engenharia de software magazine

DevMedia

Ano II - Edição 14

Planejamento
Conheça as principais mudanças da 3ª para a 4ª edição do PMBoK

Arquitetura:
Saiba como analisar a arquitetura para projetos de software

Engenharia de Software:
Ontologias: Filosofia aplicada à Engenharia de Software?

Melhoria de Processo

- Através do uso de abordagens ágeis
- Através do uso do MPS.BR

Projeto
Entenda o que é gerência de configuração conhecendo suas principais tarefas

Projeto
Gestão de mudanças utilizando o PMBoK e a ferramenta Rational ClearQuest

Arquitetura
Maximize a geração de valor de projetos e produtos com arquiteturas de software

Aulas desta edição:

- Soluções Concretas para Problemas Práticos da Engenharia de Requisitos – Parte 4
- Soluções Concretas para Problemas Práticos da Engenharia de Requisitos – Parte 5
- Soluções Concretas para Problemas Práticos da Engenharia de Requisitos – Parte 6

ISSN 1983127-7

9 771983 127008 00014

Conhecimento faz diferença!

Mais de 60 mil downloads na primeira edição!

Faça já sua assinatura digital! | *www.devmedia.com.br/es*

Faça um *up grade* em sua carreira.
Em um mercado cada vez mais focado em qualidade ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre outros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lança para os desenvolvedores brasileiros sua primeira revista digital totalmente especializada em Engenharia de Software. Todos os meses você irá encontrar artigos sobre Metodologias Ageis; Metodologias tradicionais (*document driven*); ALM (*application lifecycle management*); SOA (aplicações orientadas a servicos); Analise de sistemas; modelagem; Métricas; orientação à objetos; UML; testes e muito mais. **Assine já!**

DevMedia GROUP

engenharia de software magazine

Ano 2 - 14ª Edição 2009 Impresso no Brasil

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:

Deise Aleis – 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

EDITORIAL

A melhoria de processos é algo que sempre procuramos, afinal, é clara a necessidade de aperfeiçoamento constante nos diferentes projetos em que trabalhamos. Infelizmente, não existe uma receita de bolo ou conjunto de passos pré-definidos que precisamos seguir para ter sucesso em uma caminhada ruma à melhoria da forma como desenvolvemos software.

Neste contexto, a Engenharia de Software Magazine destaca nesta edição duas matérias muito interessantes que abordam o tema sob duas perspectivas distintas: abordagens ágeis e abordagens tradicionais.

• Implantação de Processo - Os desafios da implantação de um processo de software: neste artigo veremos quais são os principais desafios e benefícios da implantação de um processo de desenvolvimento de software. Serão descritos aspectos importantes do processo que foi definido, bem como as ferramentas que foram desenvolvidas para dar suporte à sua implantação.

• Melhoria de Processo de Software no Desenvolvimento Ágil: a procura pelo conhecimento e aplicação de metodologias ágeis vem aumentando. Desta forma, este artigo tem a finalidade de prover (i) um melhor entendimento de como e quando realizar um processo MPS; e (ii) apresentar etapas a serem seguidas e resultados esperados tratados através do uso de técnicas utilizadas nas metodologias ágeis para iniciativas de MPS.

Além destas matérias, esta edição traz mais seis artigos:

• Gerência de Configuração: Definições Iniciais;
• Mudanças no PMBoK 4ª Edição;
• Gerenciamento de Mudanças em Projetos de TI;
• Ontologias: Filosofia aplicada à Engenharia de Software?;
• Ciclo de Vida da Gestão em Arquitetura de Software;
• Análise da Arquitetura de Software.

Desejamos uma ótima leitura!
Equipe Editorial Engenharia de Software Magazine

Rodrigo Oliveira Spínola
(rodrigo@sqlmagazine.com.br)

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

Marco Antônio Pereira Araújo
(maraujo@devmedia.com.br)

Doutor e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ - Linha de Pesquisa em Engenharia de Software, Especialista em Métodos Estatísticos Computacionais e Bacharel em Matemática com Habilitação em Informática pela UFJF, Professor e Coordenador do curso de Bacharelado em Sistemas de Informação do Centro de Ensino Superior de Juiz de Fora, Professor do curso de Bacharelado em Sistemas de Informação da Faculdade Metodista Granbery, Professor e Diretor do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas da Fundação Educacional D. André Arcoverde, Analista de Sistemas da Prefeitura de Juiz de Fora, Colaborador da Engenharia de Software Magazine.

Eduardo Oliveira Spínola
(eduspinola@gmail.com)

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

Caro Leitor,

Para esta edição, temos um conjunto de 3 vídeo aulas. Estas vídeo aulas estão disponíveis para download no Portal da Engenharia de Software Magazine e certamente trarão uma significativa contribuição para seu aprendizado. A lista de aulas publicadas pode ser vista abaixo:

Tipo: Engenharia de Requisitos
Título: Concretas para Problemas Práticos da Engenharia de Requisitos – Parte 4
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Existem diferentes maneiras de estruturar um processo com atividades relacionadas à engenharia de requisitos em empresas desenvolvedoras de software. Modelos de maturidade, como o MPS, podem servir como um arcabouço para a definição deste processo. Além disso, é fundamental para um processo de engenharia de requisitos que ele seja capaz de lidar com dificuldades e problemas relacionados a requisitos que possam surgir durante o desenvolvimento de software na prática. Uma iniciativa rumo ao levantamento destas dificuldades e problemas e de maneiras de estruturar um processo para lidar com estes problemas será apresentada nesta série de vídeo aulas. Nesta quarta aula, serão apresentados dois problemas práticas e duas soluções concretas associadas à atividade de análise e negociação de requisitos.

Tipo: Engenharia de Requisitos
Título: Soluções Concretas para Problemas Práticos da Engenharia de Requisitos – Parte 5
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Existem diferentes maneiras de estruturar um processo com atividades relacionadas à engenharia de requisitos em empresas desenvolvedoras de software. Modelos de maturidade, como o MPS, podem servir como um arcabouço para a definição deste processo. Além disso, é fundamental para um processo de engenharia de requisitos que ele seja capaz de lidar com dificuldades e problemas relacionados a requisitos que possam surgir durante o desenvolvimento de software na prática. Uma iniciativa rumo ao levantamento destas dificuldades e problemas e de maneiras de estruturar um processo para lidar com estes problemas será apresentada nesta série de vídeo aulas. Nesta quinta aula, serão apresentados dois problemas práticas e duas soluções concretas associadas à atividade de documentação de requisitos.

Tipo: Engenharia de Requisitos
Título: Concretas para Problemas Práticos da Engenharia de Requisitos – Parte 6
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Existem diferentes maneiras de estruturar um processo com atividades relacionadas à engenharia de requisitos em empresas desenvolvedoras de software. Modelos de maturidade, como o MPS, podem servir como um arcabouço para a definição deste processo. Além disso, é fundamental para um processo de engenharia de requisitos que ele seja capaz de lidar com dificuldades e problemas relacionados a requisitos que possam surgir durante o desenvolvimento de software na prática. Uma iniciativa rumo ao levantamento destas dificuldades e problemas e de maneiras de estruturar um processo para lidar com estes problemas será apresentada nesta série de vídeo aulas. Nesta sexta aula, serão apresentados mais dois problemas práticas e duas soluções concretas associadas à atividade de documentação de requisitos.

ÍNDICE

Você não está mais sozinho.

Melhoria de Processo de Software no Desenvolvimento Ágil

Célio Santana

Celio.santana@scrum.org.br

Professor Assistente das Faculdades Integradas Barros Melo e do Grupo Universitário Maurício de Nassau. Mestre em Engenharia da Computação pela Universidade de Pernambuco e Pós Graduado em Melhoria de Processo de Software pela Universidade Federal de Lavras (UFLA). Graduado em Ciência da Computação pela Universidade Federal de Pernambuco. É Certified Scrum-Master (CSM), Certified Product-Owner (CSPO) bem como Implementador do Modelo de Melhoria de Processos do Software Brasileiro MPS.Br (P2 MPS.Br)

Cristine Gusmão

cristine@dsc.upe.br

Professora Assistente do Departamento de Sistemas e Computação 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.

De que se trata o artigo?

Este artigo apresenta o conceito de melhoria de processo de software e sua aplicação em aplicações que utilizam abordagens de metodologias ágeis.

Para que serve?

Serve como fonte de exemplos de aplicações da utilização de Melhoria de Processo de Software (MPS) em ambientes ágeis e entender sua diferença em relação ao MPS tradicional.

Em que situação o tema é útil?

A procura pelo conhecimento e aplicação de metodologias ágeis vem aumentando. Desta forma, este artigo tem a finalidade de prover (i) um melhor entendimento de como e quando realizar um processo MPS; e (ii) apresentar etapas a serem seguidas e resultados esperados tratados através do uso de técnicas utilizadas nas metodologias ágeis para iniciativas de MPS.

O surgimento das metodologias ágeis foi de fato um importante marco na indústria do desenvolvimento de software. Algumas definições, disponíveis na literatura [Schuh 2004; Mnkandla & Dwolatzky 2004], tratam o tema como algo revolucionário, sendo considerada uma nova disciplina de engenharia que modificava os valores do processo de desenvolvimento de software do mecânico (orientado a processos e utilizando regras da ciência) para o orgânico (dirigido por questões sobre pessoas e suas interações).

De uma forma geral, a apresentação destas metodologias trouxe mudanças culturais em vários aspectos no desenvolvimento de software, a exemplo da proposição de técnicas e procedimentos, até a contribuição na realização da Melhoria de Processo de Software (MPS).

Baseado nesse contexto de mudança cultural, uma nova forma para a realização da Melhoria de Processo de Software era necessária sem que fosse tirado o foco do "orgânico", valorizando ainda mais as pessoas e suas competências. Neste artigo será mostrado como é proposto o MPS nas metodologias ágeis e sua diferença do MPS tradicional.

Histórico

Para melhor entendermos o contexto do MPS em ambientes tradicionais, devemos entender o que aconteceu com o rumo que a Qualidade de Software tomou a partir daquela reunião da OTAN em 1968 onde o termo "Engenharia de Software" foi utilizado pela primeira vez por F. L. Bauer.

Naquela reunião foi utilizado também o termo "Crise do Software" para definir a situação em que a indústria do software atravessava naquele momento. E a crise foi atribuída à complexidade de desenvolver sistemas cada vez maiores, bem como à falta de gerenciamento no processo de desenvolvimento de software.

A partir daí, "engenheiros de software" tentaram imitar a engenharia convencional, para resolver problemas de qualidade e falhas em sistemas de informação. Uma quantidade significativa de experiência foi obtida através de processos de garantia da qualidade praticados na indústria de manufatura e essa adaptação para a indústria de software foi, em alguns casos, um fracasso e, em outros, um sucesso, como, por exemplo, a utilização de controle estatístico de processo (base do Six Sigma) para avaliação de processos de software.

Ainda no fim da década de 1980, o controle de qualidade existente na indústria de software era centrado no produto final e com utilização de métodos corretivos em inspeções no fim da linha de produção, e se mostrava pouco efetivo para a solução de problemas gerenciais como prazos e custos.

No início da década de 1990, o mercado substituía aquele controle de qualidade pela Garantia da Qualidade com um foco centrado no processo e que utilizava auditorias durante todo o ciclo de vida de desenvolvimento. A partir daí, foram surgindo normas (ISO 9000-3, ISO 15504, ISO 12207), padrões (IEEE 1074, IEEE 1298) e Modelos (SW-CMM, CMMI) para Qualidade de Software.

A partir daí começaram a surgir os modelos para Melhoria de Processos de Software. A maioria é baseada no PDCA (Plan-Do-Check-Action) de Eduard Deming. Os modelos para MPS mais utilizados foram o IDEAL [Mcfeeley 1996], utilizado em conjunto com o SW-CMM, e o Modelo DMAIC utilizado pelo Six Sigma.

Com o advento da Garantia da Qualidade, a indústria de software passou a ser centrada em documentação e orientada a planejamento, e essa forma de desenvolvimento de software ficou conhecida como modelo tradicional [Boehm 1988].

Nesse contexto, a Melhoria do Processo de Software pode ser genericamente estereotipada com as seguintes características:

i. Criação de grupo responsável pelo programa de MPS, realizando atividades de levantamento de não conformidades e avaliação de desempenho. Na visão do Modelo IDEAL tem-se o Software Process Group Engineering (SEPG) e no DMAIC, os Green Belts e Black Belts;

ii. Monitoração do processo através de indicadores, permitindo a identificação das oportunidades de melhoria;

iii. Elaboração de projetos de melhoria com base nas oportunidades identificadas em (ii);

iv. Avaliação dos resultados dos projetos de melhoria através de critérios objetivos e previamente estabelecidos;

v. Utilização de projetos piloto para realização dos projetos de melhoria definidos a partir dos dados obtidos em (iv). Desta forma, a melhoria é institucionalizada para o processo da organização;

vi. Aplicação de ações corretivas continuamente à medida que uma não conformidade for encontrada;

Com a criação do Manifesto Ágil (www.agilemanifesto.org) em 2001, são formalizadas as metodologias ágeis que, de certa maneira, já existiam desde o meio da década de 1990, e que surgiram de descobertas comuns entre os participantes que aos poucos foram substituindo os processos da tradicional documentação pesada e desenvolvimento centrado no processo por abordagens mais centradas em pessoas e menos orientadas a documentos [Boehm & Turner 2004].

Diante do quadro de ruptura em relação aos modelos tradicionais pelas metodologias ágeis, aquela forma de MPS "tradicional" vinha de encontro às novas idéias propostas pelo manifesto ágil, sendo esta MPS "pesada" demais para as necessidades do desenvolvimento ágil de software.

Manifesto Ágil e a "Nova" Cultura

O termo "Metodologias Ágeis" foi formalizado em 17 de Fevereiro de 2001, por 17 líderes de desenvolvimento que propuseram metodologias até então conhecidas como metodologias "leves". Este encontro verificou pontos comuns dessas metodologias e resultou em um acordo entre os participantes que possuía quatro níveis.

O primeiro nível considera a necessidade da existência de métodos construídos para responder a mudanças durante o projeto de software. O termo "Ágil" foi adotado para esses métodos, pois o termo "leve" não era apropriado para certos projetos que teriam restrições em empregar metodologias leves mas, que ainda assim, poderiam precisar de agilidade.

O segundo nível do acordo foi à publicação do "Manifesto Ágil" que é composto por quatro valores que continham os valores centrais de todas as metodologias:

- Indivíduos e iterações mais do que processos e ferramentas;
- Software funcionando mais do que documentação extensa;
- Colaboração do cliente mais do que negociação de contratos;
- Responder a mudanças mais do que seguir um plano.

Ainda que haja valor nos termos à direita, são valorizados mais os termos à esquerda.

O terceiro nível é composto por um conjunto de doze princípios. Os valores empregados são fornecidos com maior detalhamento, conforme segue:

- A prioridade é cumprir as entregas programadas e, por vezes, antecipadas. Desta forma, o cliente se sentirá mais confiante e seguro no produto solicitado (valor para o cliente);
- Mudanças de requisitos são bem-vindas, mesmo em fases tardias do desenvolvimento. Processos ágeis abraçam mudanças para promover a vantagem competitiva do cliente;

• Entregar software funcionando frequentemente, entre poucas semanas a poucos meses, dando preferência à escala de tempo mais curta;
• Pessoas de software e de negócios devem trabalhar juntas diariamente através do projeto;
• Construa projetos em torno de indivíduos motivados, dê-lhes o ambiente e apoio que necessitem, e confie neles para que o trabalho seja feito;
• A forma mais eficiente e efetiva de trazer informação para o time, e dentro do mesmo, é a comunicação face a face.
• Software funcionando é a primeira métrica de progresso;
• Processos Ágeis promovem um ritmo sustentável. Patrocinadores, desenvolvedores e usuários devem estar aptos a manter o ritmo indefinidamente;
• Atenção contínua, excelência técnica e bons projetos ajudam a agilidade.
• Simplicidade: a arte de maximizar a quantidade de trabalho que não deve ser realizado é essencial;
• As melhores arquiteturas, requisitos e projetos surgem de times auto-organizados;
• Em intervalos regulares, o time deve refletir sobre como se tornar mais efetivo, então ajustar seu comportamento de acordo com a reflexão.

O quarto e último nível do acordo seriam formados com o passar do tempo. Então foi acordado que esse nível seria aquele no qual cada abordagem ágil deveria definir a si mesma. Algumas destas abordagens são: Extreme Programming (XP), Scrum, Crystal Clear e Lean.

Melhoria de Processo de Software

Um processo de software pode ser definido como uma sequência de passos necessários para o desenvolvimento ou manutenção de um software, com o objetivo de prover um conjunto de boas práticas técnicas e gerenciais para a utilização de técnicas e ferramentas por pessoas que irão realizar tarefas para alcançar o objetivo determinado [Humphrey 1995].

Atualmente todos os programas que carregam a alcunha de Melhoria de Processo que são criados dentro de uma organização, visam o Retorno do Investimento (Return On Investment – ROI) para que mais recursos estejam disponíveis para que a mesma continue crescendo [Rico 2004].

Vimos anteriormente os seis passos comuns a todos os programas de Melhoria de Processo de Software executados nos modelos tradicionais, contudo serão mostrados seis elementos que são considerados em qualquer programa de MPS executado nos moldes tradicionais [Salo 2007].

1. MODELOS ORGANIZACIONAIS PARA MELHORIA DE PROCESSO DE SOFTWARE

Nos modelos tradicionais, a melhoria de processo tem um caráter organizacional, ou seja, os processos da instituição são mapeados, avaliados e as melhores práticas são institucionalizadas, ou se tornarão parte do processo padrão da empresa.

Esse processo padrão apresenta todos os processos de uma organização e critérios definidos de adaptação. Ou seja, a empresa possui algumas boas práticas internas mapeadas e, no processo, existem indicações de quando usar cada uma delas.

Neste ponto o mais importante é entender que todos os projetos da empresa serão executados baseados nesse processo padrão e que as lições aprendidas são incorporadas ao processo padrão da organização, o que podemos definir como aprendizado organizacional. Para ajudar uma organização a escolher a melhor forma de "aprender" sobre si mesma existem alguns modelos organizacionais para MPS [Deming 1990; Cohn & Ford 2003; Boehm & Turner 2005]

2. PROCESSOS PADRÕES E AVALIAÇÃO DOS PROCESSOS

Alguns modelos para Maturidade de Melhoria de Processo, como o CMMI e o MPS.Br, são hoje referências no mercado para a avaliação de processos. Esses modelos fornecem um conjunto de boas práticas de como o processo pode ser melhorado e fornecem também um método de avaliação (SCAMPI para o CMMI e o MA.MPS para o MPS.Br) que ajuda a "medir" a evolução do programa de MPS na organização.

É entendido aqui que todo programa de melhoria de processo deve ser avaliado de forma objetiva seguindo critérios bem definidos e não ambíguos. Não existe ainda nenhum método de avaliação que se mostre melhor para todos os casos, normalmente cada modelo criado é adaptado de acordo com a situação.

3. ADAPTAÇÃO DE PROCESSOS

Abordagens de software tradicionais vêm sendo criticadas por não possuir critérios para a sua aplicabilidade, ou não se mostrarem universalmente aplicáveis [Malouin & Landry 1983].

Este ponto retrata uma questão na melhoria de processo de software sobre como uma organização, com um processo padrão rico em boas práticas e com uma variedade grande de projetos, pode escolher quais são as melhores práticas para maximizar o ROI de seus projetos. Este ponto do processo de MPS é conhecido como adaptação de processos. Basili e Rombach (1987) apresentaram uma das primeiras propostas para adaptar processos.

4. IMPLANTAÇÃO DE PROCESSOS

A implantação do processo é uma instância da MPS nas organizações. Ela pode incluir atividades como realização de projetos pilotos, métodos e ferramentas como potenciais soluções para metas ou problemas existentes, e a avaliação dos efeitos daquela mudança no desenvolvimento de software.

Nas abordagens tradicionais de desenvolvimento de software, é mais comum que a melhoria aborde o processo organizacional como um todo, ao invés de questões de um determinado processo em particular.

Basicamente, uma organização pode escolher uma estratégia Big-Ben ou por pedaços para implantar novos processos, métodos e ferramentas. Enquanto o Big-Ben é uma forma revolucionária que acredita que cada mudança súbita resulta em uma forma

distinta de trabalho, a estratégia por pedaços é uma abordagem evolucionária que prefere um período longo de evolução dentro da organização a partir de pequenas fases de melhoria.

5. MEDIÇÃO

Você não pode controlar aquilo que não pode medir [De Marco 1982]. A partir dessa premissa é verificado que sem atividades de medição e controle, fica extremamente difícil pensar em Melhoria de Processo de Software. Na verdade, a medição possui três propósitos [Fenton & Pfleeger 1997]:

- Entender o desenvolvimento e manutenção;
- Controlar Projetos;
- Melhorar Processos e Produtos.

Na melhoria de processo, a medição é fundamental para a promoção da melhoria contínua no processo. Vários mecanismos para obtenção de dados quantitativos existem hoje para auxiliar o processo de medição. Os mais comuns em programas de MPS são o GQM (Goal Question Metric) e o PSM (Practical Software and Systems Measurement) [Fenton & Pfleeger 1997].

6. EXPERIÊNCIA, CONHECIMENTO E APRENDIZADO

O valor representado pela experiência e conhecimento não deve ser subestimado dentro de um programa de MPS e dentro de sua expectativa de melhoria contínua. Deming (1990) afirmava que perder conhecimento é mais prejudicial do que perder material. Em programas de MPS é considerado crucial que as lições aprendidas dentro dos projetos correntes sejam testadas em outros projetos e, se o resultado for positivo, que esse conhecimento seja adicionado à base de conhecimento da organização, e isso normalmente ocorre com uma adição ao processo padrão da organização.

Neste aspecto, podemos ressaltar o verdadeiro objetivo dos programas de MPS dentro das organizações, que é o aprendizado daquilo que funciona ou não dentro daquela organização, e em que contexto este resultado pode ser reproduzido.

MPS no Desenvolvimento Ágil de Software

É correto afirmar que técnicas e métodos de MPS estão limitados dentro do desenvolvimento ágil de software, mesmo tendo essa Melhoria de Processo de Software um papel fundamental dentro de qualquer método ágil.

Esta importância é observada em um dos princípios presentes no 3° nível do acordo que gerou o manifesto ágil. O último princípio das metodologias ágeis foi definido como: "Em intervalos regulares o time deve refletir sobre como se tornar mais efetivo e ajustar seu comportamento de acordo com esta reflexão".

De fato, o desenvolvimento ágil de software provê uma forma "não-tradicional" para a realização da MPS, pois este considera a melhoria do processo de obtenção e compartilhamento do conhecimento entre os desenvolvedores, sendo necessário criar uma "escala" para poder pontuar a experiência desse time e maximizar o ganho dessa experiência e sua disseminação por toda a equipe. Assim, o desenvolvimento ágil de software possibilita novas formas para realização desta melhoria de processo além daquelas apresentadas pelos modelos tradicionais.

Para o desenvolvimento ágil de software, a melhoria de processo possui as seguintes características:

i. Todo o time é responsável pela melhoria de processo;
ii. As melhorias propostas são resultados da experiência do time sobre o que pode melhorar em seu próprio comportamento sem a necessidade de indicadores formais ou critérios bem definidos;
iii. As mudanças propostas pelo time entram em vigor a partir da próxima iteração, ou seja, já é incorporado ao processo da equipe não precisando de projetos pilotos;
iv. A avaliação das mudanças ocorre pelo sentimento da equipe durante o que aconteceu durante aquela iteração, então se a mudança "funcionou", a equipe pode manter aquela mudança;
v. As mudanças são propostas em intervalos bem definidos, esses intervalos podem de durar de uma a poucas iterações;
vi. O time tem autonomia para se auto-organizar e determinar como o processo será seguido e determinar que ações realizar quando não conformidades forem encontradas.

Avaliando os seis elementos presentes na melhoria de processo quando tratamos do desenvolvimento ágil, temos os seguintes resultados:

1. MODELOS ORGANIZACIONAIS PARA MELHORIA DE PROCESSO DE SOFTWARE

No desenvolvimento ágil de software, o time é a unidade responsável pela execução de um projeto. No 3° nível do acordo que gerou o manifesto ágil é observado o seguinte princípio: "As melhores arquiteturas, requisitos e projetos surgem de times auto-organizados".

Esse princípio esclarece que, no contexto ágil, as decisões do time são mais prioritárias do que aquelas vindas da organização. A uniformidade não é encorajada. Todos os times trabalham, logo podemos considerar que nesse contexto não há um processo padrão definido. Porém, se pensarmos que os times da organização utilizam uma mesma abordagem, por exemplo, XP, pode-se pensar em uma forma muita parecida de trabalho.

Boehm & Turner (2005) afirmam que existem desafios gerenciais em implantar processos ágeis, pois eles não seguem a hierarquia Topdown encontradas nas organizações tradicionais, uma vez que o empowerment dado aos times tornam essa mesma estrutura auto-gerenciável.

2. PROCESSOS PADRÕES E AVALIAÇÃO DOS PROCESSOS

A avaliação do processo, bem como de todas as melhorias propostas pela equipe, não possuem indicadores definidos nem critérios de aceitação formais que indiquem a aprovação ou não do mesmo. As avaliações são realizadas de forma subjetiva, de acordo com a experiência que o time obteve em experimentar aquele "novo" processo durante a iteração.

Da mesma maneira informal com a qual a avaliação é realizada, de forma subjetiva e sem a utilização de indicadores, também é a forma com que se escolhe aquilo que deve ser alterado no processo. Não há uma maneira sistemática para decidir, ou avaliar, o que deve ser modificado e no que está baseada a escolha, a não ser pela experiência do próprio time.

O método de avaliação que vem sendo aceito comumente pelos times Scrum é o Nokia Test [Vodde 2006]. Outra iniciativa interessante que está se definindo como um padrão na área das metodologias ágeis é o IEEE 1648 (http://standards.ieee.org/board/nes/projects/1648.pdf).

3. ADAPTAÇÃO DE PROCESSOS

Nas metodologias ágeis, os processos são adaptados com mais freqüência e de forma não sistematizada. Contudo, o desenvolvimento ágil considera que processos que são criados para serem repetidos são falhos e consideram que cada situação exige um processo diferente, uma vez que processos repetitivos só funcionam com as mesmas pessoas, nos mesmos ambientes, com as mesmas ferramentas, para solucionar os mesmos problemas repetidas vezes.

De fato, o desenvolvimento ágil precisa de adaptações dentro de projetos individuais dentro de uma organização, e esse processo é continuamente melhorado a partir de pequenas mudanças. Afinal, o desenvolvimento de software proposto pelas metodologias ágeis não está baseado em processos preditivos em ambientes estáveis para o desenvolvimento, e sim, baseado em um ambiente de mudanças frequentes, que ainda assim necessitam de qualidade.

4. IMPLANTAÇÃO DE PROCESSOS

A maioria das implantações de processos no desenvolvimento ágil está mais voltada para a abordagem Big-Ben, onde grandes mudanças podem ser sugeridas, no entanto não há critérios para essas mudanças.

O maior desafio nestes tipos de implantação é a falta de transparência entre os diversos níveis da organização (hierarquia) em relação ao time auto-gerenciável, o que pode resultar em um relacionamento fracassado entre as partes interessadas de mais alto-nível e o time. Mais dificuldades sobre a implantação de processos em organizações ágeis são listadas em [Cohn & Ford 2003].

5. MEDIÇÃO

Boehm & Turner (2005) afirmam que a maioria das técnicas de medição tradicionais podem se tornar inadequadas para o apoio aos métodos ágeis. As razões mais comuns desta falha provêm da diferente forma como as estruturas analíticas de projeto (EAP) são construídas. Enquanto nos métodos tradicionais essas EAPs são um conjunto de atividades e tarefas normalmente exibidas em gráficos de Gantt, nos métodos ágeis temos a manipulação de cartões de estórias (XP) ou itens de Backlog (Scrum) que são detalhados em nível satisfatório apenas quando são desenvolvidos.

Falando em métricas para melhoria de processos, em particular, não vemos nenhuma indicação no manifesto ágil de que esta seja uma ferramenta para o MPS. Contudo, XP indica que sempre que for necessária uma adaptação ao processo, deve ser criado um mecanismo para avaliar o resultado dessa mudança, e Scrum afirma que se as coisas corretas forem medidas, é possível a realização de melhorias.

Observando o princípio: "Software funcionando é a primeira métrica de progresso" pode-se considerar que as medições são uma importante ferramenta para controle da execução das atividades de engenharia do produto (gerenciamento do projeto) do que para a Melhoria de Processos no desenvolvimento ágil.

6. EXPERIÊNCIA, CONHECIMENTO E APRENDIZADO

Este é um dos pontos mais valorizados no desenvolvimento ágil de software, especialmente sobre os times que desenvolvem um software específico. A ênfase dada ao aprendizado coletivo do time é observado no princípio ágil: "Em intervalos regulares, o time deve refletir sobre como se tornar mais efetivo, então ajustar seu comportamento de acordo com a reflexão".

Beck & Andrés (2004) afirmam que, em XP, bons times não fazem apenas o seu trabalho, mas pensam sobre como realizar seu trabalho e por que realizá-lo. O time analisa o porquê de seu sucesso e sua falha, e eles não tentam esconder os seus erros, pelo contrário, eles são encorajados a mostrá-los e aprender com eles.

O time deve ser encorajado a aprender não só com a experiência deles, mas também com a experiência do cliente [Koch 2005]. No desenvolvimento ágil é encorajada a comunicação face a face para que haja um entendimento melhor durante o ciclo de vida do desenvolvimento.

Contudo, não é considerado aqui o aprendizado organizacional. Os times podem adaptar técnicas diferentes, para atender necessidades diferentes no processo de desenvolvimento, e este tipo de experiência fica implícito, uma vez que não existe um mecanismo definido para troca de experiências dentro dos times ágeis. O que normalmente é feito, é que um membro de cada time seja trocado de lugar e este compartilhe as suas experiências com os outros times, homogeneizando o conhecimento.

A técnica mais utilizada para a realização de MPS dentro do desenvolvimento ágil são as retrospectivas. Esta técnica foi desenvolvida originalmente para o método Crystal Clear, mas foi incorporado ao SCRUM (Sprint Retrospectives Meetings) e ao XP (Reflexões).

Esta técnica se resume em uma reunião que deve ocorrer sempre no final de uma iteração. Nessa reunião o time deve opinar sobre três aspectos:

- O que está funcionando e deve ser mantido?
- O que está ruim e deve ser modificado?
- O que podemos tentar fazer diferente?

Estes três tópicos são discutidos de acordo com a experiência de cada desenvolvedor e todos dão a sua opinião sobre cada idéia surgida. Contudo, não há critérios ou indicadores para justificar/avaliar cada uma das sugestões, ficando apenas a critério do time se a mudança será incorporada ou não ao processo.

As mudanças que forem aceitas pela equipe entram em vigor já na iteração seguinte e, na próxima reunião de retrospectivas, ela pode ser mantida, retirada ou até mesmo melhorada.

Considerações Finais

A execução da Melhoria de Processo de Software dentro de um ambiente ágil é muito distinta daquela vista nos modelos tradicionais. Essa mudança não reflete apenas a mudança cultural, mas também a necessidade de novas formas de melhoria de processo que são inerentes ao desenvolvimento ágil de software.

O pouco formalismo, aliado a uma forma não sistemática da realização desse tipo de MPS, o torna de difícil convivência com estruturas tradicionais e partes interessadas de mais alto nível, se tornando, muitas vezes, arriscado para o projeto. A falta de um mecanismo eficiente para que a organização se aproveite da experiência obtida por cada time é também um ponto crítico nas relações de MPS tradicionais versus ágeis.

Esta forma não-tradicional de MPS é focada na melhoria do comportamento em ambientes de mudanças constantes, aproveitando ao máximo a experiência de cada um dos indivíduos e a agilidade com que cada um dos times pode se auto-organizar para se adaptar a uma situação. ●

Referências

[Basili & Rombach 1987] Basili, V. R. & Rombach, H. D (1987) Tailoring the Software Process to Project Goals and Environments. In: The proceedings of the 9th International Conference on Software Engineering. March 1987. Monterey, CA. Pp. 345-357.

[Beck & Andres 2004] Beck, K. & Andres, C. (2004) Extreme Programming Explained: Embrace Change. Second Edition. Addison-Wesley. Boston. pp 29.

[Boehm 1988] Boehm, B. (1988) A Spiral Model of Software Development and Enhancement Computer, Vol. 21, 5 (5), May 1988, pp. 61-72.

[Boehm & Turner 2004] Boehm, B.; Turner, R. (2005) Balancing agility and discipline: A guide for the perplexed (1st ed., pp. 165-194, Appendix A). Addison Wesley.

[Boehm & Turner 2005] Boehm, B. & Turner, R. (2005) Management Challenges to Implementing Agile Processes in Traditional Development Organizations. IEEE Software, Vol. 22(5), September–October. pp. 30-39.

[Cohn & Ford 2003] Cohn, M. & Ford, D. (2003). Introducing an Agile Process to an Organization. Computer, Vol. 36 (6), June, pp. 74-78.

[De Marco 1982] De Marco, T. (1982) Controlling Software Projects: Management, Measurement and Estimation. Yourdon Press. New York. pp-296.

[Deming 1990] Deming, W. E. (1990) Out of the Crisis. 10° Printing ed. Massachussetts Institute of Technology, Center for Advanced Engineering Study. Cambridge. pp- 507.

[Fenton & Pfleeger 1997] Fenton, N. & Pfleeger, S. L. (199). Software Metrics: A Rigorous & Practical Approach. Second Edition ed. PWS Publishing Company. Boston. pp- 63.

[Humphrey 1995] Humphrey, W. S (1995) A Discipline for Software Engineering. Addison Wesley, Longman. pp. 242 .

[Koch 2005] Koch, A. S. (2005) Agile Software Development: Evaluating the Methods for Your Organization. Artech House. Boston.

[Malouin & Landry 1983] Malouin, J. L. & Landry, M. (1983) The Miracle of Universal Methods in Systems Design. Journal of Applied Systems Analysis, Vol. 10, pp. 47–62.

[McFeeley 1996] McFeeley, R. (1996) "IDEAL: A User's Guide for Software Process Improvement" CMU-SEI.

[Mnkandla & Dwolatzky 2004] Mnkandla, E.; Dwolatzky, B. (2004) Balancing the human and the engineering factors in software development. Proceedings of the IEEE AFRICON 2004 Conference (pp. 1207-1210).

[Rico 2004] Rico, D. F. (2004) ROI of Software Process Improvement: Metrics for Project Managers and Software Engineers. J. Ross Publishing. Florida, U.S.A. pp - 218.

[Salo 2007] Salo, O. (2007) Enabling Software Process Improvement in Agile Software Development. PHD Thesis.

[Schuh 2004] Schuh, P. (2004) Integrating agile development in the real world (pp. 1-6). MA: Charles River Media.

[Vodde 2006] Vodde, B. (2006) Nokia Networks and Agile Development: in EuroMicro Conference.

Implantação de Processo

Os desafios da implantação de um processo de software

De que se trata o artigo?

Neste artigo veremos quais são os principais desafios e benefícios da implantação de um processo de desenvolvimento de software. Serão descritos aspectos importantes do processo que foi definido, bem como as ferramentas que foram desenvolvidas para dar suporte à sua implantação.

Para que serve?

A implantação de um processo de desenvolvimento tem se mostrado como um fator primordial de sucesso para as empresas de software. Entretanto, para ser um projeto de sucesso, deve ser planejado para refletir a realidade e dinâmica da empresa, caso contrário pode se tornar algo oneroso e difícil de manter.

Em que situação o tema é útil?

Para as empresas que estão planejando implantar um processo de desenvolvimento de software.

Edite Martins

edite.martins@dextra-sw.com

Engenheira de Computação pela Unicamp, com especialização em Administração de Empresas pela FGV-SP. Trabalha na área de TI há 16 anos, atuando como Gerente de Projetos e Gerente de Qualidade em empresas de desenvolvimento de software sob medida. Já atuou no gerenciamento de projetos de desenvolvimento de sistemas para grandes empresas como Natura, BankBoston, Globosat, entre outros. Recentemente gerenciou o processo de avaliação MPS. BR nível F da Dextra Sistemas.

A necessidade de um processo de software

O processo de software é uma seqüência lógica de atividades com o objetivo final de produzir ou evoluir um produto de software. Essas atividades englobam a especificação dos requisitos, análise e design, implementação, testes e implantação, e caracterizam-se pela interação de ferramentas, pessoas e métodos.

Estabelecer um processo significa definir todas as atividades a serem realizadas, bem como a seqüência e dependência entre elas, como as atividades serão executadas e quem é responsável por cada uma. A execução bem sucedida de um processo requer um conjunto de ferramentas de apoio e, principalmente, a conscientização da equipe sobre a necessidade do processo.

A implantação de um processo de software em uma empresa é um investimento custoso, porém tem se mostrado o fator primordial para o sucesso das empresas no alcance de seus objetivos, principalmente aqueles que se referem a satisfação dos clientes e previsibilidade de custo e prazo.

O estabelecimento de um processo, bem definido, diminui a dependência da empresa em relação às pessoas e possibilita a disseminação de melhores práticas, documentos e conhecimento.

Atualmente, existem vários modelos de processo de software e um dos mais conhecidos e utilizados é o RUP – Rational Unifed Process, comprado pela IBM.

O RUP é baseado em orientação a objetos e documentado utilizando a linguagem UML – Unified Modeling Language. É considerado um processo pesado, que preferencialmente deve ser utilizado em grandes projetos e equipes. Porém, como é fortemente customizável, é possível adaptá-lo para qualquer empresa.

Quando o RUP foi lançado, grandes empresas tentaram aplicá-lo em suas áreas de desenvolvimento de software. Foram realizados grandes investimentos e nem todos obtiveram o sucesso almejado. O principal problema foi acreditar que o RUP poderia ser utilizado da forma como estava definido sem que fosse gasto esforço e tempo para identificar de que forma o RUP se encaixaria na estrutura da empresa.

O caso que iremos detalhar é exatamente como a Dextra Sistemas (www.dextra.com.br) extraiu do RUP o que era realmente aplicável às suas necessidades e como esse esforço e investimento culminou na melhoria do resultado dos seus projetos e na avaliação MPS.BR nível F obtida em Março de 2009.

Definição do processo

A Dextra teve um crescimento muito expressivo nos anos de 2006 e 2007 e com isso surgiu a necessidade da definição e implantação de um processo mais formal. A empresa já contava com o ProUD (Processo Unificado Dextra) que se baseava fortemente no RUP. A versão 1.0 do ProUD era bastante enxuta e se aplicava muito bem ao tamanho e características da empresa e de suas equipes de trabalho até então.

O objetivo da evolução e melhoria do processo da Dextra era ter um nível maior de formalização das atividades e papéis, bem como coletar e analisar indicadores de projeto que dessem a visibilidade necessária em relação à qualidade do produto, pontualidade nas entregas e previsibilidade de custo.

Diante disso, a estratégia da empresa foi novamente aplicar o RUP, utilizando o feedback já obtido no uso do ProUD 1.0 em diversos projetos. A principal meta da Dextra era definir um processo que representasse o que a empresa fazia de melhor nos seus projetos e corrigisse o que ainda poderia melhorar. Com esse foco o plano de trabalho contou com a formação de grupos de trabalho formados por membros das equipes de projetos. Os líderes de cada grupo eram membros do SEPG – Software Engineer Process Group e davam o direcionamento necessário às atividades. O resultado de um ano de trabalho foi o ProUD 2.0.

O processo foi dividido em quatro grandes fases, como se vê na **Figura 1**. Na versão 1.0 do ProUD, os nomes das fases eram exatamente iguais às do RUP – Concepção, Elaboração, Construção e Transição. Porém, com a utilização do processo percebeu-se que havia atividades que no RUP eram feitas na fase de Elaboração e que na Dextra fazia mais sentido que fossem executadas na fase de Construção e vice-versa. Desta forma, quisemos desvincular as fases do ProUD das fases do RUP, criando fases e atividades dentro delas que representam a realidade da empresa e a melhor forma de trabalho de suas equipes. Segue abaixo uma breve descrição de cada uma das fases:

Figura 1. Fases do ProUD 2.0

Planejamento

Nesta fase ocorre a definição do escopo do projeto e de suas fronteiras, assim como o planejamento detalhado de sua execução. São gerados todos os planos do projeto – plano de comunicação, riscos, infra-estrutura, recursos humanos, gestão de configuração e etc. Todos os compromissos internos e com o cliente são formalizados e aprovados. Esta fase recebe muitos insumos da fase de proposta, que também segue um processo bem definido e formal. Parte das atividades desta fase é feita durante a fase de Concepção do RUP.

Refinamento da Solução

Nesta fase os requisitos levantados durante a fase de proposta são detalhados, bem como a arquitetura final da solução e os aspectos de integração com outros sistemas. Deve-se, também, fazer uma extensiva análise dos riscos do projeto associados aos requisitos e à tecnologia utilizada. O resultado desta fase é o detalhamento inicial dos requisitos, protótipos de telas e a definição da arquitetura do sistema. As atividades desta fase são executadas parte na fase de Concepção e parte na fase de Elaboração do RUP.

Execução

O foco nessa fase é dado à implementação das funcionalidades do sistema. A implementação desenvolve-se de maneira iterativa e incremental. A cada iteração é implementado um conjunto de casos de uso definidos previamente, tendo passado pelas fases de análise, projeto, codificação, testes e integração.

O conceito de iterações faz com que os riscos do projeto sejam reduzidos, pois é possível liberar funcionalidades para os usuários antes de concluir todo o projeto, verificando a adequação do sistema aos requisitos funcionais e técnicos (escalabilidade, segurança e confiabilidade).

Ainda nessa fase, os componentes de software desenvolvidos devem ser implantados nos ambientes de homologação e de produção do cliente. Os treinamentos planejados são preparados e aplicados aos usuários.

As atividades desta fase correspondem à fase de Construção e Transição do RUP.

Encerramento

Esta fase se inicia após o aceite formal do cliente. São gerados documentos de encerramento do projeto, indicadores, lições aprendidas e reunião de fechamento com a equipe. Nesta fase inicia-se a atividade de acompanhamento da operação (operação assistida) e o período de garantia do produto.

As atividades desta fase são realizadas na fase de Transição do RUP.

As disciplinas definidas para o ProUD são muito similares às do RUP e largamente executadas durante os ciclos iterativos da fase de Execução. São nas iterações que os requisitos são detalhados, projetados, implementados, testados e integrados ao sistema para homologação pelo cliente. Acreditamos que quanto antes o cliente receber o produto, mesmo que uma versão parcial, mais chances o produto final terá de atender as expectativas do cliente. As disciplinas implementadas no ProUD estão descritas abaixo:

- **Requisitos:** tem como objetivo identificar os requisitos do sistema através do entendimento das necessidades dos stakeholders. O escopo identificado deve ser mantido e controlado no decorrer do projeto;
- **Análise e Projeto (ou Análise e Design):** tem como objetivo elaborar a arquitetura do sistema e garantir que a solução seja implementada no decorrer do projeto;
- **Implementação:** tem como objetivo implementar os requisitos identificados, seguindo a arquitetura definida, gerando código fonte testado unitariamente;
- **Testes:** tem como objetivo garantir que o produto que será entregue ao cliente implementa corretamente o que foi solicitado. Além disso, provê a garantia de que mudanças no sistema não introduziram defeitos ou efeitos indesejados;
- **Implantação/Integração:** tem como objetivo preparar o produto para ser liberado/entregue ao cliente e disponibilizá-lo no ambiente de homologação;
- **Gerência de Projetos:** tem como objetivo planejar e controlar o escopo do projeto, gerenciando as expectativas da equipe, do cliente e da alta direção da empresa;
- **Gerência de Configuração e Versão:** tem como objetivo manter um conjunto consistente de produtos de trabalho, à medida que vão sendo produzidos e alterados.

A **Figura 2** mostra como as disciplinas estão relacionadas dentro de um ciclo iterativo da fase de Execução. As disciplinas de Gestão de Projetos e Gestão de Configuração e Versão permeiam toda a cadeia de desenvolvimento, pois são disciplinas de suporte ao desenvolvimento de software.

Toda definição do processo foi feita utilizando a ferramenta OpenSource EPF Composer, que possibilita diferentes visões aos usuários, facilitando a busca e visualização das informações. O EPF Composer publica o processo em formato de um website que normalmente fica disponível na Intranet da empresa com acesso a todos os seus colaboradores.

As fases do ciclo de vida são exibidas em forma de diagramas, onde as atividades estão agrupadas por papel. Cada atividade do diagrama é um link para a sua descrição, onde estão detalhados os artefatos de entrada, de saída, templates dos produtos de trabalho e papéis responsáveis pela atividade.

A **Figura 3** mostra uma tela do ProUD 2.0 publicado através do EPF. Pode-se notar que há uma barra de navegação onde o usuário pode percorrer o processo pelo ciclo de vida, pelas disciplinas, pelos papéis e produtos de trabalho. Além de ter um acesso direto a todos os templates de documentos e orientações de forma rápida e direta.

Figura 2. Ciclo de Vida Iterativo da fase de Execução

Implementando o MPS.BR Nível F

Um dos objetivos do projeto de melhoria e evolução do ProUD, era a de implementar também um modelo de qualidade reconhecido no mercado de software. A empresa escolheu o MPS.BR por ser um programa brasileiro, voltado para a realidade do mercado de pequenas e médias empresas de desenvolvimento de software. O MPS.BR é baseado no CMMI, nas normas ISO/IEC 12207 e ISO/IEC 15504 e o Nível F corresponde ao CMMI 2.

O trabalho de implementação do MPS.BR ocorreu em paralelo à evolução do ProUD, e direcionou por vezes, o seu foco. O nível de maturidade F (Gerenciado) é composto pelos processos de Gerência de Projeto, Gerência de Requisitos, Aquisição, Gerência de Configuração, Garantia da Qualidade e Medição. O processo de Aquisição só é implementado por empresas que terceirizam o desenvolvimento de software.

Todos os processos têm integração entre si, por isso era necessário pensar em uma infra-estrutura de ferramentas que possibilitasse a agregação das informações e, desta forma, minimizasse o custo com replicação de dados. A solução escolhida foi a implementação e customização de algumas ferramentas que permitiram a implementação do modelo e que hoje são a grande base dos projetos de desenvolvimento de software da empresa.

TRAC

O TRAC é uma ferramenta OpenSource, com interface web para gerenciamento e controle de mudanças em projetos de desenvolvimento de software. Além de possibilitar a documentação colaborativa através do Wiki, tem integração com o subversion, permitindo o rastreamento de mudanças nos artefatos do projeto.

O TRAC foi customizado de forma a ser o grande centralizador de informações do projeto. As informações estão

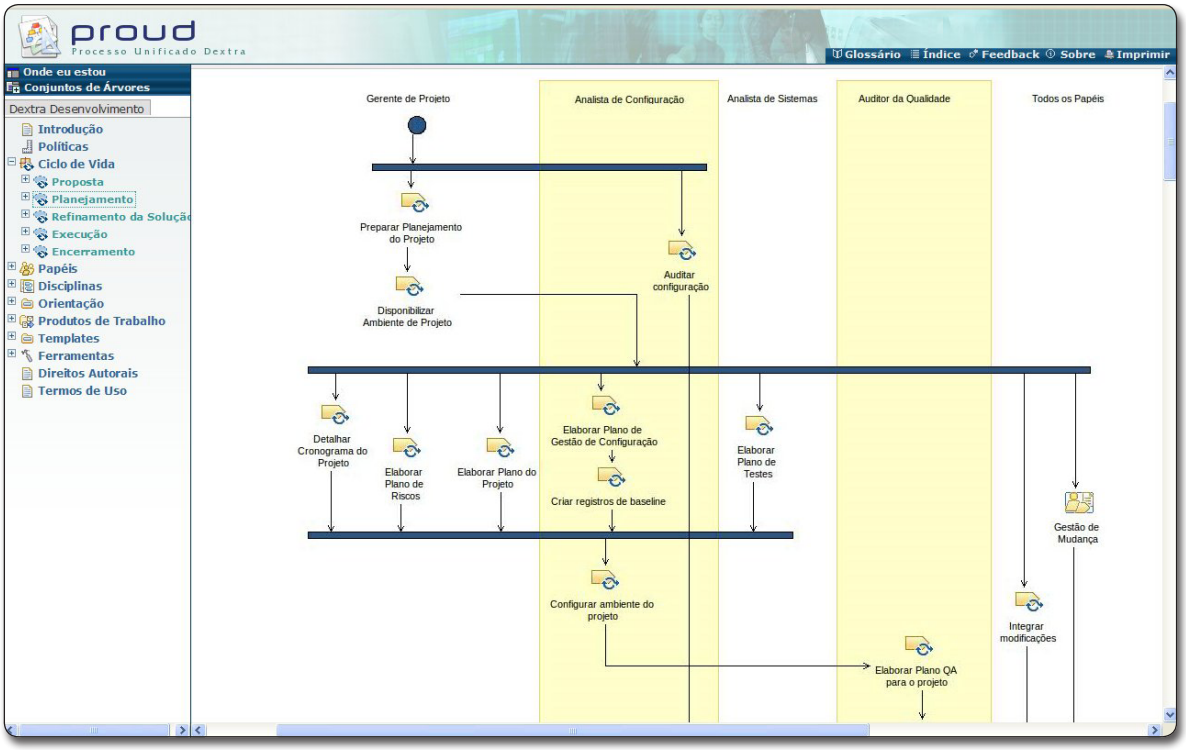


Figura 3. Site do ProUD 2.0 publicado pelo EPF Composer

no Trac, ou em forma de wiki, ou em forma de tickets. Para que a documentação do projeto pudesse ser wiki, foi feita uma customização onde os wikis visualizados no site são obtidos diretamente do subversion e não na base de dados do Trac. Desta forma é possível manter toda a documentação disponível via web, porém com controle formal de versão. São exemplos de documentos que estão hoje em formato wiki: plano do projeto, atas de reunião, documento de visão, glossário, dicionário de dados, documento de arquitetura, casos de uso entre outros. O formato wiki minimiza vários problemas referentes a resolução de conflitos e merge de arquivos no subversion, devido ao fato de ser um arquivo texto. A Figura 4 é um exemplo de Documento de Arquitetura disponível no Trac.

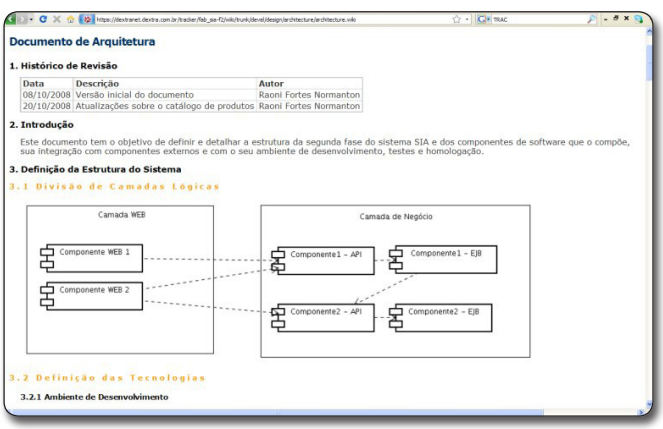


Figura 4. Documento de Arquitetura em Wiki

Para que o registro das demais informações do projeto pudesse ser realizado através de tickets, estes foram customizados em vários tipos. Como o TRAC oferece a referência cruzada entre seus elementos, é possível definir e manter toda a rastreabilidade necessária ao projeto e também ao MPS.BR. As seguintes informações são registradas no TRAC em forma de tickets: requisitos, casos de uso, riscos, mudanças, defeitos, não-conformidades, ações gerenciais e tarefas (importadas do MSProject ou criadas manualmente).

DET

A DET – Dextra Estimating Tool é uma ferramenta de desenvolvimento próprio para realizar as estimativas de todos os projetos de desenvolvimento de software da empresa. As estimativas são baseadas em tamanho funcional e o resultado é a estimativa de esforço para todas as atividades definidas no processo. Além disso, é possível também realizar a identificação de riscos, requisitos não-funcionais e atividades suplementares. A ferramenta exporta, seguindo templates padrões, a lista de requisitos funcionais e não-funcionais do projeto, a lista de riscos e o cronograma detalhado do projeto. Estas informações são importadas no TRAC, onde serão gerenciadas ao longo do projeto.

PMA

O PMA – Sistema de Gestão de Projetos é também uma ferramenta de desenvolvimento próprio que gerencia o apontamento de horas das equipes do projeto e o seu custo. O apontamento é realizado de acordo com as atividades do processo, desta forma é possível extrair informações sobre a

acuracidade das estimativas e gerar dados para a revisão do método sempre que necessário. Além disso, a ferramenta controla o custo dos recursos, gerando as medidas necessárias para o cálculo de indicadores referentes ao índice de performance de custo do projeto.

A seguir vamos descrever sucintamente, como implementamos cada um dos processos tendo como base o conjunto de ferramentas descritas acima:

Gerência de Projetos

Propósito: estabelecer e manter planos que definem as atividades, recursos e responsabilidades do projeto, bem como prover informações sobre o andamento do projeto que permitam a realização de correções quando houver desvios significativos no desempenho do projeto.

Nós já praticávamos a gerência de projetos desde o ProUD 1.0, e o trabalho de melhoria de processo foi focado na padronização dos documentos e artefatos produzidos e a sua disponibilização para a equipe e o cliente. A documentação passou a ser disponibilizada na ferramenta TRAC de cada projeto, em formato wiki. Foram definidas formas obrigatórias de acompanhamento do projeto cujo registro são atas de reunião e relatórios de status. O acompanhamento do projeto tem o objetivo de controlar e monitorar todos os itens que foram detalhados durante o planejamento e confrontá-los com o realizado. Um fator essencial para a maturidade no processo de gerência de projetos é conseguir fazer uma análise crítica do que está sendo executado em relação ao planejado em termos de custo, escopo, prazo e riscos e verificar se o projeto ainda continua viável ou que ações devem ser tomadas para que o projeto seja finalizado com sucesso.

Gerência de Requisitos

Propósito: gerenciar os requisitos dos produtos e componentes do produto do projeto e identificar inconsistências entre os requisitos, os planos do projeto e os produtos de trabalho do projeto.

O processo de gerência de requisitos era largamente praticado na empresa desde o ProUD 1.0 e o trabalho de melhoria de processo foi focado na padronização dos documentos gerados por esta disciplina. Os documentos de Visão e de Casos de Uso são disponibilizados no TRAC do projeto, em formato wiki. Além disso, o uso da DET passou a ser obrigatório para todos os projetos de desenvolvimento de software, padronizando as estimativas

Contudo, a maior mudança neste processo foi a definição do mecanismo de gerenciamento de mudanças dos requisitos. Foi redefinido um processo de análise de solicitações de mudanças, onde o uso dos mecanismos de rastreabilidade implementados pelo TRAC é imprescindível. Com a realização de uma análise de impacto efetiva das solicitações de mudança é possível controlar o escopo, custo e prazos do projeto de forma objetiva e transparente para a equipe e cliente.

Gerência de Configuração

Propósito: estabelecer e manter a integridade de todos os produtos de trabalho de um processo ou projeto e disponibilizá-los a todos os envolvidos.

Apesar da empresa já utilizar o TRAC integrado ao subversion, o que já estabelece uma base bastante sólida para o gerenciamento de configuração e versão, a evolução deste processo no ProUD 2.0 foi grande, e com resultados surpreendentes para os projetos. Foi criado formalmente o papel de Analista de Configuração e foram redefinidas as estruturas de diretório, padrão de nomenclatura, registros de baselines, regras de integração de arquivos ao repositório e checklists para auditorias do sistema de configuração. O objetivo principal deste processo é manter o repositório íntegro e gerar informações para que se possa extrair a rastreabilidade entre requisitos e código fonte e vice-versa. Para que a rastreabilidade seja construída e mantida é necessário que o sistema de configuração esteja fortemente amarrado e com regras rígidas de integração de modificações ao repositório. As customizações realizadas no TRAC e a integração com o subversion foram fatores chave para que se alcançasse a maturidade exigida neste processo.

Garantia da Qualidade

Propósito: assegurar que os produtos de trabalho e a execução dos processos estejam em conformidade com os planos e recursos predefinidos.

Este processo foi criado no ProUD 2.0. A definição do processo consistiu em criar as atividades de auditoria, definir políticas, periodicidade e checklists de avaliação. As auditorias são realizadas no decorrer do projeto e sempre que há entregas para o cliente. As não-conformidades são registradas como tickets no TRAC e tem uma máquina de estados definida para seu acompanhamento. Todas as não-conformidades têm um prazo definido para resolução e caso não sejam solucionadas são escaladas para alta gerência da empresa.

As auditorias de qualidade são uma forma da empresa garantir que os projetos estão seguindo o processo definido, gerando os produtos de trabalho esperados e dentro dos padrões estabelecidos. As auditorias de qualidade em produtos de trabalho que serão entregues ao cliente são de extrema importância para a empresa, pois garantem uma uniformidade em termos de documentação além de aumentar a qualidade do produto gerado.

Medição

Propósito: coletar, analisar e relatar os dados relativos aos produtos desenvolvidos e aos processos implementados na organização e em seus projetos, de forma a apoiar os objetivos organizacionais.

Todos os indicadores da empresa foram refeitos no ProUD 2.0. Foi utilizado o método GQM – Goal Question Metrics, onde a diretoria define os objetivos a serem alcançados, perguntas são feitas sobre como os objetivos podem ser atingidos e medidas são identificadas a partir das questões. Foi montada uma

planilha com 22 indicadores, onde são definidos os métodos de coleta e análise, bem como sua periodicidade. Alguns dos indicadores são CPI, SPI, satisfação do cliente, desvio de estimativa entre outros. A fácil extração das medidas para o cálculo dos indicadores só é possível pelo fato destas medidas serem registradas nas ferramentas DET, TRAC e PMA.

A coleta e análise de indicadores dos projetos tem se mostrado como um dos principais benefícios obtidos no ProUD 2.0. A padronização na coleta das medidas em todos os projetos trouxe um ganho expressivo de visibilidade em termos de custos, prazos, qualidade, desvio de estimativas entre outros. É evidente o ganho que este processo trouxe para a alta gerência da empresa.

A avaliação MPS.BR nível F

O ProUD 2.0 foi lançado oficialmente na Dextra em Setembro de 2008. Foram realizados diversos treinamentos e workshops na empresa para divulgação e capacitação das equipes. Todos os projetos que começaram após o lançamento utilizaram a nova versão do processo.

A avaliação foi marcada para Fevereiro de 2009, apenas 5 (cinco) meses após o lançamento da nova versão do processo. A avaliação envolveu 4 (quatro) projetos e cerca de 35 (trinta e cinco) pessoas que estavam divididas entre os projetos que seriam avaliados. A avaliação é composta por uma fase de análise de documentos e uma fase posterior de entrevistas com os membros das equipes. O resultado da avaliação foi excelente e a empresa foi recomendada para o Nível F de maturidade do MPS.BR, tendo como principais pontos fortes: a) o conjunto de ferramentas adotado, que automatiza as atividades e traz maior controle e ganho de produtividade da equipe e b) o envolvimento das equipes dos projetos com o processo.

O que é interessante ressaltar é que esses dois pontos foram justamente a maior preocupação da empresa desde o início: envolver a empresa na definição do processo e investir em ferramentas que suportassem a sua implantação.

O que a experiência tem nos mostrado é que a implantação de um processo de software será sempre custosa e representa um investimento financeiro importante para a empresa. Por outro lado, a empresa terá maior sucesso e retorno deste investimento se o processo for elaborado e construído dentro da própria empresa. Um processo não se impõe às pessoas, ele se constrói junto com elas.

Links

Site do Softex Brasil – www.softex.br/mpsbr/

Site onde é possível obter informações sobre o RUP – www.wthreex.com/rup/

Site da ferramenta TRAC – trac.edgewall.org/

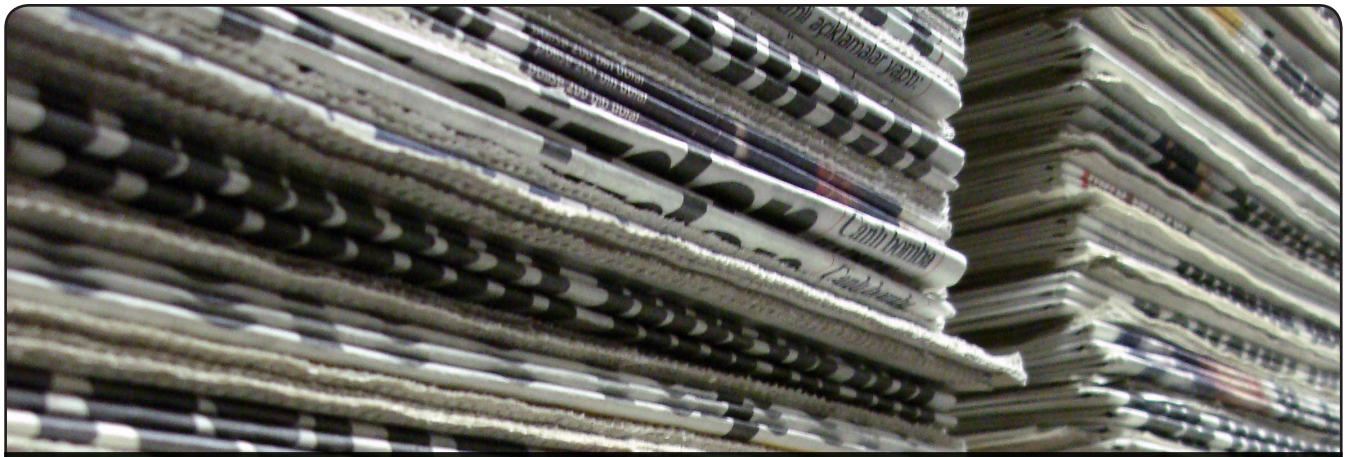
Site da ferramenta EPF Composer – www.eclipse.org/epf/

Publicação da avaliação MPS.BR F realizada na Dextra em Março/2009 – www.softex.br/mpsbr/_avaliacoes/avaliacao.asp?id=2465

Dê seu feedback sobre esta edição!

A Engenharia de Software Magazine tem que ser feita ao seu gosto.
Para isso, precisamos saber o que você, leitor, acha da revista!
Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/esmag/feedback



Gerência de Configuração

Definições Iniciais

De que se trata o artigo?

Neste artigo são apresentados os principais conceitos de gerência de configuração de software e onde esses se encaixam no processo de desenvolvimento de software. Também são apresentadas descrições das principais tarefas de gerência de configuração de software, tais como definição e implementação do processo, identificação da configuração, controle da configuração, relato da situação da configuração, avaliação da configuração e controle de subcontratados e fornecedores.

Para que serve?

Para que exista um maior controle no processo de desenvolvimento, e não haja inconsistências nos itens considerados importantes para o projeto, é necessário que sejam estabelecidas normas para controlar a criação e alteração dos itens de configuração.

Em que situação o tema é útil?

O tema é útil para qualquer equipe envolvido no desenvolvimento de software que necessite de um controle mínimo sobre os diferentes artefatos que são gerados ao longo do desenvolvimento.

Durante o desenvolvimento de software, uma grande quantidade de informações é produzida, tais como: especificações, planos de projeto, arquivos de código fonte, casos e planos de testes, manuais, arquivos de dados, entre outros. Cada um desses documentos produzidos poderá ser considerado um item de configuração de software. A configuração de software é composta pelos itens de configuração produzidos durante o processo de engenharia de software, ou seja, no processo de desenvolvimento disciplinado de sistemas (Pressman, 2005).

Os itens de configuração podem sofrer alterações ao longo do ciclo de vida do software, gerando novas versões, e até mesmo a criação de novos itens. Para que exista um maior controle no processo de desenvolvimento, e não haja inconsistências nos itens considerados importantes para o projeto, é necessário que sejam estabelecidas normas para controlar a criação e alteração dos itens de configuração. A essas normas dá-se o nome de gerência de configuração de software (Tichy, 1994).

Thaís Miranda Cia

thaismcia@gmail.com

É mestre em Ciência da Computação e Matemática Computacional pelo ICMC-USP.

A gerência de configuração vem sendo estudada desde os anos sessenta (Berlack, 1992). Inicialmente, era aplicado da mesma forma para software e hardware, sendo que no final dos anos setenta já havia padrões de gerência de configuração específicos para software.

A gerência de configuração de software é um processo abrangente, ao mesmo tempo técnica e gerencial, que se aplica a todas as atividades de engenharia de software, e pode ser visto como um dos principais elementos que compõem o sistema de garantia de qualidade de uma empresa de informática (Leblang, 1988). O processo visa identificar e definir os itens considerados relevantes ao projeto, controlar as modificações dos itens, registrar e reportar a situação dos itens e das requisições das alterações, garantir a integridade e consistência dos itens e controlar o armazenamento, manipulação, liberação e entrega dos itens (ISO/IEC 12207, 1995).

Os ganhos em tempo e qualidade pela aplicação da gerência de configuração são comprovados e mensuráveis. Esses ganhos podem ser reconhecidos constatando-se que o uso de gerência de configuração de software facilita a comunicação entre as equipes de desenvolvedores relatando a situação do software a cada momento, assim como as alterações que foram efetuadas. Isso traz como conseqüência a redução no esforço necessário para efetuar alterações e a redução no número de erros. O resultado final é melhor cumprimento dos cronogramas, redução nos custos e melhora na qualidade do software (Berlack, 1992).

Algumas normas e modelos internacionais como a ISO/IEC 12207, ISO/IEC 15504 e o CMMI citam a gerência de configuração de software como requisito para que uma empresa inicie a melhoria de qualidade do processo de desenvolvimento de software. Dessa forma a gerência de configuração de software vem ocupando cada vez mais espaço no escopo de desenvolvimento de software (Berczuk, 2003).

A implantação da Gerência de Configuração não costuma ser fácil (Micallef, 1996). Isso ocorre pela grande diversidade de normas, padrões e modelos que não seguem um mesmo padrão na definição das atividades. O custo de implantação também costuma ser bastante alto, impossibilitando que instituições com menos recursos realizem a gerência de configuração de software (Berczuk, 2003).

Além disso, devido ao aumento de interesse pela implantação da gerência de configuração de software, existem atualmente no mercado várias ferramentas que se propõem a auxiliar a execução de práticas e processos de gerência de configuração (Lampen, 1988). No entanto, a maioria dessas ferramentas cobre apenas uma parte das atividades necessárias para implantação da gerência de configuração, confundindo muitas vezes o usuário leigo que vê nessas ferramentas a solução para todos os seus problemas de gerência de configuração.

Neste contexto, neste artigo são apresentados os principais conceitos de gerência de configuração de software e onde esses se encaixam no processo de desenvolvimento de software.

Também são apresentadas descrições das principais tarefas de gerência de configuração de software, tais como definição e implementação do processo, identificação da configuração, controle da configuração, relato da situação da configuração, avaliação da configuração e controle de subcontratados e fornecedores.

Conceitos Fundamentais

O desenvolvimento de um software pode ser organizado de várias formas. A cada uma dessas formas de organização dá-se o nome de um paradigma de ciclo de vida. Os paradigmas mais estudados são o clássico, a prototipação, o modelo espiral, as técnicas de quarta geração e os modelos evolutivos tais como RUP e XP (Pressman, 2005).

Um processo de desenvolvimento de software, independente do paradigma de ciclo de vida adotado, inclui as fases de engenharia de sistemas, análise de requisitos, projeto de software, codificação, testes e manutenção (Pressman, 2005). Na **Tabela 1** é apresentada uma breve descrição de cada uma dessas fases.

Fases	Descrição
Engenharia de Sistemas	Coleta dos requisitos em nível do sistema, com uma pequena quantidade de projeto e análise de alto nível.
Análise de Requisitos	Compreensão do domínio da informação através dos requisitos coletados na fase anterior.
Projeto de Software	Desenvolvimento de quatro atributos distintos do software: estrutura de dados, arquitetura de software, detalhes procedimentais e caracterização de interfaces.
Codificação	Tradução do projeto de software numa forma legível para a máquina.
Testes	Realização de testes dos programas. Esses testes concentram-se nos aspectos lógicos internos do software e nos aspectos funcionais externos.
Manutenção	Modificações após o software ser liberado. Essas mudanças ocorrem por erros, adaptações, novos ambientes e novas funcionalidades.

Tabela 1. Fases do Desenvolvimento de Software

Durante o processo de desenvolvimento de um software, uma grande quantidade de itens de informação é produzida. Alguns desses itens são selecionados de acordo com sua relevância e necessidade de controle tanto de versão quanto de mudança. Esses itens selecionados são chamados itens de configuração de software e o conjunto dos mesmos compõe a configuração de software (Mahler, 1994).

A criação e as alterações de cada item de configuração de software devem ser acompanhadas e controladas pelo gerente do projeto. Para isso, é necessário o estabelecimento de pontos bem definidos dentro do processo de desenvolvimento do software: as Linhas de Referência (baselines). Esses pontos podem ocorrer ao final de cada uma das fases do processo de desenvolvimento de software, ou de algum outro modo definido pela gerência.

Nos pontos estabelecidos pelas linhas de referência, os itens de configuração devem ser identificados, analisados, corrigidos, aprovados e armazenados como uma configuração de software. Os itens aprovados são armazenados em um local sob controle de acesso, denominado repositório dos itens de configuração. Esses itens só poderão ser alterados após uma

solicitação de mudança formalmente aprovada pelo gerente de configuração. Essa é uma forma de prover controle sobre a situação de cada um dos itens de configuração, evitando inconsistências.

O método utilizado para trabalhar com itens de configuração que já estão no repositório é chamado de Check In/Check Out (Bersoff, 1979), ou seja, conferência na entrada e conferência na saída. Quando for desejada uma alteração em algum item de configuração do repositório, uma cópia do item é colocada numa área de trabalho do desenvolvedor ("check out"). A partir desse momento, nenhum outro desenvolvedor poderá alterar o mesmo item: a isso dá-se o nome de controle de concorrência. Dentro de sua área, o desenvolvedor tem total liberdade de trabalho. Após o final das alterações no item de configuração, ele será revisado e recolocado no repositório ("check in"). Nesse momento, uma nova baseline poderá ser estabelecida, de modo que uma nova configuração, contendo o item alterado, seja formada e armazenada no repositório (**Figura 1**).

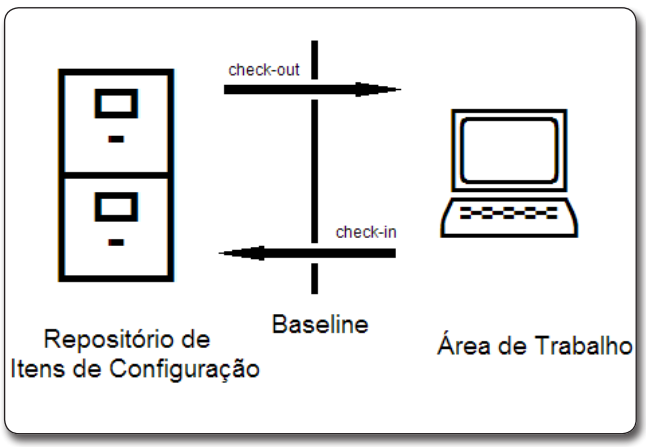


Figura 1. Controle de concorrência (check-in / check-out)

Depois do armazenamento e da definição da baseline, o acesso é liberado, permitindo que outros desenvolvedores possam acessar e também executar alterações sobre esse item de configuração.

Tarefas de Gerência de Configuração de Software

As tarefas de gerência de configuração de software, que respondem às questões apresentadas na **Tabela 2** são descritas a seguir.

Tarefa 1 - Definição e Implementação do Processo

O processo de gerência de configuração de software deve ser estabelecido de acordo com uma política organizacional definida. Para cada projeto de desenvolvimento de software é preciso elaborar um plano de gerência de configuração, respeitando o processo de gerência de configuração da organização (IEEE Std 828, 1998). Quaisquer desvios do processo devem estar documentados no plano de gerência de configuração.

Deve ser designado um grupo para ser responsável pelo controle da configuração do projeto. Os membros do grupo de gerência de configuração devem ser treinados nos objetivos, procedimentos, e métodos para desenvolver as atividades de gerência de configuração de software.

Também é preciso prover e adequar recursos para que seja possível realizar as atividades de gerência de configuração, tais como a disponibilização de ferramentas para dar suporte às atividades de gerência de configuração. Para isso deve ser designado um gerente com responsabilidades específicas de gerência de configuração de software.

Tarefa 2 - Identificação da Configuração

O primeiro passo para a identificação é selecionar os itens a serem gerenciados. Como exemplo, é apresentado abaixo uma série de itens sugeridos por Pressman (Pressman, 2005):

1. Especificação do Sistema
2. Plano de Projeto de Software
3. Especificação de Requisitos do Software
4. Manual Preliminar do Usuário
5. Especificação do Projeto
   a. Descrição do Projeto de Dados
   b. Descrição do Projeto Arquitetural
   c. Descrições do Projeto Modular
   d. Descrições do Projeto de Interface
   e. Descrições de Objetos (se forem usadas técnicas orientadas a objetos)
6. Listagem do código-fonte
7. Planos, Procedimentos, Casos de Testes e Resultados Registrados
8. Manuais Operacionais e de Instalação
9. Programa Executável e Módulos Interligados
10. Descrição do Banco de Dados
    a. Esquema e estrutura de arquivo
    b. Conteúdo inicial

Tarefa	Questões
Definição e Implementação do Processo	Existe uma política organizacional definida? Qual o grupo responsável pelo controle da configuração? Quem será responsável pela elaboração do plano de gerência de configuração?
Identificação da Configuração	Como uma organização identifica quais itens entrarão na configuração do software?
Controle da Configuração	Quem tem a responsabilidade pela aprovação e pela determinação de prioridades para as mudanças? Como uma organização controla as várias versões geradas pelas mudanças feitas antes e depois que o software é liberado?
Relato da Situação da Configuração	Qual o mecanismo usado para avisar outras pessoas sobre mudanças que são feitas?
Avaliação da Configuração	Como se pode garantir que as mudanças foram feitas adequadamente?
Controle de Subcontratados e Fornecedores	Como garantir que módulos do sistema construídos por terceiros estejam corretos e coerentes com o restante do sistema?

Tabela 2. Tarefas de gerência de configuração de software

11. Manual do Usuário
12. Documentos de Manutenção
    a. Relatórios de problemas de software
    b. Solicitações de manutenção
    c. Pedidos de mudança
13. Padrões e procedimentos para engenharia de software
14. Ferramentas de produção de software (editores, compiladores, CASE, etc.)

É importante que seja efetuada uma seleção dos itens relevantes, porque uma super-documentação torna a gerência de configuração muito onerosa (Tuscany, 1987). Geralmente, devem sofrer gerência de configuração os itens mais usados no ciclo de vida, os mais genéricos, os mais importantes para a segurança, os itens projetados para reuso e os que podem ser modificados por vários desenvolvedores ao mesmo tempo (Bersoff, 1984). Somente os itens selecionados serão controlados, sendo que os outros itens poderão ser alterados livremente.

Após a seleção, deve-se descrever como esses itens se relacionam. Consideram-se cinco classes de relacionamento: equivalência (ex: BD em disco rígido e em CD), dependência (ex: a descrição do projeto modular é dependente da especificação do projeto), derivação (ex: código objeto é derivado do código fonte), sucessão (ex: a versão 1.2 é sucessora da versão 1.1) e variante (ex: versão para DOS ou para UNIX). A identificação desses relacionamentos é muito importante para a manutenção, pois permite que se localize rapidamente os itens afetados por cada alteração.

Depois de escolhidos os itens e estabelecidos os relacionamentos, deve-se criar um esquema de identificação dos itens com a atribuição de nomes únicos a cada um dos componentes, de forma que seja possível reconhecer a evolução de cada uma das versões dos componentes e a hierarquia existente entre componentes, a partir de seus nomes (Bersoff, 1979).

Um exemplo simples para um pequeno programa cuja sigla é "PROG" é apresentado na **Tabela 3**. O esquema de identificação utiliza a combinação de nome do projeto, tipo de item, nome do item e versão.

Após o estabelecimento do esquema de identificação, devem ser planejadas as baselines dentro do ciclo de vida do projeto. Geralmente, cria-se uma linha de referência ao final de cada fase do ciclo de vida do projeto e, periodicamente, depois de cada manutenção. Deve-se especificar quais itens serão revisados e armazenados em cada uma das linhas de referência planejadas. O último passo da identificação é descrever a maneira como os itens serão arquivados e recuperados do repositório.

Item	Projeto	Tipo	Nome	Versão	Nome completo
Especificação do Sistema	PROG	ES		1.1.0	PROG_ES_1.1.0
Plano de Trabalho	PROG	PT		1.1.0	PROG_PT_1.1.0
Especificação de Requisitos Alocados ao Software	PROG	ER		1.1.0	PROG_ER_1.1.0
Especificação de Projeto	PROG	EP		1.1.0	PROG_EP_1.1.0
Executável do Sistema	PROG	PF	EXE	1.1.0	PROG_PF_EXE_1.1.0
Plano e Casos de Testes	PROG	TT		1.1.0	PROG_TT_1.1.0
Nova versão do Programa	PROG	PF	EXE	1.1.1	PROG_PF_EXE_1.1.1

Tabela 3. Identificação dos itens de configuração

Tarefa 3 - Controle da Configuração

Dois controles básicos são instituídos no processo de gerência de configuração de software: Controle de Mudanças e Controle de Versões.

a) Controle de Mudanças

Durante o processo de desenvolvimento de software, mudanças descontroladas podem levar rapidamente ao caos (Pressman, 2005). Assim, deve ser instituído na organização um processo que combine procedimentos humanos e ferramentas automatizadas para proporcionar um mecanismo de controle das mudanças. Esse processo deve ser implementado depois que uma linha de referência for fixada - antes disso somente um controle de mudanças informal precisa ser aplicado (Bersoff, 1979; Bersoff, 1984; Pressman, 2005). A **Figura 2** (Pacheco, 1997) ilustra um processo de controle de mudanças que pode ser implementado para os itens que já passaram por uma linha de referência.

De acordo com esse processo de controle de mudanças, quando um pedido de mudança é solicitado, primeiramente ele deve ser analisado, gerando um relatório de mudanças. Esse relatório é encaminhado para avaliação; se aprovado, o relatório de mudança segue para o gerente de configuração. O Gerente de configuração controla o acesso aos itens no repositório, liberando-os para a equipe de desenvolvimento para que a mudança seja efetuada, e recebendo os itens, quando atualiza o repositório. Caso o pedido de mudança não seja aprovado, o relatório e o pedido são arquivados e é dado um retorno ao solicitante.

Esse controle possibilita que as mudanças sejam efetuadas, comunicadas e incorporadas de um modo disciplinado. Entretanto, para que esse controle ocorra de forma satisfatória, qualquer mudança que ocorra nos itens de configuração de software, após o estabelecimento de uma linha de referência, deve seguir efetivamente sempre o mesmo caminho (Bersoff, 1979).

No processo de controle de mudanças, as alterações aprovadas são efetuadas de maneira sincronizada. O objetivo dessa sincronização é evitar que duas pessoas efetuem, ao mesmo tempo, mudanças incompatíveis em um mesmo item, criando inconsistências (Bersoff, 1984). O método mais utilizado para evitar inconsistências é controlar o acesso ao repositório, de forma que, quando um desenvolvedor retira um item para alterações, ele bloqueia o acesso de escrita no item para os outros desenvolvedores (Mackay, 1995).

Os procedimentos de controle das mudanças asseguram que as mudanças em um software sejam feitas de modo controlado, permitindo-se prever o efeito das mesmas em todo o sistema (Leblang, 1997). Procedimentos formais de organização e de controle das mudanças no sistema permitem que os pedidos de alteração possam ser considerados em conjunto com outros pedidos (Honda, 1988). Desse modo, os pedidos similares podem ser agrupados, e os pedidos incompatíveis entre si ou com os objetivos do sistema identificados. Também podem ser atribuídas prioridades aos pedidos e, de acordo com essas prioridades, pode-se gerar um cronograma (Rigby, 2003).

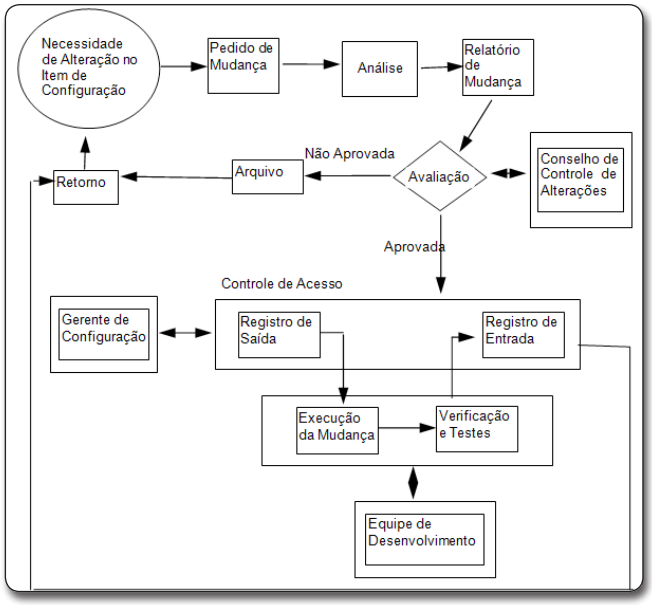


Figura 2. Processo de Controle de Mudanças (Pacheco, 1997)

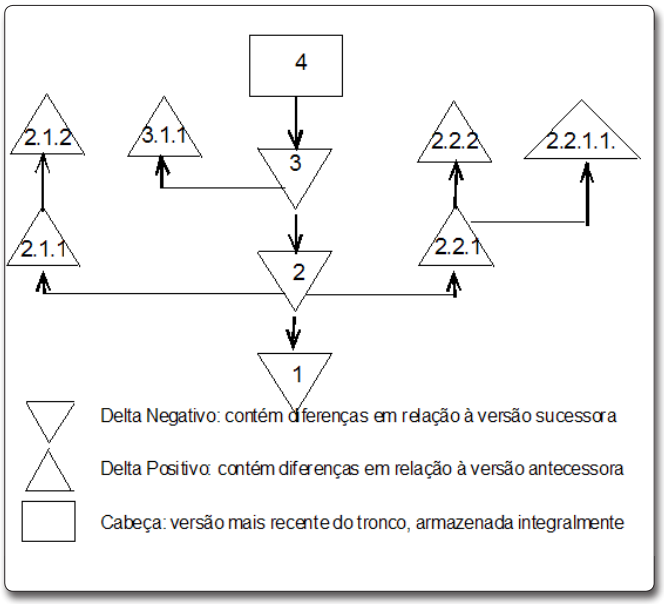


Figura 3. Árvore de revisões em um item de configuração, usando delta

b) Controle de Versões

Um item, ao ser desenvolvido, evolui até que atinja um estado em que atenda aos propósitos para o qual foi criado. Isso implica em diversas alterações, gerando uma versão do item a cada estado (Munch, 1996). Para estabelecer o controle sobre as diversas versões, todas as versões devem ser armazenadas e identificadas. Isso, geralmente, é feito com o auxílio de uma ferramenta.

A versão do item pode ser incluída no esquema de identificação ou ser acessível a partir de uma tabela à parte. É conveniente que o esquema de identificação das versões dos itens seja feito em forma de árvore, pois ao mesmo tempo em que mantém um histórico das versões dos itens, permite identificação única e ramificações a partir de qualquer versão (**Figura 3** (Pacheco, 1997)).

Quando um item existe simultaneamente em duas ou mais formas diferentes que atendam a requisitos similares, temos variantes do item, representados por ramificações na árvore. Um exemplo seria o de duas sub-rotinas para retornar a data do sistema operacional: uma para Unix e outra para DOS (versões 2.1.1 e 2.2.1 na **Figura 3**).

Para minimizar o espaço de armazenamento das versões utiliza-se o conceito de delta, ou seja, são armazenadas uma versão completa e as diferenças entre as versões (Brown, 1991; Humphrey, 1989). Há duas variações desse conceito: delta negativo e delta positivo. Com o delta negativo, armazena-se integralmente a versão mais recente e as diferenças (deltas) existentes até então. Com o delta positivo, armazena-se a versão mais antiga e, para montar as versões mais recentes, processam-se as diferenças (deltas) armazenadas (Clemm, 1999)

Os sistemas atuais de gerência de versões utilizam o conceito de delta negativo no tronco, por ser mais comum a utilização de versões mais recentes do item de configuração (Berczuk, 2003). A **Figura 3** representa um caso em que se utiliza delta negativo. A única versão armazenada integralmente é a 4. As outras versões são construídas, quando solicitadas, a partir da 4 e das diferenças armazenadas. Utilizam-se deltas negativos no tronco da árvore, que representa o caminho principal de evolução do item. As ramificações representam as variantes dos itens e são obtidas pela utilização de delta positivo.

Tarefa 4 - Relato da Situação da Configuração

O objetivo dessa tarefa de gerência de configuração é relatar a todas as pessoas envolvidas no desenvolvimento e na manutenção do software as seguintes informações sobre as alterações na configuração de software:

a) O que aconteceu?
b) Quem o fez?
c) Quando aconteceu?
d) O que mais será afetado?

Para isso, deve ser criado um banco de dados sobre as ocorrências na gerência de configuração. Esse banco de dados deve estar disponível aos desenvolvedores com acesso através de palavras-chave. Além disso, deve ser gerado regularmente um relatório de situação para informar as alterações mais importantes. O acesso rápido às informações sobre a configuração agiliza o processo de desenvolvimento e melhora a comunicação entre as pessoas, o que é uma maneira de eliminar muitos problemas relativos à modificação do mesmo item de informação, com intenções diferentes e conflitantes.

Tarefa 5 - Auditoria da Configuração

A identificação e controle das alterações ajudam a manter ordem, mas para assegurar que a alteração foi implementada apropriadamente, há necessidade de auditorias na configuração do software.

Existem dois tipos de auditoria de configuração de software que são pré-requisitos para o estabelecimento das baselines no ciclo desenvolvimento de software: a Auditoria Funcional e a Auditoria Física.

A auditoria funcional preocupa-se com aspectos internos dos arquivos, compreendendo uma verificação técnica formal na configuração de software, que deve ser realizada ao ser fixada uma baseline. Esta verificação é uma atividade de controle de qualidade que tenta descobrir omissões ou erros na configuração, que degradam os padrões de construção do software (Capretz, 1994; Pressman, 2005).

A auditoria física é um processo administrativo que ocorre no final de cada fase do ciclo de vida do software e consiste em verificar se a configuração que será baselined, ou seja, fará parte de uma baseline, está composta da versão mais recente dos itens de configuração, determinados para a fase do ciclo de vida específica (Bersoff, 1979; Bersoff, 1984) e se os procedimentos e padrões foram devidamente aplicados.

Tarefa 6 - Controle de Subcontratados e Fornecedores

As atividades de controle de subcontratados e fornecedores coordenam a forma como os itens que foram desenvolvidos por solicitação a outras empresas ou foram adquiridos já prontos são testados e incorporados ao repositório do projeto.

Para itens subcontratados o plano deve descrever:

a) Os requisitos de gerência de configuração de software a serem satisfeitos pelo subcontratado;
b) Como será feito o monitoramento sobre o subcontratado;
c) Como o código, documentação e dados externos serão testados, aceitos e adicionados ao projeto;
d) Como serão tratadas as questões de propriedade do código produzido, como direitos autorais e royalties.

Para itens adquiridos prontos o plano deve descrever:

a) Como serão recebidos, testados e colocados sob controle de gerência de configuração;
b) Como as mudanças no software do fornecedor serão tratadas;
c) Se e como o fornecedor participará no processo de gerência de mudança do projeto.

Itens de configuração poderão ser adquiridos de fornecedores, subcontratados, clientes, outros projetos ou outras fontes.

Conclusões

Neste artigo foram apresentados conceitos gerais e as principais tarefas de gerência de configuração de software. Um estudo detalhado das necessidades específicas de cada ambiente de desenvolvimento de software, no que diz respeito à gerência de configuração de software, é necessário para que seja possível a execução das tarefas de forma mais adequada a cada situação. ●

Referências

(Pressman, 2005) PRESSMAN, R. S. Software Engineering: a practitioner´s approach. Mc Graw Hill Higher Educational, 6ª. Edição. 2005.

(Mahler, 1994) MAHLER, A. Variants: Keeping things together and telling them apart. In Configuration Management, Vol. 2 of Trends in Software, Wiley, New York, 1994.

(Bersoff, 1979) BERSOFF, E. H.; Henderson, V. D. e Siegel, S.G. Software Configuration Management: A tutorial. Los Alamos, Califórnia. IEEE Computer. v.12, n.1, 1979.

(IEEE Std 828, 1998) IEEE for Software Configuration Management Plans. 1998.

(Tuscany, 1987) TUSCANY. P. A. Software development environment for large switching projects. In Proceedings of Software Engineering for Telecommunications Switching Systems Conference,1987.

(Bersoff, 1984) Bersoff, E.H. Elements of Software Configuration Management. IEEE Transactions on Software Engineering, v.se-1.0, n.1, 1984.

(Pacheco, 1997) PACHECO, R. F. Uma Forma de Implantação de Gerenciamento de Configuração de Software em Empresas de Pequeno Porte. Dissertação (Mestrado) – Instituto de Ciências Matemáticas de Computação, Universidade de São Paulo, São Carlos, 1997.

(Mackay, 1995) MACKAY, S. A. The state-of-the-art in concurrent, distributed configuration management. In Software Configuration Management: Selected Papers SCM-4 and SCM-5 (Seattle, WA, April), J. Estublier, 1995.

(Leblang, 1997) LEBLANG, D.B.: Managing the Software Development Process with ClearGuide, in Software Configuration Management - ICSE'97 SCM-7 Worhxhop, LNCS 1235, Springer, Berlin, 1997.

(Honda, 1988) HONDA M. Support for parallel development in the Sun network software environment. In Proc. 8nd International Workshop on Computer-Aided Software Engineering, 1988.

(Rigby, 2003) RIGBY, K. Software Configuration Management Template. Rigby Publishing Limited, 2003.

(Munch, 1996) MUNCH, B. HiCOV: Managing the version space. In Software Configuration Management:ICSE'96 SCM-6 Workshop (Berlin,March), Sommerville, 1996.

(Brown, 1991) BROWN, H. Like a Version. Computer Languages, v.8, n.8, 1991.

(Humphrey, 1989) HUMPHREY, W. S. Managing the Software Process. 1. Ed. Massachusetts. Addison-Wesley, 1989.

(Clemm, 1999) CLEMM. G. Versioning Extensions to WebDav. Rational Software, 1999. http://www.ietf.org/internet-drafts/draft-ietlLwebdav-versioning-02.txt

(Berczuk, 2003) BERCZUK, S. P. Software Configuration Management Patterns. Addison-Wesley, 2003.

(Capretz, 1994) Capretz M. A. M.; Munro M. Software Configuration Management Issues in the Maintenance of Existing Systems. Software Maintenance: Research and Practice, v.6, 1994.

Mudanças no PMBoK 4ª Edição

Resumo das Principais Mudanças da 3ª para a 4ª Edição do PMBoK

De que se trata o artigo?
Este artigo apresentará uma visão geral das principais mudanças ocorridas entre a terceira e quarta edição do PMBOK.

Para que serve?
Apresentar o conjunto de atualizações efetuadas no PMBOK.

Em que situação o tema é útil?
Para aqueles que utilizam o PMBOK como referencial para suas atividades profissionais e gostaria de conhecer as novidades de sua nova versão.

Em edições anteriores da revista, apresentamos uma visão geral do PMBoK 3ª edição. Periodicamente o PMI atualiza o conteúdo deste guia. Esta atualização ocorre tipicamente de quatro em quatro anos e no final de 2008 foi lançado o PMBoK 4ª edição.

Este artigo apresentará uma visão geral das principais mudanças ocorridas entre a terceira e quarta edição.

Com relação ao conteúdo, não existe nenhuma grande mudança na nova versão. Na quarta edição houve a adição e remoção de alguns processos, mas áreas de conhecimento permanecem as mesmas e a grande maioria dos processos, entradas e saídas permaneceram inalterados.

A maior parte do esforço foi voltada para tornar o padrão mais consistente internamente e fazer com que os capítulos ficassem mais coerentes parecendo escritos por uma única pessoa, em vez de por um grupo de pessoas.

Além disso, uma das diretrizes de trabalho na elaboração da nova edição foi o alinhamento dos conceitos do PMBoK com a segunda edição do "Standard for Portfolio Management" e do "Standard for Program Management", ambos também do PMI.

O "Standard for Program Management" define o que é gerenciamento de programas além de descrever os processos relacionados e o ciclo de vida do gerenciamento de programas dentro de uma organização. Um programa é um grupo de projetos relacionados gerenciados de modo coordenado para obter benefícios e controle que não estariam disponíveis se fossem gerenciados individualmente.

Paulo Augusto Oyamada Tamaki
paulotamaki@gmail.com.br
Engenheiro de Computação formado com honra ao mérito na Escola Politécnica da USP. Mestre em Engenharia de Software pelo departamento de Sistemas Digitais da Escola Politécnica da USP, cujo tema de mestrado é "Uma extensão do RUP com ênfase no gerenciamento de projetos do PMBoK baseada em process patterns". Escritor de artigos científicos e palestrante de congressos e conferências na área de TI. Possui 5 anos de experiência em gerenciamento de projetos de TI voltados para a área de saúde.

O "Standard for Portfolio Management" descreve um conjunto de processos geralmente aceitos no gerenciamento de portfólios. Portfólio é uma coleção de projetos e/ou programas que possam ser agrupados para facilitar o gerenciamento do trabalho para atingir objetivos estratégicos do negócio.

Os dois padrões citados acima foram elaborados como uma expansão da terceira edição PMBoK, por essa razão, houve a preocupação em alinhar seus conceitos. Os primeiros capítulos do PMBoK falam sobre portfólio e programas, em muitos momentos o conteúdo apresentado é muito parecido com o conteúdo presente nos padrões.

Na quarta edição houve uma mudança geral no quesito da apresentação, diversos diagramas foram atualizados. Os diagramas dos processos foram reformulados e deixam mais claros como funciona o fluxo de entradas e saídas entre os processos e as dependências ou relações entre os processos.

A estrutura de capítulos permaneceu inalterada. Os capítulos estão na mesma ordem e apresentam os mesmos objetivos. Para facilitar a explicação das mudanças, a seguir serão detalhadas as principais mudanças de cada um dos capítulos do PMBoK.

Mudanças no Capítulo 1 – Introdução

O capítulo 1 apresenta uma visão geral do gerenciamento de projetos e como ele é enquadrado em programas, portfólios, organizações e operações. Como mencionado anteriormente, a principal questão abordada foi o alinhamento de conceitos entre os padrões.

Outra mudança, é que na terceira edição havia uma boa discussão sobre a tripla de restrições de um projeto: escopo, prazo e custo. Na quarta edição, o enfoque foi ampliado para outras restrições. Nela existem recomendações de como os gerentes de projeto devem balancear as restrições de qualidade, escopo, prazo, orçamento, recursos e riscos.

Mudanças no Capítulo 2 – Ciclo de Vida e Organização do Projeto

Não houve muitas mudanças no capítulo 2. A mais relevante foi um aumento no conteúdo do capítulo, abordando mais profundamente temas como o ciclo de vida do projeto e as fases do projeto, e dos tipos de partes interessadas.

Sobre o ciclo de vida do projeto e das fases do projeto, existem melhores definições sobre as relações entre as passagens das fases. Na edição anterior, havia uma indicação que as fases do projeto geralmente são seqüenciais. Agora são abordados três tipos de passagens ou relacionamentos entre fases:

• **Seqüencial:** Onde uma fase só pode ser iniciada quando a anterior for completada.

• **Sobreposta:** Onde uma fase pode ser iniciada antes do término da fase anterior. Ou seja, duas fases ficam temporariamente concorrentes.

• **Iterativo:** Onde o planejamento existe apenas para a fase atual. O planejamento da próxima fase é elaborado durante o trabalho da fase em andamento

Com relação aos tipos de partes interessadas, houve um detalhamento maior nas descrições de cada uma delas. Na quarta edição temos os seguintes tipos de partes interessadas:

• **Gerentes de Projeto:** Responsável pelo gerenciamento do projeto. Clientes/Usuários: A pessoa ou organização que utilizará o produto do projeto.

• **Gerentes de Operação:** Indivíduo com papel gerencial em alguma área operacional chave do negócio. Diferentemente dos gerentes funcionais, estes gerentes lidam diretamente com a produção e manutenção dos produtos principais da organização.

• **Membros da Equipe do Projeto:** O grupo que está executando o trabalho do projeto.

• **Patrocinador:** Pessoa ou grupo que fornece recursos financeiros para o projeto.

Escritório de Projetos: Organiza e gerencia o controle sobre todos os projetos dentro da organização.

• **Fornecedores/Parceiros:** Empresas terceiras que, através de um acordo contratual, fornecem componentes ou serviços necessários para o projeto.

• **Gerentes Funcionais:** Indivíduos com papel gerencial em uma área administrativa ou funcional. Como recursos humanos, financeiro, comercial, etc.

• **Gerentes de Programas:** Responsável pelo gerenciamento de programa.

• **Gerentes de Portfólio:** Responsável pelo gerenciamento do portfólio.

Mudanças no Capítulo 3 – Processos de Gerenciamento de Projetos de um Projeto

Este capítulo introduz um resumo geral dos processos, indicando uma breve descrição do processo, suas entradas e saídas. Neste capítulo já pode ser notada a redução no número total de processos além de modificações nas tabelas de entradas e saídas de diversos processos. Porém, o detalhamento efetivo das mudanças é mais destacado em cada um dos capítulos seguintes.

Na edição anterior, existiam 44 processos, enquanto que na edição atual existem 42 processos. Entretanto, essa redução não significa que alguns processos foram considerados obsoletos e descartados. De forma geral, o que houve foi a consolidação de dois ou mais processos em um único processo. Mesmo com essa redução do total de processos, ainda foram adicionados dois novos processos.

Uma mudança geral ocorreu nas entradas e saídas. Nenhum processo do grupo de processos de planejamento utiliza mais o Plano de Gerenciamento do Projeto com entrada. O entendimento dos autores é que o planejamento ocorre durante todo o projeto e que o grupo de processos de planejamento não é uma fase. Portanto, eles preferiram utilizar as saídas dos outros processos de planejamento como entrada para o processo Desenvolver Plano de Gerenciamento do Projeto e não o contrário. Na edição anterior, o Plano de Gerenciamento do Projeto era entrada de diversos processos de planejamento, como Planejar Escopo.

Apesar desta mudança, o Plano de Gerenciamento do Projeto continua uma entrada chave nos processos de execução e monitoramento e controle.

Outra mudança foi a adição do conceito Documentos do Projeto. Os Documentos do Projeto são utilizados para simplificar diversas tabelas de entradas e saídas de processos. Em cada um dos processos estão representadas

as principais entradas e saídas, nos casos em que existem diversos documentos que opcionalmente possam ser atualizados ou utilizados, o processo indica o uso de Documentos do Projeto. Na descrição de cada um dos processos, existe uma seção para os Documentos do Projeto especificando em maiores detalhes quais documentos podem fazer parte deste pacote.

Mudanças no Capítulo 4 – Gerenciamento de Integração do Projeto

O total de processos caiu de sete para seis. O processo Desenvolver a Declaração do Escopo Preliminar do Projeto foi eliminado. A razão da remoção é que esse processo foi considerado como parte do processo Definir Escopo através do conceito de elaboração progressiva dos documentos.

Além disso, houve uma fusão dos documentos: Pedido de Mudança, Ação Corretiva, Ação Preventiva, e Reparo de Defeito. Todos ficaram agrupados no conceito Pedidos de Mudança. Onde cada um deles é um tipo de Pedido de Mudança.

Mudanças no Capítulo 5 – Gerenciamento do Escopo do Projeto

A principal mudança foi a adição do novo processo: Coletar Requisitos.

Coletar Requisitos

O objetivo deste processo é definir e documentar características e funcionalidades do produto necessárias para atender as necessidades e expectativas das partes interessadas.

As entradas do processo são:

• **Termo de Abertura:** Este documento fornece uma visão de alto nível dos requisitos e da descrição do produto.

• **Registro das Partes Interessadas:** Utilizado para identificar partes interessadas que podem oferecer informações mais detalhadas sobre os requisitos do produto ou projeto.

As saídas são do processo são:

• **Documento de Requisitos das Partes Interessadas:** Este documento descreve como requisitos individuais atendem as necessidades do negócio.

• **Plano de Gerenciamento de Requisitos:** Este plano define como os requisitos serão analisados, documentados e gerenciados ao longo de todo o projeto.

• **Matriz de Rastreabilidade de Requisitos:** Tabela que relaciona os requisitos com sua origem e rastreia ele ao longo do ciclo de vida do projeto. Seu propósito é auxiliar a garantir que cada requisito agrega valor ao negócio, isto é feito associando cada requisito ao objetivo de negócio ou de projeto que o originou. Além disso, ele oferece meios de rastrear os requisitos, ajudando a garantir que cada requisito aprovado foi entregue.

Mudanças no Capítulo 6 – Gerenciamento de Tempo do Projeto

Nesta área de conhecimento não houve mudança nos processos.

Mudanças no Capítulo 7 – Gerenciamento de Custos do Projeto

Os processos desta área de conhecimento permaneceram inalterados. Entretanto, as entradas e saídas dos processos Controlar Custos, Controlar Escopo e Controlar Cronograma estão mais consistentes.

Foi adicionada uma nova ferramenta ou técnica para o processo Controlar Custos: o Índice de Desempenho de Custo Futuro (TCPI - To-Complete Performance Index). Este apóia a projeção de viabilidade de se cumprir o orçamento.

Mudanças no Capítulo 8 – Gerenciamento da Qualidade do Projeto

Não houve mudanças nos processos de gerenciamento da qualidade. Mas foi adicionada uma discussão mais aprofundada sobre os custos da qualidade, e sobre os gráficos de controle da qualidade, em particular dos conceitos de limite superior e inferior da especificação.

O termo baseline da qualidade foi removido.

Mudanças no Capítulo 9 – Gerenciamento de Recursos Humanos do Projeto

Os processos do gerenciamento de recursos permaneceram iguais. A única modificação é que o processo Gerenciar Equipe do Projeto foi movido do grupo de processos de monitoramento e controle para o grupo de processos de execução.

Uma mudança menor foi uma discussão mais abrangente sobre habilidades interpessoais nos processos Desenvolver Equipe do Projeto e Gerenciar Equipe do Projeto.

Por fim, foi adicionada uma discussão sobre as etapas de montagem de equipe, gerenciamento de conflitos, liderança, influência e tomada de decisão.

Mudanças no Capítulo 10 – Gerenciamento de Comunicações do Projeto

Nesta área de conhecimento, foi adicionado um novo processo de iniciação: Identificar as Partes Interessadas.

Identificar as Partes Interessadas

Este processo tem como objetivo identificar todas as pessoas ou organizações afetadas pelo projeto, além de documentar informações relevantes a respeito de seus interesses, envolvimento e impacto sobre o sucesso deste projeto. A identificação das partes interessadas é essencial para aumentar a probabilidade do sucesso do projeto. De modo geral, é importante analisar o nível de interesse, expectativas, importância e influência de cada parte interessada, pois com essas informações pode ser elaborada uma estratégia sobre quando e como uma determinada parte interessada pode contribuir mais positivamente no projeto.

As entradas deste processo são:

• **Termo de Abertura:** O Termo de abertura fornece informações sobre grupos e contatos internos e externos ao projeto que podem afetar ou estar envolvidos com o projeto.

• **Documentos de Aquisição:** Se o projeto tiver um contrato externo, as partes envolvidas no contrato são partes interessadas chave.

• **Ativos de Processos Organizacionais:** Eventuais lições aprendidas em projetos anteriores, ou tipos de partes interessadas conhecidas, registros de partes interessadas de projetos anteriores.
• **Fatores Ambientais da Empresa:** Cultura e estrutura organizacional, ou padrões governamentais ou de indústria.

As saídas deste processo são:
• **Registro das Partes Interessadas:** documenta as seguintes informações sobre cada uma das partes interessadas: Nome, cargo, papel no projeto, informações de contato, principais requisitos, principais expectativas, influência potencial no projeto, e eventuais classificações (ex.: interno/externo, a favor/neutro/contra).
• **Estratégia de Gerenciamento das Partes Interessadas:** Define uma abordagem ou estratégia para aumentar o apoio e minimizar os impactos negativos das partes interessadas ao longo do ciclo de vida do projeto.

O processo conhecido na terceira edição como Gerenciar as Partes Interessadas foi renomeado para Gerenciar as Expectativas das Partes Interessadas e movido do grupo de processos de monitoramento e controle para o de execução.

Mudanças no Capítulo 11 – Gerenciamento de Riscos do Projeto

Nesta área de conhecimento, os processos não foram alterados e não ocorreu nenhuma mudança significativa.

Mudanças no Capítulo 12 – Gerenciamento de Aquisições do Projeto

No gerenciamento de aquisições do projeto, seis processos foram consolidados em quatro. Os processos agora são:
• **Planejar Aquisições:** É o processo de documentar as decisões de compra no projeto, a abordagem, e a identificação de potenciais fornecedores. Ele registra as necessidades do projeto que serão mais bem resolvidas adquirindo produtos ou serviços externos à equipe do projeto.
• **Conduzir Aquisições:** Processo de obter propostas de fornecedores, selecionar fornecedores e estabelecer o contrato.
• **Administrar Aquisições:** Processo responsável por gerenciar o contrato, monitorar o andamento do contrato e realizar mudanças e correções conforme necessárias.
• **Encerrar Aquisições:** É o processo de finalizar cada um dos contratos do projeto. Ele envolve a verificação de que todos os serviços e entregas externos foram aceitos.

Apêndice

A quarta edição apresenta um novo apêndice sobre habilidades interpessoais. Nesse apêndice estão informações consideradas pelos autores como importantes para o gerenciamento de um projeto, porém que não eram consistentes com as intenções de um padrão. As habilidades apresentadas são:
• **Liderança:** desenvolver uma visão e uma estratégia e motivar as pessoas para que alcancem essa visão e essa estratégia;
• **Montagem de Equipe:** capacidade de montar a equipe considerando os perfis adequados ao trabalho do projeto;
• **Motivação:** estimular as pessoas para que alcancem altos níveis de desempenho e superem as barreiras que impedem as mudanças;
• **Comunicação:** capacidade de trocar informações;
Influência: a capacidade de "fazer com que as coisas aconteçam";
• **Tomada de Decisão:** capacidade em tomar decisão a partir de um cenário atual;
• **Consciência política e cultural:** capacidade de respeitar os limites impostas por questões políticas e culturais;
• **Negociação:** Conversar com outras pessoas para chegar a um entendimento ou um acordo.

Conclusão

É um fato que os conhecimentos e necessidades do gerenciamento de projetos evoluem com o tempo. Nesse contexto, as atualizações do PMBoK são necessárias e têm se mostrado alinhadas com as novas necessidades e experiências.

O novo processo Coletar Requisitos apresenta um uso direto para atividades de análise de requisitos de desenvolvimento de software.

Ao observar o RUP (Rational Unified Process), este processo é alinhado com as atividades existentes na disciplina de requisitos. As saídas deste processo são análogas a diversos artefatos presentes no RUP também. Por exemplo, o plano de gerenciamento de requisitos pode incluir o processo de priorização de requisitos, que é uma das principais atividades do RUP.

Sob a perspectiva do CMMI, diversas práticas recomendadas na terceira edição PMBoK auxiliavam na implementação de alguns de seus processos. A adição do processo Coletar Requisitos traz o PMBoK ainda mais próximo do CMMI, pois auxilia na implementação do processo Desenvolvimento de Requisitos deste modelo de qualidade.

Finalmente, a expansão dos tipos de relacionamento entre fases do projeto é alinhada com necessidades ou opções de desenvolvimento comuns no desenvolvimento de software. O modelo iterativo de relacionamento entre fases é útil para projetos em ambientes voláteis, com incertezas, ou muito indefinidos. Sua dificuldade é o planejamento de longo prazo ou o controle de escopo. Em muitos casos, estes cenários são encontrados quando empregamos em métodos ágeis de desenvolvimento, por exemplo.

Se o PMBoK já era um guia com aplicação direta ao desenvolvimento de software, as suas atualizações tornam claro que a quarta edição apresenta uma série de melhorias com benefícios diretos para o gerenciamento deste tipo de projetos. Porém, é importante ter em mente que sua aplicação não precisa ser uniforme em todos os projetos. Dependendo da organização ou do projeto, seus processos e documentos devem ser adaptados para serem apropriados para a cultura ou necessidades envolvidas. ●

Gerenciamento de Mudanças em Projetos de TI

Gestão de mudanças utilizando o PMBoK e a ferramenta Rational ClearQuest

Felipe La Rocca Teixeira

felipe.lt@pjf.mg.gov.br

Pós-Graduado em Gestão de Projetos - PMI, Bacharel em Sistemas de Informação, Analista de Sistemas da Prefeitura de Juiz de Fora e possui certificações MCITP Database Administrator, MCITP Business Intelligence Developer e OCA 10g.

Marco Antônio Pereira Araújo

maraujo@cesjf.br

Doutor e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ, Especialista em Métodos Estatísticos Computacionais e Bacharel em Informática pela UFJF, Professor e Coordenador do Curso de Bacharelado em Sistemas de Informação do Centro de Ensino Superior de Juiz de Fora (CES/JF), Analista de Sistemas da Prefeitura de Juiz de Fora.

De que se trata o artigo?

Este artigo procura apresentar processos que garantam um controle completo das modificações em todo o clico de vida de um projeto de TI, realizando-a com um risco mínimo, permitindo assim, um aumento da qualidade e a redução dos riscos e custos relativos a estes projetos.

Para que serve?

Para que serve: O artigo demonstra como modificações podem ser gerenciadas em um projeto de TI, além dos benefícios alcançados como a redução do impacto de mudanças sobre a qualidade do projeto, e isto pode ser alcançado através do uso de ferramentas para a gestão de mudanças como o Rational ClearQuest.

Em que situação o tema é útil?

Gerenciamento de mudanças em projetos de TI, como o desenvolvimento de softwares.

Devido ao rápido avanço da tecnologia, o desenvolvimento e a manutenção de projetos de software tornou-se algo difícil e complexo. Toda essa nova tecnologia também gera um aumento de novas solicitações e expectativas ao longo de todas as fases de um projeto de TI. Por isso, para que se possa elaborar um projeto e realizar serviços de qualidade, é necessário não somente uma equipe técnica competente, mas a utilização de métodos e processos que contemplem as melhores práticas para a gestão de TI.

Ao longo do ciclo de vida de um projeto, modificações são inevitáveis, podendo ocorrer desde mudanças no escopo do projeto, alterações dos requisitos levantados, correções de defeitos ou melhorias. Essas modificações podem exigir ajustes no plano de gerenciamento do projeto, na declaração do escopo ou em outras entregas do projeto. Seguindo as melhores práticas do PMBoK (Project Management Body of Knowledge), o processo de Controle Integrado de Mudanças, pertencente à área de conhecimentos de Gerenciamento da Integração, é o responsável pela aprovação e monitoração das mudanças solicitadas. Esse processo será mais bem detalhado no decorrer deste artigo.

Gerenciamento de Projetos com o PMBoK

Gerenciar um projeto significa, resumidamente, planejar a sua execução antes de iniciá-lo e, posteriormente acompanhar a sua execução e controle. Algumas práticas gerenciais, quando bem aplicadas, contribuem para a melhoria da qualidade da gerência de um projeto. Os processos de desenvolvimento atuais sugerem a adoção de abordagens iterativas e incrementais.

Com a utilização do gerenciamento de projetos, alguns benefícios podem ser alcançados, dentre eles, evitar que surpresas durante a execução dos trabalhos aconteçam, antecipar situações desfavoráveis, desde que ações preventivas e corretivas sejam tomadas antes que estas situações se tornem um problema para o projeto, disponibilizar os orçamentos antes do início dos trabalhos, gerar documentação no intuito de facilitar estimativas para futuros projetos, bem como outros benefícios.

O PMBoK, desenvolvido pelo PMI (Project Management Institute), é um guia prático para gestão de projetos de qualquer natureza, inclusive para projetos da área de TI, como por exemplo, o desenvolvimento de software. A aplicação de seus processos, conhecimentos e técnicas buscam atender ou superar as expectativas de todos os envolvidos em um projeto.

Criado em 1969, com sede na Filadélfia, Pensilvânia, o PMI define um projeto como sendo "um esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo". Temporário porque todos os projetos possuem um início e um fim definidos, não importando o seu tamanho, podendo ser de curta ou de longa duração. Exclusivo por tratar da singularidade destes elementos, ou seja, onde cada produto ou serviço possuem características especificas que os diferenciam.

O PMBoK analisa o gerenciamento de um projeto da seguinte maneira:

- Divisão do projeto em fases, Ciclo de Vida;
- Em cada fase ocorrem processos;
- Em cada processo são executadas ações gerenciais que contemplam nove áreas de conhecimento.

Ciclo de Vida de um Projeto:

Um ciclo de vida é caracterizado por várias fases distintas, certamente dependendo do tipo de projeto (construção e desenvolvimento de software, entre outros), onde suas fases possuem particularidades próprias. Em cada fase de um projeto são executados diversos processos com o objetivo de produzir o resultado esperado daquela fase.

O ciclo de vida do projeto define o início e o fim do projeto e também qual trabalho técnico deve ser realizado em cada fase e quem deve estar envolvido nelas. A transição de uma fase para outra ocorre normalmente através de alguma forma de transferência técnica ou entrega. As entregas de uma fase são revisadas para garantir que estejam completas e exatas, e aprovadas antes que o trabalho seja iniciado na próxima fase. Mas pode acontecer que uma fase seja iniciada antes da aprovação das entregas da fase anterior, e isto pode ocorrer quando os riscos envolvidos são considerados aceitáveis. De acordo com o PMBoK, os ciclos de vida de um projeto geralmente definem:

- Qual trabalho técnico deve ser realizado em cada fase, por exemplo, em qual fase deve ser realizado o trabalho de um programador em um projeto de desenvolvimento de software;
- Em que momento do projeto as entregas devem ser geradas em cada fase e como cada entrega é revisada, verificada e validada;
- Quem está envolvido em cada fase. Por exemplo, no desenvolvimento de um software, exige-se que os programadores estejam envolvidos com a implementação e o teste do software;
- Como controlar e aprovar cada uma das fases;
- As descrições do ciclo de vida do projeto.

Processos da Gerência de Projeto:

Em cada fase de um projeto são executados diversos processos com o objetivo de produzir o resultado esperado daquela etapa. Conforme definido pelo PMBoK, esses processos se enquadram nos seguintes grupos (ver **Figura 1**):

- **Processo de Inicialização:** é a fase inicial do projeto, onde o objetivo do projeto é definido, bem como as melhores estratégias são identificadas e selecionadas;
- **Processo de Planejamento:** é a fase responsável em detalhar e descrever tudo aquilo que será realizado pelo projeto, incluindo a definição do escopo, estrutura analítica do projeto, estimativas, análise de custos e outros;
- **Processo de Execução:** é a fase onde se materializa tudo o que foi planejado na fase anterior. Qualquer erro cometido anteriormente fica evidente durante essa fase;
- **Processo de Monitoramento & Controle:** é a fase de acompanhamento e controle de tudo o que está sendo realizado pelo projeto. Assim, pode-se garantir que seus objetivos sejam alcançados através da monitoração e da mensuração de seu progresso, ao se tomar ações corretivas e proativas sempre que houver necessidade;
- **Processo de Encerramento:** é a fase onde a execução dos trabalhos é avaliada junto ao cliente ou patrocinador, e ocorre seu encerramento de forma ordenada.

Figura 1. Grupos de processos de gerenciamento de projetos (PMBoK)

Áreas de Conhecimento:

As áreas do gerenciamento de projetos descrevem o gerenciamento em termos de seus processos componentes. Esses processos estão organizados em nove grupos integrados (ver **Figura 2**).

O **Gerenciamento da Integração** define os processos necessários para assegurar a integração de todas as demais áreas de conhecimento. Seu objetivo é estruturar todo o projeto de modo a garantir que os diversos elementos do projeto sejam adequadamente coordenados. Seus processos constantes são: desenvolver o termo de abertura do projeto, desenvolver a declaração do escopo preliminar do projeto, desenvolver o plano de gerenciamento do projeto, orientar e gerenciar a execução do projeto, monitorar e controlar o trabalho do projeto, controle integrado de mudanças (esse será mais bem detalhado no decorrer deste artigo) e encerrar o projeto.

O **Gerenciamento do Escopo** é responsável pelos processos necessários para garantir que o projeto contemple somente aquilo que é necessário para ser concluído com sucesso, sem abandonar nenhuma função estabelecida no objetivo do projeto. Seus processos são: o planejamento do escopo, definição do escopo, criar a estrutura analítica do projeto, verificação do escopo e controle do escopo.

O **Gerenciamento do Tempo** consiste em processos necessários para garantir que o projeto termine dentro do seu prazo estipulado. Trabalha na definição da atividade, sequenciamento de atividade, estimativa de recursos da atividade, estimativa de duração da atividade, desenvolvimento do cronograma e controle do cronograma.

O **Gerenciamento do Custo** define processos necessários para assegurar que o capital disponível seja suficiente para concluir o projeto dentro do orçamento previsto. Seus principais processos são: estimativa de custos, orçamento e o controle de custos.

O **Gerenciamento dos Recursos Humanos** são os processos necessários para garantir o melhor uso das pessoas envolvidas no projeto. Os processos constantes são: planejamento de recursos humanos, contratar ou mobilizar a equipe do projeto, desenvolver a equipe de projeto e gerenciar a equipe do projeto.

O **Gerenciamento dos Riscos** define os processos relacionados à identificação, análise e resposta aos riscos do projeto. É responsável pelos processos de planejamento do gerenciamento de riscos, identificação de riscos, análise qualitativa de riscos, análise quantitativa de riscos, planejamento de respostas a riscos e monitoramento e controle de riscos.

O **Gerenciamento das Comunicações** é responsável em assegurar que todas as informações do projeto cheguem às pessoas certas de forma adequada e no tempo certo. Seus processos são: planejamento das comunicações, distribuição de informações, relatório de desempenho e gerenciamento das partes interessadas.

O **Gerenciamento das Aquisições** são os processos necessários para garantir a aquisição de bens e serviços fora da organização. Seus processos constantes são: planejar compras e aquisições, planejar contratações, solicitar respostas de fornecedores, selecionar fornecedores, administrar contratos e o encerramento do contrato.

Figura 2. Áreas de Conhecimentos (PMBoK)

Gerenciamento de Mudanças

Ao longo de todo o ciclo de vida de um projeto, diversos desafios são encontrados. Métodos, processos, técnicas e ferramentas devem ser integrados no intuito de apoiar o desenvolvimento de um projeto de TI, como por exemplo, o desenvolvimento de software. O processo de mudanças é algo inevitável em um projeto, e isto deve ser gerenciado de forma efetiva através de planejamento, ou seja, é preciso detalhar como o processo de mudanças irá acontecer. Para que isso ocorra da maneira correta, algumas questões devem ser respondidas, tais como:

- Como solicitar mudanças no projeto?
- Para onde encaminhar as solicitações?
- Quem as analisa?
- Com que freqüência?
- De que forma?

Sem um gerenciamento definido de mudanças é impossível garantir que as alterações propostas estejam de acordo com os objetivos do projeto. Segundo o PMBoK, o processo Controle Integrado de Mudanças (pertencente à área de conhecimentos Gerenciamento da Integração do Projeto) tem como seu objetivo principal a aprovação de mudanças solicitadas, de não-conformidades recomendadas e de reparos de defeitos recomendados, observados nas fases de monitoramento e controle do projeto.

Entretanto, o gerenciamento de mudanças em um ambiente de serviços TI, é controlada pelo processo de Gerenciamento de Alterações pertencente à área de Suporte de Serviços da ITIL. Este processo trata da requisição, avaliação, autorização e implementação de mudanças em serviços de TI, visando gerar o menor impacto possível para a organização em relação a mudanças e minimizar possíveis interrupções destes serviços.

Controle Integrado de Mudanças - PMBoK

O processo Controle Integrado de Mudanças é realizado desde o início do projeto até o seu término. Mudanças no decorrer dos projetos são inevitáveis, e raramente a sua execução segue com exatidão o plano de gerenciamento do projeto.

A importância deste processo se faz necessária, uma vez que sempre há diferença entre o planejado e o realizado, além do controle de mudanças através de um gerenciamento contínuo, cabendo a esse processo rejeitar ou aceitar tais mudanças. Para que isso ocorra, esse processo inclui as seguintes atividades de gerenciamento de mudanças em níveis diferentes de detalhes:

• Identificar que uma mudança precisa ocorrer ou ocorreu;
• Controlar fatores que poderiam dificultar o controle integrado de mudanças de forma que somente mudanças aprovadas sejam implementadas;
• Revisar e aprovar as mudanças solicitadas;
• Gerenciar as mudanças aprovadas quando e como ocorrem, regulando o fluxo de mudanças solicitadas;
• Manter a integridade das linhas de base liberando somente as mudanças aprovadas para serem incorporadas aos produtos ou serviços do projeto e manter sua configuração e sua documentação de planejamento relacionada;
• Revisar e aprovar todas as ações preventivas e corretivas recomendadas;
• Controlar e atualizar o escopo, custo, orçamento, cronograma e requisitos de qualidade, de acordo com as mudanças aprovadas, através de um controle das mudanças em todo o projeto;
• Documentar o impacto total nas mudanças solicitadas;
• Validar o reparo de defeito;
• Controlar a qualidade do projeto de acordo com as normas, com base nos relatórios de qualidade.

O gerenciamento de configuração com controle de mudanças disponibiliza um processo eficaz, eficiente e padronizado para gerenciar de maneira centralizada mudanças em um projeto. Esse gerenciamento inclui a identificação, documentação e controle das mudanças realizadas. O nível aplicado de controle de mudanças depende da área de aplicação, da complexidade do projeto, dos requisitos levantados e do contexto e ambiente em que o projeto será desenvolvido.

Segundo o PMBoK, a aplicação do gerenciamento de configuração com controle de mudanças em todo o projeto deve realizar três objetivos principais, que são:

• Estabelecer um processo evolutivo para identificar e solicitar mudanças de forma consistente e avaliar a importância e a eficácia dessas mudanças;
• Oferecer oportunidades para validar e melhorar continuamente o projeto, considerando o impacto de cada mudança;
• Fornecer um mecanismo para que a equipe de projeto possa comunicar todas as mudanças de forma consistente e exata para as partes interessadas.

Atividades relacionadas ao Controle Integrado de Mudanças

Todas as mudanças solicitadas e documentadas devem ser aceitas ou rejeitadas por uma autoridade de dentro da equipe do projeto ou pelo iniciador, patrocinador ou cliente. O processo de controle integrado de mudanças muitas vezes inclui um comitê de controle de mudanças, que é o responsável em aprovar ou rejeitar as mudanças solicitadas. As responsabilidades e funções desses comitês são definidas no processo de controle de mudanças e com a participação do patrocinador, cliente e outras partes interessadas.

As atividades realizadas pelo controle integrado de mudanças podem ser divididas em Entradas, Ferramentas e Saídas (ver **Figura 3**).

Uma das principais entradas desse processo refere-se às mudanças solicitadas, e essas, por sua vez, podem ser aprovadas ou rejeitadas. As solicitações de mudança aprovadas referem-se àquelas mudanças que foram solicitadas, recomendadas e aprovadas, como por exemplo, ampliar ou limitar o escopo do projeto. As solicitações de mudanças rejeitadas se referem às mudanças solicitadas que podem gerar alterações críticas para o projeto, e aquelas que não são factíveis de implementação por vários outros fatores, como por exemplo, a não-conformidade com o escopo do projeto.

Figura 3. Entradas, ferramentas e saídas do Controle de Mudanças - PMBoK

Em relação às ferramentas e técnicas utilizadas neste processo, pode-se destacar o sistema de informações de gerenciamento de projetos. O uso de ferramentas, no intuito de automatizar esse processo, é utilizado pela equipe de gerenciamento de projetos para auxiliar na implementação de um processo de controle integrado de mudanças do projeto, facilitar o feedback do projeto e controlar as mudanças em todo o projeto.

Ferramentas para a Gestão de Mudanças

Não é objetivo do gerenciamento de mudanças evitar modificações, mas sim permitir que ocorram de maneira controlada. Esse controle pode acontecer através do uso de várias ferramentas existentes no mercado e, dessa forma, permitir o rastreamento dessas mudanças e identificar o impacto no projeto como um todo.

O uso de ferramentas conhecidas como Bug Tracking possibilita a melhoria dos processos utilizados no desenvolvimento e manutenção de produtos de software e projetos da área de TI. Essas ferramentas precisam oferecer processos para identificar, analisar, rastrear e controlar mudanças, além de permitir que usuários façam o acompanhamento completo dos pedidos de alterações sobre erros encontrados em sistemas. No mercado, existem várias ferramentas open source de Bug Tracking disponíveis, tais como o Bugzilla e o Trac, e em relação às ferramentas comerciais pode-se destacar o Rational ClearQuest.

O Rational ClearQuest é uma ferramenta de Bug Tracking que busca um monitoramento flexível de mudanças e defeitos, automação de processo, elaboração de relatórios e rastreabilidade, visando uma melhor visibilidade e controle do ciclo de desenvolvimento de um software. Como essa ferramenta pode se ligar ao código fonte de um projeto de TI, é possível de maneira simples cadastrar e acompanhar defeitos e mudanças, bem como controlar qualquer outro tipo de serviço realizado pela a equipe de desenvolvimento. No intuito de facilitar o aprendizado dessa ferramenta, a Rational ClearQuest disponibiliza um projeto completo com registros de defeitos e usuários já pré-cadastrados, e para que se possa utilizar este projeto de exemplo, deve-se marcar a opção Create sample database, selecionar o valor "Enterprise" para o campo Schema e informar o valor "SAMPL" para o campo Database Name. Com isso, várias informações referentes a pedidos de alterações, defeitos e usuários serão gravadas no banco de dados de exemplo.

No intuito de demonstrar na prática a utilização da ferramenta Rational ClearQuest em um projeto de TI, este artigo irá apresentar todos os passos necessários para um efetivo controle de mudanças, demonstrando desde a criação de um registro de defeito, suas alterações, até a sua solução e criação de consultas (ler **Nota 1**).

> Nota 1
>
> Para a construção deste exemplo, será necessário a instalação do Microsoft Access e a ferramenta Rational ClearQuest.

Ao utilizar o Rational ClearQuest, primeiramente deve-se criar um repositório de dados comum, visando oferecer às equipes de desenvolvimento acesso rápido e seguro às informações referentes ao projeto. Para este exemplo, será necessário definir duas bases de dados para a gravação das informações, uma para a base principal e outra para a base de exemplo, configurada pela própria ferramenta. Para isso, deve-se ir ao menu Iniciar > IBM Rational ClearQuest > Ferramenta de Manutenção do ClearQuest e, na tela IBM Rational ClearQuest Maintenance Tool, clicar no botão Create para criar um novo repositório de dados, nomeá-lo como "Stage" e fazer as configurações apresentadas na **Figura 4**. No campo Feature Level deve-se informar qual versão do Rational ClearQuest está sendo usado, nesse caso, a versão 7.1. No campo Vendor deve-se definir qual o tipo de banco de dados será utilizado para a gravação dos dados, desta forma, deve-se definir o Microsoft Access. Por último, no campo Physical Database Name deve-se informar o nome para o banco de dados como "Master.mdb".

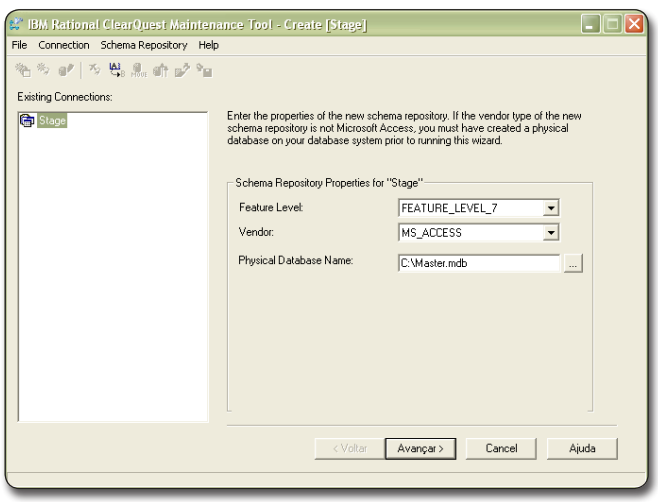


Figura 4. Visualização da Janela do IBM Rational ClearQuest Maintenance Tool

Continuando com a configuração do repositório de dados, após clicar em Avançar, na próxima janela deve-se definir qual Data Code Page será utilizado, ou seja, o conjunto de caracteres que serão suportados pela base de dados. Para isso, deve-se selecionar o valor 1252 (MS Windows Latin1) e clicar em Avançar.

Por último, devem-se definir as propriedades do banco de dados de exemplo. No campo Vendor, para o tipo de banco de dados, informar Microsoft Access e, em Physical Database Name, informar o nome do banco como "Sample.mdb". Para finalizar, deve-se clicar em Concluir.

Agora, com o repositório de dados criado e devidamente configurado, para conectar-se à base de dados de exemplo utilizando a ferramenta Rational ClearQuest, deve-se clicar em menu Iniciar > IBM Rational ClearQuest > ClearQuest Client e em Arquivo > Banco de Dados > Conectar > Nova Conexão. Na janela Repositório do Esquema, deve-se selecionar o repositório

Stage criado anteriormente e clicar em Avançar. Na próxima janela, deve-se informar o Id de usuário, como "admin", este usuário já é criado por default pela ferramenta e possui todos os privilégios necessários para administrar um projeto no Rational ClearQuest. Ao continuar com a configuração, agora na janela Conectar, deve-se deixar a senha do usuário admin em branco, selecionar o valor "SAMPL" no campo Banco de Dados e clicar em Ok para finalizar a conexão (ver **Figura 5**). É importante destacar que, por default, a senha do usuário admin vem em branco, mas se for necessário acrescentar uma senha para este usuário, deve-se ir ao menu Iniciar > IBM Rational ClearQuest > Administração de Usuário ClearQuest, selecionar o usuário desejado e definir uma senha para ele.

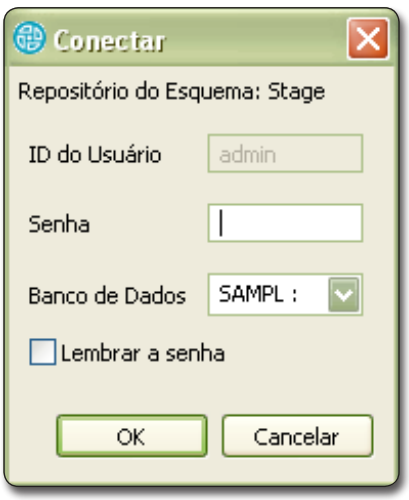


Figura 5. Criação de uma conexão com o banco de dados no Rational ClearQuest

As equipes de desenvolvimento utilizam controles de mudanças para registrar defeitos, pedidos de aprimoramento para recursos existentes e pedidos de novos recursos, sendo que o Rational ClearQuest armazena estes controles como registros em um banco de dados. Desta forma, para que se possa registrar um defeito, deve-se selecionar Arquivo > Novo > Defect. Nesta janela, pode-se destacar que é atribuído um número ID de registro para esse controle de mudança e seu estado por default aparece como Submetido, além dos nomes de campos obrigatórios aparecem em vermelho.

Assim, para poder cadastrar um registro de defeito, deve-se digitar as informações levantadas, conforme **Figura 6**. Na aba Main, em Headline, deve-se definir o título do defeito como "Valores totais incorretos no relatório de Vendas". Nos campos Priority e Severity serão definidas as prioridades e severidades desse defeito, onde os valores estão ordenados do estado mais crítico para o menos crítico, neste caso, deve-se selecionar "2-Give High Attention" e "2-Major", pois o defeito não irá causar uma interrupção do sistema, mas pode ocasionar problemas por disponibilizar uma informação errada.

Também é possível acrescentar informações adicionais ao registro do defeito como no campo Keywords, que fornece uma maneira útil de identificar os tipos de controles de mudanças através da utilização de palavras-chave nas definições de consultas de modo que estas retornem registros que tenham as palavras-chave correspondentes. Já no campo Symptoms podem-se definir sintomas que ocorrem rotineiramente, facilitando o diagnóstico do problema. Para finalizar, é possível acrescentar uma descrição para o problema, para isso, digitar no campo Description o texto "Os valores totais do relatório de Vendas estão sendo exibidos incorretamente". Ao clicar em OK esse registro de defeito será gravado no banco de dados do projeto.

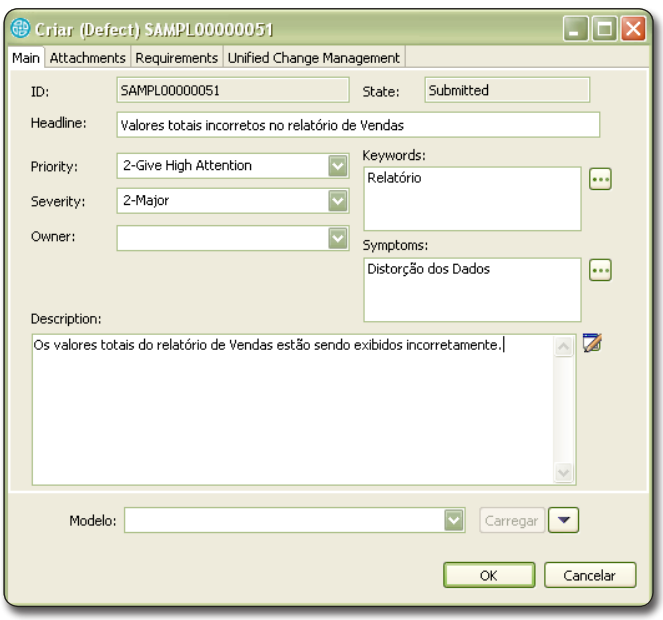


Figura 6. Visualização da Janela para cadastar um defeito

Se for necessário realizar alguma alteração no registro do defeito cadastrado pela equipe de suporte, como por exemplo, acrescentar novas informações referentes ao problema identificado basta selecionar Editar > Localizar Registro > admin,Stage@SAMPL. Na janela Localizar Registro, deve-se selecionar o tipo de registro e informar o ID do defeito que se deseja abrir. Para este exemplo, deve-se selecionar "Defect" e o valor do ID "SAMPL00000051". Para facilitar, não é necessário digitar o prefixo e os zeros à esquerda do ID para localizar o registro (ver **Figura 7**).

Após localizar o defeito desejado, a janela Visualizar Defect será aberta, mas todos os seus campos estarão desabilitados. Para habilitar os campos para que sofram alterações, deve-se clicar no botão Modify e, dessa forma, todos os campos da janela ficarão habilitados para sofrerem as alterações necessárias. Agora, no campo Description deve-se acrescentar a informação "Este problema acontece todas as vezes que se utiliza a opção de desconto no filtro do relatório de Vendas". Também é possível adicionar um arquivo anexo para o registro de defeito, visando assim, facilitar a correção do problema e, para isso, na aba Attchments pode-se incluir qualquer tipo de arquivo, tal como, um print screen da tela com o problema citado anteriormente.

Figura 7. Visualização da Janela para consultar um registro de defeito

Com isso, pode-se observar que o Rational ClearQuest começa a realizar um controle de todas as modificações sofridas pelo registro do defeito. Até agora, o defeito continua com o estado de Submitted, ou seja, apenas cadastrado, e para poder mudar esse estado inicial, designando-o a um profissional para a solução desse defeito, deve-se abrir o registro de defeito novamente. Agora, na janela Visualizar Defect deve-se clicar no botão Assign e na aba Main alterar os valores dos campos Priority para "1-Resolve Immediately" e Owner para "engineer", desta forma, este defeito passa a ter a prioridade mais alta e um profissional responsável para a correção deste problema, e seu estado passa a ser Assigned (ver **Figura 8**). O usuário engineer, é um usuário de exemplo que já vem cadastrado no banco de dados SAMPL que contém vários dados de registros de defeitos e de informações de usuários cadastrados, além de exemplos de consultas e relatórios, auxiliando o aprendizado nesta ferramenta.

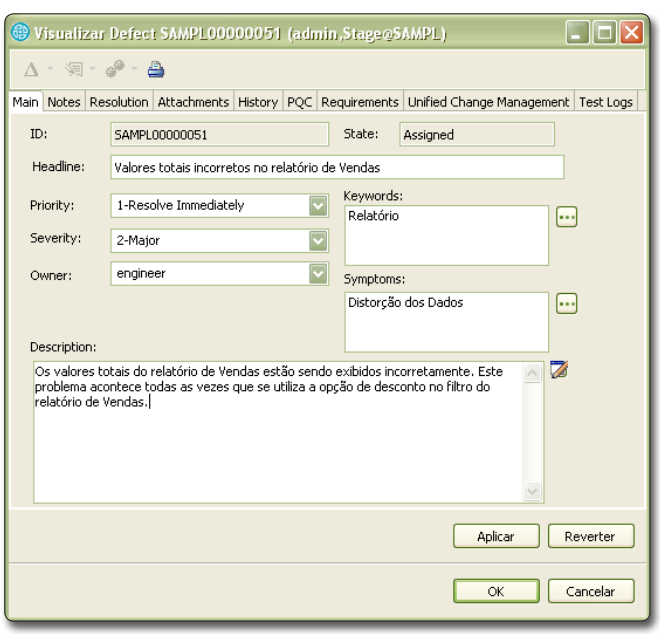


Figura 8. Visualização da Janela Visualizar Defect com estado Assigned

Após ter designado um desenvolvedor ao registro do defeito, o próximo estado deste será de Opened, isto significa, que as próximas alterações deste registro só poderão ser realizadas pelo desenvolvedor responsável pela correção do problema. Desta forma, assim que o erro encontrado no relatório de Vendas for corrigido pelo profissional, esta correção deve ser informada ao registro do defeito no ClearQuest. Para isso, deve-se abrir o registro do defeito no ClearQuest Client em Editar > Localizar Registro > admin,Stage@SAMPL. Na aba Resolution, informar o valor "Fixed" no campo Resolution, para definir que o problema foi resolvido com sucesso e clicar no botão Resolved para finalizar esse registro de defeito.

Com estes passos, pode-se demonstrar a transição do defeito de um estado para o outro ao se utilizar a ferramenta Rational ClearQuest. Na aba History da janela Visualizar Defect, pode-se visualizar toda a evolução de modificações sofridas pelo defeito. O Rational ClearQuest salva todas as alterações sofridas, facilitando assim futuras consultas e comparações, contribuindo em muito para o controle de modificações (ver **Figura 9**).

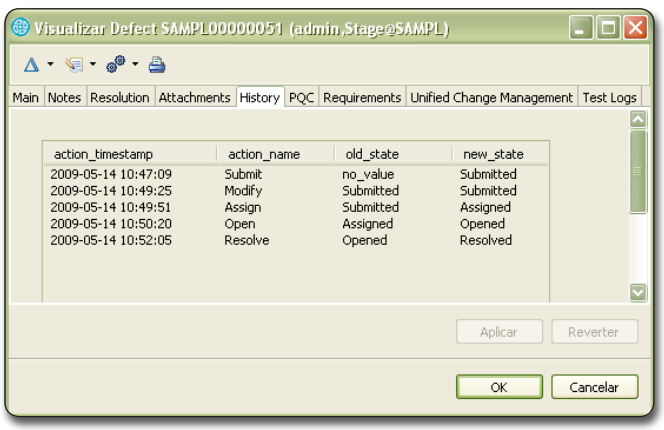


Figura 9. Visualização da aba History do defeito SAMPL00000051

Ao longo do projeto de TI pode ser altamente necessário criar e trabalhar com consultas no Rational ClearQuest. Consultas é o mecanismo pelo qual se procura controles de mudanças de uma maneira fácil e rápida. Ao criar uma consulta, devem-se especificar os critérios de seleção dos controles de mudanças, como por exemplo, retornar todos os controles de mudanças que estejam no estado de Resolvido. O Rational ClearQuest irá exibir os resultados através de uma Query Results View, sendo esses passos descritos a seguir.

Para criar uma consulta utilizando o Assistente de Consulta do Rational ClearQuest, deve-se clicar em Arquivo > Novo > Consultar. Na janela Assistente de Consulta, definir um nome para a consulta como "Defeitos Resolvidos", selecionar o valor Defect e clicar em Personal Queries. Agora, na tela Selecionar Campos, devem-se definir quais campos serão utilizados como filtros da consulta. Para isso, deve-se, na área da janela de Campos, selecionar o campo State e, em seguida, clicar em Avançar (ver **Figura 10**).

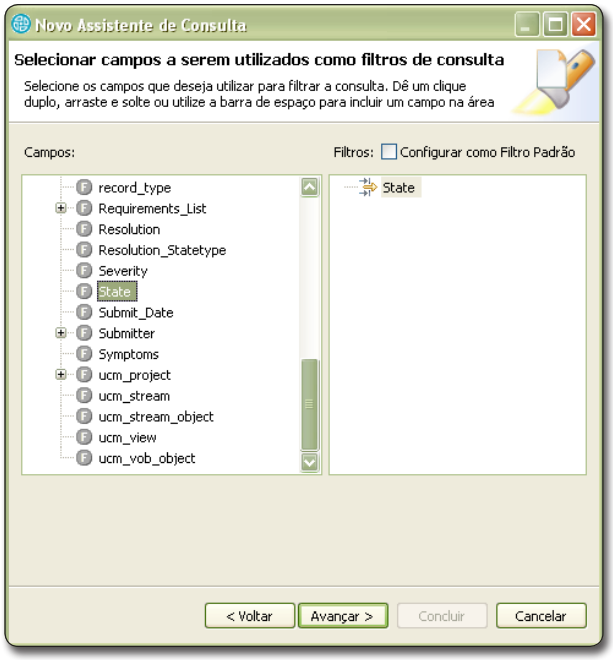


Figura 10. Visualização do Assistente de Consulta

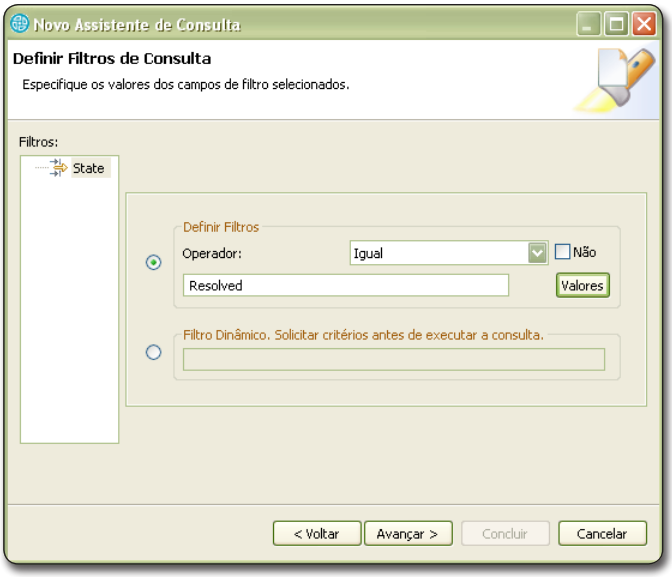


Figura 11. Visualização da janela Definir Filtros de Consulta

Na janela Definir Filtros de Consulta deve-se especificar valores para os campos de filtro, neste exemplo, o resultado da consulta deverá retornar somente os defeitos resolvidos. Para isso, com o filtro State selecionado, deve-se no campo Operador selecionar o valor "Igual" e informar o valor "Resolved", desta forma, somente os registros com State igual a Resolved serão retornados (ver **Figura 11**). O próximo passo será definir quais campos serão exibidos pela consulta. Assim, na janela Definir Campos de Exibição, deve-se selecionar os seguintes campos Headline, Owner, Priority e State, e clicar em Concluir para criar a consulta.

Para executar a consulta, na janela principal do Rational ClearQuest, na aba Navegador deve-se clicar no botão Executar, os dados serão exibidos em uma Query Results View. Pode-se visualizar que o resultado da consulta retornará somente os defeitos que possuem o estado Resolvido (ver **Figura 12**).

No Rational ClearQuest, quando se utiliza uma consulta criada por um membro da equipe de projetos ou compartilha uma de suas consultas, fica mais fácil modificar uma consulta existente para atender a novas necessidades do que criar uma consulta totalmente nova. Para isso, deve-se utilizar a opção de exportar e importar consultas.

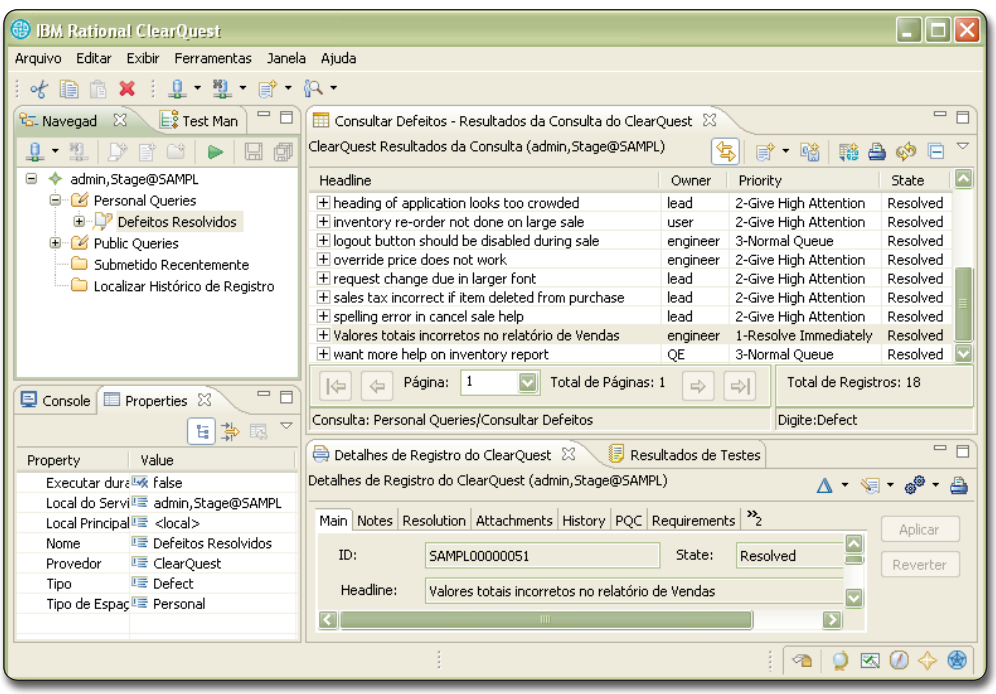


Figura 12. Visualização da janela principal do Rational ClearQuest

Para exportar uma consulta, deve-se na aba Navegador clicar com o botão direito do mouse na consulta "Defeitos Resolvidos" e selecionar Exportar Consulta. No campo Nome do arquivo da janela Exportar Consulta, alterar o nome da consulta, como por exemplo, "Defeitos Resolvidos 1" e, para salvar a nova consulta, clicar em Salvar.

Para importar a consulta, deve-se clicar com o botão direito do mouse na pasta Personal Queries e selecionar Importar. Na janela Importar Consulta, deve-se selecionar o arquivo "Defeitos Resolvidos 1.qry", criado anteriormente, e clicar em Abrir. Com isso, podem-se realizar alterações nessa nova consulta sem modificar a consulta anterior.

Conclusões

Pode-se constatar que mudanças são inevitáveis ao longo do ciclo de vida de projetos e serviços de TI. Além do uso de metodologias para o gerenciamento de mudanças, o uso de ferramentas para o rastreamento e o controle dessas mudanças é um passo importante, gerando melhoria na qualidade tanto no processo de desenvolvimento de software, quanto na melhoria dos serviços prestados para a área de TI.

Conforme visto neste artigo, o guia PMBoK apresenta as melhores práticas para gestão de projetos, e podem ser aplicadas de maneira eficiente em projetos de software. O PMBoK está dividido em nove áreas de conhecimento, e a área de Gerenciamento de Integração, através do seu processo de Controle Integrado de Mudanças é o responsável em controlar e monitorar mudanças ao longo de todo o ciclo de vida de um projeto.

Também se pode demonstrar que a ferramenta Rational ClearQuest fornece um controle flexível de defeitos e alterações em todo o ciclo de vida de um projeto de TI, dando suporte para um efetivo Gerenciamento de Mudanças.

Referências

MAGALHÃES, Ivan Luizio; PINHEIRO, Walfrido Brito. Gerenciamento de Serviços de TI na Prática. São Paulo: Novatec, 2007.

PMBOK. Um Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos. 3 ed. Newtown Square, Pennsylvania: Project Management Institute, Inc. 2004.

PRESSMAN, R. S. Engenharia de Software. São Paulo: Makron Books, 2004

VARGAS, Ricardo Viana. Manual Prático do Plano de Projetos. 3 ed. Rio de Janeiro: Brasport, 2007.

BRAGA, Aclair Rodrigues. Gerência de Projetos - Preparação para a Certificação PMP. Disponível em: <http://www.scribd.com/doc/4905536/braga-preparacao-para-a-certificacao-pmp>. Acesso em: 10 nov 2008.

TECHNET, Microsoft. Material do curso Academia de Gerenciamento. Disponível em: <http://www.microsoft.com/brasil/technet/academia/default.aspx>. Acesso em: 03 nov. 2008.

IBM. Rational ClearQuest Introduction. Disponível em: <http://www.usd.edu/csci/docs/rational/Rational%20ClearQuest/IntroductionClearQuest.pdf>. Acesso em: 14 mai 2009.

Cursos Online

Assinatura

ClubeDelphi PLUS

Mais conteúdo DELPHI por muito menos!

A Revista **ClubeDelphi** oferece para seus assinantes uma série de **Cursos Online** de alto padrão de qualidade . Conheça abaixo os cursos já disponíveis..

Curso em destaque

Aplicações Client/Server com dbExpress e Firebird

Confira neste curso online como criar uma aplicação client/server completa no Delphi 7, utilizando dbExpress e Firebird.
Aprenda também: Como trabalhar com um driver dbExpress específico e gratuito para o Firebird ; Como utilizar Orientação a Objetos, e técnicas como herança visual de formulários e relatórios ; saiba como colocar regras de negócios no banco de dados, usando Triggers e Stored Procedures; E crie relatórios com o Quick Report, Rave Reports e Report Builder, e muito mais.

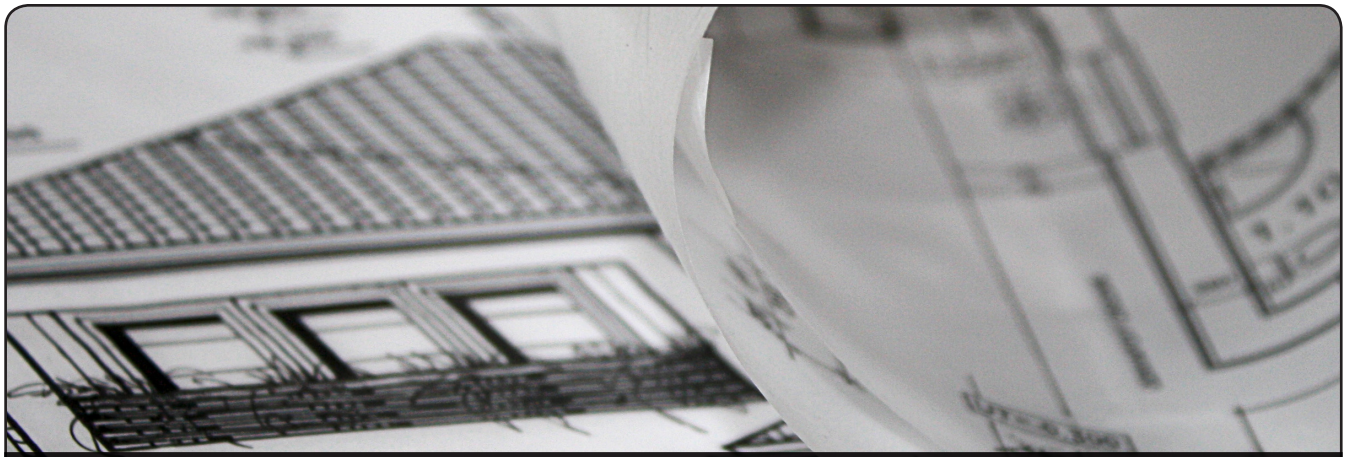
Confira o plano de aula completo:
www.devmedia.com.br/clienteserver

A sua melhor opção de aprendizagem!
Assine a **ClubeDeplhi** e comece já seu treinamento!
www.devmedia.com.br/assine

Outros cursos disponíveis: **www.devmedia.com.br/curso**

- Curso Online - Sistema SysCash
- Aplicações client/server com Windows Forms no Delphi 2006
- Criando uma Aplicação multi-camadas Completa com Delphi
- Aplicações WEB com Delphi 7 (Delphi Win32)

DevMedia GROUP **Mais Informações: www.devmedia.com.br/central - Tel.: 21 3382-5038/ 3382-5025**



Ciclo de Vida da Gestão em Arquitetura de Software

Maximizando a geração de valor de projetos e produtos com arquiteturas de software

Eros Viggiano

eros.viggiano@arkhi.com.br

No ramo de TI desde 1991, trabalha com consultoria e ensino de arquitetura de software. Atua principalmente nas áreas de telecomunicações e finanças como arquiteto de software. É coordenador e professor do curso de Estratégias em Arquitetura de Sistemas no IGTI e professor do IEC/PUC Minas. É bacharel em Ciência da Computação (DCC/UFMG) e especialista em Engenharia de Software (DCC/UFMG). Algumas certificações profissionais relevantes: Sun Certified Enterprise Architect (SCEA), IBM Rational Unified Process 7 (RUP).

Marco Aurélio S. Mendes

marco.mendes@arkhi.com.br

No ramo de TI desde 1990, trabalha com consultoria e ensino de engenharia de software. Possui treze anos de experiência Java e atua profissionalmente nas áreas de melhoria de processos de software e arquitetura de software. É coordenador e professor do curso de Estratégias em Arquitetura de Sistemas no IGTI e professor do IEC/PUC Minas. Possui formação como bacharel e mestre em Ciência da Computação (DCC/UFMG). Algumas certificações profissionais relevantes: IBM SOA Certified Solution Designer, IBM Rational Unified Process 7 (RUP).

A arquitetura de software é uma área recente dentro da engenharia de software, tendo seus primórdios na literatura no começo da década de 90. Não obstante, ela tem obtido cada vez mais visibilidade nos projetos de TI no Brasil. É cada vez mais comum que empresas e projetos tenham pessoas ou times exercendo o papel do arquiteto de software e atividades em cronogramas relacionadas ao desenho de uma arquitetura de software. Mas o que é a arquitetura de software e quais são as atividades que devem ser realizadas pelo arquiteto de software? A arquitetura é uma atividade puramente técnica? Criar uma arquitetura de software é apenas modelar diagramas e implementar provas de conceito? A arquitetura de software produz realmente valor de negócio para um projeto? E como poderíamos organizar as atividades de arquitetura de software para gerar valor concreto para um projeto, produto ou organização?

Este artigo endereça estas e outras questões sobre as atividades de arquitetura de software e propõe um ciclo de

De que se trata o artigo?

O artigo apresenta um ciclo de vida para a gestão das atividades de arquitetura de software em um projeto, de forma a gerar maior valor de negócios para um projeto, produto ou empresa.

Para que serve?

Conhecer e aplicar as atividades essenciais da arquitetura de software em um projeto, tais como: garantir o alinhamento técnico do projeto à organização, atender às restrições de custo, tempo e qualidade de um projeto, identificar e desenvolver requisitos arquiteturalmente significativos, antecipar e mitigar riscos técnicos, realizar a análise e desenho arquitetural, construir provas de conceito, gerar código executável, orientar e acompanhar o time técnico, validar a estabilidade da arquitetura executável e coletar lições aprendidas para o próximo ciclo de atividades arquiteturais.

Em que situação o tema é útil?

No desenvolvimento de projetos de software não triviais, na montagem de novos produtos, na evolução de tecnologias de produtos, em manutenções evolutivas complexas e, sobretudo, em esforço de aumento de alinhamento das ações de TI com as áreas de negócio de uma organização.

vida para organização e ordenação temporal destas atividades em projetos de software de qualquer natureza, com o uso de qualquer tipo de processo de software.

À primeira vista, podemos imaginar que arquitetar um software é escolher que tecnologias um projeto irá adotar, como por exemplo, Java EE ou .NET. Esta visão míope, difundida por "arquitetos empilhadores de frameworks", não pode estar mais longe da realidade. Em uma segunda análise, podemos imaginar que a arquitetura de software consiste de montarmos modelos em linguagens sofisticadas como UML2. Esta visão estrábica difundida por "arquitetos de papel" também está bem longe da verdade.

Coloquemos lentes de correção na visão de arquitetura de software, que foi inicialmente delineada no excelente livro Software Architecture: Perspectives on an Emerging Discipline, ainda em 1996. A arquitetura de software tem por objetivo fazer o melhor uso dos recursos técnicos de um projeto para garantir a maior eficiência possível aos objetivos do projeto, produto e mesmo à visão e missão de uma empresa.

Podemos identificar, nesta visão, atividades tais como:

- garantir o alinhamento técnico do projeto às diretrizes e estratégias tecnológicas de uma área ou organização, bem como à sua cultura organizacional;
- atender as restrições gerenciais de um projeto tais como custo, prazo e competências técnicas da equipe;
- identificar e detalhar requisitos significativos para a arquitetura de software;
- antecipar e mitigar os riscos técnicos de um projeto;
- realizar a análise e desenho técnico da arquitetura através de técnicas de modelagem arquitetural;
- construir provas de conceito e gerar código executável para os principais pontos de risco do projeto;
- orientar e acompanhar o time técnico para garantir a aderência do código construído às diretrizes arquiteturais do projeto;
- validar a estabilidade e objetivos da arquitetura do produto;
- coletar lições aprendidas e promover um novo ciclo de melhorias.

Em verdade, o ciclo de vida da arquitetura de software deve ser compreendido através de um processo mais amplo que aspectos puramente técnicos, que denominamos aqui como a gestão da arquitetura de software.

Arquiteturas de software são sempre desenvolvidas dentro de um contexto de negócios como, por exemplo, a criação ou evolução de um produto, a automação de processos de negócio ou a integração entre sistemas de fornecedores distintos. Assim como a arquitetura de software realiza tecnicamente objetivos organizacionais, ela deve prover feedback do mundo real conforme mostrado na **Figura 1**.

Ciclo de Vida da Gestão da Arquitetura de Software

O ciclo de vida de um projeto organiza as fases do início ao fim das atividades, conforme mostrado na **Figura 2**. Tipicamente as transições entre fases são caracterizadas por algum tipo de entrega e sua respectiva revisão. Para cada fase, o ciclo de vida do projeto permite definir trabalho realizado, entregas, envolvidos e critérios de aprovação e controle [1].

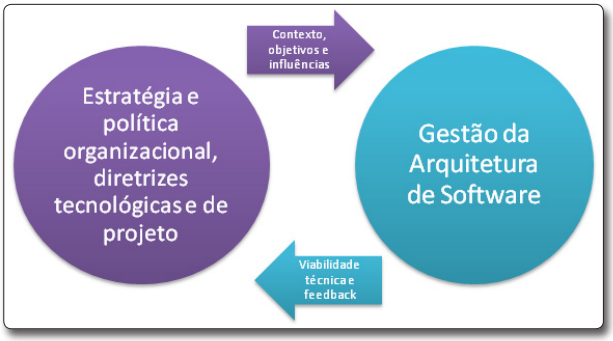


Figura 1. Influências sobre a gestão da arquitetura de software

Figura 2. Fases típicas de um projeto

No caso de projetos de desenvolvimento de software, o ciclo de vida do projeto geralmente respeita um processo de desenvolvimento de software como o IBM Rational Unified Process (RUP). Mais adiante, abordaremos a relação dos processos de desenvolvimento com a gestão da arquitetura de software.

Apresentaremos o ciclo de vida da gestão da arquitetura de software alinhado ao ciclo de vida do projeto. Situaremos as principais atividades desempenhadas pelo time de arquitetura durante toda a execução do projeto de desenvolvimento de software. Entretanto, lembramos que a arquitetura de software pode abranger o escopo de um produto, isto é, além do ciclo do projeto conforme mostrado no **quadro** "Ciclo de vida do produto versus ciclo de vida do projeto".

Ciclo de vida do produto versus ciclo de vida do projeto

O ciclo de vida do produto é mais abrangente que o do projeto. Um produto pode ser criado e evoluído por vários projetos (ver **Figura 3**).

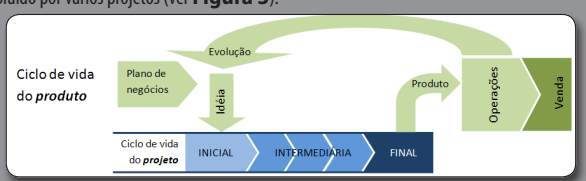


Figura 3. Relação entre produto e ciclo de vida de projeto (Fonte: PMBOK [1])

As atividades da gestão da arquitetura de software são um subconjunto das atividades executadas em um projeto de software. A **Figura 4** define os grupos de atividades do ciclo de vida da gestão da arquitetura que evidenciamos como IDEA (Iniciação-Definição-Edificação-Avaliação).

Figura 4. Grupos de atividades de IDEA, ciclo de vida da gestão da arquitetura de software

Em um típico projeto de desenvolvimento de software, as atividades do grupo I (Iniciação) coincidem com as fases iniciais do projeto e caracterizam o início dos trabalhos arquiteturais. Os grupos D (Definição) e E (Edificação) se intercalam nas fases intermediárias, sendo que D visa definir e validar arquiteturas candidatas e, em E, procura-se garantir que as arquiteturas estão sendo adequadamente realizadas. Enfim, as atividades do grupo A (Análise) ocorrem no fim do projeto e tem por objetivo analisar e aprender com os resultados, objetivando a melhoria de processos e práticas arquiteturais.

A **Figura 5** representa a projeção do ciclo de vida de um projeto típico sobre o ciclo de vida da gestão de arquitetura de software.

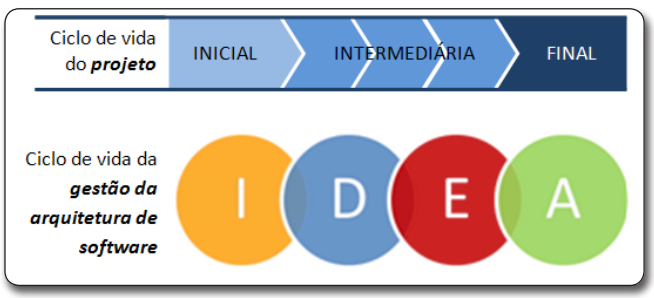


Figura 5. Projeção do ciclo de vida da gestão de arquitetura de software sobre o ciclo de vida do projeto

De uma forma geral, a literatura especializada trata principalmente das atividades relacionadas à definição da arquitetura, em especial, da modelagem e da avaliação arquiteturais. Entretanto, na prática, a disciplina de arquitetura de software requer uma gestão mais complexa. As atividades do arquiteto tendem a envolver questões mais estratégicas que merecem ser contextualizadas no ciclo de vida do projeto.

Dentre aqueles que defendem uma participação mais estratégica do arquiteto de software, destacamos os métodos arquiteturais VAP [2] de Dana Bredemeyer e o CFCAR [3] de Gerrit Muller. A iniciativa de arquitetura de software do SEI, denominada SAT, também ressalta o envolvimento do arquiteto no alinhamento estratégico do produto e trata o business case como um insumo para a arquitetura de software.

O ciclo de vida IDEA serve como uma espécie de moldura para instanciar processos de arquitetura de software. Para projetos que visam o desenvolvimento de um novo produto, o escopo do nosso artigo, sugerimos o processo diagramado na **Figura 6**. Repare que, em termos gerais, o processo arquitetural segue o ciclo de PDCA (Plan, Do, Check, Act).

A **Tabela 1** enumera as atividades e os produtos de trabalho sugeridos pelo IDEA.

Na seção "Atividades da gestão em arquitetura de software", exemplificaremos as atividades e seus produtos de trabalho com um nível um pouco maior de detalhes.

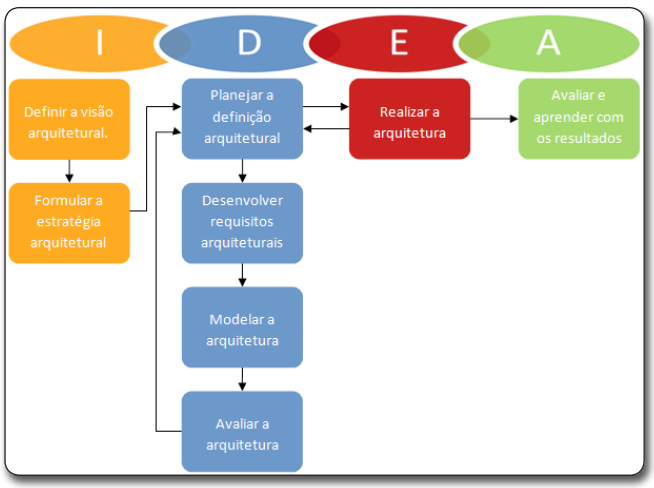


Figura 6. Processo arquitetural para um novo produto de software baseado no IDEA

Grupo	Atividade	Produtos de trabalho
Iniciação	Definir a visão arquitetural	Visão arquitetural
	Formular a estratégia arquitetural	Diretrizes arquiteturais Análise de viabilidade técnica Estratégia arquitetura
Definição	Planejar a definição arquitetural	Plano arquitetural
	Desenvolver requisitos arquiteturais	Requisitos arquiteturais detalhados
	Modelar a arquitetura	Modelo da arquitetura
	Avaliar a arquitetura	Avaliação da arquitetura
Edificação	Realizar a arquitetura do software	Arquitetura executável
Análise	Avaliar e aprender com os resultados	Lições aprendidas Sugestões de melhorias

Tabela 1. Atividades e produtos de trabalho da gestão da arquitetura de software

IDEA e Processos de Desenvolvimento de Software

O IDEA pode ser aplicado sobre qualquer processo de desenvolvimento de software e seu uso tende a ser mais natural em processos iterativos. A seguir, apresentamos simulações de projetos para diferentes processos considerando o ciclo IDEA.

Projetos conduzidos pelo IBM RUP (Rational Unified Process) são marcados pelas fases Iniciação (por vezes, denominada de Concepção), Elaboração, Construção e Transição [2]. A **Figura 7** exercita uma projeção hipotética do ciclo IDEA sobre o ciclo de vida de um projeto dirigido pelo RUP. As atividades da gestão da arquitetura tendem a ser organizadas da seguinte forma nas fases do RUP [3]:

• **Iniciação:** objetiva definir o escopo e aferir a viabilidade do projeto. Coincide com as atividades iniciais (I) da gestão de arquitetura de software e, muitas vezes, um esforço mínimo de definição da arquitetura candidata (D). As atividades de arquitetura devem subsidiar a análise de viabilidade do projeto;

Figura 7. Projeção hipotética do ciclo IDEA sobre o ciclo de vida de um projeto dirigido pelo RUP

Mito	Quem costuma acreditar nisto	Realidade
Arquitetura de software produz "muito papel".	Alguns adeptos de métodos ágeis.	O processo de software adotado determina quais documentos são realmente necessários. Comunica-se somente o estritamente necessário.
Arquitetura de software implica em big design up front [4] (intenção de criar todos os modelos no início do projeto).	Alguns agilistas e arquitetos.	A arquitetura deve respeitar a natureza do método. Em projetos ágeis, a arquitetura do software deve ser evolutiva [5].
Requisitos arquiteturais não podem mudar a partir de um certo momento.	Alguns arquitetos e engenheiros de processos.	Métodos ágeis aceitam mudanças a qualquer momento, tendo impacto ou não sobre a arquitetura. O cliente deve sempre estar ciente das consequências de uma mudança no requisito (arquitetural ou não).
Softwares desenvolvidos com métodos ágeis não têm arquitetura.	Pessoas envolvidas no desenvolvimento do software.	Todo software tem uma arquitetura, independente se alguém a projetou intencionalmente ou não [6].
"Arquiteto de software é somente um novo e pomposo título que programadores pedem para ter em seus cartões." Projetos ágeis não precisam do arquiteto.	Alguns adeptos de métodos ágeis.	Vários métodos ágeis prescindem de papéis. Mesmo que ninguém na equipe tenha o papel ou cargo de arquiteto de software, convém planejar a arquitetura.
Toda a arquitetura deve ser modelada no início do projeto.	Alguns arquitetos de software.	O arquiteto deve respeitar a natureza do projeto. Se o método prescreve "prove com código sempre que possível", é interessante realizar a arquitetura em software executável mesmo que não esteja completamente modelada.

Tabela 2. Mitos sobre métodos ágeis versus arquitetura de software

• **Elaboração:** seu principal objetivo é estabilizar a arquitetura do sistema, conhecida, nesse ponto, como arquitetura executável. Assim, na Elaboração, as atividades de definição da arquitetura (D) se intensificam e há uma forte preocupação em edificá-la (E);

• **Construção:** nesta fase, o projeto está em franco caráter construtivo do software. Ocorre o predomínio de atividades relacionadas à realização da arquitetura (E). O arquiteto deve garantir que a arquitetura é bem compreendida por desenvolvedores, engenheiros de testes, integradores, etc.;

• **Transição:** enfoca a liberação do produto. O arquiteto apóia as ações de qualidade e implantação (E). Na transição, ocorrem as atividades finais da arquitetura, caracterizadas pela análise e aprendizado dos resultados (A).

Atualmente, existe uma aparente tensão entre a comunidade de praticantes de métodos ágeis e arquitetos de software ortodoxos [3]. Os chamados agilistas entendem que os arquitetos produzem "muito papel", enquanto que mudança nos requisitos (principalmente arquiteturais) provoca incômodo em alguns arquitetos de software em qualquer estágio do projeto. Comentaremos rapidamente alguns mitos de agilidade versus arquitetura de software na **Tabela 2**.

Nossa experiência diz que a disciplina de arquitetura de software pode contribuir para a redução de riscos técnicos mesmo em projetos que empreguem métodos ágeis. Para tal, em primeiro lugar, os trabalhos arquiteturais devem respeitar a natureza evolutiva de tais projetos. Em segundo, deve se ater a comunicar modelos arquiteturais apenas na medida exigida pelo método. Por exemplo, se a filosofia de desenvolvimento prega abandonar diagramas após a realização no modelo através do código, o arquiteto assim deve proceder. Outra situação: caso a equipe não faça uso de ferramentas CASE ou de modelagem avançadas, o arquiteto pode considerar a modelagem coletiva usando um quadro branco ou flip chart.

O ciclo de vida de gestão da arquitetura que apresentamos acomoda perfeitamente os projetos ágeis. Na **Figura 8**, apresentamos uma projeção hipotética do ciclo IDEA sobre um projeto ágil dirigido pelo SCRUM ou XP. Como pode ser percebido, os grupos de definição (D) e edificação (E) arquiteturais se revezam constantemente durante as iterações intermediárias do projeto. E, mesmo nas últimas iterações, a descoberta ou mudança de um requisito arquitetural pode implicar em redefinição (D) na arquitetura. As iterações do SCRUM são denominadas sprints. As primeiras iterações do XP compõem as fases de Exploração e Planejamento.

O ciclo de vida IDEA e o processo exposto na Figura 6 comportam a realização parcial da arquitetura, isto é, prova-se com código mesmo que a arquitetura não esteja completamente definida. Esta abordagem é totalmente aderente à filosofia dos métodos ágeis.

Por fim, mesmo que todo o senso moderno da engenharia de software não recomende processos em cascata, é possível vislumbrar uma possível aplicação do ciclo de vida IDEA nessa situação (nossa experiência em gestão arquitetural é limitada a projetos baseados no Processo Unificado e em métodos ágeis. A aplicação descrita para processos em cascata é meramente especulativa. Ainda hoje, existem alguns projetos - principalmente aqueles derivados de editais públicos – que sustentam este tipo de desenvolvimento.). A **Figura 9** demonstra esta situação, apesar de um tanto inusitada.

Figura 8. Projeção hipotética do ciclo IDEA sobre o ciclo de vida de um projeto ágil

Figura 9. Aplicação hipotética do IDEA em processo cascata

Atividades da Gestão de Arquitetura de Software

Iremos ilustrar as atividades do ciclo de vida do IDEA através de um exemplo fictício no contexto de empréstimos bancários na modalidade de crédito consignado, um empréstimo com desconto em folha de pagamento do trabalhador e, portanto um empréstimo de menor risco.

Definição do Problema – Visão do Produto

É recomendável que o produto a ser modelado tenha uma visão definida. Uma visão é um documento que fornece as diretrizes e funcionalidades básicas do produto que deve ser construído. Fornecemos aqui um extrato da visão do produto de empréstimo bancário, resumido na **Tabela 3**.

O problema de	demora e ineficiência na análise e concessão de crédito consignado.
Afeta	trabalhadores aposentados, servidores públicos, funcionários da iniciativa privada, diretoria e acionistas do banco Optimus.
Impacto atual é	pequenos volumes de empréstimos realizados, grande demora para os clientes, custos mais altos de financiamento, perda de clientes para outras financeiras e bancos.
Uma solução de TI bem sucedida seria	a automação do processo de análise e concessão de crédito consignado através da criação de uma solução de portal que integre os clientes de empréstimo, as bases de dados com as informações históricas destes clientes e informações de clientes disponíveis em bases de dados de terceiros tais como SERASA e Secretaria da Receita Federal
Benefícios esperados	• aumento do número de vendas. • redução dos custos de suporte com TI. • evolução facilitada.

Tabela 3. Definição do problema de crédito consignado do banco Optimus

Incluímos na **Tabela 4** as principais necessidades de negócio do produto a ser modelado, que também fariam parte do documento de visão do produto.

Necessidades	Prioridade	Características	Planejamento para Liberação
Portal Web para Automação da Análise e Concessão de Empréstimos de Crédito Consignado	Alta	• Usabilidade facilitada • Solicitação de Empréstimo • Abertura de empréstimo • Acompanhamento de empréstimo • Simulações de financiamento • Análise de risco • Concessão de Empréstimo • Acompanhamento de Pagamentos	1.0
Eliminação de formulários manuais	Alta	• Cadastro unificado de clientes • Formulário unificado para solicitação de empréstimo	1.0
Portal para dispositivos móveis.	Baixa	Abertura e acompanhamento de solicitações de empréstimo	2.0
Interface EDI para conveniados	Baixa	• Abertura e acompanhamento de solicitações de empréstimo por conveniados	2.0

Tabela 4. Definição das principais necessidades de crédito consignado do banco Optimus

Iniciação

As tarefas iniciais do time de arquitetura devem garantir o alinhamento técnico da arquitetura de software com a visão do produto e demais premissas organizacionais (restrições orçamentárias e temporais, conhecimentos da equipe e cultura organizacional). Para isso, devemos desenvolver a visão e estratégia arquiteturais.

Visão Arquitetural

A visão arquitetural nasce a partir da visão do produto e dos princípios técnicos do projeto, isto é, os objetivos primários da arquitetura de software. Por exemplo, no nosso contexto bancário poderíamos ter o conjunto de princípios da **Figura 10**.

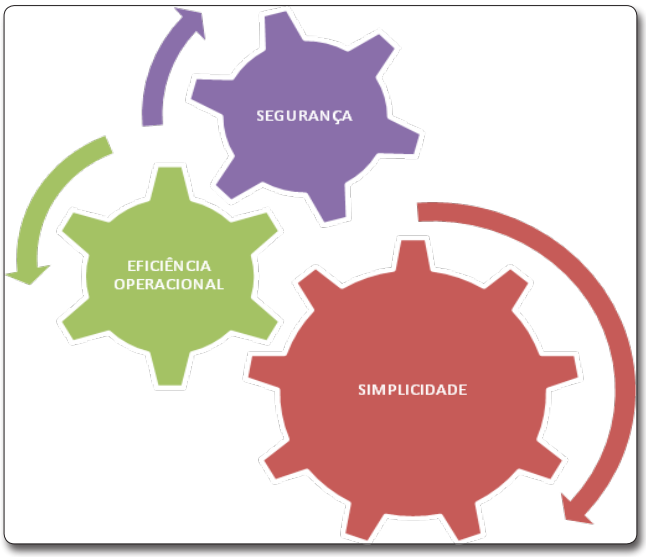


Figura 10. Princípios arquiteturais do sistema de crédito consignado

Esses princípios norteiam qualquer atividade técnica do projeto. Por exemplo, ao avaliar uma determinada solução, framework ou técnica, os confrontamos com os princípios arquiteturais. Soluções ou técnicas que promovam mais segurança ou eficiência operacional teriam preferência sobre soluções menos seguras ou menos eficientes.

A visão arquitetural exprime o plano arquitetural e apresenta uma primeira visualização arquitetural, a visualização de negócio. A visualização de negócio resumida para o nosso produto é expressa na **Figura 11**.

Figura 11. Visualização de negócio do sistema de crédito consignado

Estratégia Arquitetural

A formulação da estratégia arquitetural é a outra atividade do grupo de iniciação do IDEA. Em suma, o arquiteto deve realizar uma análise estratégica e desenvolver a estratégia da arquitetura de software.

Um dos produtos de trabalho da formulação estratégica é a identificação das diretrizes arquiteturais. As diretrizes arquiteturais são os requisitos em alto nível significativos para o time de arquitetura de software e refletem decisões que são difíceis de serem revertidas. Diretrizes arquiteturais devem ser identificadas explicitamente pelo time de arquitetura de software com apoio dos analistas de negócio, analistas de requisitos e demais interessados técnicos do projeto.

Geralmente não encontramos um número muito grande de diretrizes arquiteturais no início de um projeto de software (valores típicos entre 5 a 20). Essas diretrizes normalmente refletem o subconjunto de requisitos prioritários (na ótica dos interessados do produto) e de alta complexidade técnica. É interessante notar que as diretrizes arquiteturais podem advir de requisitos funcionais, de requisitos não-funcionais (atributos de qualidade) e de restrições tecnológicas.

No nosso exemplo, foram identificadas as seguintes diretrizes arquiteturais (As diretrizes foram propositadamente simplificadas e contém palavras ambíguas como excelente e alta disponibilidade. Em projetos reais, métodos de verificação da qualidade de escrita de requisitos como o SMART ou métodos de desenvolvimento formais de requisitos arquiteturais como o SEI QAW poderiam ser usados para apoio ao time de arquitetura):

- Suporte a 1000 usuários simultâneos em períodos de pico;
- Tempo de resposta inferior a sete segundos;
- Alta disponibilidade em horário comercial;
- Solicitação de empréstimo;
- Simulações de financiamento;
- Análise de risco de empréstimo;
- Portal para a hospedagem de páginas e elementos visuais;
- Portal para dispositivos móveis;
- Operação em plataforma .NET;
- Segurança com autenticação baseada em LDAP/Kerberos e transporte seguro para comunicação com clientes;
- Suporte aos navegadores Internet Explorer 6.0/7.0 e Firefox 3.0;
- Interoperabilidade com SGBD Sybase;
- Interoperabilidade com plataforma alta (CICS/COBOL).

O time de arquitetura deve também elencar e analisar os principais riscos técnicos do projeto que, na nossa abordagem, chamamos de riscos arquiteturais. Exemplos de riscos que poderiam compor o problema:

- Conhecimento da equipe na plataforma .NET não é avançado;
- Conhecimento da equipe na plataforma .NET para ambientes móveis é precário;
- Não existe experiência prévia do Banco Optimus em integração com plataforma alta (CICS/COBOL).

O time de arquitetura também pode endereçar oportunidades para a empresa com o desenvolvimento de uma arquitetura de software robusta para o produto. Exemplos de oportunidades poderiam incluir:

Aumento da segurança da informação com o aprendizado do protocolo Kerberos e outros aspectos de segurança da informação, o que pode eliminar riscos de auditoria e elevar o valor do banco para diretores e acionistas;

Capacitação do time técnico em tecnologias móveis.

A estratégia arquitetural também endereça aspectos como o estilo arquitetural e a metáfora sistêmica. O estilo arquitetural, como na arquitetura da construção civil, classifica um produto quanto ao seu estilo primário.

A metáfora sistêmica, idéia originária da metodologia XP, é um recurso visual e muitas vezes lúdico para explicar a arquitetura a pessoas com poucos conhecimentos técnicos. Um exemplo para metáfora sistêmica é representado na **Figura 12**.

O estilo e a metáfora sistêmica do nosso exemplo estão representados na **Tabela 5**.

Estilo Arquitetural	O nosso sistema é um sistema de informação com diferentes tipos de visualizações (portais Web e móveis) e interoperabilidade com sistemas legados. Portanto, o estilo arquitetural associado ao nosso sistema é um sistema n-tier, ancorado no padrão arquitetural MVC (Model View Controller).
Metáfora Sistêmica	O nosso sistema está desenhado conforme um bolo em camadas (camada visual, camada de negócios e camada de interoperabilidade com banco de dados e legados).

Tabela 5. Estilo arquitetural e metáfora do sistema de crédito consignado

Figura 12. Metáfora Sistêmica

Definição

A definição é o grupo de atividades do IDEA que lida com análise e desenho arquitetural dos requisitos sob análise. Grande parte do trabalho que arquitetos usualmente praticam, principalmente a modelagem arquitetural, está situada neste grupo de atividades. No artigo corrente, abordaremos a definição arquitetural com um nível muito superficial de detalhes.

A definição no IDEA, entretanto, tende a ser mais ampla e lida com:

- O planejamento das atividades de definição da arquitetura para a iteração ou fase atual do projeto. Arquiteturas de software são naturalmente evolucionárias (o conceito de arquiteturas evolucionária é descrito em processos como o OpenUP ou no

método de Arquiteturas Ágeis (Agile Architectures) de Scott Ambler) e evolutivas e, portanto, o ciclo de definição parte para o endereçamento de um determinado subconjunto dos requisitos arquiteturais.

• O desenvolvimento dos requisitos a partir das diretrizes arquiteturais. Técnicas como o FURPS+ [9], a ISO SQUARE [10] e o modelo SMART [11] podem ser usadas para apoiar neste processo de detalhamento de requisitos técnicos. O uso de cenários [12] pode ser útil para representar requisitos arquiteturais refinados. É comum encontrarmos sistemas com várias dezenas de requisitos arquiteturais discretos.

• A modelagem arquitetural através do uso de múltiplas visualizações. Técnicas como o modelo de visualizações 4+1 [9], de Phillipe Kruchten, ou o modelo SEI conhecido por V&B [10] poderiam ser usados aqui.

• O detalhamento das táticas arquiteturais. Uma tática arquitetural lida com um determinado aspecto da arquitetura. Por exemplo, poderíamos endereçar uma tática para autenticação e single sign-on com o LDAP/Kerberos.

• A exploração de alternativas de análise e desenho. A avaliação e comparação destas alternativas e a proposta da arquitetura parcial para o conjunto de requisitos endereçados até o momento.

Exemplos de visualizações para o modelo do nosso exemplo são fornecidos nas **Figuras 12**, **13**, **14** e **15**. A **Figura 12** apresenta o conceito da metáfora arquitetural derivada de escolas ágeis como o XP. No nosso exemplo, o padrão referenciado é o padrão camadas (Layers), que denota uma organização em níveis de apresentação, domínio e interoperabilidade com sistemas legados e bancos de dados.

A **Figura 13** apresenta uma visualização lógica, que captura os grandes agrupamentos do domínio da aplicação. Cada grande elemento é capturado como um pacote UML, que pode apresentar ou requerer dependências de outros pacotes. Esta visão fornece um primeiro "diagrama de contexto" para iniciados no projeto e também permitir expressas dependências para a montagem de um cronograma de trabalho.

A **Figura 14** apresenta uma visualização de componentes (que são elementos físicos como DLLs Windows ou .JARs em Java). Esta visão captura a gestão de configuração dos elementos centrais a serem produzidos pelo time de projeto e dependências para a sua operação em produção. Por exemplo, vemos nesta figura que o domínio que cuida do processo de negócio de consignação de crédito depende do servidor de integração (MS BizTalk), que por sua vez depende de um adaptador de fila de mensagens para que as mensagens sejam enviadas ao sistema do Banco Central do Brasil.

Finalmente, vemos uma visão de provisionamento (máquinas e servidores de infra-estrutura) do nosso software na **Figura 15**. Vemos, por exemplo, que teremos um servidor dedicado para hospedar o banco de dados e um servidor dedicado para hospedar o código.

Estes modelos, em verdade, seriam um pequeno fragmento de uma solução completa para este exemplo hipotético. Um modelo arquitetural completo não é apresentado aqui, naturalmente, por questões de espaço.

No mundo real, um dos trabalhos de um arquiteto de sistemas é definir que visões são ou não aplicáveis conforme os riscos técnicos e requisitos do projeto sendo endereçado.

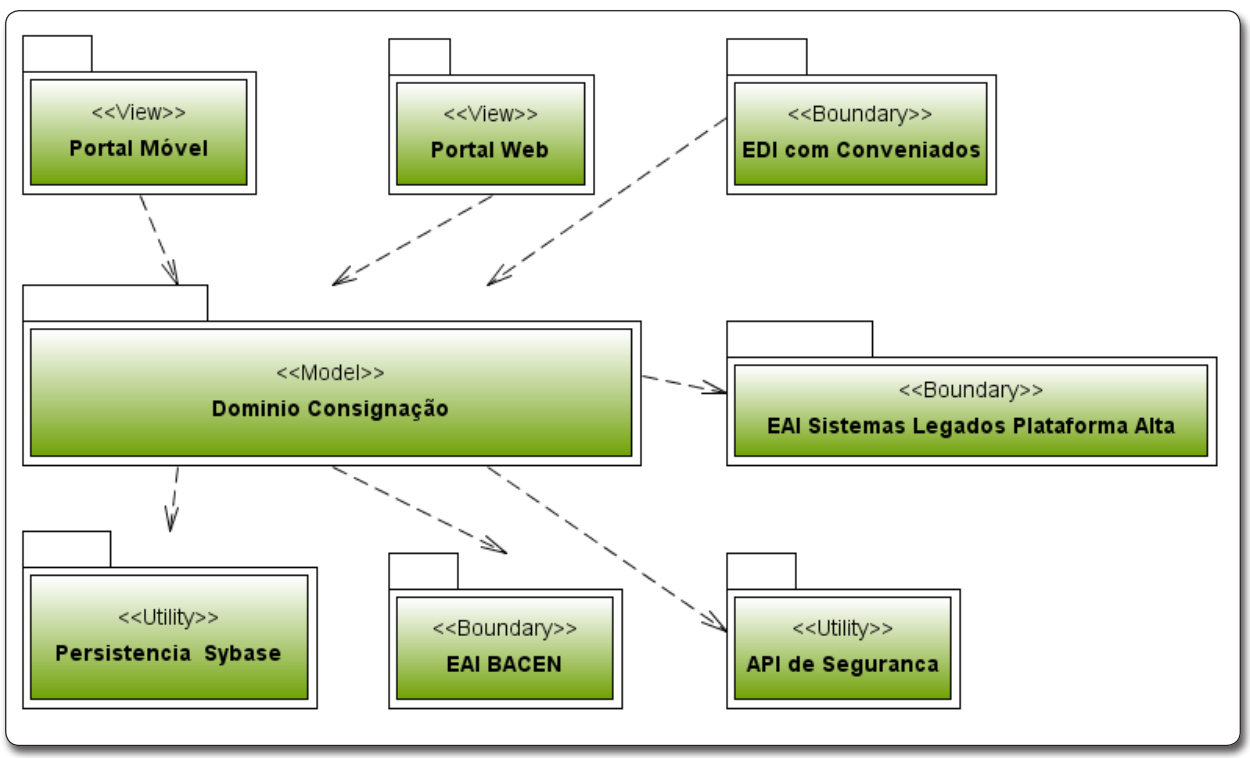


Figura 13. Visualização lógica do sistema de crédito consignado

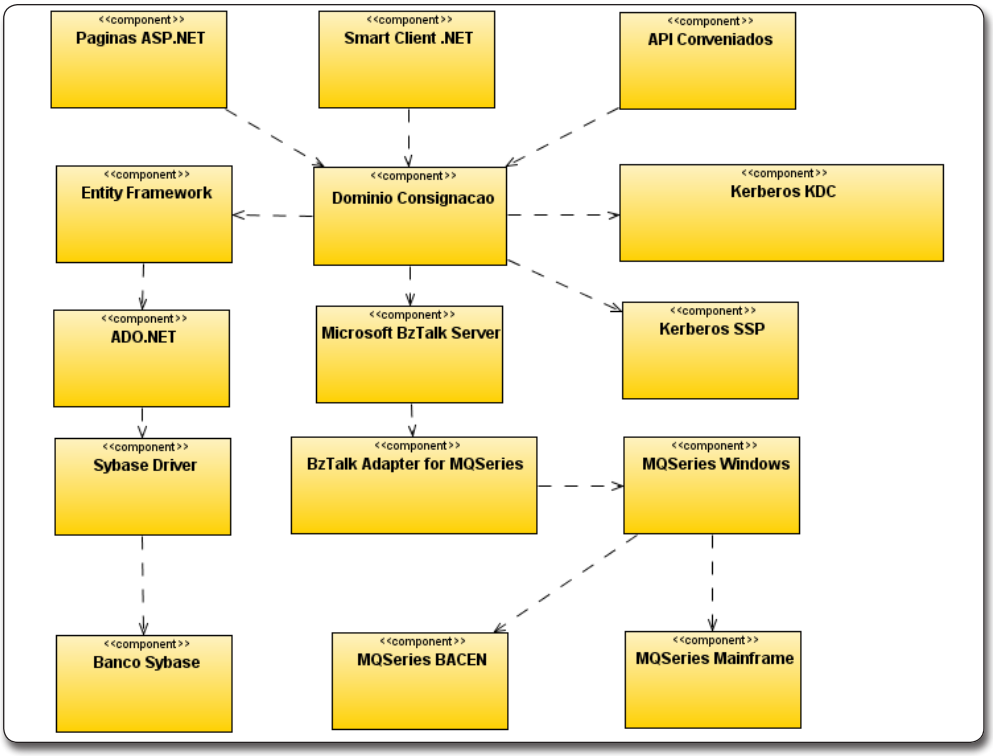


Figura 14. Visualização de implementação do sistema de crédito consignado

As atividades do grupo de definição arquitetural são encadeadas em um ciclo. Cada execução do ciclo produz uma arquitetura candidata que, potencialmente, pode ser realizada em código (atividade do grupo de Edificação). O ciclo se repete até que se considere que uma arquitetura estável foi atingida.

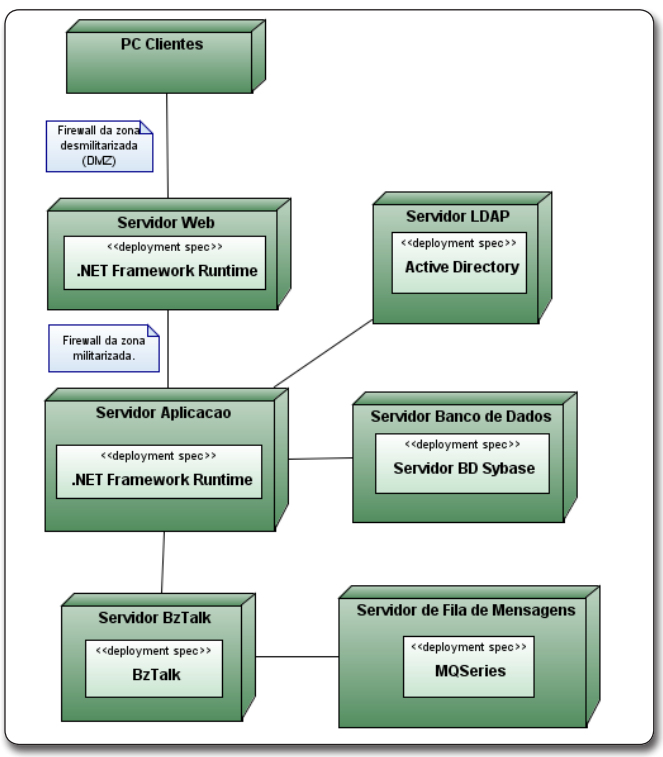


Figura 15. Visualização de implantação do sistema de crédito consignado

Nota do DevMan

Ciclo PDCA

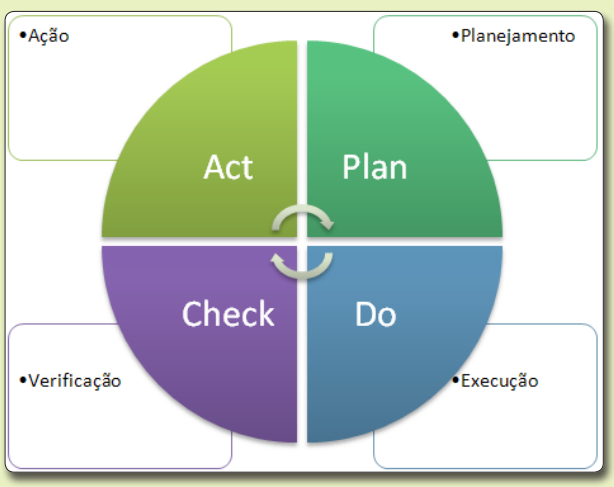
O ciclo PDCA, também conhecido por ciclo de Shewhart ou ciclo de Deming, é um processo em quatro passos para solução de problemas. O PDCA foi criado por Walter Shewhart e popularizado por W. Edwards Deming. Resumidamente, as fases significam:

Plan (planejamento): estabelece objetivos e processos para atingir os resultados;

Do (execução): execução das atividades planejadas;

Check (verificação): monitoração ou estudo periódico dos resultados, consolidando-os com o planejado;

Act (ação): agir conforme o estudo promovido pela verificação com fins de melhorar a qualidade, a eficiência e a eficácia.



Edificação

Até este momento temos normalmente uma arquitetura candidata, i.e, uma arquitetura baseada em estudos e experiência do time e também na análise de alternativas. Entretanto, devemos provar a arquitetura com código robusto.

O IDEA propõe, neste particular, um conjunto de atividades de edificação. A edificação realiza a arquitetura, isto é, fornece evidências concretas através de código executável que a arquitetura acomoda os requisitos do projeto.

A edificação é realizada através do esforço conjunto do time de arquitetura do projeto (arquiteto e desenvolvedores especialistas). O resultado da edificação é uma arquitetural executável, que poderia ser gerada em pequenos incrementos (builds) nas fases iniciais de um projeto.

No nosso exemplo, poderíamos elencar um plano de builds (códigos com maturidade Alfa - um produto alfa é um código executável, mas que ainda não possui estabilidade de produção. Ele pode (e deve) ser usado para validar requisitos arquiteturais com usuários chave, estabelecer níveis maiores de confiança e reduzir riscos técnicos do projeto) a ser construído para provar aspectos da arquitetura com código executável. Seis builds são mostrados para exemplificar nosso produto bancário:

- Alfa 1 – Prova de Conceito Single Sign On Kerberos;
- Alfa 2 – Prova de conceito com o fluxo de solicitação, abertura e acompanhamento de empréstimos;
- Alfa 3 – Prova de conceito de interoperabilidade e troca de informações (EDI) com Microsoft MQSeries para o Banco Central;
- Alfa 4 – Prova de conceito de escalabilidade para 1000 usuários simultâneos;
- Alfa 5 – Implementação do conjunto de casos de uso de solicitação e abertura de empréstimo;
- Alfa 6 - Implementação do conjunto de casos de uso de simulações de financiamento.

Ao final do ciclo de atividades de edificação, temos uma arquitetura provada para os mecanismos centrais elencados na fase de definição.

O trabalho de arquitetura não se encerra ao termos uma arquitetura executável e estável. É muito fácil quebrar uma arquitetura definida e, portanto, o arquiteto deve acompanhar o time na resolução contínua de dúvidas e na verificação do código sendo construído.

Além do trabalho junto ao time de desenvolvimento, o arquiteto também apóia outros profissionais como o engenheiro de testes, o analista de integração, gerente de configuração, etc. O objetivo é garantir que todos compreendam perfeitamente a arquitetura definida.

Avaliação

O IDEA está inspirado nas idéias clássicas do PDCA e como tal possui um ciclo de atividades para análise de lições aprendidas. A avaliação deve ser ampla e envolver o time técnico, time gerencial, time de analistas e demais interessados no projeto. As ações de arquitetura devem ser avaliadas e os erros e lições aprendidas devem ser discutidas e comunicadas para toda a equipe.

A arquitetura, como peça estratégica em uma empresa, deve gerar informações para a gestão do conhecimento corporativo e promover cada vez mais alinhamento entre as ações técnicas e as necessidades de negócio de uma organização.

Conclusões

A disciplina de arquitetura de software tende a ser mais eficaz se aplicada em um contexto mais amplo e responsável – a gestão da arquitetura de software. IDEA, o ciclo de vida da gestão da arquitetura de software, encadeia as atividades de arquitetura de software de uma forma compreensível e organizada.

O artigo corrente concentrou-se no processo de arquitetura de software para a criação de um produto, mas, de uma forma análoga, os conceitos podem ser aplicados para famílias de produtos e arquiteturas de referência. ●

Referências

1. Project Management Institute. [PMBOK] A Guide to the Project Management Body of Knowledge. s.l. : Project Management Institute, 2004.
2. Malan, Ruth and Bredemeyer, Dana. Architecture as a Business Competency. [Online] http://www.bredemeyer.com/pdf_files/WhitePapers/ArchitectureAsBusinessCompetency.PDF.
3. Muller, Gerrit. CAFCR: A Multi-view Method for Embedded Systems Architecting; Balancing Genericity and Specificity. [Online] http://www.gaudisite.nl/info/Thesis.info.html.
4. IBM. IBM Rational Unified Process 7.5. s.l. : IBM.
5. Rozanski, Nick and Woods, Eoin. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. s.l. : Addison-Wesley Professional, 2005.
6. Ambler, Scott. Big Modeling Up Front (BMUF) Anti-Pattern. Agile Modeling. [Online] 2007. [Cited: 04 02, 2009.] http://www.agilemodeling.com/essays/bmuf.htm.
7. InfoQ. Paulo Merson on Documenting Application Architectures Using UML 2.0. [Online] InfoQ, 11 27, 2008. [Cited: 04 02, 2009.] http://www.infoq.com/news/2008/11/paulo-merson-architecture.
8. IBM Rational Quality Manager. Entrevista: Grady Booch e Scott W. Ambler - Architecture? Agile Don't Need No Architecture! [Online] 2009. [Cited: 04 02, 2009.] http://www.facebook.com/video/video.php?v=1090941519721.
9. Grady, Robert. Practical software metrics for project management and process improvement. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1992. ISBN:0-13-720384-5.
10. ISO. Software Engineering -- Software product Quality Requirements and Evaluation (SQuaRE) -- Guide to SQuaRE. ISO/IEC 25000:2005. [Online] 2005. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35683.
11. Mannion, Mike and Keepence, Barry. SMART requirements. ACM SIGSOFT Software Engineering Notes. 1995, Vol. 20, 2.
12. Bass, Len, Clements, Paul and Kazman, Rick. Software Architecture in Practice. 2nd. s.l. : Addison-Wesley, 2003.
13. Kruchten, Phillippe. The 4+1 View Model of Architecture. IEEE Software. 6, 1995, Vol. 12, 6.
14. Clements, Paul, et al. Documenting software architectures: views and beyond. ICSE '03: Proceedings of the 25th International Conference on Software Engineering . 2003.

Cursos Online

Web Mobile PLUS

A **Revista WebMobile** oferece para seus assinantes uma série de Cursos Online de alto padrão de qualidade **.Conheça abaixo os cursos já disponíveis.**

Curso de .net em destaque

Aplicação completa de orçamento Doméstico no Visual Studio 2005

Confira neste curso online como criar uma aplicação completa no Visual Studio 2005, usando ASP.NET, Web Services, Mobile e muito mais! Veja como criar uma aplicação de orçamento doméstico, usando diagrama de classes de uma forma muita produtiva, criar classes de negócios de acesso a dados.

Confira o plano de aula completo:
www.devmedia.com.br/domesticovs

Curso de Java em destaque

Introdução ao desenvolvimento para celulares com J2ME

Confira neste curso os principais recursos do J2ME. Aprenda também o passo a passo para criar sua primeira aplicação J2ME. Neste curso você irá aprender diversas funcionalidades desta tecnologia para desenvolvimento de dispositivos móveis.

Confira o plano de aula completo:
www.devmedia.com.br/celularesj2me

Assine a **WebMobile** e comece já seu treinamento!
www.devmedia.com.br/assine

A sua melhor opção de aprendizagem!

Em breve mais novidades para você! www.devmedia.com.br/curso

DevMedia GROUP **Mais Informações: www.devmedia.com.br/central - Tel.: 21 3382-5038/ 3382-5025**

Análise da Arquitetura de Software

Essencial para Arquitetura 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 80 artigos publicados, tendo feitos 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.

De que se trata o artigo?

Apresenta o uso de requisitos arquiteturais na análise da arquitetura de software e a destaca no desenvolvimento de um sistema de software.

Para que serve?

Entender o papel da análise arquitetural dentro do processo de desenvolvimento de software orientado para arquitetura e definição da arquitetura de software a ser adotada.

Em que situação o tema é útil?

Identificação de requisitos arquiteturais importantes a um sistema de software em desenvolvimento e construção de cenários para definição da arquitetura e como esta suporta esses requisitos.

A arquitetura de software de um sistema serve para definir a organização das funcionalidades desse sistema e propriedades ou requisitos não funcionais suportados pelo mesmo. Características peculiares da arquitetura determinarão quais propriedades serão preponderantes. A necessidade da arquitetura de software prover suporte a um conjunto de requisitos, geralmente conflitantes, requer que uma análise arquitetural seja realizada, tema tratado neste artigo.

Desenvolvimento de Sistemas de Software

O processo de desenvolvimento de sistemas de software, similarmente a outros sistemas, compreende três fases genéricas – definição, desenvolvimento e manutenção – conforme ilustrado na **Figura 1**.

A fase de definição engloba a identificação de informações que deveriam ser processadas, funções e desempenho desejados, tipo de interface a ser utilizada, tarefas que o sistema deveria prover suporte, perfil de usuários do sistema, dentre outras.

A fase de desenvolvimento concentra-se no projeto de estruturas de dados e arquitetura de software do sistema, conversão do projeto para alguma linguagem de

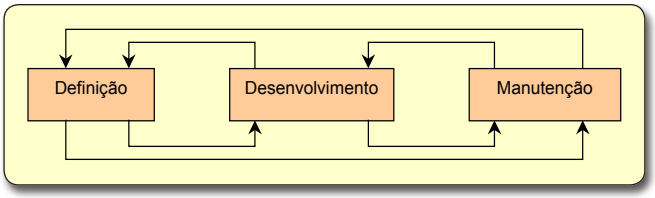


Figura 1. Fases genéricas no processo de desenvolvimento de software

programação, realização de testes e avaliação. Finalmente, a manutenção considera modificações e/ou correções necessárias no sistema a fim de que este atenda aos requisitos do usuário. Perceba que o processo de desenvolvimento de um sistema de software tem duas grandes atividades de interesse: o desenvolvimento da porção de software que implementa as funcionalidades do sistema, e a atividade que a antecede e norteia o desenvolvimento, que é o projeto da arquitetura de software. Essa última atividade é resultado da análise arquitetural que provê informações para decisões arquiteturais, como ilustrado na **Figura 2**. A análise arquitetural e os critérios para definição de uma arquitetura são apresentadas nas seções subsequentes.

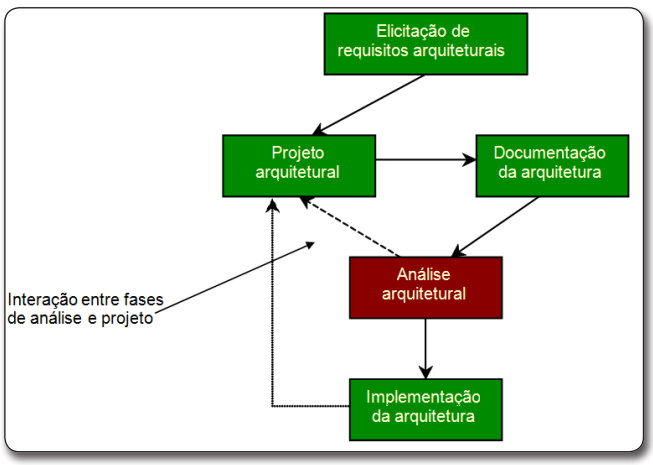


Figura 2. Etapas no projeto da arquitetura de software.

Análise da Arquitetura de Software

O projeto da arquitetura de software é uma etapa essencial no desenvolvimento de sistemas de software de grande porte. Dentro deste contexto, a arquitetura de software é fundamental para o desenvolvimento de software onde se tem funcionalidades distintas, sendo concebidas a partir da mesma arquitetura base. Entretanto, antecedendo a etapa de projeto da arquitetura de software, há a necessidade de fazer o levantamento dos requisitos do sistema, como mostrado na **Figura 2**.

De um modo geral, a análise e projeto de um sistema geralmente engloba as atividades de:

- **Desenvolvimento de modelo arquitetural** – Os dados são coletados durante a elicitação de requisitos a fim de serem analisados e incorporados ao modelo arquitetural (isto é, o produto resultante). Neste caso, o modelo passa a ser o elemento central da análise.
- **Melhoria e síntese de uma solução** – Incrementam-se as informações descritas no modelo arquitetural (inicial).
- **Análise da solução** – A análise da solução é realizada em termos do modelo arquitetural que se tem em mãos, podendo identificar a necessidade de refinar o modelo.

Perceba que o modelo arquitetural pode ser considerado como um 'esboço' inicial da arquitetura de software do sistema em desenvolvimento. É importante ainda observar que a análise de uma arquitetura de software é um processo iterativo no qual são feitos refinamento de um modelo arquitetural inicial, derivado a partir de um conjunto de requisitos arquiteturais. Em cada iteração, uma mini análise ocorre, refinando a arquitetura proposta inicialmente. Esse processo é ilustrado na **Figura 3**.

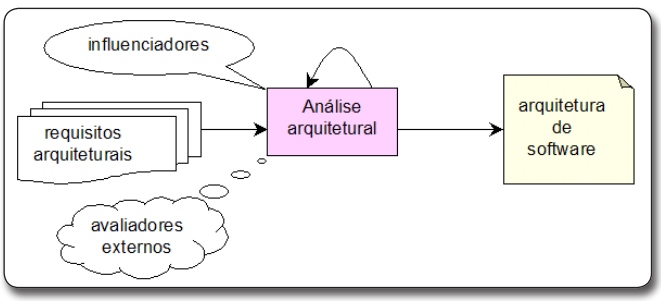


Figura 3. Análise arquitetural.

Entretanto, note que não é tarefa fácil validar uma arquitetura de software quanto ao suporte que oferece a um conjunto de requisitos. É muitas vezes difícil associar novas arquiteturas a atributos de qualidade. Observa-se, geralmente, que os desenvolvedores concentram-se nos aspectos funcionais das arquiteturas. Assim, o principal propósito da análise arquitetural é verificar se os requisitos arquiteturais têm sido levados em conta durante o desenvolvimento de um sistema de software.

Propriedades da Arquitetura de Software

A arquitetura de software é uma parte essencial de um sistema de software complexo. Com a crescente complexidade de sistemas de software, a especificação global do sistema, ou seja, sua arquitetura passa a ser um aspecto de maior significado quando comparado à escolha de algoritmos ou estruturas de dados. Um sistema possui um conjunto de requisitos arquiteturais, envolvendo atributos de qualidade (ou requisitos não funcionais) e atributos de projeto. Esses requisitos constituem propriedades desejadas ou apresentadas por uma arquitetura de software. Dentre esses requisitos, uma arquitetura de software apresenta propriedades importantes que podem ser vistas como intrínsecas a ela. Tais propriedades são: eficiência, integridade e flexibilidade

A eficiência pode ser descrita como a quantidade de recursos computacionais necessários para que um programa realize suas funções. Esses recursos computacionais podem ser vistos em termos da capacidade de armazenamento e processamento. Um dos principais requisitos associados à eficiência é o desempenho. O desempenho é um requisito não funcional essencial para um sistema de software e constitui num enorme desafio

lidar com tal requisito durante o desenvolvimento de um sistema. Por exemplo, seria possível ter os seguintes requisitos de desempenho quando do desenvolvimento de um sistema de software para administração de cartões de crédito:

• Obter um bom tempo de resposta para autorização de vendas com cartão de crédito. O termo bom poderia ser traduzido em 7 segundos.

• Obter um bom uso de memória para armazenar as informações de um cliente. Poderia ser especificado o máximo de 10 Kb para armazenar as informações de cada cliente.

A eficiência é uma propriedade de suma importância nas decisões de projeto e tem forte influência na análise arquitetural de um sistema. Perceba que a forma na qual os componentes de uma arquitetura encontram-se distribuídos, bem como os mecanismos de comunicação entre eles, são determinantes do desempenho e, portanto, da eficiência do sistema. Essa propriedade, às vezes, atua em conflito com outros requisitos do sistema, exigindo que compromissos sejam assumidos em termos de custo e desempenho de um sistema de software.

Outra importante propriedade é a integridade. Aqui, a noção de integridade refere-se à integridade em termos da unificação do projeto em níveis distintos. A arquitetura de software descreve a organização de um sistema de software num nível mais elevado, objetivando a unificação do projeto, ou seja, serve de referência para o projeto, conforme ilustrado na **Figura 4**.

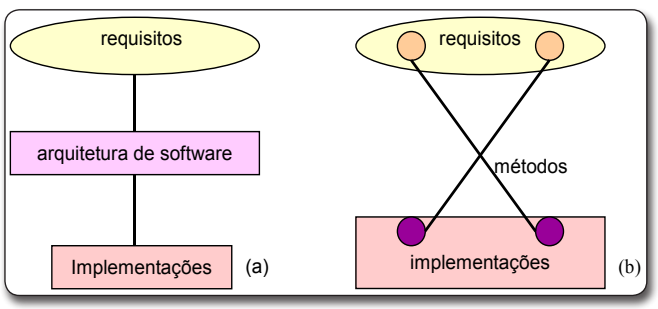


Figura 4. Arquitetura de software como referência de projeto.

Note que a arquitetura de software é central e essencial à qualidade de um sistema, associados um conjunto de atributos de qualidade. Similarmente, a integridade (arquitetural) funciona como elemento coordenador que unifica o projeto do sistema de software em diferentes níveis.

Complementando, outra importante propriedade é a flexibilidade, que diz respeito ao esforço exigido para modificar um sistema de software. Em outras palavras, é a facilidade com a qual um sistema de software pode ser estendido a fim de dar suporte a uma ou mais funcionalidade(s) adicionada(s).

Por exemplo, a criação de interfaces adicionais aumenta flexibilidade, incrementando a facilidade de fazer modificações, uma vez que as interfaces separam partes distintas de funcionalidade em componentes de software diferentes.

SAAM – Método de Análise de Arquitetura de Software

Um exemplo de método de análise de arquitetura de software, denominado SAAM (Software Architecture Analysis Method), foi inicialmente concebido com o objetivo de auxiliar arquitetos de software a compararem soluções arquiteturais, tendo sido o precursor de outros métodos de análise arquitetural. O método SAAM compreende os seguintes objetivos:

• Definir um conjunto de cenários que representem usos importantes do sistema no domínio, envolvendo os pontos de vista de todos aqueles que possuem papéis influenciadores no sistema.

• Utilizar cenários para gerar uma partição funcional do domínio, uma ordenação parcial das funções e um acoplamento dos cenários às várias funções existentes na partição.

• Usar a partição funcional e ordenação parcial, juntamente com cenários de uso, a fim de realizar a análise das arquiteturas propostas.

Assim, para cada cenário, a análise fornece uma pontuação das arquiteturas que estão sendo avaliadas, denominadas de arquiteturas candidatas. Recai, portanto, sobre o avaliador a responsabilidade de determinar os pesos dos cenários considerados, bem como a pontuação global das arquiteturas candidatas.

O método de análise arquitetural SAAM é baseado no uso de cenários. Fazendo uso de cenários, um analista pode usar uma descrição de arquitetura de software para avaliar o potencial do sistema de software prover suporte a atributos de qualidade ou requisitos não funcionais. Entretanto, é importante observar que avaliar uma arquitetura de software para determinar se ela oferece suporte a requisitos não funcionais não é tarefa fácil. Tais requisitos tendem a serem um tanto vagos.

Objetivando realizar a análise, portanto, torna-se necessário considerar o contexto no qual o sistema de software encontra-se inserido, bem como considerar as circunstâncias específicas àquele contexto. Uma forma de atingir esse objetivo é fazendo uso de cenários a fim de entender e avaliar se uma arquitetura oferece suporte ou não a um específico requisito não funcional, e sob quais circunstâncias.

SAAM compreende um conjunto de cinco passos que possuem interdependências entre si, conforme ilustra a **Figura 5** e descritas a seguir.

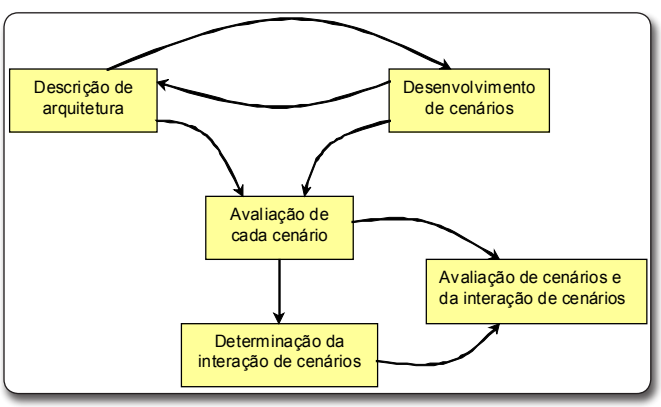


Figura 5. Etapas de SAAM.

1. Desenvolvimento de cenários – Desenvolvimento de cenários de tarefas que ilustram os tipos de atividades que o sistema de software deve prover suporte. Também, o analista deve considerar possíveis modificações que o sistema possa sofrer ao longo do tempo. Ao desenvolver esses cenários, um arquiteto deve capturar todos os usos importantes do sistema, que irão refletir os requisitos não funcionais. Além disso, esses cenários deverão apresentar interações, considerando os principais influenciadores com papéis relevantes ao sistema de software, tais como: usuário e/ou cliente, desenvolvedor, mantenedor e administrador de sistema.

2. Descrição de arquitetura – Para cada análise, deve-se possuir uma representação arquitetural do sistema, fazendo uso de uma notação arquitetural definida, descrevendo componentes e conectores utilizados na especificação da arquitetura. Isto permitirá que o pessoal envolvido na análise tenha um entendimento comum do problema. A especificação arquitetural deve informar os componentes funcionais e de dados do sistema bem como as relações existentes entre eles. Desse modo, uma representação arquitetural fará uma distinção entre componentes ativos (que efetuam transformações de dados) e componentes passivos (que armazenam dados). Adicionalmente, devemos considerar dois tipos de conexões: (a) conexões de dados (onde se tem passagem de dados entre componentes) e (b) conexões de controle (na qual um componente atua sobre um outro o habilitando a executar alguma tarefa). É importante observar que a descrição gráfica nos dá uma representação estática da arquitetura do sistema, que é complementada através de uma representação dinâmica expressa em linguagem natural que fornece uma descrição do comportamento do sistema ao longo do tempo.

3. Avaliação de cada cenário – Para cada cenário, deve-se determinar se a arquitetura candidata oferece suporte diretamente às tarefas associadas ao cenário ou se alguma modificação é necessária e, portanto, a arquitetura oferece suporte ao cenário apenas indiretamente. No primeiro caso, os cenários representam funções que a arquitetura já possui. O segundo caso compreende cenários que incluem funções além das existentes que a arquitetura de prover suporte. Neste caso, uma modificação deve ser feita na arquitetura através da adição ou alteração de um componente ou conexão. Ao final, deveria obter uma tabela sumarizando os cenários onde a arquitetura candidata provê suporte aos requisitos não funcionais associados. Nessa tabela, pode-se adotar a seguinte convenção:

- Se a arquitetura candidata suporta a(s) tarefa(s) contida(s) no cenário considerado, então se pode atribuir um escore +, e não há necessidade de fazer qualquer análise adicional.
- Se a arquitetura candidata não suporta diretamente a(s) tarefa(s) contida(s) no cenário, então se precisa continuar a avaliação a fim de determinar a extensão das modificações que devem ser realizadas. Uma forma de determinar isso é conhecendo o número de componentes e conexões que precisarão ser adicionados ou modificados. Isto requer um conhecimento profundo da arquitetura, o qual pode ser obtido junto ao desenvolvedor ou através do conjunto de especificações do sistema. Se duas arquiteturas estão sendo comparadas, então aquela que exigir menos modificação ou adição de componentes e conexões, receberá o escore +, enquanto aquela requerendo mais adições ou modificações em componentes e conexões, receberá o escore -.

4. Determinação da interação de cenários – Uma interação entre cenários ocorre quando dois ou mais cenários indiretos exigem modificações em algum componente ou conexão. A identificação da interação de cenários permite saber como a funcionalidade do sistema de software é alocada durante o projeto. Em outras palavras, a interação de cenários mostra as tarefas às quais os componentes estão associados. Na realidade, a determinação da interação de cenários avalia a extensão à qual a arquitetura provê suporte a uma separação de interesses apropriada. Por exemplo, um elevado grau de interação pode indicar que a funcionalidade de um componente específico está inadequadamente isolada. Dessa forma, para cada cenário, deve-se determinar como os componentes e conexões são afetadas pelo cenário considerado.

5. Avaliação de cenários e da interação de cenários – Esta é uma etapa subjetiva do processo de análise, envolvendo os principais influenciadores do sistema. Nesta etapa, realiza-se uma avaliação global atribuindo um escore a cada cenário e interação de cenário. Isto é baseado nas implicações que os cenários têm sobre a arquitetura do sistema. Note que, anteriormente, foram desenvolvidos cenários, fez-se o mapeamento deles na descrição arquitetural do sistema, e determinaram-se as interações de cenários existentes. Esta avaliação reflete a influência dos atributos de qualidade (ou requisitos não funcionais) associados aos cenários.

Este método de análise permite um avaliador utilizar um conjunto de métricas associadas com os cenários elaborados para que possa analisar qual arquitetura de software, dentre as arquiteturas candidatas, provê melhor suporte ao conjunto de requisitos não funcionais desejados para o sistema de software.

É importante observar que as duas primeiras etapas possuem elevado grau de dependência entre elas, como mostrado na **Figura 5**. Os tipos de cenários desenvolvidos terão influência sobre o nível de granulosidade da arquitetura. Agora, como se pode determinar um conjunto de cenários? Isto dependerá do tipo de tarefas que o sistema de software suportará. Dentro deste contexto, os cenários elaborados têm um papel de suma importância, uma vez que eles irão ajudar a compreender a descrição arquitetural do sistema.

Considere, por exemplo, uma das arquiteturas de software para interface com usuário, denominada de Serpent, ilustrada na **Figura 6**.

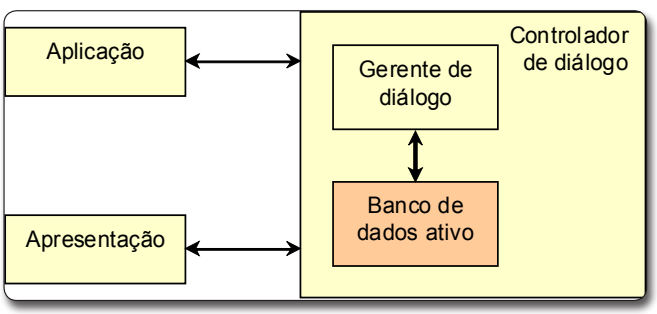


Figura 6. Arquitetura Serpent.

O componente de aplicação dá suporte à semântica computacional da aplicação. A apresentação é um componente que oferece suporte aos mecanismos de interação nos níveis lógico e físico, sendo completamente independente da semântica da aplicação. A coordenação entre estes dois primeiros componentes é feita através do controlador de diálogo. Esse terceiro componente é encarregado de mediar a comunicação da interação do usuário com a aplicação. Toda a comunicação na arquitetura Serpent é mediada por meio de restrições a dados compartilhados de um banco de dados. Esta estrutura é implementada como um banco de dados ativo. Assim, quando os valores no banco de dados são alterados, eles são automaticamente comunicados a qualquer componente registrado que possua interesse sobre o dado modificado.

Uma partição canônica para software de interface com usuário, considerada como um meta-modelo para arquiteturas de sistemas interativos, identifica que as cinco funções básicas em um software de interface compreendem:

• **Núcleo Funcional (NF)** - realiza as manipulações de dados e outras funções orientadas ao domínio. É também chamado de componente específico do domínio ou, simplesmente, aplicação.

• **Adaptador de Núcleo Funcional (ANF)** – agrega os dados específicos do domínio em estruturas de alto nível, realiza verificações semânticas e dispara tarefas de diálogo inicializadas no domínio. Ele é um mediador entre o Diálogo e o Núcleo Funcional.

• **Diálogo (D)** – faz a mediação entre partes específicas do domínio e específicas da apresentação de uma interface com usuário, realizando o mapeamento de dados quando necessário. Assegura consistência e o sequenciamento de tarefas.

• **Componente de Interação Lógica (IL)** – fornece um conjunto de objetos independentes de toolkit para o componente de Diálogo.

• **Componente de Interação Física (IF)** – implementa a interação física entre usuário e computador. Lida com dispositivos de entrada e saída e é, geralmente, conhecido como componente de toolkit de interação.

Agora, se tentarmos fazer o mapeamento dessas cinco funções para sistemas interativos (na arquitetura de software de interface de usuário de Serpent), obteremos uma arquitetura conforme ilustrada na **Figura 7**.

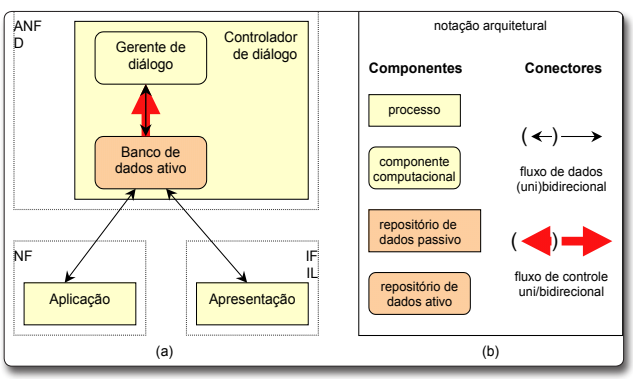


Figura 7. Arquitetura Serpent baseada no meta-modelo de arquitetura.

A arquitetura Serpent foi redesenhada fazendo uso da notação arquitetural apresentada na **Figura 7b**. As funções básicas de um software de interface com usuário foram mapeadas na estrutura inicial da arquitetura Serpent (vide **Figura 6**), resultando a arquitetura ilustrada na **Figura 7a**. Algumas observações podem ser feitas:

• Não há separação arquitetural entre as funções IL e IF no processo de apresentação;

• Também, não existe separação arquitetural entre as funções de ANF e D no processo do controlador de diálogo;

• Exceto o gerenciador de diálogo, todos os componentes de Serpent são monolíticos. Em outras palavras, eles não oferecem suporte arquitetural para subdivisão de funcionalidade dentro de um componente.

Considerando a arquitetura Serpent, vamos supor que se deseja efetuar uma modificação como, por exemplo, adicionar uma nova opção de menu. Vamos avaliar o grau no qual essa arquitetura suporta essa modificação. Perceba que a arquitetura Serpent separou o controlador de diálogo e, portanto, é fácil verificar que a modificação proposta (adicionar uma opção de menu) deverá ser feita nesse processo. O controlador de diálogo possui dois componentes: o gerente de diálogo, responsável por comandar o comportamento do diálogo, e o banco de dados ativo, o qual mantém o estado do diálogo. Esse cenário, no qual se deseja fazer uma modificação (adição de nova opção de menu), está associado à mudança no comportamento do diálogo e, portanto, está isolado no gerente de diálogo. Adicionalmente, o gerente de diálogo é subdividido em componentes menores associados a partes específicas do comportamento do diálogo como, por exemplo, um menu. Desse modo, pode-se concluir que a arquitetura Serpent fornece suporte apropriado a este tipo de cenário envolvendo modificação no aspecto de um sistema interativo.

Uso de Atributos de Qualidade na Análise Arquitetural

Uma das razões de concentrar atenção sobre a arquitetura é para identificar e analisar as decisões iniciais de projeto. Transformar estas decisões de projeto em uma estrutura que ofereça suporte a um raciocínio que permita antecipar tendências a nível arquitetural é essencial para obter os benefícios do foco na arquitetura.

Embora possamos apontar indicativos de qualidade, seria praticamente impossível fazer previsão de qualidade num estágio inicial de projeto. Note que, por exemplo, o método de análise arquitetural SAAM não objetiva precisamente prever o comportamento de atributos de qualidade. A meta desse método de análise dá-se em entender o papel e as conseqüências de decisões arquiteturais em relação aos atributos de qualidade de algum sistema que esteja sendo analisado.

Dessa forma, o método SAAM apresentado serve de ferramenta ao arquiteto ou engenheiro de software para prever suporte a atributos de qualidade. Note que isto é baseado em decisões de projeto que ocorrem no nível arquitetural.

Sabemos que a arquitetura é um elemento essencial para a obtenção de sucesso, em termos tecnológicos e de mercado de

qualquer empresa. Geralmente, as metas de qualidade e conjunto de funcionalidades constituem os principais motivadores de um sistema. Considere, por exemplo, um fabricante de centrais telefônicas que esteja desenvolvendo um projeto de uma nova central. Assim, pode ser especificado que a central deve ser capaz de rotear ligações telefônicas, prover suporte a diversos tipos de tons, redirecionar ligações, gerar contas telefônicas de usuários com discriminação de ligações efetuadas, dentre outras funcionalidades. Entretanto, para que o sistema alcance sucesso na prestação de serviços para seus assinantes, ele deve também operar com elevado nível de desempenho, facilitar quaisquer modificações (em termos de adição ou alteração de características), possuir bom nível de disponibilidade e ter baixo custo.

Como essas metas de qualidade podem ser alcançadas?

A arquitetura do sistema é um meio pelo qual podemos determinar se essas metas de qualidade podem ser realizáveis ou não. SAAM e outros métodos constituem respostas a essa questão. Perceba que essa necessidade de prever o comportamento de atributos de qualidade num sistema requer conhecimento de estilos arquiteturais prováveis de serem empregados no sistema. Um estilo arquitetural inclui:

- Descrição de tipos de componentes e sua topologia;
- Descrição de padrão de interação (no nível de dados e controle) entre os componentes;
- Descrição informal das vantagens e desvantagens na adoção do estilo.

Estilos arquiteturais permitem ao arquiteto de software distinguir classes de projeto através de evidências existentes de como cada classe tem sido utilizada juntamente com o raciocínio qualitativo para explicar porque certas classes têm determinadas propriedades. Assim, por exemplo, pode-se adotar o estilo arquitetural de pipes e filtros quando se deseja obter reuso e não há restrição quanto ao desempenho.

Agora, pode ser feito um mapeamento de um estilo arquitetural em um conjunto de modelos de atributos de qualidade. Estes modelos de atributos de qualidade ajudam a prever o comportamento desses atributos. Esta noção de mapeamento de decisões e propriedades arquiteturais em modelos de atributos de qualidade é ilustrada na **Figura 8**.

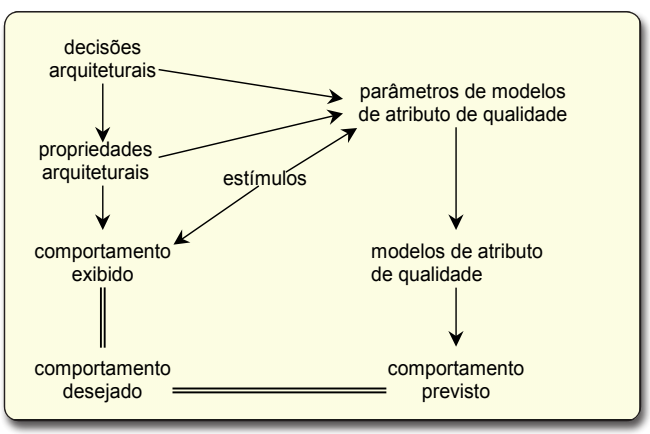


Figura 8. Mapeamento de modelos arquiteturais em modelos de atributos.

É importante observar que decisões arquiteturais influenciam direta e indiretamente no comportamento apresentado pela arquitetura. Como poderíamos caracterizar esse comportamento? Considere, por exemplo, a alocação de funcionalidade a um conjunto de processos. Isto requer que o arquiteto ou engenheiro de software tome decisões arquiteturais. Disto resulta que tais processos precisam ser alocados a processadores, requerendo ainda mais decisões arquiteturais. Note que, associado a eles, temos um conjunto de tempos de execução de processo, os quais constituem propriedades arquiteturais. Dessa forma, as propriedades arquiteturais, em conjunto com estímulos (tais como taxa de chegada de mensagens), nos leva a um comportamento de desempenho exibido pelo sistema.

É importante observar que, para avaliar uma arquitetura em função de requisitos não funcionais, é necessário expressar esses requisitos ou atributos de qualidade em termos mensuráveis, denominado de medidas de atributos de qualidade. Tais medidas dependem de propriedades da arquitetura ou parâmetros de atributos de qualidade e dos estímulos. Considere como atributo de qualidade o desempenho. Esse requisito pode ser expresso em termos das seguintes medidas de atributo de qualidade:

- Latência - tempo da ocorrência de um evento até a resposta àquele evento, dado em unidades de tempo;
- Throughput – taxa na qual o sistema pode responder a eventos, dada em transações por unidades de tempo.

Os estímulos podem ser vistos como mudanças de estado ao qual a arquitetura deve responder. Se, novamente, considerarmos o desempenho, temos o padrão de chegada de um evento que pode ser periódico, esporádico ou probabilístico. Se um estímulo ocorre, então o sistema deve responder fazendo uso de seus recursos para realizar alguma computação.

Note que analisar uma arquitetura é uma atividade investigativa na qual o avaliador busca saber quão bem uma arquitetura tem sido projetada em relação aos requisitos não funcionais desejados para o sistema. Tais requisitos estão intrinsecamente associados à qualidade do sistema. Um arquiteto de software, ao tomar decisões de projeto, busca maximizar os benefícios obtidos com o sistema e, ao mesmo tempo, minimizar os custos de implementação do projeto. Note que as metas de negócios de uma empresa também têm influência sobre decisões arquiteturais tomadas pelo projetista. Essas decisões têm implicações técnicas e econômicas.

As implicações técnicas compreendem as características do sistema de software e os atributos de qualidade desejados. As implicações econômicas referem-se ao custo de implementar o sistema. A **Figura 9** ilustra o contexto das decisões arquiteturais e implicações associadas.

Considerando a **Figura 9**, tem-se que a determinação dos custos e benefícios associados às decisões arquiteturais pode ser obtida através de:

- Avaliação da importância dos requisitos não funcionais;
- Quantificação dos escores de benefícios dos requisitos não funcionais;
- Quantificação dos custos dos requisitos não funcionais;
- Escolha de cenários e estratégias de projeto arquitetural.

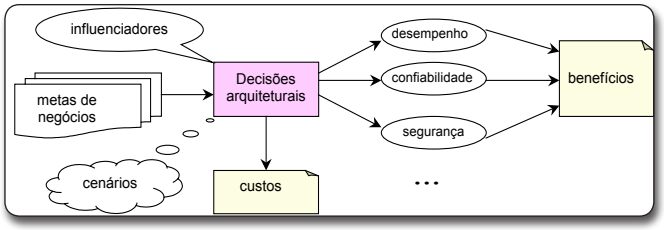


Figura 9. Contexto de decisões arquiteturais e implicações associadas.

É importante observar que, para quantificar e compreender de uma forma consistente o impacto que as estratégias de projeto arquitetural têm sobre os atributos de qualidade, os influenciadores necessitam de uma representação mais concreta dos atributos de qualidade. Para este propósito, cenários devem ser utilizados. Durante a análise arquitetural, a elicitação e análise de cenários permitem a caracterização de atributos de qualidade. Esses cenários especificam características envolvendo estímulos e respostas concretizando os atributos de qualidade. Isto permite que os influenciadores possam avaliar os efeitos desses atributos. Esta avaliação é centrada na premissa de maximizar os benefícios obtidos com a arquitetura de um sistema e, ao mesmo tempo, minimizar os custos associados com sua implementação.

Conclusão

Neste artigo, um conjunto de propriedades de arquitetura de software consideradas intrínsecas à estrutura do sistema foi apresentado. Estas propriedades, juntamente com os requisitos arquiteturais, envolvendo requisitos não funcionais e atributos de projeto, servem de base para a análise arquitetural.

O método de análise arquitetural SAAM foi apresentado, ilustrando-se os passos de como fazer a análise. Adicionalmente, foi discutido o papel dos requisitos não funcionais, ou atributos de qualidade, como mecanismo para auxiliar nas decisões arquiteturais. Cabe destacar que a análise arquitetural é realizada sob a óptica de minimizar custos e agregar benefícios, isto é, prover suporte aos requisitos não funcionais.

Links

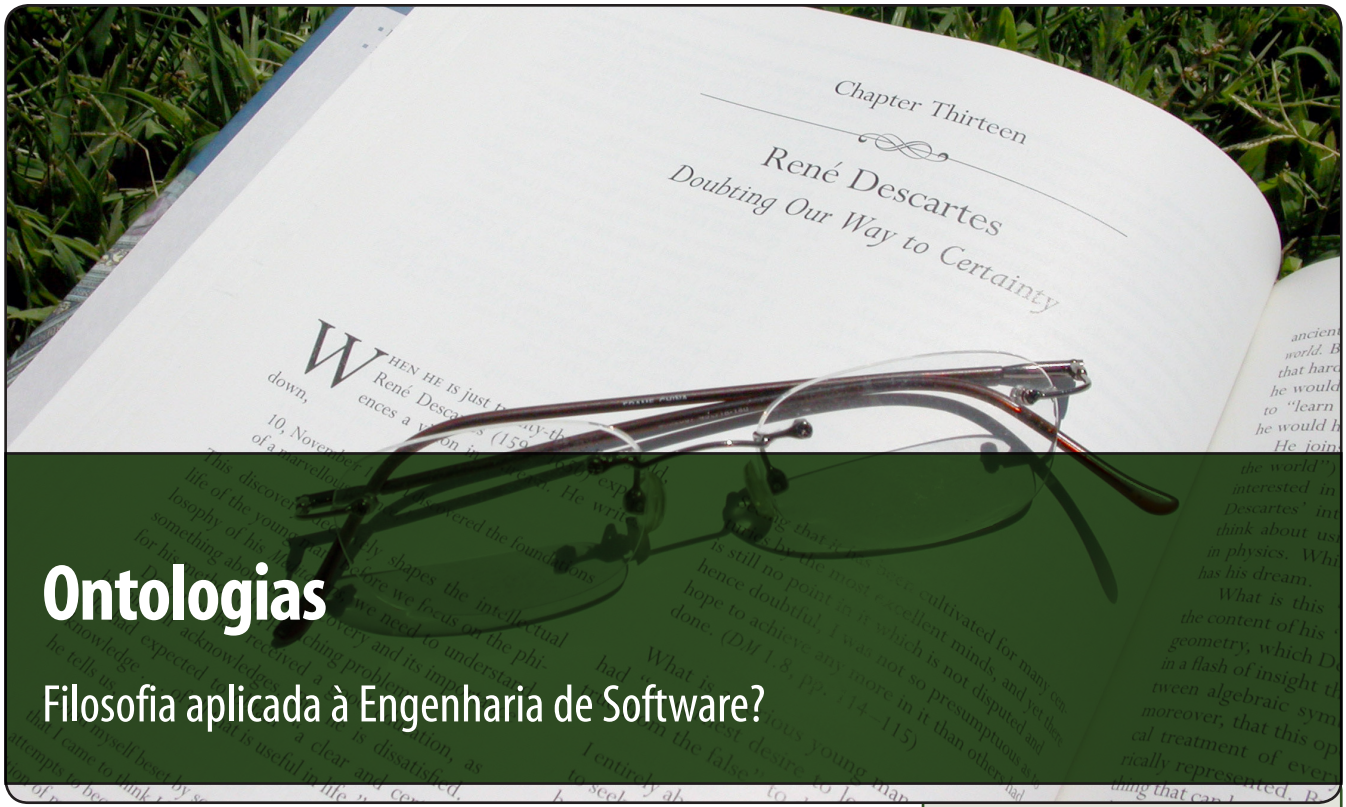
Scenario-Based Analysis of Software Architecture
http://www.sei.cmu.edu/architecture/scenario_paper/

SAAM: A Method for Analyzing the Properties of Software Architectures
http://www.sei.cmu.edu/publications/articles/saam-metho-propert-sas.html

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

The Serpent Runtime Architecture and Dialogue Model
http://www.sei.cmu.edu/pub/documents/88.reports/pdf/tr06.88.pdf

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



Ontologias

Filosofia aplicada à Engenharia de Software?

De que se trata o artigo?
Este artigo apresenta uma introdução ao tema Ontologias, destacando seus principais conceitos e aplicações na Engenharia de Software.

Para que serve?
Fornecer conhecimento essencial sobre ontologias para organizações, pesquisadores, estudantes e profissionais de software.

Em que situação o tema é útil?
Definição de vocabulário comum, claro e sem ambiguidades, seja visando a representação, compartilhamento e reuso de conhecimento, seja a interoperabilidade semântica entre sistemas.

Monalessa Perini Barcellos
monalessa@inf.ufes.br
É Doutoranda em Engenharia de Sistemas e Computação (COPPE/UFRJ), Mestre em Engenharia de Sistemas e Computação (COPPE/UFRJ), Bacharel em Ciência da Computação (UFES). Professora do Departamento de Informática, área Engenharia de Software, da Universidade Federal do Espírito Santo (UFES). Atuante, desde 1999, em projetos, consultorias, treinamentos e pesquisas da área de Engenharia de Software.

Na última década, a utilização do termo ontologia tem sido relativamente comum em Ciência da Computação, especialmente em Engenharia de Software. Esse termo, com origem na Filosofia, inicialmente soou de forma 'cacofônica' aos ouvidos dos estereotipados como 'exatos' profissionais da computação. No entanto, aos poucos foi possível perceber que, ao contrário do que muitos pensavam, aspectos filosóficos estão presentes em praticamente todas as atividades relacionadas ao desenvolvimento de software. Essa percepção abriu espaço para que os princípios das ontologias fossem eficientemente aplicados na Engenharia de Software.

Filosoficamente falando, uma ontologia pode ser definida como um sistema particular de categorização para uma certa visão do mundo, independente da linguagem. Nossa... parece complicado? Mas, de fato, não é.

Veja bem, trazendo esse conceito para a computação (Inteligência Artificial, Engenharia de Software e Web Semântica), uma ontologia pode ser definida como um artefato de engenharia. Então, você deve estar pensando: "- Agora sim! Artefato e engenharia eu sei bem o que são!".

Continuando: esse artefato de engenharia é formado por um vocabulário usado para descrever certa realidade e por um conjunto de conjecturas explícitas relacionadas ao significado

pretendido das palavras do vocabulário. Complicou de novo? Nada disso... Vamos continuar.

Esse conjunto de conjecturas tem, geralmente, a forma da teoria de lógica de primeira ordem (opa! Essa é familiar não?), onde as palavras do vocabulário aparecem como nomes de predicados unários ou binários, respectivamente chamados conceitos e relações. No caso mais simples, uma ontologia descreve uma hierarquia de conceitos relacionados. Em casos mais sofisticados, axiomas apropriados devem ser adicionados a fim de expressar outros relacionamentos entre conceitos e restringir a interpretação pretendida.

Pronto! Se você analisar o parágrafo anterior vai concluir que, na prática, para a Engenharia de Software, uma ontologia pode ser vista como um modelo que representa conceitos e relações, bem como suas restrições. Isso não lhe parece familiar? Não remete a um modelo conceitual produzido na fase de Análise no desenvolvimento de sistemas? Não lembra um diagrama de classes UML, por exemplo?

Essa similaridade é tão grande que é muito comum a utilização de perfis UML na definição das ontologias. Por exemplo, uma ontologia que descreve o vocabulário relacionado à oferta e matrícula em disciplinas em uma instituição acadêmica poderia incluir o fragmento ilustrado na **Figura 1**.

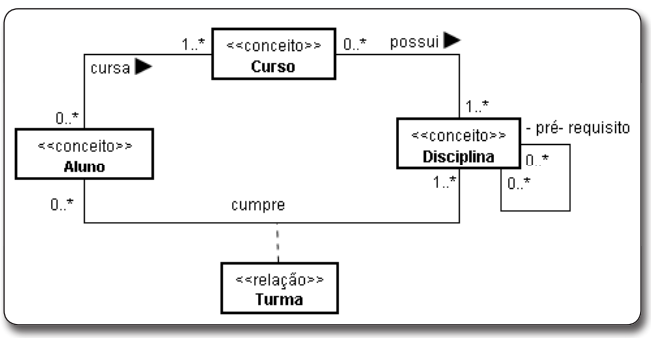


Figura 1. Fragmento hipotético de uma ontologia que inclui oferta e matrícula em disciplinas.

Ao diagrama seriam acrescentados axiomas para formalizar algumas restrições não capturadas explicitamente como, por exemplo, um axioma para restringir que um aluno só deve cumprir disciplinas do seu curso (lembrando que essa é apenas uma situação hipotética para efeito de exemplo). Esse axioma poderia ser assim representado:

$$(\forall\ a \in \text{Aluno},\ d \in \text{Disciplina},\ c \in \text{Curso})\ \text{cumpre}(a,d) \rightarrow\ (\text{possui}\ (c,\ d) \wedge \text{cursa}(a,\ d))$$

Indiscutivelmente, o fragmento apresentado na **Figura 1** também poderia ser utilizado no modelo de classes de um sistema de apoio à realização da oferta e matrícula, bem como o axioma poderia ser transformado em uma regra de negócio. Então, uma ontologia é um modelo de classes? Não. Essa é apenas uma analogia útil ao entendimento de ontologias à Engenharia de Software. De uma maneira geral, ambos são modelos conceituais.

É importante perceber que, uma mesma ontologia pode, por exemplo, apoiar a definição de vários modelos de classes diferentes, uma vez que sua principal preocupação é com o conteúdo e não com a forma. Ou seja, apesar de comumente uma ontologia ser definida com um 'vocabulário que representa elementos conceituais e suas relações', de fato ela não é o vocabulário propriamente dito, e sim o que ele representa, uma vez que traduzir esse vocabulário para uma outra linguagem não muda a ontologia. A ontologia define os conceitos, relações e interpretações pertinentes a uma parte do mundo que ela representa. A forma como seu conteúdo, ou seja, seu vocabulário, é utilizado, pode variar de acordo com o contexto de aplicação.

Isso quer dizer que, como dito anteriormente, uma mesma ontologia pode ser utilizada como fonte de informação para a construção de modelos diferentes de sistemas diferentes. Mas, se esses modelos forem corretamente elaborados levando-se em consideração o vocabulário representado pela ontologia, eles serão semanticamente equivalentes e a comunicação entre os sistemas será possível, ou seja, eles poderão ser mais facilmente integrados.

Para conhecer um pouco mais sobre ontologias, nas seções seguintes são apresentados seu histórico, destacando-se sua aplicação em Engenharia de Software, seus principais tipos e alguns dos benefícios de sua utilização em Engenharia de Software.

Histórico

Ontologias têm sido reconhecidas como um instrumento conceitual bastante útil na Ciência da Computação desde a década de 60, tendo havido nos últimos oito anos uma explosão de trabalhos relacionados a ontologias em diferentes áreas da Ciência da Computação.

A origem das ontologias se deu na Filosofia, sendo Aristóteles o primeiro filósofo ocidental a criar uma definição rigorosa para ontologia, descrevendo-a como a ciência do ser enquanto ser. Segundo Aristóteles, o objetivo de uma ontologia é estudar as características mais gerais da realidade e dos objetos reais, ou seja, estudar as características de todos os tipos de seres.

Somente no século XVII o termo ontologia foi formalmente utilizado, sendo citado paralelamente pelos filósofos Rudolf Göckele e Jacob Lorhard, e sendo popularizado em 1730 com a publicação do livro Philosophia Prima Sive Ontologia, de Christian Wolff.

No século XX, o filósofo alemão Edmund Husserl, fazendo analogia à Lógica Formal, criou o termo Ontologia Formal. A Lógica Formal lida com estruturas lógicas formais (por exemplo: verdade, validade e consistência) independente de suas veracidades. A Ontologia Formal, por sua vez, lida com estruturas ontológicas formais (por exemplo: teoria das partes, teoria do todo, tipos e instanciações, identidade, dependência e unidade). Em outras palavras, Ontologia Formal lida com aspectos formais de objetos, desprezando suas naturezas particulares. A partir de então, uma série de teorias sobre ontologias foram desenvolvidas ao longo do século XX.

A utilização do termo ontologia em Computação se deu pela primeira vez em 1967 por G. Mealy. Na década de 90, comunidades de Inteligência Artificial mostraram intenso interesse na utilização de ontologias para criar representações formais de conhecimento sobre domínios específicos. Nesse momento, percebeu-se que ontologias poderiam ser utilizadas em Computação para a obtenção de uma uniformidade de vocabulário (conceitos e relações entre conceitos), de forma a evitar ambiguidades e inconsistências, propiciando, entre outros, o compartilhamento e a reutilização de conhecimento.

Somente nos primeiros anos do século XXI houve realmente uma intensificação de trabalhos relacionados às ontologias em áreas como Sistemas de Informação, Bancos de Dados, Engenharia de Software e Inteligência Artificial, tendo sido impulsionados, sobretudo, pelo crescente interesse em Web Semântica e no papel desempenhado pelas ontologias nesse contexto.

Em Engenharia de Software, a primeira proposta conhecida para aplicação de ontologias foi apresentada por T. Gruber em 1993. Nessa área, ontologias têm sido tipicamente utilizadas para reduzir ambiguidades conceituais, tornar transparentes as estruturas de conhecimento, apoiar o compartilhamento de conhecimento e apoiar interoperabilidade entre sistemas, tendo destaque sua utilização no ramo da Engenharia de Domínio.

O propósito Engenharia de Domínio é identificar, modelar, construir, catalogar e disseminar um conjunto de artefatos de software que possam ser utilizados e compartilhados por engenheiros de software durante o desenvolvimento ou manutenção de software em um domínio particular de aplicação. Um dos principais produtos da Engenharia de Domínio é o Modelo de Domínio, que descreve os objetos e suas relações em um determinado domínio de discussão. Um modelo de domínio deve ser uma fonte unificada de referência quando surgem ambiguidades na análise de problemas ou durante a implementação de componentes reutilizáveis. Além disso, deve ser um repositório de conhecimento e uma especificação para o desenvolvedor de componentes reutilizáveis.

Analisando-se o conceito de Modelo de Domínio é possível identificar sua relação com ontologias, uma vez que ambos buscam fornecer um entendimento uniforme e não ambíguo de objetos e suas relações, provendo uma conceituação acerca de um determinado domínio. Ou seja, na prática, uma ontologia de domínio pode ser utilizada como um modelo de domínio.

Tipos de Ontologias

Várias são as classificações de ontologias encontradas na literatura. Aqui é apresentada a classificação definida por Nicola Guarino (1998).

Nicola Guarino é um dos pesquisadores de ontologias que, ao analisarem a aplicação das ontologias em Computação, ressaltam que além dos aspetos técnicos inerentes a essa área, há também aspectos filosóficos, cognitivos e linguísticos envolvidos. Esses aspectos devem ser levados em consideração no momento de construir as ontologias, caso contrário sua utilização será limitada.

Sob o ponto de vista da Engenharia de Software, é aí que reside a grande diferença entre um modelo de análise e uma ontologia. Ou seja, existem aspectos que são considerados em uma ontologia e que, normalmente são ignorados em modelos baseados estritamente em uma visão mais técnica (que busca uma solução específica), o que pode, em uma situação futura, representar problemas.

A **Figura 2** mostra os quatro tipos de ontologias e seus relacionamentos. Suas definições são descritas em seguida.

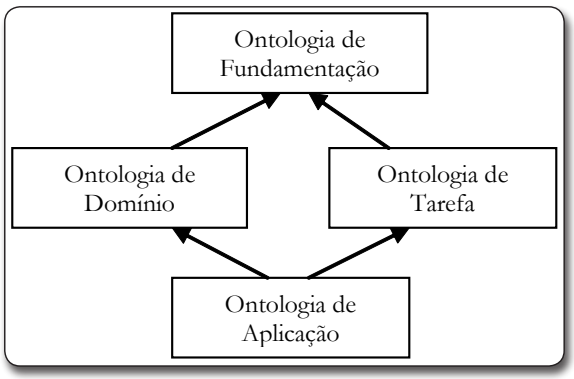


Figura 2. Relacionamentos entre os tipos de ontologias segundo Guarino.

Conforme ilustrado na **Figura 2**, ontologias de Fundamentação devem ser utilizadas como base para a construção de ontologias de Domínio e de Tarefa e essas, por sua vez, devem ser utilizadas como base para construção de ontologias que tratam de aplicações específicas.

Ontologia de Fundamentação

Uma ontologia de fundamentação (também chamada ontologia genérica ou ontologia de nível superior) é, teoricamente, uma conceituação abstrata sobre elementos genéricos demais para fazer parte de um domínio específico, como espaço, tempo, matéria, objeto, evento e ação. Essas ontologias podem ser utilizadas como meta-modelo para ontologias de domínio e tarefa, visto que cada conceito ou propriedade de uma dessas ontologias pode ser mapeado em um conceito de uma ontologia de fundamentação. Na **Figura 3** é apresentado um fragmento hipotético de uma ontologia de fundamentação.

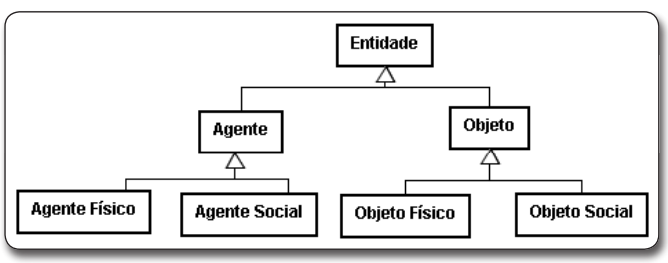


Figura 3. Fragmento hipotético de uma ontologia de fundamentação.

Considerando os conceitos da **Figura 3**, uma ontologia de fundamentação poderia definir que um Agente é uma entidade capaz de realizar ações, podendo ser Físico (por exemplo, uma pessoa) ou Social (por exemplo, uma organização); e que um

Objeto é uma entidade que não é capaz de realizar ações, mas pode participar delas, podendo ser Físico (por exemplo, um livro) ou Social (por exemplo, uma linguagem). A cada um desses conceitos, a ontologia de fundamentação proveria um conjunto de características e regras próprias.

Quando uma ontologia de fundamentação é tomada como base para a construção de ontologias de domínio ou de tarefa, todo conceito dessas ontologias pode ser classificado em um conceito da ontologia de fundamentação, herdando assim suas características e regras.

No exemplo ilustrado na **Figura 1**, o conceito Aluno poderia ser classificado como Agente segundo a definição hipotética da **Figura 3** e, como tal, teria todas as suas características filosóficas, que dificilmente seriam capturadas se fosse considerada apenas a visão técnica necessária para elaborar um modelo de classes de um software para solucionar a oferta e matrícula em disciplinas de uma instituição específica.

Ontologia de Domínio

Uma ontologia de domínio é um artefato de Engenharia de Software que expressa conceituações de domínios particulares (por exemplo: medicina, turismo e petróleo). Uma ontologia de domínio deve especificar formalmente as relações, regras e exceções de todos os elementos presentes na conceituação da parte da realidade por ela mapeada.

Conforme mencionado anteriormente, ontologias de domínio constituem a maior parte das aplicações das ontologias na Engenharia de Software. Exemplos de ontologias definidas nesse contexto são: ontologias de processo de software, ontologias relacionadas à manutenção de software, ontologias relacionadas à qualidade de software, ontologias para requisitos de software e ontologias relacionadas ao desenvolvimento de software baseado em componentes.

O exemplo ilustrado na **Figura 1** pode ser considerado um substrato hipotético de uma ontologia para o domínio Ensino Universitário.

Ontologia de Tarefa

Uma ontologia de tarefa é similar a uma ontologia de domínio, porém, ao invés de mapear os conceitos de um domínio particular, mapeia conceitos de uma tarefa ou atividade específicos (por exemplo: diagnóstico, venda e matrícula).

Ontologia de Aplicação

Uma ontologia de aplicação descreve conceitos que são dependentes de um domínio e de uma tarefa particulares e, assim, combina especializações de conceitos presentes nas ontologias de domínio e de tarefa.

Benefícios da Utilização das Ontologias em Engenharia de Software

Conforme dito antes, ontologias podem ser utilizadas para descrever o vocabulário comum de um determinado domínio. Na Engenharia de Software, a definição de vocabulários comuns, cobrindo todos os aspectos do mundo real (ou a maior parte possível) é importante, pois propicia, entre outros, os seguintes benefícios:

• Apóia a captura, representação, pesquisa, armazenamento e reutilização do conhecimento referente à Engenharia de Software: sendo a Engenharia de Software uma área composta de muitas atividades cognitivas, a representação adequada do conhecimento a ela relacionado, descrevendo um vocabulário consistente, completo e não ambíguo, facilita a recuperação e (re)utilização desse conhecimento.

• Apóia a padronização: em Engenharia de Software a padronização desempenha um papel muito importante. A utilização de padrões auxilia as organizações a realizarem suas atividades apoiadas por métodos que reforçam a Engenharia de Software como uma engenharia propriamente dita. Uma vez que o conhecimento de Engenharia de Software é adequadamente representado e reutilizado, é possível deixar práticas ad-hoc e adotar práticas padrão e repetíveis.

• Apóia reuso: além de apoiar a reutilização do conhecimento, ontologias apóiam o reuso de outros artefatos produzidos em todas as etapas do processo de desenvolvimento de software. Isso é possível, pois uma vez que todo o processo é guiado por um vocabulário comum, os artefatos produzidos são consistentes entre si, podendo ser melhor reutilizados ao longo desse processo.

Apóia a interoperabilidade: a utilização da mesma semântica permite que sistemas sejam integrados, mesmo que suas formas de representar a semântica sejam diferentes.

• Apóia a componentização: uma vez que apóiam o reuso e a interoperabilidade, como consequência, as ontologias apóiam a componentização.

Facilita a comunicação: a utilização de um vocabulário comum facilita a comunicação entre os envolvidos no desenvolvimento de software, bem como entre os sistemas propriamente ditos.

• Apóia a captura, representação, pesquisa, armazenamento e reutilização do conhecimento referente aos diversos domínios dos produtos e serviços de software: da mesma maneira que ontologias relacionadas à Engenharia de Software são úteis aos engenheiros de software e desenvolvedores, ontologias dos domínios específicos dos produtos e serviços também são importantes. Por exemplo, o desenvolvimento de um software para uma empresa do segmento de petróleo pode ser mais eficiente se apoiado por uma ontologia que descreve o vocabulário do domínio petrolífero.

Conclusão

Apesar de ter uma abordagem predominantemente filosófica, as ontologias têm sido eficientemente utilizadas em Engenharia de Software, principalmente quando se busca reutilização e interoperabilidade.

Muitas vezes, a integração ou reutilização de sistemas e/ou componentes não é possível devido a problemas relacionados à semântica (e não à técnica) de alguns dos conceitos envolvidos. Além disso, limitações em relação à semântica do mundo real também dificultam a integração.

Algumas das características diretamente responsáveis pela reutilização e interoperabilidade semântica entre sistemas são a fidedignidade à realidade e a clareza conceitual dos modelos conceituais utilizados.

Em resumo, quanto mais completos e claros forem os modelos conceituais utilizados, maiores serão as possibilidades de reutilização e interoperabilidade em sistemas. Sendo as ontologias de domínio modelos conceituais, quando definidas adequadamente, servem como arcabouços de referência para o alcance da reutilização e interoperabilidade semântica.

Tendo em vista os benefícios da aplicação de princípios filosóficos aos modelos conceituais da Computação, mais uma vez temos que dar o braço a torcer e concordar que nossa ciência exata, na verdade, não é tão exata assim. ●

Dê seu feedback sobre esta edição!

A Engenharia de Software Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!
Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/esmag/feedback

Referências

FALBO, R. A., GUIZZARDI, G., DUARTE, K. C., 2002, "An Ontological Approach to Domain Engineering", In Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, Ischia, Italy, pp. 351 – 358.

GRUBER, T. R., 1993, "A Translation Approach to Portable Ontologies", Knowledge Acquisition, Volume 5 (2), pp. 199-220.

GUARINO, N., 1998, "Formal Ontology and Information Systems", In Proceedings of International Conference in Formal Ontology and Information Systems - FOIS'98, Trento, Italy, pp. 3-15.

GUIZZARDI, G., 2005, "Ontological Foundations for Structural Conceptual Models", Universal Press, The Netherlands, ISBN 90-75176-81-3.

GUIZZARDI, G., FALBO, R. A., GUIZZARD, R. S. S., 2008, "Grounding Software Domain Ontologies in the Unified Foundational Ontology (UFO): The case of the ODE Software Process Ontology", In Proceedings of the XI Iberoamerican Workshop on Requirements Engineering and Software Environments, Recife – Brasil.

MEALY, G., 1967, "Another Look at Data", In American Federation of Information Processing Societies, Fall Joint Computer Conference, Volume 31.

PRESSMAN, R. S., 2004, "Software Engineering: A Practitioner's Approach", McGraw-Hill Science/Engineering/Math, 6th edition.