engenharia de software magazine

Ano: 01 – Edição 02

DevMedia GROUP

ISSN 1983127-7

Gestão de Projetos de Software
Conheça alguns dos principais conceitos envolvidos

Gerência de Configuração
Desenvolva software de forma eficiente e disciplinada

Planejamento
Conheça os principais conceitos envolvidos na gerência de riscos

Processo
MPS.BR – Mitos e Verdades de um Modelo de Maturidade

Análise de Pontos de Função

Entenda os principais conceitos e como estimar o tamanho de seus projetos de software

Projeto
Entenda os principais conceitos de SOA – Service Oriented Architecture

Projeto
Aprenda a construir diagramas da UML com base em bons princípios de modelagem OO

Requisitos
Desenvolvimento de software dirigido por casos de uso

Requisitos
Conheça algumas das principais técnicas para apoiar a identificação de requisitos

Caro Leitor,
A partir desta edição a Engenharia de Software Magazine disponibilizará para seus assinantes, em complemento ao conteúdo da revista em artigos, um conjunto de vídeo aulas apresentando temas relacionados aos diferentes assuntos da Engenharia de Software. Estas vídeo aulas estão disponíveis para download no Portal da Engenharia de Software Magazine e certamente trarão uma significativa contribuição para seu aprendizado. Para esta segunda edição, temos um conjunto de 11 vídeo aulas totalizando mais de 3 horas de conteúdo. 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:

Título: Introdução à Engenharia de Requisitos - Parte 01
Mini-Resumo: Esta vídeo aula é parte de um curso de introdução à engenharia de requisitos. Nesta primeira parte serão apresentados alguns conceitos básicos de engenharia de software e cenário atual de desenvolvimento.

Título: Introdução à Engenharia de Requisitos - Parte 02
Mini-Resumo: Esta vídeo aula é parte de um curso de introdução à engenharia de requisitos. Nesta segunda parte serão apresentadas a definição de requisitos e sua importância no contexto de projetos de desenvolvimento de software.

Título: Introdução à Engenharia de Requisitos - Parte 03
Mini-Resumo: Esta vídeo aula é parte de um curso de introdução à engenharia de requisitos. Nesta terceira parte será apresentada uma visão geral da engenharia de requisitos.

Título: Introdução à Engenharia de Requisitos - Parte 04
Mini-Resumo: Esta vídeo aula é parte de um curso de introdução à engenharia de requisitos. Nesta quarta parte daremos continuidade à apresentação da visão geral sobre engenharia de requisitos.

Título: Introdução à Engenharia de Requisitos - Parte 05
Mini-Resumo: Esta vídeo aula é parte de um curso de introdução à engenharia de requisitos. Nesta quinta parte serão apresentados alguns dos objetivos da engenharia de requisitos.

Título: Introdução à Engenharia de Requisitos - Parte 06
Mini-Resumo: Esta vídeo aula é parte de um curso de introdução à engenharia de requisitos. Nesta sexta parte serão apresentadas algumas definições relacionadas às atividades de identificação de requisitos, considerando também seus desafios.

Título: Processos da Gerência de Projetos - Parte 01
Mini-Resumo: Esta vídeo aula apresenta os cinco grupos de processo da gerência de projetos: iniciação, planejamento, execução, controle e encerramento. Nesta primeira parte, são destacados os processos de iniciação e planejamento.

Título: Processos da Gerência de Projetos - Parte 02
Mini-Resumo: Esta vídeo aula apresenta os cinco grupos de processo da gerência de projetos: iniciação, planejamento, execução, controle e encerramento. Nesta segunda parte, são destacados os processos de execução, controle e encerramento.

Título: Orientação a objetos na prática - Implementando classes, atributos e métodos
Mini-Resumo: Nesta vídeo aula trabalharemos com a implementação de conceitos de orientação a objetos em Java utilizando o ambiente BlueJ, abordando classes, atributos e métodos.

Título: Orientação a objetos na prática - Implementando herança e associação 1:1
Mini-Resumo: Nesta vídeo aula trabalharemos com a implementação de conceitos de orientação a objetos em Java utilizando o ambiente BlueJ, abordando herança e associação.

Título: Orientação a objetos na prática - Implementando polimorfismo e associação 1:n
Mini-Resumo: Nesta vídeo aula trabalharemos com a implementação de conceitos de orientação a objetos em Java utilizando o ambiente BlueJ, abordando polimorfismo e associação.

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

Gestão de Projetos de Software

Atividade Essencial à Engenharia 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 do 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 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.

Uma quantidade significativa das atividades atuais é orientada a equipes e envolve múltiplas organizações, sendo estas características determinantes das atividades futuras de projetos. Lidar com equipes e com ambiente corporativo diverso, visando desenvolvimento de (novos) sistemas ou produtos de software, requer uma habilidade que combina arte e ciência e a isso se denomina gestão de projetos (de software). Três pilares formam a base da gestão de projetos: ter foco no cliente, fazer a equipe trabalhar bem (leia-se de forma produtiva e colaborativa) e administrar os recursos (de tempo, pessoal, financeiro) do projeto. A gestão de projetos de software compreende atividades que visam assegurar que o (sistema ou produto de) software seja entregue ao cliente no prazo pré-definido e esteja de acordo com os requisitos definidos pelo cliente. Essa necessidade da gestão de projetos se deve ao fato do desenvolvimento de software estar sempre sujeito às restrições de qualidade, tempo e orçamento. Além disso, dada a natureza flexível do software, uma boa prática é adotar um processo de desenvolvimento orientado para arquitetura para que as funcionalidades e restrições existentes no projeto e implementação do sistema possam ser adequadamente tratadas. Este artigo caracteriza a gestão de projetos de software, destaca sua importância e mostra como ela pode impactar diretamente no sucesso do projeto e produtividade das empresas do setor.

O Que é Gestão de Projetos ?

Gestão de projetos é um conjunto de práticas que serve de guia a um grupo para trabalhar de maneira produtiva. Ela compreende métodos e ferramentas que organizam as tarefas, identificam sua seqüência de execução e dependências existentes, apóia a alocação de recursos e tempo, além de permitir o rastreamento da execução das atividades e medição do progresso relativo ao

que foi definido no plano de projeto.

Qualquer que seja o projeto com o qual você esteja envolvido, o plano de projeto é um documento essencial o qual o gerente de projeto *´vive e respira´* ao longo de todo o projeto. É obrigatório o seu desenvolvimento e manutenção. O plano de projeto define os marcos de projeto (isto é, os ´milestones´) e as principais atividades necessárias à execução do projeto. Este documento define a data de cada marco de projeto, as atividades associadas, os artefatos gerados e respectivos responsáveis.

Perceba que é, até certo ponto, natural àqueles que desenvolvem novos produtos e sistemas de software iniciarem as atividades de desenvolvimento antes mesmo que eles entendam o que tem de ser feito, ou seja, antes mesmo de saberem qual é o problema a ser tratado. Esse tipo de atitude, comumente, resulta no insucesso de projetos. Estudos apontam que cerca de 40% de insucesso de projetos são devidos ao mau planejamento ou, até mesmo, a inexistência de plano de projeto, enquanto que quase 20% se deve a uma gestão inadequada do projeto. Interessante destacar que aproximadamente 50% dos problemas e insucesso de projeto se devem a uma *´pobre´* comunicação entre os principais envolvidos no projeto (ou seja, os *stakeholders*). Num projeto, as questões essenciais que um gerente deve se fazer são:

1. Que problema precisamos solucionar?
2. Quais recursos necessito para resolver?
3. Quanto tempo disponho para o projeto e implementação da solução?

A lição que fica é: *sem o entendimento completo do problema a ser tratado e um bem elaborado plano de projeto em mãos, você (gerente) e sua equipe não saberão onde querem e precisam chegar.* A conseqüência é você (juntamente com sua equipe) deparar-se com a inserção de defeitos logo cedo no desenvolvimento, os quais virão, apenas bem mais tarde, a serem descobertos, resultando em atraso no projeto e/ou comprometendo a qualidade do produto final.

Essas situações apontadas ocorrem, com freqüência, quando não há qualquer 'preocupação' com a gestão de projeto do software. Essas, dentre outras, são razões pelas quais muitos projetos se transformam em casos de insucesso.

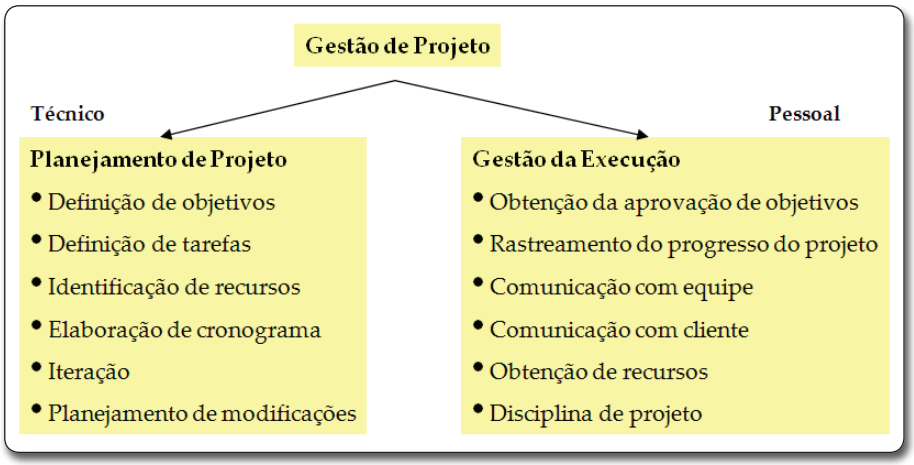


Figura 1. Perspectivas da gestão de projetos.

Por que a Gestão de Projetos é Essencial?

A gestão de projetos é uma atividade ortogonal às demais atividades de projeto e atua como guia para a *boa* execução do projeto. Todas as pessoas envolvidas com um projeto têm a necessidade de acesso às suas informações. Quando lidamos com projeto de médio a grande porte e de natureza complexa, uma atividade chave é a coordenação. Um gerente de projeto precisa coordenar:

- Múltiplas pessoas de formação diversa;
- Múltiplas tarefas onde ocorre relação de dependência;
- Uso de múltiplos recursos (como equipamentos, ferramentas, laboratórios);
- Decisão e aprovação em múltiplos pontos num projeto;
- Alocação adequada de recursos humanos e financeiros a tarefas.

Portanto, é preciso dar visibilidade e compartilhar informações de projeto, pois as decisões precisam ser tomadas com base de informações bem entendidas e explicitadas. A qualidade resultante de um produto ou sistema é determinada a partir do início de seu desenvolvimento. Uma criteriosa análise, feita logo cedo no projeto, visa encontrar erros, identificar inconsistências e averiguar quão correto e completo é o entendimento do problema e adequada é a solução trabalhada. Isto torna a gestão de projeto uma atividade essencial à execução de projetos e sucesso de produtos. A gestão de projetos de software pode ser vista sob duas perspectivas: técnica e pessoal onde a ênfase se dá sobre atividades de planejamento e execução, conforme ilustrado na **Figura 1**.

Perceba que a visão ilustrada na **Figura 1** é uma visão simplificada das reais necessidades de projetos. Dominar as habilidades necessárias a uma boa gestão de projetos requer tempo, experiência e reciclagem. Embora algumas pessoas sejam levadas a acreditar que a capacidade para fazer a gestão de projetos seja um mito, isso não passa de uma falácia. Não se trata de característica inata que o indivíduo traz consigo, mas de um conjunto de habilidades que podem ser reconhecidas, classificadas e desenvolvidas pelas pessoas.

O reconhecimento desses fatos tem motivado os profissionais a buscarem atualizar-se, bem como às empresas a considerarem a gestão de projetos no plano estratégico da instituição. Hoje em dia, profissionais e empresas buscam a capacitação em gestão de projetos que seguem as orientações da principal referência mundial no assunto: o PMI (Project Management Institute) (ver seção **Links**) o qual tem tido um crescimento quase exponencial no número de afiliações, que hoje conta com mais de 240 mil profissionais membros.

Gestão de Projetos e PMI

O PMI (Project Management Institute) é uma instituição sem fins lucrativos, criada em 1969, que tem como principal objetivo contribuir para melhoria contínua da gestão de projetos. O PMI tem sido responsável por catalogar e divulgar o conhecimento e práticas de gestão de projetos visando melhorar a taxa de sucesso de execução de projetos e assim melhor capacitar os profissionais envolvidos (isto é, gerentes de projeto).

Uma das principais metas do PMI é a de continuamente ampliar o corpo de conhecimento da área e para tanto, diversos eventos tem sido promovidos a nível mundial para promover a prática de se fazer uma ´boa´ gestão de projeto. Esse esforço de identificar as melhores práticas de gestão de projetos juntamente com sua divulgação por meio do catálogo do conhecimento em gestão de projetos, conhecido como PMBOK (Project Management Body of Knowledge), tem trazido resultados satisfatórios quanto melhoria na condução de projetos.

O PMBOK compreende um guia contendo todo corpo de conhecimento de práticas tradicionais, avançadas e inovadoras em gestão de projetos, ou seja, o PMBOK serve como guia que contém um conjunto de diretrizes para gestão de projetos e uma estrutura de como ela é decomposta em áreas de conhecimento. Segundo as diretrizes do PMBOK, a gestão de projetos compreende um conjunto de processos que contém áreas que constituem o corpo do conhecimento da gestão de projetos.

Cinco fases compõem a gestão de projetos: inicialização, planejamento, execução, controle e encerramento. A *inicialização*, como o próprio nome já diz trata de iniciar um novo projeto onde um gerente de projeto é designado, são identificados os principais envolvidos e interessados (isto é os *stakeholders*), além de finalizar o documento de caso de negócio que trata da solução a ser implementada. No *planejamento*, o gerente de projeto define o escopo do projeto, a equipe envolvida, a WBS (Work Breakdown Structure, quando se define o escopo e organização de todo o projeto numa estrutura de árvore), um conjunto das atividades mais prioritárias e um cronograma de projeto. A fase de *execução* é dedicada à implementação da solução definida para o projeto, no uso adequado dos recursos e liderança junto à equipe. O *controle* é uma importante fase já que o gerente monitora o andamento do projeto, os resultados alcançados, os artefatos gerados, todo e qualquer desvio que possa ocorrer em relação ao que foi definido no plano de projeto e, estas informações, são usadas para produzir relatório de status e progresso. Na fase de *encerramento*, o gerente é encarregado de obter a aceitação final do cliente, avaliar e relatar as lições aprendidas na execução do projeto.

A **Tabela 1** destaca as três fases mais importantes. Perceba que há um conjunto de nove *áreas de conhecimento* (integração, escopo, tempo, custo, qualidade, recursos humano, comunicações, riscos e aquisições).

Uma das atividades que requer devida atenção do gerente de projetos é a gerência de riscos que impacta diretamente sobre as demais atividades. A ocorrência e a natureza dos riscos podem acarretar em dificuldades sérias no projeto envolvendo elevação de custos projeto e atraso, dentre outros. Tais fatores são indesejáveis ao cliente e podem comprometer significativamente o sucesso do projeto. Esta atividade, especificamente, será tratada na seção seguinte.

Cabe aqui destacar que um dos documentos mais importantes para a gestão de projetos é o plano de projeto pertinente à gerência de integração. O plano de projeto é um guia essencial à condução do projeto sem o qual o gerente fica perdido. Com o plano, há um entendimento dos riscos e compromissos inerentes. O plano constitui uma base para execução sistemática do projeto além de servir como mecanismo eficiente de comunicação entre membros da equipe e entre empresa desenvolvedora e (empresa) cliente.

Um plano de projeto, geralmente, contém um conjunto de informações que permitirá ao gerente não apenas executar o projeto, mas também monitorar seu progresso. **A Tabela 2** apresenta uma relação dos itens que são imprescindíveis de compor um plano de projeto. Note que não há aqui a intenção de ser completo. Entretanto, os itens relacionados na **Tabela 2** são considerados como obrigatórios num plano de projeto de empresa.

Áreas de Conhecimento	Fases		
	Planejamento	Execução	Controle
Integração	Desenvolvimento de plano de projeto	Execução de plano de projeto	Controle integrado de mudanças
Escopo	Planejamento do escopo		Verificação do escopo
	Detalhamento do escopo		Controle de mudança do escopo
Tempo	Definição das atividades		Controle do cronograma
	Seqüenciamento das atividades		
	Estimativa de duração das atividades		
	Desenvolvimento do cronograma		
Custo	Planejamento dos recursos		Controle de custo
	Estimativa dos custos		
	Projeção de orçamento		
Qualidade	Planejamento da qualidade	Garantia da qualidade	Controle da qualidade
Recursos humanos	Planejamento organizacional	Desenvolvimento da equipe	
	Montagem da equipe		
Comunicações	Planejamento das comunicações	Distribuição das informações	Relato de desempenho
Risco	Planejamento dos riscos		Controle e Monitoração de riscos
	Identificação dos riscos		
	Análise qualitativa dos riscos		
	Análise quantitativa dos riscos		
	Planejamento de respostas a riscos		
Aquisições	Planejamento das aquisições	Requisição de propostas	
	Preparação das aquisições	Seleção de fornecedores	
		Administração de contratos	

Tabela 1. Conhecimento da gestão de projetos baseado no PMI.

Gerência de Riscos

Risco é entendido como a probabilidade de que uma situação indesejável irá acontecer. Nesse sentido, a gerência de riscos está interessada em identificar riscos e buscar mecanismos que possam atenuar ou, até mesmo, eliminar os riscos. Os riscos podem ser de três naturezas: riscos de projeto, de produto ou de negócios. O

foco desta seção recai, especificamente, no primeiro. Questões que motivam decisões e são consideradas na gerência de riscos compreende:

- Como mudanças de requisitos ou tecnologia podem afetar o cronograma e o sucesso de um projeto?
- Quais métodos ou ferramentas devem ser empregados num projeto?
- Quais atributos da qualidade devem ser priorizados?
- Quantas pessoas devem ser alocadas no projeto e quais suas habilidades mínimas?

Em muitas empresas, é comum haver a gestão de projetos sem a devida atenção à gerência de riscos. Como não há a prática da gerência de riscos, em tais casos, os riscos costumam ser identificados e analisados de forma aleatória, sem quaisquer disciplinas. Geralmente, o que é feito é apenas um *brainstorming* que pode resultar na não antecipação de futuras ocorrências que podem impactar o produto ou sistema que está sendo desenvolvido.

A boa prática da gestão de projetos recomenda a inclusão no processo de engenharia de sistemas de software da gerência de riscos que compreende quatro atividades:

1. *Identificação de riscos* onde se busca identificar riscos de projeto ou de negócios. Para cada risco identificado, associa-se uma magnitude de risco que serve para indicar o grau de severidade e, portanto, de prioridade de tratamento do risco.
2. *Análise de risco* que visa obter a probabilidade de ocorrência do risco e correspondente impacto sobre o projeto. Note que o impacto pode ser o atraso no projeto, que é tanto indesejável ao cliente quanto implica em custo adicional.
3. *Administração de risco* que tem duas metas: (a) desenvolver uma estratégia de controle que serve para mitigar ou reduzir o impacto de um risco, e (b) elaborar um plano de contingência o qual recomenda as decisões (alternativas) a serem tomadas caso o risco aconteça.
4. *Monitoração de risco* que faz uso de indicadores com o objetivo de monitorar e detectar a ocorrência ou probabilidade de ocorrência de um risco.

Diversas fontes de riscos são consideradas num projeto de software. Dentre elas podemos destacar o conjunto ilustrado na **Tabela 3**.

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.
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 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 2. Relação de itens de um plano de projeto.

Tipo de Risco	Exemplo
Equipe	Membro chave da equipe não estará disponível quando necessário.
Escopo	Falta de clareza no escopo resultará em várias mudanças de escopo.
Gerência	Falta de experiência do gerente de projeto resultará em atraso no cronograma.
Tecnológico	A tecnologia empregada é relativamente nova e pouco conhecida dos membros da equipe que pode resultar em atraso.
Equipamento	Dispositivos necessários não serão entregues no prazo programado.
Cliente	Recursos do cliente não estarão disponíveis como planejado.
Físico	Um vírus de computador infectará o ambiente de desenvolvimento do projeto.
Entrega	Os requisitos da capacidade do sistema excederão a capacidade disponível.

Tabela 3. Possíveis riscos de projeto.

Para os riscos apresentados, é importante identificar a probabilidade de sua ocorrência, o impacto que eles podem trazer (ao projeto), o grau de severidade e como eles podem ser administrados. Por exemplo, se considerarmos o risco da perda de um membro importante da equipe, isto pode ser diagnosticado quando ocorrem reclamações por parte do membro da equipe, bem como quando ele solicita ser realocado para outra atividade. Em outras situações, outros gerentes na mesma empresa consultam sobre a disponibilidade de membros de sua equipe. Em tais circunstâncias, o gerente de projeto deve fomentar junto ao grupo o forte senso de trabalho em equipe e também destacar a importância do projeto e a participação dos membros naquela equipe.

Gerente de Projetos – Também um Líder

Um conjunto de itens que auxilia o sucesso de um projeto compreende: objetivos 'claros' do produto ou sistema a ser desenvolvido, escopo bem delimitado, uso de infra-estrutura de software padrão, uso de estimativas confiáveis, apoio da alta direção, envolvimento de usuários e experiência do gerente.

Aqui, vale ressaltar que o gerente tem um papel de suma importância para o sucesso do projeto que envolve a aplicação de conhecimentos, habilidades, e técnicas para projetar atividades que visem atingir os objetivos do projeto. O gerente acompanha o projeto fazendo uso de processos tais como: iniciação, planejamento, execução, controle e encerramento. Além disso, ele tem de conceber e manter um 'esquema' de trabalho que lhe permita alcançar as metas de negócio e executar efetivamente as atividades que se encontram estabelecidas no plano de projeto (escopo, riscos, cronograma, qualidade, dentre outras).

Todavia, um excelente profissional não tem apenas o papel de gerente. Ele é bem mais que isso. Ele é também um líder de uma equipe e hábil negociador. Portanto, a gestão de projetos deve ser orientada para o cliente (como apontado no início desse artigo) e considerar:

- Pensamento estratégico corporativo;
- Valor do cliente e estratégia de relacionamento;
- Táticas e estratégias de negociação;
- Análise e planejamento estratégico;
- Indicadores de desempenho e mercadológico.

O gerente de projeto é uma peça fundamental para coordenação da execução de um projeto. Ele precisa não apenas conhecer, mas também externar o pensamento da corporação de modo a 'atrair' e reter o cliente. Precisa ainda saber como se relacionar com o cliente (do projeto). Nesse sentido, o gerente precisa dispor de táticas de negociação. Adicionalmente, indicadores de desempenho e mercadológicos podem ser considerados.

O gerente de projeto é considerado um *líder*, isto é, uma pessoa visionária, quando o rumo (do projeto) não é conhecido. Também é um *colaborador* quando o consenso é necessário e ele é capaz de motivar aqueles que estão ao seu redor de forma a canalizar os esforços da equipe para realização e sucesso do projeto. Liderança é, portanto, uma característica essencial para os gerentes de projeto. Para tanto, é necessário conhecer a cultura organizacional e o desejo de mudanças. Nesse sentido, vale lembrar um pensamento do lendário Peter Drucker que dizia: "a coisa mais importante na comunicação é ouvir o que não é dito". Pense nisso.

Conclusão

Quando as organizações têm dificuldade em entregar produtos ou sistemas de qualidade, dentro do prazo e orçamento previamente estabelecidos, e que satisfaçam às necessidades de seus usuários, isto é sinal da falta de gestão de projetos. Por outro lado, a adoção da disciplina da boa gestão seguindo as diretrizes do PMI tem resultado na melhoria da prática da engenharia de software uma vez que o desenvolvimento de sistemas ou produtos de software é realizado de forma sistemática e com adequada monitoração dos recursos alocados. Além disso, soluções de problemas que têm sido descobertas e testadas, também têm sido catalogadas, aumentando o 'corpo de conhecimento' da engenharia de software. Essa catalogação de práticas de engenharia de software e, especificamente, da gestão de projetos constitui um sinal de amadurecimento da área. ●

Links

The Project Management Institute
www.pmi.org

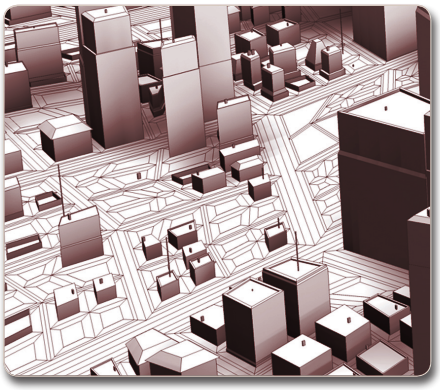
PMBOK – Project Management Body of Knowledge
www.projectsmart.co.uk/pmbok.html

The Project Management Forum
www.pmforum.org.

Integrated Online Library of Project Management
www.managementhelp.org/plan_dec/project/project.htm

Project Management na Wikipedia
en.wikipedia.org/wiki/Project_management

The Project Management Podcast
www.thepmpodcast.com



kali software

Desde 2004 Trazendo Teoria para a Prática

Treinamento e Consultoria em Engenharia de Software

Profissionais com ampla experiência prática (Consusltores Certificados) e acadêmica (Professores Universitários, Mestres e Doutores)

Know-how para implementar processos de software de acordo com modelos de qualidade (MPS e CMMI) e maximizar o retorno de investimento

Cursos 2008: *Inscrições Abertas*

Certificados emitidos pela Kali Software

Junho | Julho	Processos de Software com ênfase em CMMI e MPS – **24 horas**
Julho | Agosto	Engenharia de Requisitos – **24 horas**
Julho | Agosto	Projeto Orientado a Objetos com UML e Padrões – **24 horas**

Confira as ementas, investementos e condições de pagamento: *www.kalisoftware.com*
Oferecemos também cursos de programação (Java, Java para Web e Ajax)
Inscrição e outras informações: *info@kalisoftware.com*

kalisoftware.com | info@kalisoftware.com

Visão da Gerência de Riscos na Engenharia 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.

As organizações vivem atualmente grande competitividade comercial, demandando rápidas decisões, melhor alocação de recursos e uma clara definição de foco. Em um ambiente de desenvolvimento de software típico não é diferente. Vários tipos de projetos são propostos, com diferentes objetivos, em que é preciso gerenciar estrategicamente de acordo com as metas organizacionais.

Em gerenciamento de projetos, para garantir o sucesso das metas organizacionais e dos objetivos definidos do projeto, é preciso identificar fatores positivos e adversos, considerados críticos. Os fatores positivos, também chamados oportunidades, são diferenciais em um ambiente mercadológico e tecnológico competitivo, e o tratamento dos fatores adversos, tidos como riscos, favorecem o alcance das oportunidades identificadas.

Os riscos possuem diferentes significados, como os de ordem física, estrutural, econômica, social e ambiental. Podendo ainda se desdobrar em diversos componentes e sucessivos níveis de detalhamento.

As incertezas fazem parte do cotidiano humano. Desde os primórdios o homem procura defender-se dos riscos que o cercam, galgando níveis de satisfação das necessidades básicas, de segurança e culminando nas necessidades de cunho puramente profissional. As pessoas, em sua grande maioria, diariamente fazem escolhas, com graus diferenciados de riscos, mas também com um alto grau de oportunidade e benefícios associados.

Atualmente todos os ramos da atividade humana dependem de alguma forma da utilização de software para operar, dar suporte, controlar equipamentos e fluxos de informações, gravar ou processar atividades. A área de Engenharia de Software tem promovido vários estudos com a finalidade de produzir modelos de melhoria, processos, métodos e ferramentas para aumentar a probabilidade

de sucesso na execução de projetos de software, garantindo a qualidade de seus produtos e minimizando possíveis problemas associados [SEI 2001, PMI 2004]. Portanto, na capacidade de prevenir e controlar essas variáveis pode estar o diferencial para gerir os riscos de projetos na indústria de software.

Todo projeto de software enfrenta problemas de qualidade, de cronograma, e de custo que estão sendo afetados por riscos que são inesperados, não planejados ou ignorados simplesmente pela falta de conhecimento.

Através de perspectivas globais de negócios muitas organizações estão tornando-se cada vez mais dependentes do sucesso ou do fracasso dos softwares que desenvolvem. Neste contexto, a Gerência de Riscos não é apenas baseada em boas práticas para o desenvolvimento de software, mas sim, boas práticas para gerir negócios.

Riscos em Engenharia de Software

Ainda que o conceito de risco esteja bastante associado a perigos e impactos negativos, já vem sendo utilizado como "exposição a conseqüências da incerteza", sendo cada vez mais aplicado tanto no gerenciamento de perdas como no de ganhos potenciais. Nas próximas seções serão tratados os conceitos relativos à incerteza, oportunidade e riscos.

Incerteza, Oportunidade e Risco

Um fator comum que provoca preocupação na análise de projetos e no seu investimento é a incerteza. Surge como uma das conseqüências da falta de controle absoluto dos eventos que acontecerão num futuro próximo. Pode-se fazer a previsão sobre o comportamento de determinados eventos, mas não se pode determinar exatamente quando e em que intensidade eles deverão ocorrer. Exemplos desses eventos são os comportamentos futuros da economia de um país, as vendas futuras de determinados produtos e sua aceitação no mercado, o desgaste e custos de manutenção de equipamentos, entre outros.

Considerando que a noção de risco associada à possibilidade de dano, perda ou estrago é de conhecimento claro e direto, alguns autores fazem uma distinção teórica entre o risco e incerteza. Marshall [Marshall 2002] e Knight [Knight 1921] diferenciam risco – resultados que, embora não certos possuem probabilidades quantificáveis pela experiência ou dados estatísticos e para a qual é possível fazer uma estimativa – de incerteza, quando a ausência de experiências ou ocorrências anteriores impossibilita quantificar adequadamente o resultado. No entanto, conforme M. H. Simonsem [Simonsem 1994], em seu livro Dinâmica Macroeconômica (1994: p. 399), "*Risco é quando a variável aleatória tem uma distribuição de probabilidades conhecida e, Incerteza, quando essa distribuição é desconhecida*".

No contexto da Gerência de Projetos, o risco de projeto pode ser definido como o efeito cumulativo das incertezas que adversamente afetam os objetivos do projeto. Em outras palavras, é o grau de exposição a eventos negativos e a probabilidade de ocorrência e seu impacto nos objetivos do projeto, expressos em termos de escopo, custo, prazo e qualidade.

A meta principal do gerenciamento de riscos de projeto é afastar as incertezas relacionadas aos riscos e direcionar os projetos para oportunidades. Outra forma de tratar os riscos é listar fatores de riscos que são variáveis que podem tornar-se riscos em um baixo, médio ou alto grau de incidência no ambiente.

Categorias e Tipos de Riscos

Risco na área de software foi representado de forma sistemática por Barry W. Boehm, em 1988, através do Modelo Espiral [Boehm et al 2004], que tem como princípio ser incremental e dirigido à análise de riscos. O desenvolvimento incremental é uma estratégia para a obtenção de progresso em pequenos passos, pela divisão de um problema em subproblemas e a posterior combinação das soluções encontradas (alternativas definidas).

Atualmente, a área que trata riscos na Engenharia de Software evoluiu, passando de uma análise dentro das fases do modelo de desenvolvimento de software para se tornar uma gerência que permeia todos os processos do ciclo de vida do software (o ciclo de vida do software vai desde a concepção de idéias até a descontinuidade do produto de software).

Risco de software pode ser caracterizado como [PMI 2004]:

- **Riscos de Projeto de Software**. Define os parâmetros operacionais, organizacionais e contratuais do desenvolvimento de software. Inclui limites de recursos, interfaces, relacionamentos com fornecedores ou restrições de contratos.
- **Riscos de Processo de Software**. Relacionam-se os problemas técnicos e de gerenciamento. Nos procedimentos de gerência podem-se encontrar riscos em atividades como: planejamento, definição e contratação de equipe de trabalho, garantia de segurança e configuração de gerência. Nos procedimentos técnicos, podem-se encontrar riscos nas atividades: análise de requisitos, projeto, codificação e testes, por exemplo.
- **Riscos de Produto de Software**. Contém as características intermediárias e finais do produto. Estes tipos de riscos têm origens nos requisitos de estabilidade do produto, desempenho, complexidade de codificação e especificação de testes.

Muitas classificações são encontradas na literatura relacionada à Gerência de Projetos [Gustafson 2002, PMI 2004], uma delas é o uso de taxonomia de riscos

Categoria	Descrição
Técnico e Desempenho	Riscos associados a aspectos técnicos do projeto. Pode estar relacionado ao grau de inovação tecnológica do projeto em questão.
Negócio	Risco associado ao marketing ou ao tempo de lançamento de releases dos produtos ou novas versões. Também pode estar associado às informações dos competidores.
Gerência de Projeto	Riscos associados com o processo de gerenciamento de projeto, maturidade organizacional e habilidade.
Processo	Riscos associados ao processo de negócio ou outro processo que possa impactar a organização, o usuário (cliente) ou o projeto.

Tabela 1. Exemplos de Categorias de Riscos

(Taxonomy-Based Risk Identification) como a apresentada pelo Instituto de Engenharia de Software (SEI - Software Engineering Institute), onde os riscos são classificados dentro de categorias para melhor entendimento de sua natureza. Alguns estudos e abordagens na literatura, que tratam a área de Gerência de Riscos, evoluíram, ou mesmo adaptaram, as categorias de riscos inicialmente apresentadas na taxonomia proposta pelo SEI [Costa et al 2005, Gusmão et al 2005 e Webster et al 2005].

As categorias de riscos favorecem a identificação dos riscos em um ambiente, promovendo uma classificação e organização dos riscos identificados em grupos lógicos. Alguns exemplos de categorias de riscos e suas abrangências estão relacionados na **Tabela 1**.

Não existe uma diferenciação clara entre os termos "classificação e categorização de riscos" na literatura de Engenharia de Software. Em muitas abordagens são utilizados como sinônimos. O próprio Guia PMBOK (Project Management Body of Knowledge), tanto na sua segunda quanto terceira edições, além de sub-classificar os riscos, os organiza em categorias [PMI 2004].

De forma geral, nos modelos de processos de gerenciamento de riscos é solicitada a definição das categorias ou classes de riscos (classificações) de acordo com as características do ambiente de desenvolvimento em questão.

Gerência de Riscos e Tomada de Decisão

A incerteza é um fator que pode dificultar a tomada de decisão racional. A maioria das decisões, sobretudo aquelas importantes, está baseada em algum tipo de estimativa, colocando a incerteza como elemento do processo decisório. E mesmo que a situação não exija estimativa, deve-se considerar a insuficiência das informações para a tomada de decisão.

Um risco só deveria ser tratado quando o seu benefício potencial (oportunidade) para a organização e as chances de ganhos excedesse o valor do custo reparador de uma decisão mal sucedida e as chances de perdas dentro de uma margem satisfatória. Para que isso possa acontecer é preciso que o gerente de projeto, ou o profissional responsável pelo risco, encontre as respostas para algumas questões, tais como o motivo pelo qual o risco deve ser tratado; quais são os ganhos associados ou o que poderá ser perdido; as reais chances de sucesso (ou fracasso); o que poderá ser feito se os objetivos definidos não forem alcançados; se o custo de estratégias de tratamento vale o esforço para um risco em específico.

Dessa forma, torna-se importante fazer uma análise do grau de incerteza existente no processo de decisão, ou seja, procurar uma estimativa dos riscos envolvidos. Deve-se ressaltar a importância do uso de abordagens qualitativa e quantitativa na análise e desenvolvimento dos investimentos, de uma empresa, seja ela de pequeno, médio ou grande porte [Moura et al 2004].

A evolução das organizações, a complexidade das demandas dos mercados e o processo de globalização tornaram a utilização das práticas das finanças, as estratégias e o marketing cada vez mais desafiantes. Dominar e compreender a aplicação de técnicas de análise de oportunidade e incerteza, quantitativamente ou qualitativamente, é imposição real para a sobrevivência organizacional, quer pelas implicações do trabalho assalariado, quer pelas operações de compra e venda, quer pelos investimentos e decisões associadas.

Geralmente, quando se fala em correr riscos, pensa-se logo em perdas ou sacrifícios financeiros. Muitos riscos fazem parte de nosso dia a dia de tal forma, que mal os levamos em consideração, ao invés disso, as reações muitas vezes são subconscientes.

Ao atravessar uma rua, o pedestre deve olhar para ambos os lados e quando não houver tráfego, atravessará. Caso esteja com pressa, pode assumir riscos e atravessar entre os veículos, quando aparecer uma chance. Se o trânsito estiver pesado, prudentemente, deve se dirigir às áreas de cruzamento de pedestre, garantindo sua passagem com segurança.

Caberá à gerência optar por assumir estes riscos e enfrentar o desafio. Tudo vai depender do contexto não significando que outros riscos não serão visualizados ou percebidos.

Gerência de Riscos *versus* Gerência de Projetos

A Gerência de Riscos tem uma aplicabilidade bastante ampla em ambientes organizacionais. Quando se fala em gerir projetos, é inevitável que se fale em gerir riscos. Por este motivo, muitas vezes, é difícil definir limites entre as duas gerências.

Ainda não existe um consenso sobre o relacionamento entre a gerência de projetos e a gerência de riscos. Normas, padrões e modelos são apresentados, na área de Engenharia de Software, mas

não tratam esse relacionamento de forma explícita. Não existe um padrão que descreva o elo existente entre a Gerência de Riscos e a Gerência de Projeto, nem tão pouco apresente o objetivo do processo e das atividades da Gerência de Riscos. Isto evidencia a imaturidade da comunidade de Engenharia de Software, onde as normas e modelos deveriam consolidar este conhecimento já existente.

Uma das áreas de aperfeiçoamento futuro da Gerência de Projetos, segundo David Hillson [Hillson 2005] é a integração da Gerência de Riscos e a cultura organizacional. Esta visão enfoca o uso da Gerência de Riscos como parte integral do negócio organizacional sendo um processo construtivo e não repreensivo.

A falta de padronização dificulta o entendimento, a definição de processos e papéis em relação à Gerência de Riscos, o cumprimento dos objetivos organizacionais, pois não se tem uma visão integrada. A **Tabela 2** apresenta uma coletânea de visões, capturadas na literatura da área de Engenharia de Software, sobre os possíveis relacionamentos existentes entre a Gerência de Projetos e a Gerência de Riscos de Software.

Visão	Opiniões	Autores
Dependência	Segundo Jyrki Kontio a administração do processo de gerência de risco é responsabilidade do gerente de projeto. O PMI, no Guia PMBOK 2000, embora enfatize que a Gerência de Riscos deva ter um responsável, determina que o gerente de projetos integre todas as áreas de conhecimento Na norma NBR/ISO 10006, que define diretrizes de qualidade na gerência de projetos, considera que a gerência de riscos é uma das preocupações da gerência de projetos.	JyrkiKontio;PMI–Projectmanagement Institute; NBR/ISO 10006
Existência	Implementar Gerenciamento de Risco implica na inserção dos princípios e boas práticas no gerenciamento de projetos de uma organização. A Gerência de Risco é parte integral e vital para a gerência de projetos. Gerenciar projetos é gerenciar riscos.	Barry W. Boehm; Ronald P. Higuera e Yacov Y. Haimes; Tom De Marco e Timothy Lister.
Independência	Robert Charette enfoca a necessidade de uma comunicação aberta (entre níveis organizacionais e dentro dos projetos) como peça chave para um gerenciamento de sucesso. Os padrões ISO/IEC 12207 (versão 2000) e a ISO/IEC 15504 apresentam a Gerência de Risco como um processo separado da gerência de projetos.	Revista Software Tech News – uma publicação do Departamento de Defesa Norte-Americano (volume 2, número 2); ISO/IEC 12207 e ISO/IEC 15504.

Tabela 2. Visões da Gerência de Riscos

De acordo com Stephen Grey existem três visões, de uma forma geral, para o relacionamento entre a Gerência de Riscos e a Gerência de Projeto [Grey 1995]:

• Dependência – A Gerência de Riscos é vista como parte da gerência de projeto. A execução é de responsabilidade do gerente de projeto ou é delegada a outro membro da equipe de projeto. Desta forma, as atividades de gestão de riscos são realizadas em nível de projeto na organização (nível operacional);

• Existência – A motivação para a execução de um projeto é a gerência de seus riscos. A existência da gerência de projetos está condicionada à Gerência

de Riscos. Se não existe risco, não existe necessidade da gerência de projeto. Neste caso tornar-se-ia uma mera atividade administrativa. Este tipo de abordagem é comumente conhecido como "gerência de projetos dirigida a riscos";

• Independência – através desta visão, a Gerência de Riscos é um processo distinto e de apoio para todos os projetos da organização, desde o nível estratégico até o operacional.

Independentemente da visão adotada, a organização deve ter consciência da importância da aplicação dos processos da Gerência de Riscos de forma integrada e contínua, em seu ambiente de desenvolvimento de software.

Considerações Finais

Nos dias atuais, os ambientes organizacionais demandam uma crescente dinamicidade e proatividade nas respostas aos problemas e fatores de riscos associados, isso devido à grande influência exercida pela globalização, pelas inovações e mudanças não só internamente, mas também externamente. Decorrente dessa nova realidade, funções relacionadas à gestão de riscos estão sendo incorporadas nas organizações traduzindo um empenho em manter boas práticas nos ambientes corporativos.

Atualmente, gerir riscos envolve o estabelecimento de uma cultura e infra-estrutura adequadas, bem como a aplicação de uma metodologia lógica e sistemática para administrar os riscos associados a qualquer atividade, função, processo ou projeto.

Uma limitação atual da Gerência de Riscos é a quantidade pequena de estudos práticos divulgados, apresentando indicadores de sucesso dos projetos de software desenvolvidos, conforme relato em pesquisadas disponibilizadas na área [Coelho 2004, Leopoldino 2004].

Cada dia fica mais clara a percepção de que gerenciar projetos é gerenciar riscos. Mas isto não basta. É necessário que o gerente de projetos seja capaz de identificar problemas concretos e, de preferência, com a probabilidade de sua ocorrência. Além disso, há riscos e riscos! Determinados tipos de projetos são intrinsecamente mais arriscados do que outros.

Possuir uma visão integrada do risco é fundamental para as organizações que buscam o desenvolvimento e a continuidade do negócio, e ainda dependem de uma infra-estrutura operacional sob risco controlado.

Uma grande dificuldade atual para o gerenciamento dos riscos do processo de desenvolvimento de software é a existência de diversas visões parciais sobre as técnicas e formas de gerenciamento. Quando transportadas para a prática estas visões podem trazer muitos problemas e ineficiências. Isto porque qualquer desenvolvimento, por maior a hegemonia de um determinado conteúdo tecnológico, implica em conhecimento diversificado. Cada visão parcial carrega consigo também um vocabulário e determinados valores próprios, dificultando a integração entre os diversos profissionais pela ausência de padronização.

Enfrentar esta situação depende da construção de uma imagem única e integrada do processo de gerenciamento, em especial, da Gerência de Riscos. Nisto, percebe-se a importância da disseminação dos conceitos de gestão nos ambientes de desenvolvimento de software, como também a definição de um processo que suporte as atividades de gerenciamento dos riscos dos projetos.

A realização e concretização dos projetos organizacionais estão associadas a eventos inesperados e incertos, é por tal motivo que a Gerência de Riscos tem sua importância, permitindo avaliar a viabilidade das atividades a serem realizadas, bem como os custos e benefício (oportunidades) das mesmas.

Este artigo apresentou uma visão inicial sobre a área de gerenciamento de riscos, através de conceitos e fundamentos que são considerados chave na área de Engenharia de Software. Dentro deste contexto, nas próximas edições, serão apresentados alguns modelos e abordagens para gerenciamento de riscos; em seguida 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. ●

Referências

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

[Coelho 2004] *Coelho, P. G. (2004)* Identificação das Estratégias de Aprendizado utilizadas pelos PMP´s e Aspirantes a Certificação PMP. Projeto PMK – Environment Learning. CIn /UFPE – Centro de Informática – Universidade Federal de Pernambuco.

[Costa et al 2005] *Costa, H. R.; Barros, M. O.; Travassos, G.H. (2005)* Uma Abordagem Econômica baseada em Riscos para Avaliação de uma Carteira de Projetos de Software. In 19º Simpósio Brasileiro de Engenharia de Software. Uberlândia – MG – Brasil.

[Grey 1995] *Grey, S. (1995)* Practical Risk Assessment for Project Management. John Wiley & Sons.

[Gusmão e Moura 2004] *Gusmão, C. M. G. e Moura, H. P. (2004)* Gerência de Risco em Processos de Qualidade de Software: uma Análise Comparativa. Anais do III Simpósio Brasileiro de Qualidade de Software. Brasília – DF – Brasil.

[Gusmão et al 2005] *Gusmão, C. M. G.; Buarque, T.; Valeriano, F. L. (2005)* Ontologia de Domínio de Riscos. Relatório Técnico - Suppera Solutions, Centro de Informática, Universidade Federal de Pernambuco. Recife. Brasil.

[Gustafson 2002] *Gustafson, D. A. (2002)* Theory and Problems of Software Engineering. New York: McGRAW Hill, USA.

[Hillson 2005] *Hillson, D. (2005)* Gerenciamento de Riscos. Revista Mundo PM Nº4. pp 38-42.

[Knight 1921] *Knight, F.H. (1921)* Risk, Uncertainty and Profit. Houghton Mifflin Company, Boston. pp 22-24.

[Leopoldino 2004] *Leopoldino, C. B. (2004)* Avaliação de Riscos em Desenvolvimento de Software. Dissertação de Mestrado. Universidade Federal do Rio Grande do Sul – Escola de Administração. Porto Alegre. Brasil.

[Marshall 2002] *Marshall, C. (2002)* Medindo e gerenciando riscos operacionais em instituições financeiras. Rio de Janeiro: Qualitymark.

[Moura et al 2004] *Moura, H. P.; Gusmão, C. M. G.; Correira, B. C. S. (2004)* Portfolio Management: A Critical View of Risk Factors Balancing. Anais do NORDNET – International PM Conference. Helsinki – Finlândia.

[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.

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

[Simonsem 1994] *Simonsem, M. H. (1994)* Dinâmica Macroeconômica. São Paulo: Atlas. Brasil.

[Webster et al 2005] *Webster, K. P. B. et al. (2005)* Taxonomia de Riscos para Manutenção de Software. In Anais do IV Simpósio Brasileiro de Qualidade de Software. Porto Alegre – RS – Brasil.

Gerência de Configuração de Software

Desenvolvendo software de forma eficiente e disciplinada

Cristine Dantas
É bacharel em Informática pela UFRJ e mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ. Tem experiência na área de Ciência da Computação, com ênfase em Engenharia de Software, atuando principalmente nos seguintes temas: gerência de configuração de software, desenvolvimento baseado em componentes e UML. Atualmente é consultora da PrimeUp, onde desenvolve projetos de consultoria e treinamento em Gerência de Configuração de Software.

Os sistemas de software estão em constante evolução. A manutenção do software, isto é, modificações em artefatos existentes, chega a consumir 75% do custo total do seu ciclo de vida. Aproximadamente, 20% de todo o esforço de manutenção é usado para consertar erros de implementação e os outros 80% são utilizados na adaptação do software em função de modificações em requisitos funcionais, regras de negócios e na reengenharia da aplicação. A Gerência de Configuração de Software surgiu da necessidade de controlar estas modificações, por meio de métodos e ferramentas, com o intuito de maximizar a produtividade e minimizar os erros cometidos durante a evolução.

É uma disciplina que controla e notifica as inúmeras correções, extensões e adaptações aplicadas durante o ciclo de vida do software de forma a assegurar um processo de desenvolvimento e evolução sistemático e rastreável, sendo indispensável quando equipes manipulam, muitas vezes em conjunto, artefatos comuns.

Apesar de existir um forte apelo para o uso da Gerência de Configuração de Software durante a etapa de manutenção, a sua aplicação não se restringe somente a essa etapa do ciclo de vida do software. O uso dos sistemas de Gerência de Configuração é fundamental para prover controle sobre os artefatos produzidos e modificados por diferentes recursos desde o planejamento e levantamento de requisitos até a construção e entrega do produto. O motivo da sua importância está geralmente associado aos problemas identificados quando a Gerência de Configuração não é utilizada no desenvolvimento de software. Abaixo, vamos analisar alguns destes problemas.

Imagine que uma organização desconhece o que seja Gerência de Configuração e que em um determinado projeto um desenvolvedor esteja modificando os artefatos C1, C2 e C3 em um diretório compartilhado na rede. Simultaneamente, um segundo desenvolvedor modifica

os artefatos C4, C5 e também o artefato C3, como exemplifica a **Figura 1**.

Neste cenário, o segundo desenvolvedor não notifica o primeiro desenvolvedor sobre o impacto que a modificação do artefato C3 pode causar no código. Conseqüentemente, o primeiro desenvolvedor, que está usando o mesmo espaço de trabalho, não conseguirá identificar, de forma rápida, o motivo que levou sua implementação a falhar. Este problema acontece pela falta de notificação e pelo compartilhamento de artefatos de software por diversos desenvolvedores.

Imagine que agora foi acordado entre os desenvolvedores que o ideal seria centralizar os artefatos em um repositório e que cada desenvolvedor implementaria suas modificações em um espaço de trabalho privado. Após cada modificação, o artefato seria devolvido ao repositório. Considerando este cenário, freqüentemente ocorreriam sobreposições ou perdas de modificações implementadas nos artefatos comuns nas organizações sem a prática da Gerência de Configuração. Um desenvolvedor poderia implementar sua modificação em uma versão desatualizada do artefato e sobrepor a versão mais atual disponibilizada por outro. Este problema ocorre devido à atualização simultânea, quando dois desenvolvedores compartilham o mesmo repositório e não existe controle ou restrição quanto ao acesso a este repositório (ver **Figura 2**).

Sob a perspectiva de desenvolvimento, a Gerência de Configuração de Software abrange três sistemas principais: controle de modificações, controle de versões e controle de gerenciamento de construção.

O sistema de controle de versões permite que os artefatos sob Gerência de Configuração evoluam de forma distribuída, concorrente e disciplinada, evitando perdas ou sobreposições durante o desenvolvimento e a manutenção do artefato. Podemos citar como exemplos de ferramentas de mercado: CVS, Subversion, IBM Rational ClearCase e Microsoft Visual Source Safe.

O sistema de controle de modificações armazena todas as informações geradas durante o andamento das solicitações de modificação e relata essas informações aos participantes interessados e autorizados. Podemos citar como exemplos de ferramentas de mercado: Bugzilla, Jira,

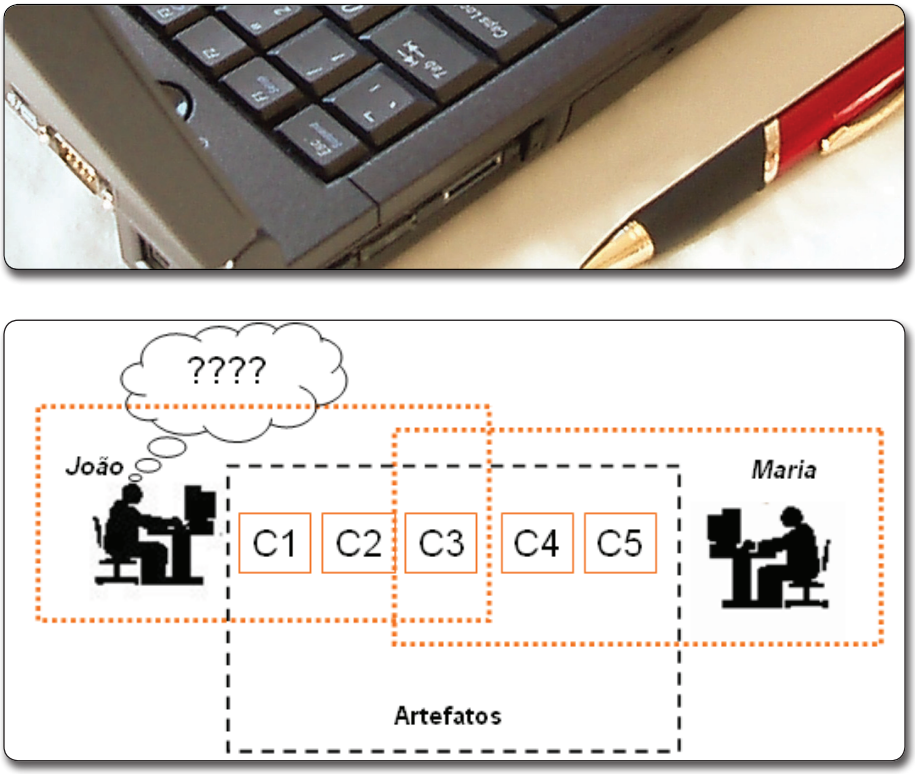


Figura 1. Espaço de Trabalho compartilhado por vários desenvolvedores

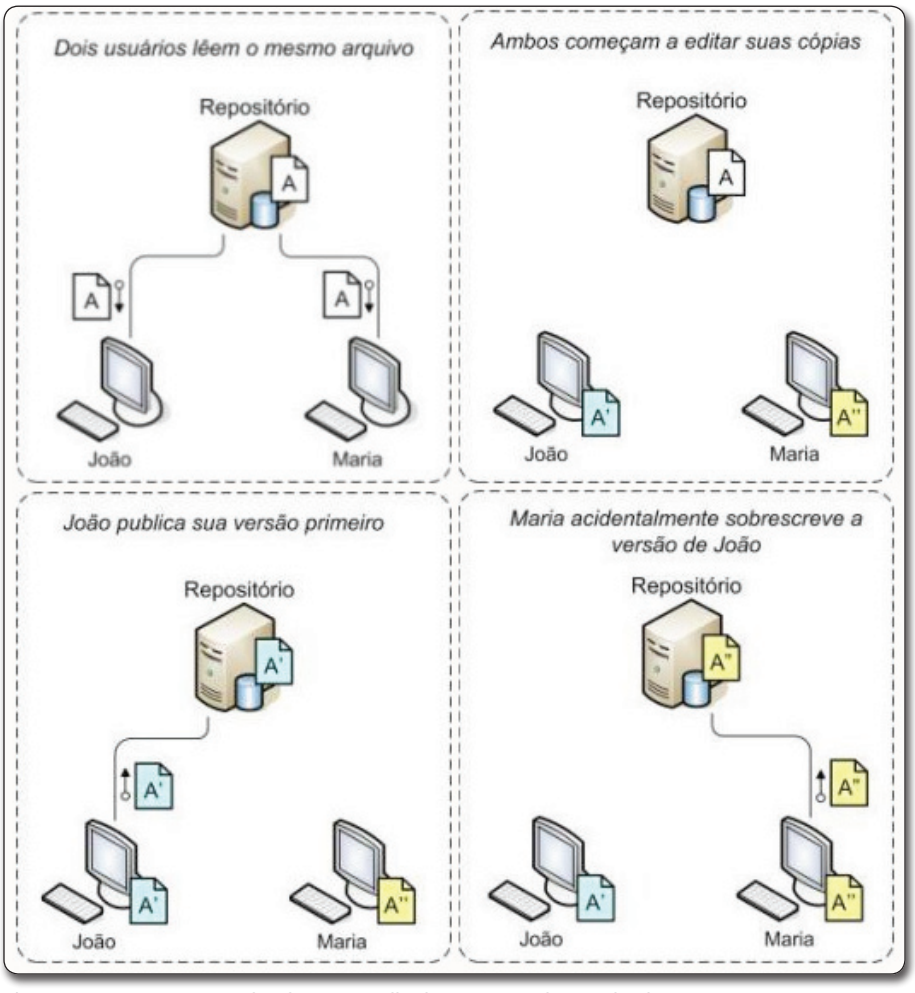


Figura 2. Repositório centralizado compartilhado por vários desenvolvedores

Trac e IBM Rational ClearQuest.

O sistema de gerenciamento de construção automatiza o processo de transformação dos diversos artefatos do software que compõem um projeto em um sistema executável propriamente dito. Este processo é nomeado construção do software que, por exemplo, testa e empacota a aplicação java como um arquivo jar. Este processo ocorre de forma aderente às normas, procedimentos, políticas e padrões definidos para o projeto. Podemos citar como exemplos de ferramentas de mercado: Maven e Apache Ant.

As vantagens da utilização da Gerência de Configuração de Software são inúmeras. Dentre elas, podemos listar: (1) ganho de produtividade e eficiência; (2) diminuição do retrabalho e dos erros; (3) aumento da disciplina no processo de desenvolvimento; (4) aumento da memória organizacional; (5) acesso às informações qualitativas e quantitativas referentes ao processo de desenvolvimento, como por exemplo, medida de esforço para efetuar uma alteração e freqüência de modificações por componente; (6) possibilidade de estabelecer uma trilha de auditoria indicando por que, quando e por quem um artefato foi alterado; (7) auxílio à gerência de projetos e (8) garantia de ambiente estável no qual o produto deve ser desenvolvido.

Neste artigo, veremos alguns conceitos da área de Gerência de Configuração de Software e como esta área se relaciona com o processo de desenvolvimento de software. Também serão apresentadas algumas estratégias de organização do trabalho.

Terminologia

O sistema de controle de versões permite que os artefatos sejam obtidos, por meio de uma operação conhecida como *check-out*, modificados dentro do espaço de trabalho do desenvolvedor e, depois, retornados ao repositório, por meio de uma operação conhecida como *check-in*, como exemplifica a **Figura 3**. O repositório é o local de armazenamento

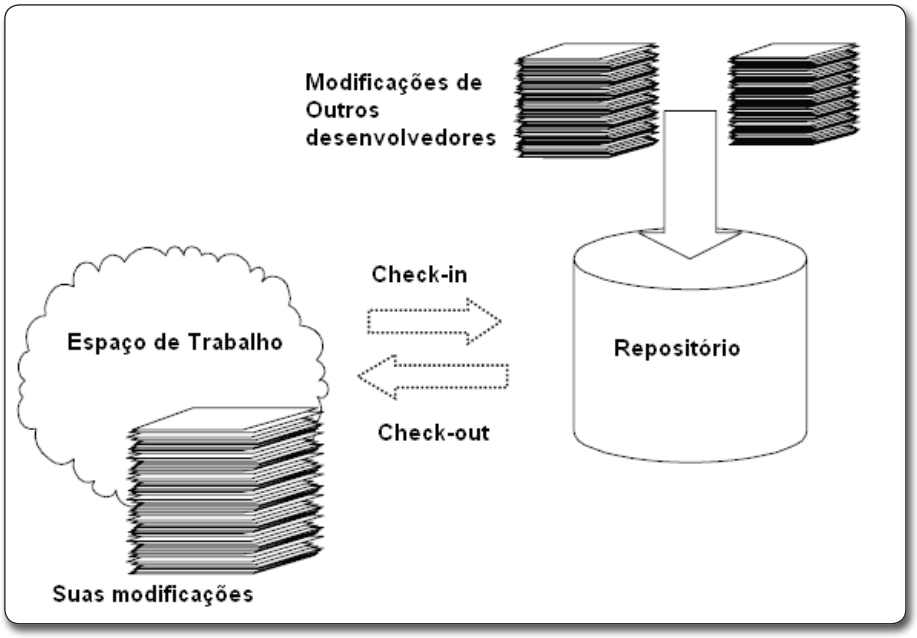


Figura 3. Operações Check-in e Check-out

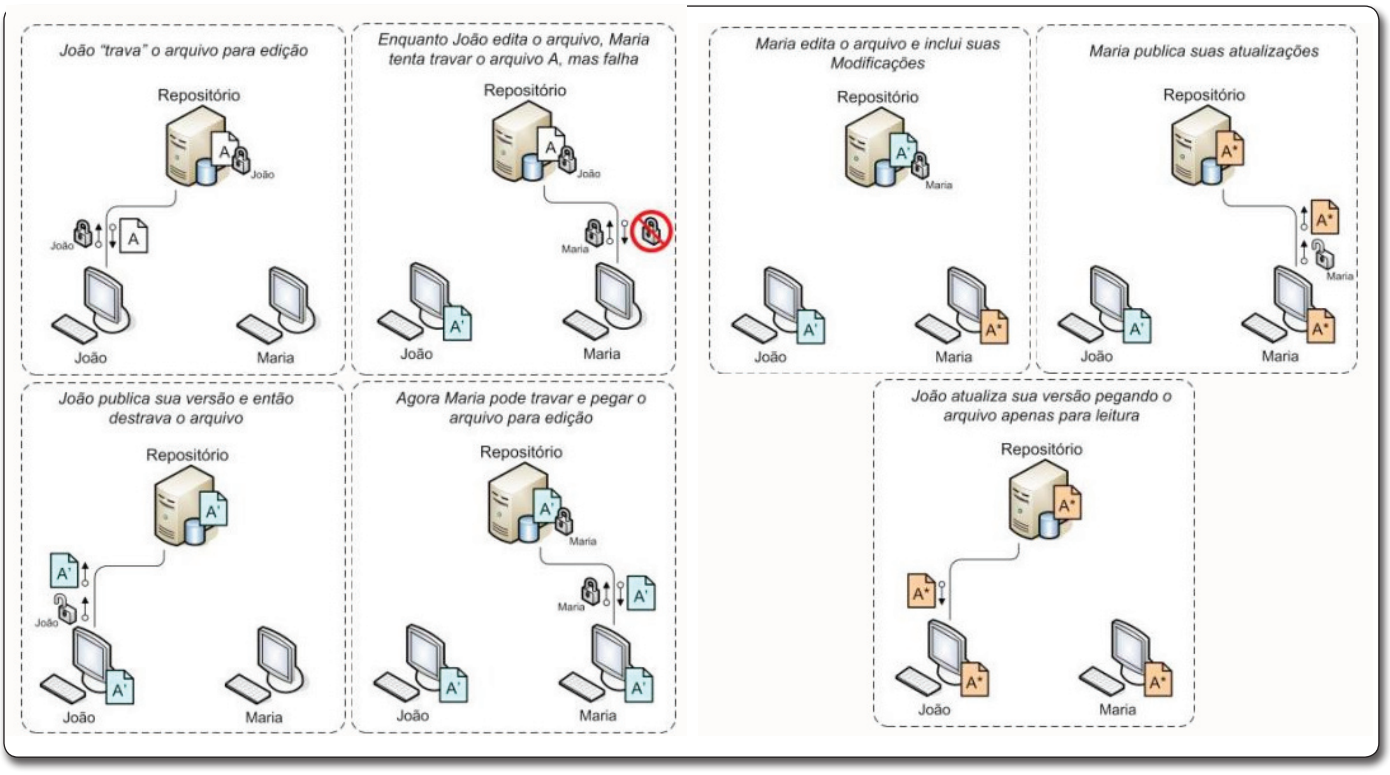


Figura 4. Política Pessimista

dos artefatos que estão sob controle da Gerência de Configuração de Software. Estes artefatos recebem o nome de itens de configuração. A cada operação de *check-in* realizada, a versão do item de configuração é incrementada de uma unidade. Quando o item é adicionado pela primeira vez no repositório, este item passa a ter a versão igual a 1. Para cada item de configuração armazenado, são anexadas informações como: datas da criação ou alteração, comentários e versões.

Neste cenário, não há perdas ou sobreposições porque políticas de trabalho foram estabelecidas, restringindo ou controlando as modificações no repositório. As ferramentas de controle de versões normalmente suportam a definição de diferentes políticas de trabalho. Dentre essas políticas, podemos citar a política pessimista, que enfatiza o uso de *check-out* reservado, fazendo bloqueio e inibindo o paralelismo do desenvolvimento sobre o mesmo artefato. O uso desta política pode ser exemplificado na **Figura 4**.

Outra política amplamente utilizada é a otimista. Neste cenário, se um artefato for alterado simultaneamente por dois desenvolvedores, a política assume que a quantidade de conflitos será naturalmente baixa e que será mais fácil tratar cada conflito individualmente, caso eles venham a ocorrer. A política otimista usa um mecanismo conhecido como junção (merge), que une as modificações efetuadas em paralelo sobre um mesmo artefato e produz uma nova versão deste artefato que contém a soma das modificações. Os conflitos ocorrem quando a mesma região ou linha do arquivo é modificada. A junção é automática, na maioria dos casos, mas quando ocorre um conflito, ela deve ser feita de forma manual. A **Figura 5** ilustra o processo.

Existem situações onde uma determinada política é mais indicada do que as demais. Nos casos onde a junção tende a ser complexa, quando, por exemplo, os artefatos não são textuais e a ferramenta não dá apoio automatizado para junções, é mais indicado trabalhar usando políticas pessimistas. Contudo, na grande maioria das situações referentes ao desenvolvimento de software, as políticas otimistas atendem satisfatoriamente.

Além desses recursos fundamentais presentes no sistema de controle de versão, outros recursos mais elaborados também são encontrados com freqüência. Em determinados momentos do ciclo de vida de desenvolvimento e manutenção do software, os itens de configuração são agrupados e verificados, constituindo configurações do software voltadas para propósitos específicos. Neste momento, cria-se marcos no versionamento de artefatos que são denominados *baselines* ou releases.

A diferença entre *baselines* e releases é sutil. As *baselines* representam conjuntos de itens de configuração formalmente aprovados que servem de base para as etapas seguintes de desenvolvimento. Mas, quando uma entrega formal é feita ao cliente, no final de uma iteração, por exemplo, denominamos esta entrega de release. *Baselines* e releases são identificadas no repositório, na grande maioria das vezes, pelo uso de etiquetas (tags). A **Figura 6** representa a *baseline* criada ao final da fase de codificação, considerando quatro itens de configuração. Cada item de configuração apresenta seu respectivo histórico de versões no repositório.

A Gerência de Configuração de Software também permite que a implementação de novas funcionalidades por uma equipe seja realizada em paralelo, mas de forma isolada e independente das modificações de outros desenvolvedores. O isolamento é obtido com o uso de ramo (*branch*). As linhas de desenvolvimento (*codelines*) são designadas para cada projeto e são compartilhadas por vários desenvolvedores. A primeira linha de desenvolvimento

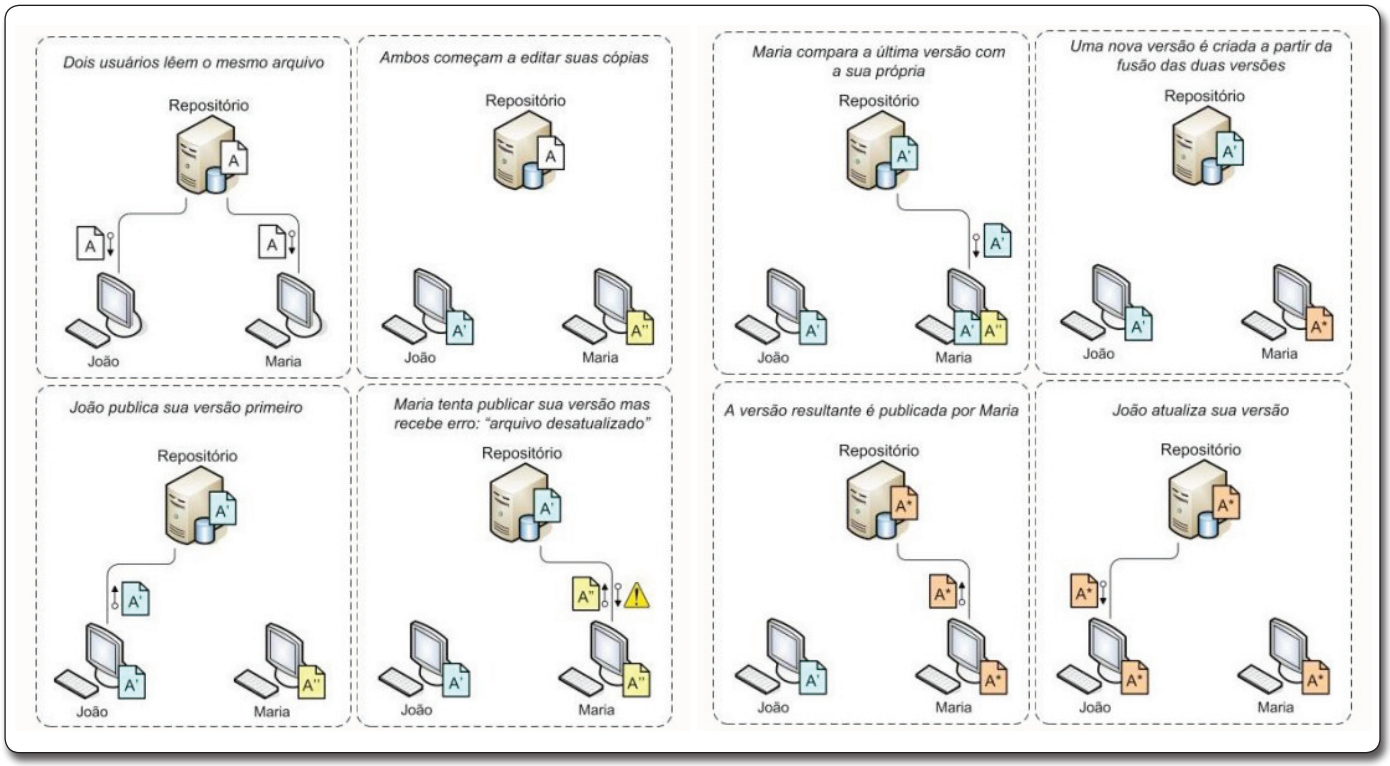


Figura 5. Política Otimista

definida no projeto é, por convenção, nomeada *mainline*. O ramo é criado no repositório e representa uma linha secundária de desenvolvimento que pode ser unida novamente à linha principal (*mainline*) por meio da operação de junção (*merge*). Atualmente, a necessidade de atender, ao mesmo tempo, as múltiplas demandas do projeto torna o uso de ramos um diferencial (ver **Figura 7**).

A **Figura 8** exemplifica a criação de um ramo e a operação de junção. A junção é efetuada para cada artefato do ramo e todas as modificações efetuadas desde o ancestral em comum são levadas em consideração. Imagine que um artefato foi modificado na *mainline* e no ramo. Digamos que, quando o ramo foi criado, este artefato estava na *mainline* com a versão 1. Mas quando ocorreu a junção do ramo, ele estava na *mainline* com a versão 2. O processo de junção soma as modificações efetuadas em paralelo, criando uma nova versão do artefato. Portanto, ao final, a versão 3 do artefato contém as modificações – efetuadas na *mainline* desde a versão 1 deste artefato e também as modificações – realizadas no ramo.

É importante ressaltar que o isolamento longo dificulta a operação de junção, portanto, é necessário realizar junções periodicamente.

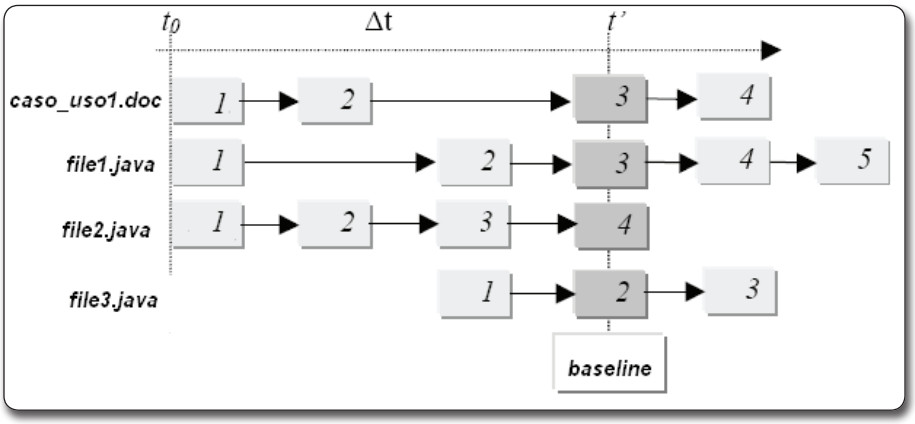


Figura 6. Definição de Baselines

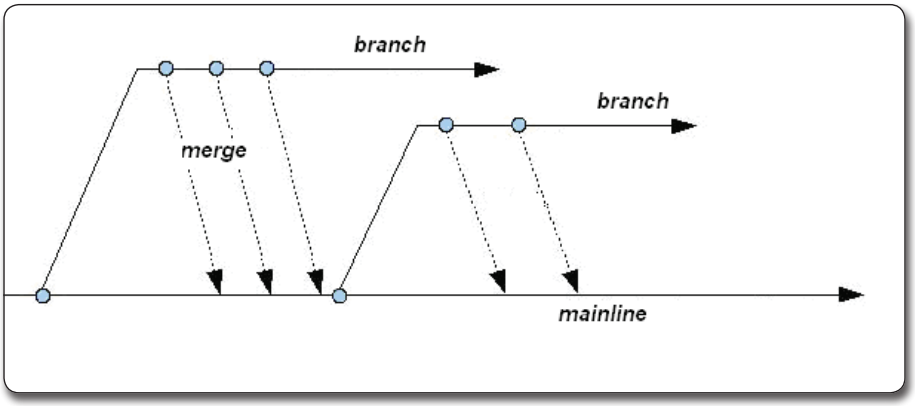


Figura 7. Definição de ramos

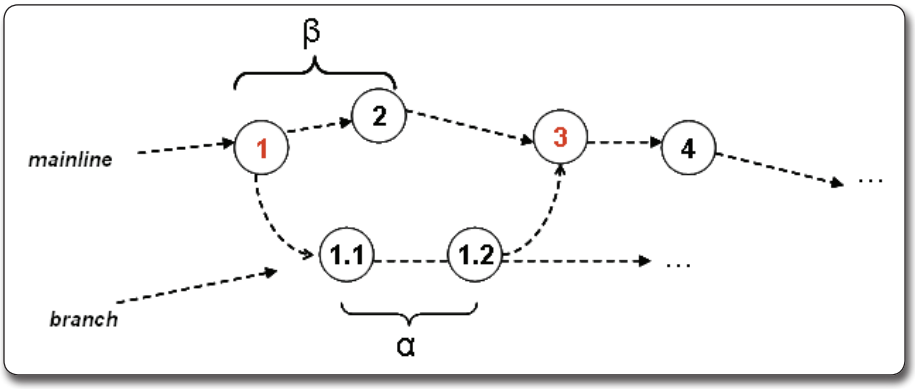


Figura 8. Junção

Estratégias de Organização

Estratégias podem ser adotadas para organizar o trabalho dos desenvolvedores no projeto. De forma genérica, criar um ramo é um meio de organizar o trabalho, pois isola o desenvolvimento de outras modificações e possibilita que o trabalho seja executado sem que uma modificação específica cause impacto nas demais alterações do software. No entanto, o isolamento que o ramo promove, tem um custo associado. Mesmo com ótimas ferramentas, a junção pode ser uma atividade difícil, em função dos conflitos que ocorrem. Portanto, deve existir um balanceamento entre custo versus benefício, avaliando duas alternativas: (1) usar isolamento com ramos e conciliar os eventuais conflitos que podem surgir ao fazer a junção deste ramo com a *mainline*; ou (2) não usar isolamento e fazer todas as modificações na *mainline*. Esta última alternativa pode minimizar os problemas de integração, mas maximizar a concorrência e dificultar a entrega de uma *release* com um subconjunto de requisitos completo e correto em um marco do projeto.

Estratégias de ramificação podem ser utilizadas, ao longo do desenvolvimento, em situações específicas. Algumas podem ser combinadas, outras são mutuamente exclusivas. Dentre as estratégias de organização, vamos citar neste artigo:

[Manutenção caótica] Estratégia onde não existe isolamento e, portanto, a impossibilidade de separar o que é manutenção corretiva da evolutiva (ver **Figura 9**). Na *mainline* ocorre a evolução e a correção do software. Não existem ramificações.

[Manutenção em série] Estratégia de organização que separa as evoluções das correções no software. Esta estratégia pode ser usada quando uma *release* do produto será entregue para a homologação, que é a fase onde testes serão executados e os erros corrigidos. Neste caso, na *mainline* só ocorre a evolução do produto, enquanto o ramo fica destinado às correções. O ramo é temporário e as junções acontecem no sentido do ramo para a *mainline*. Antes da junção, uma *release* com as correções é criada como mostra a **Figura 10**.

Para exemplificar o uso de ramos, vamos analisar um processo de desenvolvimento adotado nos projetos de uma organização e as funções da Gerência de Configuração de Software descritas na norma ISO 12207 e ver como se relacionam.

Gerência de Configuração e Desenvolvimento de Software

Por ser uma área fortemente calcada em controle, a Gerência de Configuração é referenciada em diversas normas, processos, procedimentos, políticas e padrões, como ISO 12207, CMMI e MPS.Br. Do ponto de vista gerencial, o processo de Gerência de Configuração de Software é dividido em cinco funções (ver **Nota 1**): identificação da configuração, controle da configuração, acompanhamento da situação da configuração, auditoria da configuração e gerenciamento de entrega.

Para auxiliar e garantir a execução das atividades relacionadas com as funções da Gerência de Configuração de Software, uma organização pode definir uma equipe de Gerência de Configuração, normalmente única no contexto organizacional.

A Gerência de Configuração é um processo auxiliar de controle e acompanhamento das atividades do desenvolvimento de software (ler **Nota 2**). Para exemplificar, vamos considerar um ciclo de vida iterativo e incremental que propõe inicialmente a identificação do escopo do projeto e a aprovação deste escopo pelo cliente. Posteriormente, nesta abordagem, o software é construído em ciclos sucessivos denominados iterações. A cada iteração, os requisitos são priorizados, detalhados, aprovados e o software é modelado, construído e testado, como ilustra a **Figura 11**. Ao final, uma *release* do produto é entregue para homologação e aprovação do cliente.

Na atividade homologar é importante verificar a release em um ambiente isolado do desenvolvimento, realizando testes funcionais. Neste momento, à medida que erros são identificados no teste, solicitações de modificações são registradas na atividade modificar liberação. A **Figura 12** ilustra a fase de homologação.

Para que itens de configuração de uma *baseline* ou *release* possam evoluir de forma controlada, a função de controle da configuração estabelece as atividades: (1)

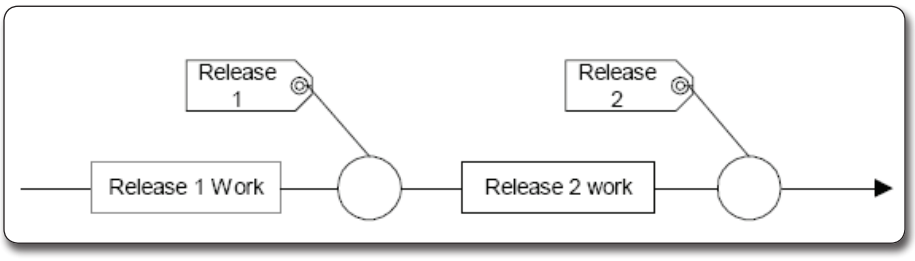


Figura 9. Manutenção caótica

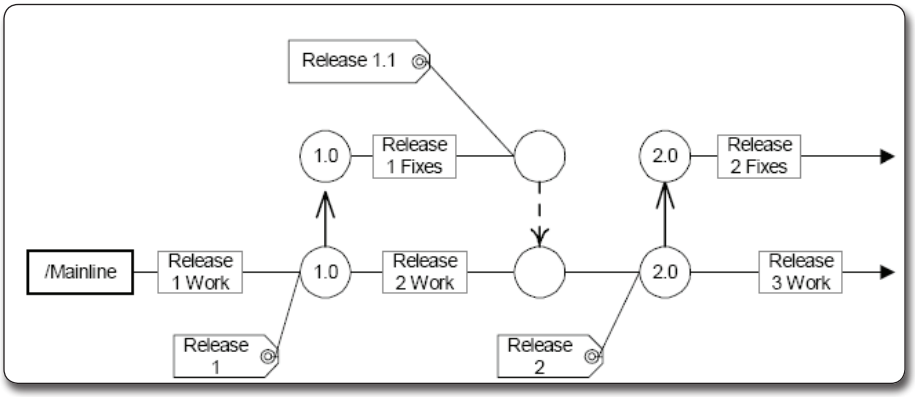


Figura 10. Manutenção em série

Nota 1. Funções da Gerência de Configuração

A função de identificação da configuração tem por objetivo: (1) a seleção de quais artefatos serão itens de configuração; (2) a definição de uma nomenclatura, que possibilite a identificação inequívoca dos itens de configuração, *baselines* e *releases*; e (3) a descrição dos itens, tanto física quanto funcionalmente.

A seleção de itens de configuração é feita no início da fase de planejamento e leva em conta: (1) se o artefato é crítico para o projeto; (2) a dependência entre artefatos; (3) o impacto que uma modificação do item tem no produto; (4) se o artefato pode ser modificado por dois ou mais grupos; (5) se é freqüentemente alterado devido a sua complexidade e (6) se é gerado manualmente, automaticamente ou ambos.

A função de controle da configuração é designada para controlar e acompanhar a evolução dos itens de configuração selecionados na função de identificação. Ferramentas como JIRA, Bugzilla, dentre outras, apóiam, em conjunto com as ferramentas de controle de versões, as atividades desta função.

A função de acompanhamento da situação da configuração visa: (1) armazenar as informações geradas pelas demais funções; e (2) permitir que essas informações possam ser acessadas em função de necessidades específicas, por exemplo, para a melhoria do processo, para a estimativa de custos futuros e para a geração de relatórios gerenciais. Estas informações podem ser obtidas, no decorrer do projeto, a partir dos sistemas de controle de versões e modificações.

A função de auditoria da configuração ocorre geralmente quando uma *release* deve ser criada. Suas atividades compreendem: (1) auditoria funcional, que abrange a revisão dos planos, dados, metodologia e resultados dos testes, assegurando que a *release* cumpre corretamente o que foi especificado; e (2) auditoria física, com o objetivo de certificar que a *release* é completa em relação ao que foi acertado em cláusulas contratuais. A auditoria pode ser feita com base nos relatórios obtidos na função anterior.

Já a função de gerenciamento de liberação e entrega descreve o processo formal de: (1) construção e liberação de uma *release* do produto; e (2) entrega, com informações de como implantar o software no ambiente final de execução. Ferramentas, como Ant, permitem que roteiros de construção sejam escritos e executados no apoio a esta fase.

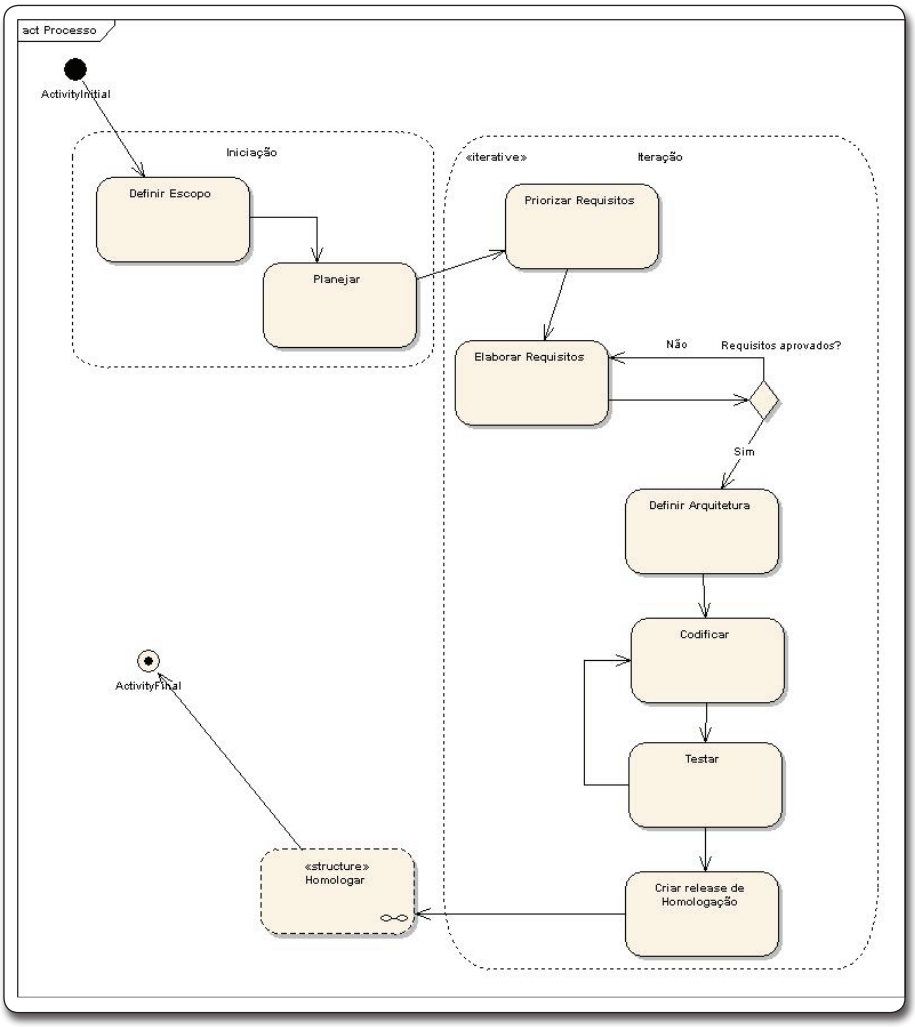


Figura 11. Processo de Desenvolvimento

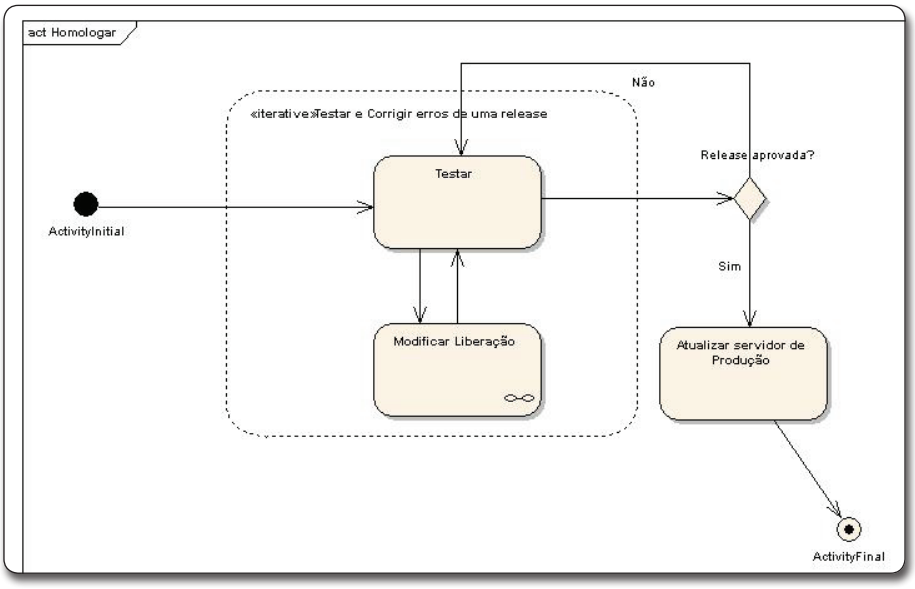


Figura 12. Fase de Homologação

solicitação de modificação, que pode ser corretiva, evolutiva, adaptativa ou preventiva; (2) classificação da modificação, que estabelece a prioridade da solicitação em relação às demais solicitações efetuadas anteriormente; (3) análise de impacto, que visa relatar os impactos em esforço, cronograma e custo; (4) avaliação da modificação, que estabelece se a modificação será implementada, rejeitada ou postergada, em função do laudo da análise de impacto da modificação; (5) implementação da modificação, caso a solicitação tenha sido aprovada pela avaliação da modificação; (6) verificação da modificação com relação à proposta de implementação levantada na análise de impacto; (7) gerência da *baseline* e *release*. Uma nova *baseline* ou *release* pode ser criada ao final, agrupando uma ou mais solicitações resolvidas. A **Figura 13** ilustra esta função que está relacionada com a atividade modificar liberação executada na fase de homologação do produto.

Analisando as atividades da **Figura 13**, vemos que para implementar e verificar a modificação podemos criar um ramo separado. Portanto, podemos criar um ramo para corrigir os erros encontrados, adotando a estratégia de organização em série, como vimos na seção Estratégias de Organização. Desta forma, a implementação de novas funcionalidades do produto pode continuar na *mainline*, considerando as próximas iterações do desenvolvimen-

Nota 2. Uso do processo de gerência de configuração

No caso de artefatos ainda em desenvolvimento, recomenda-se um processo mais *ad-hoc*, que não faça com que a dinâmica do desenvolvimento seja perdida. Neste caso, apenas as operações *check-out* e *check-in* são necessárias. Mas, recomenda-se que um resumo do que foi implementado seja exposto no comentário a cada operação de *check-in* realizada, juntamente com o item de configuração e sua respectiva versão no repositório.

No caso de *baselines* ou *releases*, isto é, versões estáveis do produto, é ideal ser mais formal, pois qualquer mudança nos itens de configuração pode causar impacto em custo e prazo de entrega do produto. O formalismo aplicado, no entanto, pode variar em função da flexibilidade que determinados processos de desenvolvimento de software necessitam.

to, enquanto a correção dos erros é feita em um ramo à parte.

Neste cenário, os erros são corrigidos no ambiente de desenvolvimento, e após todas as verificações, uma *release* é criada. Ao final, atualiza-se o ambiente de homologação com a nova *release*. Quando a *release* for aprovada pelo cliente, uma versão do produto pode ser posta em produção. A **Figura 14** detalha este processo.

A atividade gerenciar ramos na **Figura 14** mostra como proceder a implementação e verificação das correções dos erros encontrados na fase de homologação. Esta atividade apresenta os seguintes passos: (1) criar ramo no repositório, (2) controlar itens de configuração no ramo durante a atividade de implementação e teste, (3) verificar se os erros foram corrigidos corretamente, (4) criar *release* com um conjunto de solicitações atendidas, (5) integrar na *mainline* as correções feitas no ramo em função das solicitações.

Neste contexto, controlar itens de configuração significa: efetuar *check-out* atualizando o espaço de trabalho com os itens contidos no ramo criado no repositório; alterar os itens e efetuar *check-in*, ao termi-

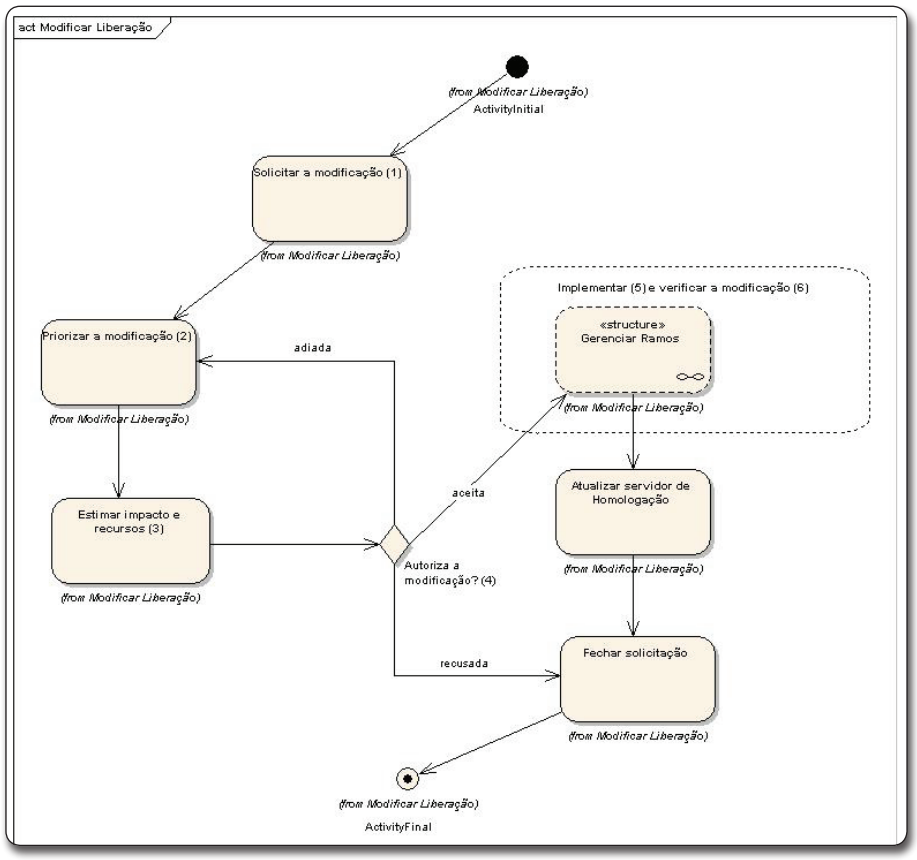


Figura 13. Modificar Liberação

nar a modificação, devolvendo os itens do espaço de trabalho para o repositório.

Conclusão

Vimos no artigo que a Gerência de Configuração pode ser tratada sob diferentes perspectivas em função do papel exercido pelo participante do processo de desenvolvimento. Na perspectiva gerencial, a Gerência de Configuração é dividida em cinco funções, que são: identificação da configuração, controle da configuração, contabilização da situação da configuração, auditoria da configuração e gerenciamento de liberação e entrega. Sob a perspectiva de desenvolvimento, a Gerência de Configuração é dividida em três sistemas principais: controle de modificações, controle de versões e gerenciamento de construção. As cinco funções podem ser implementadas pelos três sistemas descritos na perspectiva de desenvolvimento, acrescidos de procedimentos manuais quando necessário.

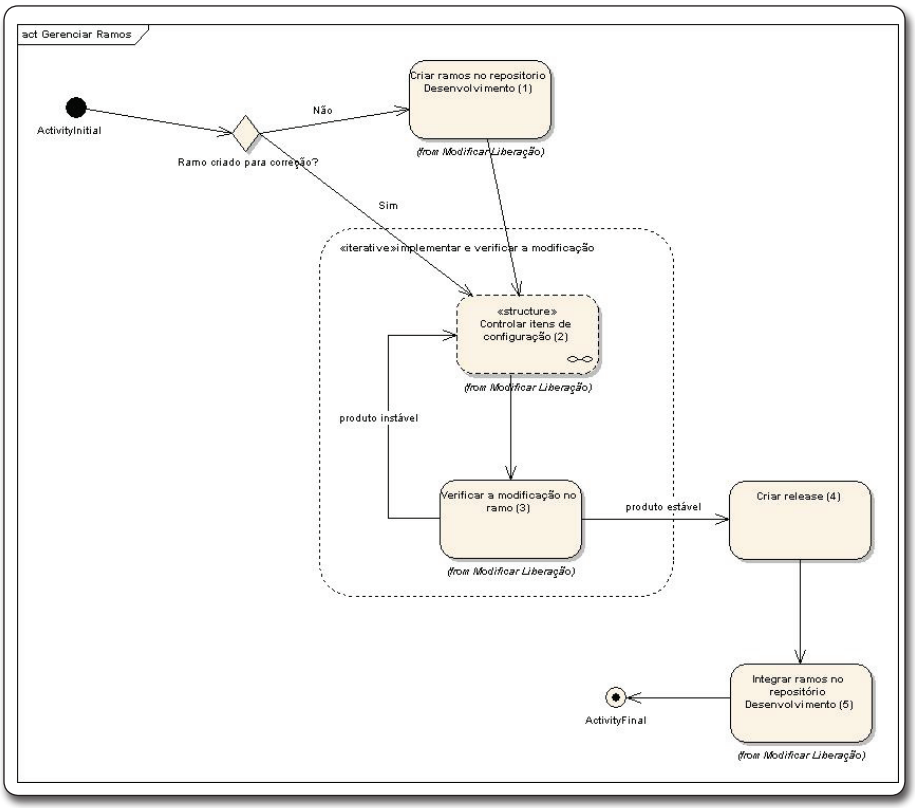


Figura 14. Gerenciar ramos

Para exemplificar como a Gerência de Configuração apóia o desenvolvimento, vimos em detalhes a função de controle de configuração na fase de homologação de um produto.

Este artigo também apresentou uma breve introdução sobre Gerência de Configuração, ressaltando os benefícios que esta área provê, além de conceitos e estratégias que podem ser utilizadas para desenvolver software de forma produtiva, eficiente e disciplinada com uso de ferramentas.

Agradecimentos

Este artigo contou com o apoio do CNPq, através do processo no 552220/2006-0, intitulado "Ferramentas Inovadoras de Engenharia de Software para a Melhoria da Qualidade em Processos". Edital MCT/CNPq nº 03/2006 - RHAE Inovação. ●

Referências

- "Software Configuration Management Handbook", Alexis Leon (2004)
- "Software Configuration Management Patterns:Effective Teamwork, Practical Integration", Stephen Berczuk e Brad Appleton (2002)
- SOFTEX, 2006b, MPS.BR - Melhoria de Processo do Software Brasileiro - Guia de Implementação (Versão 1.1), Associação para Promoção da Excelência do Software Brasileiro.
- ISO, 1995b, ISO/IEC 12207 - Information technology - Software life cycle processes, International Organization for Standardization.

Análise de Pontos de Função

Uma aplicação nas estimativas de tamanho de Projetos de Software

Claudia Hazan
claudinhah@yahoo.com
Graduada em Informática pela Universidade do Estado do Rio de Janeiro (UERJ), Mestre em Engenharia de Sistemas e Computação pelo Instituto Militar de Engenharia (IME), Em Doutoramento em Engenharia de Software pela PUC-Rio, atua como consultora de métricas e qualidade de software no Serviço Federal de Processamento de Dados (SERPRO), ministrando treinamento de Análise de Pontos de Função com estimativas e apoiando a empresa e seus clientes na contagem de Pontos de Função, nas estimativas de projetos de software e gestão de contratos baseados em Pontos de Função.

A indústria de software continua sentindo os efeitos da crise do software da década 80. Isto pode ser observado quando analisamos os três principais sintomas da crise do software apresentados por Pressman em 2006, a saber:

- A produtividade não tem acompanhado a demanda por serviços;
- A qualidade do software, em alguns sistemas, não é adequada;
- As estimativas de prazo e custo são freqüentemente imprecisas.

Este último sintoma, que está associado a um dos principais problemas que a indústria de software tem enfrentado (a falta de previsibilidade de custo e prazo de projetos de software), pode levar a conseqüências desastrosas, tais como: conflitos entre o gerente do projeto e a equipe, baixa estima da equipe, entrega de software de baixa qualidade, perda de imagem da organização, e até mesmo o cancelamento do projeto. Assim, torna-se importante o investimento na implantação de um processo de estimativas efetivo, visando a melhoria da previsibilidade de custo, prazo e esforço.

Este artigo tem como propósito apresentar um processo de estimativas de projetos de software, aderente ao modelo CMMI, utilizando a métrica de Pontos de Função para estimar o tamanho funcional do projeto de software.

Processo de Estimativas de Projetos de Software

Esta seção apresenta uma visão geral de um processo de estimativas de projetos de software aderente à área de processo de Planejamento do Projeto do nível 2 do CMMI.

A **Figura 1** ilustra um processo de Estimativas de Projetos de Software. Este processo é descrito nos parágrafos seguintes.

O principal insumo (artefato de entrada) para um processo de estimativas é o documento de requisitos. Como as

estimativas devem ser realizadas no início do processo de desenvolvimento de software, então, o artefato utilizado é freqüentemente um documento inicial de requisitos (por exemplo: Documento de Visão) ou até mesmo um documento do próprio cliente (por exemplo: modelo de processo de negócios ou manual do usuário, no caso de redesenvolvimento). O responsável pelas estimativas deve analisar os requisitos para garantir a qualidade e então estimar o tamanho do projeto de software. O próximo passo é a derivação das estimativas de esforço, prazo (cronograma), custo (orçamento) com base na estimativa de tamanho e nos dados históricos de projetos concluídos da organização, assim como o estabelecimento da estimativa de recursos computacionais críticos. Neste ponto, as principais estimativas foram geradas, no entanto estas precisam ser documentadas. As premissas e suposições utilizadas na geração das estimativas também devem ser documentadas.

A análise das estimativas por um estimador independente, um profissional que não atue na equipe do projeto, constitui uma boa prática. O estimador independente poderá analisar a consistência das estimativas e a qualidade das suas documentações. No decorrer do processo de desenvolvimento, as estimativas devem ser acompanhadas conforme o refinamento dos requisitos. O projeto deve ser re-estimado se ocorrerem mudanças significativas nos requisitos funcionais ou não funcionais. Quando o projeto é concluído, deve-se documentar o tamanho, prazo, custo, esforço e recursos realizados, assim como outros atributos relevantes do projeto, visando a coleta de dados para a melhoria do processo de estimativas. As lições aprendidas também devem ser documentadas.

O foco deste artigo está na geração de estimativas de tamanho de projetos de software. Observe que a estimativa de tamanho é a primeira a ser gerada e a partir dela são derivadas as demais estimativas. Por isso, a importância desta estimativa. A literatura apresenta várias métricas de tamanho. Na indústria brasileira, pode-se destacar a utilização de três métricas: **LOC (Line of Code)**, **Pontos por Casos de Uso** e **Pontos de Função**. A seção seguinte apresenta as vantagens e desvantagens da utilização destas métricas.

Métricas de Tamanho de Projetos de Software

A métrica **Linhas de Código (LOC)** é de fato a mais antiga. A principal vantagem é que a contagem de linhas de código pode ser automatizada por uma ferramenta. As desvantagens são as seguintes: muitas vezes, o significado de uma linha de código é subjetivo, por exemplo, contar ou não linhas de comentário no código fonte; a métrica não é adequada para ser um indicador de produtividade, por exemplo, o desenvolvedor que escrever mais linhas de código será mais produtivo do que o desenvolvedor que escrever um algoritmo mais elegante e mais eficiente com menos linhas de código. Além disso, torna-se bastante complicado e subjetivo aplicar esta métrica em um processo de estimativas cujo insumo é um documento inicial de requisitos. Portanto, não é recomendado o uso da métrica LOC como unidade de medida para as estimativas de tamanho.

A métrica **Pontos por Casos de Uso (PCU)** foi proposta por Gustav Karner com o propósito de estimar recursos para projetos de software orientados a objeto, modelados por meio de especificação de Casos de Uso. A métrica é de fácil aplicação, não requer muito tempo de treinamento ou experiência prática. No entanto, o PCU somente pode ser aplicado em projetos de software cuja especificação tenha sido expressa em casos de uso. Além disso, como não existe um padrão único para a escrita de uma especificação de caso de uso, diferentes estilos na escrita do caso de uso ou na sua granularidade podem levar a resultados diferentes na medição por PCU. Assim, a métrica se torna subjetiva. E ainda, devido ao processo de medição do PCU ser baseado em casos de uso, o método não pode ser empregado antes de concluída a análise de requisitos do projeto. Na maioria das vezes existe a necessidade de se obter uma estimativa antes da finalização desta etapa. Então, esta métrica não é recomendada para utilização como unidade de medida das estimativas de tamanho de projetos de software.

A métrica **Pontos de Função (PF)**, definida por Allan Albrecht em 1979, tem sido utilizada de forma crescente pela indústria de software. O IFPUG (*International*

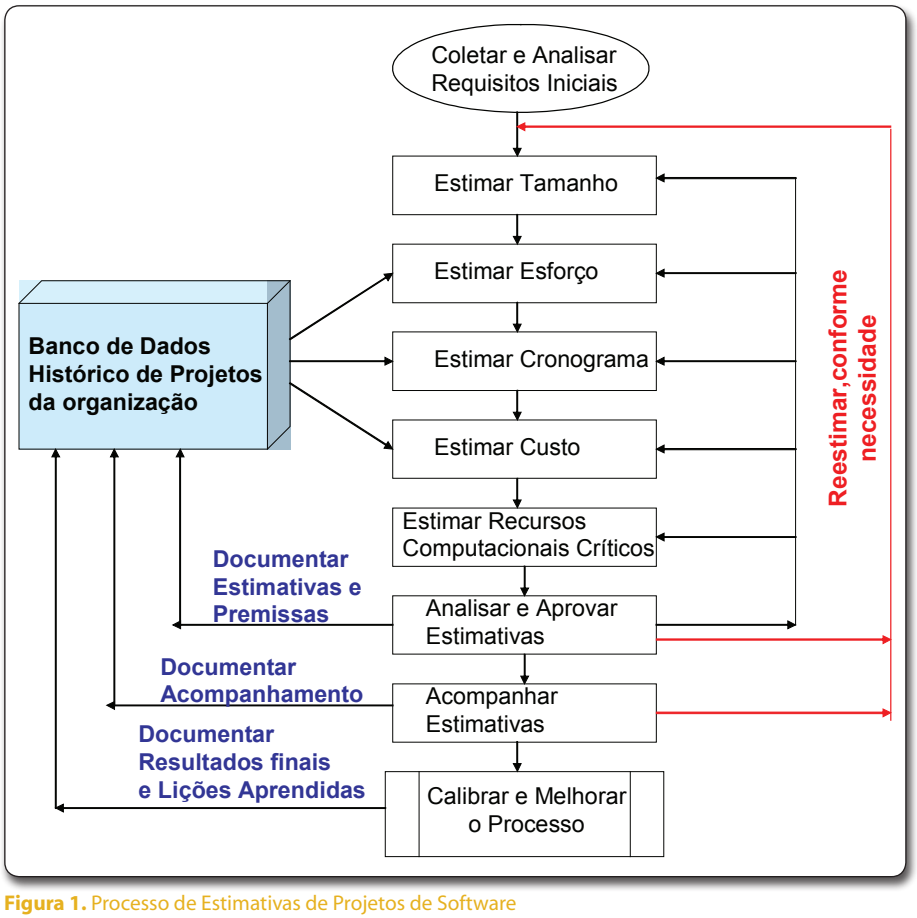


Figura 1. Processo de Estimativas de Projetos de Software

Function Point Users Group), criado em 1986, é responsável pela atualização das regras de Contagem de Pontos de Função, descritas no CPM (*Counting Practices Manual*), que se encontra na versão 4.2.1, publicada em 2005 no IFPUG. O IFPUG também é responsável pelo exame de certificação de especialistas em contagem de Pontos de Função, denominada CFPS (*Certified Function Point Specialist*). A métrica Pontos de Função é uma medida de **tamanho funcional** de projetos de software, considerando as funcionalidades implementadas, sob o ponto de vista do usuário. Tamanho funcional é definido como "tamanho do software derivado pela quantificação dos requisitos funcionais do usuário" (Dekkers, 2003).

A contagem de Pontos de Função é independentemente da metodologia de desenvolvimento e da plataforma utilizadas no desenvolvimento da aplicação. Assim, a autora recomenda a utilização da métrica Pontos de Função nas estimativas de tamanho de projetos de software. A autora tem utilizado a métrica para estimar os projetos dos clientes da organização Serviço Federal de Processamento de Dados (SERPRO) desde 2003, tendo obtido excelentes resultados.

A seção seguinte apresenta o método Contagem Estimativa de Pontos de Função (CEPF), utilizado para estimar o tamanho de projetos de software.

Contagem Estimativa de Pontos de Função

Antes de definir o método de estimativas CEPF, é importante destacar que "estimar significa utilizar o mínimo de tempo e esforço para se obter um valor aproximado dos Pontos de Função do projeto de software investigado" (Meli, 1999). Assim, para evitar confusão, é recomendável sempre fazer uma distinção entre os termos e conceitos: Contagem de Pontos de Função e Estimativa de Pontos de Função.

- **Contagem de Pontos de Função:** significa medir o tamanho do software por meio do uso das regras de contagem do IFPUG (IFPUG, 2005);
- **Estimativa de Pontos de Função:** significa fornecer uma avaliação aproximada do tamanho de um software utilizando métodos diferentes da Contagem de Pontos de Função do IFPUG.

Além do método Contagem indicativa de Pontos de Função, existem diversos métodos para estimar tamanho de projetos em Pontos de Função, descritos na literatura, dentre outros: Contagem Indicativa, Contagem Indicativa Inteligente, Estimativas Percentuais.

O método CEPF visa aferir o tamanho em PF de maneira simplificada, com base no conhecimento dos requisitos iniciais do projeto. Inicialmente, os requisitos funcionais iniciais documentados nas propostas comerciais, nos documentos de visão, ou em qualquer especificação inicial do sistema do usuário são mapeados nos tipos funcionais da Análise de Pontos de Função (APF): Arquivo Lógico Interno (ALI), Arquivo de Interface Externa (AIE), Entrada Externa (EE), Consulta Externa (CE) e Saída Externa (SE). Posteriormente, os Pontos de Função são associados a cada função identificada, baseando-se nas tabelas de complexidade e de contribuição funcional do CPM.

O estimador deve realizar uma leitura no documento inicial de requisitos, buscando informações relevantes para a identificação de processos elementares. O processo elementar é definido como a menor unidade de atividade significativa para o usuário (IFPUG, 2005). O processo elementar deve ser completo em si mesmo, independente e deixar a aplicação em um estado consistente (IFPUG, 2005). Uma vez identificado o processo elementar, o estimador deve buscar o entendimento deste para classificá-lo em Entrada Externa, Consulta Externa ou Saída Externa. Adicionalmente, o estimador deve descobrir os dados associados ao processo elementar, visando a determinação da complexidade funcional da função identificada. Caso não seja possível a identificação da complexidade da funcionalidade em questão, recomenda-se a utilização da complexidade Média. Na análise do processo elementar também são identificados, os grupos de dados lógicos da aplicação, que são classificados como Arquivos Lógicos Internos ou Arquivos de Interface Externa. Caso não seja possível a identificação da complexidade da função de dados em questão, recomenda-se a utilização da complexidade Simples.

A seguir são apresentadas dicas para ajudar no mapeamento dos requisitos funcionais da aplicação nos tipos funcionais da APF.

As necessidades e funcionalidades especificadas para o projeto, contidas no documento inicial de requisitos, devem ser enquadradas em uma das seguintes tabelas:

Tabela 1 - Contagem dos Arquivos Lógicos Internos (ALIs): Banco de Dados Lógico da Aplicação (tabelas e arquivos mantidos pela aplicação).

Considerações: Identifique os grupos de dados lógicos de aplicação nos modelos de dados ou diagrama de classes ou a partir dos requisitos funcionais, descritos nos documentos de requisitos (Documento de Visão, Relação de Casos de Uso, etc.). Não considere arquivos físicos, arquivos de índices, arquivos de trabalho e tabelas de relacionamento sem atributos próprios (tabelas que existem para quebrar o relacionamento nxn e apenas transportam as chaves estrangeiras). As entidades fracas também não são consideradas um ALI.

Se possível, tente descobrir os atributos lógicos, campos reconhecidos pelo usuário, e subgrupos de dados existentes para obter a complexidade funcional, segundo as regras de contagem do CPM (IFPUG, 2005). Caso não seja possível, a experiência tem mostrado que a maioria dos ALIs dos sistemas são de complexidade **Simples.**

Nº ALIs Simples:	X 7 PF
Nº ALIs Médio:	X 10 PF
Nº ALIs Complexo:	X 15 PF
Total PF da Tabela 1:	

Tabela 1. Identificação dos Arquivos Lógicos Internos da Aplicação

Tabela 2 - Contagem de Arquivos de Interface Externa (AIEs): Banco de Dados de outras Aplicações, **apenas referenciados** pela aplicação que está sendo estimada (tabelas e arquivos mantidos por outra aplicação).

Considerações: Identifique os grupos de dados lógicos de outras aplicações referenciados pela aplicação que está sendo estimada. Freqüentemente, o referenciamento de dados ocorre para a validação de informações em cadastros ou consultas. Algumas vezes, relatórios ou consultas referenciam dados externos de outras aplicações, também considerados AIEs. Não são considerados arquivos físicos, arquivos de índice, arquivos de

trabalho, tabelas de relacionamento sem atributos próprios e entidades fracas.

A experiência tem mostrado que praticamente 100% dos AIEs dos sistemas são **Simples**. Porque, segundo o CPM (IFPUG, 2005), são considerados para a determinação da complexidade funcional do AIE apenas os atributos referenciados pela aplicação que está sendo contada.

Nº AIEs Simples:	X 5 PF
Nº AIEs Médio:	X 7PF
Nº AIEs Complexo:	X 10 PF
Total PF da Tabela 2:	

Tabela 2. Identificação dos Arquivos de Interface Externa da Aplicação

Tabela 3 - Contagem de Entradas Externas (EEs): Funcionalidades que mantêm os Arquivos Lógicos Internos (ALIs) ou alteram o comportamento da aplicação.

Considerações: Identifique as funcionalidades de inclusão, alteração e exclusões de dados. Conte separadamente as inclusões, alterações e exclusões de dados, isto é, cada função independente de inclusão ou alteração ou exclusão deve ser contada separadamente. A aplicação possui funções de entrada de dados que alteram o comportamento dela, por exemplo: processamentos *batch*, ou processamento de informações de controle? Caso positivo, estas funções também devem ser identificadas como Entradas Externas.

Se você não possui conhecimento da aplicação de APF ou sobre o processo elementar (funcionalidade analisada), considere as Entradas Externas identificadas com complexidade **Média**.

Nº EEs Simples:	X 3 PF
Nº EEs Média:	X 4 PF
Nº EEs Complexa:	X 6 PF
Total PF da Tabela 3:	

Tabela 3. Identificação das Entradas Externas da Aplicação

Tabela 4 - Contagem de Consultas Externas (CEs): funcionalidades que apresentam informações para o usuário **sem** a utilização de cálculos ou algoritmos. São os processos elementares do tipo "lê - imprime", "lê - apresenta dados", incluindo consultas, relatórios, geração de disquetes ou CDs, *downloads*, entre outros.

Considerações: Você está desenvolvendo uma função para apresentar informações para o usuário: uma consulta, relatório, *browse*, *listbox*, *download*, geração de um arquivo, geração de CD ou de disquete? Esta função **não** possui cálculos ou algoritmos para derivação dos dados referenciados nem altera um Arquivo Lógico Interno, nem muda o comportamento do sistema? Caso positivo, estas funções devem ser identificadas como Consultas Externas.

Caso não haja conhecimento da aplicação de APF ou sobre o processo elementar (funcionalidade analisada), considere as Consultas Externas com complexidade **Média**.

Nº CEs Simples:	X 3 PF
Nº CEs Média:	X 4 PF
Nº CEs Complexa:	X 6 PF
Total PF da Tabela 4:	

Tabela 4. Identificação das Consultas Externas da Aplicação

Tabela 5 - Contagem de Saídas Externas (SEs): Funcionalidades que apresentam informações para o usuário **com** utilização de cálculos ou algoritmos para derivação de dados ou atualização de Arquivos Lógicos Internos ou mudança de comportamento da aplicação. São as consultas ou relatórios com totalização de dados, relatórios estatísticos, gráficos, geração de disquetes com atualização *log*, *downloads* com cálculo de percentual, entre outros.

Considerações: Você está desenvolvendo uma funcionalidade para apresentar informações para o usuário: uma consulta ou relatório com totalização de dados, etiquetas de código de barras, gráficos, relatórios estatísticos, *download* com percentual calculado, geração de arquivo com atualização de *log*? Caso positivo, estas funções devem ser identificadas como Saídas Externas. Observe que esta função *deve* ter cálculos ou algoritmos para processar os dados referenciados nos arquivos lógicos *ou* atualizar campos (normalmente indicadores) nos arquivos *ou* mudar o comportamento da aplicação.

Caso não haja conhecimento da aplicação de APF ou sobre o processo elementar (funcionalidade analisada), considere as Saídas Externas com complexidade **Média**.

Nº SEs Simples:	X 4 PF
Nº SEs Média:	X 5 PF
Nº SEs Complexa:	X 7 PF
Total PF da Tabela 5:	

Tabela 5. Identificação dos Saídas Externas da Aplicação

A Estimativa de tamanho do projeto em PFs deve ser gerada totalizando-se os PFs obtidos nas **Tabelas 1**, **2**, **3**, **4**, e **5.**

Uma aplicação da Contagem Estimativa de Pontos de Função

Esta seção tem como propósito apresentar um exemplo utilizando o método CEPF, bem como a derivação das estimativas de esforço e prazo, a partir da estimativa de tamanho do projeto em PF. No exemplo apresentado, é descrito um sistema hipotético, parte de um sistema real, visando apresentar uma visão geral de um procedimento de estimativas de maneira prática.

Suponha que o setor de treinamento de uma empresa solicitou um sistema, denominado STREINA, para apoiar as atividades de planejamento e acompanhamento das atividades de capacitação dos funcionários. A **Tabela 6** apresenta

as necessidades e funcionalidades do STREINA, retiradas do Documento de Visão do projeto. Além das funcionalidades apresentadas na **Tabela 6**, deve-se considerar os requisitos de cadastro de usuários e controle de acesso da aplicação. Estas funcionalidades estarão disponíveis para o perfil administrador do sistema.

Para facilitar o entendimento da aplicação da CEPF, a **Tabela 7** mostra a contribuição para a contagem de PFs não ajustados dos tipos funcionais da APF. A complexidade funcional das funcionalidades identificadas é inferida em uma contagem estimativa, quando não se possui informação suficiente do projeto para aplicar as regras de contagem do CPM. Conforme mencionado, recomenda-se utilizar a complexidade Simples para os Arquivos Lógicos (ALI e AIE) e a complexidade Média para as Funções Transacionais (EE, CE, SE).

Aplicando-se o método Contagem Estimativa de Pontos de Função, tem-se o resultado apresentado na **Tabela 8**.

Totalizando o tamanho em PFs das funcionalidades descritas na **Tabela 8**, tem-se 170 PFs não ajustados. Suponha que o fator de ajuste da contagem seja de 1,10. O fator de ajuste da contagem de PF é determinado com base na avaliação das 14 Características Gerais dos Sistemas, que descrevem as funcionalidades gerais das aplicações, por exemplo: performance, reuso, usabilidade, etc. O manual de práticas de contagem (CPM) apresenta a descrição das características e de suas ponderações, denominadas níveis de influência. O cálculo do PF Ajustado é obtido multiplicando-se os PFs Não Ajustados pelo Fator de Ajuste. Assim, temos 187 Pontos de Função Ajustados estimados.

De posse da estimativa de tamanho, procede-se com a geração da estimativa de esforço. Neste exemplo, utiliza-se o modelo simplificado de estimativas de esforço, descrito por Hazan em 2005. Para isto, deve-se obter um índice de produtividade por meio de análise do banco de dados de histórico de projetos da organização, observando-se os atributos do projeto em questão e o esforço realizado em projetos similares. Este projeto é pequeno (menos de 200 PFs) e será desenvolvido em JAVA por uma equipe com experiência intermediária na plata-

Necessidade 1		**Benefício**
Controlar capacitações		**Crítico**
Id Func.	Descrição das Funcionalidades/atores envolvidos	
F 1.1	Cadastrar evento de capacitação	
	Técnico	
F 1.2	Planejar evento de capacitação	
	Técnico	
F 1.3	Planejar cronograma de capacitação	
	Técnico	
F 1.4	Consultar cronograma de capacitação	
	Técnico	
F 1.5	Consultar eventos de capacitação por data e local	
	Técnico	
F 1.6	Consultar detalhes de um evento de capacitação	
	Técnico	
F 1.7	Informar resultado da avaliação de um evento de capacitação	
	Técnico	
F 1.8	Emitir Certificado de Participantes	
	Técnico	
F 1.9	Cadastrar avaliação do evento de capacitação (após treinamento)	
	Participantes do treinamento	
F 1.10	Consultar acompanhamento de comunidade capacitada	
	Técnico	
F 1.11	Gerar Consultas Gerenciais (Gráficos e Relatórios) de Capacitação **Gerente**	

Tabela 6. Funcionalidades do Sistema de Treinamentos (STREINA)

Descrição do	Complexidade		
Tipo Funcional	Simples	Média	Complexa
Arquivo Lógico Interno (ALI)	7 PFs	10 PFs	15 PFs
Arquivo de Interface Externa (AIE)	5 PFs	7 PFs	10 PFs
Entrada Externa (EE)	3 PFs	4 PFs	6 PFs
Saída Externa (SE)	4 PFs	5 PFs	7 PFs
Consulta Externa (CE)	3 PFs	4 PFs	**6 PFs**

Tabela 7. Contribuição para Contagem de PF dos Tipos Funcionais da APF

Descrição da Função	Tipo Funcional	Complexidade	Tamanho (PF)
Usuários	ALI	Simples	7 PF
Incluir usuário	EE	Simples	3 PF
Alterar usuário	EE	Simples	3 PF
Excluir usuário	EE	Simples	3 PF
Lista de usuários	CE	Simples	3 PF
Consultar usuários	CE	Simples	3 PF
Controle de acesso da aplicação	SE	Simples	4 PF
Alterar senha	EE	Simples	3 PF
Esqueceu senha	SE	Simples	4 PF
Capacitação	ALI	Média	10 PF
Incluir evento de capacitação	EE	Média	4 PF
Alterar evento de capacitação	EE	Média	4 PF
Planejar evento de capacitação	EE	Média	4 PF
Consultar plano evento capacitação	CE	Média	4 PF
Definir cronograma de capacitação	EE	Média	4 PF
Consultar cronograma evento capacitação	SE	Média	5 PF
Consultar eventos de capacit. por data e local	SE	Média	5 PF
Consultar detalhes de evento de capacitação	CE	Complexa	6 PF
Incluir participantes para evento	EE	Média	4 PF
Alterar participantes para evento	EE	Média	4 PF
Excluir participantes para evento	EE	Simples	3 PF
Consultar participantes cadastrados no evento	CE	Média	4 PF
Enviar para e-mail para participação do evento	SE	Média	5 PF
Informar avaliação de participante - resultados	EE	Simples	3 PF
Consultar avaliação de participante - resultados	CE	Média	4 PF
Emitir Certificado para o Participante	SE	Média	5 PF
Lista de Participantes com Emissão de Certificado Pendente	CE	Simples	3 PF
Avaliação de Evento de Capacitação	ALI	Simples	7 PF
Cadastrar avaliação de evento de capacitação	EE	Simples	3 PF
Alterar avaliação de evento de capacitação	EE	Simples	3 PF
Consultar avaliação de evento de capacitação	CE	Simples	3 PF
Consultar dados de acompanhamento de comunidade capacitada – detalhada	SE	Média	5 PF
Enviar e-mail de notificação para avaliação do evento	SE	Média	5 PF
Consultas Gerenciais (3 gráficos e 3 relatórios com dados calculados)	6 SE	Média	30 PF

Tabela 8. Estimativa de Tamanho do Sistema de Treinamentos (STREINA)

forma. Assim, o índice de produtividade hipotético utilizado é o de 12 horas/PF. Então, a estimativa de esforço é de: 187 x 12 = 2244 HH (homens_hora).

O próximo passo é a estimativa de prazo, aplicando-se a fórmula de Caper Jones (1998) de aproximação de Tempo Ótimo de Desenvolvimento (Td) com um expoente t = 0,34, tem-se: $Td = 187^{0,34} = 5,92$ meses. Considerando-se o prazo estimado de 6 meses e o esforço estimado de 2244 HH, o próximo passo é a estimativa do tamanho da equipe de desenvolvimento ideal para atuar no projeto em questão. Segundo Jones (1997), a produtividade média diária no Brasil é de 6 horas/dia. E ainda, em média um mês possui 22 dias úteis. Então, tem-se: prazo = (esforço em HH) / (tamanho equipe * 6 * 22). Então, aplicando-se a fórmula, obtém-se o tamanho da equipe ideal para atuar no projeto, que deve ser constituída por três recursos, incluindo desenvolvimento e gestão.

O COCOMO II também possui fórmulas para o cálculo do Td, baseadas em parâmetros de calibragem e de mapeamento (Boehm, 2000). É importante destacar que as organizações devem ter ou construir um banco de dados histórico, contendo informações de projetos concluídos, com a finalidade de gerar as estimativas de prazo, custo e esforço com uma acurácia adequada.

Aplicando-se o COCOMO II para as estimativas de esforço e prazo do STREINA, tem-se o seguinte:

- Fator de Produtividade Linear (Pessoas_Mês/KSLOC) para desenvolvimento WEB: 2,51 (Roetzheim, 2005);
- Fator Exponencial para desenvolvimento WEB: 1,030 (Roetzheim, 2005)
- 1 PF = 33 SLOC em JAVA (Jons, 1997);
- Esforço = Fator de Produtividade x $KSLOC^{\text{Fator Exponencial}}$ (Roetzheim, 2005);
- Esforço = 2,51 x $[(33 \times 170)/1000]^{1,030}$ =

14,83 Pessoas_Mês. Note o que o insumo para o COCOMO é PF Não Ajustado, por isso considerou-se 170 PFs;

• O COCOMO considera o dia com 7h e o mês com 22 dias úteis, então o esforço em HH é o seguinte: 14,83 * 7 * 22 = 2284 HH. Note que o cálculo do esforço estimado é próximo ao 2244 HH obtido pelo modelo simplificado;

• Prazo (Td) = 2,5 x (Esforço^{0,32}) = 5,9 meses (Aguiar, 1999);

• Note que o prazo estimado pela fórmula de Caper Jones – 5,92 meses, ficou bastante próximo do prazo estimado pelo COCOMO.

Na prática, torna-se importante aplicar mais de um modelo de estimativas e analisar-se os resultados obtidos. Caso fosse utilizada a fórmula de esforço da ferramenta Cost Xpert, apresentada por Aguiar em 1999, o resultado seria o seguinte: E= 2,4 x (V ^{1,05}) = 14,68 Pessoa_Mês, sendo V o tamanho do projeto em KSLOC. Note que este é bem próximo do esforço estimado, utilizando-se os parâmetros propostos por Roetzheim em 2005. Caso fosse utilizado o esforço de 14,68 como insumo para o cálculo do prazo, este continuaria sendo estimado em 5,9 meses.

A estimativa de custo deve considerar o valor da hora da equipe alocada ao projeto (custo de pessoal), bem como outros custos de ambiente, ferramentas, deslocamentos, consultoria, etc. Algumas empresas possuem dados históricos de custo por PF de projetos concluídos, possibilitando a derivação direta da estimativa de custo a partir da estimativa de volume em Pontos de Função.

A estimativa de recursos computacionais críticos em um projeto WEB deve considerar, dentre outros, a disponibilidade dos servidores utilizados para desenvolvimento, homologação e produção do sistema e outros recursos de hardware relevantes.

É importante ressaltar que após a geração das estimativas, deve-se adicionar um percentual, prevendo uma evolução natural dos requisitos do projeto. Este percentual deve ser obtido por meio de dados de históricos de um indicador de estabilidade de requisitos de projetos concluídos da organização.

As ferramentas de estimativas também trabalham desta forma, solicitando que o usuário (estimador) forneça tal percentual. No entanto, devido à ausência de dados históricos analisados para tal indicador, a autora tem se baseado em análise de publicações e sua experiência profissional. A autora tem utilizado um percentual de 20% a 35% para os projetos com Documento de Visão. Esta variação de percentual é diretamente proporcional ao detalhamento do Documento de Visão e conhecimento dos requisitos do projeto pelo analista de negócios.

Para alguns projetos estimados, baseados apenas em atas de reunião ou documentos menos detalhados, foi utilizado o percentual de 40%. Em projetos pequenos com documentos de requisitos com um detalhamento adequado, o percentual varia de 10% a 25%.

A CEPF deve ser refinada, conforme o conhecimento dos requisitos do projeto evolui, em marcos definidos no plano do projeto. Assim, não há retrabalho no processo de estimativas, as estimativas de PF anteriores não são descartadas e sim refinadas, conforme os requisitos do projeto evoluem.

Conclusão

A indústria tem demonstrado dificuldade na previsibilidade de prazo e custo dos projetos de software. No entanto, muitas organizações ainda estimam projetos, sem a utilização de processo, de maneira "artesanal", baseando-se apenas na opinião dos líderes ou gerentes do projeto. De fato, o método de estimar projetos baseando-se na opinião de especialistas é bastante eficaz. O problema é quando a equipe não possui especialistas no domínio do projeto em questão.

Este trabalho apresentou um processo para as organizações nas estimativas de projetos de software, utilizando métodos aderentes às melhores práticas do CMMI nível 2. Espera-se que com a utilização de processos, o cenário caótico em relação à previsibilidade de projetos seja melhorado.

O método Contagem Estimativa de Pontos de Função tem sido utilizado com sucesso pela autora, além deste apoiar nas estimativas dos projetos, este também tem se mostrado bastante eficaz no suporte ao processo de engenharia de requisitos.

Referências

AGUIAR, M. ***Estimativas Confiáveis de Prazos para Gerentes de Projetos***. Developer's Magazine, Maio 1999, pp. 30-32.

AGARWAL, R. et al. ***Estimating Software Projects***. ACM SIGSOFT, Software Engineering Notes vol 26 nº4, July 2001, pp.60-67.

BOEHM, B.W. ***Software Cost Estimation With COCOMO II***. Prentice Hall, New Jersey, 2000.

DEKKERS, C. ***Measuring the "logical" or "functional" Size of Software Projects and Software Application***. Spotlight Software, ISO Bulletin May 2003 pp10-13.

HAZAN C.; STAA, A. v. ***Análise e Melhoria de um Processo de Estimativas de Tamanho de Projetos de Software.*** Monografias em Ciências da Computação nº 04/05, Departamento de Informática PUC-Rio, ISSN 0103-9741, Fevereiro 2005.

IFPUG. ***Counting Practices Manual***. Version 4.2.1, January, 2005.

JONES, C. *Applied Software Measurement: Assuring Productivity and Quality*. Prentice Hall, Second Edition, 1997.

JONES, C. ***Estimating Software Costs***. McGraw-Hill, 1998.

MELI, R.; SANTILLO, L. ***Function Point Estimation Methods: A Comparative Overview***. Proceedings of FESMA 99, Amsterdam, Netherlands, October 1999, pp. 271-286.

PRESSMAN, R. S. ***Engenharia de Software***, 6ª edição, MC Graw Hill, São Paulo, 2006.

ROETZHEIM, W. ***Estimating and Managing Project Scope for New Development***. Crosstalk –The Journal of Defense Software Engineering, April 2005, pp. 4 -7.

MPS.BR – Mitos e Verdades a Respeito de um Modelo de Maturidade

Andriele Ferreira Ribeiro
Bacharel em Ciência da Computação (UFMG). Mestre em Administração de Empresas (UFMG). Pós-graduado em Melhoria de Processo de Software (UFLA). Certificado como Project Management Professional (PMP) pelo Project Management Institute (PMI). Certificado como implementador e avaliador oficial do modelo MPS.BR. Professor de disciplinas relacionadas à gerência de projetos e administração de empresas. Vice-Presidente de Filiação do PMI-MG no ano de 2006. Gerente de Modernização do Desenvolvimento de Sistemas da PRODEMGE, tendo atuado como gerente de projetos, consultor em escritório de projetos e melhoria de processos no DCC – UFMG, PRODEMGE e FUMSOFT.

Qualquer área de conhecimento atualmente é controversa. Frase de efeito para se iniciar um artigo, não é mesmo? Mas pense bem se ela não faz sentido. A quantidade de informações relativas a qualquer ramo de atuação – administração, medicina, nutrição, publicidade, preparação física, etc – é muito grande. Vamos tomar como exemplo a nutrição. Existem dietas dos mais diferentes tipos visando o emagrecimento: as que envolvem proteínas predominantemente, as que dosam os diversos tipos de alimento, as que excluem quase todos os alimentos, as que associam seu signo à rotina alimentar e muitas outras. A presença de muitas informações tem como conseqüência direta a existência de pontos de vista diferentes.

Quando falamos de desenvolvimento de software e qualidade, a realidade não poderia ser diferente. Abordagens técnicas e gerenciais são criadas, discutidas, modificadas com o objetivo de se produzir software de qualidade. Existem metodologias, modelos, normas, corpos de conhecimento, enfim, muitas linhas escritas em diversos formatos e com diversos objetivos tratando do tema. Isto gera divergências, naturalmente. Às vezes porque elas existem de fato. Mas em muitos casos por desconhecimento de quem critica.

Ultimamente tenho lido muitos textos, depoimentos, e-mails em listas de discussão e até ouvido podcasts em que modelos de maturidade tais como o MPS.BR (a denominação correta do modelo é MR-MPS, mas o nome do programa que lhe deu origem – MPS.BR – tem sido mais comumente utilizado para designá-lo) e o CMMI (Capability Maturity Model Integration) são questionados. Diria que em alguns casos eles são até mesmo execrados. Até aí, problema nenhum. A liberdade de expressão garante o direito de todos se manifestarem livremente e críticas dirigidas a qualquer coisa podem, em muitos casos, trazer benefícios à entidade sendo criticada. Estas são as

chamadas críticas construtivas, baseadas em argumentos sólidos e consistentes. Infelizmente não é exatamente isto o que tenho visto por aí...

Para que vocês possam ter uma idéia melhor do que estou dizendo, vou dar alguns exemplos de comentários absolutamente equivocados que têm circulado na Internet. Escrevo algumas palavras ao fim de cada um deles, tentando mostrar o porquê do equívoco.

1. Modelos de maturidade só se preocupam com processos. A competência das pessoas é desconsiderada.

Este é um engano recorrente. Os processos são de fato bastante valorizados nos modelos. Mas há também resultados / práticas / objetivos específicos, tanto no MPS.BR quanto no CMMI, referenciando diretamente a necessidade de competência e conhecimento das pessoas em relação às atividades que executam. Mais à frente ainda neste artigo falarei mais a respeito deste assunto.

2. Modelos de maturidade estão ressuscitando o taylorismo. Um grupo de pessoas completamente alheias à realidade de quem "põe a mão na massa" cria um bando de processos burocráticos.

Qualquer iniciativa de melhoria de processo de software está condenada ao fracasso se os executores das atividades que estão sendo descritas no processo são alijados das discussões. Portanto, não há cabimento em se associar este tipo de comportamento ao modelo A, B ou C. Se isto ocorre dentro de alguma empresa, a falha é de quem está conduzindo o projeto de melhoria ou de quem está orientando a sua execução (consultores, implementadores, etc) e não do modelo.

3. Scrum ou XP são preferíveis ao CMMI / MPS.BR porque usam uma abordagem iterativa e não cascata.

Não há a definição de uso de um ciclo de vida de projeto específico nos modelos de maturidade. O que é necessário para que haja aderência aos modelos é a definição de qual ciclo de vida será utilizado para cada um dos projetos. Pode ser cascata, iterativo, misto, incremental, entrega evolutiva ou qualquer outro.

4. Dezoito meses é o tempo necessário para se avaliar com um sucesso uma empresa no nível G do MPS.BR.

Não há um tempo padrão para se implementar os processos de um ou outro nível. Isto varia de empresa para empresa em função de uma série de fatores tais como grau apoio da alta direção, investimento direto no projeto de melhoria de processos e nível de conhecimento dos colaboradores. A Softex (instituição mantenedora do modelo) tem dado apoio financeiro para cobertura de parte dos custos de consultoria para empresas que atendam a alguns requisitos. Entre eles está o compromisso de se realizar uma avaliação após 12 meses e não 18. Mas isto não quer dizer que o tempo seja este ou aquele. Este compromisso existe apenas para empresas que participam de algum grupo que recebe este apoio financeiro.

5. O MPS.BR é burocrático demais. Se eu for gerar todos os documentos que são solicitados pelo modelo, não tenho tempo para fazer o mais importante: software de qualidade.

O MPS.BR não obriga a geração de nenhum documento específico. O modelo apenas exige que algumas formalizações sejam realizadas. E estas formalizações, em projetos não são mera burocracia. A obtenção de compromissos por parte de clientes e fornecedores, por exemplo, é muito mais segura se for formalizada. Às vezes leio ou escuto algo do tipo: "Se alguma mudança ocorreu no projeto, não preciso formalizar com meu cliente. No meu método de trabalho isso é apenas falado e pronto". Eu gostaria muito de trabalhar em um ambiente em que as relações funcionassem desta forma e em que a memória das pessoas fosse boa o suficiente a ponto de lembrar de todas as mudanças e impactos no projeto acordados apenas oralmente. Mas creio que o mundo real seja bem diferente disto na maioria absoluta das situações.

Agora que já falamos do que é errado em relação ao CMMI / MPS.BR, vamos falar do que os modelos realmente preconizam. Aqui cabe uma ressalva: alguns de vocês podem estar se perguntando por que estou tratando CMMI e MPS.BR de forma indistinta. A resposta é simples: porque tecnicamente falando eles são completamente compatíveis. Não estou entrando no mérito do reconhecimento internacional de ambos. Nesse ponto o CMMI é muito mais forte. Mas tecnicamente há apenas algumas pequenas diferenças, a saber: 1) Há alguns processos e, portanto, requisitos adicionais no MPS.BR ligados às áreas de gestão do conhecimento e gerência de reuso. 2) O MPS.BR apresenta maior flexibilidade na exclusão de processos a serem considerados na avaliação.

Estas diferenças não impedem a compatibilidade entre eles. Para que após uma avaliação MPS.BR a empresa avaliada esteja apta a realizar uma avaliação CMMI em nível similar com sucesso, basta que parte da flexibilidade apresentada pelo MPS.BR não seja exercitada. Ou seja, se um determinado processo pode ser excluído em uma avaliação MPS.BR e tal exclusão não pode ocorrer no CMMI, a empresa deve se preocupar em atender também aos resultados esperados deste processo, mesmo que ele possa não ser avaliado oficialmente no caso do MPS.BR. Feita a ressalva, voltemos ao que os modelos realmente objetivam. Para mostrar o pragmatismo da melhoria de processos de software via MPS.BR serão citados e comentados alguns resultados esperados de alguns processos do modelo. Ficará fácil de visualizar que não há nada de burocrático ou sobrenatural sendo proposto. Apenas para fins de esclarecimento, as siglas no início de cada resultado são abreviações dos nomes dos processos aos quais eles pertencem e o número representa a posição do resultado dentro do processo. Nos exemplos abaixo são citados resultados dos processos Gerência de Projetos (GPR), Gerência de Requisitos (GRE) e Medição (MED).

GPR1. O escopo do trabalho para o projeto é definido;

Responda com sinceridade: há algo errado em se definir o escopo de um projeto? Eu não acho que haja. E mesmo aqueles que defendem um escopo "flutuante" em que as solicitações de mudanças são tratadas como benéficas e naturais não podem prescindir de algum nível de definição inicial deste escopo. Para que a mudança seja possível é preciso saber o que se quer mudar.

GPR 6. Os riscos do projeto são identi-

ficados e o seu impacto, probabilidade de ocorrência e prioridade de tratamento são determinados e documentados;

Antes de comentar este resultado, vamos recordar a definição de projeto presente no PMBOK: "Um projeto é um esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo." A organização do trabalho em projetos, sejam eles de desenvolvimento de software ou não, se dá fundamentalmente para que seja possível lidar com os desafios inerentes à criação de um resultado novo, exclusivo, diferente em algum aspecto de tudo que já foi criado anteriormente. Para gerenciar a rotina, métodos mais simples são suficientes. Com projetos a coisa é diferente. Como sempre há algum grau de novidade, incertezas são inerentes ao conceito de projeto e é fundamental que o gerenciamento dos riscos ocorra em algum nível. A única certeza é que os riscos existem e se os mesmo não forem devidamente tratados, a probabilidade de que algum deles ocorra e prejudique algum aspecto do projeto é de quase 100%.

Esse resultado, presente no processo Gerência de Projetos, é o mínimo que se espera em relação ao gerenciamento de riscos no MPS.BR. Exige-se que os riscos sejam identificados (nada mais natural), classificados em relação à sua probabilidade e impacto e priorizados em função destas características. A priorização é importante para que aquilo que é mais importante tenha um cuidado maior do que aquilo que potencialmente pode trazer menos danos ao projeto.

GPR 7. Os recursos humanos para o projeto são planejados considerando o perfil e o conhecimento necessários para executá-lo;

Falamos de pessoas agora há pouco, no comentário sobre uma das críticas infundadas ao modelo. Este resultado é um dos pontos de valorização das pessoas dentro do modelo. O que ele pede? Simples. Que as alocações dos recursos humanos aos projetos não se dêem apenas por disponibilidade, ou seja, que alguém não seja alocado simplesmente porque está livre ou menos atarefado. Cada papel ou função dentro de um projeto requer uma série de conhecimentos e habilidades e o não atendimento destes "pré-requisitos" normalmente traz consigo muitos problemas. É razoável alocar alguém que nunca desenvolveu em Java como programador em um projeto que só usará esta linguagem, sem que nenhuma ação de capacitação seja realizada? Faz sentido promover alguém a gerente de projetos se esta pessoa não conhece sequer o conceito de projeto? A resposta NÃO parece óbvia, mas surpreendentemente isto ocorre dentro das empresas. Este resultado visa justamente mudar esta cultura, fazer com que as empresas prestem mais atenção ao tratamento dispensado às pessoas que nela trabalham.

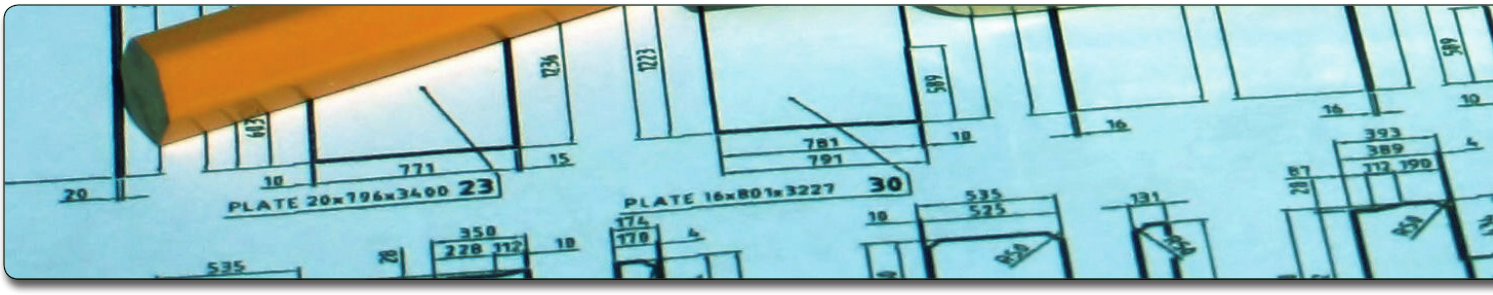
Certa vez, quando eu estava em uma reunião de consultoria em uma empresa, meu interlocutor deu gargalhadas quando expliquei o significado deste resultado para ele. E comentou: "Isto é piada! Nossos funcionários são estes e pronto. Se eles não têm conhecimentos ou habilidades necessárias, durante o projeto eles dão um jeito de adquiri-los". É claro que não há uma rigidez tão grande no modelo a ponto de impedir que a falta de um conhecimento num primeiro momento inviabilize a alocação de uma pessoa ao projeto. Mas a coisa também não pode se dar de maneira tão informal. Se um analista está acostumado a especificar requisitos usando alguma técnica da análise essencial, por exemplo, ele poderia ser alocado à função de especificador usando casos de uso, desde que algum treinamento ou qualquer outra ação de capacitação que suprisse suas necessidades fosse planejada e executada. O que o modelo quer evitar é que "vá se dando um jeitinho" que a pessoa com um déficit de conhecimento importante vá adquirindo este conhecimento de qualquer forma.

GPR 13. O progresso do projeto é monitorado com relação ao estabelecido no Plano do Projeto e os resultados são documentados;

Planejamento e controle são dois lados de uma mesma moeda. Não há muito sentido em se planejar se não houver um monitoramento constante que avalie se o plano está de fato sendo concretizado ou se necessita de alguma ação de correção de rota – ação esta que tanto pode estar ligada à execução quanto à modificação do próprio plano original, desde que haja a concordância dos diversos envolvidos no projeto.

Já ouvi muito a pergunta: pra que planejar se meu projeto sofre mudanças o tempo todo? Apesar de freqüente, não vejo muito sentido neste questionamento. Se há muitas mudanças, o planejamento poderá mudar mais. Só isso. Mas dependendo do contexto do projeto, se o monitoramento não for disciplinado o suficiente, tais mudanças irão ocorrer de forma totalmente desordenada e aí sim o fracasso é certo. Planejamento, controle e alguma dose de formalização são aliados e não inimigos dos projetos ditos mais "dinâmicos". Para evitar um trabalho muito grande na construção do plano inicial e nas modificações subseqüentes, uma abordagem que pode ajudar muito é o planejamento por ondas sucessivas. Apesar do nome diferente, planejar desta forma nada mais é do que refinar o plano de forma progressiva. Ou seja, detalhamos apenas aquilo que é um pouco mais conhecido, aquilo que está próximo do momento atual. Etapas mais avançadas do projeto são planejadas de forma mais genérica dado que é impossível detalhar o que não se conhece.

Para ilustrar, vamos a um exemplo prático do que está sendo dito acima: se



ainda estou definindo os requisitos de um software, é inútil tentar prever datas de início de fim para a construção de cada um destes requisitos em separado. Ainda não há informação suficiente para este nível de detalhamento. Planejar desta forma nada mais é do que um exercício de futurologia. Entretanto, baseado em alguma medida de tamanho de software, pode-se planejar de forma menos detalhada e estimar que a etapa de construção de um conjunto de requisitos que totalizam N pontos de função, se iniciará no dia X e finalizará no dia Y.

GRE 1. O entendimento dos requisitos é obtido junto aos fornecedores de requisitos;

O texto deste resultado não parece nada absurdo, não é mesmo? Entender o que tem que ser feito (requisitos) junto a quem sabe o que tem que ser feito (fornecedor de requisitos) é algo tão óbvio que ficamos até desconfiados, e nos perguntamos se é isto mesmo o que o modelo quer dizer. Mas não há nenhuma pegadinha não. O que se espera é que os requisitos sejam entendidos, definidos e que tal definição tenha a participação de uma pessoa (ou de mais pessoas) chamada fornecedor de requisitos, que nada mais é do que o eleito por outros stakeholders do projeto (normalmente o patrocinador) para ser aquele que diz o que o software deve fazer para quem irá desenvolvê-lo. A única necessidade adicional não presente no texto acima é o fato de que tais requisitos devem ser documentados. Alguns críticos dos modelos de maturidade fazem críticas ferrenhas às exigências de documentação, mas convenhamos: algum nível de registro de requisitos é necessário haver. Mais uma vez, dizer que não se documenta requisitos porque eles mudam demais não me parece uma boa justificativa por motivos já expostos anteriormente – para que se possa mudar sem levar o projeto para o buraco é preciso saber e se ter controle do que está sendo mudado. Além disso, não há um formato específico obrigatório para se documentar os requisitos. Deve-se documentar o necessário para que possa ser possível entender o que deve ser feito. Cada empresa pode e deve definir a sua forma de fazer isto.

MED2 - Um conjunto adequado de medidas, orientado pelos objetivos de medição, é identificado e/ou definido, priorizado, documentado, revisado e atualizado.

"Não se gerencia o que não se mede, não se mede o que não se define, não se define o que não se entende, não há sucesso no que não se gerencia."

A frase acima é de William Edwards Deming, um dos papas da qualidade. Sua conclusão é clara: a empresa precisa se conhecer objetivamente para alcançar o sucesso. E medidas, bem entendidas, definidas, coletadas e analisadas são a base para este auto-conhecimento. Como saber se a produtividade da minha empresa está baixa ou alta? Ou até antes disto: como saber qual é a produtividade de minha empresa? Como avaliar se medidas tomadas para reduzir o número de defeitos em um produto foram ou não eficazes? Como aumentar a previsibilidade de meus projetos? Medindo, medindo e medindo – e claro, analisando e tirando o melhor proveito destas medições.

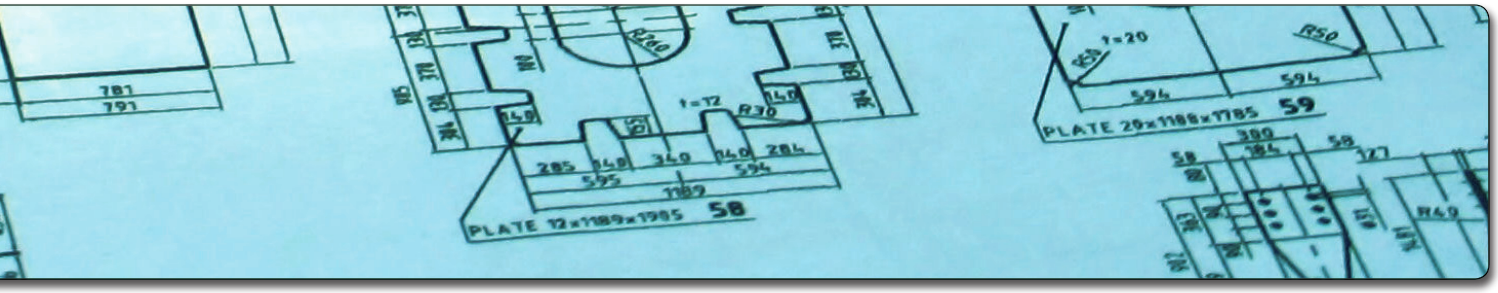
O resultado MED2 fala também em objetivos de medição. Estes objetivos são derivados das necessidades de informação da organização. Ou seja, devemos medir aquilo que é importante para a empresa. É importante que o processo operacional de medição esteja completamente alinhado com aquilo que a empresa precisa para melhorar sua gestão. Isso é verdade na indústria de software, mas também em qualquer outro ramo de atuação.

Os resultados citados e explicados neste artigo são apenas uma pequena amostra do modelo. Eles foram extraídos de processos ligados aos níveis G e F apenas (primeiros níveis numa escala que se inicia no G e vai até o A). Entretanto, esta amostra, apesar de pequena, representa bem o propósito de um modelo de maturidade: ser um guia de referência para a aplicação de boas práticas dentro das empresas que desenvolvem software. É interessante observar que dentre os resultados que apresentei não há nada ligado a processos de engenharia de software. Não citei nada referente a design, codificação, ferramentas, testes automatizados, etc. Isso costuma gerar uma certa frustração no profissional essencialmente técnico. Mas há uma explicação para esta ausência. Os níveis iniciais do MPS.BR (G e F) se preocupam em arrumar a casa do ponto de vista gerencial. Processos mais técnicos são também bastante relevantes, mas é impossível aplicá-los de forma consistente se a gestão dos projetos de software é caótica. Esses processos mais técnicos são tratados mais à frente, em níveis superiores depois que uma base sólida já foi construída.

Conclusão

Qualidade de software não é uma ciência exata. Discussões e divergências sempre ocorrerão. A internet potencializa as discussões e gera um fluxo maior de informações. O problema é que há informações certas e erradas, fundamentadas e levianas, racionais e apaixonadas. É cada vez mais importante saber separar o joio do trigo.

Modelos de maturidade não são garantia de sucesso absoluto para nenhuma empresa. Mas certamente servem muito bem como um guia de referência, um caminho a ser seguido por empresas que desejam melhorar seu desempenho operacional de forma consistente ao longo do tempo. ●



Desenvolvimento de Software Dirigido por Caso de Uso

Parte I: Conceituando e Entendendo Caso de Uso

Vinicius Lourenço de Sousa
viniciuslsousa@hotmail.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 Analista de Sistemas 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.

Cada vez mais o desenvolvimento de software fica mais complexo. Devemos isso às regras de negócios das grandes corporações que por sua vez estão cada vez mais complexas. O mundo não é mais o mesmo, ele tornou-se pequeno graças à internet. Milhares de pessoas têm acesso às informações e aos produtos vendidos no mercado praticamente de forma instantânea. Isso fez com que as empresas também tivessem que investir para ganhar espaço nesse "Mundo Novo".

Então, para que o desenvolvimento de software tenha sucesso, é fundamental que nós desenvolvedores de software tenhamos a total compreensão do processo de negócio dos clientes. Com isso pode-se entender claramente a real necessidade dos clientes em relação ao seu próprio negócio. Com o conhecimento sobre o negócio, temos a noção dos riscos para o desenvolvimento e como o software deverá se comportar no dia a dia do cliente.

Mas sabemos que em desenvolvimento de software nada é constante e sim mutável. As regras mudam a toda hora, manutenções são feitas em funcionalidades já entregues, novos sistemas surgem e se comunicam com o nosso através de vários protocolos e tecnologias diferentes. Entretanto, o pior de tudo é quando uma determinada informação do negócio muda ou uma nova funcionalidade passa a existir, pois isso acaba acarretando várias alterações em cascata no desenvolvimento do software, como: fazer novamente a análise, o projeto, a implementação, os testes e etc. Dessa forma, é necessário ter uma maneira de mapear e manter registrado todos os requisitos e regras de negócio do cliente para que no futuro possamos ter uma rastreabilidade de impacto no caso de alteração de um requisito ou de uma regra de negócio. Essa rastreabilidade é de total importância, pois com ela pode-se determinar, por exemplo, a complexidade da alteração, os artefatos (documentos) que necessitarão serem alterados e o prazo da alteração do

software, pois como diz o ditado, "tempo é dinheiro".

Atualmente, existe um artefato muito utilizado que nos permite dirigir todo o desenvolvimento de software. Com ele, podem-se rastrear os impactos nos requisitos de software e nos demais artefatos que foram criados a partir dele. Esse artefato chama-se Caso de Uso (Use Case). O caso de uso é usado em todo desenvolvimento através de vários "cenários" e por vários membros da equipe de desenvolvimento. Por exemplo: na criação da arquitetura do software, na análise e projeto físico, na criação dos planos de testes e etc.

A motivação em escrever este artigo está no fato de ter percebido que muitas pessoas não sabem o que é um caso de uso realmente, ou seja, criam um caso de uso de maneira errada que ao invés de ajudar no desenvolvimento do software, acabará trazendo problemas no futuro. Para que isso não ocorra, temos que primeiro entender o que é um caso de uso antes de sair especificando-o.

Entendendo o que é um Caso de Uso

Muitos analistas de sistemas criam casos de usos e seus diagramas de forma inapropriada. Com isso, todo o desenvolvimento de software é afetado, gerando inúmeras dificuldades no entendimento da aplicação. Na verdade, para um caso de uso ser considerado válido, sua especificação deve seguir algumas regras. Essas regras existem exatamente para deixar os casos de usos com fácil entendimento não só pela equipe de desenvolvimento, mas também pelos clientes. Vejam quais são essas regras:

Um caso de uso é uma seqüência de interações entre o ator (alguém ou algo que interage com o sistema) e o sistema, que acontece de forma atômica, na perspectiva do ator.

Isso significa que quando criamos um caso de uso apenas nos preocupamos com "o que" o sistema deve fazer e não com "o como" deve fazer. E aí está o maior erro existente em vários casos de uso que eu tenho encontrado em meus anos como desenvolvedor de software. Vários analistas quando especificam os casos de uso, acabam colocando como deve ser feito um determinado passo ou fluxo, descrevendo que aquela informação deve ser salva naquela tabela e/ou que aquela informação deve ser lida daquela tabela e daquele campo.

Nunca devemos esquecer que a especificação do caso de uso será lida e validada pelo nosso cliente e que eles podem ser ou não da área de TI. Neste caso, mesmo se forem, o objetivo principal do caso de uso é mapear o que o sistema deve fazer. Essa é a principal validação que o cliente fará no caso de uso e não se as informações ficarão na tabela XYZ e no campo ABC. Mas então, em que momento no desenvolvimento de software deve-se começar a pensar em como o sistema executará as funcionalidades? Isso fica para o projeto físico, onde aí sim nos preocuparemos com toda a parte tecnológica do sistema como banco de dados, servidores de aplicação, linguagem de desenvolvimento, estratégias de transação, alocação de memória e etc.

O fato das interações entre o ator e o sistema serem atômicas na perspectiva do ator, significa que cada passo representa uma única operação tanto por parte do ator quanto do sistema. Também significa que essa operação ao final de sua execução retornará o resultado esperado em caso de sucesso, não podendo haver resultados parciais.

Ao ser executado, um caso de uso deve fornecer um resultado observável e significativo para o ator.

Todo caso de uso deve representar uma solução sistêmica para o negócio do cliente, ou seja, não devem existir casos de uso para soluções tecnológicas como explicado no item anterior. E isso tem bastante influência no resultado que um caso de uso deve retornar para o ator, pois esse resultado também deve estar inserido dentro do contexto do negócio do cliente.

Como exemplo, posso citar um caso de uso para saque de dinheiro em um caixa eletrônico. Quando vamos ao caixa eletrônico para sacar dinheiro nós iniciamos um processo de negócio que conterá vários passos. Ao final do processo o resultado será o dinheiro que a máquina liberará, contando que tudo deu certo durante o processo. Perceba que o ato de sacar dinheiro pelo cliente é uma operação que está dentro do processo de movimentação bancária de qualquer banco existente, assim como as demais operações como transferência, depósito e etc. Portanto, percebemos que o resultado do caso de uso realmente está dentro do contexto do processo de negócio, pois o que se deseja é o dinheiro que é objeto da movimentação.

Um caso de uso representa uma visão externa do sistema. Ficam registradas todas as informações trocadas entre o ator e o sistema. Com isso obtemos o que chamamos de "contorno do sistema" ou "fronteira do sistema".

Como expliquei no primeiro item, em um caso de uso não se deve colocar como o sistema deve fazer o processo e sim o que ele tem que fazer. Dessa forma podemos determinar as responsabilidades dos atores e do sistema na execução de suas tarefas, pois como veremos mais a frente, atores também podem ser outros sistemas. Então imagine se o nosso sistema tivesse que interagir com outro sistema que já está pronto. Nós apenas queremos saber que tipo de funcionalidade (serviço) esse sistema pode nos oferecer e não como ele executa essa funcionalidade internamente. Vejamos a **Figura 1** para entendermos melhor.

Na **Figura 1**, vemos um diagrama de caso de uso para venda de ingressos pela internet. Nele o cliente envia as informações do seu cartão de crédito para a compra do ingresso. O nosso sistema deverá enviar essas informações para o sistema da operadora do cartão de crédito afim de validação. Repare que o diagrama tem dois atores, um (Cliente) sendo um usuário do nosso sistema e o outro (Operadora de Cartão de Crédito) sendo o sistema ex-

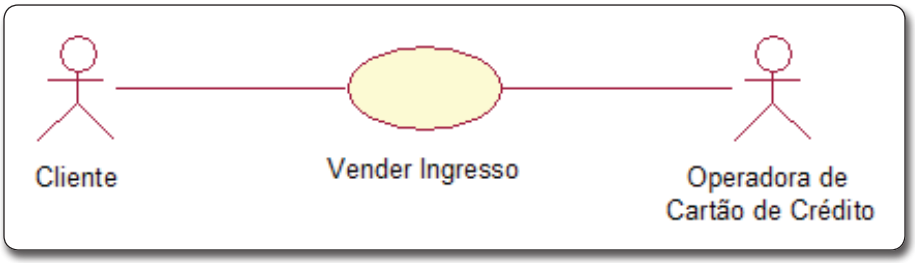


Figura 1. Venda de ingresso pela Internet

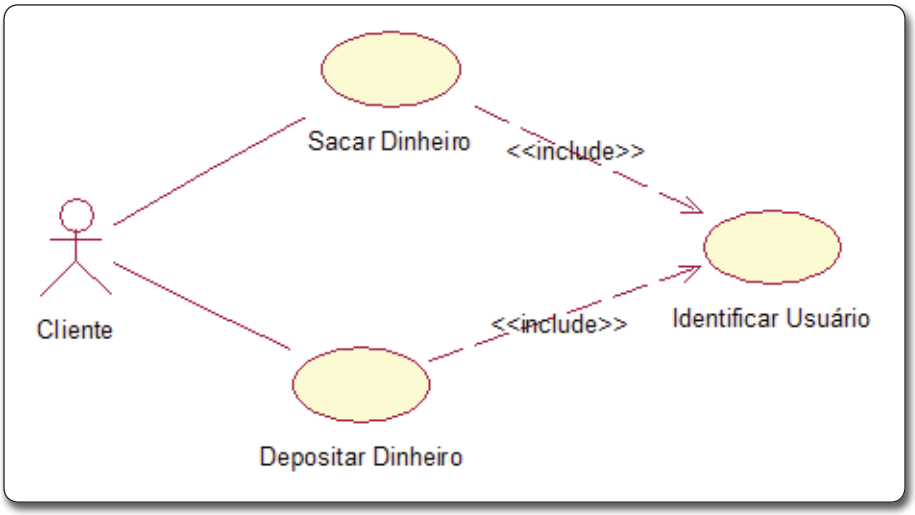


Figura 2. Movimentação Bancária

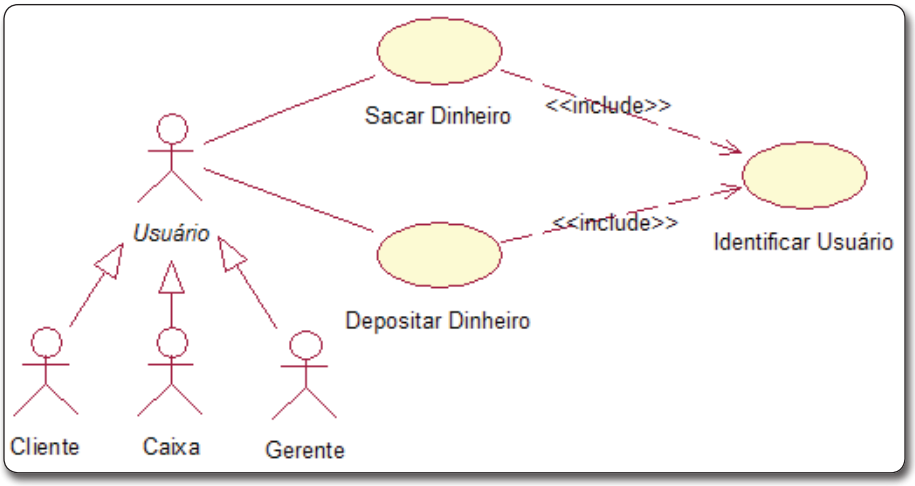


Figura 3. Movimentação Bancária

terno que validará os dados do cartão de crédito do cliente. Note que, assim vamos obtendo a "fronteira do sistema" que é tudo que o ator enxerga e/ou percebe do sistema, ou seja, as informações passadas para o sistema pelo ator e as informações passadas para o ator pelo sistema. Quando a informação dada pelo ator atravessa a fronteira do sistema, ela está dentro do sistema e o ator não sabe como essa informação será tratada, só importando o resultado.

Casos de uso com poucos passos devem ser avaliados se são realmente significativos para serem casos de uso e não parte de um caso de uso maior.

Esse item deve ser levado com muita atenção, pois às vezes é muito difícil decidir se um caso de uso é na verdade passos de outro caso de uso maior ou se um caso de uso é na realidade vários casos de uso embutidos que devem ser separados. Para tomarmos essa decisão devemos levar em conta alguns aspectos:

- Ter total conhecimento do processo de negócio do cliente;
- Sabermos se os passos de um caso de uso serão usados por outros casos de uso e assim podermos fazer reuso desses passos como outro caso de uso.

Lembre-se que um caso de uso é uma seqüência de interações entre o ator e o sistema e, por isso, casos de uso com poucos passos tendem a não mostrar essa interação de forma que possamos entender o processo de negócio. Por outro lado, existem passos tão importantes no contexto do negócio do cliente que muitas vezes é interessante separá-los para podermos fazer reuso desses passos. Vejamos a **Figura 2** para entendermos melhor.

Na **Figura 2**, vemos um diagrama de caso de uso para movimentação bancária de saque e depósito. Sabemos que nessas movimentações, existem passos que sempre serão executadas e um desses passos é a identificação do cliente, onde o cliente coloca seu cartão e o sistema validará os dados como agência, número da conta e senha para saber se é um cliente válido que pode fazer a movimentação. Por ser um passo de muita importância no contexto do negócio do cliente e por mais de um caso de uso ter esses mesmos passos, nós podemos separá-los a fim de reutilizarmos essa parte da interação ator e sistema. Mas devemos ter muito cuidado ao fazer isso, pois podemos acabar tendo uma enxurrada de casos de uso sem necessidade. A ponderação do que é importante para o negócio e para o sistema deve predominar nesse momento.

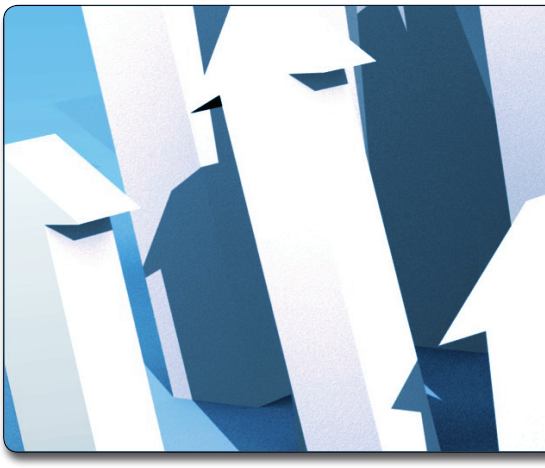
Agora que entendemos o que é um caso de uso, mostrarei algumas características importantes de um diagrama de caso de uso.

Características importantes de um diagrama de caso de uso

Ao criarmos um diagrama de caso de uso, também devemos nos atentar para certas características para que o diagrama não fique errado ou confuso no momento da especificação:

- Ator: Alguém ou algo que interage com o sistema. Um ator pode ser uma pessoa, um sistema externo (por exemplo: Figura 1) ou um hardware.

Para atores que representam pessoas, não podemos nomeá-los com os nomes das pessoas e sim com os papéis que essas pessoas representam dentro do contexto do negócio. Por exemplo: na **Figura 2**, temos o ator Cliente que pode efetuar um saque ou um depósito. O



nome Cliente representa um papel nesse contexto do negócio do nosso cliente, ou seja, qualquer pessoa que possa sacar ou depositar dinheiro é considerado um cliente. Essa característica é idêntica a um diagrama de classe, sendo o ator a classe e os clientes (João, Maria, José, etc.) as instâncias desse ator.

Para atores que representam sistemas externos, apenas os nomeamos com o nome do sistema.

Para atores que representam um hardware, devemos tomar um pouco de cuidado, pois não é qualquer hardware que será um ator, senão nosso diagrama de caso de uso terá diversos atores e que não representam nenhuma importância para o negócio. Um ator que representa um hardware deve ser de extrema importância para o negócio do cliente. Como exemplo, posso citar uma impressora fiscal que possui funções específicas, um leitor óptico e etc. Novamente a ponderação do que é importante para o negócio e para o sistema deve predominar nesse momento.

• Herança de Atores: Outra característica com atores é a possibilidade de criar heranças entre eles (o mesmo que herança de classes). Fazer herança entre atores tem um papel fundamental bem específico no diagrama de caso de uso. Voltando para o exemplo da movimentação bancária (Figura 2), sabemos que não é só o cliente que pode sacar ou depositar dinheiro, mas os próprios funcionários do banco (caixa e gerente) também podem. Nesse caso, poderíamos colocar mais dois atores no diagrama de caso de uso indicando que eles também podem efetuar essas movimentações. Quando nos deparamos com essa situação, poderemos trabalhar com o conceito de herança de atores (**Figura 3**).

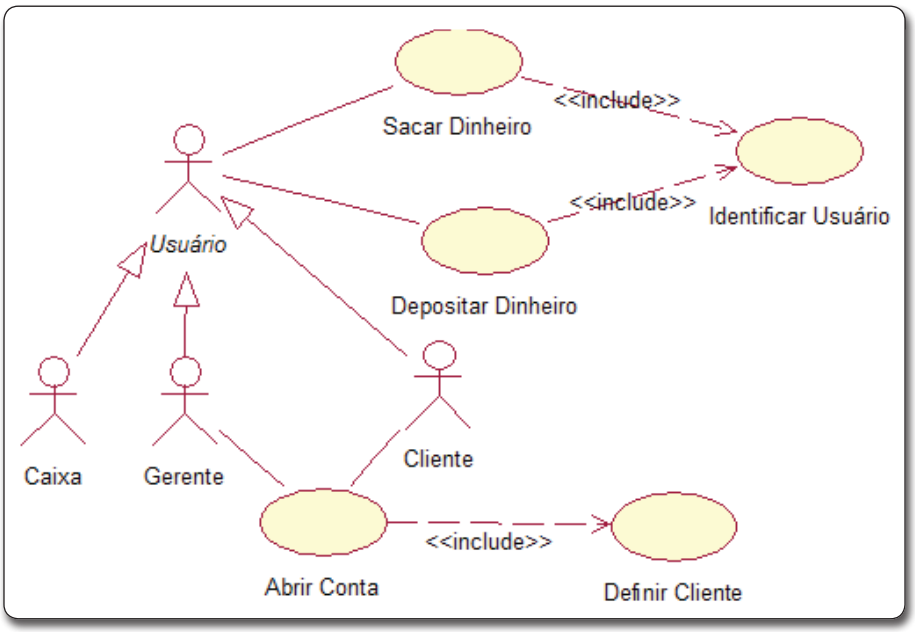


Figura 4. Movimentação Bancária

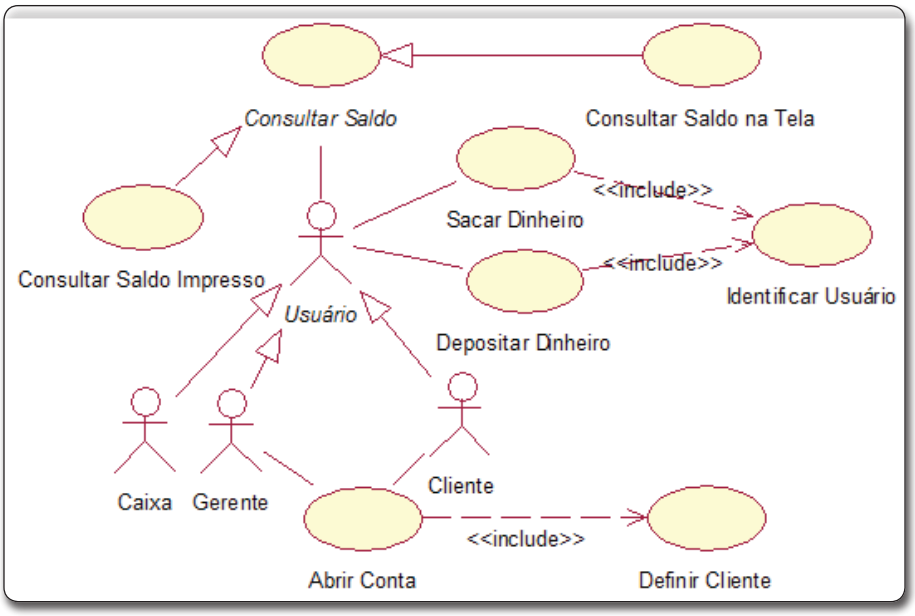
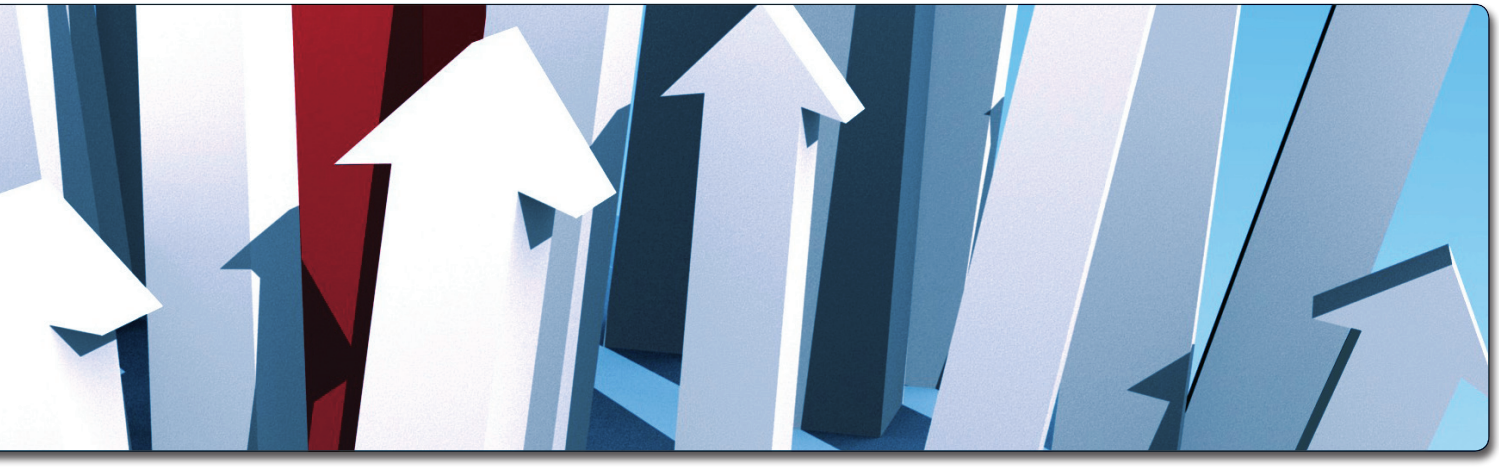


Figura 5. Exemplo de Herança



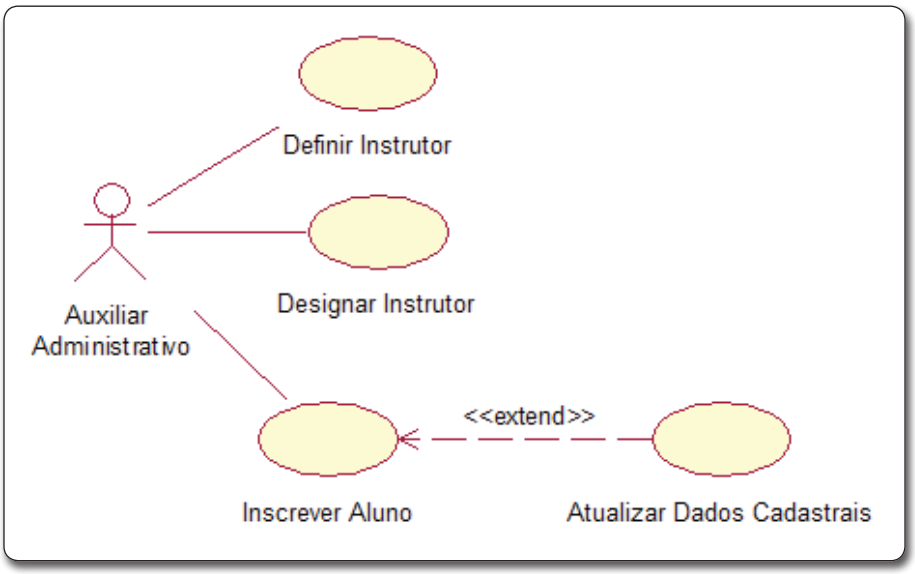


Figura 6. Exemplo de Extend

Na **Figura 3**, foi criado um ator abstrato chamado Usuário que é herdado pelos atores Cliente, Caixa e Gerente. O ator Usuário é quem irá executar os casos de uso Sacar Dinheiro e Depositar Dinheiro, sendo que dependendo do contexto na realidade será ou o cliente ou o caixa ou o gerente. Repare que o caso de uso Identificar Cliente foi renomeado para Identificar Usuário, pois agora pode ser qualquer um dos três a fazer a movimentação.

Outro ganho que temos ao usar a herança de atores é que começamos a mapear as permissões de acesso ao sistema. Por exemplo, na **Figura 4** somente o ator *Gerente* é que pode abrir uma conta na agência, ou seja, os outros atores não terão essa permissão. O ator *Cliente* participa como ator secundário, pois ele deverá informar a senha da nova conta.

• Caso de Uso: Como já explicado, um caso de uso mapeia a interação entre o ator e o sistema. Nesta seção apenas incrementarei as características de caso de uso dentro do diagrama de caso de uso.

Para uma perfeita compreensão do que um caso de uso controla, é fundamental que seu nome esteja dentro do contexto do negócio. Portanto, bons nomes de casos de uso são aqueles que o próprio cliente utiliza no seu dia a dia, por exemplo, Abrir Conta, Sacar Dinheiro e etc.

• **Herança de Casos de Uso:** Assim como podemos criar herança entre os atores, também podemos criar herança entre os casos de uso. Essa característica tem o mesmo objetivo que uma herança de classes em um diagrama de classe, ou seja, quando queremos atribuir novas características ao caso de uso sem perder a sua essência, criamos casos de uso (filho) que herdam do caso de uso mais abstrato (pai). Para entendermos melhor vejamos a **Figura 5**. Observe que na **Figura 5** existe um caso de uso abstrato chamado *Consultar Saldo* que serve para fazer as consultas do saldo da conta. No entanto, essa consulta pode ser feita na tela de um terminal ou ser impressa e dependendo da opção escolhida, a formatação das informações da consulta será diferente e até mesmo algumas informações poderão ser ou não apresentadas. Para resolver esse caso, foi feita a herança do caso de uso, criando os casos de uso *Consultar Saldo Impresso* e *Consultar Saldo na Tela*. O caso de uso *Consultar Saldo* terá todos os passos e regras comuns que serão usados pelos casos de uso filhos. Repare que a essência deles permanece inalterada, ou seja, consultar saldo. Apenas foram atribuídas funcionalidades diferentes para cada situação.

• **Reuso de Caso de Uso:** Como demonstrado na **Figura 4**, podemos reutilizar casos de uso para aproveitarmos as mesmas interações entre ator e sistema. Em um diagrama, podemos reutilizar um caso de uso de duas formas possíveis:

o **Include:** Essa característica significa que o caso de uso sempre será chamado. Como exemplo, na **Figura 4**, onde os casos de uso *Sacar Dinheiro* e *Depositar Dinheiro* incluem o caso de uso *Identificar Usuário* e o caso de uso *Abrir Conta* inclui *Definir Cliente*. Isso significa que os casos de uso *Identificar Usuário* e *Definir Cliente* sempre serão chamados quando *Sacar Dinheiro*, *Depositar Dinheiro* e *Abrir Conta* forem executados.

o **Extend:** Essa característica significa que o caso de uso poderá ou não ser executado, ou seja, dependerá do resultado de uma condição de negócio para ser decidido se ele será executado. Casos de uso que estendem são executados nos cenários alternativos (cenários alternativos serão detalhados mais para frente). Como exemplo, veja na **Figura 6** que o Auxiliar Administrativo ao inscrever um aluno em um treinamento poderá ou não atualizar os dados cadastrais do aluno. Se for a primeira vez que o aluno é inscrito no treinamento, então ele será cadastrado. Se for o segundo treinamento em diante o aluno poderá ter ou não seus dados cadastrais atualizados.

Conclusão

Nesta primeira parte vimos alguns erros na criação e especificação de casos de uso e como evitá-los. Também vimos seus conceitos, características e a importância do caso de uso no desenvolvimento de software. Na segunda parte do artigo explicarei detalhadamente como especificar casos de uso. ●

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

CBD x SOA

Serviços, processos de negócio e componentes que realizam os serviços

Gustavo Serafim
gustavo@sinis.com.br
Especialista IBM RUP. Pós-graduado em análise e design orientados a objetos. Atualmente apoia projetos de SOA/BPM na CPM. Experiências como consultor/instrutor em: adaptação do RUP para projetos com foco na arquitetura, dirigido por UC e desenvolvimento iterativo; técnicas de design de software (incluindo as técnicas de análise e design orientados a objetos); identificação/modelagem de processos de negócios e exploração de automação do processo; desenvolvimento baseado em componentes (CBD).

Muitas comparações já foram feitas entre desenvolvimento orientado a objetos (OO) e o desenvolvimento baseado em componentes (CBD). Componentes e objetos, possuem características similares, como a necessidade de interfaces bem definidas e a disponibilização de operações aos seus clientes. Essas semelhanças dificultam ainda mais o entendimento de OO e CBD. Algo muito parecido vem acontecendo no caso de CBD e SOA.

Como veremos, existem conceitos de CBD e SOA similares, por exemplo, encapsulamento, contratos e reuso, que podem dificultar a comparação dessas abordagens. Para uma melhor comparação é necessário compreender os conceitos essenciais envolvidos, quais sejam, processos de negócio, componentes realizadores dos serviços, entre outros.

O presente artigo não trata de tecnologia, como "REST versus SOAP/WS", ou "serviços implementados com EJB/RMI". Atualmente, existe uma infra-estrutura tecnológica pronta, que suporta o desenvolvimento OO + CBD + SOA sem muito esforço, mas não garante, apenas permite a aplicação das abordagens citadas. Web Services não são necessariamente SOA, e EJB não é necessariamente componente. Uma comparação de CBD x SOA é muito abrangente e, portanto, envolve questões de várias dimensões. Sendo assim, este artigo focará nos serviços que refletem os processos de negócio. Isto porque, é apenas nesse âmbito que o SOA pode cumprir sua promessa de alinhar TI com negócio.

SOA: evolução, não inovação

Desconfie quando alguém fala que uma abordagem em TI é totalmente nova, pois, normalmente, as novidades do mercado complementam ou corrigem as antigas, e não as substituem. Soluções como CBD e SOA se destinam, na maioria das vezes, a superar limitações humanas, como lidar com grande quantidade de informações complexas. Para lidar com tal comple-

xidade, a abstração é uma ferramenta chave, e está intimamente relacionada com o conceito de encapsulamento. Focando nesse aspecto essencial, Page Jones descreve em seu livro "*Fundamentals of Object-Oriented Design in UML*" os níveis de encapsulamento. Baseando-se nesses níveis, é possível partir de simples linhas de código e chegar ao SOA, passando por CBD, mesmo considerando que essas abordagens foram concebidas em contextos diferentes. Para uma visão geral desses dois conceitos, observe o esquema da **Figura 1**.

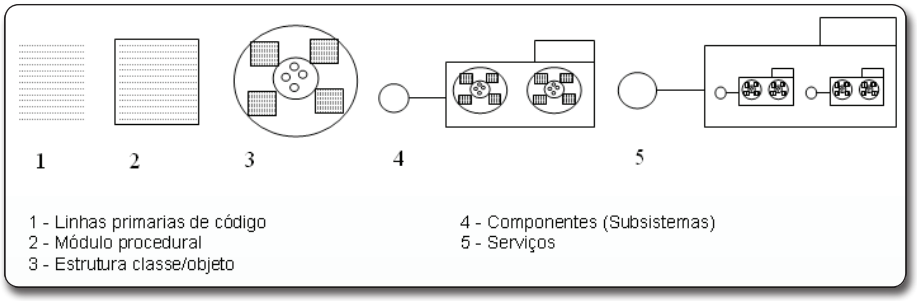


Figura 1. Esquema dos níveis de encapsulamento com serviços

No início, eram apenas linhas de código e vários GOTOs. Quem já programou assim lembra que, muitas vezes, face à necessidade de se fazer uma alteração, era mais fácil recodificar o programa inteiro em função da desorganização que existia. Além disso, para realizar qualquer manutenção, era preciso memorizar todo o código envolvido.

Com a adoção de sub-rotinas (primeiro nível de encapsulamento), foi possível reaproveitar as linhas de código. Estas linhas, encapsuladas através de uma sub-rotina, podiam ser chamadas quantas vezes fossem necessárias. Além disso, usava-se a decomposição funcional, ou seja, um problema era dividido em problemas menores (sub-rotinas) sucessivamente, e depois, as operações eram agrupadas em módulos lógicos. Infelizmente, a maioria das hierarquias de sub-rotinas geradas com essa abordagem não são suficientemente independentes para garantir reuso ou alterações controladas, pois, existe um acoplamento forte e sem controle. Assim, apenas esse nível de encapsulamento deixava a manutenção cara em ambientes com aplicações numerosas ou complexas. Neste caso, uma alteração no código poderia afetar de forma imprevisível todo o sistema, sendo necessário testá-lo novamente por completo.

Com a dificuldade na manutenção que a decomposição funcional provocava em sistemas grandes ou complexos, em determinado momento da história os dados passaram a ser o centro das aplicações, pois era possível mantê-los de forma relativamente estável. O código era tratado como algo descartável, ou seja, muito pouco esforço era feito para criar um código de qualidade e garantir o reuso no desenvolvimento. Dessa forma, o banco de dados passou a ser o centro de tudo. O efeito colateral dessa abordagem foi muita replicação de código.

Em função das limitações da abordagem de decomposição funcional, fez-se necessária a criação de mais um nível de encapsulamento: a classe. Nessa abordagem, as "sub-rotinas" e os "dados" estão em uma única estrutura. Os "dados" são protegidos através das "sub-rotinas", e os "dados" também são objetos. Para uma definição um pouco mais formal, temos que citar a retenção do estado e a identidade única. Na abordagem OO, o foco está em atribuir responsabilidades (comportamento) coesas e para cumprir tais responsabilidades, os objetos trocam mensagens entre si. O paradigma orientado a objetos busca meios de diminuir o "gap semântico". Em outras palavras, diminuir a distância entre o problema no mundo real e o modelo abstrato construído.

Com a estratégia de "dividir para conquistar", e também inspirado nos princípios da engenharia eletrônica, foi criado mais um nível de encapsulamento, qual seja, o componente de negócio. O componente encapsula objetos, protegendo-os com uma interface, de maneira contratual. O uso de contrato (Design by Contract) implica que cada uma das operações do componente é definida em termos de sua assinatura (seus tipos de argumentos de entrada e saída) e de suas pré-condições e pós-condições. Um componente oferece funcionalidades, por meio de interfaces bem definidas, para o meio externo. Componentes de negócio são subsistemas projetados para serem reutilizáveis corporativamente, mas a reutilização não é a única motivação. A necessidade de lidar com modificações de maneira rápida e controlada tornou-se essencial atualmente.

Serviços são parecidos com componentes em muitos aspectos, por exemplo, a ênfase em interfaces. Contudo, ele é a representação lógica de uma tarefa de negócio repetitiva que tem um resultado específico (por exemplo, verificação de crédito dos clientes e fornecer dados meteorológicos). Uma abordagem promissora mostra serviços sendo encontrados diretamente em processos de negócio. SOA é um estilo arquitetônico, que suporta a orientação a serviços, com foco nos processos de negócios**.** Para uma definição formal, veja o site The Open Group (http://www.opengroup.org/projects/soa/doc.tpl?gdid=10632). O desenvolvimento baseado em componente fornece uma base experimentada e testada para a implementação de SOA. Pode-se considerar que serviços, no caso do SOA, encapsulam componentes.

Conceituando SOA

Martin Fowler descreve a confusão em torno de SOA em um post no seu bliki, de leitura obrigatória, chamado Service Oriented Ambiguity (endereço relacionado no quadro Links). O cientista sustenta que é impossível responder o que é SOA, pois, para pessoas diferentes, SOA possui significados distintos. Entretanto, podemos relacionar algumas idéias e mitos recorrentes na conceituação de SOA, que, juntos, podem auxiliar na definição de tal estratégia - SOA é uma estratégia de TI para melhor atender o negócio e não apenas um projeto de desenvolvimento ou algumas ferramentas. Não existe SOA, por exemplo, se a empresa possui apenas um sistema.

Um dos principais erros na conceituação de SOA atualmente, mostra SOA como pura tecnologia, por exemplo, Web Services e EJBs. SOA tem dois aspectos fundamentais: a necessidade de se mapear **processos** e de ter **governança SOA**. Sem esses elementos não é possível obter os benefícios com SOA.

Considerando que as empresas são grandes coleções de processos e que SOA promove o alinhamento com o negócio, concluímos que existe uma forte conexão entre os processos de negócios e a identificação de serviço. Em geral, os processos de negócios concentram-se nas tarefas repetitivas executadas em uma seqüência lógica. As tarefas, em uma organização, realizam alguma meta, freqüentemente para fornecer valor na forma de produto ou serviço para uma parte externa, como um cliente ou um parceiro. O serviço é a representação desta tarefa de negócio repetitiva. O mesmo está relacionado intimamente com o processo de negócio e possibilita que ele seja um vocabulário comum entre TI e as áreas de negócio, facilitando a comunicação. O negócio é a base para a identificação de serviços, e, para a identificação ser efetiva, cada processo precisa ser modelado.

Governança SOA é focada no ciclo de vida dos serviços. Governança SOA estende a Governança de TI para assegurar os conceitos e princípios de SOA. Em outras palavras, é uma especialização de Governança de TI. A Governança SOA endereça questões como: Maturidade dos Serviços; Infra-estrutura para a gerência dos serviços; Educação e Treinamento; Regras e Responsabilidades; e Mudanças organizacionais.

Freqüentemente, relaciona-se SOA à integração de sistemas, o que não condiz com a realidade. Basta observar que EAI (Enterprise Application Integration) surgiu com o objetivo principal de resolver os problemas da integração ponto a ponto, também conhecido como "spaghetti integration", fornecendo um meio uniforme de integração, SOA tem ambições muito maiores. Além disso, EAI possui características algumas vezes conflitantes com SOA, como o fato de ser centrado em dados e não em processos e de não se direcionar pelo negócio. O que motiva a criação de um serviço não é a integração ponto a ponto. Entretanto, SOA dá grande ênfase às interfaces que, quando bem definidas, "facilitam" a integração e, até mesmo, o reuso de código. Para atender as necessidades de integração no contexto de SOA, evitando integração ponto a ponto, temos o Enterprise Service Bus (ESB). O ESB é um conceito, com algumas características de EAI, que fornece uma camada para a lógica de integração de serviços. Existem ferramentas para ESB que disponibilizam uma infra-estrutura flexível de conectividade para integração de aplicações e serviços, suportando SOA. O ESB tem como característica: roteamento de mensagens entre serviços; conversão de protocolos de transporte entre requisitante e serviços; transformação do conteúdo de mensagens entre o requisitante e o serviço; Mensagens Síncronas e Assíncronas (ver **Figura 2**).

O foco nas interfaces é o que dá ao SOA a habilidade de obter acoplamento fraco, composição, reuso e vários outros objetivos do projeto (design goals). A essência de SOA está mais relacionada à habilidade de atender o negócio rapidamente. Para tanto, obter um nível de granularidade para os serviços identificáveis nos processos de negócio é uma abordagem fundamental. Isto contribui principalmente em questões como a orquestração no contexto de BPM (Business Process Management) e a justificativa de investimentos em TI, uma vez que permite a visibilidade da associação do serviço com o negócio.

Podemos resumir BPM como a combinação de pessoas, tecnologia e processos. BPM é uma prática para melhorar a eficiência das organizações, automatizando os processos de negócios. A modelagem, a execução (orquestração de Serviço) e o monitoramento dos processos são disciplinas de BPM. A associação de BPM e SOA é importante para aumentar a capacidade de responder às mudanças do negócio.

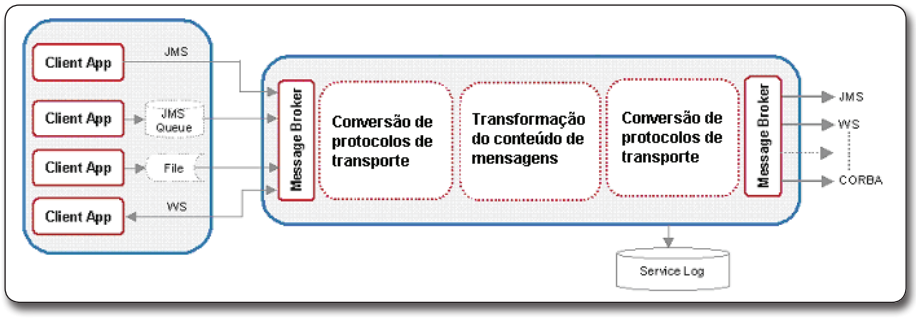


Figura 2. Enterprise Service Bus (ESB)

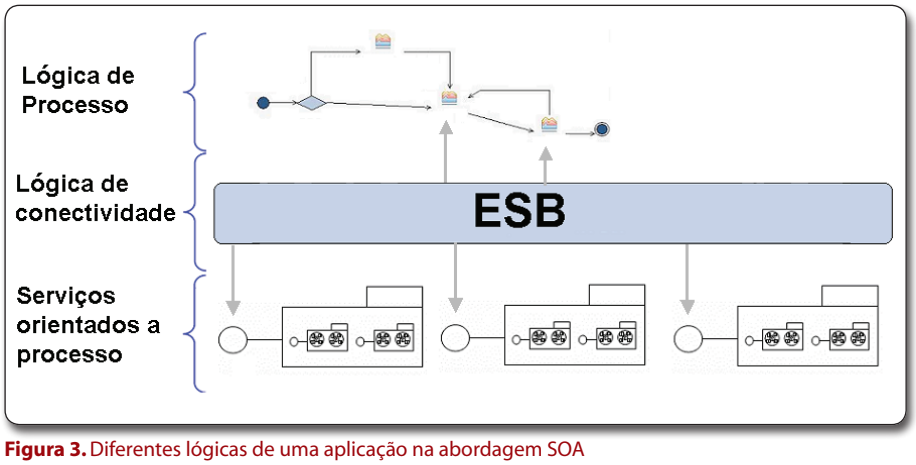


Figura 3. Diferentes lógicas de uma aplicação na abordagem SOA

Orquestração de Serviço

Depois de encontrados os componentes e serviços, a fim de se atender a uma aplicação específica, são necessários que se efetue a orquestração, ou seja, que tais serviços sejam "chamados" em uma ordem lógica, e que sejam cumpridas todas as pré-condições das interfaces envolvidas (Lógica de Processo). Em CBD, essa orquestração pode ser codificada pelo desenvolvedor na chamada "camada de sistema". Tal camada poderia também orquestrar serviços no caso do SOA, mas, para uma maior flexibilidade e para atender as necessidades impostas pelo BPM, o padrão de mercado atual utilizado é o Business Process Execution Language for Web Services (BPEL4WS ou BPEL). Para obter mais informações, consulte o site OASIS WSBPEL (endereço relacionado no quadro **Links**).

Em alguns casos será necessário algum tipo de workflow em nível de integração

de serviços (Lógica de conectividade). Isto não deve ser confundido com o workflow de processos de negócios. Os ESB's provêem formas de realizar orquestração para a lógica de conectividade.

Sistemas de informação em geral possuem a lógica de processo de negócio e a lógica de integração misturada com a regra do negócio. Um grande desafio na abordagem SOA é dividir corretamente a lógica da aplicação, como na **Figura 3**.

TI alinhada ao negócio

Existem autores que classificam serviços em "de negócio" ou "de infra-estrutura", entre outras classificações possíveis. No caso dos serviços "de negócio", o nível de granularidade dos serviços é tal que, as operações fornecidas pelos serviços podem ser identificadas nos processos de negócio, formando um vocabulário comum, facilitando a comunicação da TI com as áreas de negócio. Portanto, os usuários do negócio de soluções de TI tornam-se parte do processo de design e análise.

Também é interessante observar que esta conexão mais próxima com o modelo de processo de negócio associa mais diretamente os investimentos em TI às necessidades do negócio. Em outras palavras, um serviço baseado no processo de negócio pode ser usado para endereçar o orçamento de TI, justificando desta forma os custos de TI correlacionados aos resultados dos processos de negócio. Sem uma aliança entre as áreas de negócio e a TI, SOA não irá obter o sucesso esperado. SOA deveria ser exigido pelo negócio, não somente pela TI.

Componentes que realizam os serviços

A adoção do SOA não revogou nada do CBD, pelo contrário, representou uma evolução deste. Isto porque o CBD trouxe a base essencial para que fosse possível suportar SOA, ou seja, a infra-estrutura fundamental para a implementação de serviços. A diferença é que, com CBD, têm-se na interface dos componentes todas as operações necessárias para "realizar" os passos dos UCs (Casos de uso). Os componentes são identificados decompondo-se o modelo de classe conceitual (de análise) observando os objetos principais (conceitos mais fortes) e o tipo de associação (agregação, herança...) entre os objetos relacionados; já com SOA, há serviços apenas se observados na perspectiva do negócio. Em outras palavras, a existência de um serviço só faz sentido se o mesmo estiver associado a uma atividade no processo de negócio (**Figura 4**).

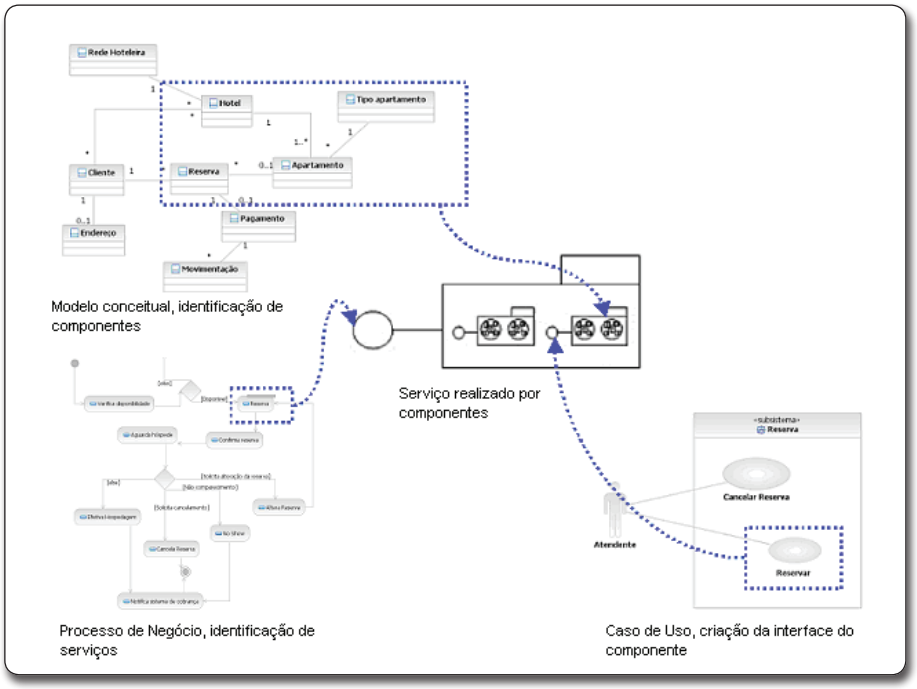


Figura 4. Visão simplificada de identificação de serviços e componentes.

Como SOA trata do alinhamento da TI com os processos de negócio da empresa, os serviços devem ser rastreáveis. Pode-se identificar sua origem no negócio, ou seja, tudo começa nos processos de negócio da empresa. Por isso, normalmente temos uma granularidade grossa nos serviços. Não existe uma fórmula e conseqüentemente uma granularidade padrão, ou seja, a granularidade de serviços é definida em função das necessidades do negócio. As técnicas para encontrar serviços partem sempre da análise do processo. Não existe uma relação direta de uma atividade do processo com um serviço (1 para 1), mas ele sempre está relacionado, no final da análise do negócio (processo), a uma atividade do workflow. Os serviços encontrados com essa abordagem podem ser "realizados" a partir do uso de componentes de negócio.

Todavia, nem sempre as interfaces de componentes são muito granulares. Na verdade, para a interface dos componentes de negócio, podemos considerar os casos de uso (UC). UC são soluções para os requisitos funcionais, que produzem um resultado observável para um ator. Na descrição dos casos de uso temos as interações ator/sistema. Analisando essas interações, encontramos, por exemplo, a necessidade de exibir uma simples lista para uma "caixa de seleção" (alta granularidade) ou, até mesmo, coisas do tipo "aprovar um empréstimo" (baixa granularidade). Já com SOA, na maioria dos caos, temos uma baixa granularidade, já que a interface é identificada na perspectiva do negócio (processos de negócio), envolvendo uma análise do mesmo.

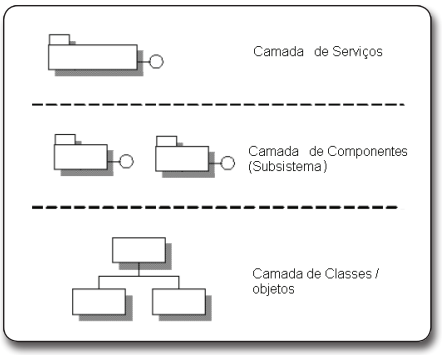


Figura 5. Camadas de uma implementação SOA

Para encontrar serviços, geralmente considera-se que as operações em uma especificação de serviço corresponderão com as tarefas atômicas identificadas em um modelo de processo de negócios. Caso feita cuidadosamente, esta abordagem pode atingir os resultados espera-

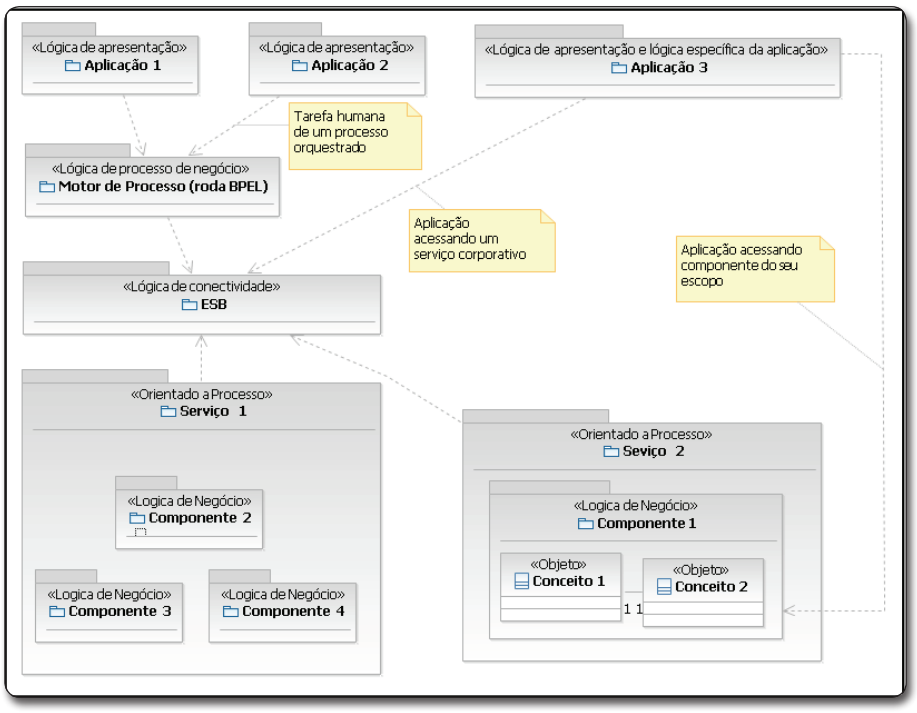


Figura 6. Abstração em pacotes de SOA

dos, mas o nível de detalhe geralmente obtido por esse tipo de abordagem não é suficiente para garantir serviços consistentes. Para tanto, é interessante usar um método de análise de processos de negócio. Os Casos de Usos de Negócios, aliados a uma análise de negócio, podem contribuir para a obtenção de serviços com a granularidade correta.

Depois que os serviços são identificados (serviços candidatos), é necessário determinar quais devem ser expostos como serviços. Os serviços que fazem parte do processo de negócio que está sendo automatizado, serão implementados, mas não necessariamente exigirão a geração de WSDL ou de outras formas de exposição. Serão tratados como componentes da aplicação, evitando problemas de desempenho e gerenciamento de serviço. Portanto, são necessários alguns critérios para ajudar a decidir se um serviço deve ser exposto, como relevância para o negócio, possibilidade de reutilização e viabilidade técnica.

Os componentes são a melhor abordagem para implementar serviços, embora se deva entender que um aplicativo baseado em componente bem projetado não é necessariamente uma abordagem SOA. Conforme mostrado, temos que considerar muitas questões para melhor atender as mudanças no negócio, que é o objetivo. Uma abordagem orientada a serviços implica em uma camada de arquitetura de aplicativo adicional. A **Figura 5** demonstra como as camadas de tecnologia podem ser aplicadas à arquitetura.

Camada de Serviços: A camada de serviços é caracterizada por serviços que realizam funções individuais de um negócio. Serviços são compostos de Contrato, Interface, Componentes de Negócio (Lógica de Negócios) e são identificados no processo de negócio.

Camada de Componentes de negócio (Subsistemas): Um componente de negócio é uma parte de um sistema que encapsula as regras de negócio e as expõe através de interfaces bem definidas. Componentes são independentes, substituíveis e modulares, eles ajudam a gerenciar a complexidade e encorajam a reutilização. Estes são identificados decompondo-se o modelo de classe conceitual (de análise).

Camada de Classes e Objetos: Na camada de Classes e objetos é onde estão as classes, atributos e relacionamentos de um objeto. Os objetos colaboram entre si para realizar as regras de negócio de um componente específico.

Por fim, o exemplo da **Figura 6** mostra que SOA absorve o projeto baseado em componentes, ou seja, o serviço vai ser implementado por um ou mais componentes. ●

Links

The Open Group, Service Oriented Architecture
http://www.opengroup.org/projects/soa/

IASA
http://www.iasahome.org/web/home/home

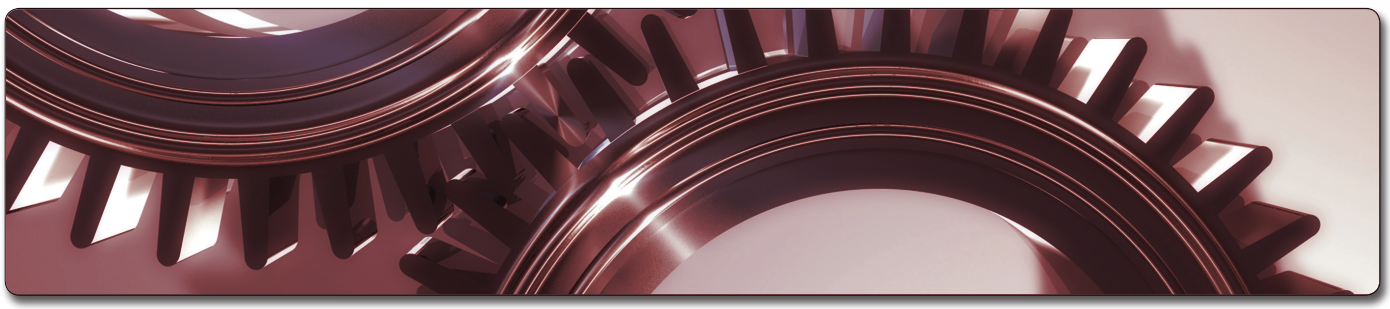
Artigo "Service Oriented Ambiguity"
http://www.martinfowler.com/bliki/ServiceOrientedAmbiguity.html

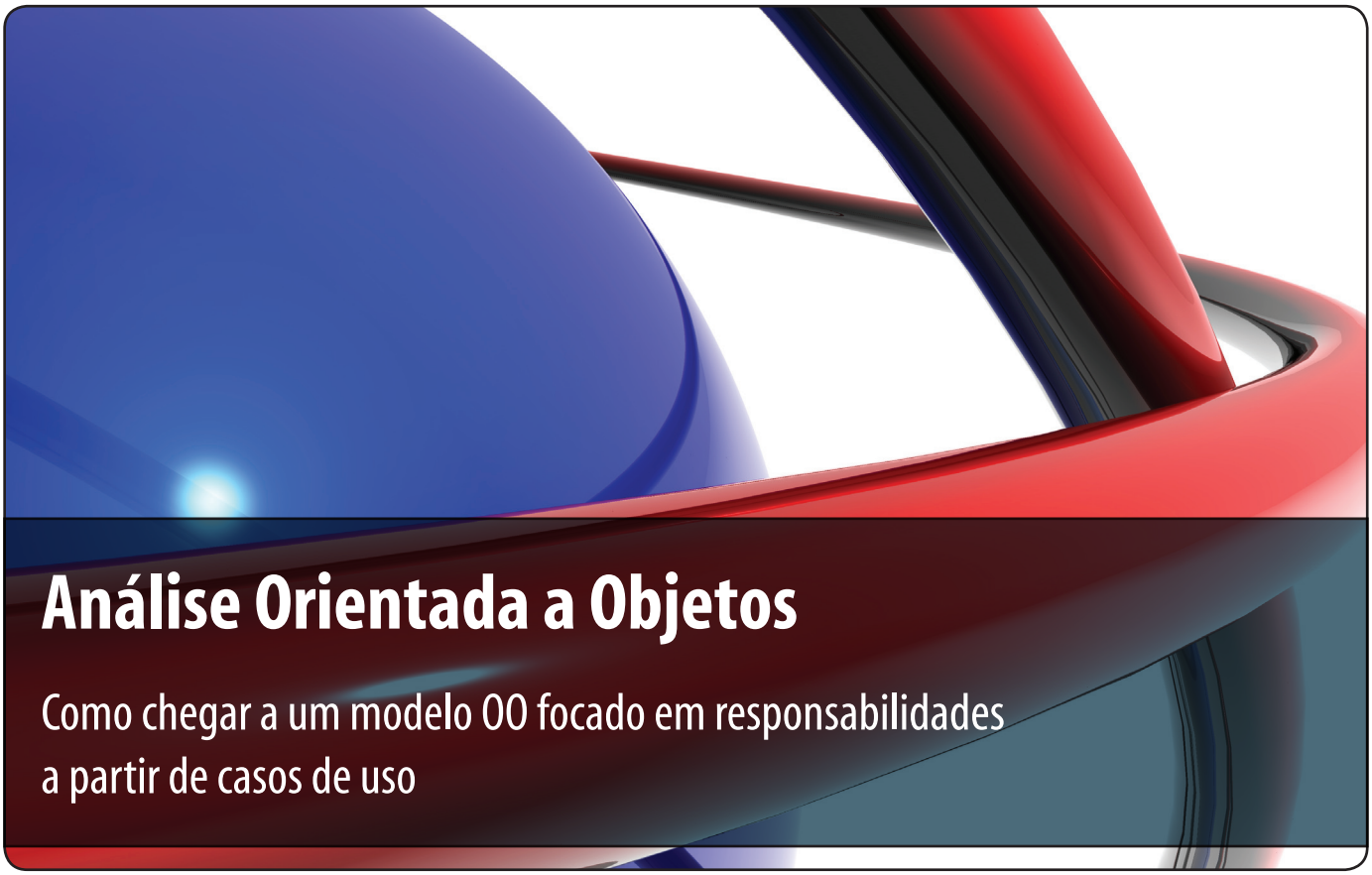
Service-oriented modeling and architecture
http://www.ibm.com/developerworks/webservices/library/ws-soa-design1/

OASIS WSBPEL
http://www.oasis-open.org/home/index.php

Livro **"Fundamentals of Object-Oriented Design in UML"** by *Meilir Page-Jones*

Livro **"Service-Oriented Architecture (SOA): Concepts, Technology, and Design"** *Thomas Erl - Prentice Hall*





Análise Orientada a Objetos

Como chegar a um modelo OO focado em responsabilidades a partir de casos de uso

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

O conceito de orientação a objetos surgiu com o intuito de minimizar os problemas encontrados até então na criação de softwares complexos, projetados por meio de decomposição funcional e sub-rotinas.

Podemos identificar como um dos maiores problemas a não existência de encapsulamento lógico para operações e dados, o que leva a não existência da divisão de tarefas por responsabilidades. O que leva a construção de longos trechos de código, muitas vezes difíceis de compreender devido ao acúmulo de responsabilidade que lhe é atribuído.

Por conseqüência, quanto mais complexo o software se torna, mais difícil se torna também a sua manutenção. Com isso aumentam os custos e o risco de confiabilidade do mesmo.

Nesse artigo veremos como efetuar uma analise orientada a objetos, com base em responsabilidade, extraída a partir das descrições de casos de uso.

Conceituando análise Orientada a Objetos

O foco da análise OO é no mapeamento de uma solução sistêmica para algum processo de negocio.

No inicio da análise OO, elaboramos os casos de uso. Estes juntamente com as descrições dos casos de uso formam uma espécie de ponte funcional entre o processo de negocio e a solução de software a ser produzida.

Outros dois documentos podem ser usados como fontes complementares aos casos de uso na análise OO:

- Glossário: onde estão definidos os significados de todos os termos inerentes ao negócio mapeado nos casos de uso;
- Documento de arquitetura: na utilização de possíveis mecanismos de análise, que são padrões de comportamento ou estrutura de uso recorrente e comprovadamente válido. Iremos falar mais sobre eles mais adiante.

Normalmente, o resultado da análise orientada a objetos se traduz em diagramas UML de seqüência e classe, como mostra a **Figura 1**. Entretanto, outros diagramas UML podem ser usados nessa tarefa, desde que seja verificado algum ganho no entendimento ou mapeamento da solução com seu uso.

Através da **Figura 1** também percebemos que os estereótipos utilizados no diagrama de classe não são os que costumamos ver normalmente. Pois bem, ali estamos usando os estereótipos de análise.

O uso desses estereótipos nos oferece uma orientação mais específica para o processo de identificação de classes.

Os estereótipos de análise se dividem em: classe de fronteira (ou boundary), classe de controle (ou control) e classe de entidade (ou entity), como mostra a **Figura 2**.

A classe de fronteira é responsável por modelar a interação entre o ambiente do sistema e seus trabalhos internos. Essa interação envolve transformar e converter eventos, bem como observar mudanças na apresentação do sistema. É a classe de fronteira que trata das questões relativas à camada mais externa do sistema.

A classe de controle é usada para modelar um comportamento de controle específico de um ou alguns casos de uso. Geralmente estão controlando chamadas a classes de entidade. Por isso seu comportamento é muito ligado à idéia de coordenação, de ponte entre a camada mais externa do sistema (classes de fronteira) e a camada mais interna do sistema (classes de entidade).

E finalmente, com a classe de entidade, modelamos comportamentos e informações que devem ser armazenados. É de responsabilidade das classes de entidade manter e atualizar informações relativas ao negócio do sistema, como: pessoas, eventos, objetos reais, ou qualquer outra informação ligada ao negocio ao qual o sistema está inserido. Por exemplo: em um sistema voltado para a área de ensino, provavelmente teremos uma classe de entidade chamada aluno, outra chamada disciplina e assim por diante.

Efetuando a análise

Extrair classes orientadas a objeto com base em descrições de casos de uso não é uma tarefa simples. Para isso, é necessário ter uma boa capacidade de abstração. Entretanto, se dividirmos esse trabalho em etapas consecutivas e complementares, o processo de análise como um todo se torna mais simples.

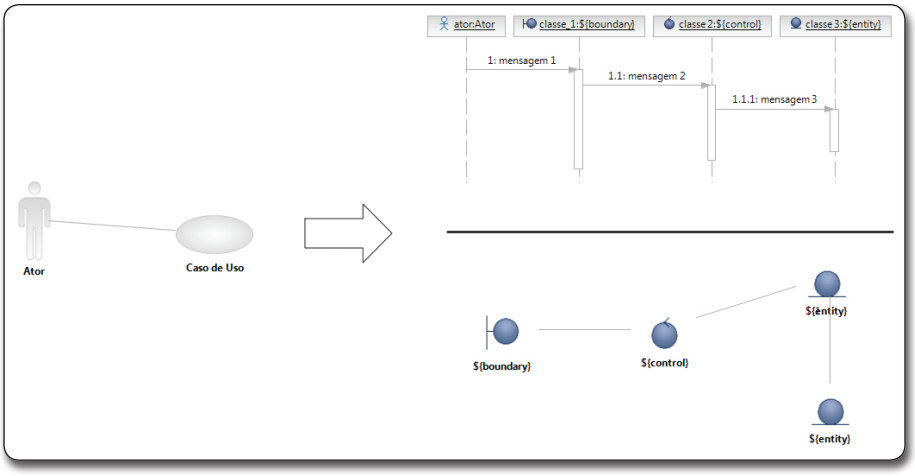


Figura 1. Produtos da análise.

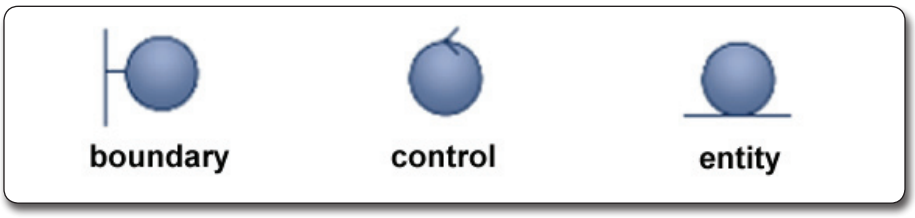


Figura 2. Estereotípos de análise

Sendo assim, nossa análise OO será construída ao longo dos quatro tópicos a seguir.

Identificando Abstrações Chave

A partir da leitura e entendimento dos casos de uso e do glossário, devemos identificar quais serão os conceitos mais importantes e óbvios do sistema. Esses conceitos são chamados de abstrações chave. Eles devem ser acompanhados de uma breve descrição e dos possíveis relacionamentos entre eles.

O objetivo em se identificar essas abstrações é justamente dar um guia, uma referência primária para toda a análise que virá pela frente.

Vale ressaltar que as abstrações chave encontradas não devem ser vistas como verdades absolutas e imutáveis. Com o avanço da análise do sistema, e conseqüentemente, com o entendimento mais amplo sobre a solução OO que está sendo desenvolvida, as abstrações podem sofrer alterações.

Criando diagramas de seqüência com base em responsabilidade

Para cada cenário do caso de uso (fluxo principal e fluxos alternativos) devemos executar uma releitura dos passos do cenário. À medida que vamos avançando na leitura, devemos ir identificando as classes de fronteira, controle e entidade envolvidas no texto.

Nesse ponto do trabalho, a identificação de abstrações chave feita anteriormente se torna bastante útil, pois, iremos utilizá-las como base para a identificação das classes de entidade. Tanto as abstrações chave quanto as classes de entidade estão intimamente ligadas aos conceitos de negócio envolvidos no caso de uso. As abstrações chave nos dão uma primeira visão sobre o negocio, uma visão um pouco turva. Começamos a dar mais nitidez a essa visão quando transformamos as abstrações chave em classes de entidade.

A coisa mais importante nesse momento é despender total atenção para a correta divisão de responsabilidade entre as classes encontradas.

A responsabilidade de uma classe representa o conjunto de ações que ela pode desempenhar e o conjunto de informações que ela pode ser solicitada a fornecer.

E porque iremos fazer um diagrama de seqüência com essas classes e não um diagrama de classe?

Porque à medida que vamos lendo gradualmente os passos do caso de uso podemos identificar não só as classes,

mas também a interação entre o ator do caso de uso e as classes.

A princípio, essa abordagem pode parecer um pouco complexa demais. Afinal, normalmente aprendemos a fazer um diagrama de classe com base no caso de uso e só depois passamos para algum diagrama dinâmico, como o de seqüência.

Mas na verdade, começar pelo diagrama de seqüência e não pelo de classe é uma forma bastante natural de se trabalhar. Pois, enquanto estamos lendo o caso de uso e visualizando as interações entre ator e sistema, nada nos impede de já transformarmos essas interações em mensagens trocadas pelas classes do diagrama de seqüência.

Trabalhando dessa forma, as classes vão aparecendo gradualmente no diagrama de seqüência, em decorrência da leitura do passo do caso de uso e sua interpretação como um conjunto de mensagens trocadas entre classes.

Ao final teremos um ou mais diagramas de seqüência de análise para cada cenário descrito no caso de uso. Vale lembrar que se um cenário oferece uma alta complexidade de entendimento ou é longo de mais, podemos criar mais de um diagrama de seqüência focado nele.

Muito provavelmente, teremos também um diagrama de classe de todo o caso de uso. Afinal de contas, na maioria das ferramentas para diagramação UML hoje em dia, quando geramos diagramas de seqüência estamos automaticamente criando um digrama de classe contemplando as classes usadas no diagrama de seqüência e os relacionamentos entre elas.

Preenchendo e refinando as classes

Agora que já criamos diagramas de seqüência para todos os cenários do caso de uso, chegou a hora de arrumar o diagrama de classe resultante desse trabalho e também o de seqüência, se for o caso.

Como tratamos cada cenário de forma isolada, é possível que tenhamos criado classes com responsabilidades muito parecidas, porém, em diagramas de seqüência diferentes. Nesse momento devemos fazer uma varredura no diagrama de classes, objetivando eliminar possíveis redundâncias.

A partir daí passamos a analisar os vínculos entre classes, que são as mensagens trocadas entre elas nos diagramas de seqüência. Essas mensagens representam a comunicação entre as classes, o relacionamento entre elas. O refinamento aqui é justamente identificar corretamente qual o tipo de relacionamento existente entre duas classes do diagrama de classes. É imprescindível que tenhamos um bom entendimento do caso de uso, pois sem isso, seria difícil, por exemplo, identificar se determinado relacionamento deve ser uma agregação ou uma composição.

Também podemos identificar os atributos das classes com base nas mensagens trocadas no diagrama de seqüência quando a mensagem é criada com o intuito de trafegar informações entre classes. Nesse caso, essas informações se tornam atributos da classe responsável por fornecê-las. Mas tenhamos cuidado com isso! Pois se uma classe A precisa acessar uma classe B para fornecer uma informação, então provavelmente a responsabilidade de guardar essa informação na forma de atributo, é da classe B e não da classe A.

Aliás, desconfie quando as classes de entidade não trocam mensagens entrei si para resolver passos do caso de uso no diagrama de seqüência. Isso **pode** significar que a divisão de responsabilidade entre as classes não foi bem feita, como mostra a **Figura 3**.

Uma boa divisão de responsabilidade entre classes de negócio (entidades) normalmente se traduz em diagramas como os da **Figura 4**.

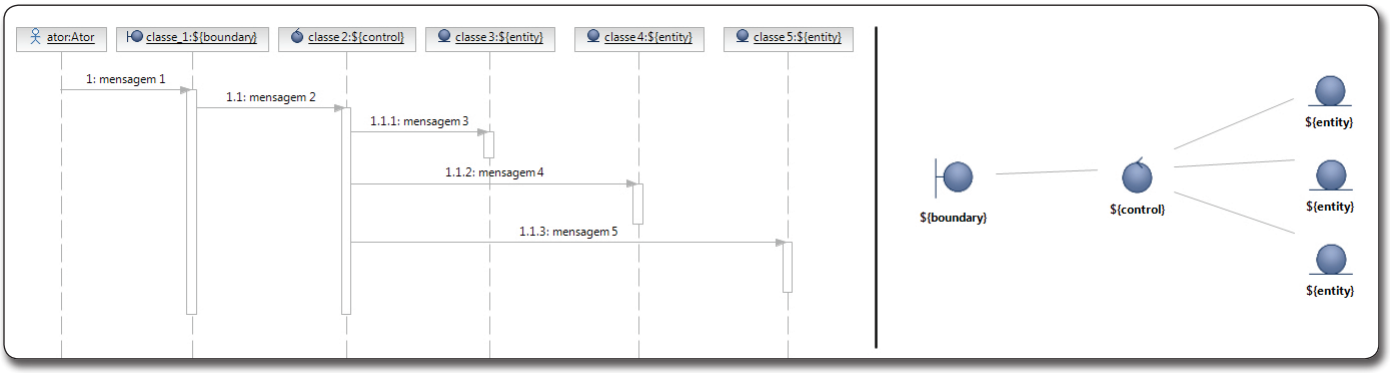


Figura 3. Divisão de responsabilidade provavelmente mal feita.

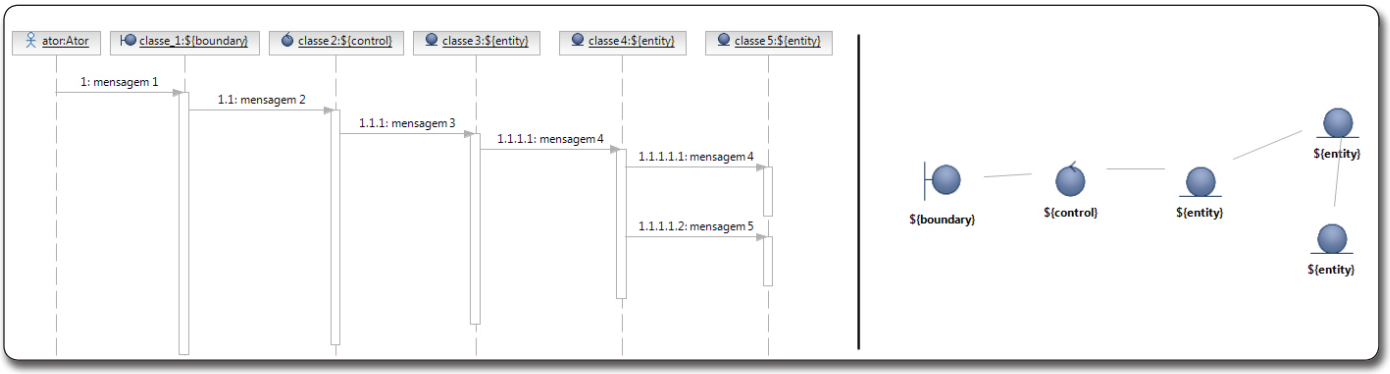


Figura 4. Divisão de responsabilidade bem feita.

Observe que na **Figura 3** a classe de controle chamada *Classe2* se responsabiliza pela comunicação com as classes de entidade 3, 4 e 5, as quais não trocam mensagens entre si. Agora dando nome aos bois, imagine que a classe 3 se chama *Pedido*, a classe 4 *ItemPedido* e a classe 5 *Produto*.

Com essa visão de negócio em mente, chegaríamos facilmente à seguinte constatação: s**omente o Pedido tem conhecimento de quais são os Itens de Pedido que o compõe.**

Após essa constatação, teria sentido pedir para a classe de controle se responsabilizar diretamente pelas informações referentes à *IntemPedido*, por exemplo?

De forma alguma!

O mais coerente, de acordo com a correta divisão de responsabilidade seria somente a classe Pedido ter o direito de se comunicar com o *ItemPedido*. Então, quando alguma classe precisar de informações sobre *ItemPedido*, ela terá de pedir a informação para a classe Pedido.

Um diagrama de seqüência de análise feito com essa mentalidade baseada em divisão de responsabilidades normalmente se aproxima da forma exposta na **Figura 4**, onde as classes de entidade costumam se relacionar com foco na responsabilidade.

Padrões de análise

No início do artigo, o documento de arquitetura foi apresentado como um dos documentos que servem de insumo para se fazer a análise OO devido aos mecanismos de análise que ele pode apresentar. Pois bem, existem mecanismos de análise focados em divisão de responsabilidade. Eles foram criados de forma bem genérica, para que seu uso seja válido e aconselhável em praticamente qualquer análise OO. Eles são chamados de padrões GRASP (General Responsibility Assignment Software Patterns). Os padrões GRASP representam as boas práticas que podemos usar para atribuir responsabilidades às classes. São eles:

Especialista (Expert)

É o padrão mais usado na atribuição de responsabilidade. Ele determina quem é o especialista em determinada informação. A classe especialista em determinada informação é a classe que tem a informação necessária para satisfazer tal responsabilidade ou que sabe como obter tal informação. Por exemplo: em um sistema de vendas onde existem as classes *Venda* e *ItemVenda*, sendo que *Venda* representa uma composição de *ItemVenda*, qual das duas classes seria a responsável por manter a informação total de uma venda?

De acordo com o padrão especialista, a classe responsável por manter o total de uma venda seria a classe *Venda*, pois a *Venda* é que detém o conhecimento sobre todos os seus *ItemVenda*. Conseqüentemente, é nela que devemos manter o somatório dos valores de seus *ItemVenda*.

A classe *ItemVenda* não deve ter a responsabilidade de guardar o total de uma venda. Afinal, um *ItemVenda* só conhece o valor do seu item e não de todos os itens de uma venda (ver **Figura 5**).

A atribuição de responsabilidades muitas vezes não tem correspondente no mundo real. Nesse caso da venda, por exemplo, no mundo real uma venda não calcula seu próprio total, isso seria feito por uma pessoa. No mundo OO, entidades inertes, como produtos, ou conceitos, como uma venda, podem ter responsabilidades.

Criador (Creator)

Padrão que define qual classe deve ser responsável por criar instâncias de outra classe. A classe criadora deve ser aquela que possui a outra como parte dela ou que esteja fortemente associada a ela. Sendo assim, atribuímos à classe B a responsabilidade de criar uma instância da classe A se uma das seguintes condições for verdadeira:

- B agrega objetos A
- B contém objetos A
- B registra instâncias de objetos A
- B usa de maneira muito próxima objetos de A
- B é um especialista com relação à criação de A

Controlador (Controller)

Padrão que define a classe responsável por tratar um acontecimento externo ao sistema e que desencadeia alguma ação do mesmo. É o padrão responsável por tratar eventos do sistema.

Podemos atribuir essa responsabilidade de duas formas. Uma seria criando uma classe que vá tratar todos os eventos de um caso de uso, nesse caso, estamos criando um "controlador de caso de uso". A outra seria criando uma classe que vá controlar todo o sistema, nesse caso, estamos criando um "controlador de fachada".

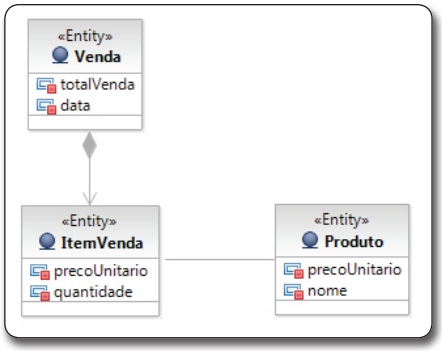


Figura 5. Usando o padrão especialista.

Lembra do estereótipo "control" apresentado há pouco? Pois é, qualquer semelhança com o padrão controlador, não é mera coincidência.

Já que voltamos a falar dos estereótipos de análise, vale salientar que as classes de fronteira (ou boundary) não devem ter a responsabilidade de manipular eventos do sistema, elas devem apenas delegar essa tarefa para o controlador. Dessa forma, temos uma maior capacidade de reuso, pois estaremos criando interfaces "plugáveis" para o sistema. Afinal, se a camada do sistema que faz interface com o usuário (classes de fronteira) não se responsabiliza por resolver os eventos gerados, então ela pode ser trocada facilmente. Ou seja, ela oferece um baixo nível de acoplamento, que alias, é o nome de mais um padrão GRASP.

Baixo acoplamento (Low Coupling)

Acoplamento é a medida de quão fortemente uma classe está conectada a outras classes. Essa conexão se mostra em qualquer tipo de relacionamento entre classes, como dependência ou herança, por exemplo. Uma classe com acoplamento baixo (ou fraco) não depende de muitas outras. Já uma com acoplamento alto (ou forte) é mais difícil de compreender isoladamente, mais difícil de reutilizar (seu uso depende da reutilização das outras classes da qual ela depende) e também mais sensível a mudanças nas classes associadas a ela.

Sendo assim, na modelagem de interações, sempre que for possível, evitamos a criação de mensagens que impliquem em associações redundantes no modelo de classes.

Coesão Alta (High Coesion)

Coesão é a medida do quão fortemente relacionadas e focalizadas são as responsabilidades de uma classe. Uma classe com baixa coesão acaba se responsabilizando por mais coisas do que ela realmente deveria ser responsável. Classes assim acabam oferecendo dificuldade para o seu entendimento e manutenção, além de serem mais difíceis de reutilizar, por não terem um foco bem definido.

Exemplo de execução da análise OO

Para exemplificar de forma rápida o processo de análise OO apresentado até aqui, vamos usar um cenário de caso de uso bastante simplificado, ligado a área de negócio financeira, como mostra a **Figura 6**.

Caso de Uso: Consultar Saldo

Descrição: Este caso de uso permite fazer uma consulta simples de saldo em conta bancária.

Ator Principal: Cliente

Pré-Condições: O Cliente deve estar devidamente identificado no sistema, como correntista do banco.

Fluxo de Eventos Básico – Consultar Saldo:

1. Cliente informa agência e conta;
2. Cliente solicita saldo;
3. Sistema solicita senha;
4. Cliente informa senha;
5. Sistema informa nome do cliente, agencia, conta e saldo atual.

Fluxos de Eventos Alternativos: N/A (não se aplica)

Pós-Condições: N/A (não se aplica)

Regras: Caso a conta corrente do cliente esteja vinculada a uma conta poupança, o saldo atual deve apresentar os valores disponíveis nas duas contas.

Fluxos de Exceção das Regras: N/A (não se aplica)

Mensagens dos Fluxos de Exceção: N/A (não se aplica)

Analisando o caso de uso, identificamos as seguintes abstrações chave:

- **Cliente** - pessoa ou empresa que utiliza os serviços do banco;
- **Conta** - representação dos valores monetários do cliente no banco;
- **Agência** - representação de um conjunto de contas.

Figura 6. Caso de Uso Consulta Saldo.

Com base nos passos do caso de uso e usando os estereótipos de análise, chegamos ao diagrama de seqüência da **Figura 7** e por conseqüência, ao digrama de classe da **Figura 8.**

Observando esses dois diagramas e lembrando dos conceitos sobre estereótipos de análise, podemos deduzir que:

- Criamos a classe *ConsultaSaldo*, de estereótipo boundary, para que ela sirva de elo entre o usuário do caso de uso e o sistema, responsabilizando-se pela interação entre os dois;
- Criamos a classe *ContaControl*, de estereótipo control, para coordenar as chamadas a classes de entidade, que esse caso de uso necessite;
- Criamos as classes *Agencia* e *Conta*, de estereótipo entity, para armazenar as informações do negócio necessárias ao cumprimento dos passos do caso de uso.

E finalmente, utilizando os padrões de análise, chegamos a um refinamento nas responsabilidades das classes e também a uma identificação mais precisa dos atributos e operações envolvidos. É muito comum esse refinamento nos levar a descoberta de muitas novas classes, atributos e operações, quando estamos sendo iniciados na análise OO, como mostra o diagrama de seqüência da **Figura 9** e o diagrama de classe da **Figura 10**.

Vejamos então, como as classes e as

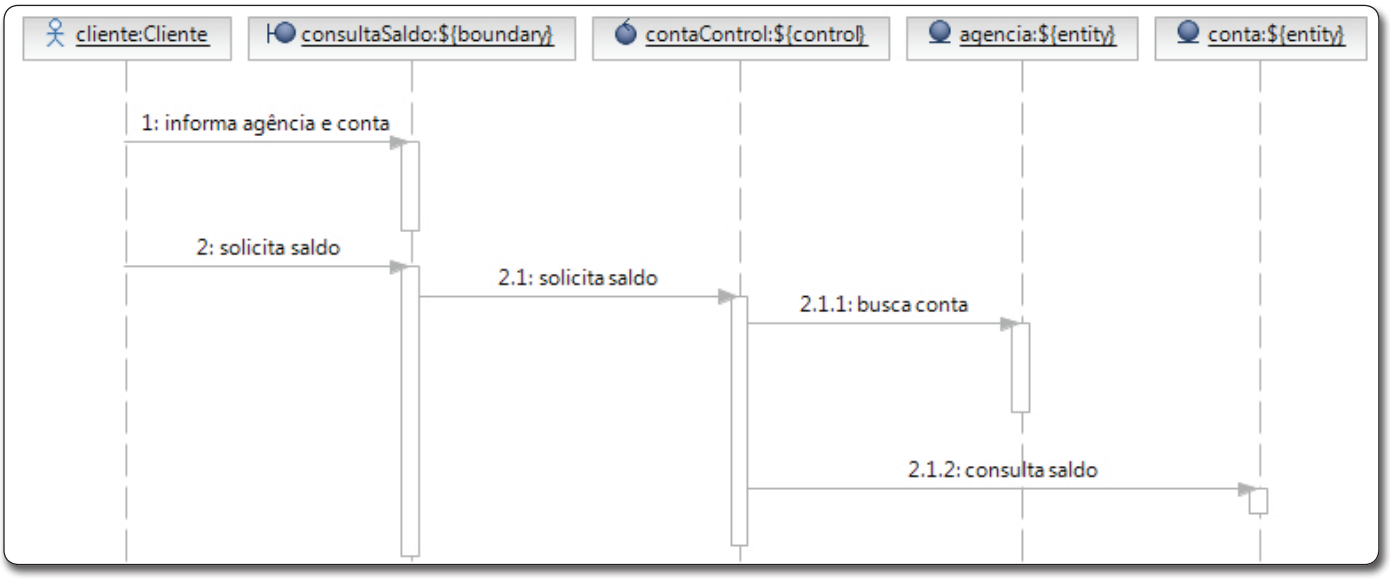


Figura 7. Diagrama de Seqüência Consulta Saldo.

mensagens trocadas entre elas mudaram depois do refinamento.

Inicialmente a classe *ContaControl* tinha dependência direta para as classes *Agencia* e *Conta*. Ela se responsabilizava por buscar a conta pedindo-a para a classe *Agencia* e por buscar diretamente o saldo contido na classe *Conta*.

Pelo princípio do padrão **Especialista**, quem deveria se responsabilizar pelo acesso às informações da classe *Conta* é a classe *Agencia*. Afinal, a agência é dona de suas contas e essas contas não existem sem sua agência. Sendo assim, criamos uma composição entre as classes *Agencia* e *Conta* e também eliminamos o acesso direto da classe *ContaControl* a classe *Conta*.

Após essas modificações, diminuímos o acoplamento entre as classes, visto que *ContaControl* não está mais acoplada diretamente a *Conta*. Então usamos também o padrão **Baixo Acoplamento**.

Analisando o caso de uso e as abstrações chave, percebemos algumas coisas, como:

- De acordo com a única regra do caso de uso, a conta pode ser corrente ou poupança;
- A definição da abstração chave Cliente mostra o cliente como uma pessoa ou uma empresa.

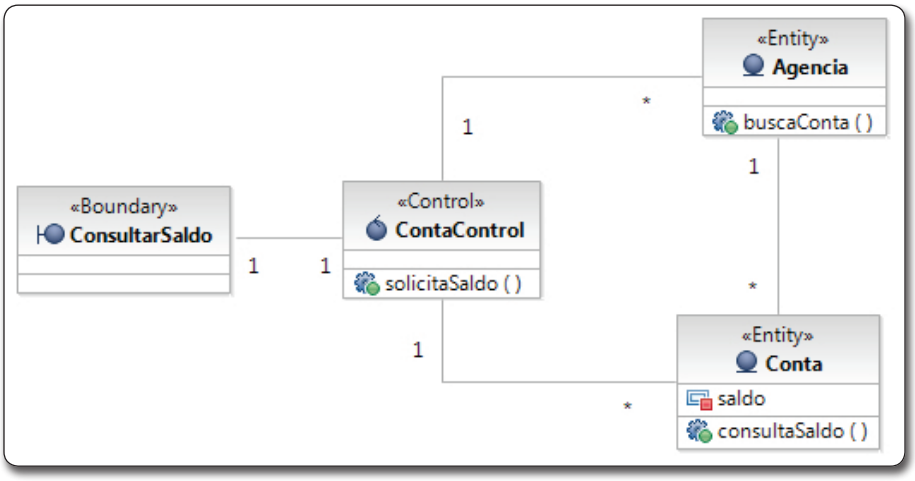


Figura 8. Diagrama de Classe Consulta Saldo.

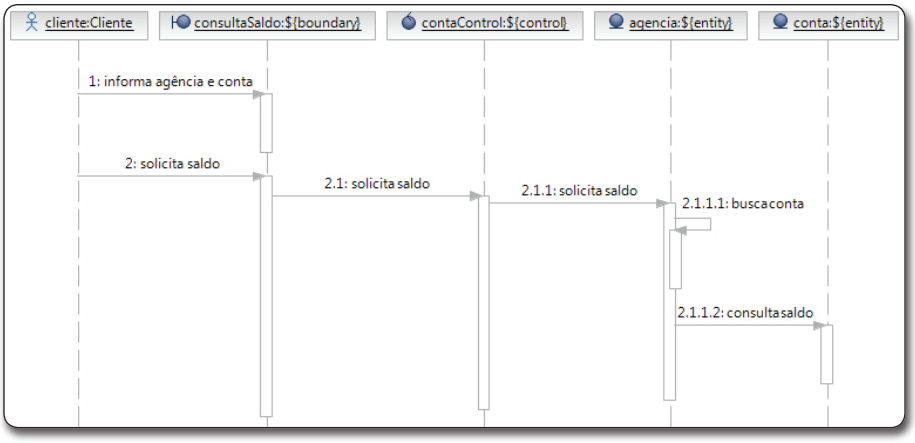


Figura 9. Diagrama de Seqüência Consulta Saldo – Após refinamento.

Batendo essas informações com o diagrama de classe, vemos que precisaremos melhorar duas coisas: a coesão da classe *Conta* e tratar o conceito de negócio cliente.

A classe *Conta* está se responsabilizando tanto pelas particularidades da conta corrente quanto pelas da conta poupança. Para melhorar sua coesão e assim respeitar o padrão **Coesão Alta**, fizemos o seguinte:

- Tornamos a classe *Conta* abstrata, responsabilizando-a apenas pelo comportamento e informações que forem comuns tanto a conta corrente quanto a conta poupança;
- Criamos as classes concretas *ContaCorrente* e *ContaPoupança*, filhas de *Conta*, para tratar de questões específicas sobre conta corrente e conta poupança.

Analogamente, percebemos que criar apenas uma classe para cliente a levaria ao mesmo problema de coesão baixa. Então ela recebeu o mesmo tratamento que a classe Conta no diagrama de classe.

Com a experiência no uso dos conhecimentos apresentados nesse artigo, conseguimos definir de forma correta boa parte das responsabilidades de cada classe. Nesse nível de maturidade profissional, a etapa de refinamento acaba sendo mais usada para a eliminação de possíveis redundâncias entre os modelos de classe feitos para cada cenário do caso de uso.

Conclusão

Efetuar uma boa análise OO na criação de um sistema é fundamental para o seu sucesso. Por incrível que pareça, muita gente ainda enxerga a análise OO apenas como uma forma de documentar um sistema. Sim, esse é um dos ganhos que se tem com ela. Afinal, com seu uso, o conhecimento sobre o sistema não fica preso apenas na cabeça de quem o construiu, como também não fica submerso em meio a milhares de linhas de código fonte. Isso facilita a reutilização, o entendimento e a manutenção do software. Mas o benefício da documentação não é o único ao se utilizar a análise OO.

A maior vantagem em utilizá-la é justamente poder pensar em uma solução sistêmica focada em responsabilidades, antes de começar a construí-la em linguagens de programação.

É claro que para o sistema que vai controlar seu estoque pessoal de DVDs, a análise OO não se faz tão necessária assim. Mas na medida em que vamos aumentando o nível de complexidade e o tamanho das soluções de software, mais a análise OO se apresenta como uma grande aliada.

Quanto maior a complexidade de um software, maior a necessidade de:

- Ter disponível uma visão mais contemplativa possível sobre os problemas que se quer resolver, para assim achar os melhores caminhos a seguir;
- Poder traçar estratégias isoladas ou abrangentes;
- Poder identificar de quem é a responsabilidade sobre determinado conceito.

Unindo UML, estereótipos de análise e padrões de divisão de responsabilidade, a Análise Orientada a Objetos se encaixa como uma boa alternativa para a solução destas necessidades. ●

Links

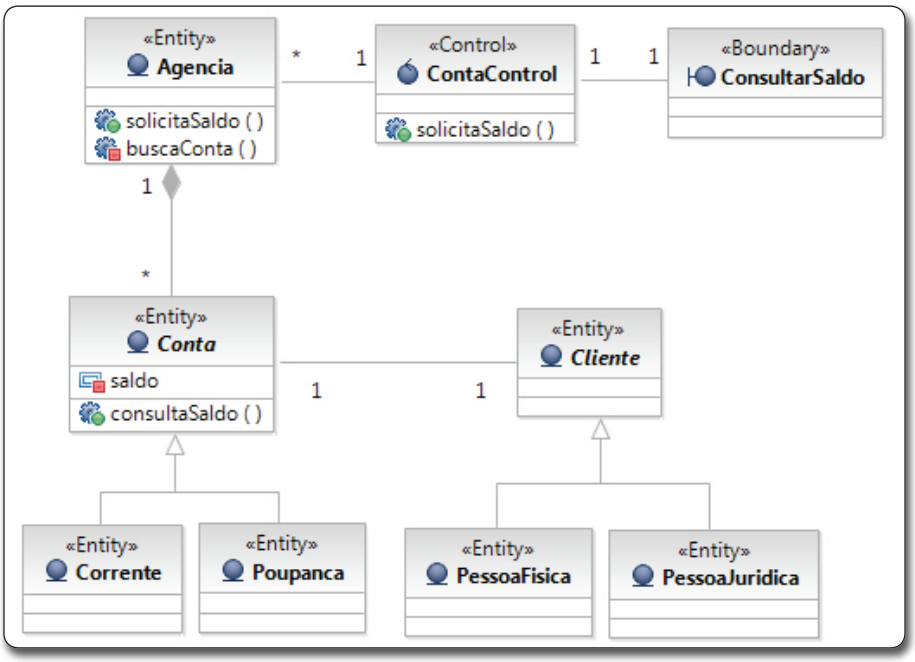
OMG - The Object Management Group
www.omg.org

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

Bibliografia

UML Essencial
Martin Fowler, Editora Bookman

Utilizando UML e Padrões
Craig Larman, Editora Bookman



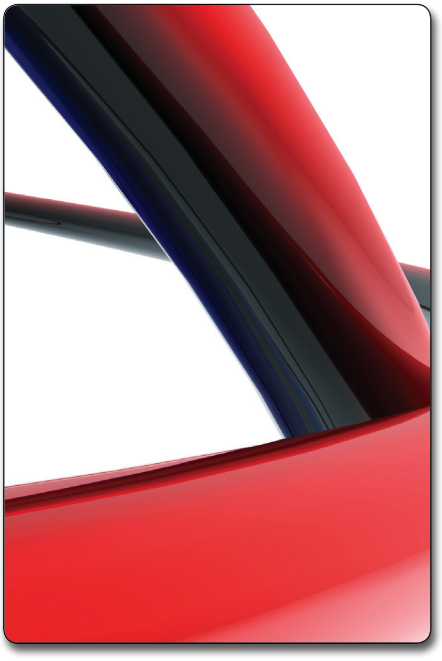


Figura 10. Diagrama de Classe Consulta Saldo – Após refinamento.

Introdução a Abordagens de Identificação de Requisitos

Janaína Bedani Dixon Moraes

jana_dixon2001@yahoo.com.br

É especialista em Concepção e Gerência de Sistemas Orientado a Objeto pela UNIRONDON. Tem exercido a atividade de analista de sistemas desde 2002 e prestado serviço a diversos órgãos do Estado de Mato Grosso. Trabalha atualmente como analista de requisitos e gerente de projeto.

O início para toda a atividade de desenvolvimento de software é o levantamento de requisitos, sendo esta atividade repetida em todas as demais etapas da engenharia de requisitos.

Sommerville (2003) propõe um processo genérico de levantamento e análise que contém as seguintes atividades:

- **Compreensão do domínio:** Os analistas devem desenvolver sua compreensão do domínio da aplicação;
- **Coleta de requisitos:** É o processo de interagir com os *stakeholders* do sistema para descobrir seus requisitos. A compreensão do domínio se desenvolve mais durante essa atividade;
- **Classificação:** Essa atividade considera o conjunto não estruturado dos requisitos e os organiza em grupos coerentes;
- **Resolução de conflitos:** Quando múltiplos *stakeholders* estão envolvidos, os requisitos apresentarão conflitos. Essa atividade tem por objetivo solucionar esses conflitos;
- **Definição das prioridades:** Em qualquer conjunto de requisitos, alguns serão mais importantes do que outros. Esse estágio envolve interação com os *stakeholders* para a definição dos requisitos mais importantes;
- **Verificação de requisitos:** Os requisitos são verificados para descobrir se estão completos e consistentes e se estão em concordância com o que os *stakeholders* desejam do sistema.

O levantamento e análise de requisitos é um processo iterativo, com uma contínua validação de uma atividade para outra, conforme exemplificado pela **Figura 1**.

Dificuldades encontradas

O problema de não saber especificar corretamente o que o sistema deverá fazer é muito antigo. Pompilho (1995) cita um exemplo do relatório produzido por McKinsey, em 1968, e mencionado por B. Langefords e B. Sundgren onde se afirmava que dois terços das empresas

ali estudadas estavam desapontadas com o atendimento recebido em sistemas de informação.

Após mais de 30 anos da elaboração do relatório a situação não é muito diferente. Algumas das razões para o baixo grau de satisfação dos usuários para os sistemas destacam-se:

• Na fase de levantamento de requisitos do projeto, onde não é utilizada uma técnica adequada para extrair os requisitos do sistema;

• A falha do analista em não descrever os requisitos do sistema de modo claro, sem ambigüidades, conciso e consistente com todos os aspectos significativos do sistema proposto.

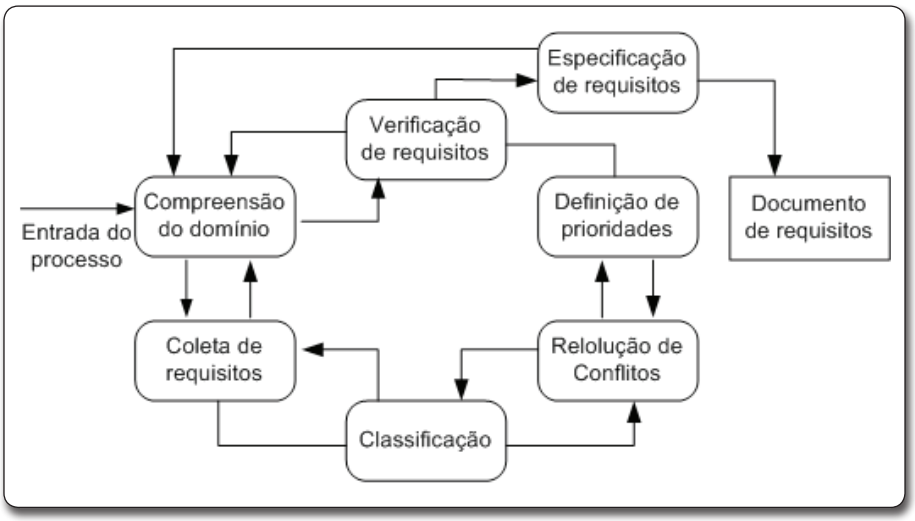


Figura 1. Processo de levantamento e análise de requisitos (SOMMERVILLE, 2003)

Entre as dificuldades encontradas na fase de levantamento de requisitos estão: o usuário principal do sistema não sabe o que quer que o sistema faça ou sabe e não consegue transmitir para o analista; requisitos identificados, mas que não são realistas e não identificam os requisitos similares informados por pessoas diferentes. Um *stakeholder* errado afetará em perda de tempo e dinheiro para ambas as partes envolvidas no desenvolvimento do sistema.

Identifica-se um levantamento de requisitos adequado através da boa definição do projeto, da efetividade do projeto, de informações necessárias a um perfeito diagnóstico e de soluções inteligentes. Quanto ao levantamento de requisitos inadequado, o resultado é um diagnóstico pobre com conclusões comprometidas, não identificação das causas dos problemas, custos elevados, prazos vencidos ou comprometedores, omissão de processos fundamentais e descréditos.

Técnicas de Levantamento de Requisitos

As técnicas de levantamento de requisitos têm por objetivo superar as dificuldades relativas a esta fase. Todas as técnicas possuem um conceito próprio e suas respectivas vantagens e desvantagens, que podem ser utilizadas em conjunto pelo analista.

Serão apresentadas de forma resumida nesse artigo algumas técnicas de levantamento de requisitos.

Levantamento orientado a pontos de vista

Para qualquer sistema, de tamanho médio ou grande, normalmente há diferentes tipos de usuário final. Muitos *stakeholders* têm algum tipo de interesse nos requisitos do sistema. Por esse motivo, mesmo para um sistema relativamente simples, existem muitos pontos de vista diferentes que devem ser considerados. Os diferentes pontos de vista a respeito de um problema 'vêem' o problema de modos diferentes. Contudo, suas perspectivas não são inteiramente independentes, mas em geral apresentam alguma duplicidade, de modo que apresentam requisitos comuns.

As abordagens orientadas a ponto de vista, na engenharia de requisitos, reconhecem esses diferentes pontos de vista e os utilizam para estruturar e organizar o processo de levantamento e os próprios requisitos. Uma importante capacidade da análise orientada a pontos de vista é que ela reconhece a existência de várias perspectivas e oferece um *framework* para descobrir conflitos nos requisitos propostos por diferentes *stakeholders*.

O método VORD (*viewpoint-oriented requirements definition* – definição de requisitos orientada a ponto de vista) foi projetado como um *framework* orientado a serviço para o levantamento e análise de requisitos.

A primeira etapa da análise de ponto de vista é identificar os possíveis pontos de vista. Nessa etapa os analistas se reúnem com os *stakeholders* e utilizam a abordagem de *brainstorming* para identificar os serviços em potencial e as entidades que interagem com o sistema.

A segunda etapa é a estruturação de pontos de vista, que envolve agrupar pontos de vista relacionados, segundo uma hierarquia. Serviços comuns estão localizados nos níveis mais altos da hierarquia e herdados por pontos de vista de nível inferior.

A etapa de documentação do ponto de vista tem por objetivo refinar a descrição dos pontos de vista e serviços identificados.

O mapeamento de sistema conforme ponto de vista envolve identificar objetos em um projeto orientado a objetos, utilizando as informações de serviço que estão encapsuladas nos pontos de vista.

A **Figura 2** exemplifica a técnica de levantamento orientado a ponto de vista.

Etnografia

A etnografia é uma técnica de observação que pode ser utilizada para compreender os requisitos sociais e organizacionais, ou seja, entender a política organizacional bem como a cultura de trabalho com objetivo de familiarizar-se com o sistema e sua história. Os cientistas sociais e antropólogos usam técnicas de observação para desenvolver um entendimento completo e detalhado de culturas particulares.

Nesta técnica, o analista se insere no ambiente de trabalho em que o sistema será utilizado. O trabalho diário é observado e são anotadas as tarefas reais em que o sistema será utilizado. O principal objetivo da etnografia é que ela ajuda a descobrir requisitos de sistema implícitos, que refletem os processos reais, em vez de os processos formais, onde as pessoas estão envolvidas.

Etnografia é particularmente eficaz na descoberta de dois tipos de requisitos:

1. Os requisitos derivados da maneira como as pessoas realmente trabalham, em vez da maneira pelas quais as definições de processo dizem como elas deveriam trabalhar;
2. Os requisitos derivados da cooperação e conscientização das atividades de outras pessoas.

Alguns itens importantes que devem ser executados antes, durante e depois do estudo de observação:

• Antes, é necessário identificar as áreas de usuário a serem observadas; obter a aprovação das gerências apropriadas para executar as observações; obter os nomes e funções das pessoas chave que estão envolvidas no estudo de observação; e explicar a finalidade do estudo;

• Durante, é necessário familiarizar-se com o local de trabalho que está sendo observado. Para isso é preciso observar os agrupamentos organizacionais atuais; as facilidades manuais e automatizadas; coletar amostras de documentos e procedimentos escritos que são usados em cada processo específico que está sendo observado; e acumular informações estatísticas a respeito das tarefas, como: freqüência que ocorrem, estimativas de volumes, tempo de duração para cada pessoa que está sendo observada. Além de observar as operações normais de negócios acima é importante observar as exceções;

• Depois, é necessário documentar as descobertas resultantes das observações feitas. Para consolidar o resultado é preciso rever os resultados com as pessoas observadas e/ou com seus superiores.

A análise de observação tem algumas desvantagens como, consumir bastante tempo e o analista ser induzido a erros em suas observações. Mas em geral a técnica de observação é muito útil e freqüentemente usada para complementar descobertas obtidas por outras técnicas.

Workshops

Trata-se de uma técnica de elicitação em grupo usada em uma reunião estruturada. Devem fazer parte do grupo uma equipe de analistas e uma seleção dos *stakeholders* que melhor representam a organização e o contexto em que o sistema será usado, obtendo assim um conjunto de requisitos bem definidos.

Ao contrário das reuniões, onde existe pouca interação entre todos os elementos presentes, o *workshop* tem o objetivo de acionar o trabalho em equipe. Há um facilitador neutro cujo papel é conduzir a *workshop* e promover a discussão entre os vários mediadores. As tomadas de decisão são baseadas em processos bem definidos e com o objetivo de obter um processo de negociação, mediado pelo facilitador.

Uma técnica utilizada em *workshops* é o *brainstorming*. Após os *workshops* serão produzidas documentações que refletem os requisitos e decisões tomadas sobre o sistema a ser desenvolvido.

Alguns aspectos importantes a serem considerados: a postura do condutor do seminário deve ser de mediador e observador; a convocação deve possuir dia, hora, local, horário de início e de término; assunto a ser discutido e a documentação do seminário.

Prototipagem

Protótipo tem por objetivo explorar aspectos críticos dos requisitos de um produto, implementando de forma rápida um pequeno subconjunto de funcionalidades deste produto. O protótipo é indicado para estudar as alternativas de interface do usuário; problemas de comunicação com outros produtos; e a viabilidade de atendimento dos requisitos de desempenho. As técnicas utilizadas na elaboração do protótipo são várias: interface de usuário, relatórios textuais, relatórios gráficos, entre outras.

Alguns dos benefícios do protótipo são as reduções dos riscos na construção do sistema, pois o usuário chave já verificou o que o analista captou nos requisitos do produto. Para ter sucesso na elaboração dos protótipos é necessária a escolha do ambiente de prototipagem, o entendimento dos objetivos do protótipo por parte de todos os interessados no projeto, a focalização em áreas menos compreendidas e a rapidez na construção.

Entrevistas

A entrevista é uma das técnicas tradicionais mais simples de utilizar e que produz bons resultados na fase inicial de obtenção de dados. Convém que o entrevistador dê margem ao entrevistado para expor as suas idéias. É necessário ter um plano de entrevista para que não haja dispersão do assunto principal e a entrevista fique longa, deixando o entrevistado cansado e não produzindo bons resultados.

As seguintes diretrizes podem ser de grande auxilio na direção de entrevistas bem sucedidas com o usuário: desenvolver um plano geral de entrevistas, certificar-se da autorização para falar com os usuários, planejar a entrevista para fazer uso eficiente do tempo, utilizar ferramentas automatizadas que sejam adequadas, tentar descobrir que informação o usuário está mais interessado e usar um estilo adequado ao entrevistar.

Para planejar a entrevista é necessário que antes dela sejam coletados e estudados todos os dados pertinentes à discussão, como formulários, relatórios, documentos e outros. Dessa forma, o analista estará bem contextualizado e terá mais produtividade nos assuntos a serem discutidos na entrevista.

É importante determinar um escopo relativamente limitado, focalizando uma pequena parte do sistema para que a reunião não se estenda por mais de

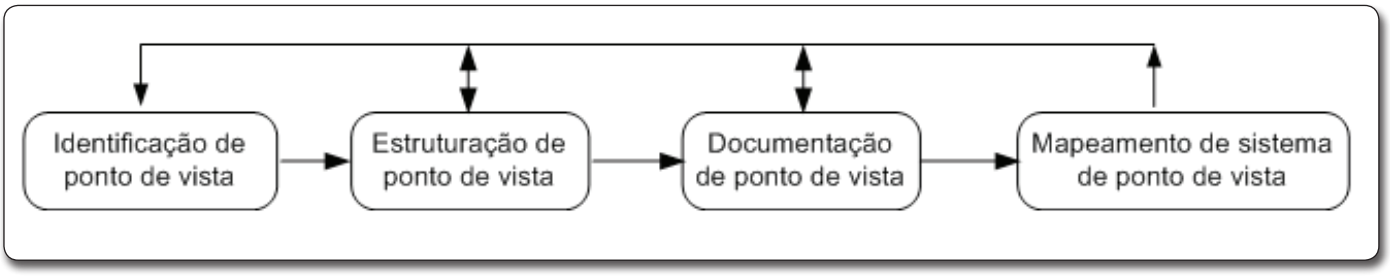


Figura 2. Método VORD, SOMMERVILLE – 2003

uma hora. O usuário tem dificuldade de concentração em reuniões muito longas, por isso é importante focalizar a reunião no escopo definido.

Após a entrevista é necessário validar se o que foi documentado pelo analista está de acordo com a necessidade do usuário, que o usuário não mudou de opinião e que o usuário entende a notação ou representação gráfica de suas informações.

A atitude do analista em relação à entrevista é determinar seu fracasso ou sucesso. Uma entrevista não é uma competição, deve-se evitar o uso excessivo de termos técnicos e não conduzir a entrevista em uma tentativa de persuasão. O modo como o analista fala não deve ser muito alto, nem muito baixo, tampouco indiretamente, ou seja, utilizar os termos: ele disse isso ou aquilo na reunião para o outro entrevistado. O modo melhor para agir seria, por exemplo, dizer: O João vê a solução para o projeto dessa forma. E o senhor André, qual é a sua opinião? Em uma entrevista o analista nunca deve criticar a credibilidade do entrevistado. O analista deve ter em mente que o entrevistado é o perito no assunto e fornecerá as informações necessárias ao sistema.

Para elaborar perguntas detalhadas é necessário solicitar que o usuário:

- Explique o relacionamento entre o que está em discussão e as demais partes do sistema;
- Descreva o ponto de vista de outros usuários em relação ao item que esteja sendo discutido;
- Descreva informalmente a narrativa do item em que o analista deseja obter informações;
- Perguntar ao usuário se o item em discussão depende para a sua existência de alguma outra coisa, para assim poder juntar os requisitos comuns do sistema, formando assim um escopo conciso.

Pode-se utilizar a confirmação, para tanto o analista deve dizer ao usuário o que acha que ouviu ele dizer. Neste caso, o analista deve utilizar as suas próprias palavras em lugar das do entrevistado e solicitar ao entrevistado confirmação do que foi dito.

Questionários

O uso de questionário é indicado, por exemplo, quando há diversos grupos de usuários que podem estar em diversos locais diferentes do país. Neste caso, elaboram-se pesquisas específicas de acompanhamento com usuários selecionados, que a contribuição em potencial pareça mais importante, pois não seria prático entrevistar todas as pessoas em todos os locais.

Existem vários tipos de questionários que podem ser utilizados. Entre estes podemos listar: múltipla escolha, lista de verificação e questões com espaços em branco. O questionário deve ser desenvolvido de forma a minimizar o tempo gasto em sua resposta.

Na fase de preparação do questionário deve ser indicado o tipo de informação que se deseja obter. Assim que os requisitos forem definidos o analista deve elaborar o questionário com questões de forma simples, clara e concisa, deixar espaço suficiente para as repostas que forem descritivas e agrupar as questões de tópicos específicos em um conjunto com um título especial. O questionário deve ser acompanhado por uma carta explicativa, redigida por um alto executivo, para enfatizar a importância dessa pesquisa para a organização.

Deve ser desenvolvido um controle que identifique todas as pessoas que receberão os questionários. A distribuição deve ocorrer junto com instruções detalhadas sobre como preenchê-lo e ser indicado claramente o prazo para devolução do questionário. Ao analisar as respostas dos participantes é feito uma consolidação das informações fornecidas no questionário, documentando as principais descobertas e enviando uma cópia com estas informações para o participante como forma de consideração pelo tempo dedicado a pesquisa.

Brainstorming

Brainstorming é uma técnica para geração de idéias. Ela consiste em uma ou várias reuniões que permitem que as pessoas sugiram e explorem idéias.

As principais etapas necessárias para conduzir uma sessão de *brainstorming* são:

- **Seleção dos participantes:** Os participantes devem ser selecionados em função das contribuições diretas que possam dar durante a sessão. A presença de pessoas bem informadas, vindas de diferentes grupos garantirá uma boa representação;
- **Explicar a técnica e as regras a serem seguidas:** O líder da sessão explica os conceitos básicos de *brainstorming* e as regras a serem seguidas durante a sessão;
- **Produzir uma boa quantidade de idéias:** Os participantes geram tantas idéias quantas forem exigidas pelos tópicos que estão sendo o objeto do *brainstorming*. Os participantes são convidados, um por vez, a dar uma única idéia. Se alguém tiver problema, passa a vez e espera a próxima rodada.

No *brainstorming* as idéias que a princípio pareçam não convencionais, são encorajadas, pois elas frequentemente estimulam os participantes, o que pode levar a soluções criativas para o problema. O número de idéias geradas deve ser bem grande, pois quanto mais idéias forem propostas, maior será a chance de aparecerem boas idéias. Os participantes também devem ser encorajados a combinar ou enriquecer as idéias de outros e, para isso, é necessário que todas as idéias permaneçam visíveis a todos os participantes.

Nesta técnica é designada uma pessoa para registrar todas as idéias em uma lousa branca ou em papel. À medida que cada folha de papel é preenchida, ela é colocada de forma que todos os participantes possam vê-la.

Analisar as idéias é a fase final do *brainstorming*. Nessa fase é realizada uma revisão das idéias, uma de cada vez. As consideradas valiosas pelo grupo são mantidas e classificadas em ordem de prioridade.

JAD

JAD (*Joint Application Design*) é uma técnica para promover cooperação, entendimento e trabalho em grupo entre os usuários desenvolvedores.

O JAD facilita a criação de uma visão compartilhada do que o produto de software deve ser. Através da sua utilização os desenvolvedores ajudam os usuários a formular problemas e explorar soluções. Dessa forma, os usuários ganham um sentimento de envolvimento, posse e responsabilidade com o sucesso do produto.

A técnica JAD tem quatro princípios básicos:

1. Dinâmica de grupo: são realizadas reuniões com um líder experiente, analista, usuários e gerentes, para despertar a força e criatividade dos participantes. O resultado final será a determinação dos objetivos e requisitos do sistema;

2. Uso de técnicas visuais: para aumentar a comunicação e o entendimento;

3. Manutenção do processo organizado e racional: o JAD emprega a análise *top down* e atividades bem definidas. Possibilita assim, a garantia de uma análise completa reduzindo as chances de falhas ou lacunas no projeto e cada nível de detalhe recebe a devida atenção;

4. Utilização de documentação padrão: preenchida e assinada por todos os participantes. Este documento garante a qualidade esperada do projeto e promove a confiança dos participantes.

A técnica JAD é composta de duas etapas principais: planejamento, que tem por objetivo elicitar e especificar os requisitos; e projeto, em que se lida com o projeto de software.

Cada etapa consiste em três fases: adaptação, sessão e finalização. A fase de adaptação consiste na preparação para a sessão, ou seja, organizar a equipe, adaptar o processo JAD ao produto a ser construído e preparar o material. Na fase de sessão é realizado um ou mais encontros estruturados, envolvendo desenvolvedores e usuários onde os requisitos são desenvolvidos e documentados. A fase de finalização tem por objetivo converter a informação da fase de sessão em sua forma final (um documento de especificação de requisitos).

Há seis tipos de participantes, embora nem todos participem de todas as fases:

- **Líder da sessão:** é responsável pelo sucesso do esforço, sendo o facilitador dos encontros. Deve ser competente, com bom relacionamento pessoal e qualidades gerenciais de liderança;
- **Engenheiro de requisitos:** é o participante diretamente responsável pela produção dos documentos de saída das sessões JAD. Deve ser um desenvolvedor experiente para entender as questões técnicas e detalhes que são discutidos durante as sessões e ter habilidade de organizar idéias e expressá-las com clareza;
- **Executor:** é o responsável pelo produto sendo construído. Tem que fornecer aos participantes uma visão geral dos pontos estratégicos do produto de software a ser construído e tomar as decisões executivas, tais como alocação de recursos, que podem afetar os requisitos e o projeto do novo produto;
- **Representantes dos usuários:** são as pessoas na empresa que irão utilizar o produto de software. Durante a extração de requisitos, os representantes são frequentemente gerentes ou pessoas-chave dentro da empresa que tem uma visão melhor do todo e de como ele será usado;
- **Representantes de produtos de software:** são pessoas que estão bastante familiarizadas com as capacidades dos produtos de software. Seu papel é ajudar os usuários a entender o que é razoável ou possível que o novo produto faça;
- **Especialista:** é a pessoa que pode fornecer informações detalhadas sobre um tópico específico.

O conceito do JAD de abordagem e dinâmica de grupo poderá ser utilizado para diversas finalidades, como: planejamento de atividades técnicas para um grande projeto, discussão do escopo e objetivos de um projeto e estimativa da quantidade de horas necessárias para desenvolver sistemas grandes e complexos.

A maioria das técnicas JAD funciona melhor em projetos pequenos ou médios. Para um sistema grande e complexo podem ser usadas múltiplas sessões JAD para acelerar a definição dos requisitos do sistema.

Técnicas tradicionais	São aplicadas em várias áreas do conhecimento. Exemplo: questionários, entrevistas, observação, e análise de documentos.
Técnicas de elicitação de grupo	Tem por objetivo compreender melhor o pensamento e comportamento dos grupos e as necessidades dos usuários. Exemplo: *brainstorming* e as sessões JAD (*Joint Application Design*).
Prototipação	O uso de protótipo auxilia na elicitação e validação dos requisitos de sistema. A prototipação pode ser utilizada para elicitar requisitos quando há um alto grau de incerteza ou quando é necessário um rápido *feedback* dos usuários.
Técnicas contextuais	Surgiram como uma alternativa para as técnicas tradicionais e cognitivas e inclui técnicas de etnografia e análise social.

Tabela 1. Grupos de técnicas para levantamento de requisitos

Conclusão

Não existe uma técnica padrão para o processo de levantamento de requisitos. Para alcançar um levantamento de requisitos mais preciso é importante o conhecimento de diversas técnicas para saber que técnica de levantamento aplicar em cada situação.

É primordial que o analista possua perfil adequado. O analista de sistemas precisa de mais do que apenas a capacidade de desenhar fluxogramas e outros diagramas técnicos. O analista de sistemas tem a função de projetar e analisar sistemas de ótimo desempenho. Para que esse objetivo seja alcançado, é necessário o analista de sistema possuir a capacidade de:

- Compreender conceitos abstratos, reorganizá-los em divisões lógicas e sintetizar soluções baseadas em cada divisão;
- Absorver fatos pertinentes de fontes conflitantes ou confusas;
- Entender os ambientes do usuário/cliente;
- Aplicar elementos do sistema de hardware e/ou software aos elementos do usuário/cliente;
- Comunicar bem nas formas escrita e verbal e entender o objetivo global do software.

A **Tabela 1** apresenta os grupos de técnicas para o levantamento de requisitos.

Referências

CARVALHO, Adriane M. B. Rizzoni; CHIOSSI, Thelma C. dos Santos. **Introdução à engenharia de software.** Campinas, SP. Ed UNICAMP, 2001

FILHO, Wilson de Pádua Paula. **Engenharia de software Fundamentos, Métodos e Padrões**. Rio de Janeiro, RJ. Ed LTC, 2001

FOURNEIR, Roger. **Guia prático para desenvolvimento e manutenção de sistemas estruturados**. São Paulo. Ed. Makron Books, 1994

POMPILHO, S. **Análise Essencial Guia Prático de Análise de Sistemas**. Rio de Janeiro: Ed Ciência Moderna Ltda, 1995.

PRESSMAN, Roger S. **Engenharia de Software**. São Paulo. Ed. Markon Books, 1995

REZENDE, Denis Alcides. **Engenharia de software e sistemas de informação**. 2.ed. Rio de Janeiro: Ed. Brasport, 2002.

SOMERVILLE, I. **Engenharia de software**. 6° ed. Tradução Maurício de Andrade. São Paulo: Ed Addison-Wesley, 2003.

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