kali Software (www.kalisoftware.com). Professor do curso de Ciência da Computação do Centro Universitário Metodista Bennett. Especialista em CMMI, tendo participado de implementações e avaliações deste modelo.

Este artigo fornece uma visão geral de como a análise causal de defeitos pode ser aplicada para obter melhoria de processo baseada no produto. A base teórica a respeito de análise causal de defeitos é descrita, bem como maneiras eficientes de implementar a mesma em organizações de software. Adicionalmente, um exemplo real de sua aplicação na indústria é apresentado. Considerando que apenas dados relacionados a defeitos precisam ser coletados, a análise causal de defeitos é indicada mesmo para empresas com baixo nível de maturidade, permitindo a estas reduzir significativamente suas taxas de defeitos e conseqüentemente o custo e tempo de desenvolvimento.

Introdução

Análise causal é considerada na maioria das abordagens de melhoria de processo de software, tais como MPS, CMMI, Six Sigma, e Lean. A análise causal de defeitos visa identificar as causas dos defeitos para que ações possam ser tomadas para prevenir sua ocorrência em iterações ou projetos futuros. Embora a identificação das causas seja realizada em projetos específicos, estas causas podem revelar oportunidades de melhoria dos ativos de processo organizacionais.

Por trás da idéia da melhoria do processo de software baseada em produto (product focused software process improvement) está realizar um programa de melhoria de processos que trate características de qualidade do produto de uma maneira explícita. Inquestionavelmente, uma das principais características de qualidade do produto de software são os defeitos encontrados em seus artefatos. Desta forma, a análise causal de defeitos fornece uma oportunidade eficiente de melhoria de processos de software baseada em produto, conforme apontado em (Kalinowski, 2007).

Considerando ainda que dados relacionados a defeitos são comumente coletados em organizações de software,

a análise causal de defeitos fornece um meio para que organizações melhorem seus processos, migrando para níveis mais altos de maturidade, com ênfase direta na diminuição da taxa de defeitos e conseqüentemente no retorno de investimento das melhorias implementadas. Assim, a implementação eficiente de métodos de análise causal de defeitos tem se tornado uma vantagem competitiva para organizações de software.

Além desta base conceitual introdutória, este artigo também apresenta uma fundamentação teórica a respeito de análise causal de defeitos e maneiras como ela pode ser implementada em organizações de software são descritas, e um exemplo de sua aplicação na indústria.

Análise Causal de Defeitos

Análise causal de defeitos melhora a qualidade e produtividade tratando da melhoria de processos de modo a prevenir a introdução de defeitos no produto. Como conseqüência, de acordo com o SEI (Software Engineering Institute) (SEI, 2006), ajuda a atingir objetivos de qualidade e produtividade dos projetos. Neste contexto, Robitaille (2004) ressalta a importância de se tratar o processo de análise causal como um meio para identificar oportunidades de melhoria dos ativos de processo organizacionais. Card (2005), por sua vez, discute a importância de gerenciar e aprender a partir dos defeitos de software.

Dada tal importância, a análise causal de defeitos tem sido discutida desde os anos 70 (Endres, 1975) e é citada por Boehm (2006) como uma das principais contribuições daquela década para a engenharia de software. Desde aquele artigo inicial, várias abordagens têm sido formalizadas e implementadas em organizações de software de diferentes tamanhos e níveis de maturidade. De acordo com uma revisão sistemática sobre análise causal de defeitos (Kalinowski et al., 2008) o esforço para realizar análise causal é extremamente baixo e varia entre 0,5 e 1,5% do esforço total do desenvolvimento, já incluindo o esforço para implementar as oportunidades de melhoria identificadas. De acordo com esta mesma revisão, a redução média na taxa de defeitos, por outro lado, tem sido superior a 50%. Tendo em vista o impacto positivo da redução da taxa de defeitos em variáveis como esforço, custo e tempo, trata-se de um processo cuja implementação traz um alto retorno de investimento para a organização.

A análise causal de defeitos é um processo sistemático para identificar e analisar causas associadas à ocorrência de tipos específicos de defeitos, permitindo identificar oportunidades de melhoria para os ativos de processo organizacionais a fim de prevenir a ocorrência daquele mesmo tipo de defeito em projetos futuros. Card (2005) resume a análise causal de defeitos em seis passos:

I. Selecionar a amostra do problema. Em organizações maduras a análise causal pode ser disparada por uma instabilidade detectada em um gráfico de controle estatístico de processos (Florac e Carleton, 1999). Os dados relacionados a esta instabilidade tornam-se então alvo da análise causal e uma amostra destes dados deve ser selecionada para a reunião de análise causal. Em organizações de menor maturidade, a análise causal pode ser disparada de uma maneira mais informal, por exemplo, pela observação de algumas tendências ou comportamentos estranhos em métricas coletadas a respeito de defeitos.

Um exemplo real de gráfico de controle de processo estatístico, considerando os defeitos encontrados em uma inspeção de requisitos por unidade de tamanho é mostrado na Figura 1. Esta apresenta dois gráficos. O primeiro refere-se ao projeto individual que mostra o número de defeitos por unidade de tamanho em cada inspeção de requisitos do projeto. O segundo gráfico é uma faixa móvel que mostra a variação entre dois projetos consecutivos. As linhas vermelhas representam os limites de controle superior e inferior, sendo estes 3 desvios padrão acima e abaixo da média respectivamente (no caso do gráfico com faixa móvel o limite inferior do gráfico é sempre zero, o que significa nenhuma variação). Este gráfico é conhecido como gráfico XmR e pode ser gerado em ferramentas como o MiniTab, do SEI. Embora o gráfico XmR seja o mais utilizado, o gráfico mais adequado para controlar dados de defeitos (que comumente se aproximam mais de uma distribuição de Poisson do que de uma distribuição normal) é o gráfico U, que apresenta limites de controle variáveis.

II. Classificar problemas selecionados. Entre as informações básicas a serem coletadas a respeito dos defeitos estão o tipo do defeito, o momento (ou fase desenvolvimento) da sua introdução, o momento da sua detecção, e o esforço necessário para removê-lo. Uma taxonomia comumente utilizada para expressar a natureza dos defeitos encontradas em revisões por pares é a descrita por Shull (1998), que envolve os seguintes tipos: omissão, informação estranha, ambigüidade, fato incorreto e informação inconsistente. Entretanto, tipos de defeitos mais específicos (customizados para o contexto organizacional) ou alguma informação adicional pode ser

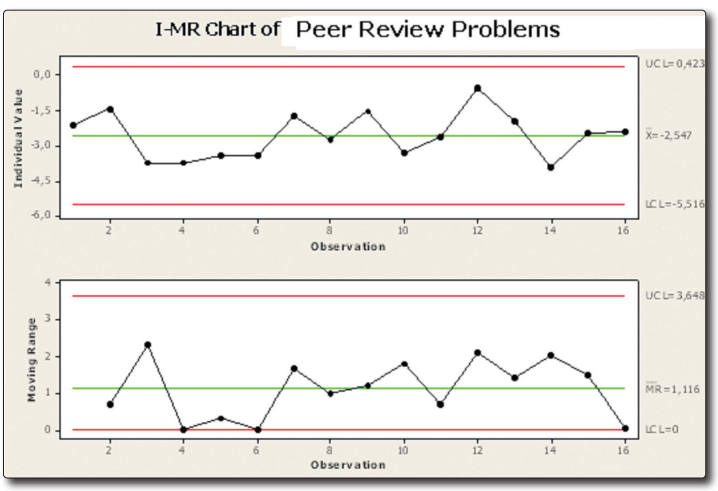


Figura 1. Gráfico XmR para controle estatístico de processos.

útil. Por exemplo, se um checklist é utilizado durante a revisão por pares, uma informação adicional interessante seria o item do checklist que levou a encontrar o defeito. O evento que disparou a detecção do defeito é também considerado pela abordagem ODC (Orthogonal Defect Classification) criada pela IBM (Chillarege et al., 1992), que sugere registrar os seguintes atributos para os defeitos: tipo do defeito, omitido/incorreto, evento disparador, fonte (fase de introdução), e impacto. O uso de tabelas de índice–cruzado, histogramas ou gráficos de Pareto pode ajudar a agrupar defeitos baseados na sua respectiva classificação. Erros sistemáticos (tópico do próximo passo) possuem maior probabilidade de serem encontrados nos tipos de defeitos mais comuns.

Um gráfico de pareto agrupa ocorrências em categorias de acordo com o esquema de classificação, ressaltando as categorias mais comuns e a soma destas categorias no próprio gráfico. Basicamente, este é um histograma partindo da categoria mais comum para a menos comum junto com um gráfico de linha mostrando a soma das categorias do histograma. Um exemplo de gráfico de Pareto é mostrado na Figura 2. Neste gráfico os tipos de defeito A e B juntos são responsáveis por mais de 70% dos defeitos.

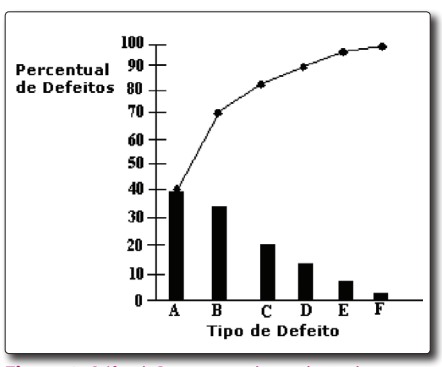


Figure 2. Gráfico de Pareto reportadas por de tipo de atraso em chegadas.

III. Identificar erros sistemáticos. Erros sistemáticos são aqueles que resultam na introdução de tipos de defeitos similares em diferentes ocasiões (CARD, 2005). Usualmente erros sistemáticos são relacionados a uma atividade específica ou parte de um produto. Exemplos de erros sistemáticos podem ser encontrados em (Card, 1998), (Leszak et al., 2000) e na conclusão deste artigo. Encontrar erros sistemáticos indica a existência de oportunidades de melhoria para os ativos de processo organizacionais.

IV. Identificar as principais causas. De acordo com Card (2005), vários fatores poderiam gerar um erro sistemático. Tratar todos eles pode não ser economicamente viável. Desta forma, esforço deve ser empregado a fim de encontrar as principais causas. Neste passo, conhecer o contexto organizacional torna-se uma habilidade útil. A participação dos autores responsáveis pela introdução dos defeitos no artefato é fortemente recomendada. Uma das técnicas utilizadas para encontrar as principais causas é a aplicação do diagrama de Iskikawa (ou "espinha de peixe") (Ishikawa, 1976). De acordo com Ishikawa as causas podem ser agrupadas em uma das quatro seguintes categorias: métodos, ferramentas e ambiente, pessoas, entrada/requisitos.

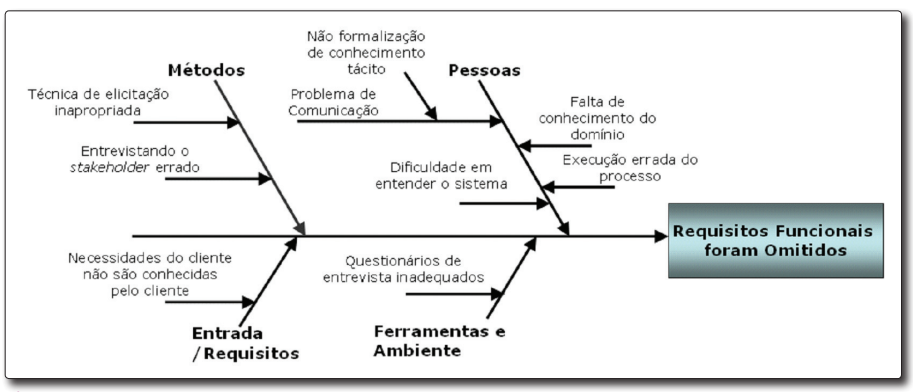


Figure 3. Diagrama de Ishikawa. Adaptado de (Kalinowski et al., 2008b).

A principal idéia desses diagramas é discutir as causas dos problemas, agrupando-os em espinhas do diagrama. Cada espinha pode, então, ser recursivamente refinada pela adição de novas espinhas. Um diagrama de Ishikawa, preenchido com exemplos de causas para o erro sistemático "Requisitos Funcionais foram Omitidos", agrupadas em suas categorias, é mostrado na Figura 3. Esta figura pretende ser apenas ilustrativa e não completa, as causas irão depender do contexto organizacional onde a análise causal de defeitos está sendo realizada. No diagrama da figura uma causa para omitir requisitos funcionais da categoria pessoas foi um problema de comunicação, que por sua vez foi causado pela não formalização de conhecimento tácito.

V. Desenvolver propostas de ação. A partir do momento que as principais causas foram identificadas, propostas de ação devem ser desenvolvidas para evitar a ocorrência daquelas causas em iterações ou projetos futuros. O número de propostas de ação deve ser pequeno. Propostas de ação devem ser específicas e endereçar as principais causas do erro sistemático de forma que a implementação possa ser facilmente gerenciada. Geralmente as causas da categoria "Métodos" são endereçadas através de propostas de ação que implicam em mudanças no processo de desenvolvimento. As causas da categoria "Pessoas", por sua vez, são comumente endereçadas através de ações que envolvem treinamento. As causas da categoria "Entradas/Requisitos" podem sugerir, por exemplo, mudanças nos planos de comunicação com o cliente. Por fim, as causas da categoria "Ferramentas e Ambiente" podem sugerir, por exemplo, mudanças no ambiente de desenvolvimento (incluindo ferramentas e os artefatos de apoio utilizados).

VI. Documentar resultados da reunião. Os registros dos resultados das reuniões são necessários para assegurar que as ações sejam implementadas.

De acordo Card (2005), uma equipe de ação deve ser estabelecida de maneira que a implementação das ações possa ser gerenciada.

Estes passos podem ser implementados como parte do processo de prevenção de defeitos descrito por Mays et al. (1990), ilustrado na Figura 4. Desta Figura, a análise causal de defeitos compreende a reunião de análise causal e atividades da equipe de ação desse processo. As demais atividades ilustradas são a própria atividade de desenvolvimento, onde os dados sobre defeitos são coletados e a atividade de lançamento (kickoff), onde as ações implementadas são comunicadas para a equipe de desenvolvimento.

O uso de análise causal de defeitos também fornece mecanismos eficientes para disseminar conhecimento na organização. Por exemplo, lições aprendidas através da análise de vários projetos descrevem tipos de defeitos, erros sistemáticos, propostas de ações e os efeitos obtidos com a implementação destas propostas de ações para tratar erros sistemáticos. Essas lições constituem importante conhecimento para ser gerenciado a fim de evitar as causas de erros sistemáticos no futuro e para apoiar o tratamento caso algum tipo de erro sistemático ocorra novamente. Na Figura 4 a manutenção deste tipo de conhecimento é ilustrada através da base de experiências.

Informações adicionais sobre como implementar o processo e as técnicas de análise causal de defeitos descritos nesta seção em uma organização de software podem ser encontradas em (Kalinowski et al., 2008a). Este artigo detalha, com base em uma revisão sistemática da literatura em que 50 artigos foram analisados, as pré-condições para se utilizar análise causal, abordagens e técnicas que podem ser seguidas, como defeitos e suas causas podem ser categorizados e que resultados esperar de sua implementação em uma organização de software. Segue um resumo dos resultados:

- Pré-condições: coletar dados sobre defeitos (momento de introdução, momento de detecção e tipo) e idealmente ter um processo definido para os projetos;
- Abordagens e técnicas: uma das abordagens simples e que tem apresentado resultados muito positivos é o processo de prevenção de defeitos. Entre as técnicas se destacam o uso dos gráficos de Pareto e de Ishikawa. O gráfico U é indicado para o controle estatístico do processo;
- Categorização de defeitos: mostra-se útil capturar tanto a natureza do defeito quanto o seu tipo. As taxonomias podem variar de acordo com a organização e o propósito da análise causal de defeitos. Caso uma taxonomia de classificação de defeitos ainda não exista na organização sugerimos utilizar a categoria definida por Shull (1998) como ponto de partida para a natureza dos defeitos: ambigüidade, omissão, fato incorreto, informação inconsistente e informação estranha. Para o tipo de defeito sugerimos como ponto de partida as categorias do ODC (Chillarege et al., 1992): interface, funcionalidade, build/package/merge, atribuição, documentação, verificação, algoritmo e timing/serialização;
- Categorização de causas: as categorias de causa propostas inicialmente por Ishikawa no contexto da manufatura (Ishikawa, 1976) podem ser utilizadas também para o desenvolvimento de software. Refinamentos podem ser realizados neste esquema de acordo com as necessidades da organização. Para organizações geograficamente distribuídas pode ser interessante considerar também a categoria "organização";
- Resultados esperados: esforço variando entre 0,5 a 1,5% do esforço total do desenvolvimento e redução da taxa de defeitos em aproximadamente 50%.

Um Exemplo Prático do Uso de Análise Causal de Defeitos na Indústria de Software

Nesta seção é descrito um exemplo de análise causal de defeitos aplicado a defeitos encontrados em requisitos de um projeto de desenvolvimento de software real de larga escala utilizando um processo iterativo incremental para desenvolver os diferentes módulos do software. Assim, a atividade de desenvolvimento sob análise era a atividade de elicitação e análise de requisitos. Esta atividade era realizada de forma isolada para cada um dos módulos. Os dados de defeitos eram provenientes de inspeções de software aplicadas aos requisitos. Maiores detalhes sobre a experiência com as inspeções de requisito do projeto, explicitando como as equipes de inspeção foram organizadas e informações sobre os defeitos encontrados e custo e tempo das inspeções são descritos em (Kalinowski et al., 2007). Em relação aos resultados das inspeções, como já era de se esperar, o uso de inspeções neste projeto apresentou baixo custo e atingiu o seu objetivo de encontrar defeitos (cedo) dentro do processo de desenvolvimento, quando sua correção é mais barata. Em relação aos tipos de defeitos encontrados, a distribuição dos tipos encontrados depois das inspeções dos requisitos dos dois primeiros módulos é apresentada nas Figuras 5 e 6, respectivamente.

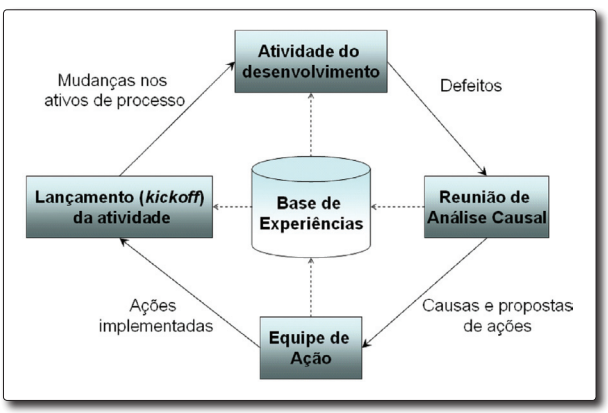


Figura 4. O processo de prevenção de defeitos, adaptado de (Mays et al., 1990 apud Rombach e Endres, 2003).

Tendo em vista a distribuição dos defeitos dos requisitos dos dois primeiros módulos, o alto percentual de omissões pôde ser observado como uma possível tendência no projeto. Assim, o alto número de omissões motivou uma análise a fim de encontrar erros sistemáticos no processo de elicitação dos requisitos que possam ter gerado os defeitos do tipo omissão.

A análise foi executada baseada na descrição de defeitos de omissão. A conclusão da análise foi que o erro sistemático estava relacionado a entrevistas conduzidas apenas com funcionários do nível operacional. Em várias situações os empregados do nível operacional não tinham uma visão geral da funcionalidade do sistema e de como estas funcionalidades se integram. Considerando que a equipe de inspeção tinha um representante do cliente no nível gerencial, ele foi capaz de encontrar várias omissões. A proposta de ação resultante foi mudar o processo de elicitação de requisitos para assegurar que sejam entrevistados um representante gerencial e um representante operacional do cliente para os próximos módulos. A distribuição de defeitos de requisitos do terceiro modulo, após a implementação da proposta de ação, é apresentada na Figura 7.

É possível notar que a proporção de defeitos do tipo omissão na terceira inspeção foi menor que dos dois primeiros módulos. Acreditamos que esta variação possa ter sido conseqüência da implementação da proposta de ação. O restante da distribuição aparentemente seguiu uma tendência similar. Outro ponto que pôde ser observado foi uma redução significativa na taxa de defeitos (medida neste exemplo pela métrica defeitos por página) entre os módulos, passando de 1,4 no primeiro módulo para 0,84 no segundo módulo e 0,58 no terceiro. Entretanto esta variação positiva pode estar relacionada a outros fatores, como o próprio aprendizado decorrente do uso de inspeções de software, entre outros.

Note que neste exemplo técnicas sofisticadas como gráficos de controle estatístico de processos, gráficos de Pareto ou digramas de Ishikawa, não foram utilizadas. É importante ressaltar, entretanto, que para situações mais complexas de análise causal de defeitos estas técnicas e o uso de abordagens mais sistemáticas podem mostrar-se extremamente úteis.

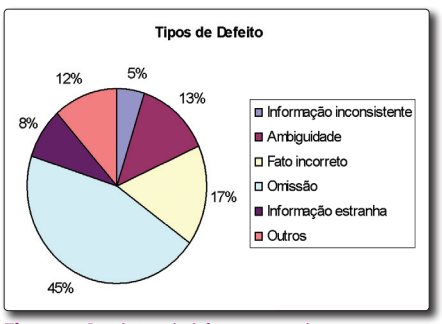


Figura 5. Distribuição de defeitos encontrados na inspeção do primeiro módulo.

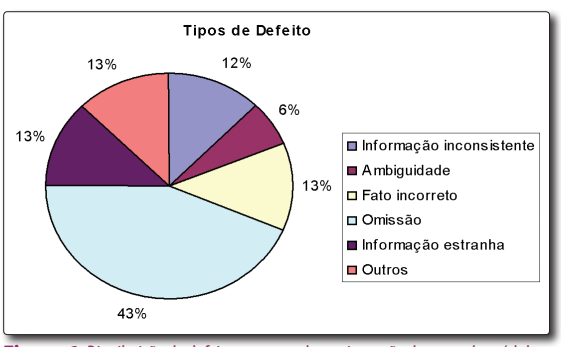


Figura 6. Distribuição de defeitos encontrados na inspeção do segundo módulo.

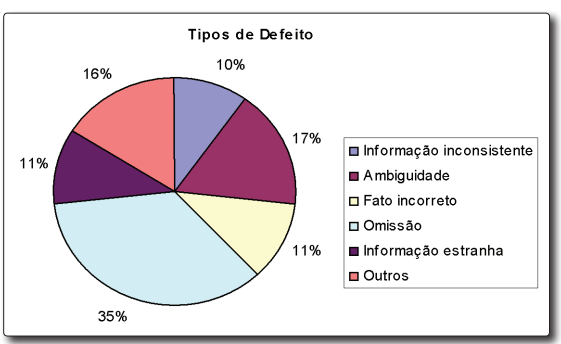


Figura 7. Distribuição de defeitos encontrados na inspeção do terceiro módulo.

Conclusões

Neste artigo apresentamos a análise causal como um meio de identificar oportunidades de melhoria para processos de software com base em uma característica concreta do produto: os defeitos introduzidos nos artefatos ao longo do desenvolvimento. O propósito da análise causal de defeitos e uma breve fundamentação teórica foram apresentados. Adicionalmente, algumas técnicas comumente utilizadas para apoiar a análise causal de defeitos foram descritas. Insights adicionais foram fornecidos através de um exemplo real de desenvolvimento iterativo incremental em que defeitos de requisitos foram analisados.

Considerando que dados relacionados a defeitos são comumente coletados em organizações de software, a análise causal fornece um meio para que organizações de software melhorem seus processos rumo a níveis mais altos de maturidade. A visão de utilizar análise causal como um instrumento para se alcançar processos mais maduros contrasta com seu posicionamento em modelos de maturidade como o MPS e o CMMI, onde a análise causal de defeitos é exigida somente no nível mais alto de maturidade. ●

Referências

BOEHM, B., "A View of 20th and 21st Century Software Engineering", in 'ICSE '06: Proceeding of the 28th International Conference on Software Engineering', ACM Press, New York, NY, USA, pp. 12-29, 2006.

CARD, D.N., Learning from our mistakes with defect causal analysis, IEEE Software, 15(1), pp. 56-63, 1998.

CARD, D. N., "Defect Analysis: Basic Techniques for Management and Learning", Advances in Computers 65: 260-297, 2005.

CHILLAREGE, R., BHANDARI, I., CHAAR, J., HALLIDAY, M., MOEBUS, D., RAY, B., WONG, M.Y., "Ortogonal Defect Classification – A Concept for In-Process Measurement", IEEE Transactions on Software Engineering, vol. 18, pp. 943-956, 1992.

ENDRES, A., "An Analysis Of Errors and Their Causes in System Programs", ACM Sigplan Notices, Volume 10, Issue 6, 1975.

FLORAC, W.A., CARLETON, A.D., Measuring the Software Process – Statistical Process Control for Software Process Improvement, Addison-Wesley, 1999.

GRADY, R. B., "Software Failure Analysis for High-Return Process Improvement Decisions", Hewlett-Packard Journal, 47 (4), 15 - 24, 1996.

ISHIKAWA, K., Guide to Quality Control, Asian Productivity Organization, Tokyo, 1976.

KALINOWSKI, M., Defect Causal Analysis: An Opportunity for Product Focused Software Process Improvement, Invited paper (keynote) at the V CICIS (Congreso Internacional de Computación y Ingenieria de Sistemas), Moquegua, Peru, 2007.

KALINOWSKI, M., SPINOLA, R.O., DIAS NETO, A.C., BOTT, A., TRAVASSOS, G.H., "Inspeções de Requisitos em Desenvolvimento Incremental: Uma Experiência Prática", VI Simpósio Brasileiro de Qualidade de Software, Porto de Galinhas – PE, Brasil, 2007.

KALINOWSKI, M., TRAVASSOS, G.H., CARD, D.N., "Guidance for Efficiently Implementing Defect Causal Analysis", VII Simpósio Brasileiro de Qualidade de Software, Florianópolis - SC, Brasil, 2008a.

KALINOWSKI, M., TRAVASSOS, G.H., CARD, D.N., "Towards a Defect Prevention Based Process Improvement Approach", 34th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Parma, Italy, 2008b.

LESZAK, M., PERRY, D., STOLL, D., "A Case Study in Root Cause Defect Analysis", in 'International Conference on Software Engineering, ICSE 2000', pp. 428-437, 2000.

MAYS, R.G., "Applications of Defect Prevention in Software Development", IEEE Journal on Selected Areas in Communications, Vol.8, No.2, February 1990.

ROBITAILLE, D., Root Cause Analysis – Basic Tools and Techniques, Paton Press, 2004.

ROMBACH, D., ENDRES, A., A Handbook of Software and Systems Engineering – Empirical Observations, Laws and Theories, Pearson Addison Wesley, 2003.

SEI. SOFTWARE ENGINEERING INSTITUTE. CMMI for Development (CMMI-DEV), Version 1.2, Technical report CMU/SEI-2006-TR-008. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2006.

SHULL, F., "Developing Techniques for Using Software Documents: A Series of Empirical Studies", Ph.D. thesis, University of Maryland, College Park, 1998.

WALIA, G., CARVER, J., PHILIP, T., "Requirements Error Abstraction and Classification: An Empirical Study", Proceedings of the 2006 International Symposium on Empirical Software Engineering (ISESE), Sept. 21-22, Rio de Janeiro, Brazil. p. 336-345, 2006.

Modelos e Abordagens para Gerenciamento de Riscos de Projetos de Software

Cristine Gusmão
cristine@dsc.upe.br
Professora Assistente do Departamento de Sistemas Computacionais da Escola Politécnica da Universidade de Pernambuco (POLI – UPE), onde leciona várias disciplinas na graduação e pós-graduação (especialização e mestrado) e das Faculdades Integradas Barros Melo. Doutora e Mestre em Ciência da Computação pela Universidade Federal de Pernambuco. Graduada em Engenharia Elétrica – Eletrotécnica pela Universidade Federal de Pernambuco.

Em muitas organizações que desenvolvem software, a Gerência de Riscos é vista como um processo difícil de ser executado de maneira adequada em função das pressões de custo e prazo que caracterizam o ambiente de negócios e da pouca disponibilidade de informações. Por outro lado, o aumento contínuo da qualidade e produtividade exigido pelo mesmo mercado, cada vez mais competitivo, só é conseguido através da melhoria e evolução dos processos de gerência e de desenvolvimento de software.

Nas seções seguintes serão apresentados e analisados alguns modelos e abordagens para gerenciamento de riscos de projetos de software.

Modelos e abordagens para gerir riscos

Riscos em ambientes de desenvolvimento de software passaram a receber uma atenção maior no final da década de 80, quando Barry Boehm propôs e apresentou uma abordagem para gerir riscos [Boehm 1991].

De acordo com Harold Kerzner, as atividades de gerenciamento de riscos associadas ao desenvolvimento de software passaram a ter destaque no cenário internacional em 1996, ainda como reflexo da crise do software e a necessidade de melhoria dos sistemas [Kerzner 2000].

Existe uma diversidade de modelos e abordagens que tratam atividades para gerenciamento de projetos de desenvolvimento de software. Para facilitar o estudo analítico foram estabelecidos dois critérios de comparação entre os modelos:

Modelo de Referência – Dentro do estudo dos modelos de Gerência de Riscos um dos pontos fundamentais é verificar a base teórica utilizada. Essa base teórica é composta por características e requisitos que são refletidos no processo utilizado e nas atividades definidas no modelo. A base teórica também reflete a atualidade das pesquisas para a definição

e adaptação do modelo. De forma adicional, promove a evolução do conhecimento para a construção do modelo em análise;

Componentes do Modelo e Relacionamentos – Outro ponto a considerar, não menos importante, é a captura dos componentes arquiteturais do modelo e suas relações, como componentes que foram criados para garantir a aplicação da Gerência de Riscos promovendo a confiabilidade do resultado esperado. Este critério está intimamente ligado ao modelo de referência utilizado.

A escolha de modelos e abordagens que serão apresentadas a partir de agora foi baseada em sua influência nos atuais modelos de gestão de projetos e de qualidade de software, em especial gerenciamento de riscos em ambientes de desenvolvimento de software.

Gerência de riscos segundo Barry W. Boehm

O modelo que introduziu o estudo da gerência de riscos na Engenharia de Software foi apresentado por Barry Boehm no final da década de 80. Nesta época Boehm era membro integrante do Departamento de Defesa dos Estados Unidos – DoD (Department of Defense).

O modelo apresentado por Boehm está baseado no modelo espiral, também de sua autoria [Boehm 1988]. Esse modelo foi desenvolvido ao longo de muitos anos com base na experiência adquirida pela aplicação do modelo cascata (Waterfall Model) em grandes projetos do governo norte-americano.

O modelo espiral, apresentado na Figura 1, tem um típico ciclo de vida apresentando como atividade inicial a identificação dos objetivos relacionados:

- Ao produto em elaboração;
- Às alternativas de solução para os objetivos definidos; e
- Às restrições de implementação destas alternativas.

O passo seguinte é avaliar as alternativas identificadas para implementação dos objetivos do produto em elaboração e suas restrições. Freqüentemente, nesse processo são identificadas áreas de incertezas que são origens significantes de riscos de projetos.

Uma vez avaliados os riscos é necessário definir as formas de tratá-los. Deste modo, técnicas como prototipagem, simulações e benchmarking são realizadas. O processo é contínuo e existirão validações dos requisitos, especificação do projeto (design), implementação e testes, onde cada ciclo é planejado e avaliado.

Boehm sugere um processo de gerência de riscos composto por duas grandes etapas: Avaliar Riscos e Controlar Riscos. Cada uma destas etapas é composta por atividades e estas últimas, por passos, conforme mostra a Figura 2.

A etapa Avaliar Riscos contempla atividades de identificação, análise e priorização dos riscos, Controlar Riscos pelas atividades de planejamento de respostas aos riscos identificados, resolução dos riscos e monitoração [Boehm 1991].

Para cada uma das atividades do modelo, Boehm sugere a realização de passos relacionados ao uso de métodos e técnicas. Na realidade, as atividades são apresentadas através dos objetivos e metas que devem ser alcançados.

Em avaliando os riscos, a atividade Identificar Riscos tem como finalidade o desenvolvimento de lista de riscos relacionados aos fatores de sucesso do projeto. As técnicas sugeridas para esta atividade são as listas de verificação, avaliação das metas, comparação com experiências passadas e decomposição.

A segunda atividade é Analisar Riscos, onde são tratados os aspectos qualitativos e quantitativos dos riscos. Os aspectos qualitativos são identificados através da probabilidade e impacto da perda; já os aspectos

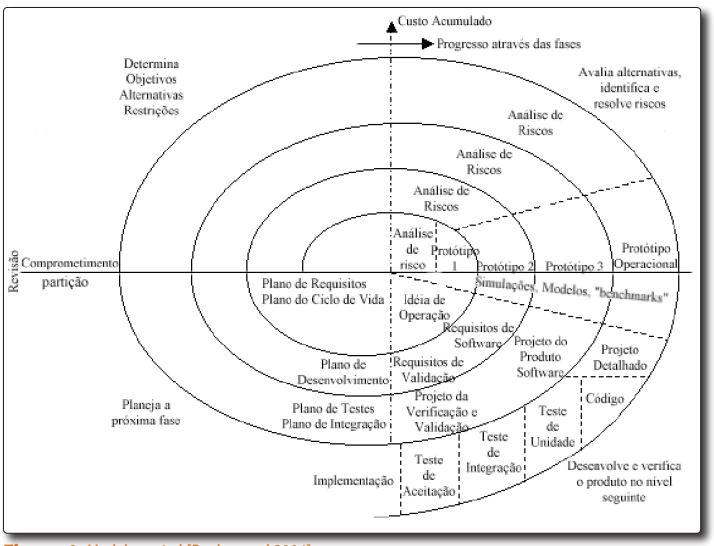


Figura 1. Modelo espiral [Boehm et al 2004]

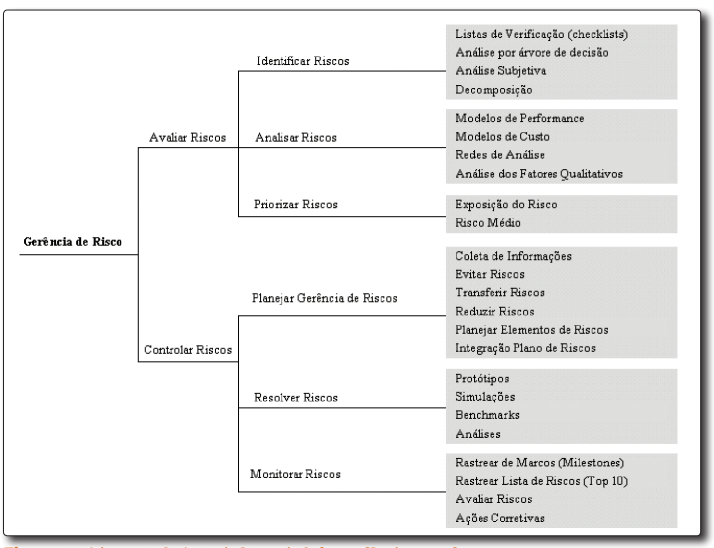


Figura 2. Adaptação Gerência de Riscos de Software [Boehm 1991]

quantitativos são investigados através dos modelos de custo e das análises estatísticas de decisão.

Priorizar Riscos é a terceira e última atividade. Nela os riscos identificados e analisados serão ordenados de acordo com o impacto para o projeto. Algumas técnicas são sugeridas como a análise da Exposição do Risco, Técnica Delphi ou técnicas de consenso entre os grupos de trabalho, como a técnica Nominal Technical Group.

Em controlando os riscos, a atividade Planejar Gerência de Riscos ajuda a endereçar todos os riscos priorizados em um documento para acompanhamento e controle, de acordo com as estratégias utilizadas. Algumas técnicas como listas de verificação e estratégias de respostas aos riscos são indicadas.

A atividade Reduzir Riscos tem a finalidade de gerar uma situação onde todos os riscos identificados e priorizados são eliminados ou tratados (resolvidos). Algumas técnicas sugeridas são, por exemplo, protótipos, simulação e benchmarking.

Por fim, a atividade Monitorar Riscos tem o objetivo de acompanhar a execução do projeto avaliando seu progresso e fazer uso de ações corretivas quando necessário.

Para ilustrar e facilitar o entendimento dos conceitos e atividades da Gerência de Riscos, o Quadro 1 apresenta o Projeto Arriscado (esse projeto é totalmente fictício).

A partir deste contexto é possível identificarmos alguns riscos como:

- A interface entre os sistemas pode não ser alcançada; e
- O gerente do projeto não tem experiência no gerenciamento de projetos complexos.

As empresas geralmente desenvolvem uma lista de riscos, onde estes eventos adversos são analisados e priorizados de acordo com o impacto negativo que podem influenciar o ambiente organizacional. Desta forma, de acordo com a experiência dos integrantes da organização, ferramentas de apoio utilizadas, processo definido de avaliação e controle de riscos é possível definir estratégias para gerenciá-los. A Tabela 1 mostra um exemplo de matriz de riscos para a situação apresentada na Nota 1.

A Tabela 1 é composta por seis colunas que representam informações importantes para o registro dos riscos identificados no projeto. Inicialmente têm-se o campo de identificação do risco (Id) que representa um número ou composição de caracteres que identificam o risco.

Em seguida, encontra-se o tipo do risco, ou seja, uma classificação para os tipos de ricos identificados no projeto. Logo após, é disponibilizada a descrição do risco que nada mais é do que a situação visualizada como um possível risco.

Por fim, as três últimas colunas que representam os elementos essenciais de uma situação de risco: a probabilidade de ocorrência, ou seja, quão iminente está o risco de ocorrer; o impacto que representa a potencialidade que o risco pode trazer em termos de perdas ou prejuízos e, por último, a exposição, métrica utilizada para representar o grau de risco em que a situação expõe o projeto.

A exposição ao risco é composta por um componente que representa a probabilidade de insatisfação e um segundo componente que é a perda associada. Este tipo de análise está enquadrado na avaliação através de aspectos qualitativos, que tem sua vantagem relacionada ao uso de lições aprendidas e experiência das equipes envolvidas. Para subsidiar a análise destas exposições encontradas, Boehm propõe o uso conjunto da árvore de decisão como forma de compor as exposições de riscos priorizados, estimando os erros críticos do projeto em avaliação.

É importante salientar que para se chegar a uma matriz de riscos, tarefas de planejamento da gerência de riscos

Nota 1. Contexto do Projeto Arriscado

REX (Risk Expert) é um respeitado líder de projeto da Companhia do Sr. Bonzinho SA, uma organização que trabalha com o desenvolvimento de sistemas de apoio à decisão.

Estes sistemas, na sua grande maioria, trabalham atividades de Relacionamento com o Cliente (CRM – Customer Relationship Management). Recentemente, REX foi nomeado gerente de projeto para um novo projeto da organização, chamado Arriscado. Este projeto tem o objetivo de integrar informações entre sistemas, melhorando a produtividade e atendimento das organizações. A meta é integrar o sistema ERP com o CRM, evitando a duplicidade de informações e potencializando a definição dos perfis dos clientes e conseqüente conhecimento de suas necessidades.

REX tem uma boa experiência no gerenciamento de projetos, pois há um bom tempo vem lidando com projetos de pequeno porte, mas nunca trabalhou com projetos desta complexidade e tamanho. Este projeto é considerado de médio porte para as estratégias organizacionais. REX tem a impressão de ser um grande projeto. Seu objetivo é garantir o sucesso. REX já documentou algumas informações passadas pelo seu superior imediato e identificou as partes interessadas do projeto Arriscado. Já fez a revisão do processo com os membros da equipe, e o próximo passo é preparar o Plano de Gerenciamento de Riscos do projeto.

Id	Tipo do Risco	Descrição do Risco	Probabilidade	Impacto	Exposição
1	Interface	O Sistema CRM pode não fazer interface com o Sistema ERP	0,05	0,80	0,04
2	Transferência de Dados	Transferência do conhecimento de um sistema para o outro	0,60	0,40	0,24
3	Migração de Dados	Transferência das informações dos clientes	0,05	0,80	0,04
4	Equipamentos	Mais equipamentos podem ser necessários	0,40	0,40	0,16
5	Pessoas Chaves	Experiência importante de pessoa do CRM pode ser negligenciada	0,60	0,60	0,36
6	Gerência de Projeto	O processo de Gerenciamento não está completamente estabelecido	0,20	0,20	0,04
7	Espaço	Não ter espaço para equipamentos adicionais	0,60	0,20	0,12
8	Inserção de Novos Dados	Inserção de novos dados nos registros antigos dos clientes	0,20	0,80	0,16

Tabela 1. Matriz de Riscos do Projeto Arriscado

devem ser executadas. A definição das técnicas e métodos de identificação dos riscos, a forma de comunicação do conhecimento sobre os riscos na organização, a estruturação das informações sobre os riscos são algumas das tarefas assinaladas para o Planejamento da Gerência de Riscos.

Uma vez conhecidos os riscos do projeto, atividades e estratégias para mitigação são estudadas. O monitoramento deve ser contínuo promovendo o acompanhamento do plano, inicialmente traçado, e permitindo a tomada de ações corretivas quando necessário.

O trabalho de Boehm foi base para a grande maioria dos trabalhos em gerência de riscos na área de Engenharia de Software, apresentados ao longo da década de 90. Suas contribuições estabeleceram a gerência de riscos como um importante campo de estudo e introduziram algumas medidas-chave para o risco, sintetizando um conjunto de técnicas em um simples framework para o gerenciamento de riscos de projetos.

Nas décadas de 80 e 90 não existia uma preocupação associada com a melhoria do processo de desenvolvimento de software, logo o modelo de Boehm possui esta deficiência e também não viabiliza sua aplicação em múltiplos projetos.

A abordagem apresentada por Boehm é baseada na quantificação de riscos em projetos, utilizando o termo exposição de risco como uma medida de risco, também chamado de impacto do risco (risk impact) ou fator de risco (risk factor).

O modelo de Boehm trata cada projeto individualmente, embora indique explicitamente a utilização do conhecimento e experiência passada em projetos anteriores.

Gerência de riscos segundo Robert Charette

O trabalho desenvolvido por Robert Charette é complementar ao trabalho desenvolvido por Boehm. Em 1988, Robert Charette publicou seu primeiro livro sobre a Gerência de Riscos na área de Engenharia de Software [Charette 1990].

Robert Charette foi o primeiro a considerar a Gerência de Riscos como um processo contínuo e dinâmico.

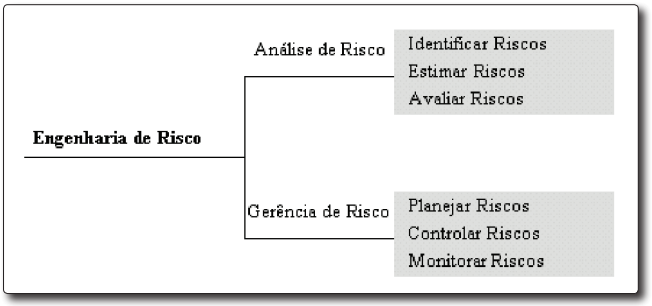
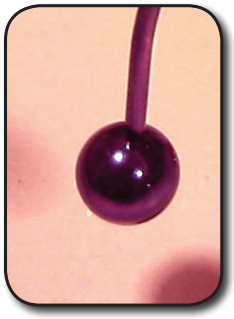


Figura 3. Modelo de Engenharia de Risco [Charette 1990]



Robert Charette e Barry Boehm trabalharam, na mesma época, no Departamento de Defesa dos Estados Unidos, desta forma os modelos propostos por ambos são, em muitos aspectos, similares e tomam como base suas experiências com a utilização do modelo cascata.

Da mesma forma que Boehm, o modelo apresentado por Charette, chamado de Engenharia de Risco, também é dividido em duas grandes fases: Análise de Risco e Gerência de Risco, conforme a Figura 3 [Charette 1990].

A Análise de Risco é composta pelas atividades de identificação, estimativa e avaliação de riscos gerando ao final um plano para a redução dos riscos identificados. Já na Gerência de Risco, a atividade de planejamento define as estratégias a serem utilizadas para a criação das respostas aos riscos (mitigação) e, por fim, a atividade de monitoração que tem o objetivo de acompanhar a execução do projeto e realizar as devidas correções, de acordo com o plano definido.

O modelo de Engenharia de Risco é uma evolução nos processos de gerenciamento de riscos, pois incorpora critérios de qualidade. Apresenta um processo contínuo e dinâmico através de uma espiral e faz a diferenciação entre riscos e oportunidades, mas não trata questões associadas a mais de um projeto em um ambiente de desenvolvimento de software.

Gerência de riscos do Instituto de Engenharia de Software (SEI)

De acordo com o SEI (Software Engineering Institute), as finalidades da gerência de riscos são identificar riscos assim que possível; ajustar a estratégia do desenvolvimento para diminuir estes riscos; e desenvolver e executar um processo de Gerência de Riscos como parte integral do processo padrão de software da organização.

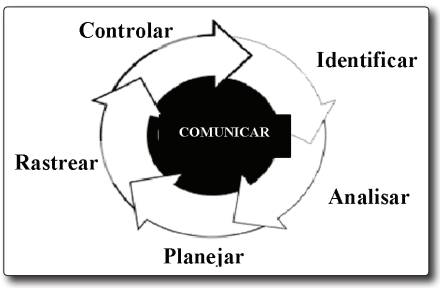


Figura 4. Modelo de Gerência de Riscos do SEI

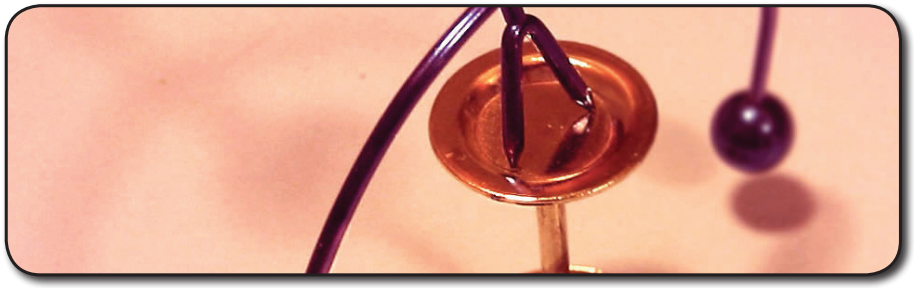
O modelo de Gerência de Riscos do SEI é uma evolução dos modelos propostos por Boehm [Boehm 1991] e Charette [Charette 1990], além de considerar critérios de qualidade total através de características do ciclo PDCA (Plan-Do-Check-Act) [SEI 2001].

O modelo define um conjunto de atividades que devem ser realizadas continuamente através do ciclo de vida de um projeto. O SEI enfatiza que a Gerência de Riscos deve ter uma gestão contínua no processo de desenvolvimento de software.

O modelo é composto por um conjunto de atividades, conforme a Figura 4 e está dividido em duas partes:

- Paradigma – composto pelas atividades do processo de Gerência de Riscos.
- Programas – composto pelas atividades de avaliação de riscos de software (levantamento inicial dos riscos), Clínica de Riscos (análise, planejamento, acompanhamento e controle), Clínica de Riscos em Equipe (utilização de recursos humanos para gerir os riscos – desenvolvedor e cliente) e o Gerenciamento Contínuo de Riscos (ciclo de vida).

A proposta do SEI está baseada fortemente na comunicação. A finalidade é fornecer informações dos resultados de ações internas e externas ao projeto nas atividades do risco (riscos atuais e riscos emergentes).



- Identificar: usa taxonomia de riscos para identificar os riscos potenciais.
- Analisar: transforma os dados obtidos em informação útil para a tomada de decisão.
- Planejar: planeja as estratégias relacionadas às alternativas de mitigação dos riscos identificados, priorizando-as e integrando-as ao plano de gerência de riscos definido.
- Rastrear: monitora a situação do risco ao longo do desenvolvimento e suas ações de mitigação através do uso de métricas e métodos de rastreamento.
- Controlar: executa ações corretivas definidas no plano de mitigação para corrigir os possíveis desvios.

O elemento central do modelo de gerência de riscos do SEI é a taxonomia de riscos e seu método de aplicação. O método é composto por um questionário e a documentação dos riscos está baseada na condição da ocorrência do risco e na conseqüência associada.

Como qualquer outro método de coleta de dados, o uso de questionário apresenta pontos fortes e fracos. Algumas vantagens comumente mencionadas são: pode-se obter uma boa amostra da população, a confidencialidade é garantida, não há pressão por parte do entrevistador, a tabulação de dados pode ser feita com maior facilidade e rapidez que outros instrumentos (entrevista, por exemplo), o custo é reduzido (não há necessidade de entrevistadores), rapidez e ausência de vieses por parte dos entrevistadores na aplicação. Como desvantagem principal, todos citaram a percentagem pequena de devoluções, ou seja, poucas são as pessoas que participam do preenchimento de pesquisas presenciais ou mesmo através da internet [Alves-Mazzotti e Gewandsznajder 1999].

Através dos levantamentos dos riscos novos insumos e novas necessidades de classificação são identificados. Estes podem ser acrescentados à estrutura existente ou reformulados em uma nova estrutura da taxonomia especificada, mas é preciso ter um processo de manutenção bem definido (automatizado), pois, caso contrário, os investimentos feitos na sua elaboração, bem como os benefícios decorrentes da implementação, podem ser perdidos.

O modelo de gerência de riscos do SEI foi testado e adaptado às condições reais do desenvolvimento do software e é utilizado atualmente para projetos de grande porte, mas apresenta como dificuldades a dependência de consultores externos e a não existência de ferramenta computacional que integre todos os conceitos de forma a tornar as atividades mais produtivas.

Gerência de riscos segundo o Guia PMBOK

O PMI (Project Management Institute) é uma associação de profissionais de gerenciamento de projetos que existe desde 1969. Esta associação criou em 1986 a primeira versão do Guia PMBOK (Project Management Body of Knowledge Guide), guia que descreve a somatória de conhecimento e as melhores práticas dentro da profissão de gerenciamento de projetos.

A meta do gerenciamento de projetos, segundo o Guia PMBOK, é conseguir exceder as necessidades e expectativas dos stakeholders. O Guia PMBOK organiza os processos de gerenciamento em cinco grupos: iniciação, planejamento, execução, monitoramento e controle e encerramento.

O Guia PMBOK está baseado em boas práticas para o gerenciamento de projetos. Em particular, a disciplina de gerência de riscos tem por base o processo apresentado pelo SEI.

Os processos propostos pelo Guia PMBOK estão organizados em nove áreas de conhecimento que se referem a aspectos considerados dentro do gerenciamento de projetos: Integração, Escopo, Tempo, Custo, Qualidade, Recursos Humanos, Comunicação, Riscos e Aquisições.

A gerência de riscos inclui os seguintes processos [PMI 2004]:

- Planejamento da Gerência de Riscos: tem como objetivo determinar qual a abordagem de gestão de riscos a ser utilizada e planejar as atividades de gerência que serão executadas para o projeto.
- Identificação dos Riscos: tem a finalidade de determinar quais riscos são mais prováveis de afetar o projeto e documentar as características de cada um.
- Análise Qualitativa dos Riscos: tem a finalidade de realizar uma análise qualitativa dos riscos e das condições para priorizar seus efeitos nos objetivos do projeto.
- Análise Quantitativa dos Riscos: tem como motivação medir a probabilidade de ocorrência e as conseqüências dos riscos e estimar suas implicações nos objetivos do projeto.
- Planejamento das Respostas aos Riscos: tem como objetivo disponibilizar respostas às mudanças nos riscos no decorrer do projeto, através da definição de planos de contingências.
- Monitoração e Controle dos Riscos: tem como principal finalidade monitorar riscos residuais, identificar novos riscos, executar planos de redução de riscos e avaliar seus efeitos através do ciclo de vida do projeto.

Todas as atividades do processo de gerência de riscos interagem fortemente com as atividades das outras áreas de conhecimento do Guia PMBOK, tendo como objetivo principal aumentar a probabilidade de ocorrência dos eventos positivos, ou seja, as oportunidades e, minimizar os eventos adversos ao projeto, os riscos.

O Guia PMBOK já está em sua terceira edição onde apresenta algumas alterações na quantidade de processos: 7 (sete) novos processos foram adicionados, 13 (treze) processos foram renomeados e 2 (dois) processos removidos, totalizando 44 (quarenta e quatro) processos ao final.

Gerência de riscos segundo o MSF

O MSF (Microsoft Solutions Framework) foi criado em meados da década de 90. Seu objetivo era dar suporte à execução dos serviços de consultoria da Microsoft [Robin et al 2002].

O MSF foi definido com base no modelo do SEI – Processo Contínuo de Gerência de Riscos (Continuous Risk Management Process) somada `a experiência das equipes da Microsoft (Microsoft Consulting Services e Microsoft Partners) [Robin et al 2002].

O MSF prega uma gerência de riscos proativa através da avaliação contínua de riscos e a da integração das informações de suporte à tomada de decisão dentro do ciclo de vida do projeto. O processo é composto de seis fases:

- Identificação: permite a identificação dos principais eventos que podem ser futuros problemas na execução do projeto.
- Análise e Priorização: transforma os dados e valores levantados para cada risco identificado, de forma que a equipe possa priorizá-lo.
- Planejamento e Cronograma: define ações estratégicas para evitar os riscos priorizados e o cronograma garante que estas ações serão implementadas.
- Rastreamento e Documentação: relaciona as atividades de rastreamento que garantem um constante acompanhamento dos planos de ações e possíveis correções necessárias de acordo com os desvios encontrados. A documentação facilita a comunicação da equipe, mantendo-os informados sobre a situação de cada um dos riscos.
- Controle: executa conjunto de atividades dos planos de ações e documenta corretamente os desvios ocorridos.
- Aprendizado: formaliza as lições aprendidas, os artefatos relevantes do projeto e ferramentas.

O processo proposto pelo MSF possui conceitos pré-definidos e institucionalizados, sendo desenvolvido e instanciado para a Microsoft, logo não existe adaptação do processo de gerência de riscos. Desta forma, ele não apresenta a preocupação em planejar estratégias para a gerência de riscos. Além de ser um modelo descritivo.

Com relação ao tratamento de riscos, o MSF é bastante explícito nas questões relacionadas ao conjunto de atividades definidas para o Controle e Monitoração de Riscos.

Não aborda questões relacionadas aos riscos entre projetos, muito embora trate a importância e benefícios ora adquiridos pelas práticas relacionadas à gerência de portfólio de projetos.

Gerência de riscos segundo DMAIC

O DMAIC é uma metodologia constituída por cinco fases básicas: Definir (Define); Medir (Measure); Analisar (Analyze); Melhorar (Improve); Controlar (Control). É considerado um "processo incremental para melhorias", utilizado por um grande número de empresas que estão adotando o programa Seis Sigma nos Estados Unidos.

Em 1986 Bill Smith, engenheiro e cientista da Motorola, introduziu o conceito da metodologia Seis Sigma. Inicialmente foi definido como métricas para medição de defeitos e melhoria da qualidade; e uma metodologia para diminuir os níveis de defeitos.

O DMAIC é baseado no Ciclo PDCA (Plan-Do-Check-Act), logo é considerado um método de solução de problemas que integra, de forma lógica, um conjunto de ferramentas para coleta, processamento e disposição das informações necessárias para a execução de cada uma das fases.

O DMAIC é aderente ao grupo de processos do Guia PMBOK. O modelo DMAIC promove análise e melhoria do desempenho dos negócios de forma sistemática e é composto por cinco fases:

- Definir oportunidades: visa identificar e/ou validar a oportunidade de melhoria, desenvolver o processo do negócio, definir os requisitos críticos do cliente e preparar para ser uma equipe de projeto altamente eficaz.
- Medir o desempenho: identifica medições críticas que são necessárias para avaliar o sucesso na satisfação dos requisitos mínimos do cliente e começar a desenvolver a metodologia para coletar dados eficazmente para medir o desempenho do processo.
- Analisar as oportunidades: estratifica e analisa a oportunidade para identificar um problema específico e definir uma declaração de problema facilmente compreensível.
- Melhorar o desempenho: identifica, avalia e seleciona as melhores alternativas de soluções para melhoria. Desenvolver uma abordagem de administração de mudança para ajudar a organização a se adaptar às mudanças introduzidas através da implementação da solução.
- Controlar o desempenho: compreende a importância de planejar e executar com base no plano estipulado e determinar a abordagem a tomar para garantir o alcance dos resultados definidos. Entender como disseminar as lições aprendidas, identificar oportunidades e processos para replicação e padronização, e desenvolver planos relacionados.

A seguir são apresentadas as atividades previstas na abordagem DMAIC para a gerência de riscos, conforme apresentado por Torres [Torres et al. 2005]:

- Definição de Riscos: tem a finalidade de identificar quais são os riscos que podem afetar o projeto.
- Mensurar os Riscos: com base na identificação realizada pela Definição de Riscos, esta atividade tem o objetivo de determinar o impacto e a probabilidade da ocorrência dos riscos.
- Analisar os Riscos: analisa detalhadamente os riscos identificados, quantificando-os.
- Melhoria e Controle: planeja ações voltadas para a melhoria e controle do progresso.

A metodologia DMAIC busca alcançar oportunidades de melhoria através da abordagem projeto a projeto. Já o Guia PMBOK trata da gestão de projetos, com forte base em planejamento e controle do tempo de execução e dos recursos utilizados, não possuindo os elementos de análise e melhoria de que dispõe o DMAIC.

Existem alguns exemplos de aplicação da metodologia DMAIC no Brasil, nas áreas de financeira e varejo respectivamente, como no Citibank, Wal-Mart e Carrefour.

A metodologia DMAIC pode ser utilizada tanto em pequenas como em grandes corporações, não sendo o porte da empresa um limitante para a decisão de investir ou não na metodologia.

Gerência de riscos segundo RUP

O RUP (Rational Unified Process) é um conjunto de processos da engenharia de software que tem por base o uso das melhores práticas voltadas para o desenvolvimento de software e de princípios fundamentais (direcionado a casos de uso, centrado na arquitetura, direcionado a riscos e interativo).

O RUP está disponibilizado na forma de software, fornecido pela Rational Software, e através de um conjunto de processos.

Como citado anteriormente, o RUP é uma metodologia de desenvolvimento, com uma estrutura formal bem definida. Como qualquer metodologia, é composta de conceitos, práticas e regras.

Um dos principais pilares do RUP é o conceito de melhores práticas, que são regras que visam reduzir o nível de risco (existente em qualquer projeto de software) e tornar o desenvolvimento mais eficiente e eficaz. O RUP define seis melhores práticas:

- Desenvolver iterativamente: significa desenvolver em ciclos. Cada ciclo contém um objetivo que deve ser alcançado (lançamento de uma versão beta, correção de um bug, etc).
- Gerenciar requisitos: provê uma maneira prática de produzir, organizar e comunicar os requisitos de um projeto. Somado a isso, os casos de uso e cenários descritos nos processo são técnicas excelentes para capturar e assegurar requisitos.
- Utilizar arquiteturas baseadas em componentes: foca o desenvolvimento em módulos do sistema, através do uso de componentes, como forma de criar um sistema adaptável, intuitivamente entendível e reutilizável.
- Modelar visualmente: permite melhor entender não só a concepção e a complexidade do sistema, mas também "dimensionar" (no sentido de qual a forma do sistema), além de facilitar a identificação e solução de problemas.
- Verificação contínua de qualidade: verifica critérios de qualidade relacionados ao produto e ao processo.
- Controle de mudanças: como resultado de um processo de desenvolvimento iterativo, muitas são as mudanças ocorridas no decorrer do projeto. Controle de mudanças durante todo o projeto, como forma de manter a qualidade do projeto.

De acordo com o RUP [Kruchten 2003], os riscos devem ser identificados e tratados o quanto antes no ciclo de vida do projeto, sempre com o objetivo de garantir a produção de software de alta qualidade, de acordo com as necessidades dos usuários dentro do tempo e do prazo previstos. Todo projeto de desenvolvimento de software tem um conjunto de riscos envolvidos e muitos deles não são identificados até que a integração do sistema seja realizada.

O RUP é composto por duas dimensões. A primeira representa a estrutura estática do processo, descrevendo como os elementos do processo são agrupados logicamente em disciplinas. Por sua vez, as disciplinas são agrupamentos lógicos de papéis, atividades, artefatos e outros guias para a descrição de um processo, e são representadas por um fluxo de trabalho. A segunda é a dinâmica que é representada pelo tempo e expressa o processo por meio de ciclos, decompostos em fases, que são divididas em iterações com marcos (milestones) de conclusão.

O processo de desenvolvimento é dividido em ciclos, sendo que o ciclo de desenvolvimento é subdividido em quatro fases consecutivas. Estas fases são concluídas tão logo um marco é alcançado. Um marco pode definir uma etapa, na fase, na qual decisões críticas são realizadas e/ou objetivos são alcançados.

As fases do RUP são:

- Concepção: foco na discussão do problema, definição do escopo do projeto, estimativa de recursos necessários para a execução do projeto, entre outros. É nesta fase que é apresentado o plano de projeto com foco no tratamento dos riscos relacionados aos casos de negócio.
- Elaboração: foco na análise do domínio do problema, desenvolvimento do plano de projeto, estabelecendo a fundação arquitetural e eliminando os elementos de alto risco. Os elementos de risco a serem analisados, nesta fase, são os riscos de requisitos, tecnológicos (referentes à capacidade das ferramentas disponíveis), de habilidades (dos integrantes do projeto) e políticos.
- Construção: foco nos riscos de "logística" e na obtenção da conclusão da maior parte do trabalho. Esta fase compreende a fase de modelagem e a fase de desenvolvimento em si, aquela em que o sistema é efetivamente programado.
- Transição: foco nos riscos associados com a logística de entrega do produto.

O RUP é uma metodologia de desenvolvimento de software baseada fortemente na análise de riscos. As atividades de gerenciamento de riscos estão na disciplina de gerenciamento de projeto.

O processo de desenvolvimento é iterativo e aborda um projeto por vez. O RUP mostra como aplicar várias práticas

Modelo	Referência	Componentes e Relacionamentos
Barry Boehm	- Modelo Espiral - Experiência no Departamento de Defesa Norte-Americano – Modelo Cascata	O modelo é composto por duas grandes etapas: Avaliar Riscos e Controlar Riscos
Robert Charette	- Experiência no Departamento de Defesa Norte-Americano – Modelo Cascata	O modelo de Engenharia de Risco é composto por: Análise de Risco e Gerência de Risco
SEI	- Barry Boehm - Robert Charette - Ciclo PDCA	Duas grandes fases: Paradigma, Programas. Está fortemente pautado em Comunicação
Guia PMBOK	- Boas práticas da Gerência de Projetos - Modelo SEI - Ciclo PDCA	Apresenta seis processos
MSF	- Modelo SEI	Composto por seis fases
DMAIC	- Ciclo PDCA	O modelo DMAIC é uma metodologia sistemática para análise e melhoria do desempenho dos negócios, que consiste em cinco fases
RUP	- Modelo Espiral - Modelo Cascata - Análise de riscos	O modelo é composto por fases e estas têm iterações que possuem workflow e geram artefatos

Tabela 2. Visão dos Modelos de Gerência de Riscos segundo Critérios de Análise

da engenharia de software. Também provê mentoring para a utilização de diversas ferramentas de automatização do processo específico de engenharia de software. Algumas destas práticas envolvem desenvolvimento iterativo, gerenciamento de requisitos, arquitetura baseada em componentes, modelagem visual, melhoria de qualidade contínua e gerenciamento de mudança.

Estudo analítico

Com o intuito de permitir uma melhor visibilidade, a Tabela 2 apresenta uma síntese da análise dos modelos e abordagens de gerência de riscos apresentados nesta matéria e suas respectivas características de acordo com os critérios de referência, componentes e relacionamentos.

Inicialmente, com base nos modelos de referência, verifica-se que a literatura analisada segue uma evolução do modelo apresentado por Boehm, desde a década de 80 até o final da década de 90. Além de alguns serem desenvolvidos com base em necessidades específicas da época como a preocupação com a evolução e melhoria de processos.

De posse dos resultados, individualmente para cada um dos modelos apresentados, nota-se a predominância das abordagens voltadas para a utilização dos conceitos apresentados pelo modelo do Instituto de Engenharia de Software, com exceção ao modelo do RUP, uma vez que é uma metodologia de desenvolvimento e integra, em uma das suas disciplinas, a área de gerenciamento de riscos.

O modelo do SEI possui um processo definido e especificado, embora necessite de um grande esforço organizacional para sua implementação e implantação, sendo assim, é considerado um modelo complexo.

Ainda com relação ao modelo do SEI, bem como o Guia PMBOK e o DMAIC, existe a preocupação com a melhoria dos processos, pois todos estão pautados nos conceitos do Ciclo PDCA. Esse é um aspecto relevante, pois permite que a organização evolua seus processos, através da análise dos resultados e da aplicação de ações corretivas tanto no projeto quanto no processo.

Os modelos apresentados por Boehm e Charette são uma conseqüência da época de seus desenvolvimentos (década de 80), onde não era evidenciada a preocupação com a adaptação e melhoria de processos.

O modelo apresentado pelo MSF não permite a adaptação do processo de gerência de riscos, uma vez que possui conceitos pré-definidos e institucionalizados, além de ser um modelo instanciado para a empresa.

Também com base no estudo dos modelos foi desenvolvida a Tabela 3, onde é apresentada uma análise das atividades apresentadas por cada um dos modelos.

Os modelos de gerência de riscos apresentados possuem algumas similaridades em relação à nomenclatura utilizada para a descrição das atividades e tarefas, muito embora alguns deles, como os de Boehm e Charette, não possuam a preocupação de evolução dos processos.

Dos modelos analisados apenas dois apresentam a atividade Planejar a Gerência de Riscos. O Guia PMBOK foca na definição das estratégias de gerenciamento dos riscos de acordo com os objetivos organizacionais. O RUP, por sua vez, trata a atividade dentro do fluxo de trabalho (workflow) Planejamento de Projetos, objetivando a definição de um plano de gerência de riscos.

A atividade Identificar Riscos é uma constante em todos os modelos até porque não se pode gerenciar o que não se conhece.

A atividade Analisar Riscos também aparece em todos os modelos, embora varie a terminologia em alguns casos. Os modelos de Charette e DMAIC enfatizam a necessidade de medição quando explicitam as atividades de mensurar e estimar riscos. Já os modelos de Boehm e MSF expressam uma preocupação com a priorização dos riscos, envolvendo uma análise detalhada do impacto do risco no projeto e da probabilidade de ocorrência.

A atividade Planejar Respostas aos Riscos não aparece explicitamente apenas no modelo DMAIC. Ela está internamente relacionada às tarefas de análise e medição.

A atividade Monitorar Riscos aparece em todos os modelos, embora os modelos do SEI e MSF tratem esta atividade através do rastreamento de riscos. O modelo do DMAIC não apresenta explicitamente esta atividade, mas menciona tarefas associadas na atividade de Melhoria e Controle.

Controlar Riscos também é uma preocupação unânime nos modelos. No modelo de Boehm não existe uma atividade com este nome, mas internamente a atividade Monitorar Riscos, proposta em seu modelo, critérios de controle são destacados.

A comunicação é um elemento importante dentro dos ambientes de desenvolvimento de software. Desta forma, todos os modelos são pautados no uso de comunicação.

Considerações finais

Este artigo tem a finalidade de avançar nos conceitos fundamentais da área de gerência de riscos apresentando um estudo sobre as aplicações em modelos de gestão e qualidade de software.

É importante ressaltar que esses modelos e abordagens figuram como base para a grande maioria dos modelos de processos atualmente apresentados (desde 2002), principalmente os modelos de qualidade de software.

Com base nestes conceitos, nas próximas edições, serão apresentadas algumas ferramentas, técnicas e métodos de apoio e por fim, serão tratadas algumas tendências, como maturidade de projetos, portfólio e gerenciamento de riscos em ambientes de múltiplos projetos.

Dê seu feedback sobre esta edição!

A Engenharia de Software Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/esmag/feedback

Referências

[Alves-Mazzotti e Gewandsznajder 1999] Alves-Mazzotti, A.J.; Gewandsznajder, F. (1999) O Método nas Ciências Naturais e Sociais – Pesquisa Quantitativa e Qualitativa. São Paulo: Pioneira. pp 65-81.

[Boehm 1988] Boehm, B. W. (1988) A Spiral Model of Software Development and Enhancement, IEEE Software. Technical Report 0018-9162/88. pp 61-72.

[Boehm 1991] Boehm, B. W. (1991) Software Risk Management: Principles and Practices, IEEE Software, Volume 8. No1. pp 32-40.

[Boehm et al 2004] Boehm, B. W.; Brown, A. W; Basili, V; Turner, R. (2004) Spiral Acquisition of Software-Intensive Systems of Systems, Cross talh – The Journal of Defense Software Engineering, DoD – Department of Defense. pp 4-9.

[Charette 1990] Charette, R. (1990) Application Strategies for Risk Analysis. New York: MultiScience Press. pp 17-21.

[Kerzner 2000] Kerzner, H. Applied project management: best practices on implementation. Nova York: John Wiley & Sons Inc., 2000

[Kruchten 2003] Kruchten, P. (2003) Introdução ao Rup: Rational Unified Process. 2ª Ed. Ciência Moderna. São Paulo. pp 25-36.

[PMI 2004] PMI - Project Management Institute. (2004) A Guide to the Project Management Body of Knowledge – ANSI/PMI 99-01-2004. Project Management Institute. Four Campus Boulevard. Newtown Square. USA.

[Robin et al 2002] Robin, A.; Preedy, D.; Campbell, D.; Paschino, E. and Hargrave, L. (2002) Microsoft Solutions Framework. MSF Risk Management Discipline v.1.1. White Paper. Microsoft Solutions Corporation. USA.

[SEI 2001] SEI - Software Engineering Institute. (2001) CMMI - Capability Maturity Model Integration version 1.1 Pittsburgh, PA. Software Engineering Institute, Carnegie Mellon University. USA.

[Torres et al 2005] Torres, J.; Boesso, R. e Sant'Anna, M. (2005) Gerenciamento de Riscos: Planejamento ou Futurologia? In: MundoPM – Project Management – número 4. pp 43-47.

MODELO	ATIVIDADES						
	Planejar a Gerência de Riscos	Identificar Riscos	Analisar Riscos	Planejar Respostas aos Riscos	Monitorar Riscos	Controlar Riscos	Comunicar os Riscos
Barry Boehm	Não faz menção a esta atividade	Identificar Riscos	- Analisar Riscos - Priorizar Riscos	- Planejar a Gerência de Riscos - Resolver Riscos	Monitorar Riscos		Comunicação implícita
Robert Charette	Não faz menção a esta atividade	Identificar Riscos	- Estimar Riscos - Avaliar Riscos	Planejar Riscos	Monitorar Riscos	Controlar Riscos	Comunicação implícita
SEI	Não faz menção a esta atividade	Identificar Riscos	Analisar Riscos	Planejar	Rastrear Riscos	Controlar Riscos	Comunicar os Riscos
Guia PMBOK	Planejar a Gerência de Riscos	Identificar Riscos	- Análise de Riscos Quantitativa - Análise de Riscos Qualitativa	Planejar Respostas aos Riscos	Monitorar e Controlar Riscos		Comunicação implícita
MSF	Não faz menção a esta atividade	Identificação de riscos	Análise e Priorização de Riscos	Planejamento e Cronograma	Rastreamento de Riscos	Controle de Riscos	Aprendizado Documentação de Riscos
DMAIC	Não faz menção a esta atividade	Definição dos Riscos	- Mensurar os Riscos - Analisar Riscos	Analisar Riscos	Melhoria e Controle		Comunicação implícita
RUP	No Planejamento do Projeto – Desenvolver o Plano de Gerenciamento de Riscos	Na avaliação do Escopo e dos Riscos – Identificar e Avaliar os Riscos			No monitoramento e controle do Projeto – Monitorar o Status do Projeto		Comunicação implícita

Tabela 3. Análise Comparativa dos Modelos segundo Atividades do Processo de Gerência de Riscos

local
Data Center

Out Sourcing Total

Preparados para nova era em qualidade de serviços, garantimos operações full-time dentro de uma gestão única desenvolvida pela nossa equipe para atuar preventivamente, somos a única empresa a operar por 25hs por dia garantindo disponibilidade total.

Escalabilidade, Conectividade e Disponibilidade.

- Servidor Dedicado
- Gerenciamento
- Banda IP
- Cross Connection
- Link PAP
- Serviços compartilhados
- Sites
- E-mail
- Backup

CERTIFICAÇÕES

- MICROSOFT
- LINUX
- CISCO
- ITIL

Data Center Próprio

Segurança e certeza
de bons negócios e serviços

contato@localdatacenter.com.br
+55(21) 3527-6510

Identificação e Especificação de Componentes de Negócio

Como identificar e especificar componentes de negócio usando como base casos de uso e diagramas UML

Marcelo C. Araújo
mrclcst@bol.com.br
Atua no ramo de engenharia de software há 6 anos, trabalhando com customização de metodologias basedas em UP(Unified Process) e participação em vários projetos nos papeis de: analista de processo de negócio, especificador de requisitos, analista de sistema e projetista em soluções JEE de alta criticidade.

Componentes partem do princípio da divisão e conquista - dividir um grande problema em partes menores e resolver essas partes menores individualmente. Ao final, o problemão terá sido resolvido, parte a parte. Pois, quando focamos o trabalho em um problema menor, podemos pensar em soluções específicas para ele.

E qual a novidade disso? Afinal, dividir para conquistar é um conceito utilizado desde que a programação estruturada foi criada.

A programação estruturada usava a idéia de dividir para conquistar na forma de decomposição funcional. Porém, os dados relacionados a essas funções não recebiam o mesmo tratamento. Eles eram mantidos em grandes bases, responsáveis por dados relacionados a diversas funcionalidades. Dessa forma o reuso e a gerência de mudanças ficavam extremamente comprometidos. Com isso, não só as funções eram bastante dependentes entre si, como os dados também.

A diferença da abordagem estruturada para a que utilizamos aqui pode ser resumida na união entre funções e dados relacionados, conceito que foi introduzido pela orientação a objetos.

A orientação a objetos tratou de minimizar o problema da dependência com a utilização de classes para unir funções e dados correlatos.

Um componente de negócio aglomera um conjunto de classes e dados altamente relacionados, em uma mesma unidade. De forma análoga à idéia de classe da orientação a objetos.

Devido a essa característica, arquiteturas baseadas em componentes de negócio oferecem uma maior capacidade de reuso, afinal, soluções baseadas em componentes geralmente são formuladas com o intuito de reutilizar código. Uma arquitetura que não se preocupa com reuso, tende a tratar o projeto de cada aplicação como algo individual e isolado. Seguindo essa linha, é enorme a possibilidade de

encontrarmos funcionalidades redundantes entre as várias aplicações existentes em uma empresa, como pode ser visto na Figura 1, que demonstra um exemplo onde aplicações contêm funcionalidades redundantes entre si, caracterizando essas redundâncias na forma de porcentagem.

Entretanto, o maior objetivo a ser alcançado quando se projeta um componente de negócio é atingir um alto nível de flexibilidade no gerenciamento de mudanças.

Sempre que a lógica de um determinado processo de negócio informatizado sofre uma modificação, a estrutura de software responsável por manter essa lógica deverá, por conseqüência, ser modificada também. Se toda essa lógica for projetada em forma de componentes de negócio, com suas dependências bem formuladas e delimitadas, a troca de um componente por outro ou a modificação de regras internas de um componente é de longe bem menos traumática do que a manutenção tradicional em sistemas não componentizados.

Isso só é possível porque um componente só se comunica através de suas interfaces. As interfaces de um componente representam a especificação daquilo que ele se propõe a oferecer.

O desenvolvimento baseado em componentes se destaca das outras abordagens existentes justamente por separar a interface de sua implementação.

Como os componentes só se comunicam por meio de suas interfaces, suas implementações ficam isoladas, o que reduz o grau de dependência entre classes.

A Figura 2 exemplifica uma troca de componentes. Observe que a classe ClienteExistente não precisou sofrer qualquer tipo de modificação para suportar o novo componente.

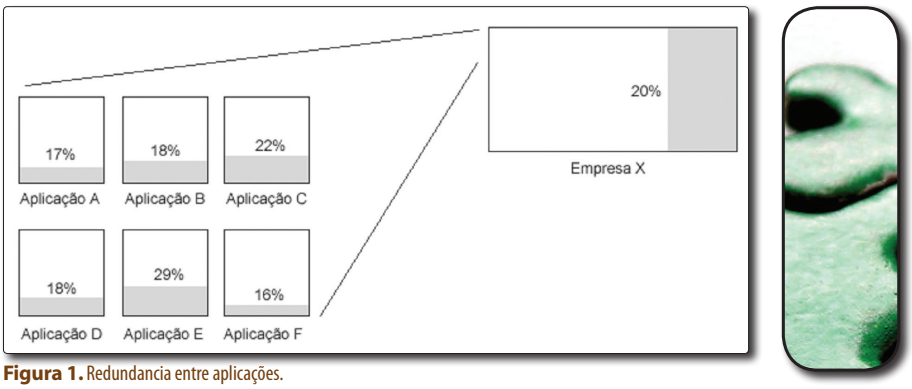


Figura 1. Redundancia entre aplicações.

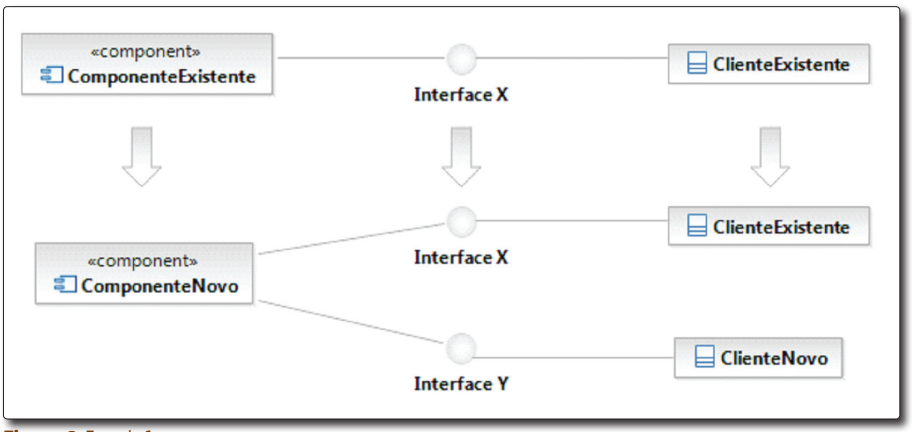


Figura 2. Troca de Componentes

Diretrizes gerais sobre componentes de negócio

Para identificar e especificar componentes de negócio se faz necessário o entendimento dos seguintes conceitos:

Subsystem

É um termo usado como sinônimo de "componente de negócio". Os subsystems encapsulam comportamento, expõem um conjunto de interfaces e empacotam um conjunto de elementos do modelo de design (ou modelo de projeto, também chamado algumas vezes de projeto físico).

Do lado de fora, um componente de negócio é um simples elemento do modelo de design que colabora com outros elementos do modelo, com o intuito de cumprir suas responsabilidades. As interfaces do componente são visíveis externamente e seu comportamento é documentado em sua especificação.

Do lado de dentro ficam uma coleção de elementos do modelo de design, como, classes, que realizam as interfaces do componente, bem como o comportamento atribuído a ele através de sua especificação.

System Interface

Ou interface do sistema, representa o mapeamento dos fluxos (passos) dos casos de uso de um sistema e também controla a chamada aos componentes de negócio. Ela é a ponte entre as necessidades sistêmicas representadas no caso de uso e as interfaces dos componentes de negócio. Dessa forma, a aplicação não fica diretamente conectada aos componentes de negócio que irão suprir suas necessidades.

Design by Contract

No âmbito dos componentes de negócio, o design by contract (projeto por contrato) se divide em duas faces: contrato de uso e contrato de realização.

Contrato de Uso

Um contrato de uso releva as relações que as interfaces do componente podem assumir com seus clientes. O cliente em si não é especificado, pois não é possível prever quais clientes poderão usar essa mesma interface no futuro.

O contrato de uso se divide em modelo da informação e operações.

- Modelo da Informação – A definição e as restrições de qualquer informação retida entre a solicitação do cliente por um objeto e a interface que suporta essa solicitação.
- Operações – Uma lista de métodos que a interface provê, incluindo suas assinaturas e definições. Nelas são definidas as entradas, as saídas e seus relacionamentos, alem de suas pré e pós-condições. Cada operação é tratada como um contrato de granularidade fina, o que significa algo mais específico, algo com pouco volume funcional.

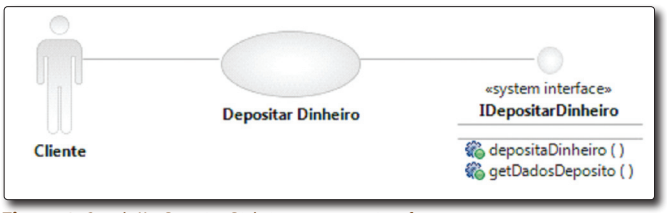


Figura 3. Caso de Uso Depositar Dinheiro e sua system interface

- Pré-condição - Define o que deve ser verdadeiro na estrutura da informação dos parâmetros de entrada para que a operação possa ser executada. Ou seja, uma operação só garante o seu funcionamento perfeito caso suas pré-condições sejam satisfeitas pelo cliente que solicita a operação.
- Pós-condição - Uma descrição dos efeitos da operação, em seus parâmetros de saída, caso existam, e no modelo de informação (dados) manipulado por ela.

Vale frisar que o cliente é quem deve assegurar o cumprimento de uma pré-condição antes de fazer a chamada a uma operação. Por acordo no contrato, é aceitável que se a pré-condição não for cumprida, o comportamento esperado como resposta da operação possa não acontecer. O resultado da operação passa a ser indefinido, ou seja, qualquer coisa pode acontecer.

Invariante de Classe

Uma invariante de classe é uma condição que todo objeto dessa classe deve satisfazer, enquanto o objeto estiver em equilíbrio. Um objeto é visto como "em equilíbrio" no instante antes e no instante após a execução de métodos da classe.

Por exemplo: Em uma classe Triangulo, cujos atributos são os lados a, b e c. Se determinarmos para essa classe a invariante (a+b>c), isso quer dizer que antes de se executar um método da classe e também após executar, (a+b>c) deve obrigatoriamente ser uma verdade, independente dos valores a serem atribuídos aos atributos.

No que tange os conceitos sobre componentes de negócio, caso uma invariante seja declarada para um componente, a veracidade da invariante deverá ser testada tanto na pré quanto na pós-condição das operações do componente.

Contrato de Realização

Um contrato de uso é um contrato em tempo de execução, mas um contrato de realização é um contrato em tempo de projeto (design). Ele é um contrato entre uma especificação do componente na parte de design (seu projeto físico) e uma implementação do componente (seu código fonte). Dessa forma, o contrato de realização é algo que deve ser respeitado pela pessoa que está criando a implementação: o programador. Visto que o contrato de realização é incorporado na especificação do componente.

Esse tipo de contrato serve para que o programador siga todas as decisões de projeto dos componentes, pois a especificação do componente pode conter definições de como interagir com outros componentes e definições para requisitos não funcionais.

Como identificar system interfaces e subsystems

Primeiramente devemos identificar as interfaces de sistema através dos casos de uso e em seguida os subsystems com base nos diagramas de análise OO.

Identificando system interfaces e suas operações

Para cada caso de uso será criado uma System Interface correspondente. O agrupamento das interfaces de sistema de todos os casos de uso de um sistema formará a system layer (camada de sistema), que será a ponte de comunicação entre os subsystems e seus clientes.

Vamos então formular um caso de uso para a identificação de uma system interface:

Caso de Uso: Depositar Dinheiro
Descrição: Este caso de uso permite fazer um deposito em dinheiro em uma conta corrente.
Ator Principal: Cliente
Pré-Condições do caso de uso: N/A (não se aplica)
Fluxo de Eventos Básico – Depositar:
Cliente informa agência, conta e valor.
Sistema solicita senha.
Cliente informa senha.
Sistema informa nome do correntista, agência, conta e valor para confirmação.
Cliente confirma depósito.
Sistema efetua depósito.
Fluxos de Eventos Alternativos: N/A
Pós-Condições do caso de uso: N/A
Regras:
1 - O valor informado deverá ser somado ao saldo atual da conta informada.
2 - Caso o correntista beneficiado pelo depósito seja assinante do serviço Aviso de Depósito, um e-mail deverá ser enviado a ele, informando: valor, data e hora do depósito.
Fluxos de Exceção das Regras: N/A
Mensagens dos Fluxos de Exceção: N/A

Na Figura 3 vemos que a partir dos passos do caso de uso Depositar Dinheiro, identificamos e definimos uma system interface chamada IDepositarDinheiro com duas operações. A primeira operação é para efetuar o depósito e a segunda para obter os dados do cliente que receberá o depósito - nome, agência e conta.

Identificando as Interfaces dos Componentes de Negócio

As interfaces de negócio são abstrações das informações que devem ser gerenciadas pelo sistema. O processo de identificação das interfaces de negócio deve seguir a seguinte linha:

1. Fazer a união dos diagramas de classes de análise de cada caso de uso para formar um diagrama único;
2. Identificar os Core Business Types (tipos de negócios centrais) a partir do diagrama de classe de análise único;
3. Criar as interfaces de negócio para cada Core Business Type;
4. Refinar o diagrama de classe único, indicando as responsabilidades das associações das interfaces de negócio.

Identificando os Core Business Types

No diagrama de classe de análise devemos identificar as classes que representam o negócio central do sistema. O propósito da identificação dos core business types é descobrir qual informação é dependente de outra e qual informação

não é dependente de outra. Esta é uma etapa muito útil para a alocação de responsabilidades das informações da interface de negócio.

Um core business type é um tipo de negócio que tem sua existência independente dentro do negócio. Se retirarmos do modelo todas as classes associadas à classe core, ela (a classe core) ainda preserva seu sentido de existência.

Observando as Figuras 4 e 5 vemos a identificação dos core business type. Seguindo as regras mencionadas anteriormente, as classes Agencia e Pessoa são consideradas cores por não terem dependência entre si e por possuírem existência independente no negócio. Assim, elas recebem o estereótipo <<core>>. As demais classes são dependentes umas das outras ou são dependentes das classes core, por isso elas recebem o estereótipo <<type>>. Com os core business type identificados, devemos atualizar o diagrama de classe com as interfaces de negócio. Para isso, veja a Figura 6.

Alocando as responsabilidades das associações entre interfaces de negócio

Quando existe uma associação entre types gerenciados por diferentes interfaces de negócio, nós as chamamos de uma Associação Inter-Interface. A associação entre as classes Conta e Pessoa é justamente esse tipo de associação. Uma decisão deverá ser tomada sobre quem será o responsável por manter essa associação (qual interface armazenará a referência do objeto que está ligado a outra interface e onde a integridade referencial será mantida). Nesse caso, devemos sempre nos decidir pelo maior grau de reuso em relação ao negócio. No exemplo da Figura 6 a classe Conta será gerenciada pela interface IGerenciaAgencia, pois a interface IGerenciaPessoa, em termos de negócio, é independente da classe Conta.

Associando a system layer aos subsystems

Como já foi dito, a partir dos casos de uso de um sistema, geramos as system interfaces. O conjunto de system interfaces de um sistema formam a system layer desse sistema.

Imaginando que o caso de uso Depositar Dinheiro pertença a um sistema chamado Caixa Eletrônico, teremos a

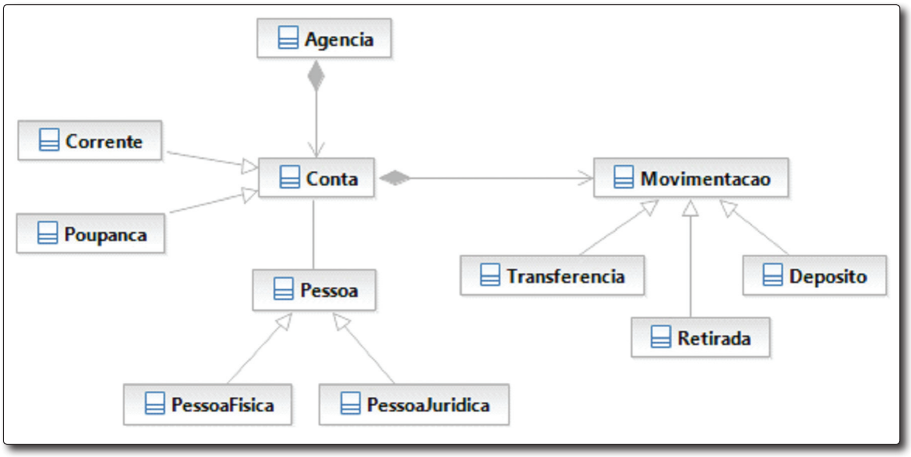


Figura 4. Diagrama de classe de análise único.

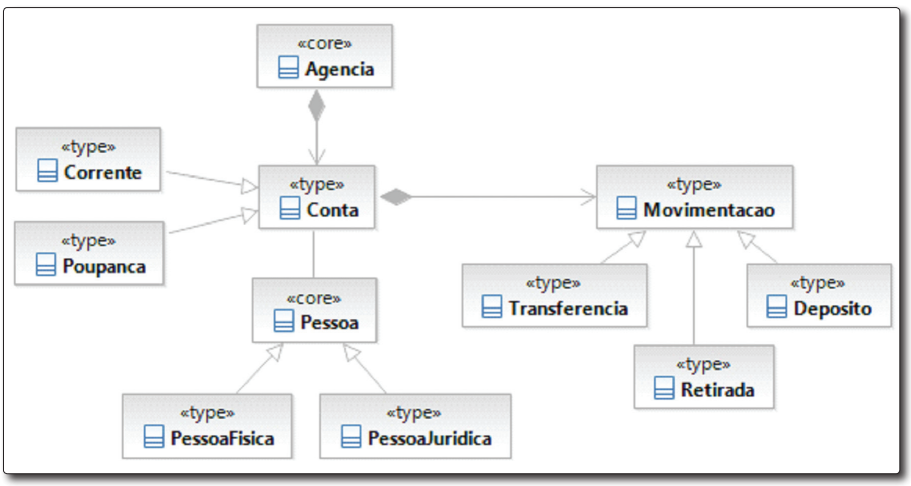


Figura 5. Diagrama de Classe de Análise – Identificação dos Core Business Type

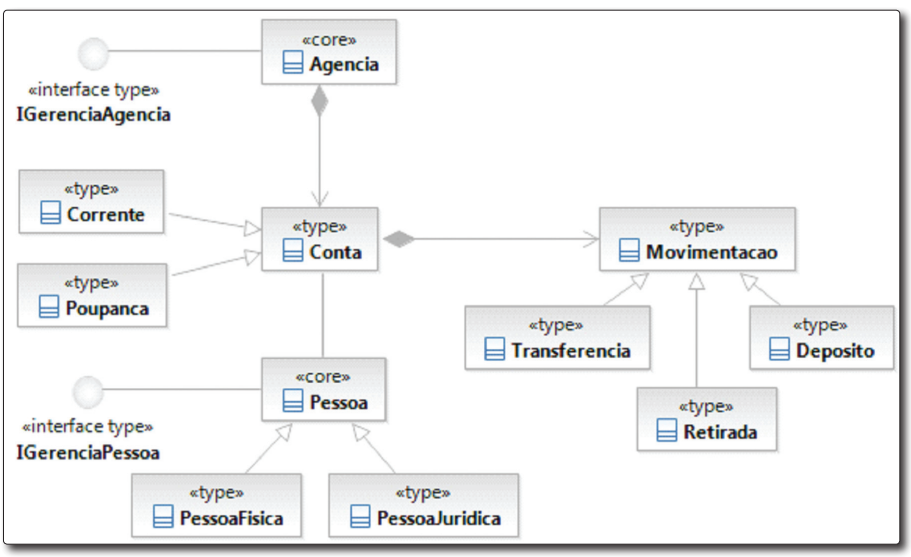


Figura 6. Diagrama de Classe de Análise com as interfaces de negócio

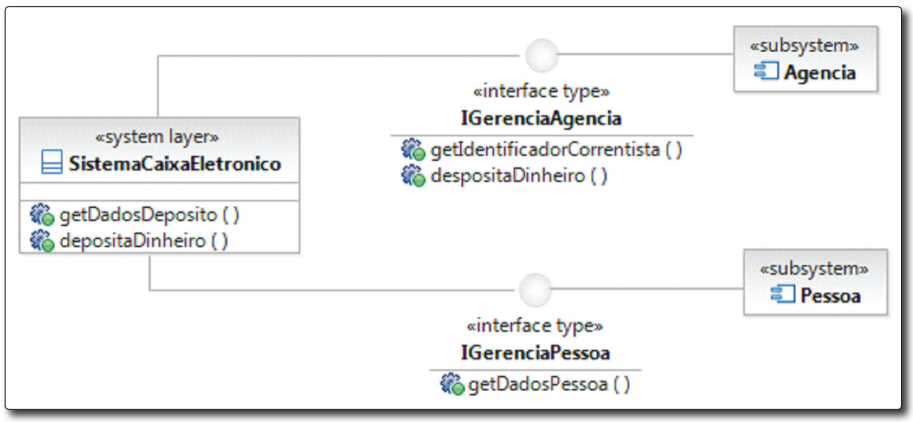


Figura 7. System layer associada as interfaces dos subsystems

configuração exemplificada pela Figura 7. Nessa figura vemos a ligação da camada de sistema SistemaCaixaEletronico (representando ali o conjunto de system interfaces do sistema) acessando as interfaces dos subsystems Agencia e Pessoa, para satisfazer os passos do caso de uso Depositar Dinheiro.

O subsystem Agencia contém as classes Agencia, Conta, Corrente, Poupanca, Movimentacao, Transferencia, Retirada e Deposito. Já o subsystem Pessoa contém as classes Pessoa, PessoaFisica e PessoaJuridica.

O modo como a system layer SistemaCaixaEletronico se comunica com os subsystems Agencia e Conta é demonstrado na Figura 8.

Perceba que o SistemaCaixaEletronico, com posse do número da agência e conta informados pelo cliente, solicita ao IGerenciaAgencia o identificador do correntista. Já de posse do identificador, a system layer usa essa informação para solicitar ao IGerenciaPessoa os dados do correntista, incluindo aí o seu nome.

Dessa forma, cumprimos o passo do caso de uso que diz "Sistema informa nome do correntista, agencia, conta e valor para confirmação".

Isso mostra que o componente Pessoa não é dependente do componente Agencia, e que este último só tem dependência de dados perante o componente Pessoa. Na solução da Associação Inter-Interface entre as classes Conta e Pessoa, enquanto o componente Pessoa é responsável pela classe Pessoa e seus dados (o que inclui seu identificador), o componente Agencia também mantém o identificador das "pessoas" que estiverem relacionadas às suas "contas", sob a forma de "identificador de correntista".

Especificando o componente de negócio

Nessa fase devemos formular o contrato de realização do componente. Que nada mais é do que detalhar melhor as responsabilidades do componente. Para tal devemos seguir uma série de passos. São eles:

Descobrir as operações de negócios

Nesse passo devemos utilizar as regras dos casos de usos para descobrir as operações de negócios relacionadas aos subsystems.

Utilizando como exemplo as regras 1 e 2 do caso de uso Depositar Dinheiro, deduzimos que a Regra 1 (o valor informado deverá ser somado ao saldo atual da conta informada) deverá ficar sob a responsabilidade da operação depositaDinheiro(), do subsystem Agencia.

Enquanto a Regra 2 (caso o correntista beneficiado pelo depósito seja assinante do serviço Aviso de Depósito, um e-mail deverá ser enviado...) deverá gerar duas novas operações no subsystem Agencia, como mostra a Figura 9.

Refinar as interfaces de negócio

Neste ponto, devemos levar em consideração as operações da system layer e suas finalidades. No decorrer da identificação das operações de negócios poderemos ter operações (assinaturas de métodos) diferentes com finalidades parecidas. Então, deveremos refatorá-las a ponto de não existir essa "duplicidade" de operações. O mesmo ocorre com as interfaces dos Subsystems.

Especificar as operações

Isso significa definir os parâmetros de entrada (informações passadas para o componente), parâmetros de saída (informações atualizadas ou retornadas pelo componente) e alterações no estado dos objetos do componente, para cada operação da interface do subsystem.

A especificação da operação getIdentificadorCorrentista() poderia ficar assim: getIdentificadorCorrentista (numeroAgencia: Integer, numeroConta: Integer): identificadorCorrentista.

Definir Pré-Condição e Pós-Condição

Nesse passo devemos, para cada operação da interface do subsystem, definir as pré-condições e as pós-condições, firmando assim o contrato do componente de negócio com o meio externo representado por seus clientes.

Usando novamente a operação getIdentificadorCorrentista(), sua pré e pós-condição seriam:

Pré-condição: numeroAgencia e numeroConta serem números inteiros válidos.

Pós-condição: identificadorCorrentista será igual ao número identificador do cliente vinculado à combinação numeroAgencia e numeroConta passados como parâmetros de entrada.

Em UML pré e pós-condições podem ser descritas usando OCL, que é uma linguagem declarativa usada na construção de expressões lógicas.

Conclusão

A única premissa que nunca muda no desenvolvimento de aplicações é a que diz: as coisas mudam!

Com processos de negócio bem elaborados e profissionais de TI bem capacitados, podemos criar ótimas aplicações que irão sanar todos os requisitos de negócio necessários. Mas, não podemos impedir que os processos mudem. Não dá para

pedir a uma empresa de telecomunicação que desista de fazer uma promoção no dia dos pais, devido ao número de regras de negócio que serão modificados.

O desenvolvimento baseado em componentes nos deixa mais bem posicionados em um mercado onde a necessidade de aplicações robustas, porém adaptáveis, se torna cada vez mais comum. Necessidade esta, que é um dos motivadores da service oriented architecture (SOA). Mas isto já é assunto para outro artigo. ●

Figura 8. Sequência entre system layer e subsystems.

Figura 9. Sequência entre system layer e subsystems com novas operações

Dê seu feedback sobre esta edição!

A Engenharia de Software Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/esmag/feedback

Links

RUP - Rational Unified Process 7.0
http://www-306.ibm.com/software/awdtools/rup/

Bibliografia

UML Components – A Simple Process for Specifying Component-Based Software
John Cheesman / John Daniels, Paperback

Objects, Components, and Frameworks with UML: The Catalysis(SM) Approach
Desmond Francis D'Souza / Alan Cameron Wills, Paperback

A Importância dos Testes Automatizados

Controle ágil, rápido e confiável de qualidade

Paulo Cheque Bernardo
paulocheque@agilcoop.org.br
É formado em Ciência da Computação pelo Instituto de Matemática e Estatística da Universidade de São Paulo, está atualmente cursando mestrado na área de métodos ágeis e testes automatizados com ferramentas de software livre com apoio do projeto QualiPSo (http://www.qualipso.org) que é uma aliança entre Brasil, China e Europa para estudos em software livre.

Fabio Kon
kon@ime.usp.br
É PhD em Ciência da Computação pela Universidade de Illinois em Urbana-Champaign e atua há 20 anos com desenvolvimento de sistemas software. É professor Livre-Docente do Departamento de Ciência da Computação do IME-USP e, atualmente, está engajado na criação do Centro de Competência em Software Livre da USP (http://ccsl.ime.usp.br). Fabio escreveu seu primeiro teste automatizado em 1996 e não se arrepende disso.

Controlar a qualidade de sistemas de software é um grande desafio devido à alta complexidade dos produtos e às inúmeras dificuldades relacionadas ao processo de desenvolvimento, que envolve questões humanas, técnicas, burocráticas, de negócio e políticas. Idealmente, os sistemas de software devem não só fazer corretamente o que o cliente precisa, mas também fazê-lo de forma segura, eficiente e escalável e serem flexíveis, de fácil manutenção e evolução.

Salvo honrosas exceções, na indústria de software brasileira, estas características são muitas vezes asseguradas através de testes manuais do sistema após o término de módulos específicos ou até mesmo do sistema inteiro. Como veremos mais adiante neste artigo, esta abordagem manual e ad hoc leva à ocorrência de muitos problemas e deve ser evitada. Este artigo se inspira na filosofia dos Métodos Ágeis de Desenvolvimento de Software e em práticas recomendadas pela Programação eXtrema (XP), em especial nos Testes Automatizados, que é uma técnica voltada principalmente para a melhoria da qualidade dos sistemas de software. Este artigo apresenta também alguns exemplos de testes automatizados baseados em excelentes ferramentas disponíveis como software livre.

Cenário comum de desenvolvimento

O modo convencional de desenvolvimento de uma funcionalidade é estudar o problema, pensar em uma solução e, em seguida, implementá-la. Após esses três passos, o desenvolvedor faz testes manuais para verificar se está tudo funcionando como o esperado. É normal que defeitos sejam detectados ao longo do processo de desenvolvimento, então os desenvolvedores precisam encontrar o defeito, corrigi-lo e refazer o conjunto de testes manuais.

Além disso, para verificar se algum erro deixou de ser identificado durante a fase de desenvolvimento, antes de colocar o sistema em produção é muito comum submeter o software a um processo de

avaliação de qualidade. Esse controle de qualidade geralmente é realizado com o auxílio de testes manuais executados por desenvolvedores, usuários ou mesmo por equipes especializadas em testes.

Este cenário é comum principalmente em empresas que utilizam metodologias rígidas que possuem fases bem definidas, geralmente derivadas do modelo de cascata. Este tipo de metodologia pode levar à aparição de diversos problemas recorrentes na indústria de software, tais como atrasos nas entregas, criação de produtos com grande quantidade de erros e dificuldade de manutenção e evolução. Isto se deve em parte às dificuldades da realização de testes manuais.

A execução manual de um caso de teste é rápida e efetiva, mas a execução e repetição de um vasto conjunto de testes manualmente é uma tarefa muito dispendiosa e cansativa. É normal e compreensivo que os testadores não verifiquem novamente todos os casos a cada mudança significativa do código; é deste cenário que surgem os erros de software, trazendo prejuízo para as equipes de desenvolvimento que perdem muito tempo para identificar e corrigir os defeitos e também prejuízo para o cliente que, entre outros problemas, sofre com constantes atrasos nos prazos combinados e com a entrega de software de qualidade duvidosa.

Mas o aspecto mais crítico deste cenário é o efeito "bola de neve". Como é necessário muito esforço para executar todo o conjunto de testes manuais, dificilmente a cada correção de um defeito, a bateria de testes será executada novamente como seria desejável. Muitas vezes, isso leva a erros de regressão (erros em módulos do sistema que estavam funcionando corretamente e deixam de funcionar). A tendência é este ciclo se repetir até que a manutenção do sistema se torne uma tarefa tão custosa que passa a valer a pena reconstruí-lo completamente.

Uma nova abordagem

Muitos métodos ágeis como Lean, Scrum e XP (vide referências) recomendam que todas as pessoas de um projeto (programadores, gerentes, equipes de homologação e até mesmo os clientes) trabalhem controlando a qualidade do produto todos os dias e a todo momento, pois acreditam que prevenir defeitos é mais fácil e barato que identificá-los e corrigi-los. A Programação eXtrema, em particular, recomenda explicitamente testes automatizados para ajudar a garantir a qualidade dos sistemas de software. Vale ressaltar ainda que os métodos ágeis não se opõem a quaisquer revisões adicionais que sejam feitas para aumentar a qualidade.

Testes automatizados são programas ou scripts simples que exercitam funcionalidades do sistema sendo testado e fazem verificações automáticas nos efeitos colaterais obtidos. A grande vantagem desta abordagem é que todos os casos de teste podem ser facilmente e rapidamente repetidos a qualquer momento e com pouco esforço.

A reprodução dos testes permite simular identicamente e inúmeras vezes situações específicas, garantindo que passos importantes não serão ignorados por falha humana e facilitando a identificação de um possível comportamento não desejado.

Além disso, como os casos para verificação são descritos através de um código interpretado por um computador, é possível criar situações de testes bem mais elaboradas e complexas do que as realizadas manualmente, possibilitando qualquer combinação de comandos e operações. A magnitude dos testes pode também facilmente ser alterada. Por exemplo, é trivial simular centenas de usuários acessando um sistema ou inserir milhares de registros em uma base de dados, o que não é factível com testes manuais.

Todas estas características ajudam a minimizar os problemas encontrados nos testes manuais, diminuindo a quantidade de erros e aumentando a qualidade do software. Como é relativamente fácil executar todos os testes a qualquer momento, mudanças no sistema podem ser feitas com segurança, o que aumenta a vida útil do produto.

Na grande maioria das vezes, os testes automatizados são escritos programaticamente, por isso é necessário conhecimento de programação. Nas seções seguintes veremos alguns exemplos de código de testes automatizados que utilizam algumas ferramentas e arcabouços livres bem populares. No entanto, existem alguns tipos de testes que possuem ferramentas gráficas que escondem os detalhes de implementação permitindo que outros profissionais também consigam escrever seus próprios testes, por exemplo JMeter para testes de estresse e desempenho e Selenium-IDE para criação de testes de interface gráfica para aplicações Web (ver seção Links).

Exemplo de Teste Automatizado de Unidade

Se o leitor quer começar a colocar em prática testes automatizados, o primeiro tipo de teste que se deve fazer são os de unidade. Uma unidade pode ser entendida como o menor trecho de código de um sistema que pode ser testado, podendo ser uma função ou módulo em programas procedimentais ou métodos ou classes em programas orientados a objetos. O teste de uma unidade é o tipo mais importante de teste para a grande maioria das situações, já que é ele que deve testar se um algoritmo faz o que deveria ser feito e garantir que o código encapsulado por uma unidade deve produzir o efeito colateral esperado. Outra vantagem importante é que o teste de unidade é focalizado em um trecho específico do código, desta forma os defeitos encontrados são facilmente localizados, diminuindo o tempo gasto com depuração.

Uma prática comum no passado (até a década de 90) era criar uma função de teste em cada módulo ou classe do sistema que continha algumas simulações de uso da unidade. O problema desta prática está na falta de organização, já que o código adicional era misturado ao próprio sistema afetando a legibilidade do código. Por isso, surgiram os arcabouços (frameworks) que auxiliam e padronizam a escrita de testes automatizados, facilitando o isolamento do código de teste do código da aplicação. Estes arcabouços de testes de unidade são conhecidos como parte da família de arcabouços xUnit. Por exemplo, existe o arcabouço pioneiro SUnit para SmallTalk criado por Kent Beck, que foi seguido por implementações para outras linguagens, tais como JUnit e TestNG para Java, JSUnit para Javascript, CppTest para C++, csUnit para .NET, entre outras (ver seção Links).

Para ilustrar, vamos ver um exemplo de teste automatizado para aplicações Java com o auxílio da versão 4 do popular JUnit. Os testes são organizados em casos de teste definidos através de anotações Java @Test antes da definição de cada método de teste. O arcabouço também possui uma gama de métodos auxiliares para comparar os efeitos colaterais esperados com os obtidos, tais como assertEquals que compara a

igualdade do conteúdo de objetos, assertSame que compara se duas referências se referem ao mesmo objeto, assertNull que verifica se uma dada referência é nula, entre outros (ver Listagem 1).

Uma característica comum a muitos arcabouços de testes que facilita ainda mais o desenvolvimento dos testes automatizados é o conjunto de ferramentas auxiliares que as integram com ferramentas e ambientes de programação (IDE), facilitando a execução da bateria de testes e a leitura dos resultados obtidos. Na Figura 1 vemos o relatório gerado com os resultados dos testes executados dentro da IDE Eclipse (ver seção Links). O relatório contém uma barra que fica verde quando todos os testes passam com sucesso, ou vermelha quando pelo menos um caso de teste falha. Para facilitar a localização de erros, é impressa a pilha de execução apenas dos testes que falharam.

O arcabouço é bem simples, assim como os códigos dos testes devem ser. Com esta noção básica da ferramenta já é possível ao leitor criar seus próprios testes automatizados para seus projetos. No entanto, é recomendado um estudo um pouco mais profundo para que seus testes tenham mais qualidade. Duas características fundamentais para garantir a qualidade dos testes são: simplicidade e legibilidade.

O código do teste, assim como o código do sistema, pode conter erros; por isso, é imprescindível que o código do teste seja o mais simples possível. Evite testes longos, código repleto de condições (ifs), testes responsáveis por muitas verificações e também nomes obscuros.

É também importante que a legibilidade dos testes seja boa para facilitar a sua manutenção e para se obter pistas explícitas de qual funcionalidade está quebrada quando um teste falhar. Além disso, testes claros e legíveis podem ser utilizados como documentação do sistema ou como base para gerar a documentação através dos relatórios dos resultados. A Listagem 2 contém um exemplo de um teste para um algoritmo que recebe um inteiro representando uma quantidade de segundos (enviado ao construtor da classe Horario) e devolve uma String representando um display digital (método formatoDeRelogioDigital).

Listagem 1. Exemplo de teste automatizado com o JUnit 4

```
public class ExemploJUnitTest {
    @Test // Este método é um caso de teste
    public void testaAdicaoDeDiasEmUmaData() throws Exception {
        SimpleDateFormat formatador = new SimpleDateFormat("dd/MM/yyyy");
        Date dataDeReferencia = formatador.parse("05/06/2008");
        Date dataDaqui5Dias = formatador.parse("10/06/2008");
        Date dataObtida = DateUtil.adicionaDiasEmUmaData(dataDeReferencia, 5);
        assertEquals(dataDaqui5Dias, dataObtida);
    }
}
```

Listagem 2. Bateria de testes para a classe Horario

```
@Test
public void displayDeSegundosDeveExibirDe00Ate59() {
assertEquals("00:00:00", new Horario(0).formatoDeRelogioDigital());
assertEquals("00:00:01", new Horario(1).formatoDeRelogioDigital());
assertEquals("00:00:05", new Horario(5).formatoDeRelogioDigital());
assertEquals("00:00:59", new Horario(59).formatoDeRelogioDigital());
assertTrue(new Horario(60).formatoDeRelogioDigital().endsWith(":00"));
assertTrue(new Horario(61).formatoDeRelogioDigital().endsWith(":01"));
}

@Test
public void displayDeMinutosDeveExibirDe00Ate59() {
assertEquals("00:01:00", new Horario(60).formatoDeRelogioDigital());
assertEquals("00:01:01", new Horario(61).formatoDeRelogioDigital());
assertEquals("00:02:00", new Horario(120).formatoDeRelogioDigital());
assertEquals("00:02:59", new Horario(179).formatoDeRelogioDigital());
assertEquals("00:03:00", new Horario(180).formatoDeRelogioDigital());
assertEquals("00:59:59", new Horario(59 * 60 + 59).formatoDeRelogioDigital());
assertTrue(new Horario(60*60).formatoDeRelogioDigital().endsWith("00:00"));
assertTrue(new Horario(60*60 + 1).formatoDeRelogioDigital().endsWith("00:01"));
assertTrue(new Horario(60*60 + 120).formatoDeRelogioDigital().endsWith("02:00"));
}

@Test
public void displayDeHorasDeveExibirDe00Ate23() {
assertEquals("00:59:59", new Horario(3599).formatoDeRelogioDigital());
assertEquals("01:00:00", new Horario(3600).formatoDeRelogioDigital());
assertEquals("01:00:01", new Horario(3601).formatoDeRelogioDigital());
assertEquals("01:10:01", new Horario(4201).formatoDeRelogioDigital());
assertEquals("02:00:01", new Horario(7201).formatoDeRelogioDigital());
assertEquals("02:26:01", new Horario(7201 + 26 * 60).formatoDeRelogioDigital());
assertEquals("23:59:59", new Horario(24 * 3600 - 1).formatoDeRelogioDigital());
assertEquals("00:00:00", new Horario(24 * 3600).formatoDeRelogioDigital());
}

@Test
public void displayDeveReiniciarSeAQuantidadeDeSegundosForMaiorQueDeUmDiaCompleto() {
assertEquals("00:00:01", new Horario(24 * 3600 + 1).formatoDeRelogioDigital());
assertEquals("00:01:00", new Horario(24 * 3600 + 60).formatoDeRelogioDigital());
}

@Test
public void displayQuantidadeDeSegundosNegativaDeveExibirDisplayZerado() {
assertEquals("00:00:00", new Horario(-1).formatoDeRelogioDigital());
assertEquals("00:00:00", new Horario(-100).formatoDeRelogioDigital());
}
```

Exemplo de Teste de Inteface Web

Outro tipo de teste importante é o de aceitação, que visa verificar se o que foi implementado atende corretamente ao que o cliente esperava, ou seja, validar o sistema do ponto de vista do cliente. Normalmente, estes testes são realizados através da interface de usuário que pode ser, por exemplo, um console textual, uma interface de uma aplicação local ou uma interface Web. A escrita deste tipo de teste exige mais do que chamadas de métodos e procedimentos. Para se testar uma funcionalidade é necessária a simulação das ações de um usuário interagindo com o programa, isto é, um clique do mouse, uma tecla pressionada, uma opção selecionada, entre outras ações. Por isso é fundamental a utilização de arcabouços que consigam abstrair as ações de usuário encapsulando o funcionamento interno das interfaces para facilitar a escrita e manutenção dos testes de aceitação.

Este é um tipo de teste bem abrangente, pois envolve todas as camadas do sistema, dependendo da modelagem correta da base de dados, do funcionamento correto dos módulos internos do sistema, da integração entre eles e da interação do usuário com a interface. Portanto, escrever bons testes de aceitação que verifiquem adequadamente todos estes componentes é uma tarefa não-trivial que exige um bom conhecimento e experiência.

A Listagem 3 ilustra um exemplo de teste automatizado de uma interface Web com o auxílio das ferramentas Selenium, Selenium Remote Control (Selenium RC) e JUnit. O Selenium, diferentemente de outras

ferramentas, permite executar os testes diretamente nos navegadores mais populares (Firefox, Internet Explorer, Opera, Safari), o que torna possível testar facilmente a compatibilidade de um projeto Web às diferentes plataformas. Já o Selenium RC permite que o código dos testes seja escrito em várias linguagens de alto nível, por exemplo, Java, Ruby, Python ou C#. O JUnit recebe as informações obtidas do navegador e compara com os valores esperados.

A classe DefaultSelenium possui métodos prontos que abstraem o funcionamento interno de eventos dos navegadores, tais como click ou select que simulam um clique do mouse em um objeto e a seleção de uma opção em caixas de seleção, respectivamente. Todos os métodos que fazem uma interação com um objeto em particular exigem um argumento que permite a identificação do objeto. Este argumento pode ser expressões DOM (Document Object Model - linguagem para permitir que programas acessem e modifiquem dinamicamente objetos de um documento HTML) ou XPath (XML Path Language - linguagem para identificar trechos específicos de um arquivo XML) (ver seção Links), ou simplesmente um identificador definido no código HTML. Desta forma, o código do teste passa a ser um exemplo de uso do sistema.

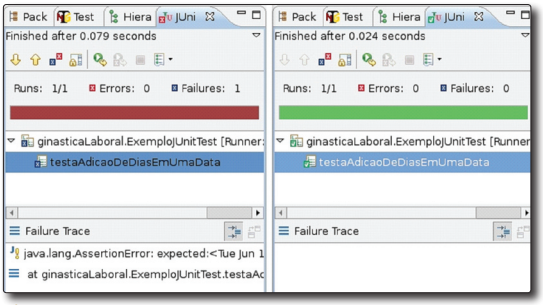


Figura 1. Relatório dos testes do plug-in do JUnit para a IDE Eclipse

Listagem 3. Exemplo de teste com Selenium-RC e JUnit

```
@Test
public void testaPaginaDoProjetoQualipso() throws Exception {
String url = "http://www.qualipso.org";
String servidor = "localhost";
String porta = 4444;
String navegador = "*firefox";
Selenium selenium = new DefaultSelenium(servidor, porta, navegador, url);

selenium.start(); // Abre o navegador

// Entra no site
selenium.open("/");
// Verifica se um texto apareceu
assertTrue(selenium.isTextPresent("It is all about trust"));
// Clica em um link
selenium.click("link=Work Areas");
// Espera carregar a nova página
selenium.waitForPageToLoad("30000");
// Verifica se apareceu um outro texto
assertTrue(selenium.isTextPresent("TRUSTWORTHY RESULTS"));

selenium.stop(); // Fecha o navegador
}
```

Considerações Finais

Desenvolvimento de software é uma tarefa complexa que exige conhecimento técnico, organização, atenção, criatividade e também muita comunicação. É previsível que, em alguns momentos do desenvolvimento, em algumas das milhares de linhas de código, alguns destes requisitos falhe, mas é imprevisível o momento no qual eles irão falhar. Por isso, é imprescindível que exista uma maneira fácil e ágil de executar todos os testes em qualquer instante, e isso só é viável com o auxílio de testes automatizados.

A automação dos testes dá segurança à equipe para fazer alterações no código, seja por manutenção, refatoração ou até mesmo para adição de novas funcionalidades. Além disso, representar casos de teste através de programas possibilita a criação de testes mais elaborados e complexos, que poderão ser repetidos inúmeras vezes.

Conseqüentemente, a quantidade de tempo gasto com a verificação do sistema aumenta, enquanto diminui o tempo gasto com a identificação e correção de erros, já que eles tendem a ser encontrados mais cedo. À medida que o tempo passa, o sistema vai crescendo e os programadores vão esquecendo certos detalhes de implementação, por isso quanto mais tarde um erro for encontrado, mais trabalhosa será a depuração para a sua localização.

Com essas noções básicas de testes automatizados, o leitor pode se aprofundar no assunto pesquisando a automação de outros tipos de testes, estudando padrões de escrita de bons testes automatizados e o Desenvolvimento Dirigido por Testes, que é uma técnica na qual os testes guiam a implementação do sistema. Outro tópico importante é a execução automática dos testes que pode ser feita com o auxílio de ferramentas que ficam constantemente verificando se um código foi alterado ou com ferramentas que executam a bateria de testes periodicamente. ●

Dê seu feedback sobre esta edição!

A Engenharia de Software Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/esmag/feedback

Links

JMeter: http://jakarta.apache.org/jmeter
Selenium IDE: http://selenium-ide.openqa.org
Selenium RC: http://selenium-rc.openqa.org
SUnit: http://sunit.sourceforge.net
JUnit: http://www.junit.org
TestNG: http://testng.org
JSUnit: http://www.jsunit.net
CppTest: http://cpptest.sourceforge.net
csUnit: http://www.csunit.org
Eclipse: http://www.eclipse.org
XPath: http://www.w3.org/TR/xpath

Referências

AgilCoop: Possui artigos, palestras, podcasts e vídeos relacionados a Métodos Ágeis

www.agilcoop.org.br

QualiPSo: Possui material de pesquisa de software livre, incluindo artigos relacionados a testes automatizados.

www.qualipso.org

Beck, K.; Andres C. Extreme Programming Explained: Embrace Change. Segunda edição. Ed. Addison-Wesley, 2004.

Poppendieck M.; Poppendieck T. Lean Software Development: An Agile Toolkit. Ed. Addison-Wesley, 2003

Schwaber, K.; Beedle, M. Agile Software Development with SCRUM. Ed. Prentice Hall, 2001.

Beck, K. Test-Driven Development: By Example. Ed. Addison-Wesley Professional, 2002

Delamaro, M.; Maldonado, J.; Jino, M. Introdução ao Teste de Software. Ed. Campus, 2007

Gestão de Testes

Ferramentas Open Source e melhores práticas na gestão de testes

Cristiano Caetano
c_caetano@hotmail.com
É certificado CBTS pela ALATS. Consultor de teste de software sênior com mais de 10 anos de experiência, já trabalhou na área de qualidade e teste de software para grandes empresas como Zero G, DELL e HP Invent. É colunista na área de Teste e Qualidade de software do site linhadecodigo.com.br e autor dos livros "CVS: Controle de Versões e Desenvolvimento Colaborativo de Software" e "Automação e Gerenciamento de Testes: Aumentando a Produtividade com as Principais Soluções Open Source e Gratuitas". Criador e mantenedor do portal TestExpert: A sua comunidade gratuita de teste e qualidade de software (www.testexpert.com.br).

Estima-se que o custo decorrente da correção de um bug cresce bastante à medida que ele é descoberto em fases mais avançadas no processo de desenvolvimento de software. No entanto, ainda existe uma forte tendência nas empresas em negligenciar essa realidade e não dedicar o tempo mínimo necessário para a realização das atividades de teste de software.

As atividades de teste são muitas vezes realizadas de maneira pouco estruturada ao final do projeto, quando não existe mais solução para os problemas. Segundo Pressman, a atividade de teste seria um dos elementos críticos da garantia da qualidade de software e pode assumir até 40% do esforço gasto em seu desenvolvimento.

A qualidade é um atributo do software que deve ser introduzida ao longo do processo de desenvolvimento do software, haja vista que ela não pode ser imposta depois que o produto tenha sido finalizado.

Glenford Myers, no seu livro "The Art of Software Testing", destaca que teste de software é o processo de executar um sistema com o objetivo de revelar falhas. No entanto, as atividades de teste de software não se resumem apenas a isso. Teste de software é uma atividade estruturada e sistemática baseada em técnicas, ferramentas e processos formais, como veremos ao longo deste artigo.

Validação e Verificação

A validação e verificação são atividades de apoio de um processo de garantia de qualidade de software. A motivação principal dessas atividades é prevenir e detectar os defeitos e minimizar os riscos do projeto.

Os defeitos podem ser introduzidos ao longo do processo de desenvolvimento do software. É necessário que eles sejam identificados o quanto antes dentro do processo de desenvolvimento, de preferência na própria fase onde foram inseridos, mas nem sempre isso acontece.

As atividades de validação e verificação são baseadas em técnicas de análise estática ou dinâmica dos artefatos (documentos, código fonte, código executável, etc) com o intuito de detectar os defeitos ou revelar falhas na própria fase onde eles foram inseridos ou em fases posteriores.

A verificação tem o objetivo de avaliar se o software está sendo desenvolvido conforme os padrões e metodologia estabelecidos no projeto. A verificação normalmente é realizada por meio da análise estática (revisões, inspeções, etc) dos artefatos (documentos, código fonte, etc) produzidos ao longo do processo de desenvolvimento do software.

A validação, por outro lado, tem o objetivo de avaliar a aderência, ou conformidade, do software implementado em relação ao comportamento descrito nos requisitos A validação normalmente é realizada por meio da análise dinâmica (execução de testes contra o código executável).

Níveis de teste

As atividades de testes são normalmente divididas em níveis. O nível de teste define, de certa forma, a fase do processo de desenvolvimento do software na qual os testes serão realizados. Existem quatro níveis de testes, como pode ser observado na Tabela 1.

Tipos de teste

Os tipos de teste normalmente são definidos em função das características ou dimensões da qualidade que serão avaliadas no software. A escolha da característica da qualidade de um software é, às vezes, um processo subjetivo. No entanto, essa escolha normalmente é realizada com base nos riscos associados a um problema causado por uma falha em uma dessas características.

A norma internacional ISO/IEC 9126, publicada em 1991 e que na versão brasileira de agosto de 1996 recebeu o número NBR 13596, define qualidade de software como "A totalidade de características de um produto de software que lhe confere a capacidade de satisfazer necessidades explícitas e implícitas".

Necessidades explícitas são as condições e objetivos propostos por aqueles que produzem o software. São, portanto fatores relativos à qualidade do processo de desenvolvimento do produto e são percebidos somente pelas pessoas que trabalharam no seu desenvolvimento.

As necessidades implícitas são necessidades subjetivas dos usuários (inclusive operadores, destinatários dos resultados do software e os mantenedores do produto). As necessidades implícitas são também chamadas de qualidade em uso e devem permitir a usuários atingir metas com efetividade, produtividade, segurança e satisfação em um contexto de uso especificado.

A ISO/IEC 9126 (NBR 13596) fornece um modelo de propósito geral que define seis amplas categorias de características de qualidade de software que são, por sua vez, subdivididas em sub-características, como pode ser observado na Tabela 2.

Conforme o que foi exposto anteriormente, a escolha do tipo de teste dependerá do grau de importância de cada uma das características de qualidade que serão avaliadas no software. Os tipos de testes mais comuns conforme o Guide to the CSTE Common Body of Knowledge do QAI são descritos resumidamente na Tabela 3. É importante destacar que não foram esgotadas todas as opções disponíveis, você poderá encontrar outros tipos de testes em outros artigos ou livros.

Nível de Teste	Descrição
Testes de unidade	Nesta fase são testadas as menores unidades de software desenvolvidas (por exemplo: métodos de uma classe).
Testes de integração	Nesta fase é testada a integração entre os componentes do sistema (por exemplo: classes, módulos, sub-sistemas, etc).
Testes de sistema	Nesta fase o sistema é testado como um todo com o objetivo de encontrar discordâncias entre o que foi implementado e o comportamento descrito nos requisitos.
Testes de aceitação	Nesta fase o sistema é testado como um todo com o objetivo de encontrar discordâncias entre o que foi implementado e o comportamento descrito nos requisitos, sob o ponto de vista das necessidades do usuário final.

Tabela 1. Níveis de teste.

Característica	Sub-características
Funcionalidade O conjunto de funções satisfaz as necessidades explícitas e implícitas para a finalidade a que se destina o produto?	Adequação Acurácia Interoperabilidade Segurança de acesso Conformidade
Confiabilidade O desempenho se mantém ao longo do tempo e em condições estabelecidas?	Maturidade Tolerância a falhas Recuperabilidade
Usabilidade É fácil utilizar o software?	Inteligibilidade Apreensibilidade Operacionalidade
Eficiência Os recursos e os tempos utilizados são compatíveis com o nível de desempenho requerido para o produto?	Comportamento em relação ao tempo Comportamento em relação aos recursos
Manutenibilidade Há facilidade para correções, atualizações e alterações?	Analisabilidade Modificabilidade Estabilidade Testabilidade
Portabilidade É possível utilizar o produto em diversas plataformas com pequeno esforço de adaptação?	Adaptabilidade Capacidade para ser instalado Capacidade para substituir Conformidade

Tabela 2. Categorias de características de qualidade de software da ISO/IEC 9126 (NBR 13596).

Técnicas de teste

Considerando o tamanho e a complexidade dos sistemas desenvolvidos na atualidade, podemos afirmar que é muito difícil executar testes que garantam 100% de cobertura de todos os requisitos ou linhas de código existentes. Somado a isso, prazos curtíssimos e orçamentos pequenos podem inviabilizar qualquer tentativa de alcançar 100% de cobertura.

Nessa condição, as técnicas de teste têm o objetivo de auxiliar os analistas de teste a identificar os casos de teste mais importantes. Ou seja, os casos de teste que exercitem a maior quantidade de linhas de código e garantam a maior cobertura possível, como pode ser observado na listagem:

- Teste Estrutural: nesta técnica, também conhecida como "Teste de Caixa Branca", são usados critérios para a geração de casos de teste com o objetivo de identificar defeitos nas estruturas internas do software.
- Teste Funcional: nesta técnica, também conhecida como "Teste de Caixa Preta", são usados critérios para a geração de casos de teste com o objetivo de avaliar a aderência, ou conformidade do software implementado em relação ao comportamento descrito nos requisitos.

Como você deve ter notado, as técnicas de teste avaliam o software sob pontos de vista diferentes. Dessa forma, podemos afirmar que as técnicas de teste são complementares e devem ser usadas em conjunto, em virtude de que elas identificam classes distintas de defeitos.

Modelo em "V" de teste de software

O modelo em "V" de teste de software é um dos modelos mais aceitos da atualidade. Este modelo enfatiza as atividades de validação e verificação com o intuito de prevenir/detectar defeitos e minimizar os riscos do projeto.

Para cada fase do processo de desenvolvimento do software, o modelo em "V" introduz uma fase, ou nível de teste correspondente. Neste modelo, o planejamento e a especificação dos testes ocorrem de cima para baixo, ou seja, ao longo das fases de desenvolvimento de software os testes são planejados e especificados. A execução dos testes ocorre no sentido inverso, como pode ser visto na (Figura 1).

Além disso, este modelo reforça o entendimento de que o teste não é uma fase que deve ser executada ao final do projeto, mas uma atividade que deve ser exercida ao longo do processo de desenvolvimento do software.

Estratégia de Testes

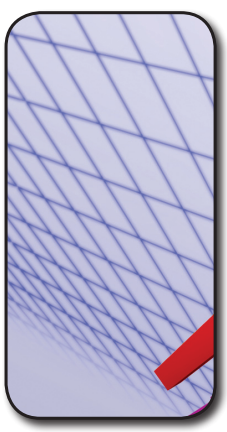
Os conceitos apresentados anteriormente são os pilares fundamentais para a elaboração de uma estratégia de testes. No livro "Teste de Software", Emerson Rios e Trayahu Moreira afirmam que durante a formulação da estratégia de testes devem ser levados em consideração diversos fatores, tais como: o porte e a importância do software, os seus requisitos, os prazos estabelecidos, o risco para o negócio, entre outros.

O diagrama da Figura 2 apresenta a relação entre as técnicas, tipos e níveis de testes que devem ser considerados durante a elaboração de uma estratégia de testes. Na Tabela 4 é apresentado um exemplo de uma estratégia de testes para uma aplicação web.

Perceba que neste exemplo hipotético houve uma atenção redobrada nos tipos de testes de segurança, usabilidade e desempenho em virtude de que essas características de qualidade são mais importantes em se tratando de aplicações web.

Tipo de teste	Descrição
Teste de Estresse	Avalia o desempenho do sistema com um volume de acesso/transações acima da média esperada e em condições extremas de uso.
Teste de Execução	Avalia se o sistema atende os requisitos de performance (proficiência) com um volume de acesso/transações dentro do esperado.
Teste de Contingência	Avalia se o sistema retorna a um status operacional após uma falha.
Teste de Operação	Avalia se o sistema (aplicação, pessoal, procedimentos e manuais) pode ser executado corretamente em ambiente de pré-produção.
Teste de Conformidade	Avalia se o sistema foi desenvolvido em consonância com os padrões e metodologia estabelecidos no projeto.
Teste de Segurança	Avalia se o sistema foi desenvolvido em consonância com os padrões de segurança da organização.
Teste de Regressão	Avalia por meio do re-teste se uma funcionalidade que estava funcionando ainda funciona após uma modificação no sistema.
Teste de Integração	Avalia se a interconexão entre as aplicações funciona corretamente.

Tabela 3. Tipos de teste.



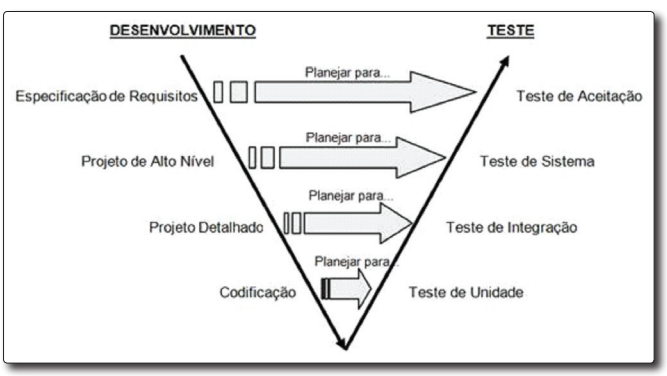


Figura 1. Modelo V descrevendo o paralelismo entre as atividades de desenvolvimento e teste de software (CRAIG e JASKIEL, 2002)

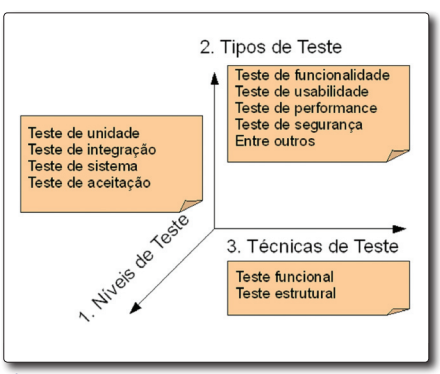


Figura 2. Fatores considerados durante a elaboração de uma estratégia de testes.

Processo estruturado de teste de software

Uma metodologia ou processo estruturado de teste de software tem a finalidade de formalizar as fases, atividades, papéis, artefatos e responsabilidades necessárias para o planejamento e a execução dos testes sistematicamente.

O TMAP (Testing Management Approach), ou abordagem estruturada de gestão de testes, é uma das metodologias mais populares da atualidade. Em virtude da sua simplicidade e foco no resultado, o TMAP é amplamente aceito pelo mercado.

O lema do TMAP é: "Entregar mais com menos. Mais rápido e melhor". O processo de gestão recomendado pelo TMAP é baseado em diversas práticas fundamentais e em um ciclo de vida estruturado de testes.

O ciclo de vida do TMAP (ver Figura 3) é bastante genérico e pode ser utilizado em diferentes tipos de projeto de teste de software. O ciclo de vida é dividido em sete fases distintas, como pode ser observado a seguir:

- Planejamento: Nesta fase é realizado o planejamento e a definição geral da estratégia e planos de testes.
- Controle: Nesta fase são realizados o controle e a monitoração das atividades planejadas.
- Configuração e manutenção da infra-estrutura: Nesta fase é preparada e mantida a infra-estrutura (software e hardware) necessária para a plena realização dos testes.
- Preparação: Nesta fase é realizado o refinamento da estratégia de testes e plano de testes criados na fase de Planejamento.
- Especificação: Nesta fase é realizada a especificação dos casos de testes e demais documentos.
- Execução: Nesta fase é realizada a execução dos testes, reporte do progresso e indicadores de qualidade.
- Conclusão: Nesta fase o processo de teste é avaliado a fim de promover as melhorias para os próximos projetos.

Documentação das atividades de testes

Uma metodologia ou processo estruturado de teste exige um conjunto mínimo de documentos padronizados para documentar os planos e estratégias, assim como, o relato do progresso das atividades de testes.

Níveis de Teste	Teste Funcional	Teste Estrutural
	Tipos de Teste	
Teste de unidade		Teste Unitário Teste de Conformidade
Teste de integração	Teste de Integração	Teste de Estresse Teste de Execução Teste de Segurança
Teste de sistema	Teste de Requisitos Teste de Regressão Teste de Usabilidade	Teste de Estresse Teste de Segurança Teste de Execução
Teste de aceitação	Teste de Aceitação Teste de Usabilidade	Teste de Contingência Teste de Operação

Tabela 4. Exemplo da formulação de uma estratégia de testes.

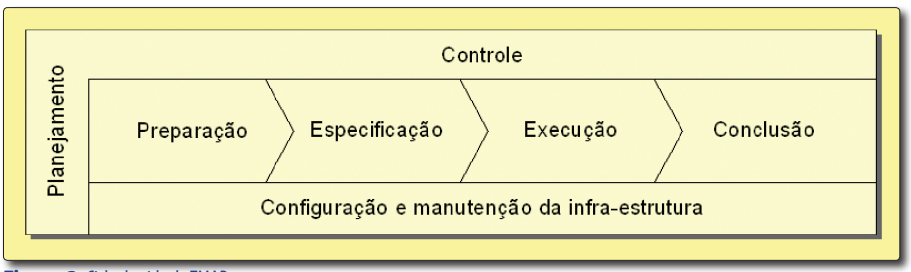


Figura 3. Ciclo de vida do TMAP.

O padrão IEEE Std 829-1998 (IEEE Standard for Software Test Documentation) define a documentação e relatórios necessários para a execução de um projeto de teste de software. Os oito documentos essenciais deste padrão são:

- Plano de Teste: Define o planejamento para execução do teste, incluindo a abrangência, abordagem, recursos e cronograma das atividades de teste. Identifica os itens e funcionalidades a serem testados, as características dos itens a serem testados, as tarefas a serem realizadas e os riscos associados com a atividade de teste.
- Especificação do Projeto de Teste: Refina a abordagem apresentada no Plano de Teste e identifica as funcionalidades e características a serem testadas pelo projeto e pelos seus testes associados. Também identifica os casos e os procedimentos de teste, se existirem, e apresenta os critérios de aprovação para esses elementos.
- Especificação dos Casos de Teste: Define os casos de teste, incluindo as pré-condições, passos, resultados esperados e condições gerais para a execução do teste.
- Especificação dos Procedimentos de Teste: Define a seqüência de ações necessárias para executar um conjunto de casos de teste.
- Relatório de Encaminhamento de Item de Teste: Identifica os itens encaminhados para teste, no caso da existência de times distintos responsáveis pelas tarefas de desenvolvimento e de teste.
- Log de Testes: Documenta os registros cronológicos dos fatos relevantes durante a execução dos testes.
- Relatório de Incidente de Testes: Documenta qualquer evento que ocorra durante a atividade de teste e que necessite de alguma análise posterior.
- Relatório Sumário de Testes: Documenta de forma resumida os resultados das atividades de teste associadas com uma ou mais especificações de projeto de teste e fornece avaliações baseadas nesses resultados.

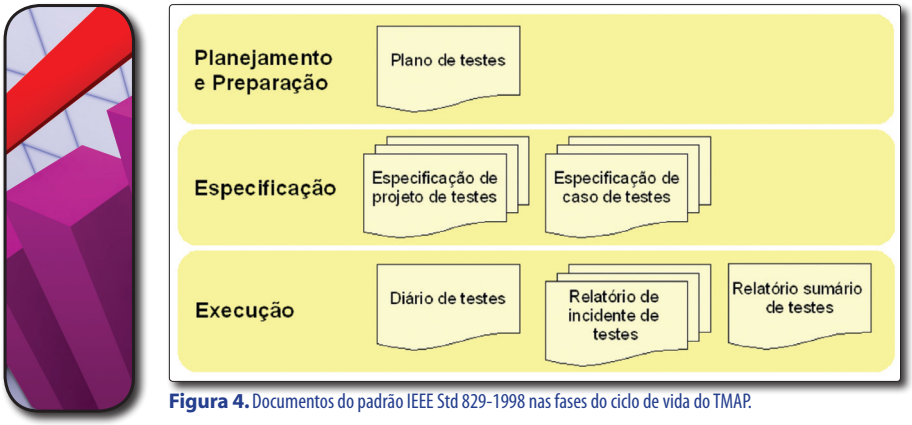


Figura 4. Documentos do padrão IEEE Std 829-1998 nas fases do ciclo de vida do TMAP.

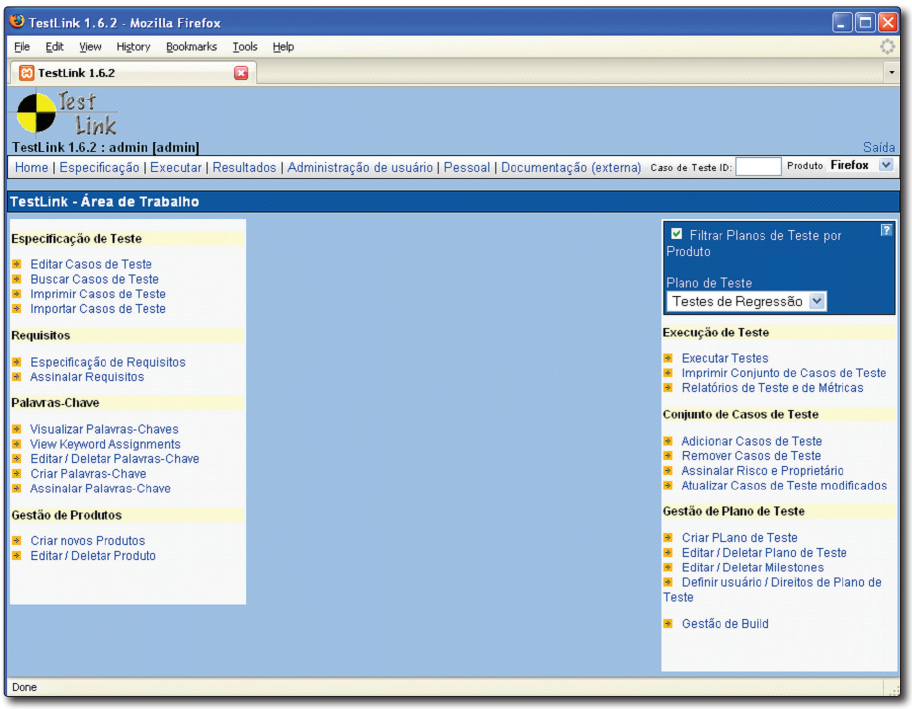


Figura 5. Página principal do TestLink.

Na Figura 4 é apresentada uma sugestão de utilização dos documentos do padrão IEEE Std 829-1998 nas fases do ciclo de vida do TMAP (Testing Management Approach).

TestLink

TestLink é uma ferramenta open source automatizada escrita em PHP cujo principal objetivo é dar suporte às atividades de gestão de testes, como pode ser observado no exemplo apresentado na Figura 5.

Você poderá criar planos de testes e gerar relatórios com diversas métricas para o acompanhamento da execução dos testes. TestLink oferece um recurso para que você possa registrar e organizar os requisitos do projeto, assim como, associar os casos de teste aos requisitos. Dessa forma, você poderá garantir o rastreamento entre os requisitos e os casos de teste utilizando uma matriz de rastreabilidade.

O endereço do site da ferramenta TestLink está disponível na seção Links. Caso você tenha interesse em conhecê-la com maior profundidade, os passos para a instalação são:

1. As pré-condições para a instalação são (PHP 4.0.6 ou superior, MySQL 3.23.2 ou superior, Apache);
2. Faça o download do TestLink no site disponível na seção Links;
   - Descompacte o arquivo zip na pasta www ou htdocs do servidor WEB (Apache).
3. Abra o seu navegador e acesse o seguinte endereço: (http://localhost/testlink-1.6.2/install/index.php);
4. Na janela de instalação, selecione a opção New installation;
5. Na janela TestLink Setup, preencha o campo login com "root" e o campo password com a senha de acesso ao seu banco de dados MYSQL;
6. Deixe os valores default nos demais campos;
7. Pressione o botão "Setup TestLink";
8. Abra o seu navegador e acesse o seguinte endereço: (http://localhost/testlink-1.6.2/login.php);
9. Faça o login com o usuário padrão (admin/admin). Lembre-se de mudar a senha deste usuário.

TestLink é instalado em inglês por padrão. Caso você prefira utilizar TestLink com os textos traduzidos para a língua portuguesa, então siga os passos:

1. Faça o login normalmente com o seu usuário e senha;
2. Clique no menu "Personal";
3. No campo Locale, selecione a opção "Portuguese (Brazil)";
4. Pressione o Botão "Update".

Entre as diversas funcionalidades oferecidas por TestLink, devemos destacar:

- Controle de acesso e níveis de permissões por papéis (líder, testador, etc);
- Os casos de testes são organizados hierarquicamente em suítes;
- Os casos de testes podem ser classificados por palavras-chave "keywords" para facilitar a pesquisa e organização;
- Criação ilimitada de projetos e casos de teste;
- Os planos de testes podem ser priorizados e atribuídos aos testadores;
- Gerador interno de relatórios e gráficos (possibilidade para exportar os dados nos formatos CSV, Excel e Word);
- Integração com ferramentas de gestão de defeitos (Bugzilla, Mantis, Jira).

A fim de dar ao leitor uma exposição sobre as principais funcionalidades da ferramenta TestLink e como elas auxiliam no planejamento execução dos testes, vamos simular na prática o cadastro de um caso de teste e acompanhar todos os passos até a sua execução.

No nosso cenário, vamos simular o cadastro e execução de um caso de teste para o gerenciador de temas do

navegador Firefox. Este cenário é baseado em um dos casos de testes disponibilizados pelo time de Quality Assurance do Firefox. Para saber mais, visite o site: http://litmus.mozilla.org/.

Primeiro, devemos criar um projeto em TestLink. No nosso exemplo, foi criado um projeto hipotético chamado "Firefox". Tão logo o projeto seja criado, devemos criar os requisitos. Para realizar tal tarefa, acesse o menu "Requisitos → Especificação de Requisitos".

Nesta janela você poderá criar os requisitos. No nosso exemplo, foi criada uma especificação de requisito para o recurso "Add-On". Para este recurso foram criados os requisitos "Extensions" e "Themes", como pode ser visto no exemplo apresentado na Figura 6.

Assim que os requisitos forem criados, você estará apto para criar os casos de teste. Antes, no entanto, você deverá criar um "Componente" e uma "Categoria". A arquitetura de TestLink exige que os testes sejam organizados em componentes e categorias. Para o nosso exemplo, foi criado um "Componente" chamado "Add-Ons" e uma "Categoria" chamada "Smoke Tests".

Satisfeitas as pré-condições de TestLink, o próximo passo é a criação do caso de teste. Para isto, você deverá acessar o menu "Especificação", selecionar a "Categoria" desejada e então clicar no botão "Criar Casos de Teste".

Na janela de cadastro de caso de teste, você deverá informar o sumário, os passos e os resultados esperados, como pode ser observado na Figura 7. Assim que o teste for cadastrado, você poderá criar uma matriz de rastreabilidade bidirecional entre os requisitos através da opção de menu "Requisitos → Assinalar Requisitos".

TestLink também oferece uma funcionalidade para imprimir um documento com todos os dados fornecidos durante a criação do "Componente", "Categoria" e caso de teste. Este documento é muito parecido com os documentos propostos pelo padrão IEEE Std 829-1998 (IEEE Standard for Software Test Documentation), como pode ser visto no exemplo da Figura 8.

TestLink permite que os casos de teste sejam agrupados em Planos de Teste (ou Teste Suítes). Para criar um Plano de

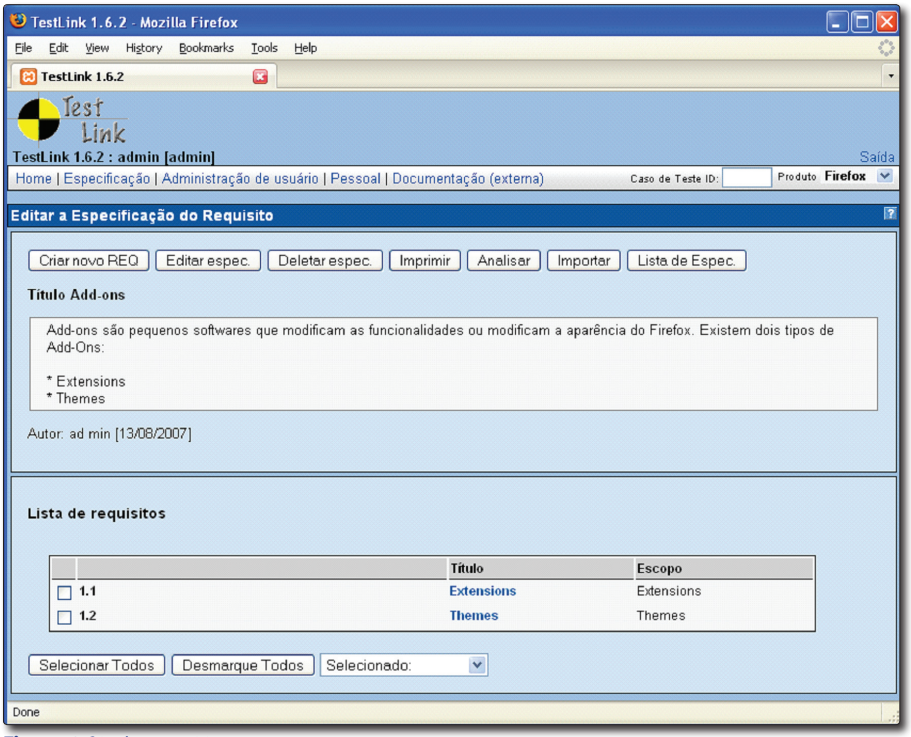


Figura 6. Criando um requisito.

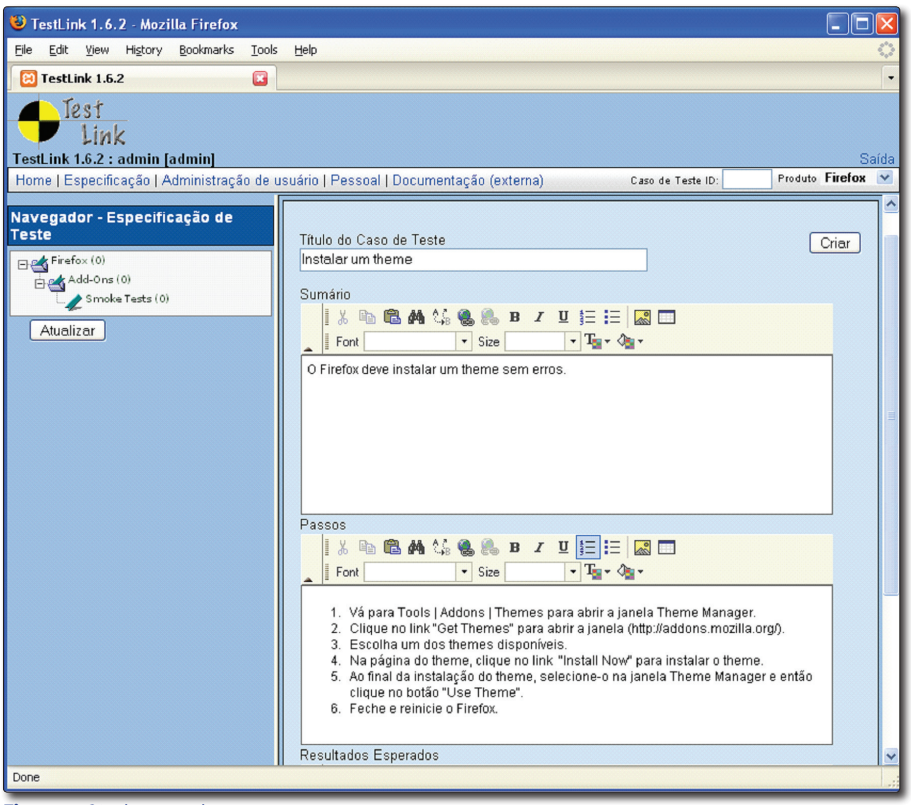


Figura 7. Criando um caso de teste.

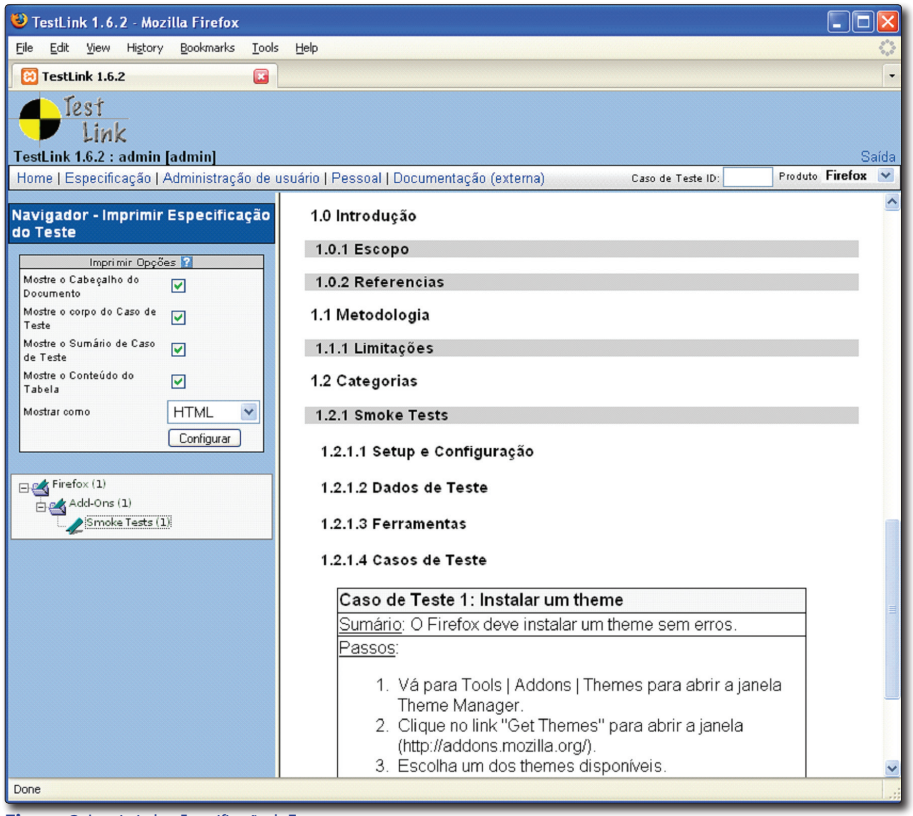


Figura 8. Imprimindo a Especificação do Teste.

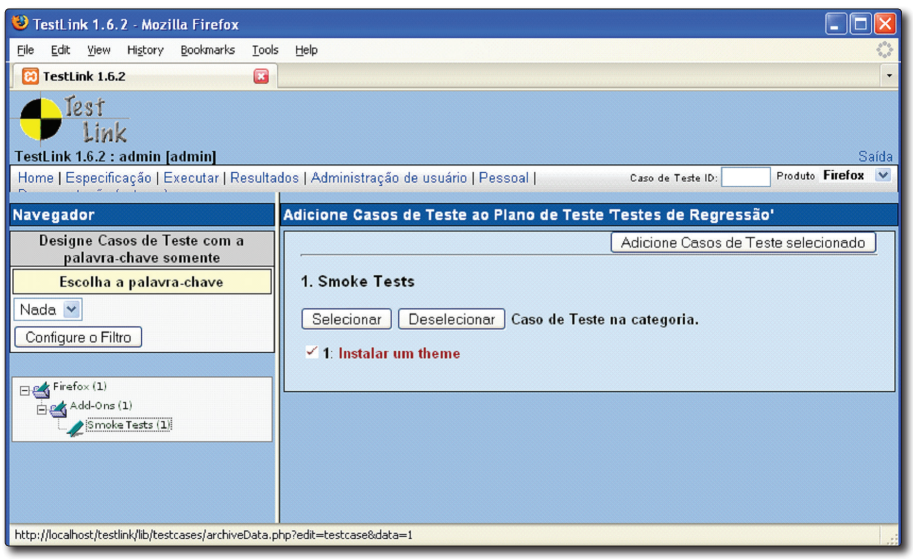
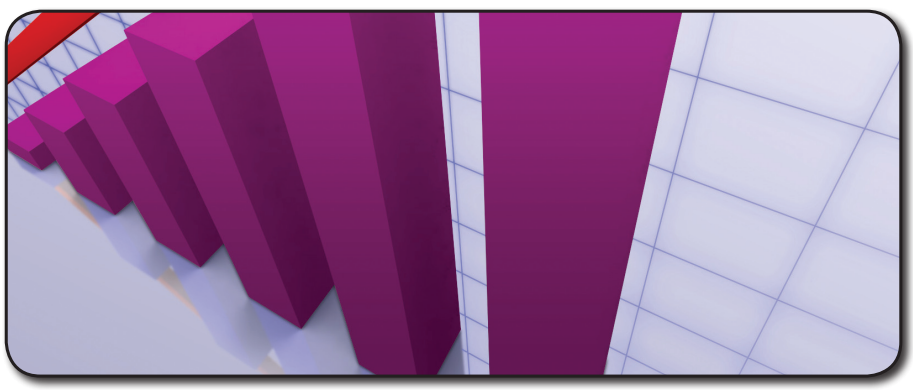


Figura 9. Associando um caso de teste a um Plano de Teste.



Teste, você deverá acessar o menu "Gestão de Plano de Teste → Criar Plano de Teste". Para o nosso exemplo, foi criado o Plano de Teste chamado "Testes de Regressão".

Tão logo o Plano de Teste seja criado, você poderá associá-lo aos casos de testes na janela "Adicione Casos de Teste ao Plano de Teste" por meio da opção de menu "Conjunto de Casos de Teste → Adicionar Casos de Teste", como pode ser observado na Figura 9.

Antes de começar a execução dos testes, você deverá associar os Planos de Testes aos testadores responsáveis por cada área ou funcionalidade do sistema. Para realizar tal tarefa, você deverá abrir a janela "Assinalar usuários para o Plano de Teste" acessando o menu "Gestão de Plano de Teste → Definir usuário/Direitos de Plano de Teste" e associar o Plano de Teste aos testadores, como pode ser visto no exemplo apresentado na Figura 10.

Para iniciar a execução dos testes, o testador deverá selecionar o Plano de Teste associado a ele e então ir para a janela "Execução dos casos de teste" por meio do menu "Execução de Teste > Executar Testes". Nesta janela, o testador poderá executar todos os testes existentes no Plano de Teste.

Para cada caso de teste, são exibidos os passos a serem executados e os resultados esperados. Ao final da execução do teste, o testador poderá relatar alguma nota ou comentário, assim como, o resultado do teste (Não executado, Passado/Verificado, Com falha ou Bloqueado), como pode ser observado na Figura 11.

É importante destacar que a arquitetura de TestLink exige que a execução dos testes seja associada a um Build. Um Build representa uma versão ou liberação do sistema. Para criar um Build, você deverá abrir a janela "Gestão de Build" acessando o menu "Gestão de Plano de Teste → Gestão de Build".

Durante ou ao final da execução dos testes, você poderá acompanhar o progresso através da geração de relatórios gerenciais. Estes relatórios apresentam a situação da execução dos testes, assim como, métricas importantes para determinar a qualidade do sistema, ou melhor, do Build atual do sistema. Entre os relatórios existentes, podemos destacar:

- Métricas gerais do Plano de Teste: Apresenta as métricas dos testes agrupados por prioridade, componente, testador, entre outros;
- Status Geral do Build: Neste relatório é apresentado o sumário dos testes planejados em relação aos executados, os testes que falharam, os que passaram, etc;
- Relatórios de Testes baseados nos requisitos: Apresenta os requisitos cobertos pelos testes e o sumário da execução agrupado por Especificação de Requisitos.

Para visualizar os relatórios, você deverá abrir a janela "Relatórios de Teste e de Métricas" clicando em "Execução de Teste > Relatórios de Teste e de Métricas", como pode ser visto na Figura 12.

Conclusão

Neste artigo foram apresentados os conceitos e algumas das melhores práticas na gestão de testes. O objetivo principal do artigo era reforçar o entendimento de que o teste não é uma fase que deve ser executada ao final do projeto, mas uma atividade que deve ser exercida ao longo do processo de desenvolvimento do software.

Teste de software é uma atividade cara que exige tempo, planejamento, conhecimento técnico, infra-estrutura e comprometimento. Por maior e mais organizado que seja o esforço para a realização das atividades de testes, é impossível garantir 100% de cobertura de todos os requisitos ou linhas de código existentes no sistema.

Edsger Dijkstra sabiamente resumiu este fato com a seguinte frase: "Testes podem mostrar a presença de erros, mas não a sua ausência". Dessa forma, está em suas mãos a responsabilidade de conhecer e aplicar as melhores práticas na gestão de testes para garantir a maior cobertura com o mínimo de esforço.

Ao longo desse artigo foram apresentadas as principais funcionalidades de TestLink, ferramenta Open Source para gestão de testes. No entanto, caso TestLink não atenda suas necessidades, são apresentados na Tabela 5 algumas alternativas comerciais e open source. Não foram esgotadas as opções disponíveis, mas já é um bom ponto de partida para auxiliar o leitor a encontrar a solução ideal para a sua necessidade. ●

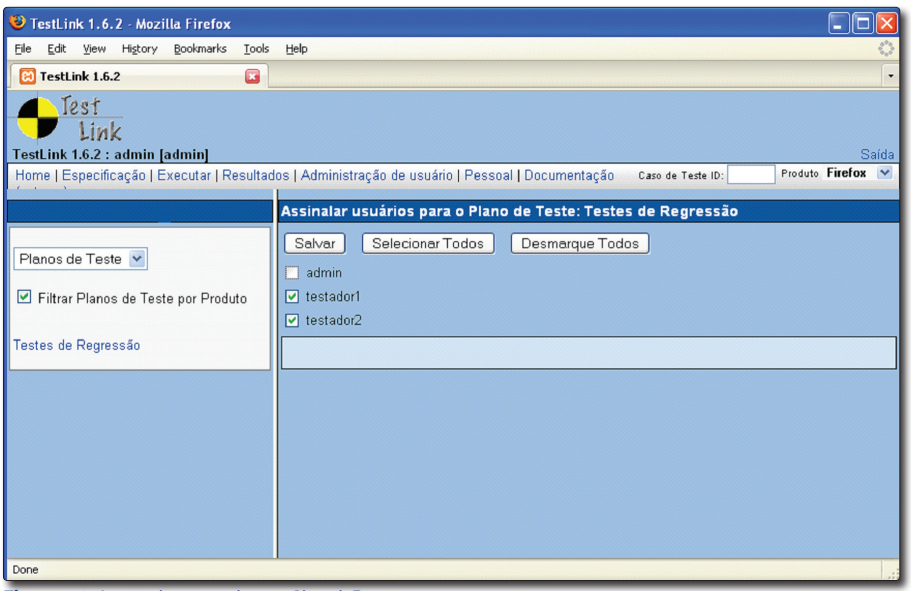


Figura 10. Associando um testador a um Plano de Teste.

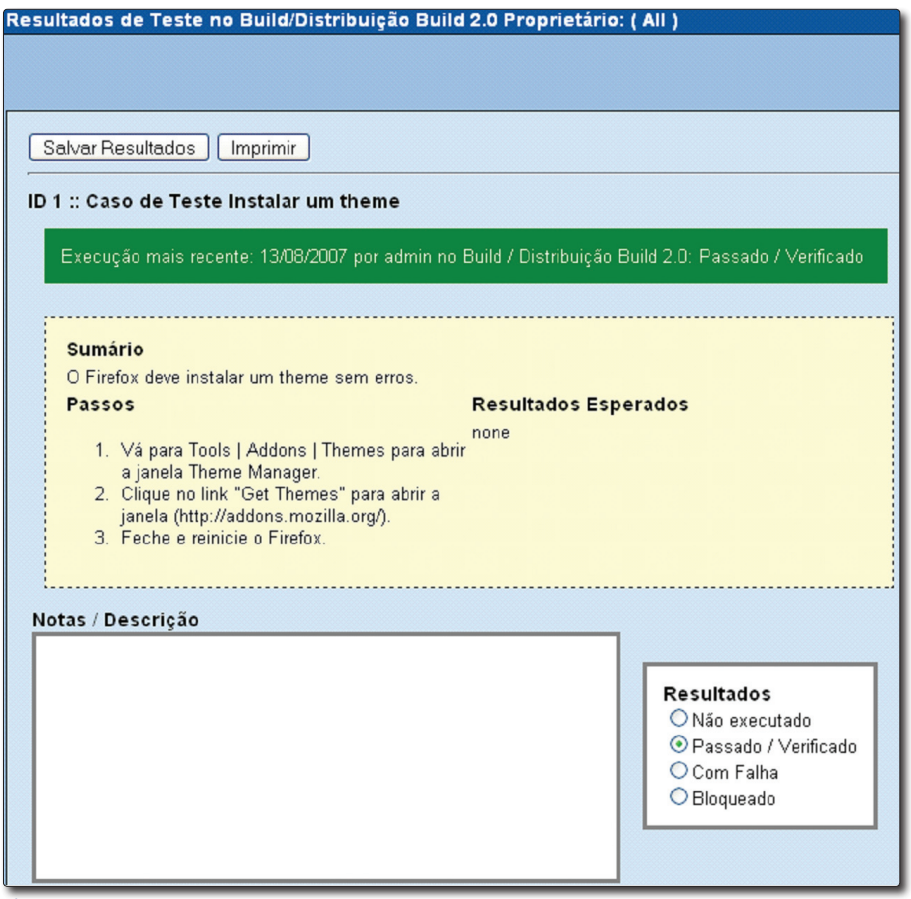


Figura 11. Executando os casos de teste de um Plano de Teste.

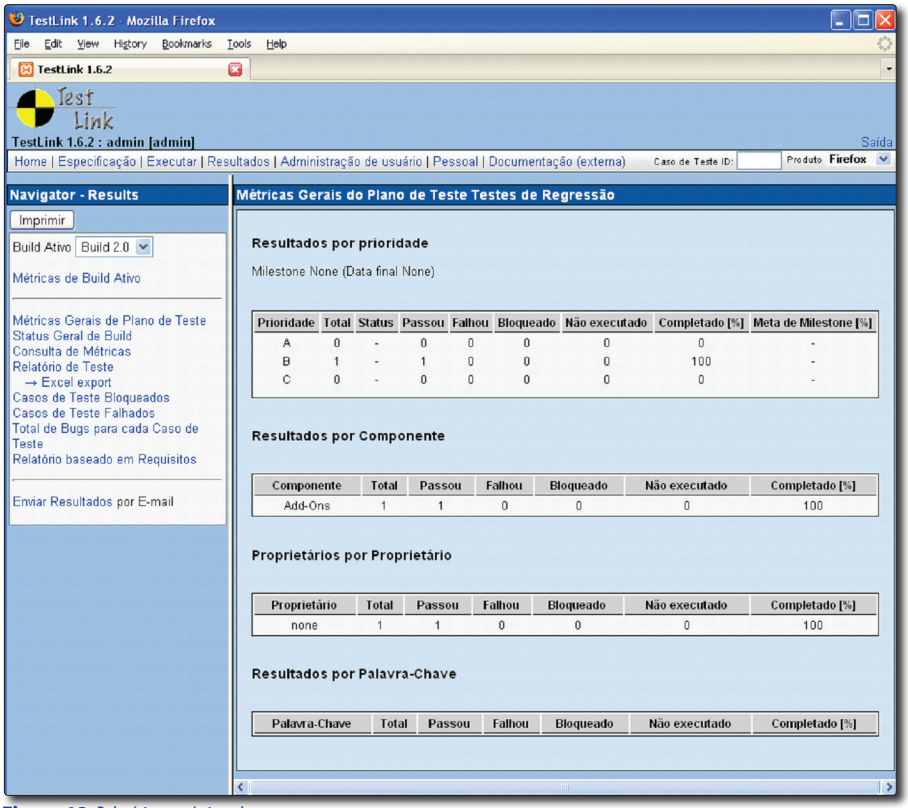


Figura 12. Relatórios e métricas de teste.

Dê seu feedback sobre esta edição!

A Engenharia de Software Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/esmag/feedback

Links

Site do TestLink
http://www.teamst.org/

Norma ISO/IEC 9126
http://pt.wikipedia.org/wiki/ISO_9126

Modelo FURPS
http://en.wikipedia.org/wiki/Furps

IEEE Std 829-1998 - Standard for Software Test Documentation
http://en.wikipedia.org/wiki/IEEE_829

RUP – Rational Unified Process
http://pt.wikipedia.org/wiki/Rational_Unified_Process

Qualidade de Software – Uma Necessidade
http://www.fazenda.gov.br/ucp/pnafe/cst/arquivos/Qualidade_de_Soft.pdf

Open Source	Comercial
rth http://www.rth-is-quality.com	**RSI/QA-Teste** http://www.rsinet.com.br/modules/content/index.php?id=9
TestMaster http://testmaster.sourceforge.net/	**TestLog** http://www.testlog.com/
Testitool http://majordojo.com/testitool/	**Rational** TestManager http://www-306.ibm.com/software/awdtools/test/manager/
Testopia http://www.mozilla.org/projects/testopia/	**Mercury Quality Center** http://www.mercury.com/us/products/quality-center/

Tabela 5. Ferramentas de gestão de testes alternativas.

Chegou o Sistema de Créditos DevMedia

Agora o **conteúdo completo** do nosso site está **ao seu alcance!**

A partir de agora todas as **2000 vídeo-aulas** do site DevMedia podem ser compradas individualmente.

Economia - Você não precisa mais assinar oito produtos diferentes para ter acesso ao conteúdo completo do site!

Todas as vídeos que você sempre quis ver a partir **R$ 0,75!**

Saiba mais sobre o Sistema de Créditos!

DevMedia GROUP

www.devmedia.com.br/creditos