engenharia de software magazine

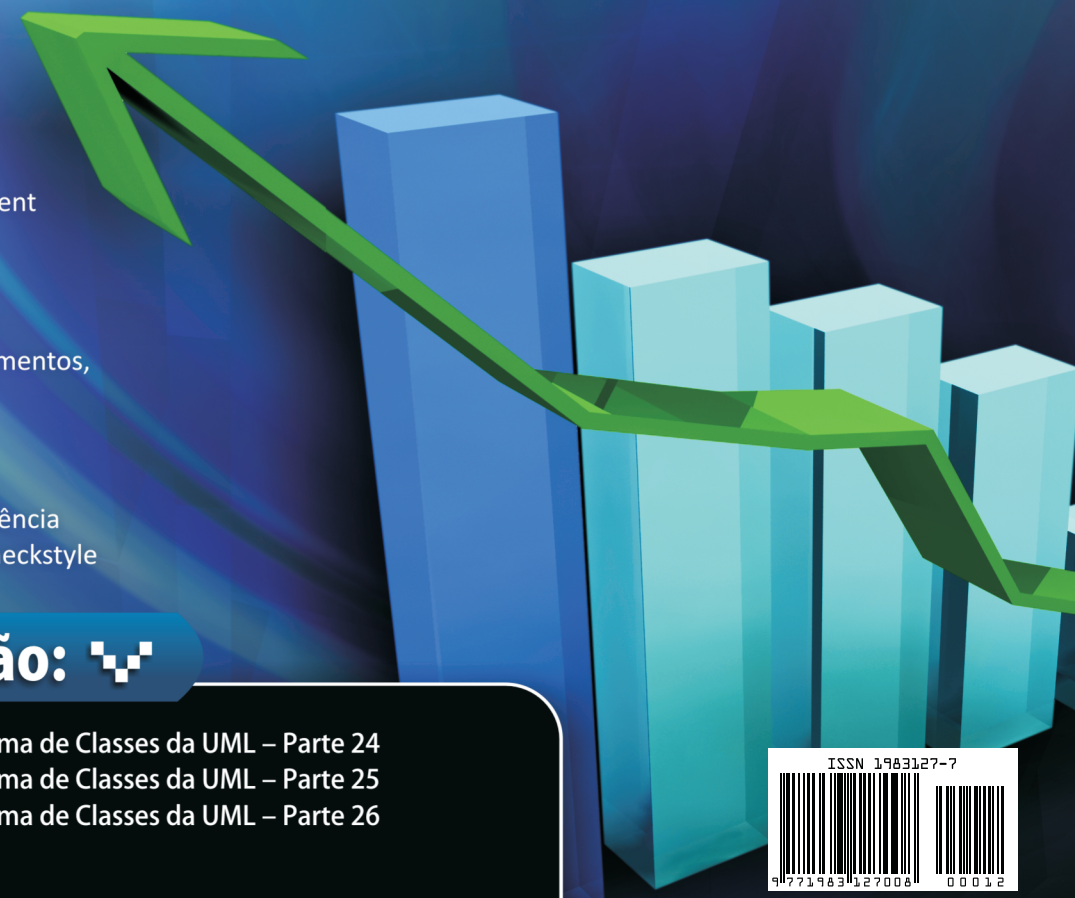
DevMedia group Ano I - Edição 12

Projeto
Aprenda a elaborar diagrama de classes – Parte I

Processo
Saiba como utilizar controle estatístico de processos na melhoria de processos de software

METODOLOGIAS ÁGEIS

Veja como acompanhar seus projetos através do uso de métricas



Planejamento
Introdução ao Project Management Body of Knowledge (PMBoK)

Planejamento
Estimativas de Software - Fundamentos, Técnicas e Modelos – Parte 2

Desenvolvimento
Automação do processo de aderência a padrões de codificação com Checkstyle

Aulas desta edição:

- Introdução à Construção de Diagrama de Classes da UML – Parte 24
- Introdução à Construção de Diagrama de Classes da UML – Parte 25
- Introdução à Construção de Diagrama de Classes da UML – Parte 26

ISSN 1983127-7

engenharia de software magazine

Ano 1 - 12ª Edição - 2009 **Impresso no Brasil**

EDITORIAL

"Encontrar maneiras eficazes de avaliar o processo e a equipe de desenvolvimento não é uma tarefa simples. Isso leva a uma proliferação de medidas baseadas na premissa de que se cada parte do processo for otimizada, os resultados do processo como um todo serão otimizados também. No entanto, essa premissa nem sempre é verdadeira. Ao tentar micro-otimizar partes de um sistema por meio de diversas métricas, o verdadeiro objetivo se perde em meio a tantos substitutos e a equipe perde sua capacidade de tomar decisões de balanceamento (trade-off). Além disso, a preocupação com as medidas erradas pode gerar incentivos errados, levando a conseqüências indesejáveis.

"Em metodologias ágeis, a escolha das melhores formas de medição é uma tarefa do Time Completo e o tracker possui um papel especial. O tracker é responsável por prover informações para a equipe sobre o progresso do time, utilizando as métricas apropriadas para destacar os pontos de melhoria e atualizando regularmente essas informações nos gráficos e pôsteres na área de Trabalho Informativa, que Cockburn chama de radiadores de informação."

Neste contexto, a Engenharia de Software Magazine destaca uma matéria que apresenta os conceitos relacionados às métricas para auxiliar o tracker de uma equipe ágil, discutindo definições, classificações, diferentes abordagens para escolha das melhores métricas e, por fim, apresentando alguns exemplos que podem ser utilizados no acompanhamento de uma equipe ágil.

Além desta matéria, esta edição traz mais cinco artigos:

- UML – Diagrama de Classes: Encontrando classes e desenhando seu diagrama – Parte I;
- Utilização do Controle Estatístico de Processos na Melhoria de Processos de Software;
- Estimativas de Software - Fundamentos, Técnicas e Modelos – Parte 2;
- Introdução ao Project Management Body of Knowledge (PMBoK);
- Ferramentas de Qualidade de Código.

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 - Editor
(*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 Colaborador das revistas Engenharia de Software Magazine, Java Magazine e SQL Magazine. 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 ao lado:

Tipo: Projeto
Título: Introdução à Construção de Diagrama de Classes da UML – Parte 24
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula dá continuidade ao estudo de caso de um sistema de gestão de festas. Neste estudo de caso, partiremos dos requisitos funcionais do software. A partir deles, será elaborado o diagrama de casos de uso do sistema. Com o diagrama de casos de uso elaborado, serão especificados três casos de uso. Por último, será elaborado o diagrama de classes com base nos requisitos funcionais e casos de uso especificados. Nesta aula, o foco é a atualização do diagrama de classes na Visual Paradigm através da definição das operações identificadas nas classes existentes.

Tipo: Projeto
Título: Introdução à Construção de Diagrama de Classes da UML – Parte 25
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula dá continuidade ao estudo de caso de um sistema de gestão de festas. Neste estudo de caso, partiremos dos requisitos funcionais do software. A partir deles, será elaborado o diagrama de casos de uso do sistema. Com o diagrama de casos de uso elaborado, serão especificados três casos de uso. Por último, será elaborado o diagrama de classes com base nos requisitos funcionais e casos de uso especificados. Nesta aula, o foco é a atualização do diagrama de classes na Visual Paradigm através da definição das operações identificadas nas classes existentes.

Tipo: Projeto
Título: Introdução à Construção de Diagrama de Classes da UML – Parte 26
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula dá continuidade ao estudo de caso de um sistema de gestão de festas. Neste estudo de caso, partiremos dos requisitos funcionais do software. A partir deles, será elaborado o diagrama de casos de uso do sistema. Com o diagrama de casos de uso elaborado, serão especificados três casos de uso. Por último, será elaborado o diagrama de classes com base nos requisitos funcionais e casos de uso especificados. Nesta aula, o foco é a identificação das associações existentes entre as classes no modelo.

ÍNDICE

06- Métricas de Acompanhamento para Metodologias Ágeis
Danilo Sato

18 - UML – Diagrama de Classes
Ana Cristina Melo

24 - Utilização do Controle Estatístico de Processos na Melhoria de Processos de Software
Monalessa Perini Barcellos

33 - Estimativas de Software - Fundamentos, Técnicas e Modelos – Parte 2
Carlos Eduardo Vazquez

44 - Introdução ao Project Management Body of Knowledge (PMBoK)
Paulo Augusto Oyamada Tamaki

60 - Ferramentas de Qualidade de Código
Wander Antunes Gaspar Valente

Métricas de Acompanhamento para Metodologias Ágeis

De que se trata o artigo?

Este artigo apresenta os conceitos relacionados às métricas para auxiliar o tracker de uma equipe ágil, discutindo definições, classificações, diferentes abordagens para escolha das melhores métricas e, por fim, apresentando alguns exemplos que podem ser utilizados no acompanhamento de uma equipe ágil.

Para que serve?

Métricas é um tema muito discutido em toda a comunidade de engenharia de software. Sua importância está no fato de que possibilita um acompanhamento mais preciso das atividades do projeto. Em particular, métricas podem ser uma ótima ferramenta no apoio ao controle de projetos de desenvolvimento de software utilizando metodologias ágeis.

Em que situação o tema é útil?

Apoiar o responsáveis pelo projeto na definição de abordagens mais precisas de acompanhamento de forma a possibilitar a melhoria dos trabalhos na organização.

Danilo Sato

danilo@dtsato.com / www.dtsato.com

Danilo Sato é Desenvolvedor, Coach e Consultor da ThoughtWorks UK. Com formação e mestrado em Ciência da Computação pelo IME/USP no tema Métodos Ágeis, é também fundador do Dojo@SP e membro da AgilCoop. Danilo já ministrou cursos sobre diversos assuntos relacionados a Métodos Ágeis e palestrou em diversas conferencias, incluindo: XP 2007 (Itália), Agile 2008 (Canadá), Agiles 2008 (Argentina) e Falando em Agile (Brasil).

Os Métodos Ágeis promovem um processo empírico para o desenvolvimento de software. Essa abordagem exige um ciclo constante de inspeção, adaptação e melhoria. Encontrar maneiras eficazes de avaliar o processo e a equipe de desenvolvimento não é uma tarefa simples. Isso leva a uma proliferação de medidas baseadas na premissa de que se cada parte do processo for otimizada, os resultados do processo como um todo serão otimizados também. No entanto, essa premissa nem sempre é verdadeira. Ao tentar micro-otimizar partes de um sistema por meio de diversas métricas, o verdadeiro objetivo se perde em meio a tantos substitutos e a equipe perde sua capacidade de tomar decisões de balanceamento (trade-off) [15]. Além disso, a preocupação com as medidas erradas pode gerar incentivos errados, levando a consequências indesejáveis. Goldratt diz que as pessoas se comportam de acordo com a forma com que estão sendo medidas: "Diga-me como serei avaliado e eu lhe direi como me comportarei" [1].

A escolha das melhores formas de medição é uma tarefa do Time Completo e o tracker possui um papel especial. Jeffries [2] e Auer [3] descrevem o papel do tracker como alguém responsável por prover informações para a equipe sobre o progresso

do time, utilizando as métricas apropriadas para destacar os pontos de melhoria e atualizando regularmente essas informações nos gráficos e pôsteres na área de Trabalho Informativa, que Cockburn chama de radiadores de informação [4].

Este artigo apresenta os conceitos relacionados às métricas para auxiliar o tracker de uma equipe ágil, discutindo definições, classificações, diferentes abordagens para escolha das melhores métricas e, por fim, apresentando alguns exemplos que podem ser utilizados no acompanhamento de uma equipe ágil.

Definições

Para discutir o papel das métricas no acompanhamento de projetos ágeis, primeiro é preciso definir alguns conceitos que nem sempre são usados da forma correta, com uma troca frequente de significados. Em particular, é importante conhecer as diferenças sutis entre os conceitos de medidas, métricas e indicadores.

Segundo o IEEE, uma medida é uma avaliação em relação a um padrão [5]; McGarry diz que é a avaliação de um atributo segundo um método de medição específico, funcionalmente independente de todas as outras medidas e capturando informação sobre um único atributo [6]. Um exemplo de medida é 5 cm: centímetro é o padrão e 5 é a medida, que indica quantos múltiplos ou frações do padrão estão sendo representados. Em desenvolvimento de software, um exemplo de medida pode ser o número de linhas de código. No entanto, não existe um padrão universal para representar linhas de código, pois as linguagens podem variar, assim como as regras para cálculo de linhas de código. Portanto, uma medida pode ser baseada em um padrão local ou universal, mas o padrão precisa ser bem definido. Uma métrica é um método para determinar se um sistema, componente ou processo possui um certo atributo [7]. Ela é geralmente calculada ou composta por duas ou mais medidas. Um exemplo de métrica é o número de defeitos encontrados após a implantação: as medidas que compõem essa métrica são o número de defeitos e a fase (ou data) onde o defeito foi identificado.

Um indicador é um dispositivo ou variável que pode ser configurado para um determinado estado com base no resultado de um processo ou ocorrência de uma determinada condição. Por exemplo: um semáforo ou uma flag [7]. Conforme a definição do IEEE, um indicador é algo que chama a atenção para uma situação particular. Ele geralmente está relacionado a uma métrica e provê a interpretação daquela métrica numa determinada situação ou contexto. Sempre que alguém interpreta alguma métrica, está considerando algum tipo de indicador, seja ele algum valor base ou outra métrica. Por exemplo: um aumento substancial no número de defeitos encontrados na última versão pode ser um indicador de que a Qualidade do software piorou. O seguinte exemplo fictício demonstra a relação entre medidas, métricas e indicadores: ao término da primeira iteração de um projeto, constata-se que a equipe entregou 4 Histórias, somando um total de 20 pontos (**Figura 1 (a)**). Conforme a equipe vai terminando as próximas iterações, percebe-se que o número total de pontos entregues aumenta aos poucos. Após certo tempo, essa tendência de subida é interrompida e o número total de pontos entregues cai um pouco, atingindo um patamar (**Figura 1 (b)**).

A **Figura 1 (a)** representa uma medida. Sem nenhuma outra informação para comparar ou uma tendência para seguir, uma medida não provê muita informação. A **Figura 1 (b)** representa uma métrica, nesse caso a velocidade da equipe. Uma métrica é composta por diversas medidas como o número de pontos entregues e o número da iteração terminada. Assim que a métrica se estabiliza, a equipe pode considerar um patamar para sua velocidade. Esse valor base, apresentado na **Figura 1 (c)** representa um indicador. Ele dá um contexto para a métrica, servindo como base para comparação. Uma métrica é sempre interpretada sob um ponto de vista específico. Por isso, é possível derivar diversos indicadores a partir da mesma métrica. O significado de um indicador sempre depende de um contexto, portanto duas equipes podem analisar a mesma métrica de forma diferente. Supondo outro cenário, onde essa mesma equipe utilizasse como indicador um valor base inferior para sua velocidade (derivado de outros projetos, por exemplo). Nesse caso, a mesma métrica **Figura 1 (b)** poderia ser interpretada como uma melhoria. No caso anterior, a equipe não tinha nenhum indicador prévio e houve apenas uma estabilização para que esse valor base fosse descoberto.

A palavra métrica será utilizada neste texto daqui em diante. No entanto, sempre que uma métrica for interpretada, avaliada ou analisada, o conceito de um indicador estará sempre implícito. Da mesma forma, toda métrica depende de medidas, portanto elas também estão sendo consideradas.

Classificações

As métricas podem ser classificadas segundo diferentes critérios. Esta seção apresenta algumas das possíveis classificações que um tracker precisa considerar quando utilizar uma métrica.

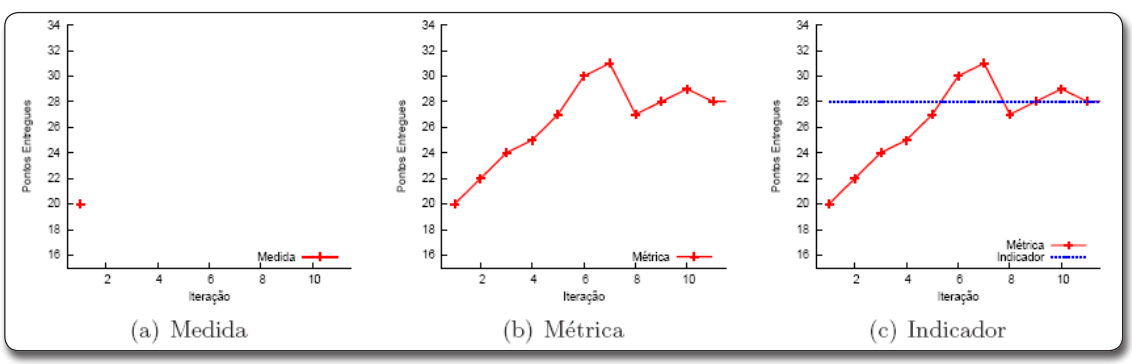


Figura 1. Total de Pontos Entregues por Iteração

Objetiva/Subjetiva

Conforme discutido anteriormente, uma métrica é composta por medidas que avaliam atributos de um objeto. O valor de uma métrica objetiva depende somente do objeto em questão e não do ponto de vista de quem a está interpretando. Por exemplo: número de commits no repositório é uma métrica objetiva, pois é obtida diretamente da ferramenta. Por outro lado, o valor de uma métrica subjetiva depende do objeto em questão e também do ponto de vista de quem a está interpretando. Um exemplo de métrica subjetiva é a qualidade do código, numa escala de 0% a 100%. Apesar da escala definir um intervalo numérico, a natureza da métrica ainda é subjetiva, pois depende do ponto de vista de quem está avaliando.

Os critérios para definir a "qualidade do código" variam de pessoa para pessoa. Métricas objetivas são passíveis de serem obtidas de forma automatizada.

Quantitativa/Qualitativa

Além da natureza objetiva/subjetiva, uma métrica pode ainda ser classificada como quantitativa ou qualitativa. O valor de uma métrica quantitativa pertence a um intervalo de uma certa magnitude e geralmente é representado por um número. Tal estrutura permite que medidas quantitativas sejam comparadas entre si. A maioria dos exemplos utilizados até aqui neste artigo representam métricas quantitativas, como o número de linhas de código ou o número de defeitos encontrados. Por outro lado, valores de uma métrica qualitativa são aqueles representados por palavras, símbolos ou figuras ao invés de números [8]. Um exemplo de métrica qualitativa é o humor da equipe ou a satisfação do cliente.

A maioria dos estudos empíricos em Engenharia de Software usa uma combinação entre métodos quantitativos e qualitativos. Uma das formas mais comuns de combinar ambas as estratégias é a codificação, que consiste na extração de valores quantitativos dos dados qualitativos para permitir o tratamento estatístico ou outra abordagem quantitativa [9].

Vale ressaltar que a classificação de uma métrica como quantitativa ou qualitativa é ortogonal à classificação como objetiva ou subjetiva. Geralmente uma métrica quantitativa é objetiva e uma qualitativa é subjetiva, mas isso não é sempre verdade. O processo de codificação transforma uma métrica qualitativa em quantitativa, mas não altera sua objetividade ou subjetividade. Por exemplo, considere a seguinte frase que constitui um fragmento de dado qualitativo: "João, Maria e Pedro foram os únicos que participaram da reunião". Isso poderia ser transformado num dado quantitativo como: "numero de participantes = 3". A informação continua objetiva após a codificação. Além disso, uma parte da informação é perdida, pois não sabemos mais os nomes dos participantes. Considerando agora outro exemplo de dado qualitativo: "Paulo disse que essa classe Java é bem simples de entender e sua complexidade é bem menor que as outras classes do sistema". Tal informação poderia ser codificada para o seguinte dado quantitativo: "complexidade = baixa". Houve novamente perda de informação no processo de codificação e, apesar do valor quantitativo parecer mais objetivo, continua tão subjetivo quanto antes.

Organizacional/Acompanhamento

Hartmann e Dymond sugerem outra categoria para classificação, distinguindo métricas organizacionais de métricas de diagnóstico [10]. Neste texto, ao invés de usar o termo "diagnóstico" usarei o termo "acompanhamento" por estar mais alinhado com o tema do artigo e por compartilhar as mesmas características de "diagnóstico" propostas por Hartmann e Dymond. Métricas organizacionais são aquelas que medem a quantidade de valor de negócio entregue ao cliente. Essa definição levanta algumas perguntas: em primeiro lugar, quem é o cliente? Collins sugere que, no nível organizacional, o cliente deve ser o dono ou o responsável pelo produto (stakeholder) ou talvez o usuário final [11]. Isso deixa a segunda pergunta em aberto, o que é valor? No seu livro, Collins discute os atributos e comportamentos em comum das empresas que passaram de um longo histórico de resultados medíocres para um longo histórico de resultados extraordinários. Ele mostra que as empresas que foram do bom para o ótimo escolheram um fator-chave único de direcionamento econômico como métrica para auxiliar na tomada de decisão. Idealmente, essa métrica-chave deve ser definida pelos executivos da empresa, porém em projetos de código aberto, onde o objetivo final nem sempre é financeiro, outros fatores de sucesso como número de usuários ou a satisfação do usuário podem ser utilizados. A seção *Métricas Organizacionais* apresentará com mais detalhes alguns exemplos de métricas organizacionais.

Métricas de acompanhamento provêem informações que ajudam o time no entendimento e Melhoria do processo que produz valor de negócio. Uma vez que uma métrica organizacional ampla é definida, a equipe precisa de medições locais para auxiliá-los a atingir o objetivo. Essas métricas agregam dados, porém não os associam a nenhum indivíduo. Elas existem e têm validade dentro de um contexto particular, porém recomenda-se que elas sejam escolhidas com cuidado e utilizadas somente durante certo período de tempo. Métricas de acompanhamento devem ser descartadas uma vez que tenham servido seu propósito. A meta mais ampla definida pela métrica organizacional deve guiar a utilização de diferentes métricas de acompanhamento temporárias. Um exemplo de métrica de acompanhamento já citado neste artigo é a velocidade da equipe.

Poppendieck utiliza uma nomenclatura diferente para as métricas organizacionais e de acompanhamento, chamando-as de métricas de avaliação de desempenho e métricas informativas, respectivamente [12]. Apesar dos nomes serem diferentes, as características descritas nesta seção são as mesmas.

O Que Medir?

Nesta seção são apresentadas algumas abordagens para escolher quais métricas utilizar, apresentando também algumas das características de uma boa métrica ágil. Um tracker deve utilizar as abordagens apresentadas aqui juntamente com o conhecimento sobre sua equipe ao escolher as melhores métricas para sua situação.

Abordagem Objetivo-Pergunta-Métrica (Goal Question Metric)

Uma das abordagens mais conhecidas e utilizadas em estudos empíricos em Engenharia de Software é a abordagem objetivo-pergunta-métrica (Goal Question Metric ou GQM), proposta por Basili [13]. O modelo de medição proposto pelo GQM é composto de três níveis:

- Nível Conceitual (Objetivo): Um objetivo é definido para um objeto em relação a algum modelo de qualidade, a partir de diversos pontos de vista e para um ambiente específico. Um objeto pode ser um produto (documento, código-fonte, testes, etc.), um processo (especificação, entrevista, codificação, etc.) ou um recurso (pessoas, hardware, espaço físico, etc.);
- Nível Operacional (Pergunta): Um conjunto de perguntas caracterizam a forma de avaliação e cumprimento do objetivo escolhido. As perguntas tentam relacionar o objeto de estudo com as características de Qualidade desejáveis a partir do ponto de vista definido;
- Nível Quantitativo (Métrica): Um conjunto de dados é associado com cada pergunta para tentar encontrar uma forma quantitativa de respondê-la. Métricas podem ser objetivas ou subjetivas.

Esse modelo define uma estrutura hierárquica que começa com um objetivo claro. O formato proposto pela abordagem GQM para definir um objetivo é composto de: uma motivação, uma preocupação ou tópico, um objeto e um ponto de vista. A partir desse objetivo, uma série de perguntas é definida e, a partir dessas perguntas, uma série de métricas é escolhida. A mesma métrica pode ser usada para responder diferentes perguntas. Da mesma forma, métricas e perguntas podem ser utilizadas em mais de um modelo GQM para diferentes objetivos, porém a forma de medição deve levar em conta os diferentes pontos de vista de cada modelo. A **Tabela 1** apresenta um exemplo de um modelo GQM.

Segundo Basili, a definição de um objetivo deve se basear em três fontes de informação [13]: a primeira fonte de informação são as políticas e estratégias organizacionais, de onde derivam a motivação e o tópico do objetivo; a segunda fonte são as descrições dos processos e produtos da empresa, de onde deriva o objeto de avaliação; por fim, a terceira fonte de informação é o modelo organizacional da empresa, de onde derivam os pontos de vista que serão levados em conta na avaliação do objetivo.

Objetivo	Motivação Tópico Objeto Ponto de vista	Melhorar o tempo gasto no processo de correção de defeitos sob o ponto de vista gerencial.
Pergunta		Qual é a velocidade atua de correção de um defeito?
Métricas		Tempo médio de ciclo Desvio padrão % de casos acima do limite máximo
Pergunta		O processo de correção de defeitos está melhorando?
Métricas		(Tempo médio de ciclo atual/Tempo médio de ciclo desejado) * 100 Avaliação subjetiva do gerente responsável

Tabela 1. Exemplo de modelo Objetivo-Pergunta-Métrica

Abordagem Lean

Ao adaptar os conceitos Lean que funcionaram bem para a manufatura, cadeia de suprimentos e desenvolvimento de produtos, Poppendieck propõe uma abordagem para escolha das métricas mais apropriadas [12, 15]. Conforme discutido na seção *Organizacional/Acompanhamento*, a abordagem Lean distingue bem as métricas organizacionais das métricas de acompanhamento e um de seus princípios para o desenvolvimento de software é "Otimizar o todo". Sua proposta para o uso de métricas na avaliação de desempenho é medir sempre um nível acima. Uma tendência natural para avaliação de desempenho é a decomposição. O senso comum diz que se as partes de um sistema forem otimizadas, o sistema todo também o será. No entanto a micro-otimização tende a degradar o resultado geral, pois não é possível medir tudo. Austin discute os problemas da avaliação de desempenho e destaca que sua principal vantagem ("Você tem o que você mede") é também seu principal problema ("Você tem exatamente o que você mede, e nada mais"). Fatores importantes acabam ficando fora da decomposição por não serem facilmente mensurados, como: criatividade, insight, colaboração, dedicação e preocupação com a satisfação do cliente [14].

Quando a avaliação de um indivíduo é realizada por uma métrica que leva em conta apenas fatores dentro do seu campo de influência, ele tende a trabalhar para otimizar essa métrica, deixando de pensar no sistema como um todo. A prática sugerida pela abordagem Lean é medir sempre em um nível acima. Quando a avaliação leva em conta o Time Completo ou a empresa toda, ela gera incentivos para que os indivíduos trabalhem de forma colaborativa para atingir um resultado comum. Por isso, ao invés de criar diversas métricas, é mais importante reduzir o número de métricas organizacionais, escolhendo uma que defina um objetivo amplo, gerando os incentivos que farão com que o comportamento dos subsistemas otimizem o todo [15].

Além das métricas organizacionais, a abordagem Lean também sugere o uso de métricas de acompanhamento para auxiliar a equipe. No entanto, essas métricas devem ser definidas de forma a ocultar o desempenho individual. Quando a contagem de defeitos leva em conta o indivíduo que causou o erro, ela passa a ser uma métrica de avaliação, gerando os incentivos errados. Deming diz que a baixa qualidade nunca é responsabilidade de um indivíduo, mas sim de um processo de gerenciamento que permite que a ausência de defeitos seja impossível [16]. Por isso, é importante que uma métrica de acompanhamento seja totalmente desassociada de qualquer avaliação de desempenho. Seu propósito deve ser meramente informacional.

Retrospectivas

Reuniões de Retrospectiva encorajam a discussão constante sobre o processo e a forma de trabalho da equipe. Elas são o momento de Reflexão que deve fazer parte do ciclo de inspeção e adaptação proposto pelos Métodos Ágeis. Segundo Cockburn, o criador da Família Crystal onde a Retrospectiva é prática fundamental, elas ajudam a encontrar o processo mais aceitável para a equipe [4].

O resultado de uma reunião de Retrospectiva costuma ser um pôster destacando os principais pontos de melhoria que a equipe escolheu se concentrar na próxima iteração. A partir dessa discussão, o tracker pode escolher algumas métricas de acompanhamento para auxiliar a equipe a entender o progresso em relação aos pontos de melhoria levantados. A seção Métricas de Acompanhamento apresenta exemplos de métricas de acompanhamento para auxiliar o tracker e a equipe nessa escolha.

Características de uma boa métrica ágil

Com base em diversas fontes, Hartman e Dymond propõem uma compilação de algumas das heurísticas que um tracker deve considerar quando estiver escolhendo uma métrica para sua equipe [10]. Uma boa métrica ágil deve:

- **Reforçar princípios ágeis:** colaboração com o cliente e entrega de valor são princípios fundamentais para os Métodos Ágeis;
- **Medir resultados e não saídas:** ao valorizar a Simplicidade, os melhores resultados podem ser aqueles que minimizam a quantidade de trabalho realizado. Medir os resultados obtidos é mais importante que medir as saídas das atividades do processo;
- **Seguir tendências e não números:** os valores representados por uma métrica são menos importantes que a tendência demonstrada. Ao medir a velocidade da equipe, é melhor se preocupar com sua estabilização do que com o valor absoluto que ela representa;
- **Responder uma pergunta específica para uma pessoa real:** toda métrica deve expor informação para um ponto de vista específico. Se outra pessoa tem outra pergunta, é melhor usar outra métrica;
- **Pertencer a um conjunto pequeno de métricas e diagnósticos:** é impossível medir tudo. Muita informação pode esconder o que realmente importa. Minimize o número de métricas e meça o que é mais importante;
- **Ser facilmente coletada:** para métricas de acompanhamento objetivas e quantitativas, o ideal é ter uma coleta automatizada;
- **Revelar, ao invés de esconder, seu contexto e suas variáveis:** uma métrica deve deixar claro os fatores que a influenciam para evitar manipulações e facilitar a Melhoria do processo;
- **Incentivar a Comunicação:** uma métrica isolada de seu contexto perde o sentido. Um bom sinal é quando as pessoas comentam o que aprenderam com uma métrica;
- **Fornecer Feedback frequente e regular:** para amplificar o aprendizado e acelerar o processo de Melhoria, as métricas devem ser frequentemente atualizadas e disponibilizadas na área de Trabalho Informativa;
- **Encorajar um alto nível de Qualidade:** o nível aceitável de Qualidade deve ser definido pelo cliente e não pela equipe. Os Métodos Ágeis exigem sempre um alto nível de Qualidade do software desenvolvido.

Discussão

As abordagens apresentadas na seção *O Que Medir?* possuem vantagens e desvantagens e o tracker de uma equipe ágil deve saber balanceá-las. O modelo GQM propõe uma abordagem top-down para a definição de métricas que por um lado esclarece os objetivos por trás de cada métrica e evita a medição das coisas erradas, porém por outro lado sua estrutura hierárquica estimula a proliferação de diversas métricas. Já a abordagem Lean distingue bem os dois níveis de medição (organizacional/acompanhamento) e estimula um menor número de métricas organizacionais. A prática de medir sempre um nível acima evita a proliferação de métricas e cria os incentivos para a colaboração entre todos os responsáveis pelo Fluxo de entrega de valor do sistema. Apesar de ambas as abordagens funcionarem para a definição e escolha das métricas organizacionais, não dão muita direção em relação à escolha das métricas de acompanhamento. As reuniões de Retrospectiva resolvem esse problema, fazendo com que a própria equipe discuta e reflita sobre os pontos do processo que podem ser melhorados. Essa discussão auxilia a escolha das melhores métricas de acompanhamento.

Com base nas características apresentadas na seção *Características de Uma Boa Métrica Ágil*, Hartman e Dymond sugerem que o tracker utilize uma lista de verificação ao escolher uma métrica [10]. Levando em consideração a discussão sobre as diferentes abordagens, o autor propõe uma adaptação dessa lista, apresentada na **Tabela 2**.

Exemplos

Esta seção apresenta alguns exemplos de medidas e métricas organizacionais e de acompanhamento. Essa lista não pretende ser exaustiva, enumerando todas as possíveis medidas e métricas.

Medidas

As medidas apresentadas nesta seção serão separadas de acordo com sua respectiva classificação (conforme descrito na Seção Classificações). Abaixo seguem exemplos de medidas quantitativas e objetivas. Por sua natureza, elas podem ser coletadas de forma automatizada.

- **Total de Linhas de Código (TLC):** representa o número total de linhas de código de produção do sistema, descartando linhas em branco e comentários. É comum a utilização de variações dessa medida, como contando aos milhares (*Thousand Lines of Executable Code* ou KLOEC) ou considerando o número de **Linhas de Código por Classe (LCC)**;
- **Total de Linhas de Código de Teste (TLCT):** representa o número total de pontos de teste do sistema, conforme definido por Dubinsky [17]. Um ponto de teste é considerado como um passo do cenário de um teste de aceitação automatizado ou como uma linha de código de teste de unidade automatizado, descartando linhas em branco e comentários;
- **Número de Linhas Alteradas:** representa o número de linhas (não apenas código-fonte) adicionadas, removidas e atualizadas no Repositório de Código Unificado;
- **Número de Commits** representa o número de commits efetuados no Repositório de Código Unificado;

Lista de Verificação de uma Métrica Ágil	
Característica	**Descrição**
Nome	Deve ser bem escolhido para evitar confusão e ambigüidade
Classificação	Conforme apresentado na Seção Classificações
Objetivo	Conforme definido no modelo GQM, incluindo uma motivação, uma preocupação ou tópico, um objeto e um ponto de vista
Pergunta	Conforme definido no modelo GQM, toda métrica deve estar associada a uma pergunta específica
Base de Medição	Uma clara definição das medidas utilizadas para cálculo da métrica
Suposições	Devem ser identificadas para um claro entendimento do que os dados estão representando
Tendência Esperada	Uma idéia de qual o comportamento esperado para a métrica
Quando utilizar?	Deve esclarecer os motivos que levaram à criação da métrica e, caso a métrica já tenha sido utilizada anteriormente, mostrar um pouco do seu histórico
Quando parar de utilizar?	É importante saber até quando uma métrica será útil, antes mesmo de utilizá-la, principalmente para métricas de acompanhamento
Formas de Manipulação	Deve esclarecer como as pessoas tentarão alterar seu comportamento em função da métrica para gerar números "mais favoráveis"
Cuidados e Observações	Recomendações sobre outras métricas similares, limites no uso e perigos associados à má utilização da métrica

Tabela 2. Lista de verificação ao escolher uma métrica, adaptado de [10]

• **Estimativas Originais:** representa o total de pontos (ou horas) originalmente estimadas pela equipe para todas as Histórias da iteração. Beck e Fowler sugerem a utilização de "horas ideais" [18] nas estimativas e controle da iteração, mas a unidade de medida efetivamente utilizada não importa tanto, contanto que seja usada consistentemente durante o projeto;
• **Estimativas Finais:** representa o total de pontos (ou horas, ou "horas ideais") efetivamente reportadas como gastas para implementar as Histórias da iteração;
• **Número de Histórias Entregues:** representa o número total de Histórias implementadas e aceitas pelo cliente;
• **Número de Pontos Entregues:** representa o número total de pontos implementados e aceitos pelo cliente;
• **Tempo:** utilizado no cálculo de diversas métricas, pode ser utilizado como duração (em meses, semanas, dias, etc.) ou como número de iterações;
• **Tamanho da Equipe:** geralmente utilizado no cálculo de métricas, representa a quantidade de pessoas na equipe;
• **Esforço:** uma medida que leva em conta o tamanho da equipe durante um certo intervalo de tempo, geralmente medido em pessoas-mês ou pessoas-ano;
• **Complexidade Ciclomática de McCabe** (v(G))**:** mede a quantidade de lógica de decisão num único módulo de software [19]. Num sistema orientado a objetos, um módulo é um método. Um Grafo de controle de fluxo descreve a estrutura lógica de um algoritmo através de vértices e arestas. Os vértices representam expressões ou instruções computacionais (como atribuições, laços ou condicionais), enquanto as arestas representam a transferência do controle entre os vértices [20]. A Complexidade Ciclomática é definida para cada módulo como $e \quad n + 2$, onde e e n são o número de arestas e vértices do grafo de controle de fluxo, respectivamente;
• **Métodos Ponderados por Classe** (*Weighted Methods per Class* ou WMC): mede a complexidade de uma classe num sistema orientado a objetos. Definida para cada classe como a soma ponderada de todos os seus métodos [21]. É comum utilizar $v(G)$ como fator de peso, então WMC pode ser calculada como $\sum ci$, onde ci é a Complexidade Ciclomática do i-ésimo método de uma mesma classe;
• **Falta de Coesão dos Métodos** (*Lack of Cohesion of Methods* ou LCOM): mede a coesão de uma classe e é calculada através do método de Henderson-Sellers [22]. Se m(A) é o número de métodos que acessam o atributo A, LCOM é calculada como a média de m(A) para todas os atributos, subtraindo o número de métodos m e dividindo o resultado por (1 - m). Um valor baixo indica uma classe coesa, enquanto um valor próximo de 1 indica falta de coesão;
• **Profundidade da árvore de Herança** (*Depth of Inheritance Tree* ou DIT): o comprimento do maior caminho a partir de uma classe até a classe-base da hierarquia;
• **Número de Filhos** (*Number of Children* ou NOC): o número total de filhos imediatos de uma classe;
• **Acoplamento Aferente** (*Afferent Coupling* ou AC): o número total de classes de fora de um pacote que dependem de classes de dentro do pacote. Quando calculada no nível da classe, essa medida também é conhecida como Fan-in da classe;
• **Acoplamento Eferente** (*Efferent Coupling* ou EC): o número total de classes de dentro de um pacote que dependem de classes de fora do pacote. Quando calculada no nível da classe, essa medida também é conhecida como Fan-out da classe, ou como CBO (Coupling Between Objects ou Acoplamento entre Objetos) na família de métricas CK (Chidamber-Kemerer) [21].

Alguns dos fatores de desenvolvimento propostos por Boehm e Turner [23] servem como exemplos de medidas quantitativas e subjetivas:
• **Dinamismo:** A quantidade de mudanças de requisitos por mês;
• **Cultura:** Uma medida da porcentagem da equipe que prefere trabalhar em um cenário caótico ao invés de um cenário ordenado, ou seja, a porcentagem da equipe capaz de aceitar mudanças durante o projeto;
• **Criticalidade:** O impacto causado por uma falha no software, podendo afetar desde uma quantia aceitável de investimento até a perda de uma vida;
• **Nível Pessoal:** A porcentagem da equipe que pertence aos diversos níveis propostos por Cockburn [4];

• **Aderência às Práticas de XP:** uma forma de medir o grau de utilização de cada prática de XP. Cada integrante dá uma nota para o nível "atual" e "desejado" da equipe em relação a cada prática de XP.

Por fim, alguns exemplos de medidas qualitativas (que podem ser codificadas para medidas quantitativas, conforme descrito na seção *Quantitativa/Qualitativa*) e subjetivas são:
• **Moral da Equipe:** mede o humor e a motivação de cada membro da equipe;
• **Satisfação do Cliente:** mede o nível de satisfação do cliente com o produto desenvolvido.

Métricas Organizacionais

Esta seção apresenta quatro exemplos de métricas organizacionais, que podem ser utilizadas para avaliação de uma equipe ágil: "Funcionalidades Testadas e Entregues" é uma métrica proposta por Ron Jeffries [24], "Tempo Médio de Ciclo" foi proposta pela abordagem Lean [12, 15], "Retorno Financeiro" é uma métrica mais amplamente citada [25, 15, 10] e "Net Promoter Score" foi proposta por Reichheld [26]. Elas serão apresentadas conforme o formato proposto na seção Discussão.

Funcionalidades Testadas e Entregues (Running Tested Features ou RTF)

• **Classificação:** Quantitativa e subjetiva, pois o cliente define quando uma funcionalidade está pronta através dos testes de aceitação;
• **Objetivo:** Maximizar a quantidade de valor de negócio entregue em cada funcionalidade do sistema, sob o ponto de vista do cliente;
• **Pergunta:** Qual é a taxa de valor de negócio entregue por funcionalidade testada e implantada? Quando o software deixa de ser inventário e passa a agregar valor?
• **Base de Medição:** Requisitos são quebrados em Histórias (Funcionalidades). Testes de aceitação automatizados são definidos pelo cliente e servem como critério para avaliar quando a História está pronta (Testada). Toda funcionalidade pronta deve estar integrada num único produto, com a possibilidade de ser implantado e prontamente utilizado (Entregue);
• **Suposições:** O cliente deve trabalhar em colaboração com a equipe, definindo Histórias e cenários de aceitação. Uma funcionalidade testada e pronta só poderá trazer valor efetivo de negócio quando entrar em produção;
• **Tendência Esperada:** RTF deve crescer linearmente logo após o início do projeto e até o seu término. As funcionalidades que trazem mais valor serão implementadas primeiro;
• **Quando utilizar?** Para avaliar a execução de projetos ágeis. Para entregar funcionalidades testadas a partir do início do projeto, a equipe vai precisar de práticas ágeis. Ela não terá tempo de fazer muito design prematuro (Big Design Up-Front), assim como não poderá se esquecer dos testes automatizados;
• **Quando parar de utilizar?** Quando o projeto terminar ou o retorno financeiro não justificar seu prolongamento;
• **Formas de Manipulação:** Uma forma de manipular essa métrica é através da entrega das funcionalidades mais fáceis, ao invés das mais importantes. Definir poucos testes também fará com que a funcionalidade esteja pronta mais rapidamente;
• **Cuidados e Observações:** Medir funcionalidade entregue pode não refletir diretamente em valor de negócio. O excesso de funcionalidades é, na verdade, um dos grandes inimigos do desenvolvimento de software. É importante que as funcionalidades entregues no início do projeto sejam as mais importantes e com maior valor de negócio.

Tempo Médio de Ciclo (Cycle Time)

• **Classificação:** Qualitativa e objetiva;
• **Objetivo:** Minimizar o tempo médio de ciclo de um sistema, sob o ponto de vista do cliente final (usuário);
• **Pergunta:** Quanto tempo é gasto do conceito ao retorno financeiro (concept to cash)? Ou entre uma "ordem" de compra de um cliente e sua realização em forma de software entregue? Ou entre a identificação de um defeito em produção e sua resolução? O importante não é descobrir o quão rápido o sistema pode entregar valor, mas sim o quão rápido o sistema entrega valor de forma repetitiva e confiável;
• **Base de Medição:** Com base num mapa de Fluxo de valor do sistema todo (Value Stream Map), que sempre começa e termina com o cliente e, para cada parte do processo, mapeia quanto tempo é gasto agregando valor ao sistema e quando tempo é desperdiçado em tarefas que não agregam valor [12]. O tempo de ciclo é calculado de forma objetiva através da medição do tempo que uma "ordem" leva para passar pelo mapa de Fluxo de valor do sistema;
• **Suposições:** A equipe conhece as etapas do processo por onde uma "ordem" deve passar e o processo está mapeado num mapa de Fluxo de valor. O tempo total gasto para processar a "ordem" considera todos os tipos de desperdício no sistema: falta de habilidades, atrasos, hand-offs entre diferentes equipes, etc.;
• **Tendência Esperada:** O tempo médio de ciclo do sistema deve diminuir gradualmente conforme o processo é melhorado, até atingir um patamar mínimo. No entanto, a equipe tem autonomia para estar constantemente desafiando o processo e melhorando o continuamente, diminuindo ainda mais o tempo médio de ciclo;
• **Quando utilizar?** Para avaliar o comprometimento das pessoas com o sistema todo. Ela exige que outras métricas mais locais sejam otimizadas, porém o contrário não é verdade. A preocupação em otimizar subsistemas provavelmente fará com que o tempo de ciclo aumente [15];
• **Quando parar de utilizar?** Quando o sistema não precisar mais produzir valor. Formas de Manipulação: Por ser uma métrica ampla, essa métrica cria os incentivos para que as equipes cruzem barreiras do processo, exigindo que as pessoas colaborem para criar um produto de Qualidade. Dessa forma, fica difícil manipular essa medida;
• **Cuidados e Observações:** O tempo médio de ciclo tende a diminuir quando o sistema trabalha com peças pequenas (small batches). No caso do desenvolvimento de software, por exemplo, essas peças podem ser Histórias ou defeitos. Se

o tempo médio de ciclo se estabilizar, isso não significa que a equipe chegou ao valor ótimo. Ela deve estar constantemente desafiando os padrões atuais para encontrar formas de reduzir o tempo médio de ciclo [15].

Retorno de Investimento (Return of Investment ou ROI)

• **Classificação:** Quantitativa e objetiva;
• **Objetivo:** Minimizar o tempo gasto até um sistema trazer retorno financeiro, sob o ponto de vista do cliente;
• **Pergunta:** Qual é a taxa de retorno financeiro do projeto? Quando ele começará a ser obtido? Qual o lucro esperado com o investimento num sistema de software?
• **Base de Medição:** Através de análise do fluxo de caixa financeiro por iteração;
• **Suposições:** A entrega de valor será realizada ao final da iteração, no entanto o retorno financeiro só será obtido quando o software entrar em produção;
• **Tendência Esperada:** As funcionalidades com maior valor e maior retorno serão entregues no início do projeto e, conforme as outras funcionalidades vão sendo implementadas, o retorno obtido por funcionalidade vai reduzindo;
• **Quando utilizar?** Ao analisar, priorizar e executar projetos onde algum retorno financeiro é esperado. Essa métrica deve ser compreendida por todos os envolvidos no projeto;
• **Quando parar de utilizar?** Quando o investimento for retirado ou quando as funcionalidades deixarem de agregar valor financeiro justificável;
• **Formas de Manipulação:** Entregar as funcionalidades menores, ao invés das que trazem mais valor financeiro, no início do projeto para que o ROI comece a ser obtido rapidamente. Isso pode afetar a taxa de retorno financeiro esperada;
• **Cuidados e Observações:** Assim que o software entra em produção, com um subconjunto das funcionalidades esperadas, o modelo de retorno financeiro deve ser reavaliado e as funcionalidades restantes re-priorizadas para levar em conta os dados reais de retorno financeiro. Além disso, a equipe toda deve ser exposta ao modelo de lucros e perdas que afetará o resultado financeiro do projeto, para que tenham o conhecimento necessário ao tomar decisões de compromisso (trade-off) durante o desenvolvimento.

Outras métricas para avaliação de retorno financeiro, como Net Present Value (NPV) ou Internal Rate of Return (IRR) também podem ser utilizados como métrica organizacional.

Net Promoter Score ou NPS

• **Classificação:** Quantitativa e subjetiva, pois depende da avaliação da satisfação do cliente;
• **Objetivo:** Maximizar a satisfação do cliente final (usuário), sob o ponto de vista do cliente (do projeto);
• **Pergunta:** Como distinguir entre um dólar de lucro do tipo bom, que trará crescimento, e um dólar de lucro do tipo ruim, que prejudicará o crescimento? Como avaliar a satisfação do cliente final de um projeto de software?
• **Base de Medição:** Calculada com base numa única e simples questão para o cliente final: "O quanto você recomendaria a empresa ou o produto X para um amigo ou colega?". Uma nota de 0 (não recomendaria) a 10 (definitivamente recomendaria) é obtida. Clientes com nota entre 9 e 10 são chamados de promotores, notas entre 7 e 8 representam clientes neutros, enquanto notas entre 0 e 6 representam clientes afastadores (retractors). O NPS é calculado subtraindo a porcentagem de clientes afastadores da porcentagem de clientes promotores, podendo variar entre -100% e 100%;
• **Suposições:** Seu produto atinge pessoas com poder de influência no público em geral;
• **Tendência Esperada:** Valores negativos do NPS representam um sério risco. Dessa forma, a tendência esperada para esta métrica é de crescimento. Empresas boas possuem um NPS médio de 10%, enquanto empresas realmente boas conseguem NPS de 50% [26];
• **Quando utilizar?** Quando o produto não possuir intenções financeiras (projetos de código aberto, por exemplo) ou quando seu público alvo tiver influência no sucesso do produto;
• **Quando parar de utilizar?** Quando o produto não estiver mais tentando penetração no mercado ou quando for finalizado;
• **Formas de Manipulação:** Como a influência da equipe nos clientes finais do produto é praticamente nenhuma, fica difícil manipulá-los para obter um NPS melhor;
• **Cuidados e Observações:** O valor bruto do NPS não pode usado para comparações diretas entre empresas ou linhas de produto devido a diferenças nos segmentos de mercado ou até a diferenças geográficas e culturais entre os clientes finais. Ele deve ser usado como um indicador de tendência dentro de uma única empresa ou produto.

Métricas de Acompanhamento

Nesta seção são apresentados três exemplos de métricas de acompanhamento, que podem ser utilizadas para auxiliar na Melhoria do processo utilizado pela equipe para trazer valor de negócio. A "Velocidade" é uma métrica amplamente utilizada por equipes ágeis [25, 18, 27, 10], enquanto o "Fator de Integração" e o "Fator de Teste" foram propostos pelo autor que, apesar de não conhecer uma citação específica na literatura, acredita que sua autoria não seja original. As métricas serão apresentadas conforme o formato proposto na seção *Discussão*. Por sua natureza mais descartável, o objetivo de uma métrica de acompanhamento, no formato proposto pelo modelo GQM, deve ser sempre "Melhorar a adoção da prática X sob o ponto de vista da equipe".

Velocidade

• **Classificação:** Quantitativa e objetiva;
• **Pergunta:** Quanto software a equipe consegue entregar por iteração?
• **Base de Medição:** Pontos (story-points) ou "horas ideais" entregues por iteração.
• **Suposições:** A equipe está entregando software funcionando a cada iteração;
• **Tendência Esperada:** A velocidade pode ser afetada por

diversos fatores, como: mudança na equipe, impedimentos, conhecimento das ferramentas e tecnologias utilizadas, etc. Um time estabilizado, durante um mesmo projeto e com todos os recursos necessários disponíveis tende a aumentar sua velocidade durante um certo tempo (geralmente no início), até atingir um patamar, onde ela se estabiliza;

• **Quando utilizar?** A velocidade é uma métrica de acompanhamento muito útil para a equipe conhecer seu limite e sempre conseguir atingir o objetivo de cada iteração. A velocidade informa a equipe e o cliente da capacidade de entregar software funcionando num ritmo constante e sustentável;

• **Quando parar de utilizar?** Quando o projeto acabar ou quando, após atingido o período de estabilização, a velocidade se tornar "conhecida" pela equipe e pelo cliente;

• **Formas de Manipulação:** Estimar mais pontos para o mesmo trabalho fará com que a velocidade aumente, por isso ela não pode ser usada para comparar diferentes equipes. Enquanto uma equipe estima 1000 pontos, uma outra pode estimar 600 pontos para fazer o mesmo trabalho, tornando a comparação inviável;

• **Cuidados e Observações:** Velocidade não representa valor agregado. Um time pode ter excelente velocidade, entregando software frequentemente durante um certo tempo, porém sem trazer o retorno (financeiro ou não) esperado. Comparar a velocidade de equipes diferentes também pode ser um problema (conforme discutido acima). Da mesma forma, medir a velocidade por membro da equipe também pode gerar conflitos e atrapalhar. Ela deve sempre ser medida no nível da equipe.

Fator de Integração

• **Classificação:** Quantitativa e objetiva;

• **Pergunta:** Quanto tempo a equipe demora para integrar com o repositório? Quanto código é alterado antes de ser integrado no repositório?

• **Base de Medição:** O fator de integração *IFi* para a iteração *i* é calculado da seguinte maneira:

$$IF_i = \frac{LA_i + LR_i + LU_i}{TC_i}$$

onde:

LAi = número total de linhas adicionadas na iteração i
LRi = número total de linhas removidas na iteração *i*
LUi = número total de linhas atualizadas na iteração *i*
TCi = número total de commits na iteração *i*

• **Suposições:** A equipe utiliza um sistema de controle de versão, onde o código é integrado frequentemente pelos integrantes da equipe;

• **Tendência Esperada:** Se a equipe está praticando a Integração Contínua de forma correta, o fator de integração deve ser baixo, indicando que há poucas linhas alteradas a cada commit. Dessa forma, a tendência esperada para uma equipe que está aprendendo a Integração Contínua é de diminuição;

• **Quando utilizar?** Quando os membros da equipe estiverem demorando muito tempo para sincronizar ou integrar código novo no repositório ou quando as alterações forem muito grandes;

• **Quando parar de utilizar?** Assim que o fator de integração chegar em um nível aceitável e a equipe deixar de ter problemas de sincronização de trabalho;

• **Formas de Manipulação:** Essa métrica pode ser manipulada se os integrantes da equipe resolverem fazer um commit no repositório a cada pequena alteração, aumentando muito a quantidade de commits. Isso deve ser desencorajado pois toda alteração no repositório deve mantê-lo num estado consistente, ou seja, os testes automatizados devem continuar passando;

• **Cuidados e Observações:** Essa métrica não leva em conta a qualidade do código que está sendo integrado ao repositório. Conforme discutido acima, um grande aumento no número de commits pode piorar a Integração Contínua e deixar o repositório num estado inconsistente. Manter a qualidade do código através de uma bateria de testes automatizados deve ser uma preocupação constante da equipe.

Fator de Teste

• **Classificação:** Quantitativa e objetiva;

• **Pergunta:** Qual a relação entre código de teste e código de produção que está sendo produzido pela equipe?

• **Base de Medição:** O fator de teste *Ti* para a iteração *i* é calculado como a razão entre o número de linhas de código de teste e o número de linhas de código de produção:

$$T_i = \frac{TLCT_i}{TLC_i}$$

onde:

TLCTi = número total de linhas de código de teste na iteração *i*
TLCi = número total de linhas de código de produção na iteração *i*

• **Suposições:** A equipe está desenvolvendo código de produção e código de testes automatizados para verificar a Qualidade do software produzido;

• **Tendência Esperada:** Se uma equipe utiliza técnicas como TDD desde o início do projeto, o fator de testes será alto (em muitos casos inclusive com valor acima de 1). Para equipes onde os testes não são desenvolvidos junto com o código de produção, o fator de testes será provavelmente baixo e a tendência é de melhoria;

• **Quando utilizar?** Quando a equipe estiver preocupada com a Qualidade do software produzido, ou estiver monitorando a melhoria de práticas como o Desenvolvimento Dirigido por Testes, a Integração Contínua, o Design Incremental ou a Refatoração;

• **Quando parar de utilizar?** é importante que a equipe tenha conhecimento da Qualidade do software produzido. Caso decidam parar de utilizar o Fator de Teste, é importante substituí-lo por outra métrica para avaliar a Qualidade e mantê-la alta;

• **Formas de Manipulação:** Essa métrica pode ser manipulada

através da produção de muito código de teste que não verifica o comportamento esperado do sistema (testes sem assert, por exemplo). Da mesma forma, medir linhas de código sempre pode gerar um incentivo para tornar o código menos legível, evitando quebras de linha que afetariam essa métrica;

• **Cuidados e Observações:** Um fator de teste maior que 1 não representa necessariamente o cenário ideal, assim como não garante que o código está totalmente testado ou com boa Qualidade. Apesar de influenciar a Qualidade do software produzido, essa métrica não serve como medida objetiva de qualidade.

Considerações Finais

Neste artigo discutiu-se o papel das métricas nos Métodos Ágeis de desenvolvimento de software. Foram apresentadas as definições de medidas, métricas e indicadores, assim como a forma que estes conceitos se relacionam na análise, contextualização e interpretação de uma métrica. Diferentes formas de classificação foram discutidas, diferenciando métricas de natureza objetiva/subjetiva, quantitativa/qualitativa e organizacional/acompanhamento.

Essa última classificação tem papel particularmente importante neste artigo, separando as métricas de acordo com diferentes objetivos. Enquanto uma métrica organizacional avalia o desempenho geral, medindo a quantidade de valor de negócio entregue ao cliente, uma métrica de acompanhamento tem natureza simplesmente informativa, provendo informações locais que ajudam a equipe no entendimento e melhoria do processo que produz valor de negócio. Tais definições, conceitos e formas de classificação ajudam a formar uma base para discussão do papel das métricas nos Métodos Ágeis.

Além disso, discutiu-se também três diferentes abordagens para definição e escolha de uma métrica. A combinação da abordagem GQM – que esclarece o objetivo por trás de cada medição – com a abordagem Lean – que estimula um menor número de métricas organizacionais e promove a medição em um nível acima – fornece uma boa base para escolha das métricas organizacionais. Por outro lado, uma terceira abordagem auxilia a equipe a refletir e discutir os pontos do processo que podem ser melhorados, servindo como base para escolha das métricas de acompanhamento.

Por fim, foram apresentadas as características de uma boa métrica ágil, juntamente com uma lista de verificação que pode ser usada pelo tracker ao avaliar a possibilidade de utilização de uma métrica. Seguindo os tópicos propostos nessa lista de verificação, diversos exemplos de medidas, métricas organizacionais e métricas de acompanhamento foram apresentados, concluindo este artigo.

Referências

1. Eliyahu M. Goldratt. The Haystack Syndrome: Sifting Information Out of the Data Ocean. North River Press, 1991.

2. Ron Jeffries, Ann Anderson, and Chet Hendrickson. Extreme Programming Installed. Addison-Wesley Professional, 2000.

3. Ken Auer and Roy Miller. Extreme Programming Applied: Playing to Win. Addison-Wesley Professional, 2001.

4. Alistair Cockburn. Agile Software Development: The Cooperative Game. Addison-Wesley Professional, 2nd edition, 2006.

5. IEEE standard glossary of software engineering terminology, 1983. IEEE Std 729.

6. John McGarry, David Card, Cheryl Jones, Beth Layman, Elizabeth Clark, Joseph Dean, and Fred Hall. Practical Software Measurement: Objective Information for Decision Makers. Addison-Wesley Professional, 2002.

7. IEEE standard glossary of software engineering terminology, 1990. IEEE Std 610.12.

8. Jane F. Gilgun. Definitions, methodologies, and methods in qualitative family research. Qualitative Methods in Family Research, 1992.

9. Carolyn B. Seaman. Qualitative methods in empirical studies of software engineering. IEEE Transactions on Software Engineering, 25(4):557–572, 1999.

10. Deborah Hartmann and Robin Dymond. Appropriate agile measurements: Using metrics and diagnostics to deliver business value. In Agile 2006 Conference, pages 126–131, 2006.

11. Jim Collins. Good to Great: Why Some Companies Make the Leap... and Others Don't. Collins, 2001.

12. Mary Poppendieck and Tom Poppendieck. Lean Software Development: An Agile Toolkit for Software Development Managers. Addison-Wesley Professional, 2003.

13. Vitor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The goal question metric approach. Encyclopedia of Software Engineering, pages 528–532, 1996.

14. Robert D. Austin. Measuring and Managing Performance in Organizations. Dorset House Publishing Company, Inc., 1996.

15. Mary Poppendieck and Tom Poppendieck. Implementing Lean Software Development: From Concept to Cash. Addison-Wesley Professional, 2006.

16. W. Edwards Deming. Out of The Crisis. Massachusetts Institute of Technology, 1986.

17. Yael Dubinsky, David Talby, Orit Hazzan, and Arie Keren. Agile metrics at the israeli air force. In Agile 2005 Conference, pages 12–19, 2005.

18. Kent Beck and Martin Fowler. Planning Extreme Programming. Addison-Wesley Professional, 2000.

19. Thomas J. McCabe and Arthur H. Watson. Software complexity. Crosstalk: Journal of Defense Software Engineering, 7:5–9, 1994.

20. Arthur H. Watson and Thomas J. McCabe. Structured testing: A testing methodology using the cyclomatic complexity metric. Technical report, NIST Special Publication 500-235, 1996. 54

21. Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. IEEE Transactions on Software Engineering, 20(6):476–493, 1994.

22. Brian Henderson-Sellers. Object-Oriented Metrics: Measures of Complexity. Prentice Hall PTR, 1996.

23. Barry W. Boehm and Richard Turner. Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley Professional, 2003.

24. Ron Jeffries. A metric leading to agility. Disponível em: www.xprogramming.com/xpmag/jatRtsMetric.htm, 2004. Acessado em: 31/05/2007.

25. Kent Beck and Cynthia Andres. Extreme Programming Explained : Embrace Change. Addison-Wesley Professional, 2nd edition, 2004.

26. Fred Reichheld. The Ultimate Question: Driving Good Profits and True Growth. Harvard Business School Press, 2006.

27. Ken Schwaber and Mike Beedle. Agile Software Development with Scrum. Prentice Hall, 2001.

UML – Diagrama de Classes

Encontrando classes e desenhando seu diagrama – Parte I

De que se trata o artigo?
Este artigo tem por objetivo apresentar as regras para se modelar um diagrama de classes, partindo da análise prática dos requisitos de um estudo de caso.

Para que serve?
Fornecer aos desenvolvedores ou estudantes da área de sistemas uma linha de entendimento com o intuito de orientá-los a modelar seus diagramas de classes.

Em que situação o tema é útil?
Para quem ainda não modelou classes, ou para quem tem experiência e quer revisar a sintaxe permitida nesse tipo de diagrama.

Ana Cristina Melo
informatica@anacristinamelo.com.br
É especialista em Análise de Sistemas e professora de graduação e pós-graduação da Universidade Estácio de Sá. Atua em análise e programação há 21 anos. Autora do livro "Desenvolvendo aplicações com UML - do conceitual à implementação", na segunda edição, e "Exercitando modelagem em UML". Palestrante em alguns eventos, entre eles, Congresso Fenasoft, OD e Sepai.

Neste artigo, faremos uso de um pequeno estudo de caso para entendermos como é feita a modelagem de um diagrama de classes, aproveitando esse passo a passo para a apresentação das regras desse diagrama.

Nessa primeira parte mostraremos como extrair classes de uma lista de requisitos ou de um caso de uso; demonstraremos as notações para representarmos uma classe, seus atributos e operações, e falaremos do relacionamento de associação.

No próximo artigo, veremos como refinar um diagrama de classes, apresentando informações relevantes como escopo, restrições, os relacionamentos de generalização e agregação, além de algumas classes especiais como classe de enumeração, abstrata e de associação.

Encontrando as classes

A tarefa de encontrar classes entre um conjunto de requisitos requer questionamento e investigação. O caminho mais usual para essa tarefa é a de se encontrar as classes a partir dos cenários de casos de uso. Contudo, podemos partir também dos requisitos ainda brutos (não modelados em casos de uso), para obter uma primeira versão das classes que o sistema

acolherá. Normalmente, essa segunda opção é realizada em sistemas de maior porte para se obter uma idéia macro de sua dimensão, antes que o trabalho duro tenha início; ou em sistemas de menor porte, para se ter a idéia precisa de seu tamanho.

Em virtude de espaço, não iremos relacionar aqui todos os cenários dos casos de uso de nosso estudo de caso para extrairmos as classes a partir deles. Mas aproveitando a simplicidade de nosso exemplo, faremos a lista de classes a partir do levantamento de requisitos apresentado no **Quadro 1**.

Uma técnica simples para se encontrar uma classe é identificar os substantivos de uma lista de requisitos. Contudo, não se pode esquecer de analisar também as frases como um todo, pois algumas podem representar a conceituação de alguma classe candidata que não tenha sido explicitamente nominada nos requisitos.

A partir desta técnica extraímos dos requisitos do **Quadro 1** os substantivos, chegando a uma primeira lista de candidatos a classes disponibilizada no **Quadro 2**.

Seguindo a regra de encontrar os substantivos, obtivemos uma lista extensa. Mas é preciso identificarmos o que seja um conceito (abstração), um objeto existente a partir dessa abstração, um atributo ou até mesmo o conteúdo desse atributo. Além disso, é comum encontrarmos sinônimos para um mesmo conceito. Quando isso ocorre, se tivermos conhecimentos suficientes da área de negócios modelada, basta eliminarmos um dos termos; caso contrário, precisaremos da ajuda do cliente para chegarmos a uma decisão.

Desta forma, identificamos, por exemplo, que Dr. Monteiro nada mais é do que a instância de uma abstração de Médico (*classe Medico*) ou até mesmo o conteúdo do *atributo nome* da *classe Medico*. E que nome da criança nada mais é do que atributo de uma *classe Paciente*. Também podemos associar, neste contexto, como sinônimos: Médico e Pediatra; Cliente e Paciente; Operadora de Plano e Plano de Saúde.

Normalmente um atributo é um tipo de dado primitivo, como por exemplo: números, strings, valores booleanos. Se um atributo se mostra com necessidades de desdobramentos, como por exemplo: Endereço completo (logradouro, número, complemento, bairro, cidade, uf), estamos diante do relacionamento de uma classe com outra classe. Veja que não podemos colocar numa *classe Paciente* os atributos *logradouro* e *número*, pois quem tem logradouro e número é o endereço de um paciente e não o paciente. Repare na expressão "endereço de um paciente". Essa expressão já nos dá a dica da notação de ponto, usada na orientação a objetos, com intuito de navegar de uma classe para outra. Nesse caso, a navegação de paciente para endereço (Paciente.Endereco.Logradouro).

Sistema de Consultas Médicas

Dr. Monteiro contratou uma empresa para informatizar seu consultório.

Dr. Monteiro é pediatra e atende através de plano de saúde ou particular. As consultas na agenda serão marcadas pela secretária, e ocorrem de meia em meia hora, em horários distintos a cada dia da semana. Só são permitidos dois encaixes por dia. Caso o paciente seja novo, deve-se registrar apenas o nome da criança, do responsável, telefone de contato e tipo de plano.

Para cada paciente é preciso manter: nome da criança, nome dos pais, data de nascimento, endereço completo (incluindo uf), telefone residencial e celular (indicando a quem pertence), sexo, prontuário de cada consulta, histórico de peso e altura. O receituário deve ser emitido pelo sistema com as prescrições, nome e CRM do médico.

O prontuário de cada consulta deve conter a descrição dos sintomas, resultado da observação clínica, os medicamentos prescritos (com dosagem, administração e tempo de uso) e exames pedidos.

Associado a cada cliente deve ser registrado o nome de seu plano de saúde ou se o mesmo é particular.

Quadro 1. Requisitos do Estudo de Caso Sistema de Consultas Médicas

- Dr. Monteiro
- Empresa
- Consultório
- Pediatra
- Plano de Saúde
- Particular
- Consultas
- Agenda
- Secretária
- Duração das consultas (refere-se a: ocorrência de meia em meia hora)
- Horários
- Dia da semana
- Encaixe
- Limite de encaixes (refere-se a: dois encaixes por dia)
- Paciente
- Nome da criança
- Nome do responsável
- Telefone de contato
- Tipo de plano
- Nome dos pais
- Data de nascimento
- Endereço completo
- Telefone residencial
- Telefone celular
- Sexo
- Prontuário
- Histórico de peso
- Histórico de altura
- Receituário
- Sistema
- Prescrição
- Nome do médico
- Descrição dos sintomas
- Resultado da observação clínica
- Medicamentos prescritos
- Dosagem
- Administração
- Tempo de uso
- Exames pedidos
- Cliente
- Nome do plano

Quadro 2. Lista de candidatos à classe

Lista de classes (atributos)
Médico (nome, CRM)
Plano de Saúde (nome)
Consulta (data, hora, encaixe)
Agenda (dia da semana, horário inicial, horário final, duração da consulta, limite de encaixes)
Paciente (nome da criança, nome dos pais, data de nascimento, sexo)
Endereço (logradouro, número, complemento, bairro, cidade, uf, cep)
Telefone (número, ddd, tipo do telefone, complemento)
Prontuário (peso, altura, descrição dos sintomas, observação clínica)
Prescrição (dosagem, administração, tempo de uso)
Medicamento (nome)
Exame (nome)

Quadro 3. Lista refinada de candidatos à classe

Algumas decisões já podem ser tomadas nesse refino. Note que o requisito determina que se tenha um histórico de peso e altura. Contudo, essas informações são registradas a cada prontuário, ou seja, a cada consulta. Isso significa que se recuperarmos essas informações de todos os prontuários, estaremos consultando o histórico de peso e altura. Desta forma, o histórico, nesse caso, não é uma classe, mas uma operação sobre a *classe Prontuario*.

Seguindo a mesma lógica, percebemos que para que o médico possa selecionar os medicamentos e exames para prescrever, será preciso um cadastro dos mesmos. Assim, exames pedidos significam o relacionamento do prontuário com os exames selecionados. E medicamentos prescritos significam o relacionamento do prontuário com os medicamentos selecionados. Mas para cada medicamento, será preciso informar a dosagem, a forma de administração e o tempo de uso. Sendo assim, além de uma *classe Medicamento*, surge nesse refino uma *classe Prescrição*.

Outros ajustes já podem ser feitos nessa etapa. Quando horário de atendimento é citado, estamos na realidade nos referindo a um horário inicial e um horário final. Esses atributos precisam ser modelados de forma separada. No caso do endereço completo, vemos que esse é um candidato à classe, e seus atributos serão aqueles que determinarão a completude desse endereço, como logradouro, número, complemento, etc. Ao chegarmos no telefone, percebemos que uma única classe pode conter todos os tipos de telefone, tomando cuidado para cadastrar informações relevantes como ddd e complemento (por exemplo: informação de ser um telefone de recado, ou indicação a quem pertence, ou até um ramal de um telefone comercial).

Partindo desse entendimento, compreendemos melhor algumas classes e podemos acrescentar atributos em outras. Então, arrastamos os candidatos a atributos para suas respectivas classes e chegamos à relação do **Quadro 3**. Perceba que embora tenhamos utilizado uma notação parecida com aquela para representar tabelas em modelo relacional, esse não é o objetivo em representar a lista desta forma. Essa notação foi utilizada apenas para facilitar a associação das classes e seus atributos.

Uma outra forma de se chegar a essa lista é relacionar esses candidatos à classe e seus atributos a partir dos cenários de um caso de uso. Usaremos a **Listagem 1** para refinar as classes encontradas a partir do cenário principal do caso de uso *Registrar prontuário*.

Listagem 1. UC Registrar Prontuário

```
Descrição: Este caso de uso tem por objetivo permitir ao
médico registrar todas as informações da consulta no prontuário
do paciente, bem como prescrever os medicamentos e exames
necessários, solicitando ao final a geração do receituário.

Atores: Médico
Pré-condição: existir cadastro prévio de medicamentos e exames.
Cenário principal:

1 - O sistema exibe todas as consultas agendadas para o dia, com
as seguintes informações para cada uma:

  - horário
  - se é encaixe
  - nome da criança
  - se é paciente novo

2 - O usuário seleciona uma consulta.

3 - O sistema exibe o prontuário para o usuário, com as
seguintes opções habilitadas:

  3.1 - Consultar últimos lançamentos do prontuário (Extends
        Caso de Uso Consultar Histórico do Prontuário)
  3.2 - Consultar histórico de peso (Extends Caso de Uso
        Consultar Histórico de Peso)
  3.3 - Consultar histórico de altura (Extends Caso de Uso
        Consultar Histórico de Altura)

4 - O sistema prepara uma lista de medicamentos cadastrados.

5 - O sistema prepara uma lista de exames cadastrados.

6 - O usuário informa no prontuário, em qualquer ordem:

  6.1 - peso do paciente
  6.2 - altura do paciente
  6.3 - descrição dos sintomas
  6.4 - resultado da observação clínica
  6.5 - para cada medicamento prescrito, o usuário informa:
    6.5.1 - nome do medicamento, selecionado da lista pré-
            existente
    6.5.2 - dosagem
    6.5.3 - administração
    6.5.4 - tempo de uso
  6.6 - para cada exame prescrito, o usuário informa:
    6.6.1 - nome do exame, selecionado da lista pré-existente

7 - O sistema habilita a opção Imprimir receituário (Extends
    Caso de Uso Imprimir Receituário)

8 - O sistema atualiza o cadastro do prontuário.

Cenários alternativos:
(...)
```

Confirmamos a partir desse caso de uso que será preciso ter uma classe que persista todos os medicamentos, e uma outra que será, na realidade, um relacionamento desses medicamentos com o prontuário. Já no caso dos exames, basta associá-los ao prontuário, pois esse relacionamento não requer atributos próprios. Quanto ao histórico do prontuário, devemos enxergar de forma diferente. Um prontuário (considerando a imagem daquelas fichas em pastas) é o conjunto de tudo que é registrado a cada consulta. Assim, podemos entender que cada consulta gera um prontuário com um conjunto de informações. Desta forma, o histórico do prontuário nada mais é do que o conjunto de todos esses prontuários, que pode ser consultado a partir de uma operação pertencente à classe.

Desenhando as primeiras classes

Essas classes foram identificadas no formato de lista, mas também já podemos representá-la graficamente. Uma das notações para representar uma classe é a descrita na **Figura 1.** Essa notação se restringe a um único compartimento, contendo o nome da classe. Desenha-se um retângulo e dentro dele, de forma centralizada e em negrito, coloca-se o nome da classe.

Figura 1. Representação de uma classe somente com o compartimento do nome

Não basta apenas escrever o nome da classe num diagrama de classes; é preciso seguir a notação determinada pela UML, que consiste de:

• o nome da classe é escrito centralizado e em negrito;

• escrever as iniciais dos nomes das classes em maiúsculas, inclusive as primeiras letras de nomes compostos (ex: Paciente; PlanoSaude);

• Não se usa acentuação nem espaço no nome das classes. Apesar de ser permitido o uso do caractere *underline* (sublinhado), o mesmo não é usual, visto que a marcação de maiúscula na inicial de cada palavra composta já determina visualmente essa separação (ex: PlanoSaude).

Após chegarmos a essa versão (ainda não definitiva) da relação de classes, devemos identificar os relacionamentos existentes entre as mesmas. O relacionamento mais comum entre classes é o relacionamento de ***associação***, representado por uma linha contínua ligando os dois lados. Nas extremidades dessa associação colocamos *multiplicidades*. Essas multiplicidades indicam a quantidade de instâncias de uma classe que se relacionam a uma única instância de outra classe. São exemplos de multiplicidades:

```
0..1 (valor opcional)
1 ou 1..1 (exatamente um)
* ou 0..* (qualquer valor maior ou igual a zero)
1..* (qualquer valor maior que zero)
2..4 (no mínimo 2, no máximo 4)
```

Então, se dissermos que um prontuário pode vir a ter associado a ele nenhuma ou várias prescrições de um medicamento, e que cada uma dessas prescrições pode ficar registrada para ser associada a outros pacientes, então teremos o seguinte:

```
Um prontuário gera 0..* (nenhuma ou várias) prescrições.
Uma prescrição está associada a 1..* (pelo menos uma ou várias)
prescrições.
```

A representação gráfica mais completa de uma classe acrescenta dois compartimentos contendo a lista de atributos e operações. No primeiro compartimento também são previstas outras informações. A primeira linha é reservada ao ***estereótipo***. Um estereótipo é um mecanismo de extensibilidade da UML que representa uma subclasse de um elemento já existente, com o mesmo formato, porém com objetivos diferentes e bem definidos. O estereótipo sempre aparece entre *guillemets* («»). No exemplo da **Figura 2**, o estereótipo *utility* indica que essa classe é do tipo utilitária, não especificamente uma classe de negócios, mas uma classe que oferece recursos úteis a várias outras.

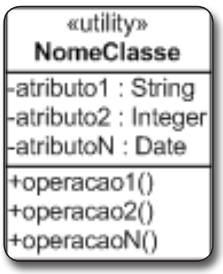


Figura 2. Representação de uma classe com todos os compartimentos

A UML oferece uma notação específica para os atributos e operações também. Veja abaixo:

• os atributos e operações devem ser escritos com formatação normal e alinhados à esquerda;

• os nomes de atributos e operações devem iniciar com letra minúscula, entretanto as iniciais das palavras compostas seguintes devem iniciar com letra maiúscula.

Um atributo pode ser expresso usando-se a seguinte sintaxe:

```
visibilidade / nome : tipo [ multiplicidade ] = valor-default {
string-propriedade }
```

Uma operação pode ser expressa usando-se a seguinte sintaxe:

```
visibilidade nome (lista-de-parâmetros): tipo-de-retorno
{ string-propriedade }
```

A primeira característica que aparece em comum para os atributos e operações é a ***visibilidade***. A visibilidade identifica por quem um atributo ou operação pode ser acessado. Divide-se em quatro tipos:

• público (*public*) ou +

• privado (*private*) ou -

• protegido (*protected*) ou #

• pacote (*package*) ou ~

A ***visibilidade pública*** permite que o atributo ou operação seja acessado dentro da classe em que foi declarado, nas classes descendentes ou por qualquer elemento externo, incluindo-se as instâncias.

A ***visibilidade privada*** é a mais restrita que existe. O atributo ou operação só poderá ser acessado dentro da classe em que foi declarado. Isso significa que ele será invisível para as classes filhas e para as instâncias.

A ***visibilidade protegida*** determina que o atributo ou operação somente poderá ser visto dentro da classe em que foi declarado ou nas classes descendentes. As instâncias não acessam (enxergam) esse elemento.

A ***visibilidade pacote*** determina que o atributo ou operação poderá ser visto dentro da classe em que foi declarado, nas

classes descendentes ou por qualquer elemento externo, desde que estejam todos dentro do mesmo pacote da classe que declarou o elemento.

Exemplo:

```
# nome : string
+ getNome() : string
+ setNome (nome: string)
```

A ***barra (/)*** indica que o atributo é derivado. Isso significa que o atributo derivado não persiste (guarda) as informações, mas as recupera de algum processamento. Por exemplo: um atributo média não pode ser informado diretamente, mas sim calculado a partir de um conjunto de notas.

Exemplo:

```
# / media : float
```

O ***nome*** de um atributo ou operação deve seguir as regras já descritas.

Para um atributo, ***tipo*** significa o tipo de dados desse atributo (ex: string, integer, float).

Exemplo:

```
# nome : string
# numero : integer
```

Em um atributo podemos definir a ***multiplicidade*** que determina a existência de mais de uma ocorrência para o mesmo atributo. Por exemplo: se temos uma classe *Triangulo* na qual precisamos definir os lados, mais correto do que modelar *lado1*, *lado2* e *lado3*, é modelarmos da seguinte forma:

```
# lado : float [3..3]
```

Se ao modelarmos um atributo, já imaginarmos um *valor default* para o mesmo, devemos especificar essa informação na sua definição. Por exemplo: suponha que na classe *Operacao*, tenhamos os atributos *termos* que indicam os termos que farão parte da operação e o atributo operação que indicará especificamente que tipo de operação faremos (adição, subtração, multiplicação ou divisão). Para que não aconteça de termos uma divisão por zero, por descuido de quem for definir os termos da conta, podemos inicializar esses termos com o valor 1 (um). Por exemplo:

```
# termo : float [2..*] = 1
```

A ***string-propriedade*** é uma propriedade que permite definirmos informações especiais, do nível de restrições, tanto para os atributos quanto para as operações. Um exemplo de *string-propriedade* para um atributo é o de *redefines*. O *redefines* indica que um atributo está redefinindo algum outro atributo, dentro de um

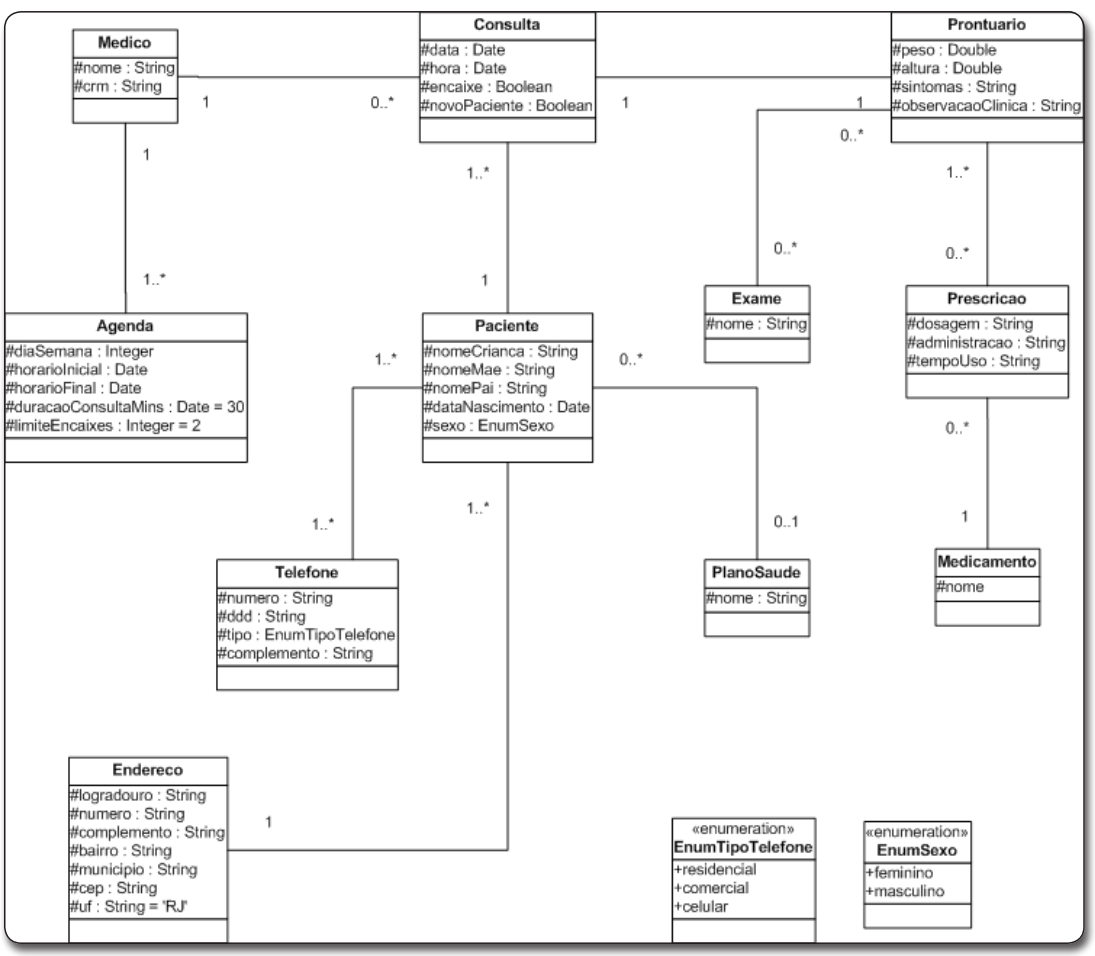


Figura 3. Primeira versão do diagrama de classes do estudo de caso do Sistema de Consultas Médicas

relacionamento de generalização. Imagine que temos um atributo *nome* na classe *Pessoa*. Ao herdarmos dela a classe *PessoaJuridica* seria natural pensarmos esse nome como sendo razão social. Desta forma, a definição desse atributo ficaria assim:

```
# razaoSocial : string {redefines nome}
```

Já no caso de uma operação, um exemplo é definir a operação como abstrata, o que significa que essa operação não terá implementação na classe que a definiu, apenas nas classes descendentes. Por exemplo, suponha a operação *calcularArea* da classe *FiguraGeometrica*. Não tem sentido definirmos um método para essa operação, ou seja, escrevermos sua implementação, pois esta depende do tipo de figura geométrica. Sendo assim, essa implementação só apareceria na classe *Retangulo*, classe-filha de *FiguraGeometrica*.

```
+ calcularArea : float {abstract}
```

Com essas informações, já podemos partir para a modelagem do diagrama de classes a respeito do estudo de caso, que trata do sistema de consultas médicas. Veja a **Figura 3**.

Vamos entender essa primeira versão do diagrama:

A *classe Medico* terá uma única instância que será o Dr. Monteiro. Cada agenda se relaciona apenas a um médico, que nesse caso será o próprio Dr. Monteiro. Contudo, nada impede em nossa modelagem que futuramente haja outro médico dividindo o consultório com ele. A instância dessa classe será usada também na impressão do receituário.

Um médico possui uma ou mais instâncias da *classe Agenda*. Isso significa dizer que o médico poderá clinicar em vários dias da semana, estabelecendo um quadro de horário (agenda) para cada dia.

Ao se marcar uma consulta, o sistema irá buscar a agenda para saber quais horários serão usados na marcação. Não há um relacionamento entre a *classe Consulta* e a *classe Agenda*, pois esta última funciona como um quadro de horários. A partir do dia e horário livre, será preciso identificar o médico e o paciente e associá-los à consulta. Assim, qualquer instância da *classe Consulta* está associada a uma única instância da *classe Medico* e uma única instância da *classe Paciente*. Ao marcar uma consulta, se o paciente for novo, somente alguns atributos serão preenchidos, e o atributo *novoPaciente* receberá o valor *true*. Se a consulta for um encaixe, o atributo *encaixe* também receberá o valor *true*.

Um paciente além de seus atributos simples (por exemplo: *nomeCrianca* e *dataNascimento*) também possui um endereço e vários telefones. Só que o endereço é uma classe. Desta forma, temos um relacionamento das classes *Paciente, Endereço e Telefone*. Assim, uma instância da *classe Paciente* se relaciona com apenas uma instância da *classe Endereco*. Contudo, considerando que o médico pode atender irmãos, a mesma instância de *Endereco* pode ser aproveitada para mais de um paciente. O mesmo acontecerá com a *classe Telefone*, sendo que um paciente deve ter no mínimo um telefone, podendo cadastrar vários. Uma instância de *Paciente* também está associada a uma única instância da *classe PlanoSaude*, podendo não estar associada a nenhuma instância, o que significaria que esse paciente é particular.

Uma consulta na sua data gera um único prontuário, o qual é preenchido pelo médico. Da mesma forma, esse prontuário pertence a uma única consulta. Cada instância da *classe Prontuario* pode gerar a prescrição de nenhum ou vários exames, o que indica seu relacionamento com a *classe Exame*. Além disso, pode gerar a prescrição de nenhum ou vários medicamentos. Como para cada medicamento prescrito deve-se determinar a dosagem, administração e tempo de uso, surge no centro desse relacionamento a *classe Prescricao*.

Repare que à frente de todos os atributos aparece a visibilidade *Protected* (protegido). De diferente nesse modelo, temos alguns tipos de dados que não são os tipos primitivos, como inteiro (*integer*), float (*double*), booleano (*boolean*), data e hora (*date*) e string. Esses tipos aparecem no atributo *sexo* da *classe Paciente* e no atributo *tipo* da *classe Telefone*. Esses atributos são conhecidos como tipo enumerados.

Mas essa será uma explicação para o próximo artigo, quando refinaremos mais ainda esse modelo e apresentaremos alguns recursos avançados de um diagrama de classes. ●

Referências

MELO, Ana Cristina. Desenvolvendo aplicações com UML 2.0: do conceitual à implementação. Rio de Janeiro: Brasport, 2004.

Utilização do Controle Estatístico de Processos na Melhoria de Processos de Software

Conhecendo Ferramentas para Análise do Comportamento dos Processos

De que se trata o artigo?

Este artigo está diretamente relacionado ao artigo "Controle Estatístico de Processos de Software: Do "Chão de Fábrica" para as Organizações de Software", publicado na edição anterior. No artigo citado foram apresentados os principais conceitos e questões relacionados ao controle estatístico de processos e sua utilização em processos de software. Foram abordados os conceitos de estabilidade, causas de variações (comuns e especiais), capacidade e uma breve introdução às técnicas estatísticas utilizadas no controle estatístico de processos foi realizada. Neste artigo, em complemento ao que foi discutido no artigo anterior, é realizada uma contextualização da utilização do controle estatístico de processos na melhoria de processos de software, as principais técnicas estatísticas são apresentadas e é discutido um exemplo de utilização de gráficos de controle para analisar o comportamento (estabilidade) de um processo, e um exemplo de utilização de histograma para analisar a capacidade de um processo.

Para que serve?

Incrementar o conhecimento essencial sobre a utilização do controle estatístico de processos para organizações, pesquisadores, estudantes e profissionais de software envolvidos em atividades relacionadas aos níveis gerencial e/ou estratégico das organizações.

Em que situação o tema é útil?

Organizações que estão a caminho dos níveis mais elevados de maturidade de seus processos e precisam implantar o controle estatístico de processos; organizações que estão iniciando o caminho de melhoria de processos e desejam, desde já, conhecer aspectos de controle estatístico de processos necessários no futuro, nos níveis mais elevados; profissionais envolvidos na implementação de programas de melhoria de processos de software; estudantes e pesquisadores que desejam explorar o tema.

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.

Constantemente as organizações se questionam se os resultados por elas desejados estão sendo alcançados. Essa questão pode ser expressa de várias maneiras: se os objetivos estão sendo atingidos, se os clientes estão satisfeitos, se está havendo retorno do investimento, se o desempenho alcançado é suficiente para o crescimento requerido para a sobrevivência e a competitividade necessários, entre outras.

Para a maioria das organizações, responder a essas questões de forma objetiva e precisa não é uma tarefa trivial, pois nem sempre seus processos são gerenciados e, com isso, seu desempenho real não é conhecido.

W. Eduards Deming, reconhecendo a importância da gerência dos processos, baseou-se nos trabalhos de Walter A. Shewart para definir uma abordagem para gerência de processos e melhoria contínua que, inicialmente, foi implementada com sucesso em indústrias do Japão. A abordagem foi popularizada como PDCA – *Plan*, *Do*, *Check*, *Act*.

Com o passar do tempo, os conceitos, métodos e práticas associados à gerência de processos e melhoria contínua foram aceitos pela comunidade de software, que percebeu que os princípios do *process-thinking* poderiam ser aplicados em organizações de software a fim de alcançar melhoria da qualidade, produtividade e competitividade.

Gerenciar processos de software envolve definir, medir, controlar e melhorar os processos de trabalho associados ao desenvolvimento, manutenção e suporte de produtos e serviços de software. A gerência efetiva dos processos corrobora para que os produtos e serviços gerados atendam aos requisitos da organização e dos clientes e, consequentemente, contribuam para o alcance dos objetivos de negócio da organização.

A aceitação e utilização do *process-thinking* pelas organizações de software levou à percepção de que os métodos de medição e análise tradicionais, que comparam os valores realizados com os valores planejados, não são suficientes para determinar o desempenho dos processos e responder objetivamente às questões citadas anteriormente.

Mais uma vez, a comunidade de software reconheceu a utilidade de métodos e práticas oriundos da indústria para resolver questões nas organizações de software. O controle estatístico de processos, que envolve gráficos de controle e métodos estatísticos, passou, então, das linhas de produção da manufatura para o desenvolvimento de software.

A utilização de gráficos de controle e métodos estatísticos provê aos engenheiros de software e gerentes uma visão quantitativa do comportamento de seus processos de software. Metaforicamente, os gráficos de controle estão para os gerentes e engenheiros de software assim como um medidor de temperatura está para um profissional que cuida do processo de transformação da matéria-prima em aço e que precisa realizar determinadas ações de acordo com a temperatura atingida pelos materiais ao longo do processo. Ou seja, os gráficos de controle guiam as decisões dos gerentes e engenheiros de software fornecendo o conhecimento necessário sobre o comportamento dos processos.

Gerência de Processos de Software e Controle Estatístico de Processos

Conforme mencionado anteriormente, a gerência de processos de software envolve definir, medir, controlar e melhorar os processos associados ao desenvolvimento, manutenção e suporte de produtos e serviços de software, a fim de que os mesmos contribuam para o alcance dos objetivos organizacionais.

Segundo princípios do controle estatístico de processos, a gerência de processos permite estabilizar e manter estáveis os níveis de variação do comportamento dos processos, possibilitando que resultados futuros sejam previstos, colaborando, assim, para um melhor alinhamento entre os objetivos e expectativas organizacionais e o desempenho dos processos.

Sendo assim, as responsabilidades da gerência de processos, considerando-se os princípios do controle estatístico de processos, podem ser descritas da seguinte forma:

a) Definir o processo

Consiste em definir a 'base' requerida para o controle e melhoria dos processos. Ou seja, envolve a definição dos processos que apóiam os objetivos técnicos e de negócio da organização; identificação das necessidades de informação e medidas relacionadas ao desempenho dos processos; e, fornecimento da infra-estrutura e habilidades necessárias para execução dos processos (métodos, pessoas e práticas).

b) Medir o processo

Consiste na coleta, armazenamento, análise e utilização de dados das medidas de desempenho para os processos da organização. Esses dados servem para determinar a estabilidade e capacidade dos processos, apoiar previsões, determinar *baselines* e identificar tendências e oportunidades de melhoria.

c) Controlar o processo

Consiste em identificar se o comportamento dos processos é estável e, caso não seja, identificar e tratar as causas de instabilidade.

d) Melhorar o processo

Consiste em identificar aspectos que interferem na capacidade dos processos, planejar e realizar ações para tratar esses aspectos, modificando os processos a fim de melhorar sua capacidade.

Melhoria de Processos de Software e Controle Estatístico de Processos

Todos os processos são executados para produzir resultados. A percepção de que um processo deve (ou pode) ser melhorado tem início quando os resultados por ele alcançados não atendem aos objetivos para ele estabelecidos.

A melhoria de processos de software utilizando controle estatístico de processos requer a análise contínua dos seguintes aspectos:

a) Desempenho: para melhorar um processo é preciso conhecer seu desempenho, ou seja, é preciso saber o que o processo está produzindo no momento em relação a medidas de atributos de qualidade, quantidade, custo e tempo.

b) Estabilidade: conhecido o desempenho, é preciso analisar se o mesmo é previsível, ou seja, se o processo é estável e, se não for, é preciso melhorá-lo a fim de que estabilize. Um processo

estável é aquele cujas variações encontram-se dentro de limites considerados aceitáveis para seu comportamento. As causas dessas variações são intrínsecas ao processo e são chamadas de causas comuns. Um processo instável possui comportamento com variações que excedem os limites considerados aceitáveis para seu comportamento. As causas dessas variações, chamadas de causas especiais, devem ser identificadas e tratadas.

c) Capacidade: um processo estável não necessariamente é capaz de alcançar os objetivos para ele determinados. Logo, uma vez estabilizado um processo, sua capacidade deve ser analisada. Processos não capazes devem ser modificados através da realização de ações de melhoria que apóiem o alcance aos objetivos de negócio da organização.

Uma vez que um processo é caracterizado como capaz, o mesmo pode receber novos objetivos a serem alcançados, o que leva à necessidade de modificar o processo para melhorar sua capacidade, impulsionando um ciclo de *melhoria contínua*.

Uma visão de um ciclo de melhoria contínua baseada nos princípios do controle estatístico de processos, ou seja, na análise do comportamento dos processos, é apresentada na **Figura 1**. Este ciclo foi proposto por FLORAC e CARLETON (1999).

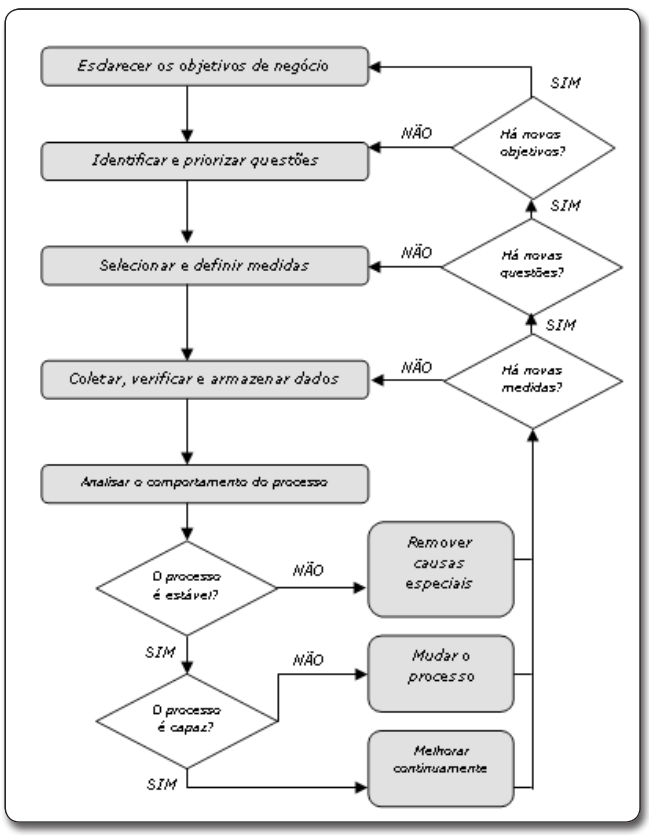


Figura 1. Ciclo para melhoria de processos (FLORAC e CARLETON, 1999)

A seguir, suas etapas são sucintamente descritas.

a) Esclarecer os objetivos de negócio: consiste em entender como os objetivos de negócio, metas, planos e estratégias da organização se relacionam com os processos de software. Ou seja, consiste na identificação das relações entre processos e objetivos organizacionais.

b) Identificar e priorizar questões: consiste em identificar quais são as necessidades de informação críticas para determinar se o processo será capaz de alcançar os objetivos de negócio estabelecidos.

c) Selecionar e definir medidas: consiste em estabelecer e definir as medidas que serão utilizadas para caracterizar os processos e produtos.

d) Coletar, verificar e armazenar dados: consiste em coletar os dados das medidas definidas e investigar se há causas especiais de variação e melhorias potenciais. Os dados devem estar organizados de maneira a facilitar o reconhecimento de padrões, tendências e relacionamentos.

e) Analisar o comportamento dos processos: consiste em utilizar técnicas e métodos estatísticos apropriados para representar em gráficos de controle as medidas coletadas, permitindo, assim, a análise do comportamento do processo.

f) Realizar ações corretivas e/ou de melhoria: consiste em realizar as ações necessárias, considerando o resultado da análise do comportamento do processo. Esse resultado pode levar a três direções: *remover as causas especiais* para tornar o processo estável, *mudar o processo* para que o mesmo seja capaz de atender os objetivos do cliente e da organização, ou *melhorar continuamente* o processo, quando este é capaz.

Ferramentas Analíticas

Para que o comportamento dos processos seja analisado, os dados coletados para as medidas relacionadas aos processos devem ser representados. Para apoiar a representação e análise dos dados de comportamento dos processos, existem diversas ferramentas de apoio. Algumas delas são definidas a seguir:

• *Diagramas Scatter:* são gráficos que mostram o relacionamento entre duas características de processo (**Figura 2**). São limitados a analisar apenas duas variáveis no tempo. São precursores da análise de regressão e podem ser utilizados como o primeiro passo para a exploração de dados em busca de relações entre causas e efeitos. Quando um diagrama *scatter* sugere que há um relacionamento entre duas variáveis, seu uso deve ser seguido de métodos estatísticos mais formais, como análise exploratória de dados ou análise de regressão, para que seja possível quantificar o relacionamento entre as variáveis analisadas.

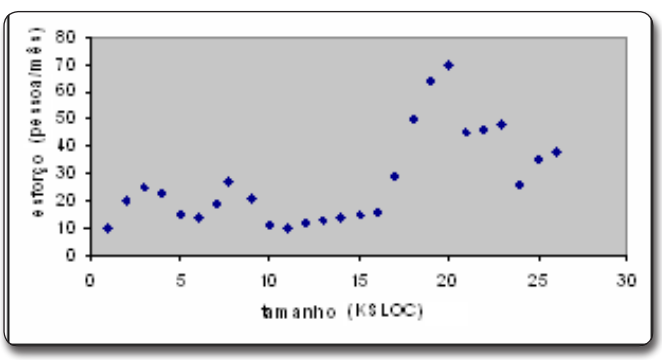


Figura 2. Diagrama Scatter relacionando esforço e tamanho.

• *Gráficos de Tendência (Run Charts):* mostram os padrões e tendências existentes nos dados ao longo do tempo (**Figura 3**). Podem ser utilizados para identificar tendências ou mudanças repentinas no comportamento dos processos. São similares aos gráficos de controle, porém sem considerar os limites de controle utilizados nesses gráficos (os gráficos de controle serão apresentados mais adiante). Devido a essa característica, é preciso tomar cuidado ao analisar uma variável em um gráfico de tendência pois, no contexto da análise do comportamento de um processo, nem toda variação é importante, e a análise isolada de um gráfico de tendência pode levar a investigações desnecessárias.

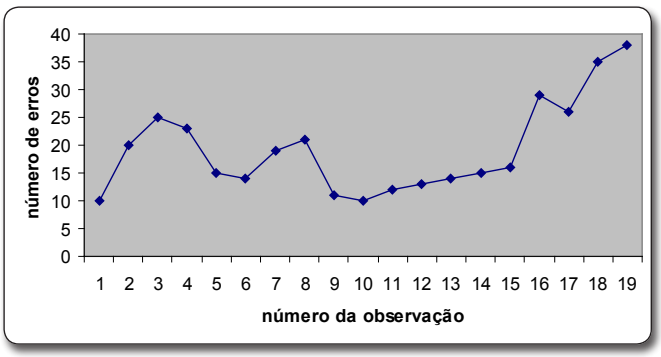


Figura 3. Gráfico de Tendência.

• *Diagramas de Causa e Efeito:* também conhecidos como diagramas *Ishikawa* ou diagramas Espinha de Peixe, podem ser utilizados para representar o relacionamento entre um problema (o efeito) e suas possíveis causas (**Figura 4**). Na análise de comportamento de processos, permitem mapear e priorizar um conjunto de fatores que podem afetar o processo. São úteis para organizar e priorizar as informações fornecidas por pessoas que conhecem o processo (por exemplo, em sessões de *brainstorm*) e podem auxiliar na investigação de problemas.

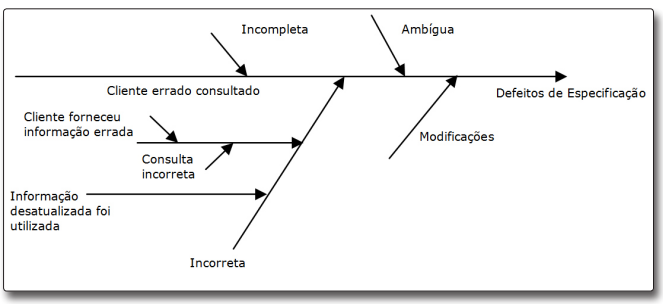


Figura 4. Diagrama de Causa e Efeito para identificar causas de defeitos de especificação.

• *Histogramas:* são gráficos que mostram as distribuições dos dados, ou seja, a freqüência dos eventos que ocorreram para um conjunto de dados em um determinado período (**Figura 5**). A freqüência dos dados é representada por barras cujo arranjo revela o 'formato' do conjunto de dados. Podem ser utilizados para caracterizar valores observados em qualquer atributo do produto ou processo. Sua representação facilita a análise da distribuição, tendências e dispersões dos dados. Aos histogramas podem ser adicionados limites de especificação para identificar os valores que o produto ou processo devem satisfazer, segundo a variável que está sendo representada. A inclusão desses limites favorece a tomada de decisão, pois facilita para a identificação de problemas.

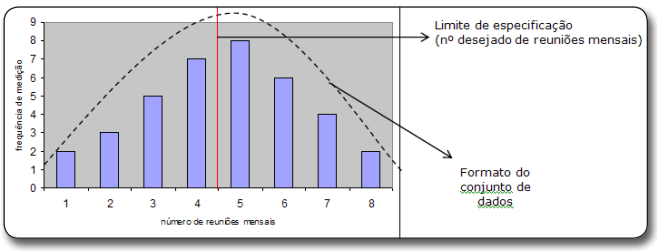


Figura 5. Histograma para representar o número de reuniões mensais realizadas nos projetos de uma organização.

• *Gráficos de Barras:* são similares aos histogramas e também são utilizados para investigar o formato de um conjunto de dados (**Figura 6**). Porém, consideram mais dimensões que o histograma, permitindo comparar um número maior de variáveis.

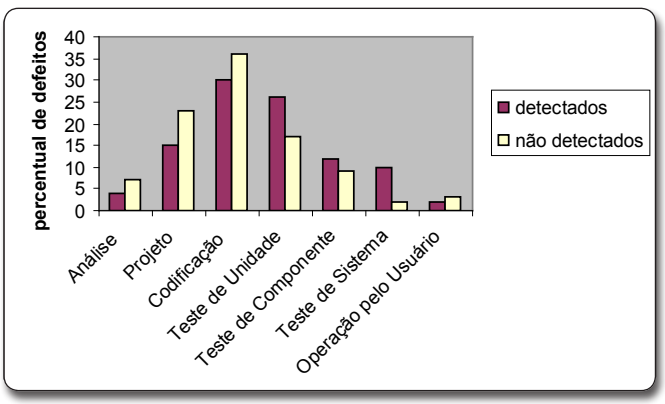


Figura 6. Diagrama de barras para analisar o percentual de defeitos ao longo das fases de um projeto.

• *Gráficos de Pareto:* constituem uma forma especial de histograma ou gráfico de barras (**Figura 7**). São diagramas de freqüência que mostram quantos resultados foram gerados por tipo ou categoria de causa identificada. No contexto do comportamento dos processos, podem ser utilizados para auxiliar a determinar a prioridade de ações corretivas e oportunidades de melhoria. Por exemplo, um gerente deve tratar as causas que estão gerando o maior número de defeitos.

Lei de Pareto: Uma quantidade consideravelmente pequena de causas irá, tipicamente, causar a grande maioria dos problemas. Normalmente, é referenciada como princípio 80/20 (80% dos problemas se devem a 20% das causas).

• *Gráficos de Controle:* são técnicas utilizadas para monitorar a variabilidade existente no comportamento dos processos, permitindo distinguir variações aceitáveis que precisam ter suas causas investigadas e tratadas. Os gráficos de controle são detalhados a seguir.

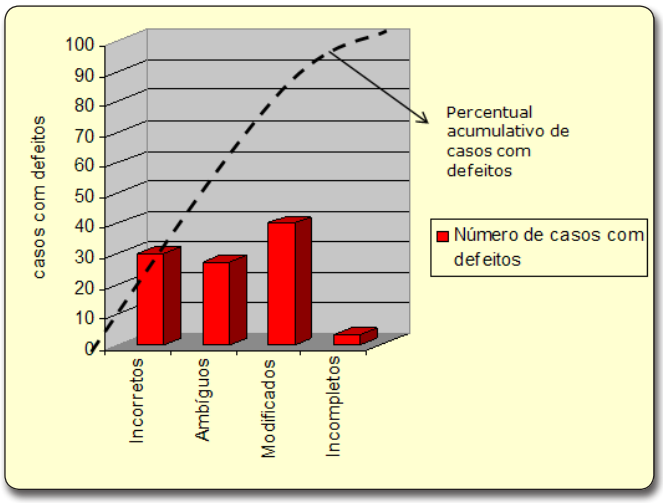


Figura 7. Gráfico de Pareto para analisar as causas de defeitos de especificação.

Gráficos de Controle: Analisando a Estabilidade dos Processos

Os gráficos de controle são técnicas utilizadas para quantificar o comportamento dos processos. Eles estão associados a métodos quantitativos e/ou estatísticos de controle de qualidade que auxiliam a detecção de causas especiais de variação no comportamento dos processos e a diferenciação destas em relação às causas comuns de variação. Ou seja, os gráficos de controle auxiliam a análise da *estabilidade* dos processos.

Tipos de Gráficos de Controle

Existem diversos tipos de gráficos de controle e cada um deles é melhor aplicável a determinadas situações. Conforme apresentado no artigo da edição anterior, todos os tipos de gráficos de controle seguem a estrutura básica mostrada na **Figura 8**, onde σ é o desvio padrão.

Cada tipo de gráfico de controle possui um conjunto próprio de expressões matemáticas e constantes que são utilizados para plotar os dados e para calcular os limites de controle. Além disso, para cada gráfico de controle podem ser aplicados determinados testes para verificar se há causas especiais a serem investigadas. Normalmente são aplicados os chamados *testes 1* a *4* propostos por WHEELER e CHAMBERS (1992) e apresentados no artigo da edição anterior. Porém, nem todos esses testes são aplicáveis a todos os tipos de gráficos de controle.

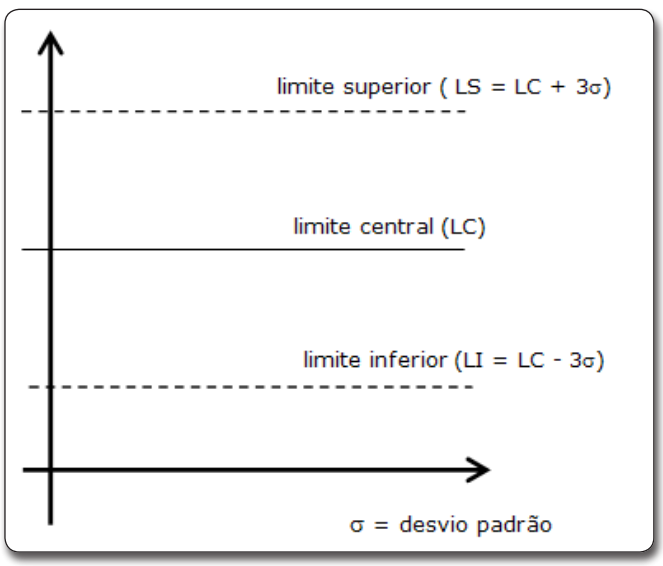


Figura 8. Layout básico de um gráfico de controle.

O primeiro passo para escolher o tipo de gráfico de controle a ser utilizado é identificar o tipo de dado que será tratado. Gráficos de controle podem ser aplicados a duas classes diferentes de dados: *dados de variáveis* e *dados de atributos*.

Dados de variáveis, comumente, são medidas de fenômenos contínuos, como por exemplo esforço, tempo e custo. Dados de atributos, por outro lado, ocorrem quando uma informação é registrada apenas quando uma entidade da população analisada satisfaz ou não um critério ou um conjunto de critérios. Normalmente indicam contagens discretas. Por exemplo: número de defeitos encontrados, número de membros do projeto que possuem determinada característica, percentual de projetos que utilizam inspeções, etc.

Para identificar se os dados são discretos ou contínuos é preciso considerar o contexto em que os mesmos são coletados e utilizados.

Analisar o número total de requisitos é realizar uma contagem, logo seria classificado como dado de atributo. Porém, ao se contar o número total de requisitos, estão sendo contadas todas as entidades em uma população e não apenas ocorrências de entidades que tenham um ou mais atributos específicos. Sendo assim, contagens de entidades que representam o tamanho total de uma população devem ser tratadas como dados de variáveis, pois têm natureza contínua, mesmo que haja instâncias de contagens discretas nelas.

O número de dias trabalhados em um mês pode ser visto como um dado de atributo se for utilizado para identificar a proporção do mês que está disponível para trabalho, e pode ser considerado um dado variável se for utilizado para normalizar alguma outra medida, como por exemplo, o retrabalho acumulado nesses dias.

Na **Tabela 1** são apresentados os tipos de gráficos de controle para dados de variáveis e para dados de atributos, destacando-se suas principais características e exemplos de situações onde podem ser aplicados.

Com tantos tipos de gráficos de controle disponíveis, diante de uma situação, é preciso analisar o tipo de dados que está sendo tratado e o contexto em que a situação está inserida para determinar o tipo de gráfico de controle mais adequado. Vale ressaltar que algumas situações permitirão o uso de mais de um tipo de gráfico. Em caso de dúvidas ou de resultados questionáveis, sempre que possível, deve-se aplicar mais de um tipo de gráfico e comparar os resultados.

Utilizando um Gráfico de Controle

Bom, mas ainda fica uma pergunta: - Como utilizar um gráfico de controle?

A seguir é apresentado um exemplo de aplicação do gráfico

de controle *XmR*. A aplicação dos demais tipos de gráficos de controle é bastante parecida, porém, as expressões matemáticas, constantes e testes aplicados são particulares a cada tipo.

O gráfico de controle *XmR* é composto por dois gráficos: o gráfico *X* que representa os valores individuais das observações e o gráfico *mR*, chamado *moving range*, que representa as variações ocorridas entre o valor de uma observação e o valor de sua observação anterior.

Considerando os *testes 1 a 4* para análise da estabilidade, aos gráficos de controle *XmR*, apenas pode ser aplicado o *teste 1* que diz que a presença de causas especiais de variação é identificada pela presença de pontos fora dos limites de controle calculados.

Tipo de Gráfico de Controle	Características	Exemplo de situação onde se aplica
Para Dados de Variáveis		
X-bar e R	São adequados para analisar o comportamento do processo através de sub-agrupamentos de medidas obtidas, basicamente, sob as mesmas condições, em determinados períodos de tempo. O gráfico X-bar (average) analisa a média dos valores em cada sub-agrupamento e o gráfico R (range) indica a variação interna dos subgrupos. Este tipo de gráfico se limita a subgrupos formados por até 10 observações.	Um gerente deseja analisar a quantidade de horas semanais que são dedicadas a atividades de suporte. Para coletar os dados são registradas as horas de trabalho com suporte diariamente. Como a análise desejada é semanal, para serem analisados, os dados devem ser sub-agrupados por semana, logo, cada subgrupo tem 5 observações (uma para cada dia da semana).
X-bar e S	Também são adequados para analisar o comportamento do processo através da análise de subgrupos de medidas, porém, consideram subgrupos com mais de 10 observações.	Um gerente deseja analisar a taxa de inspeção de código das releases de um de seus produtos. O produto tem 4 releases e em cada uma delas foram realizadas 15 inspeções. Como a análise desejada é por release, os dados devem formar 4 subgrupos de 15 observações.
XmR	São adequados para analisar o comportamento de um processo quando uma mesma medida é coletada frequentemente. Neste tipo de gráfico de controle o gráfico X representa os valores individuais das medidas analisadas e o gráfico mX (moving range) representa a variação existente entre uma medida e sua antecessora.	Um gerente deseja analisar o esforço diário despendido com manutenção no último mês. Nesse caso, o gerente deseja analisar uma única variável (esforço) medida frequentemente, sem necessidade de criar subgrupos.
XMmR	Assim como os gráficos XmR, são adequados para analisar o comportamento de um processo quando uma mesma medida é coletada frequentemente. Porém, nos gráficos XmR é utilizada a média como base para cálculo dos limites de controle e, nos gráficos XMmR, é utilizada a mediana. A mediana pode ser mais sensível às causas assinaláveis, principalmente quando o moving range (variação existente entre o valor de uma medida e o valor de sua antecessora) possui alguns valores que podem elevar ou diminuir os limites desnecessariamente. Nesses casos, utilizar a mediana poderá revelar causas assinaláveis que não apareceriam se fosse utilizada a média.	Um gerente deseja analisar o esforço diário despendido com manutenção no último mês e observa que, entre os valores coletados, há três que destoam consideravelmente de seus antecessores.
mXmR	Seguem a mesma filosofia dos gráficos XmR e XMmR, porém, além de considerarem o moving range, consideram também moving average, ou seja, o gráfico mX plota a média de dois valores subseqüentes. Este tipo de gráfico é adequado para avaliar as tendências do comportamento dos processos ao longo do tempo, considerando valores acumulados.	Um gerente de projetos de software deseja analisar o progresso do desenvolvimento das unidades de um software. Para isso, mensalmente, devem ser consideradas, acumuladamente, as unidades que foram concluídas desde o início da fase de projeto até o momento atual.
Para Dados de Atributos		
C Charts	São adequados para representar a contagem de eventos discretos em um domínio finito, onde as oportunidades de observação são as mesmas para todos os eventos.	Um gerente de projetos de software deseja analisar a quantidade de falhas de um determinado software registradas diariamente.
U Charts	São adequados para representar a contagem de eventos que podem ser medidos em diferentes condições de observação. Por esse motivo, os limites de controle superior e inferior são calculados para cada observação. Nesses casos, antes de ser realizada a comparação entre as medidas coletadas, estas devem ser transformadas em taxas.	Um gerente de projetos de software deseja analisar a quantidade de defeitos encontrados por módulo em um determinado software. Nesse caso, antes de ser realizada a comparação entre os dados coletados, os mesmos devem ser normalizados pelo tamanho dos módulos, obtendo-se, por exemplo, o número de defeitos por KSLOC. Apenas depois de normalizados, os dados podem ser plotados e comparados, pois não faz sentido comparar o número de defeitos encontrados em um módulo de tamanho t com o número de defeitos encontrados em um módulo de tamanho 3t.
Z Charts	Convertem os valores de um U Chart para a escala baseada em sigma (σ), ou seja, no desvio padrão. É adequado para visualizar tendências à instabilidade no comportamento do processo por ser mais sensível quando há grandes diferenças entre os valores das observações.	Um gerente de projetos de software deseja analisar a quantidade de defeitos encontrados por módulo em um determinado software. Para isso, antes de comparar os dados, os mesmos são normalizados pelo tamanho dos módulos, obtendo-se a taxa de número de defeitos por KSLOC. Analisando-se os valores das taxas, o gerente percebe que algumas possuem valores bastante diferentes das demais.

Tabela 1. Tipos de gráficos de controle.

Para calcular os limites de controle dos gráficos *X* e *mR* devem ser utilizadas as seguintes expressões:

Para o gráfico X :

$$LS = \overline{X} + \frac{3\overline{mR}}{d_2}$$

$$LC = \overline{X}$$

$$LI = \overline{X} - \frac{3\overline{mR}}{d_2}$$

Onde:

LS = limite superior
LC = limite central
LI = limite inferior

$$\overline{X} = \frac{1}{k}\sum_{i=1}^{i=k} X_i \quad e$$

k = número de observações
d_2 *= constante = 1,128*

Para o gráfico mR:

$LI = 0$ *(uma vez que o gráfico mR considera os módulos das variações, o limite inferior é sempre zero)*

$$LC = \overline{mR}$$

$$LS = D_4\overline{mR}$$

Onde:

$mR_i = |X_{i+1} - X_i|$ *para* $1 \leq i \leq K-1$

$r = k - 1$
k = número de observações

$$\overline{mR} = \frac{1}{r}\sum_{i=1}^{i=r} mR_i$$

D_4 *= constante = 3,268*

Conhecidos os gráficos, as expressões matemáticas e os testes aplicáveis ao tipo de gráfico de controle *XmR*, só falta utilizar o gráfico de controle. Então, suponha a seguinte situação: um gerente está analisando a medida *número de problemas não resolvidos (NPNR)*, registrada e comunicada semanalmente, que indica quantos problemas informados à equipe de resolução de problemas não foram resolvidos. Observando que o valor contido no último relatório semanal é aparentemente alto (28 problemas não resolvidos), o gerente fica preocupado e decide analisar o comportamento do processo de resolução de problemas incluindo as 30 semanas anteriores, para observar se houve alguma alteração no comportamento do processo ou se os 28 problemas não resolvidos registrados no último relatório refletem o comportamento esperado para o processo.

Na **Tabela 2** são apresentados os dados coletados para a medida NPNR nas últimas 31 semanas, e os valores calculados para *mR*.

Utilizando-se as expressões definidas para os gráficos *X* e *mR*, tem-se:

• Cálculo das médias e $\overline{X}$ e $\overline{mR}$, considerando-se $k = 31$ e $r = 30$:

$$\overline{X} = \frac{1}{k}\sum_{i=1}^{i=k} X_i = 20,39 \qquad \overline{mR} = \frac{1}{r}\sum_{i=1}^{i=r} mR_i = 4,7$$

Cálculo dos limites de controle:

Para X:

$$LS = \overline{X} + \frac{3\overline{mR}}{d_2} = 20,39 + 3*4,7 / 1,128 = 20,39 + 12,5 = 32,89$$

$$LI = \overline{X} - \frac{3\overline{mR}}{d_2} = 20,39 - 3*4,7 / 1,128 = 20,39 - 12,5 = 7,89$$

Para mR:

$$LS = D_4\overline{mR} = 3,268 * 4,7 = 15,36$$

Construção dos gráficos *X* (**Figura 9**) e *mR* (**Figura 10**):

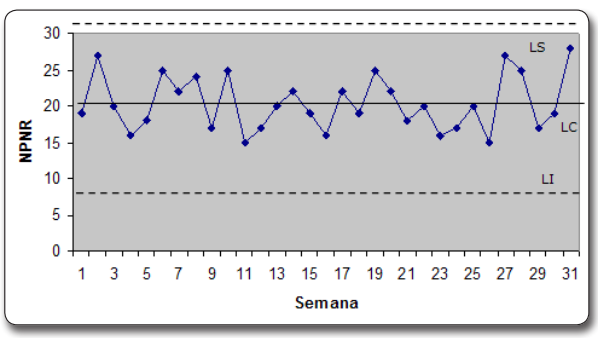


Figura 9. Gráfico X para a medida NPNR.

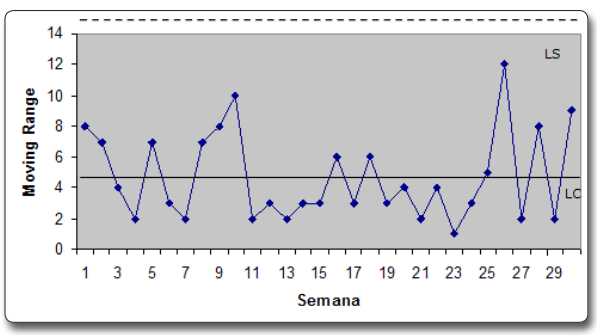


Figura 10. Gráfico mR para a medida NPNR.

Semana	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
NPNR (Xi)	19	27	20	16	18	25	22	24	17	25	15	17	20	22	19	16
mR (| Xi+1 – Xi|)	8	7	4	2	7	3	2	7	8	10	2	3	2	3	3	6

Semana	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
NPNR (Xi)	22	19	25	22	18	20	16	17	20	15	27	25	17	19	28
mR (| Xi+1 – Xi|)	3	6	3	4	2	4	1	3	5	12	2	8	2	9	

Tabela 2. Dados coletados para a medida NPNR nas últimas 31 semanas, com cálculo de mR.

Analisando-se os gráficos e aplicando-se o *teste 1*, é possível observar que o processo de resolução de problemas, quando analisado pela variável número de problemas não resolvidos, mostra-se com comportamento dentro dos limites esperados (processo estável), o que leva o gerente a concluir que os 28 problemas não resolvidos registrados no último relatório semanal não refletem um número diferente do comportamento esperado para o processo considerando as últimas 31 semanas.

Analisando a Capacidade dos Processos

Considerando a utilização do controle estatístico de processos na melhoria de processos de software, foi comentado no início deste artigo que as ações de melhoria visam à estabilização de processos, capacitação de processos ou melhoria contínua de processos.

Na seção anterior foram apresentados os gráficos de controle, utilizados para analisar a estabilidade do comportamento dos processos. Uma vez que um processo é estável, é preciso verificar se o mesmo é capaz.

Um processo é dito capaz se seu desempenho atende ou excede às expectativas da organização e do cliente, ou seja, se seu desempenho permite alcançar os objetivos técnicos e de negócio para ele estabelecidos.

Mas, como analisar a capacidade de um processo?

A forma mais simples de analisar a capacidade de um processo estável é utilizar um histograma que represente os valores coletados para o processo durante um período de estabilidade. Os limites de controle do processo, também chamados de limites naturais, podem ser utilizados para caracterizar seu desempenho e representar a *voz do processo*. Também devem ser representados no histograma os limites de especificação, que identificam os valores que se deseja que o processo alcance (*voz do cliente*). Representados os limites naturais e de especificação, é possível analisar a capacidade do processo. Limites naturais completamente dentro dos limites de especificação caracterizam um processo capaz.

Para exemplificar, considere a seguinte situação: o diretor do departamento de projetos de uma empresa deseja analisar se o processo de manutenção medido através do esforço despendido em atividades de manutenção por semana nos projetos está atendendo aos objetivos estabelecidos, ou seja, deseja verificar se o processo é capaz. A **Tabela 3** apresenta os dados que foram coletados.

Utilizando-se as expressões dos gráficos *XmR*, os limites naturais do processo foram calculados e os valores obtidos para os limites inferior, central e superior foram, respectivamente, de 36.8, 45.06 e 54.04. A **Figura 11** ilustra os dados coletados representados em um histograma onde os limites naturais do processo (voz do processo) são identificados, bem como os limites de especificação, ou seja, os valores estabelecidos para o comportamento do processo (voz do cliente).

Analisando-se o histograma, é possível perceber que o processo é capaz, uma vez que seus limites naturais estão dentro dos limites de especificação estabelecidos (limite de especificação inferior = 35 e limite de especificação superior = 55).

Esforço	38,5	39	39,5	40,5	41	41,5	42	42,5	43	43,5	44	44,5	45
Nº de projetos	2	1	2	2	2	5	3	5	5	4	5	7	2

Esforço	45,5	46	46,5	47	47,5	48	48,5	49	49,5	50	50,5	51	51,5	52
Nº de projetos	6	5	1	5	2	3	1	3	1	3	1	2	1	1

Tabela 3. Esforço despendido em manutenção nos projetos.

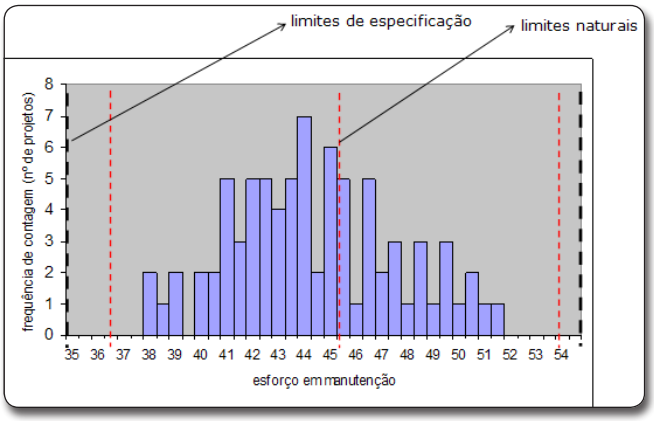


Figura 11. Histograma com limites identificados – processo capaz.

A utilização do histograma permite uma análise visual da capacidade dos processos. Mas é possível quantificar a capacidade do processo. Para isso pode ser utilizado o *índice de capacidade* (*Cp*), dado pela razão entre as amplitudes dos limites de especificação e limites naturais do processo.

Quando os limites de especificação têm amplitude maior que os limites naturais, *Cp* é maior que 1, indicando que o processo é capaz. Em contrapartida, *Cp* será menor que 1 quando os limites de especificação tiverem amplitude menor que os limites naturais, indicando que o processo não é capaz. Para o exemplo ilustrado na **Figura 11**, tem-se $Cp = (55\text{-}35)/(54{,}04\text{-}36{,}8) = 1{,}51$.

Apesar de ser possível calcular o índice de capacidade de um processo sem representar seu histograma, tal ação não é recomendada, pois se pode obter um índice de capacidade maior que 1 e, ainda assim, o processo não ser capaz. No exemplo anterior, se os limites de especificação fossem 30 e 50, *Cp* continuaria sendo 1,51, porém ao representar os limites em um histograma fica visível que parte do comportamento do processo se encontra acima do limite de especificação superior, conforme mostra a **Figura 12**. Na figura, o limite de especificação inferior, cujo valor é 30, não está representado,

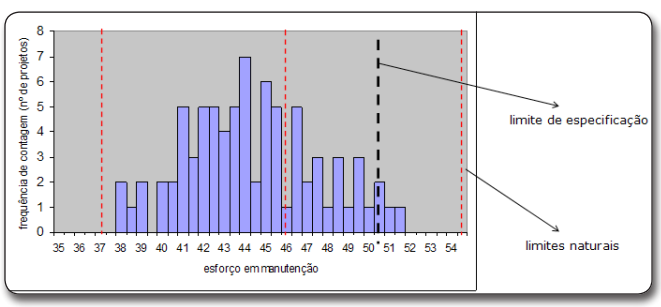


Figura 12. Histograma com limites identificados – processo não capaz.

mas é possível perceber que o limite natural inferior encontra-se dentro deste valor.

Algumas abordagens de qualidade, como o Six Sigma por exemplo, incluem a análise visual e utilizam *Cp* 2 para gerar maior confiança de que o desempenho especificado será atendido.

Considerações Finais

A aplicação bem sucedida do controle estatístico de processos na manufatura levou à sua utilização em outras áreas, como química, eletrônica, alimentação, negócios, saúde e desenvolvimento de software.

Apesar de recente, a utilização do controle estatístico de processos em organizações de software tem sido impulsionada pela competitividade do mercado, que exige produtos cada vez melhores, em cada vez menos tempo e consumindo o mínimo de recursos possível.

O controle estatístico de processos apóia a melhoria dos processos através da análise de seu comportamento e identificação de aspectos que precisam ser melhorados. Mas é importante ressaltar que utilizar o controle estatístico de processos não pode ser uma ação isolada. É desejável que sua utilização seja realizada no contexto de um programa de melhoria de processos, caso contrário, seus resultados tendem a ser pontuais e passageiros.

Os gráficos de controle utilizados no controle estatístico de processos fornecem uma visão do comportamento dos processos, facilitando a identificação de questões que precisam ser investigadas. Outras técnicas analíticas, como o diagrama de causa e efeito e o diagrama de Pareto, podem ser utilizadas como apoio à investigação dessas questões e à identificação de ações corretivas e/ou de melhoria necessárias.

A maneira como as técnicas são utilizadas e o conhecimento organizacional envolvido nas investigações e identificação de ações podem fornecer às organizações um diferencial competitivo.

Sendo assim, é importante ressaltar que utilizar o controle estatístico de processos não consiste, simplesmente, em montar gráficos e... pronto! Os gráficos de controle são um meio, e não um fim. Eles são um instrumento que molda a matéria-prima (os dados) para que as ações de melhoria sejam possíveis.

É comum organizações gerenciarem seus projetos e processos seguindo a filosofia do "retrovisor", ou seja, olhar para o que passou para fazer melhor da próxima vez. Essa é uma abordagem que funciona, mas tem seus limites. A utilização do controle estatístico de processos inserida em um programa de melhoria de processos leva à filosofia do "painel de controle", ou seja, olhar para o que está acontecendo no momento para fazer melhor a partir de já!

Sob o ponto de vista da Engenharia de Software, a utilização do controle estatístico de processos na melhoria dos processos pode ser vista como uma evolução da melhoria tradicional.

Mas, qual filosofia adotar? A do "retrovisor" ou a do "painel de controle"?

Não há dúvidas de que ambas são necessárias.

Mas posso afirmar com certeza que eu não gostaria de me aventurar a dirigir um carro sem "painel de controle". E você? ●

Referências Bibliográficas

DEMING, W. E., 1986, "Out of Crises", Massachusetts Institute of Technology, Center of Advanced Engineering, Cambridge.

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

SHEWART, W. A., 1980, "The Economic Control of Quality of Manufactured Product", D. Van Nostrand Company, New York, 1931, re-impresso por ASQC Quality Press, Milwaukee, Wiscosin.

WHEELER, D. J., CHAMBERS, D. S., 1992, "Understanding Statistical Process Control", 2nd ed. Knoxville, SPC Press.

Estimativas de Software - Fundamentos, Técnicas e Modelos – Parte 2

O COCOMOII e uma introdução ao seu processo de calibração

Carlos Eduardo Vazquez

Mini currículo: Sócio-fundador da FATTO Consultoria e Sistemas, um dos autores do livro "Análise de Pontos de Função: Medição, Estimativas e Gerenciamento de Projetos de Software", atualmente em sua 8ª edição. Pioneiro na aplicação de métricas de software no Brasil possui 20 anos de experiência em TI, notoriamente na aplicação das disciplinas do desenvolvimento e sustentação de sistemas corporativos. Graduado em Processamento de Dados pela PUC-RJ em 1990, já passou com sucesso por quatro vezes pelo processo de certificação de especialista em pontos de função pelo IFPUG - International Function Point Users Group, tendo sido um dos primeiros brasileiros a conquistar essa certificação em 1996. Desde 1993, vem formando profissionais na aplicação da Análise de Pontos de Função, tendo sido professor da UFES, atuado como consultor de grandes projetos de tecnologia em empresas do setor financeiro, bancário e de telecomunicações.

De que se trata o artigo?

Estimar é uma atividade cotidiana, sistematicamente evitada por aqueles responsáveis pela sua execução. Na busca por superar isso, uma série de técnicas e ferramentas surge no cenário do desenvolvimento e manutenção de sistemas. Muitas vezes, no desespero por uma solução imediata, são adotadas independentemente de sua adequação ao cenário específico em que serão introduzidas ou mesmo apenas com um conhecimento superficial quanto ao seu funcionamento. Ferramentas como o COCOMOII, Simulação de Monte Carlo e Pontos de Função não substituem a analista responsável pela estimativa, que enfrenta a confusão entre diferentes atos, entre o que seja uma estimativa técnica, um compromisso pessoal ou uma meta corporativa.

Para que serve?

Nosso objetivo é diferenciar entre esses diferentes atos e como se portar diante de cada um deles; destacar que simples cuidados podem ajudar a produzir estimativas de mais qualidade; apresentar como funciona uma série de ferramentas isoladamente e como integrá-las no estabelecimento de um ambiente propício à melhoria contínua da qualidade das estimativas.

Em que situação o tema é útil?

Nas diferentes situações em que um analista deve se relacionar com seus clientes no sentido de fornecer a sua expectativa para um prazo, custo, esforço ou escopo no desenvolvimento e manutenção de software. Visa ajudar a esse analista a identificar os diferentes tipos de solicitação e evitar que ele caia em armadilhas que o leve a assumir compromissos inexeqüíveis. Adicionalmente, é útil também àquele profissional que trabalha na definição de processos de desenvolvimento e seleção de métodos e ferramentas para fins de melhorar o processo de estimativa de sua organização.

No artigo da edição anterior, os fundamentos para estimativas profissionais foram apresentados; algumas técnicas de estimativa exploradas, ainda que superficialmente; e foi feita a introdução sobre o COCOMOII. O objetivo deste texto é dar seqüência na exploração desse modelo e conhecer os passos comuns necessários à realização de qualquer estudo de produtividade e da aplicação dos seus resultados no planejamento de projetos de software. Nesse processo, vários dos conceitos básicos e gerais apresentados na edição anterior se mostram úteis e aplicados ao modelo de estimativas em questão.

A definição das fases da produção de software numa perspectiva gerencial e dos marcos que as delimitam

Um dos primeiros desafios ao conceber um modelo de custos para a engenharia de software é estabelecer um conjunto de premissas que permitam utilizar o modelo consistentemente entre diferentes organizações de software, usando diferentes abordagens e estratégias para entregar os produtos de software demandados pelos seus clientes.

O COCOMOII define e estabelece uma série de premissas que viabilizam esse objetivo. Elas também permitem que o modelo seja de muito valor na realização de estudos de produtividade na medida em que facilita a normalização na elaboração de estimativas (dados previstos), e a subseqüente análise do que foi realizado (dados realizados).

O primeiro passo nessa exploração do COCOMOII é construir um conhecimento daquilo que o modelo define e estabelece como premissa. Nesse processo, a divisão do projeto em fases (não necessariamente em disciplinas como análise de requisitos, modelagem, codificação e testes, por exemplo) é o primeiro passo nessa construção.

As fases devem ser externas à função de desenvolvimento e manutenção de software de tal forma que possam ser usadas para fins de planejamento de alto nível, quando ainda se tem poucos detalhes sobre o objeto do trabalho e os riscos de escopo e produtividade associados ao mesmo ainda são muitos. Os responsáveis pela elaboração do COCOMOII não têm a intenção de revolucionar no que diz respeito a essa definição nas fases do desenvolvimento de software e buscam usar o que é geralmente aceito pela comunidade de engenharia de software: a estratégia de desenvolvimento em cascata e a abordagem denominada de processo unificado.

A **Figura 1** é chave para o entendimento do COCOMOII e será mais explorada na medida em que for sendo construído o conhecimento sobre o modelo. Neste ponto, ela ilustra os marcos entre as fases das diferentes estratégias de desenvolvimento citadas.

As fases de iniciação, elaboração, construção e transição são aquelas definidas pelo Rational Unified Process (RUP), enquanto as fases de planos e requisitos, projeto preliminar, projeto detalhado, codificação e testes de unidade, testes e integração são aquelas encontradas em projetos utilizando um ciclo de vida em cascata (waterfall). O COCOMOII identifica e posiciona marcos comuns entre essas duas diferentes estratégias de desenvolvimento citadas, esse nivelamento é estabelecido com base nos produtos que se espera receber do projeto nesses marcos. Isso permite: (a) que sejam feitas estimativas por fase, o que facilita o planejamento na medida em que as diferentes fases têm maior intensidade de determinados tipos de trabalho com diferentes perfis de recursos humanos necessários; (b) que o responsável pela estimativa se localize no tempo, no momento do projeto e,

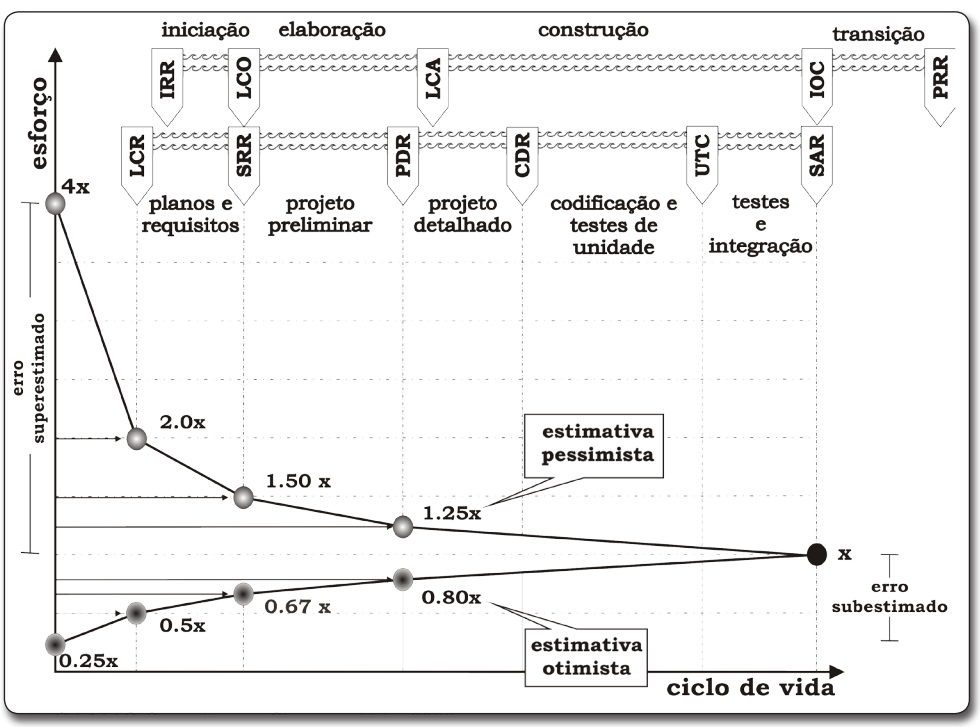


Figura 1. Fases do ciclo de vida e a sua integração com o COCOMOII

Marcos para Gestão do Processo	
Waterfall	**RUP**
LCR – Revisão de Conceito do Ciclo de Vida (*Life Cycle Concept Review*)	IRR – Revisão de Prontidão de Concepção
SRR – Revisão de Requisitos do Software (*Software Requirements Review*)	LCO – Revisão de Objetivos do Ciclo de Vida (*Life Cycle Objectives Review*)
PDR – Revisão de Projeto do Produto (*Product Design Review*)	LCA – Revisão da Arquitetura do Ciclo de Vida (*Life Cycle Architecture Review*)
CDR – Revisão de Projeto Crítico (*Critical Design Review*)	IOC – Capacidade Operacional Inicial (*Initial Operational Capability*)
UTC – Critério de Teste de Unidade (*Unit Test criteria*)	PRR – Revisão de Liberação do Produto (*Product Release Review*)
SAR – Revisão de Aceitação do Software (*Software Acceptance Review*)	

Tabela 1. Marcos conforme a estratégia de processo de desenvolvimento para fins de gestão

com isso, possa estabelecer o quanto já deveria ter sido feito e o quanto resta a fazer; e (c) que haja condições para uma análise de produtividade econômica (considerado como um requisito necessário para estimativas de qualidade).

A **Tabela 1** descreve os acrônimos dos marcos apresentados na **Figura 1**. A definição desses marcos não é do COCOMOII, que buscou usar padrões reconhecidos e estabelecidos, estando sua qualificação completa fora do escopo deste texto.

Na experiência do autor deste texto, ainda há na comunidade de profissionais de TI brasileira uma certa dificuldade em compreender a dinâmica nos processos de desenvolvimento iterativos e incrementais. Muitos utilizam o vocabulário e a terminologia do RUP, porém trabalham na verdade com um desenvolvimento em cascata.

De forma simplificada, estratégias de desenvolvimento baseadas no processo unificado buscam a realização de atividades de levantamento de requisitos em determinados pacotes de trabalho enquanto já há codificação ou modelagem de outros.

Observar o processo unificado não impede que haja fases definidas para fins de gerenciamento externo. Independentemente da estratégia de desenvolvimento utilizada, os clientes, aqueles que patrocinam o empreendimento, esperam uma série de produtos por eles reconhecidos e aceitos conforme o ciclo de vida do desenvolvimento evolui. Alguns desses produtos são definidos entre esses últimos e os responsáveis pela função de desenvolvimento no âmbito do próprio projeto já em execução e com base em princípios previamente definidos.

Por exemplo, uma empresa atuante em todo o país é responsável pela evolução de seu produto de gestão de recursos humanos em janelas sucessivas de três meses (time box). Nesse cenário, é a quantidade e tamanho das demandas que se ajusta ao prazo, e não o contrário. Procura-se atender ao maior número possível de demandas de melhorias em cada versão com um estoque de trabalho essencialmente constante. Quem cumpre o papel de demandante da função de desenvolvimento é a função de planejamento, que pode ser vista como um preposto interno, do cliente externo.

A unidade de gestão originalmente utilizada no processo de atendimento das demandas é o resultado da ação de um arquiteto de software, técnico, e não tem associado a ela um significado para os profissionais da função de planejamento. Essa unidade de gestão torna-se um pacote de trabalho que passa por diferentes etapas durante o seu atendimento na função de desenvolvimento. O processamento desse pacote de trabalho segue uma estratégia essencialmente em cascata. As etapas existentes para fins de acompanhamento do progresso são relativas a cada pacote de trabalho individualmente, e não para o projeto como um todo.

Em outras palavras, o cliente não tem condição alguma de acompanhar o seu atendimento e fica numa posição em que tem que confiar que a função de desenvolvimento atingirá os objetivos estabelecidos em comum. Esse cenário impede uma gestão externa à função de desenvolvimento. Falta uma unidade de gestão que: (a) tenha significado para a função de planejamento; (b) possa ser medida de forma relevante à função de planejamento; (c) apóie o planejamento e acompanhamento das fases relativas à execução do projeto como um todo.

Para mudar essa situação, foi estabelecida uma unidade de gestão com significado para o departamento de planejamento que cumpre o papel de uma Ordem de Serviço, e foram definidas fases para o acompanhamento do projeto como um todo. São elas: Negociação/Planejamento da Versão; Concepção; Elaboração; Construção; e Transição. A fase de Negociação/Planejamento de Versão compreende:

a) Estimativas de escopo e esforço – elaboração do cronograma macro considerando um prazo total de três meses para as fases de concepção, elaboração, construção e transição usando os percentuais do prazo em cada fase (**Tabela 2**).

	Esforço	Prazo
Negociação / Planejamento	8,54%	50,00%
Concepção	11,56%	13,33%
Elaboração	12,70%	66,67%
Construção	64,36%	66,67%
Transição	2,85%	16,67%

Tabela 2. Distribuição do esforço e prazo.

Os percentuais descritos na **Tabela 2** foram obtidos por aproximação do esforço gasto no cenário atual onde não há utilização de fases externas e servem de referência preliminar. Na medida em que houver a institucionalização das mudanças propostas, esses percentuais devem ser revistos e gradualmente cumprirão um papel mais importante na definição do cronograma macro citado.

Marco	Objetivos do Ciclo de Vida (LCO)	Arquitetura do Ciclo de Vida (LCA)
Definição do conceito de operação	Objetivos e escopo de alto nível da aplicação; fronteira entre aplicações; premissas e parâmetros do ambiente; carências atuais do sistema; conceito de operação: cenários padrões, papéis e responsabilidades dos stakeholders.	Elaboração dos objetivos e escopo do sistema por incremento; elaboração do conceito de operação por incremento; cenários padrões e de exceção chave.
Protótipo(s) do sistema	Simulação com o uso de cenários; resolução dos riscos críticos.	Simulação de uma gama de cenários de uso; resolução dos principais riscos pendentes.
Definição de requisitos de sistema e software	Recursos de alto nível, interfaces, atributos e níveis de qualidade, incluindo: Evolução dos requisitos; prioridades; e concordância dos stakeholders no essencial. (Observe que essencial é uma resolução tomada antes do início da concepção/iniciação).	Elaboração de funções, interfaces, atributos de qualidade por incremento; Identificação de itens a serem definidos; evolução dos requisitos; concordância dos stakeholders em suas preocupações prioritárias.
Definição da arquitetura do sistema e do software	Definição de alto nível de pelo menos uma arquitetura possível; elementos e relacionamentos físicos e lógicos; escolhas de "software de prateleira" e componentes reutilizáveis de software; identificação de opções de arquitetura inviáveis.	Escolha de arquitetura e elaboração por incremento; restrições, configurações, conexões, componentes lógicos e físicos; Escolha de "software de prateleira", componentes reutilizáveis; escolha de estilo e domínio de arquitetura; parâmetros de evolução da arquitetura.
Definição do plano do ciclo de vida	Identificação dos stakeholders do ciclo de vida: Usuários, clientes, desenvolvedores, mantenedores, integradores, público em geral, outros; identificação do modelo de processo para o ciclo de vida; estágios de alto nível, incrementos; WWWWWHH de alto nível por estágio. (WWWWWHH – Por que (Why), o que (What), quando (When), quem (Who), onde (Where), como (How), quanto (How Much).	Elaboração de WWWWWHH para o início da capacidade operacional; identificação e elaboração parcial dos itens a serem definidos para os incrementos subseqüentes.
Racional de Viabilidade	Garantia de consistência entre os elementos acima pela análise, medição, prototipação, simulação, dentre outros, não cabendo a este texto esgotar as providências para esse fim; análise de casos de negócio para os requisitos, arquiteturas possíveis.	Garantia de consistência entre os elementos acima; racional das principais opções rejeitadas; todos os principais riscos resolvidos ou cobertos pelo plano de gerência de riscos compreendido no plano do ciclo de vida.

Tabela 3. Produtos esperados conforme o marco e a fase do ciclo e vida

b) Definição, para cada ordem de serviço, de quais serão os produtos esperados para que a fase de Elaboração seja considerada concluída. A política é que a especificação funcional das ordens de serviço mais críticas esteja concluída e os riscos arquiteturais prioritários estejam resolvidos. Nesse ponto, espera-se que até 40% do prazo tenha se passado e até 25% do esforço consumido.

c) Marco de término: conjunto com todas as ordens de serviço da versão com o escopo validado e a primeira planilha de contagem de PF por produto.

d) Meta para término: duas semanas antes do final da construção da versão anterior.

A fase de Concepção compreende:

a) Atividades de um comitê para controle de alterações envolvendo o analista de requisitos, arquiteto e analista de negócio onde é feita uma nova validação funcional mais cuidadosa, avaliação do impacto da ordem de serviço no produto e a elaboração das unidades de gestão internas à função de desenvolvimento.

b) Revisão das estimativas de escopo e prazo.

c) Marco de término: Realização e finalização dos comitês para controle de alterações referentes a todas as ordens de serviço originalmente programadas para atendimento no projeto da versão.

d) Meta para término: Final da fase de Construção do projeto da versão anterior.

O exemplo exposto procurou ilustrar os conceitos apresentados referentes à definição de fases externas à função de desenvolvimento para uma organização em particular e em termos de uma experiência prática, no mundo real. Esses marcos apresentados foram definidos a partir das premissas estabelecidas no COCOMOII e expostos na **Tabela 3**, especificamente entre as fases de Iniciação, Elaboração e Construção.

As fases cobertas pela estimativa do COCOMOII

Existem diversas ferramentas, sites, planilhas e até scripts de awk, como o disponível em http://web.cecs.pdx.edu/~timm/dm/cocomo.html, que automatizam o COCOMOII. A **Figura 2** apresenta uma tela que ilustra os resultados fornecidos pelo software desenvolvido pela Universidade do Sul da Califórnia (University of Southern California – USC) e distribuído gratuitamente em conjunto com o livro que define o COCOMOII.

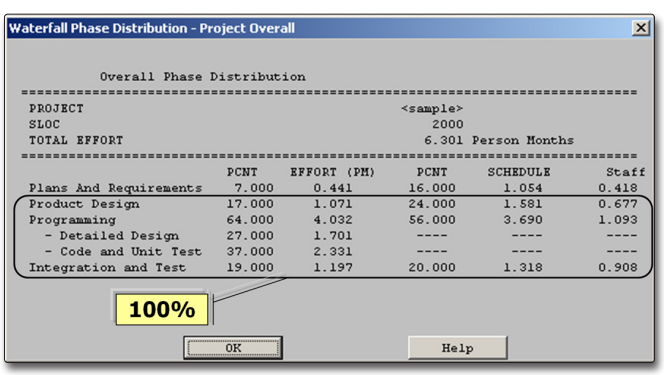


Figura 2. Distribuição geral de fases considerando a estratégia de desenvolvimento em cascata no que se refere ao esforço, prazo e equipe.

Observe na **Figura 2** que, quando os percentuais das colunas de esforço (PCNT relativos às colunas EFFORT e SCHEDULE) são totalizados, o resultado encontrado ultrapassa os 100%.

A fase de planejamento e requisitos representa 7% do total do esforço estimado pelo modelo. Quando o modelo fornece a estimativa de 0,441 Pessoas-Mês para essa fase, isso é feito por extrapolação (essa dinâmica será melhor explicada adiante na **Figura 4)**. Tal situação acontece por que pesquisas da equipe do COCOMOII verificaram que essa fase apresenta uma variação muito grande entre os 181 projetos utilizados na sua elaboração, tanto para o esforço (02 a 15%) quanto para o prazo (02 a 30%). Portanto, antes de usar o modelo, ele deve ser ajustado para que o percentual usado para essa fase seja aquele que se adéqüe ao contexto em que será usado. Por exemplo, se em minha organização esse percentual representar 15%, esse é o percentual que deve ser usado.

Isso não é exclusividade do modelo em cascata, havendo também extrapolações naqueles modelos baseados no RUP, como ilustrado na **Figura 3**.

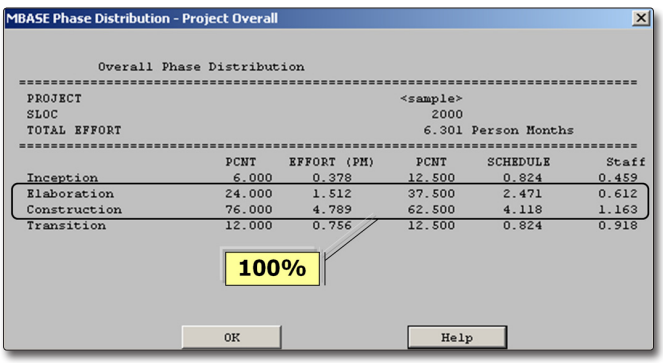


Figura 3. Distribuição geral de fases considerando a estratégia de desenvolvimento do RUP no que se refere ao esforço, prazo e equipe.

Por exemplo, considere que a aplicação do modelo em um projeto utilizando uma estratégia de desenvolvimento iterativa e incremental resulte em um esforço de 100 pessoas-mês, PM (ou 15.200 horas, já que o COCOMO utiliza a princípio 152 Hh / PM). O escritório de projetos verifica com base no levantamento dos projetos da mesma natureza que habitualmente a fase de Concepção representa 18% do esforço total do projeto e a fase de Transição, 20%. Portanto, restam para as outras fases 62% do total e essas 15.200 Hh correspondem a esse percentual. Por extrapolação, chega-se à conclusão que o projeto como um todo tenha um esforço estimado de 24.516 Hh, a fase de Concepção corresponda a 4.512 Hh e a de Transição 4.903 Hh. A **Figura 4** ilustra essa dinâmica da extrapolação aplicada às fases não cobertas pelo COCOMOII.

Os tipos de trabalho incluídos na estimativa do COCOMOII

Um modelo de custos fornece como produtos as estimativas de esforço com base nos parâmetros informados. O que está incluído no escopo das estimativas em termos do tipo de trabalho

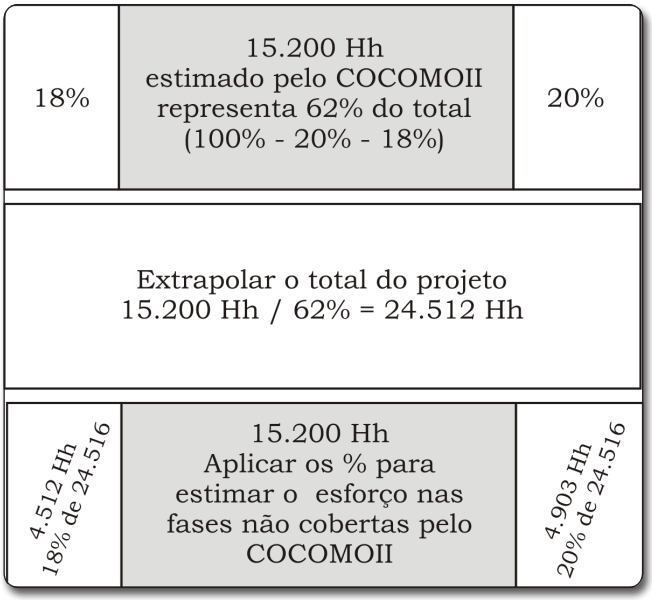


Figura 4. Dinâmica na extrapolação das estimativas para as fases não incluídas nas estimativas geradas pelo COCOMOII.

realizado? A resposta para o caso do COCOMOII é toda atividade cujo custo seja diretamente apropriado ao projeto.

Por exemplo, o trabalho de gerentes de projeto, gerentes de configuração, programadores, analistas de requisitos, arquitetos e analistas de teste podem facilmente ser alocados a um determinado projeto e o respectivo custo ser diretamente apropriado ao mesmo, e o esforço gerado pelo modelo ser representativo do mesmo.

Aquele trabalho considerado overhead, que não é facilmente alocado a um determinado projeto e, consequentemente, não pode ter o seu custo apropriado diretamente ao projeto não está presente em termos das estimativas de esforço do COCOMOII.

Esse trabalho está incluído nas estimativas de custo derivadas, na medida em que o custo médio da pessoa-mês computa a bonificação sobre despesas indiretas (BDI) que visa custear aquilo não ponderado pelo esforço estimado pelo modelo. Exemplos desse tipo de trabalho, que não estão incluídas nas estimativas de esforço do COCOMOII, é o departamento de pessoal, secretárias e executivos.

Tal qual a normalização das fases, a normalização das diferentes naturezas de trabalho tem um papel muito importante na estimativa e na análise do realizado.

A acuidade da estimativa conforme se avança

O COCOMOII traz as faixas de incerteza conforme a fase em que se está no ciclo de vida aos moldes do exposto na primeira parte do artigo da edição anterior - seção "Outras Formas de Representar e Lidar com a Incerteza".

A **Figura 1** também é conhecida como o "Cone da Incerteza" em função do formato que assume ao representar o aumento da acuidade das estimativas na medida em que os riscos de escopo e produtividade diminuem. A **Tabela 4** apresenta essas faixas de incerteza.

Fase	Estimativa Otimista	Estimativa Pessimista
Estudo de Viabilidade	0,25 x	4 x
Escopo Definido (LCR/IRR)	0,5 x	2 x
Planos e Requisitos Elaborados (SRR / LCO)	0,67 x	1,5 x
Projeto Preliminar Concluído (PDR / LCA)	0,8 x	1,25 x

Tabela 4. Níveis de incerteza nas diferentes fases do projeto.

Por exemplo, se uma estimativa for elaborada em um momento logo após o escopo estar definido (LCR) e, ao aplicar o modelo, obtenha-se um resultado de 15.200 Hh, deve-se considerar uma estimativa entre 7.600 Hh e 30.400 Hh. Já se isso acontece ao final da fase de planos e requisitos, deve-se considerar uma estimativa entre 10.184 Hh e 22.800 Hh.

Na seção "Relação entre Escopo e Esforço" do artigo da edição anterior, é apresentado o conceito de fator primário de custo, de tamanho, e de fatores secundários. No COCOMOII, os fatores secundários são aqueles denominados drives de esforço (effort drivers) e fatores de escala (scale factors) e buscam capturar as particularidades do objeto da estimativa quanto ao produto, ao projeto, à plataforma e à equipe.

Por exemplo, uma das particularidades a ser considerada em uma estimativa é o grau de compressão ou distensão do cronograma em relação ao prazo mais curto em que se utiliza o menor esforço (denominado "nominal" no vocabulário do COCOMOII). Nele, esse grau é ponderado pelo driver denominado SCED.

Dependendo do momento em que se esteja no ciclo de vida, da informação disponível e do segmento de mercado em que o objeto da estimativa estiver inserido, o COCOMOII oferece diferentes modelos, endereçando as diferentes necessidades e restrições desses contextos. Basicamente, o que distingue esses modelos é a quantidade de fatores secundários de custo que devem ser avaliados por quem está estimando. A **Figura 5** apresenta a segmentação utilizada na definição do COCOMOII.

Figura 5. Segmentos de mercado no desenvolvimento e manutenção de software utilizado no COCOMOII.

Considerando que o esforço envolvido no segmento de programação pelo usuário final esteja na ordem de até 40 horas, ou menos, e o exposto no artigo da edição anterior na seção "A Ordem de Grandeza: Uma Quinta Dificuldade ao Estimar", os profissionais inseridos nesse segmento não necessitam de um modelo de estimativas como o COCOMOII.

Para os outros segmentos, o COCOMOII oferece três modelos: a) Composição de Aplicação (Application Composition) com três parâmetros; b) Projeto Preliminar (Early Design) com 17 parâmetros; e c) Pós-Arquitetura (Post-Architecture) com 24 parâmetros.

O segmento denominado Composição de Aplicação corresponde aos projetos utilizando ambientes de desenvolvimento integrado (IDE), normalmente utilizadas em projeto de médio/pequeno porte e que consistem basicamente de compor uma aplicação com base em componentes visuais. O COCOMOII propõe a utilização de simples relações entre tamanho e esforço como discutido na seção "Modelos baseados em Simples Relações Usando a Análise de Pontos de Função" do artigo da edição anterior.

Para os demais segmentos, o COCOMOII define dois modelos: O Modelo Projeto Preliminar (Early Design) e o Modelo Pós-Arquitetura (Post-Architecture). Não há obrigatoriedade em usar necessariamente um determinado modelo em determinada fase ou segmento. A **Figura 6** ilustra isso conforme a fase em que se esteja e o nível de informação disponível.

A diferença entre os dois modelos pode ser resumida na quantidade de drivers de esforço utilizados na elaboração da estimativa que devem ser informados pelos seus usuários e a confiabilidade esperada. Tanto o modelo pós-arquitetura quanto o projeto preliminar apresentam confiabilidade de 80%, 10% acima da estimativa otimista e abaixo da pessimista.

Drivers de Esforço e Fatores de Escala x Valor do Fator de Ajuste

Os drivers de esforço e fatores de escala são definidos conforme o modelo e apresentam uma série de orientações para fins de enquadramento do caso em análise quanto a uma escala. Nesse particular, a dinâmica é bastante similar àquela estabelecida pelo IFPUG para determinação do nível de influência de cada característica geral de sistema (GSC) no processo de determinação do valor do fator de ajuste (VAF). Destaca-se, nesse aspecto, por as semelhanças cessarem nisso.

Ao contrário do modelo de VAF onde existe um piso e teto fixos, únicos para a avaliação do impacto de cada característica, no COCOMOII cada driver de esforço e cada fator de escala têm um piso e teto específicos. Por exemplo, um dos drivers do modelo é a confiabilidade exigida do software (reliability) e, quando no caso em análise houver indicação de risco à vida humana (condição de teto para o mesmo), estima-se haver um impacto de +26% ao esforço nominal estimado para sua entrega, já quando houver indicação de uma pequena inconveniência (condição de piso para o mesmo) o estima-se um impacto de -18%. Outro exemplo é o driver complexidade do produto (complexity) que pondera a maior ou menor complexidade em operações de controle, cálculos, operações dependentes de dispositivos, manipulação de dados e interface com usuário, o seu piso representa um impacto de -27% e, o seu teto, +74%.

Outra diferença é que o modelo do VAF é linear, enquanto no COCOMOII os fatores de escala têm um impacto exponencial no esforço nominal, em outras palavras, o modelo inclui economias e

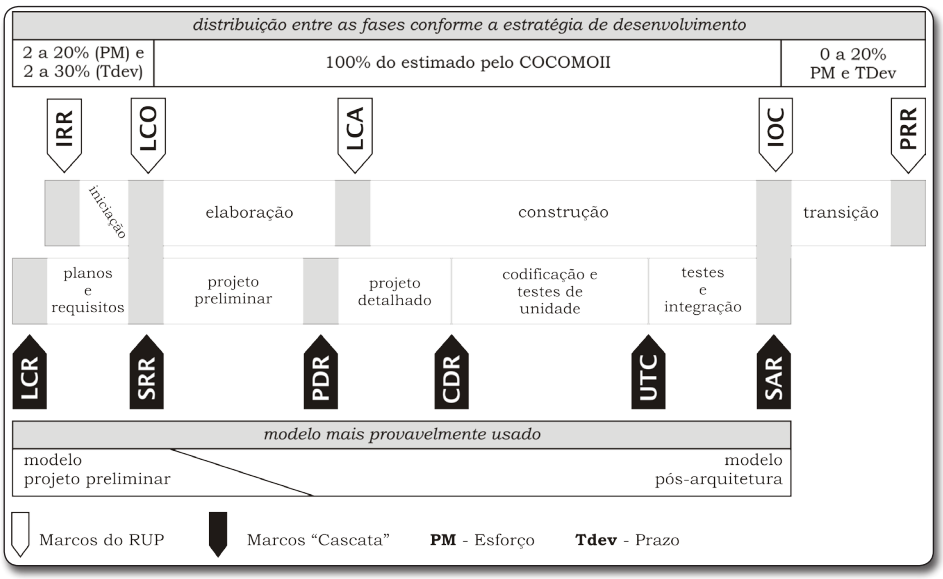


Figura 6. Momento mais provável em que determinado modelo seja o mais adequado.

deseconomias de escala. A seção "Relação entre escopo e esforço", do artigo da edição anterior, explica o papel da consideração dos fatores com impacto exponencial em um projeto de software.

Uma das funções do modelo é permitir aos seus usuários, ao analisar um caso, estabelecerem um ponto entre o respectivo piso e teto em um processo interpretativo das orientações fornecidas (subjetivo), mas ao contrário da determinação do VAF, esses percentuais são aplicados ao esforço e não ao tamanho e, portanto, há a possibilidade de amortecer essas diferentes interpretações ao ajustar as constantes de produtividade do modelo (desde que haja uma uniformidade quanto a essas interpretações, ao menos no âmbito da organização em que os dados estão sendo coletados, consolidados e analisados).

Além das aplicações dos fatores de escala e drivers de esforço em estimativas usando o COCOMOII, o conceito pode ser usado como uma referência ou inspiração para a criação de critérios de similaridade utilizados para análise das métricas de software em busca da criação de categorias de produtividade, ou na busca de indicadores estatísticos que apóiem o processo de planejamento.

Por exemplo, os autores tiveram uma experiência em que foi sugerida a criação de três critérios em que uma demanda deveria ser avaliada para fins de registro na base histórica: (a) Conhecimento do Usuário Sobre o Negócio (apelidada de CUSN); (b) Conhecimento do Analista Sobre a Aplicação (apelidada de CASA); e (c) foi utilizado um dos drivers do COCOMOII, especificamente, um que mede a familiaridade com o problema (cuja sigla é PREC) e cujas orientações para classificação estão na **Tabela 5** para fins de ilustração.

Característica	Muito Baixa	Nominal-Alta	Altíssima
Entendimento organizacional dos objetivos do produto	Geral	Considerável	Completo
Experiência trabalhando com sistemas de software afins	Moderada	Considerável	Extensiva
Desenvolvimento concorrente de novos equipamentos e procedimentos operacionais	Extensivo	Moderado	Algum
Necessidade de inovar arquiteturas de processamento de dados e algoritmos	Considerável	Algum	Mínimo

Tabela 5. Exemplo das orientações fornecidas pelo COCOMOII para classificação dos fatores de custo secundário, especificamente do fator de escala familiaridade (Precedentness – PREC).

A fórmula do COCOMOII – o coração dos modelos

A **Figura 7** apresenta a fórmula do COCOMOII. Dificilmente será necessário utilizá-la manualmente e está presente neste texto para fins didáticos e, o mais provável, é que o seu usuário utilize alguma ferramenta ou planilha para sua utilização. Nessa ferramenta, informará o tamanho do projeto e/ou de seus componentes e procederá a avaliação dos drivers de esforço e fatores de escala.

Apesar do estabelecido anteriormente, entender a fórmula ajuda a entender o funcionamento do modelo, suas entradas e seus produtos.

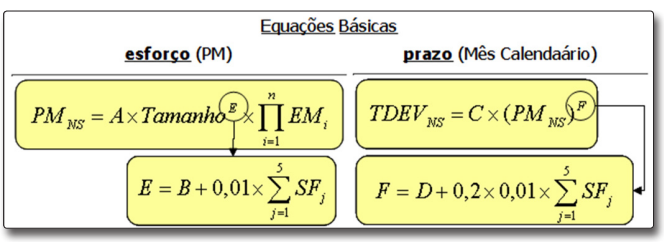


Figura 7. Fórmulas de esforço e prazo do COCOMOII.

Os produtos do modelo são representados pela variável TDEVNS, referindo-se ao prazo nominal expresso em meses calendário, e pela variável PMNS, referindo-se ao esforço nesse prazo nominal expresso em pessoas mês (vide explicação

na seção Relação entre Escopo e Esforço do artigo da edição anterior, para um melhor entendimento).

As constantes A, B, C e D representam a produtividade e o desempenho local, as duas primeiras no que se refere ao esforço e as duas últimas ao prazo. Antes de utilizar o modelo de estimativa, é necessário que essas constantes sejam calibradas às condições locais. Calibrar constantes de um modelo de estimativa é utilizar valores que sejam adequados ao contexto em que a estimativa será utilizada e aos objetivos que se deseja alcançar. A calibração é feita com base em experiências passadas compatíveis com esse contexto. O manual do COCOMOII recomenda que sejam utilizados pelo menos cinco casos para calibrar as constantes com efeitos multiplicativos A e C, e pelo menos dez projetos para calibrar as constantes com efeitos exponenciais B e D.

Os Drivers de Esforço

Na fórmula do COCOMOII (**Figura 7**), os drivers de esforço estão representados pela variável EMi e os fatores de escala pela variável SFi. No modelo Pós-arquitetura, serão dezessete os drivers de esforço **(Figura 8)** e no projeto preliminar, sete (**Figura 9**).

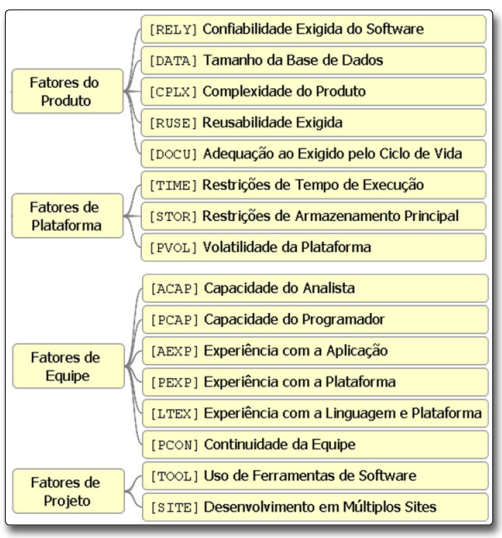


Figura 8. Drivers de esforço no modelo Pós-arquitetura; o driver de compressão de cronograma é único para todo o projeto e não é considerado na estimativa do esforço e prazos nominais.

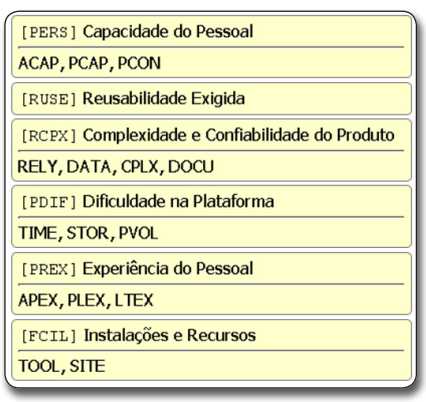


Figura 9. Drivers de esforço no modelo Projeto Preliminar e a correspondência entre os drivers de esforço no modelo Pós-Arquitetura.

EM$_i$	Muito Baixo	Baixo	Nominal	Alto	Muito Alto	Altíssimo
RELY	0.82	0.92	1.00	1.10	1.26	-
DATA	-	0.90	1.00	1.14	1.28	-
CPLX	0.73	0.87	1.00	1.17	1.34	1.74
RUSE		0.95	1.00	1.07	1.15	1.24
DOCU	0.81	0.91	1.00	1.11	1.23	-
TIME	-	-	1.00	1.11	1.29	1.63
STOR	-	-	1.00	1.05	1.17	1.46
PVOL	-	0.87	1.00	1.15	1.30	-
ACAP	1.42	1.19	1.00	0.85	0.71	-
PCAP	1.34	1.15	1.00	0.88	0.76	-
PCON	1.29	1.12	1.00	0.90	0.81	-
APEX	1.22	1.10	1.00	0.88	0.81	-
PLEX	1.19	1.09	1.00	0.91	0.85	-
LTEX	1.20	1.09	1.00	0.91	0.84	-
TOOL	1.17	1.09	1.00	0.90	0.78	-
SITE	1.22	1.09	1.00	0.93	0.86	0.80
SCED	1.43	1.14	1.00	1.00	1.00	-

Tabela 6. Drivers de esforço do modelo Pós-arquitetura

A **Tabela 6** apresenta os diferentes valores referentes aos drivers de esforço no modelo Pós-arquitetura. Esses pesos também não serão informados pelos usuários do modelo, estando mais provavelmente incluídos no software ou na planilha que automatiza o COCOMOII. Nela, estão os valores para os 17 drivers de esforço do modelo Pós-arquitetura. O driver de compressão de cronograma (SCED) é global para todas as partes do projeto, enquanto os demais são avaliados para cada parte isoladamente.

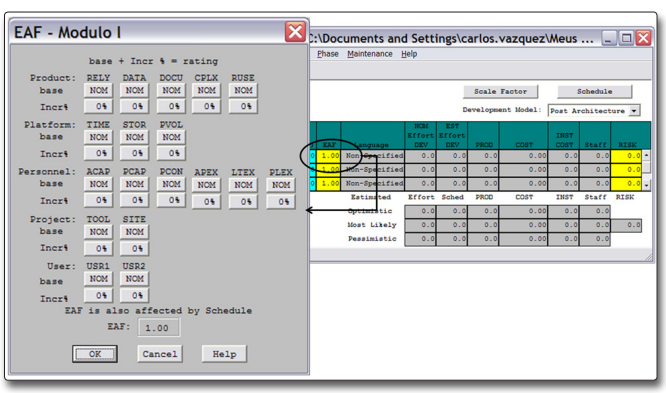


Figura 10. Formulário para informar a classificação dos drivers de esforço no modelo Pós-arquitetura ao estimar um projeto usando o COCOMOII.

A **Figura 10** ilustra como é feito o registro da análise dos diferentes drivers também no modelo Pós-arquitetura.

Lidando com a compressão de cronograma – o driver SCED

Na estimativa de esforço nominal (PMns), o driver de compressão de cronograma não é utilizado. O valor correspondente a sua avaliação na **Tabela 6** deve ser incluído conjuntamente

com os valores referentes aos outros drivers no resultado da multiplicação quando se deseja considerar os seus efeitos na estimativa de esforço (que, com isso, deixa de ser nominal). A determinação do valor correspondente deve observar a **Tabela 7**, conforme o grau de compressão exigido.

Muito Baixo	Baixo	Nominal	Alto	Muito Alto	Altíssimo
75% do nominal	85%	100%	130%	160%	-

Tabela 7. Critérios para qualificação do driver de compressão de cronograma.

Para a estimativa do prazo, considerando essa compressão do cronograma, acrescenta-se mais um componente à respectiva fórmula do prazo nominal (TDEVns), como descrito na **Fórmula 1**.

$$TDEV = C \times (PM_{ns})^{F} \times \frac{SCED\%}{100}$$

Por exemplo, se em determinado projeto deseja-se considerar uma compressão de cronograma de 25% e, portanto, o prazo estimado será 75% do prazo nominal, o esforço considerando essa compressão deve ser 43% superior ao esforço nominal.

Os Fatores de Escala

Os fatores de escala são os mesmos, independentemente do modelo utilizado ser o Pós-arquitetura ou Projeto Preliminar, onde a **Tabela 8** apresenta os valores para os cinco fatores de escala (SFi) e, a **Figura 11**, um exemplo de como eles são informados pelo usuário.

SF_j	Muito Baixa	Baixa	Nominal	Alta	Muito Alta	Impacto Máximo
PREC	6,20	4,96	3,72	2,48	1,24	
FLEX	5,07	4,05	3,04	2,03	1,01	1,26
RESL	7,07	5,65	4,24	2,83	1,41	1,39
TEAM	5,48	4,38	3,29	2,19	1,10	1,29
PMAT	7,80	6,24	4,68	3,12	1,56	1,43

Tabela 8. Valores para os cinco fatores de escala do COCOMOII.

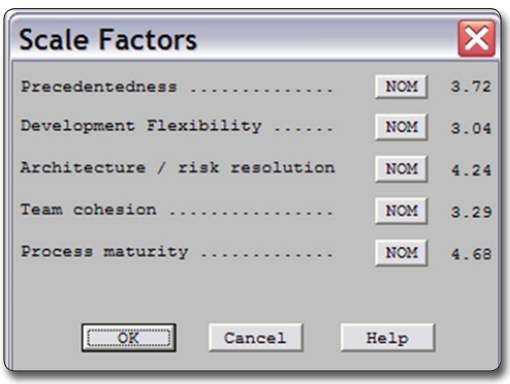


Figura 11. Formulário para informar a classificação dos fatores de escala ao estimar um projeto usando o COCOMOII.

O Fator Primário de Custo – O Tamanho

O tamanho é informado em milhares de linhas de código fonte (KSLOC). Essa unidade de tamanho é de difícil estimativa e a recomendação é que seja utilizada a estimativa em pontos de função para esse fim. Para isso, é necessário definir qual o fator de equivalência que será utilizado ao converter de pontos de função não ajustados para KSLOC.

O fator de equivalência não será constante, único, para qualquer aplicação ou projeto, mas para efeitos da estimativa, é um valor único necessário para converter de pontos de função para KSLOC. Usar a mediana de uma amostra é uma boa opção, já que a mediana é menos afetada que a média por valores extremos.

A **Tabela 9** apresenta uma série de projetos onde a mediana do fator de equivalência é de 99,49 SLOC/PF. A mediana divide uma amostra ordenada em duas partes, onde uma parte contém a metade dos elementos da amostra que são menores ou iguais à mediana, enquanto a segunda metade contém os demais elementos, maiores ou iguais à mediana.

Projeto	PF	SLOC	SLOC/PF
Projeto A	234	19.209	82,09
Projeto B	560	75.252	134,38
Projeto C	351	29.537	84,15
Projeto D	430	53.964	125,50
Projeto E	103	24.045	233,45
Projeto F	263	26.167	99,49
Projeto G	2.024	188.900	93,33
Projeto H	1.623	144.054	88,76
Projeto I	850	94.305	110,95

Tabela 9. Insumos para determinar o fator de equivalência entre a quantidade de pontos de função estimados e o fator de custo primário como esperado pelo COCOMOII.

A esse processo é dado o nome de "backfiring" e, quando utilizado no contexto do COCOMOII, os ruídos advindos da introdução desse passo no processo de estimativa tendem a ser anulados pelo processo de calibração das constantes de produtividade e desempenho. O processo inverso de converter de KSLOC para pontos de função deve ser evitado na medida em que normalmente não há a amortização desses ruídos como no COCOMOII.

Portanto, será utilizado como tamanho na fórmula do COCOMOII o valor de 49,475 KSLOC referentes a um projeto cujo tamanho funcional seja estimado em 500 PF (500 PF x 99,49 SLOC/PF / 1000).

O wCOCOMOII também prevê a consideração de reuso e código gerado automaticamente que não serão tratados neste texto. A **Figura 12** ilustra como as informações de tamanho são imputadas no COCOMOII com a aplicação desenvolvida pela Universidade da Califórnia do Sul, apresentada anteriormente. Nela há o formulário para informar o tamanho, e o campo REVL corresponde ao percentual de retrabalho advindo da volatilidade dos requisitos.

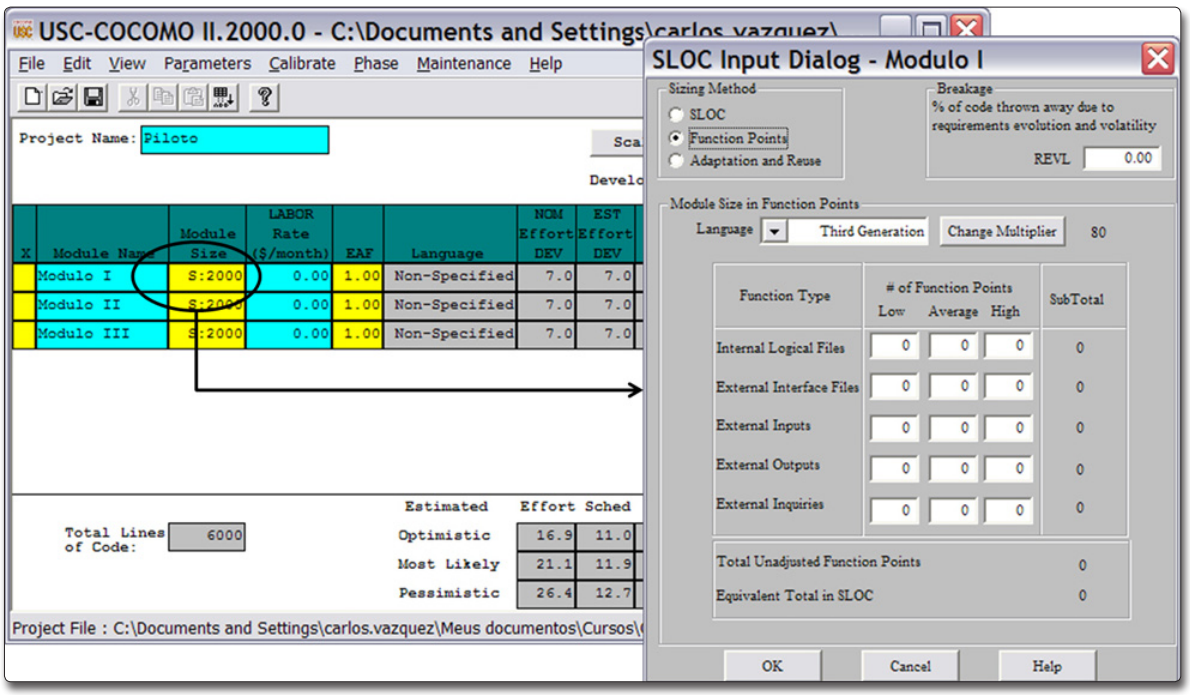


Figura 12. Entrada dos dados de tamanho – o fator primário de custos.

A Necessidade de Calibrar do Modelo

Este texto abordou até este ponto a utilização do modelo COCOMOII a partir de parâmetros de tamanho e da análise qualitativa de uma série de aspectos relacionados ao projeto, produto, usuário e processo. A partir dos mesmos, o modelo produz estimativas de esforço, prazo e tamanho da equipe (razão entre os dois itens anteriores). Mas não se deve utilizar o modelo simplesmente a partir disso!

No modelo de estimativa discutido no artigo da edição anterior, referente ao deslocamento entre o Rio de Janeiro e Niterói, considerou-se uma velocidade média de 25 Km/h. Nesse artigo foi definido um modelo de estimativa que pode ser descrito como "A estimativa de tempo para deslocar-se de um ponto A até um ponto B é a razão da distância e a velocidade média de 25 Km/h".

Esse modelo está adequado para o ponto A ser o Centro do Rio de Janeiro e o ponto B, o Centro de Niterói, em um deslocamento feito utilizando um automóvel. Está adequado se o ponto A for a Barra Funda em São Paulo e o Ponto B o Morumbi? Talvez outra velocidade média seja mais representativa que não os 25 Km/h utilizados no modelo. Descobrir essa nova velocidade media é calibrar o modelo.

Calibrar o modelo envolve definir um objetivo a ser alcançado como, por exemplo, diminuir o erro médio. Para tanto, são necessárias fotografias de eventos passados e que sejam utilizados na otimização buscando esse objetivo. A ferramenta que utilizamos até o momento para ilustrar o COCOMOII permite a calibração de suas constantes de esforço a partir da informação do realizado pelo usuário. A **Figura 13** ilustra como isso é feito e apresenta o formulário para informar o esforço e o prazo realizados de um projeto onde os parâmetros do modelo foram analisados e

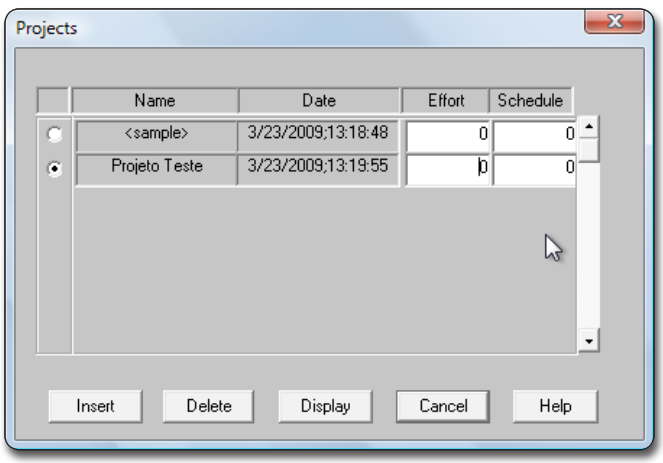


Figura 13. Informação do esforço e prazo realizados para calibrar o modelo.

informados (Tamanho, Drives de Esforço, Fatores de Escala, entre outros).

Além do software utilizado na **Figura 13**, outros também gratuitos estão disponíveis como o CALICO da SoftStar Systems. O uso de uma ferramenta para esse fim não é um requisito, desde que haja entendimento sobre as diferentes estratégias de medir o erro de modelos de estimativa.

Com uma planilha eletrônica é possível a realização de ensaios onde diferentes critérios de avaliação são utilizados, e o responsável pelo trabalho pode escolher as constantes que mais se adéqüem às suas necessidades. Ao proceder a uma calibração manual é fundamental normalizar o esforço e prazo de tal forma que sejam considerados os seus valores nominais, desconsiderando os efeitos da compressão de cronograma e que todos os prazos expressos em pessoa-mês sejam referentes a um mesmo número de homens-hora por mês-calendário.

Discutir sobre a calibração de modelos de estimativa como o

COCOMOII envolve apresentar os diferentes meios de medir o erro como, por exemplo: (a) a Média da Magnitude do Erro Relativo - estimada, realizada e balanceada; (b) o grau de predição do modelo; (c) a soma dos quadrados dos erros. O espaço disponível para este artigo não permite fazer isso, que será tratado em uma nova oportunidade.

Considerações Finais

Muitos vêem o COCOMOII como um software da Universidade do Sul da Califórnia. Com base no exposto neste texto, percebe-se que ele é muito mais do que isso e, pelo contrário, o software em questão é apenas uma das implementações do modelo, e das mais rudimentares. Outros pacotes comerciais mais completos e com dados de calibração mais abrangentes estão disponíveis como o Cost Xpert e o CoStar, contudo o principal valor está na rigorosa definição de premissas e em sua simplicidade. ●

Referências

Boehm, B. et al., "Software Cost Estimation With COCOMO II", Prentice Hall, 2000.

Aguiar, M., "COCOMOII Local Calibration Using Function Points", http://sunset.usc.edu/events/2005/COCOMO/presentations/CIILocalCalibrationPaper.pdf

Ligett, D., "Techniques for Calibrating COCOMO Estimating Models", http://www.softstarsystems.com/calico.htm.

Maxwell, K. D., "Applied Statistics for Software Managers", Prentice Hall, 2002.

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

Introdução ao Project Management Body of Knowledge (PMBoK)

De que se trata o artigo?
Este artigo descreve os principais conceitos do PMBoK, seus grupos de processos e áreas de conhecimento.

Para que serve?
Este artigo serve como uma introdução ao PMBoK.

Em que situação o tema é útil?
Gerenciamento de Projetos.

Paulo Augusto Oyamada Tamaki
paulotamaki@gmail.com
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 *Project Management Body of Knowledge* (PMBoK) é um padrão aprovado pela ANSI, e reconhecido pelo IEEE através do padrão 1490-1998. Ele descreve um conjunto de práticas consolidadas sobre gerenciamento de projetos que podem ser aplicadas em qualquer tipo de projetos (incluindo processos de desenvolvimento de software).

Neste cenário, este artigo descreve os principais conceitos do PMBoK, seus grupos de processos e áreas de conhecimento.

É importante destacar que neste artigo trabalharemos com terceira edição do PMBOK. Em um artigo futuro serão destacas a principais alterações definidas na nova versão do PMBOK.

Principais Conceitos e Definições do PMBoK

O PMBoK define projeto como "um empreendimento temporário cujo objetivo é criar um produto, serviço ou resultado distinto e único".

Um projeto é temporário, pois apresenta datas de início e término definidas. O

término do projeto pode ser determinado por vários motivos, por exemplo: cumprimento dos objetivos determinados, abandono do mesmo. Cada projeto produz um produto único, mesmo havendo semelhanças entre diversos projetos, cada projeto possui metas, diretrizes, e envolvidos distintos. Além disso, um projeto tem como característica a elaboração progressiva. Na elaboração progressiva, a especificação do projeto vai sendo refinada ao longo do ciclo de vida do mesmo.

Para o PMBoK, o gerenciamento de projeto é "a aplicação de conhecimento, habilidades, ferramentas e técnicas atividades do projeto a fim de atender aos seus requisitos".

O ciclo de vida de um projeto é dividido em diversas fases para melhorar o controle e o gerenciamento do projeto. Cada fase é marcada pela conclusão de um ou mais produtos e cada uma delas possui um ponto de revisão. O ponto de revisão tem como objetivo determinar a continuidade ou não do projeto, e a detecção e correção de defeitos.

Tipicamente, um projeto tem uma fase inicial, onde é definido o escopo do projeto. Uma ou mais fases intermediárias, onde o alvo do projeto é planejado e realizado. E uma fase final, onde o produto é aceito e entregue para o cliente (conforme apresentado na **Figura 1**).

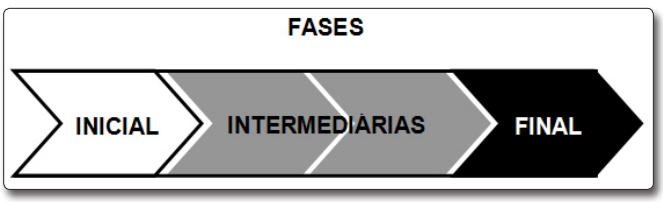


Figura 1. Exemplo de Ciclo de Vida de um Projeto.

Processos do Gerenciamento de Projeto

Cada projeto é composto por uma série de processos. Estes processos podem ser organizados em cinco grupos de processos que possuem as seguintes metas:

- **Grupo de Processos de Iniciação**: Reconhecer que um projeto ou fase deve começar e se comprometer com a sua execução;
- **Grupo de Processos de Planejamento**: Definir e refinar os objetivos. Planejar e manter um esquema de trabalho viável para atingir aqueles objetivos de negócio que determinaram a existência do projeto;
- **Grupo de Processos de Execução**: Coordenar pessoas e outros recursos para realizar o que foi planejado;
- **Grupo de Processos de Monitoramento e Controle**: Assegurar que os objetivos do projeto estão sendo atingidos, através do monitoramento constante e da avaliação do seu progresso, tomando ações corretivas quando necessárias;
- **Grupo de Processos de Encerramento**: Formalizar a aceitação do projeto ou fase e fazer o seu encerramento de forma organizada.

O relacionamento entre os grupos de processos é apresentado na **Figura 2**, onde as setas representam os fluxos de entrada e saída entre os grupos de processos.

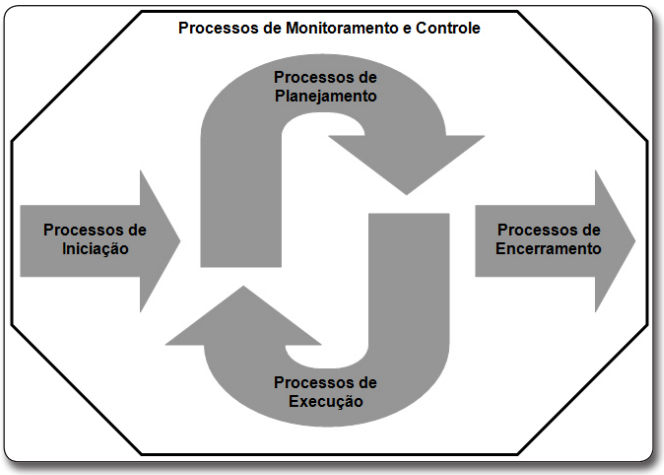


Figura 2. Relacionamentos entre os Grupos de Processos.

Em cada uma das fases do projeto, este ciclo de processos é realizado. Cabe ao Gerente de Projeto determinar quais dos processos existentes deverão ser realizados em uma determinada fase.

Cada processo dentro de um grupo de processos possui as seguintes características: uma descrição do processo, indicando seus objetivo e características; um conjunto de entradas, que são as informações utilizadas pelo processo; um conjunto de ferramentas e técnicas, que servem como guias ou diretrizes de como implementar o processo; e um conjunto de saídas, que são os produtos ou resultados deste processo. A **Figura 3** esquematiza o detalhamento de um processo no PMBoK.

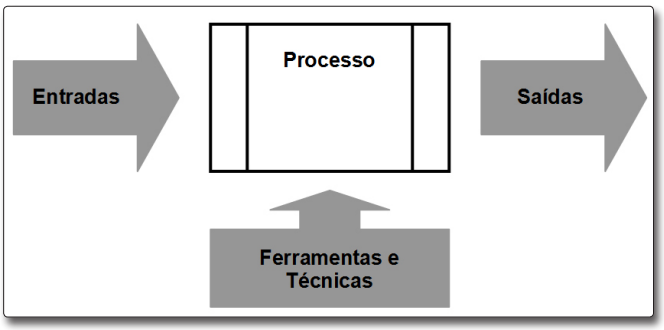


Figura 3. Esquema de um processo no PMBoK.

As Áreas de Conhecimento

O conjunto de práticas apresentadas no PMBoK são divididos em nove áreas de conhecimento. Cada uma das áreas de conhecimento descreve os conhecimentos e práticas de gerenciamento de projetos em termos de um ou mais processos. Cada processo é detalhado em entradas, saídas, ferramentas e técnicas. Entradas e saídas são documentos ou itens documentáveis. Ferramentas e técnicas são os mecanismos que transformam as entradas nas saídas.

A seguir são apresentadas as áreas de conhecimento do PMBoK e os processos que cada uma delas engloba.

Para facilitar a leitura e o entendimento do texto, a descrição

dos processos de cada área de conhecimento não detalha o conteúdo de cada entrada e cada saída do processo, e, se necessário, uma observação a respeito da entrada ou saída naquele processo. Todas as entradas e saídas dos processos do PMBoK estão descritas no **Quatro 1**.

Gerenciamento da Integração do Projeto

Esta área de conhecimento agrupa os processos necessários para assegurar que os diversos elementos do projeto sejam adequadamente coordenados.

Desenvolver o Termo de Abertura do Projeto

Processo responsável pelo desenvolvimento do termo de abertura do projeto que autoriza formalmente um projeto ou uma fase do projeto. Na fase inicial, este processo é responsável pela elaboração do termo. Em fases subseqüentes, este termo inicial é validado, ou, se necessário, atualizado. A **Tabela 1** apresenta as entradas e saídas deste processo.

Entrada
Contrato
Declaração do Trabalho do Projeto
Fatores Ambientais da Empresa
Ativos de Processos Organizacionais
Saída
Termo de Abertura do Projeto

Tabela 1. Entradas e Saídas do Processo: Desenvolver o Termo de Abertura do Projeto.

Desenvolver a Declaração do Escopo Preliminar do Projeto

Processo responsável pelo desenvolvimento da declaração do escopo preliminar do projeto que fornece uma descrição de alto nível do escopo. A **Tabela 2** apresenta as entradas e saídas deste processo.

Entrada
Termo de Abertura do Projeto
Declaração do Trabalho do Projeto
Fatores Ambientais da Empresa
Ativos de Processos Organizacionais
Saída
Declaração do Escopo Preliminar do Projeto

Tabela 2. Entradas e Saídas do Processo: Desenvolver a Declaração do Escopo Preliminar do Projeto.

Desenvolver o Plano de Gerenciamento do Projeto

Processo responsável pela documentação das ações necessárias para definir, preparar, integrar e coordenar todos os planos auxiliares em um plano de gerenciamento do projeto.

Este processo também é responsável pela elaboração do "Plano de Gerenciamento de Custos" que é um dos planos auxiliares do projeto que tem como finalidade estabelecer os critérios para planejar, estruturar, estimar, orçar, e controlar os custos do projeto. De maneira geral, a elaboração dos outros planos auxiliares é detalhada mais precisamente nas outras áreas de conhecimento do PMBoK, apenas o gerenciamento de custos referencia este processo como responsável pela elaboração deste plano. A **Tabela 3** apresenta as entradas e as saídas deste processo.

Entrada
Declaração do Escopo Preliminar do Projeto
Processos de Gerenciamento de Projetos
Fatores Ambientais da Empresa
Ativos de Processos Organizacionais
Saída
Plano de Gerenciamento do Projeto

Tabela 3. Entradas e Saídas do Processo: Desenvolver o Plano do Gerenciamento do Projeto.

Orientar e Gerenciar a Execução do Projeto

Processo responsável pela execução do trabalho definido no plano de gerenciamento do projeto com o fim de atingir os requisitos do projeto definidos na declaração do escopo do projeto. A **Tabela 4** apresenta as entradas e as saídas deste processo.

Entrada
Plano de Gerenciamento do Projeto
Ações Corretivas Aprovadas
Ações Preventivas aprovadas
Solicitações de Mudanças Aprovadas
Reparo de Defeito Aprovado
Reparo de Defeito Validado
Procedimento de Encerramento Administrativo
Saída
Entregas
Mudanças Solicitadas
Solicitações de Mudanças Implementadas
Ações Corretivas Implementadas
Ações Preventivas Implementadas
Reparo de Defeito Implementado
Informações sobre o Desempenho do Trabalho

Tabela 4. Entradas e Saídas do Processo: Orientar e Gerenciar a Execução do Projeto.

Monitorar e Controlar o Trabalho do Projeto

Processo responsável pelo monitoramento e controle dos processos utilizados para iniciar, planejar, executar e encerrar um projeto com o fim de atender aos objetivos de desempenho definidos no plano de gerenciamento do projeto. A **Tabela 5** apresenta as entradas e saídas deste processo.

Entrada
Plano de Gerenciamento do Projeto
Informações sobre o Desempenho do Trabalho
Solicitações de Mudanças Rejeitadas
Saída
Ações Corretivas Recomendadas
Ações Preventivas Recomendadas
Previsões
Reparos de Defeitos Recomendado
Mudanças Solicitadas

Tabela 5. Entradas e Saídas do Processo: Monitorar e Controlar o Trabalho do Projeto.

Controle Integrado de Mudanças

Processo responsável pela revisão de todas as solicitações de mudança, pela aprovação de mudanças e pelo controle de mudanças nas entregas e nos ativos de processos organizacionais. A **Tabela 6** apresenta as entradas e as saídas deste processo. Observe que nesta tabela existem algumas linhas destacadas em azul. Isto indica que os artefatos destacados já foram trabalhados em outros processos e estão sendo atualizados nesta etapa.

Entrada
Plano de Gerenciamento do Projeto
Mudanças Solicitadas
Informações sobre o Desempenho do Trabalho
Ações Preventivas Recomendadas
Ações Corretivas Recomendadas
Reparo de Defeitos Recomendado
Entregas
Saída
Solicitações de Mudanças Aprovadas
Solicitações de Mudanças Rejeitadas
Plano de Gerenciamento de Projeto
Declaração do Escopo do Projeto
Ações Corretivas Aprovadas
Ações Preventivas Aprovadas
Reparo de Defeito Aprovado
Reparo de Defeito Validado
Entregas

Tabela 6. Entradas e Saídas do Processo: Controle Integrado de Mudanças.

Encerrar o Projeto

Processo responsável pela finalização de todas as atividades em todos os grupos de processos de gerenciamento de projetos com o fim de encerrar formalmente o projeto ou uma de suas fases. A **Tabela 7** apresenta as entradas e as saídas deste processo. Observe que nesta tabela existe uma linha destacada em azul. Assim como na tabela anterior, isto indica que o artefato destacado já foi trabalhado em outros processos e está sendo atualizado nesta etapa.

Entrada
Plano de Gerenciamento do Projeto
Documentação do Contrato
Fatores Ambientais da Empresa
Ativos de Processos Organizacionais
Informações sobre o Desempenho do Trabalho
Entregas
Saída
Procedimento de Encerramento Administrativo
Procedimento de Encerramento de Contratos
Produto, Serviço ou Resultado Final
Ativos de Processos Organizacionais

Tabela 7. Entradas e Saídas do Processo: Encerrar o Projeto.

Gerenciamento do Escopo do Projeto

Esta área de conhecimento agrupa os processos necessários para assegurar que o projeto contemple todo o trabalho requerido, e nada mais que o trabalho requerido, para completar o projeto com sucesso.

Planejamento do Escopo

Processo responsável pela criação de um plano de gerenciamento do escopo do projeto que documenta como o escopo do projeto é definido, verificado e controlado e como a estrutura analítica do projeto (EAP) é criada e definida. A **Tabela 8** apresenta as entradas e as saídas deste processo.

Entrada
Fatores Ambientais da Empresa
Ativos de Processos Organizacionais
Termo de Abertura do Projeto
Declaração do Escopo Preliminar do Projeto
Plano de Gerenciamento do Projeto
Saída
Plano de Gerenciamento do Escopo do Projeto

Tabela 8. Entradas e Saídas do Processo: Planejamento do Escopo.

Definição do Escopo

Processo responsável pelo desenvolvimento de uma declaração do escopo detalhada do projeto como a base para futuras decisões do projeto. A **Tabela 9** apresenta as entradas e as saídas deste processo.

Criar EAP

Processo responsável pela subdivisão das principais entregas do projeto e do trabalho do projeto em componentes menores e mais facilmente gerenciáveis. A **Tabela 10** apresenta as entradas e as saídas deste processo.

Entrada
Ativos de Processos Organizacionais
Termo de Abertura do Projeto
Declaração do Escopo Preliminar do Projeto
Plano de Gerenciamento do Escopo do Projeto
Solicitações de Mudanças Aprovadas
Saída
Declaração do Escopo do Projeto
Mudanças Solicitadas
Plano de Gerenciamento do Escopo do Projeto

Tabela 9. Entradas e Saídas do Processo: Definição do Escopo.

Entrada
Ativos de Processos Organizacionais
Declaração do Escopo do Projeto
Plano de Gerenciamento do Escopo do Projeto
Solicitações de Mudanças Aprovadas
Saída
Declaração do Escopo do Projeto
Estrutura Analítica do Projeto
Dicionário da EAP
Linha de Base do Escopo
Plano de Gerenciamento do Escopo do Projeto
Mudanças Solicitadas

Tabela 10. Entradas e Saídas do Processo: Criar EAP.

Verificação do Escopo

Processo responsável pela formalização da aceitação das entregas do projeto terminadas. A **Tabela 11** apresenta as entradas e as saídas deste processo.

Entrada
Declaração do Escopo do Projeto
Dicionário da EAP
Plano de Gerenciamento do Escopo do Projeto
Entregas
Saída
Entregas Aceitas
Mudanças Solicitadas
Ações Corretivas Recomendadas

Tabela 11. Entradas do Processo: Verificação do Escopo.

Controle do Escopo

Processo responsável pelo controle das mudanças no escopo do projeto. A **Tabela 12** apresenta as entradas e as saídas deste processo.

Gerenciamento do Tempo do Projeto

Esta área de conhecimento agrupa os processos necessários para assegurar que o projeto termine dentro do prazo previsto.

Entrada
Declaração do Escopo do Projeto
Estrutura Analítica do Projeto
Dicionário da EAP
Plano de Gerenciamento do Escopo do Projeto
Relatórios de Desempenho
Solicitações de Mudanças Aprovadas
Informações sobre o Desempenho do Trabalho
Saída
Declaração do Escopo do Projeto
Estrutura Analítica do Projeto
Dicionário da EAP
Linha de Base do Escopo
Mudanças Solicitadas
Ações Corretivas Recomendadas
Ativos de Processos Organizacionais
Plano de Gerenciamento do Projeto

Tabela 12. Entradas e Saídas do Processo: Controle do Escopo.

Definição da Atividade

Processo responsável pela identificação das atividades específicas do cronograma que precisam ser realizadas para produzir as várias entregas do projeto. A **Tabela 13** apresenta as entradas e as saídas deste processo.

Entrada
Fatores Ambientais da Empresa
Ativos de Processos Organizacionais
Declaração do Escopo do Projeto
Estrutura Analítica do Projeto
Dicionário da EAP
Plano de Gerenciamento do Projeto
Saída
Lista de Atividades
Atributos da Atividade
Lista de Marcos
Mudanças Solicitadas

Tabela 13. Entradas e Saídas do Processo: Definição da Atividade.

Seqüenciamento de Atividades

Processo responsável pela identificação e documentação das dependências entre as atividades do cronograma. A **Tabela 14** apresenta as entradas e as saídas deste processo.

Estimativa de Recursos da Atividade

Processo responsável pela estimativa do tipo e das

Entrada
Declaração do Escopo do Projeto
Lista de Atividades
Atributos da Atividade
Lista de Marcos
Solicitação de Mudanças Aprovadas
Saída
Diagramas de Rede do Cronograma do Projeto
Lista de Atividades
Atributos da Atividade
Mudanças Solicitadas

Tabela 14. Entradas e Saídas do Processo: Seqüenciamento de Atividades.

quantidades de recursos necessários para realizar cada atividade do cronograma. A **Tabela 15** apresenta as entradas e as saídas deste processo.

Entrada
Fatores Ambientais da Empresa
Ativos de Processos Organizacionais
Lista de Atividades
Atributos da Atividade
Disponibilidade de Recursos
Plano de Gerenciamento de Projeto
Saída
Recursos Necessários para a Atividade
Atributos da Atividade
Estrutura Analítica dos Recursos
Calendário de Recursos
Mudanças Solicitadas

Tabela 15. Entradas e Saídas do Processo: Estimativa de Recurso da Atividade.

Estimativa de Duração da Atividade

Processo responsável pela estimativa do número de períodos de trabalho que serão necessários para terminar as atividades individuais do cronograma. A **Tabela 16** apresenta as entradas e as saídas deste processo.

Desenvolvimento do Cronograma

Processo responsável pela análise dos recursos necessários, restrições do cronograma, durações e seqüências de atividades para criar o cronograma do projeto. A **Tabela 17** apresenta as entradas e as saídas deste processo.

Controle do Cronograma

Processo responsável pelo controle das mudanças no cronograma do projeto. A **Tabela 18** apresenta as entradas e as saídas deste processo.

Entrada
Fatores Ambientais da Empresa
Ativos de Processos Organizacionais
Declaração do Escopo do Projeto
Lista de Atividades
Atributos da Atividade
Recursos Necessários para a Atividade
Calendário de Recursos
Plano de Gerenciamento de Projeto
Saída
Estimativas de Duração da Atividade
Atributos da Atividade

Tabela 16. Entradas e Saídas do Processo: Estimativa de Duração da Atividade.

Entrada
Ativos de Processos Organizacionais
Declaração do Escopo do Projeto
Lista de Atividades
Atributos da Atividade
Diagramas de Rede do Cronograma do Projeto
Recursos Necessários para a Atividade
Calendário de Recursos
Estimativas de Duração da Atividade
Plano de Gerenciamento de Projeto
Saída
Cronograma do Projeto
Dados do Modelo do Cronograma
Linha de Base do Cronograma
Recursos Necessários para a Atividade
Atributos da Atividade
Calendário de Projeto
Mudanças Solicitadas
Plano de Gerenciamento de Projeto

Tabela 17. Entradas e Saídas do Processo: Desenvolvimento do Cronograma.

Gerenciamento do Custo do Projeto

Esta área de conhecimento agrupa os processos necessários para assegurar que o projeto termine dentro do orçamento aprovado.

Estimativa de Custos

Processo responsável pelo desenvolvimento de uma estimativa dos custos dos recursos necessários para terminar as atividades do projeto. A **Tabela 19** apresenta as entradas e as saídas deste processo.

Orçamentação

Processo responsável pela agregação dos custos estimados de atividades individuais ou pacotes de trabalho para estabelecer

Entrada
Plano de Gerenciamento do Cronograma
Linha de Base do Cronograma
Relatórios de Desempenho
Solicitações de Mudança Aprovadas
Saída
Dados do Modelo do Cronograma
Linha de Base do Cronograma
Medições de Desempenho
Mudanças Solicitadas
Ações Corretivas Recomendadas
Ativos de Processos Organizacionais
Lista de Atividades
Atributos da Atividade
Plano de Gerenciamento de Projeto

Tabela 18. Entradas e Saídas do Processo: Controle do Cronograma.

Entrada
Fatores Ambientais da Empresa
Ativos de Processos Organizacionais
Declaração do Escopo do Projeto
Estrutura Analítica do Projeto
Dicionário da EAP
Plano de Gerenciamento do Projeto
Saída
Estimativas de Custos da Atividade
Detalhes que dão suporte as Estimativas de Custos da Atividade
Mudanças Solicitadas
Plano de Gerenciamento de Custos

Tabela 19. Entradas e Saídas do Processo: Estimativa de Custos.

uma linha de base dos custos. A **Tabela 20** apresenta as entradas e as saídas deste processo.

Controle de Custos

Processo responsável pelo controle dos fatores que criam as variações de custos e controle das mudanças no orçamento do projeto. A **Tabela 21** apresenta as entradas e as saídas deste processo.

Gerenciamento da Qualidade do Projeto

Esta área de conhecimento agrupa os processos necessários para assegurar que as necessidades que originaram o desenvolvimento do projeto sejam atendidas.

Planejamento da Qualidade

Processo responsável pela identificação dos padrões de qualidade relevantes para o projeto e determinação de como satisfazê-los. A **Tabela 22** apresenta as entradas e as saídas deste processo.

Entrada
Declaração do Escopo do Projeto
Estrutura Analítica do Projeto
Dicionário da EAP
Estimativas de Custos da Atividade
Detalhes que dão suporte a Estimativas de Custo da Atividade
Cronograma do Projeto
Calendário de Recursos
Contrato
Plano de Gerenciamento do Projeto
Saída
Linha de Base dos Custos
Necessidade de Financiamento do Projeto
Plano de Gerenciamento de Custos
Mudanças Solicitadas

Tabela 20. Entradas e Saídas do Processo: Orçamentação.

Entrada
Linha de Base dos Custos
Necessidade de Financiamento do Projeto
Relatórios de Desempenho
Informações sobre o Desempenho do Trabalho
Solicitações de Mudanças Aprovadas
Plano de Gerenciamento do Projeto
Saída
Estimativas de Custos da Atividade
Linha de Base dos Custos
Medições de Desempenho
Previsão de Término
Mudanças Solicitadas
Ações Corretivas Recomendadas
Ativos de Processos Organizacionais
Plano de Gerenciamento de Projeto

Tabela 21. Entradas e Saídas do Processo: Controle de Custos.

Realizar a Garantia da Qualidade

Processo responsável pela aplicação das atividades de qualidade planejadas e sistemáticas para garantir que o projeto emprega todos os processos necessários para atender aos requisitos. A **Tabela 23** apresenta as entradas e as saídas deste processo.

Realizar o Controle da Qualidade

Processo responsável pelo monitoramento de resultados específicos do projeto a fim de determinar se eles estão de acordo com os padrões relevantes de qualidade e identificação de maneiras de eliminar as causas de um desempenho insatisfatório. A **Tabela 24** apresenta as entradas e as saídas deste processo.

Entrada
Fatores Ambientais da Empresa
Ativos de Processos Organizacionais
Declaração do Escopo do Projeto
Plano de Gerenciamento do Projeto
Saída
Plano de Gerenciamento de Qualidade
Métricas da Qualidade
Listas de Verificação da Qualidade
Plano de Melhorias no Processo
Linha de Base da Qualidade
Plano de Gerenciamento de Projeto

Tabela 22. Entradas e Saídas do Processo: Planejamento da Qualidade.

Entrada
Plano de Gerenciamento da Qualidade
Métricas da Qualidade
Plano de Melhorias no Processo
Informações sobre o Desempenho do Trabalho
Solicitações de Mudança Aprovadas
Medições de Controle da Qualidade
Solicitações de Mudança Implementadas
Ações Corretivas Implementadas
Reparo de Defeito Implementado
Ações Preventivas Implementadas
Saída
Mudanças Solicitadas
Ações Corretivas Recomendadas
Ativos de Processos Organizacionais
Plano de Gerenciamento de Projeto

Tabela 23. Entradas e Saídas do Processo: Realizar a Garantia da Qualidade.

Gerenciamento dos Recursos Humanos do Projeto

Esta área de conhecimento agrupa os processos necessários para proporcionar a melhor utilização das pessoas envolvidas no projeto.

Planejamento de Recursos Humanos

Processo responsável pela identificação e documentação de funções, responsabilidades e relações hierárquicas do projeto, além da criação do plano de gerenciamento de pessoal. A **Tabela 25** apresenta as entradas e as saídas deste processo.

Contratar ou Mobilizar a Equipe do Projeto

Processo responsável pela obtenção dos recursos humanos necessários para terminar o projeto. A **Tabela 26** apresenta as entradas e as saídas deste processo.

Desenvolver a Equipe do Projeto

Processo responsável pela melhoria de competências e interação de membros da equipe para aprimorar o desempenho do projeto. A **Tabela 27** apresenta as entradas e as saídas deste processo.

Entrada
Plano de Gerenciamento da Qualidade
Métricas da Qualidade
Listas de Verificação da Qualidade
Ativos de Processos Organizacionais
Informações sobre o Desempenho do Trabalho
Solicitações de Mudança Aprovadas
Entregas
Saída
Medições de Controle da Qualidade
Reparo de Defeito Validado
Linha de Base da Qualidade
Ações Corretivas Recomendadas
Ações Preventivas Recomendadas
Mudanças Solicitadas
Reparo de Defeito Recomendado
Ativos de Processos Organizacionais
Entregas Validadas
Plano de Gerenciamento de Projeto

Tabela 24. Entradas e Saídas do Processo: Realizar o Controle da Qualidade.

Entrada
Fatores Ambientais da Empresa
Ativos de Processos Organizacionais
Plano de Gerenciamento de Projeto
Saída
Funções e Responsabilidades
Organogramas do Projeto
Plano de Gerenciamento de Pessoal

Tabela 25. Entradas e Saídas do Processo: Planejamento de Recursos Humanos.

Entrada
Fatores Ambientais da Empresa
Ativos de Processos Organizacionais
Funções e Responsabilidades
Organogramas do Projeto
Plano de Gerenciamento de Pessoal
Saída
Designações de Pessoal para o Projeto
Disponibilidade de Recursos
Plano de Gerenciamento de Pessoal

Tabela 26. Entradas e Saídas do Processo: Contratar ou Mobilizar a Equipe do Projeto.

Gerenciar a Equipe do Projeto

Processo responsável pelo acompanhamento do desempenho

Entrada
Designações de Pessoal para o Projeto
Plano de Gerenciamento de Pessoal
Disponibilidade de Recursos
Saída
Avaliação do Desempenho da Equipe

Tabela 27. Entradas e Saídas do Processo: Desenvolver a Equipe do Projeto.

de membros da equipe, fornecimento de *feedback*, resolução de problemas e coordenação de mudanças para melhorar o desempenho do projeto. A **Tabela 28** apresenta as entradas e as saídas deste processo.

Entrada
Ativos de Processos Organizacionais
Designações de Pessoal para o Projeto
Funções e Responsabilidades
Organogramas do Projeto
Plano de Gerenciamento de Pessoal
Avaliação do Desempenho da Equipe
Informações sobre o Desempenho do Trabalho
Relatórios de Desempenho
Saída
Mudanças Solicitadas
Ações Corretivas Recomendadas
Ações Preventivas Recomendadas
Ativos de Processos Organizacionais
Plano de Gerenciamento do Projeto

Tabela 28. Entradas e Saídas do Processo: Gerenciar a Equipe do Projeto.

Gerenciamento das Comunicações do Projeto

Esta área de conhecimento agrupa os processos necessários para assegurar que a geração, captura, distribuição, armazenamento e apresentação das informações do projeto sejam feitas da maneira adequada e no tempo certo.

Planejamento das Comunicações

Processo responsável pela determinação das necessidades de informações e comunicações das partes interessadas no projeto. A **Tabela 29** apresenta as entradas e as saídas deste processo.

Distribuição das Informações

Processo responsável pela disponibilização das informações necessárias às partes interessadas no projeto no momento adequado. A **Tabela 30** apresenta as entradas e as saídas deste processo.

Entrada
Fatores Ambientais da Empresa
Ativos de Processos Organizacionais
Declaração do Escopo do Projeto
Plano de Gerenciamento do Projeto
Saída
Plano de Gerenciamento das Comunicações

Tabela 29. Entradas e Saídas do Processo: Planejamento das Comunicações.

Entrada
Plano de Gerenciamento das Comunicações
Saída
Ativos de Processos Organizacionais
Mudanças Solicitadas

Tabela 30. Entradas e Saídas do Processo: Distribuição das Informações.

Relatório de Desempenho

Processo responsável pela coleta e distribuição das informações sobre o desempenho. Isso inclui o relatório de andamento, medição do progresso e previsão. A **Tabela 31** apresenta as entradas e as saídas deste processo.

Entrada
Informações sobre o Desempenho do Trabalho
Medições de Desempenho
Previsão do Término
Medições de Controle da Qualidade
Plano de Gerenciamento do Projeto
Solicitações das Mudanças Aprovadas
Entregas
Saída
Relatórios de Desempenho
Previsões
Mudanças Solicitadas
Ações Corretivas Recomendadas
Ativos de Processos Organizacionais

Tabela 31. Entradas e Saídas do Processo: Relatório de Desempenho.

Gerenciar as Partes Interessadas

Processo responsável pelo gerenciamento das comunicações para satisfazer os requisitos das partes interessadas no projeto e resolver problemas com elas. A **Tabela 32** apresenta as entradas e as saídas deste processo.

Entrada
Plano de Gerenciamento das Comunicações
Ativos de Processos Organizacionais
Saída
Problemas Resolvidos
Solicitações de Mudanças Aprovadas
Ações Corretivas Aprovadas
Ativos de Processos Organizacionais
Plano de Gerenciamento do Projeto

Tabela 32. Entradas e Saídas do Processo: Gerenciar as Partes Interessadas.

Gerenciamento dos Riscos do Projeto

Esta área de conhecimento agrupa os processos necessários para identificação, análise e resposta aos riscos do projeto.

Planejamento do Gerenciamento de Riscos

Processo responsável pela decisão de como abordar, planejar e executar as atividades de gerenciamento de riscos de um projeto. A **Tabela 33** apresenta as entradas e as saídas deste processo.

Entrada
Fatores Ambientais da Empresa
Ativos de Processos Organizacionais
Declaração do Escopo do Projeto
Plano de Gerenciamento do Projeto
Saída
Plano de Gerenciamento de Riscos

Tabela 33. Entradas e Saídas do Processo: Planejamento do Gerenciamento de Riscos.

Identificação de Riscos

Processo responsável pela determinação dos riscos que podem afetar o projeto e documentação de suas características. A **Tabela 34** apresenta as entradas e as saídas deste processo.

Entrada
Fatores Ambientais da Empresa
Ativos de Processos Organizacionais
Declaração do Escopo do Projeto
Plano de Gerenciamento de Riscos
Plano de Gerenciamento do Projeto
Saída
Registro de Riscos

Tabela 34. Entradas e Saídas do Processo: Identificação de Riscos.

Análise Qualitativa de Riscos

Processo responsável pela priorização dos riscos para análise ou ação adicional subseqüente através de avaliação e combinação de sua probabilidade de ocorrência e impacto. A **Tabela 35** apresenta as entradas e as saídas deste processo.

Entrada
Ativos de Processos Organizacionais
Declaração do Escopo do Projeto
Plano de Gerenciamento de Riscos
Registro de Riscos
Saída
Registros de Riscos

Tabela 35. Entradas e Saídas do Processo: Análise Qualitativa de Riscos.

Análise Quantitativa de Riscos

Processo responsável pela análise numérica do efeito dos riscos identificados nos objetivos gerais do projeto. A **Tabela 36** apresenta as entradas e as saídas deste processo.

Entrada
Ativos de Processos Organizacionais
Declaração do Escopo do Projeto
Plano de Gerenciamento de Riscos
Registro de Riscos
Plano de Gerenciamento do Projeto
Saída
Registros de Riscos

Tabela 36. Entradas e Saídas do Processo: Análise Quantitativa de Riscos.

Planejamento de Respostas a Riscos

Processo responsável pelo desenvolvimento de opções e ações para aumentar as oportunidades e reduzir as ameaças aos objetivos do projeto. A **Tabela 37** apresenta as entradas e as saídas deste processo.

Entrada
Plano de Gerenciamento de Riscos
Registro de Riscos
Saída
Registros de Riscos
Plano de Gerenciamento do Projeto
Acordos Contratuais Relacionados a Riscos

Tabela 37. Entradas e Saídas do Processo: Planejamento de Respostas a Riscos.

Monitoramento e Controle de Riscos

Processo responsável pelo acompanhamento dos riscos identificados, pelo monitoramento dos riscos residuais, pela identificação dos novos riscos, pela execução de planos de respostas a riscos e pela avaliação da sua eficácia durante todo o ciclo de vida do projeto. A **Tabela 38** apresenta as entradas e as saídas deste processo.

Gerenciamento das Aquisições do Projeto.

Esta área de conhecimento agrupa os processos necessários para a aquisição de mercadorias e serviços fora da organização que desenvolve o projeto.

Planejar Compras e Aquisições

Processo responsável pela determinação do que, como e quando comprar ou adquirir. A **Tabela 39** apresenta as entradas e as saídas deste processo.

Planejar Contratações

Processo responsável pela documentação dos requisitos de

Entradas e Saídas do PMBoK

Cada uma das entradas e saídas dos processos do PMBoK está descrita a seguir em ordem alfabética:

• **Ações Corretivas Aprovadas:** São orientações autorizadas e documentadas necessárias para que o desempenho esperado do projeto fique de acordo com o plano de gerenciamento do projeto;
• **Ações Corretivas Implementadas:** São as ações corretivas aprovadas que foram implementadas para que o desempenho esperado do projeto fique de acordo com o plano de gerenciamento do projeto;
• **Ações Corretivas Recomendadas:** São recomendações documentadas necessárias para que o desempenho esperado do projeto fique de acordo com o plano de gerenciamento do projeto;
• **Ações Preventivas Aprovadas:** São orientações autorizadas e documentadas que reduzem a probabilidade de conseqüências negativas associadas a riscos do projeto;
• **Ações Preventivas Implementadas:** São as ações preventivas aprovadas que foram implementadas pela equipe de gerenciamento de projetos para reduzir as conseqüências dos riscos do projeto;
• **Ações Preventivas Recomendadas:** São recomendações documentadas que reduzem a probabilidade de conseqüências negativas associadas a riscos do projeto;
• **Acordos Contratuais Relacionados a Riscos:** Acordos contratuais (como contratos de seguros, serviços e outros itens conforme adequado) podem ser preparados para especificar a responsabilidade de cada uma das partes por riscos específicos;
• **Ativos de Processos Organizacionais:** Políticas, procedimentos, planos e diretrizes organizacionais. Aprendizado e o conhecimento das organizações obtido em projetos anteriores;
• **Atributos da Atividade:** Informações das atividades, como descrição, atividades antecessoras, atividades sucessoras, responsável, etc;
• **Avaliação do Desempenho da Equipe:** Avaliações da eficácia da equipe do projeto;
• **Calendário de Recursos:** Documentação dos dias trabalhados e dos dias não trabalhados que determinam as datas nas quais um recurso específico pode estar ativo ou está ocioso;
• **Calendário de Projeto:** Calendário de turnos ou dias trabalhados que define as datas nas quais as atividades do cronograma são trabalhadas, e as datas que são ociosas;
• **Contrato:** Um contrato é um acordo legal que gera obrigações para as partes que obriga o fornecedor a fornecer os produtos, serviços ou resultados especificados e obriga o comprador a pagar ao fornecedor. Nesse contexto, o projeto pode ser o fornecedor ou o comprador;
• **Contratos Encerrados:** O comprador, em geral através do seu administrador de contratos autorizado, apresenta ao fornecedor um aviso formal por escrito de que o contrato terminou;
• **Critérios de Avaliação:** Critérios desenvolvidos e usados para classificar ou pontuar propostas, dando embasamento para a escolha da melhor proposta;
• **Cronograma do Projeto:** Apresenta as datas estimadas e início e término de cada atividade do projeto;
• **Dados do Modelo do Cronograma:** Apresentam os marcos do cronograma, as atividades do cronograma, os atributos da atividade e a documentação de todas as premissas e restrições identificadas;
• **Decisões de Fazer ou Comprar:** As decisões documentadas de quais produtos, serviços ou resultados do projeto serão adquiridos ou desenvolvidos pela equipe do projeto;
• **Declaração do Escopo do Projeto:** Este documento descreve detalhadamente as entregas do projeto e o trabalho necessário para criar essas entregas, além de fornecer um entendimento comum do projeto entre todos os envolvidos. Este documento apresenta as informações a seguir: objetivos do projeto; descrição do escopo do produto; requisitos do projeto; limites do projeto; entregas do projeto; critérios de aceitação de produtos; restrições do projeto; premissas do projeto; organização inicial do projeto; riscos iniciais definidos; marcos do cronograma; limitação de fundos; estimativa de custos; requisitos do gerenciamento de configuração do projeto; especificações do projeto; requisitos de aprovação;
• **Declaração do Escopo Preliminar do Projeto:** A declaração do escopo inclui: Os objetivos do produto e do projeto; características e requisitos do produto ou serviço; critérios de aceitação do produto; limites do projeto; entregas e requisitos do projeto; restrições do projeto; premissas do projeto; organização inicial do projeto; riscos iniciais definidos; marcos do cronograma; EAP inicial; estimativa aproximada de custos; requisitos de gerenciamento de configuração do projeto; requisitos de aprovação;
• **Declaração do Trabalho do Contrato:** Define, para os itens que estão sendo comprados ou adquiridos, apenas a parte do escopo do projeto incluída no contrato relacionado;
• **Declaração do Trabalho do Projeto:** Descrição dos produtos ou serviços que serão fornecidos pelo projeto. Ela lista a necessidade de negócio, a descrição do escopo do produto e o plano estratégico;
• **Designações de Pessoal para o Projeto:** O projeto terá o seu quadro de pessoal quando forem designadas para trabalhar nele as pessoas adequadas;
• **Detalhes que dão suporte as Estimativas de Custos da Atividade:** Documentação de apoio deve fornecer uma imagem clara, profissional e completa do que originou a estimativa de custos;
• **Diagramas de Rede do Cronograma do Projeto:** Representações esquemáticas das atividades do cronograma do projeto e dos relacionamentos entre elas, também chamados de dependências;
• **Dicionário da EAP:** Documenta para cada componente da EAP um código do identificador de conta, uma declaração do trabalho, a organização responsável e uma lista de marcos do cronograma;
• **Disponibilidade de Recursos:** As informações sobre que recursos (como pessoas, equipamentos e material) estão disponíveis;
• **Documentos de Aquisição:** Documentos para buscar propostas de possíveis fornecedores;
• **Documentação do Contrato:** Documento usado para realizar o processo de encerramento do contrato e inclui o próprio contrato, além de mudanças feitas no mesmo;
• **Entregas:** Uma entrega é qualquer produto, resultado ou capacidade para realizar um serviço verificáveis, e que devem ser produzidos e fornecidos para terminar o projeto;
• **Entregas Aceitas:** Documentação que registra as entregas terminadas que

foram aceitas, e das entregas terminadas que não foram aceitas, juntamente com a razão;

• **Entregas Validadas:** Entregas inspecionadas e consideradas corretas pelo processo de controle da qualidade;

• **Estimativas de Custos da Atividade:** Estimativa de custos da atividade é uma avaliação quantitativa dos custos prováveis dos recursos necessários para terminar as atividades do cronograma;

• **Estimativas de Duração da Atividade:** Avaliações quantitativas do número provável de períodos de trabalho que serão necessários para terminar uma atividade do cronograma;

• **Estrutura Analítica do Projeto:** Decomposição do projeto em componentes menores, chegando até o nível de pacotes de trabalho;

• **Estrutura Analítica dos Recursos:** Estrutura hierárquica dos recursos identificados por categoria de recursos e tipo de recursos;

• **Fatores Ambientais da Empresa:** Indica quaisquer sistemas e fatores ambientais da empresa que cercam e influenciam o sucesso do projeto. Como leis, normas, sistemas de informação, infra-estrutura, recursos, etc;

• **Fornecedores Selecionados:** Fornecedores selecionados são aqueles considerados como estando em uma faixa competitiva com base no resultado da proposta ou da avaliação da licitação e que negociaram uma versão preliminar do contrato, que será o contrato real quando for feita uma concessão;

• **Funções e Responsabilidades:** Lista contendo as funções, autoridades, responsabilidades e competências necessárias para terminar o projeto;

• Informações sobre o Desempenho do Trabalho: As informações sobre o andamento das atividades do projeto que estão sendo executadas são coletadas rotineiramente como parte da execução do plano de gerenciamento do projeto;

• **Linha de Base da Qualidade:** Registra os objetivos de qualidade do projeto. A linha de base da qualidade é a base para medição e emissão de relatórios de desempenho da qualidade como parte da linha de base da medição de desempenho;

• **Linha de Base do Cronograma:** É uma versão especifica do cronograma do Projeto, que foi aceita e aprovada pela equipe de gerenciamento de projetos.

• **Linha de Base do Escopo:** Composta da Declaração do Escopo Detalhada do projeto, a EAP e o dicionário da EAP após estes documentos terem sido aprovados;

• **Linha de Base dos Custos:** Orçamento dividido em fases usado como base em relação à qual será medido, monitorado e controlado o desempenho de custos geral no projeto;

• **Lista de Atividades:** Lista que apresenta todas as atividades do cronograma planejadas para serem realizadas no projeto;

• **Lista de Fornecedores Qualificados:** Listagem dos fornecedores que são solicitados a apresentar uma proposta ou cotação;

• **Lista de Marcos:** Indica todos os marcos do projeto, e se o marco é opcional ou obrigatório;

• **Listas de Verificação da Qualidade:** Uma lista de verificação é uma ferramenta estruturada que é usada para verificar se foi executado um conjunto de etapas necessárias;

• **Medições de Controle da Qualidade:** Representam os resultados das atividades de Controle da Qualidade fornecidos como feedback para a Garantia da Qualidade para reavaliar e analisar os processos e padrões de qualidade da organização;

• **Medições de Desempenho:** Medem diversas informações do projeto, como custos, prazos, etc. Os valores calculados de variação de custo, valor planejado, índice de desempenho de custos, os valores calculados da variação de prazos (VP) e do índice de desempenho de prazos (IDP) para os componentes da EAP, especialmente para os pacotes de trabalho e contas de controle, são documentados e comunicados às partes interessadas;

• **Métricas da Qualidade:** Métricas são definições operacionais que descrevem o que é alguma coisa e como ela é medida pelo processo de controle da qualidade;

• **Mudanças Solicitadas:** São mudanças solicitadas para alterar o escopo, políticas ou planos do projeto;

• **Necessidade de Financiamento do Projeto:** A necessidade de financiamento, total e periódica (por exemplo, anual ou trimestral), é derivada da linha de base dos custos e pode-se definir que ela tenha um excesso;

• **Organogramas do Projeto:** Representação gráfica dos membros da equipe do projeto e suas relações hierárquicas;

• **Pacote de Documentos de Aquisição:** Solicitação formal preparada pelo comprador enviada para cada fornecedor e é a base sobre a qual um fornecedor prepara uma proposta para os produtos, serviços ou resultados solicitados que estão definidos e descritos na documentação de aquisição;

• **Plano de Gerenciamento das Aquisições:** Descreve como os processos de aquisição serão gerenciados desde o desenvolvimento da documentação de aquisição até o encerramento do contrato;

• **Plano de Gerenciamento das Comunicações:** Descreve os requisitos de comunicação dos envolvidos; as informações que serão comunicadas (formato, conteúdo e nível de detalhes); o responsável pela envio e os receptores das informações; os mecanismos e a periodicidade da transmissão da informação; glossário de termos comuns; método para atualizar o plano de gerenciamento das comunicações;

• **Plano de Gerenciamento de Contratos:** No caso de compras ou aquisições significativas, é preparado um plano para administrar o contrato com base nos itens especificados do comprador específico dentro do contrato, como documentação e requisitos de entrega e desempenho que o comprador e o fornecedor devem cumprir;

• **Plano de Gerenciamento de Qualidade:** Descrição de como a equipe de gerenciamento de projetos implementará a política de qualidade da organização executora;

• **Plano de Gerenciamento de Custos:** Documenta os processos de gerenciamento de custos e suas ferramentas e técnicas;

• **Plano de Gerenciamento de Pessoal:** Descreve quando e como serão atendidos os requisitos de recursos humanos. Ele aborda tópicos como o recrutamento e a seleção, a tabela de horários, critérios de liberação, necessidade de treinamento, reconhecimento e premiações, conformidade, e segurança;

Continuação - Entradas e Saídas do PMBoK

• **Plano de Gerenciamento de Riscos:** Descreve como o gerenciamento de riscos é estruturado e executado no projeto. Ele inclui os seguintes tópicos: metodologia; funções e responsabilidades; orçamentação; tempos; categorias de riscos; definições de probabilidades e impactos de riscos; matriz de probabilidade e impacto de riscos; revisão da tolerância das partes interessadas; formatos de relatórios; acompanhamento;

• **Plano de Gerenciamento do Cronograma:** Estabelece como o cronograma do projeto será gerenciado e controlado;

• **Plano de Gerenciamento do Escopo do Projeto:** Fornece uma orientação sobre como o escopo do projeto será definido, documentado, verificado, gerenciado e controlado. Ele inclui um processo para a preparação do escopo detalhado do projeto, um processo para a criação da EAP, um processo para a definição dos mecanismos de verificação e aceitação das entregas, um processo para o processamento das solicitações de mudança;

• **Plano de Gerenciamento do Projeto:** Define como o projeto é executado, monitoramento, controlado e encerrado. Ele é composto dos seguintes itens: Os processos de gerenciamento de projetos que serão utilizados neste projeto; as ferramentas e técnicas que serão utilizadas; como o trabalho será executado; como as mudanças serão monitoradas e controladas; como será o gerenciamento de configuração; como a integridade das linhas de base da medição de desempenho será mantida e utilizada; a necessidade e as técnicas de comunicação entre os envolvidos; o ciclo de vida do projeto selecionado e suas fases; as principais revisões de gerenciamento em relação a conteúdo, extensão e tempo para facilitar a abordagem de problemas pendentes;

• **Plano de Melhorias no Processo:** Detalha as etapas de análise dos processos que irão facilitar a identificação de desperdícios e de atividades sem nenhum valor agregado, para permitir a melhoria do processo;

• **Previsão de Término:** Um valor de estimativa no término é documentado e o valor é comunicado às partes interessadas;

• **Previsões:** As previsões incluem estimativas de eventos futuros do projeto com base nas informações e no conhecimento disponíveis no momento da previsão. As previsões são atualizadas e refeitas com base nas informações sobre o desempenho do trabalho;

• **Problemas Resolvidos:** Documentação dos problemas que foram abordados e encerrados;

• **Procedimento de Encerramento Administrativo:** Documentação de todas as atividades, e responsabilidades necessárias para a execução do procedimento de encerramento administrativo do projeto. Como procedimentos para transferir os serviços ou produtos do projeto para a produção e/ou para as operações;

• **Procedimento de Encerramento de Contratos:** Procedimento documentado dos termos e condições dos contratos e quaisquer critérios de saída ou de término necessários para o encerramento do contrato. Ele contém todas as atividades e responsabilidades relacionadas ao encerramento de contrato;

• **Processos de Gerenciamento de Projetos:** Todos os processos das outras áreas de conhecimentos apresentados neste documento;

• **Produto, Serviço ou Resultado Final:** A aceitação formal e a entrega do produto, serviço ou resultado final que o projeto foi autorizado a produzir. A aceitação inclui o recebimento de uma declaração formal de que os termos do contrato foram atendidos;

• **Propostas:** Documentos preparados pelo fornecedor que descrevem a capacidade e a disposição do fornecedor de fornecer os produtos, serviços ou resultados solicitados descritos na documentação de aquisição;

• **Recursos Necessários para a Atividade:** Descrição dos tipos e quantidades de recursos necessários para cada unidade do cronograma em um pacote de trabalho;

• **Relatórios de Desempenho:** Compilam informações sobre o desempenho do trabalho do projeto, como entregas provisórias não terminadas;

• **Registro de Riscos:** Documenta a lista de riscos encontrados, lista das possíveis respostas aos riscos, causas raiz dos riscos, categorias de riscos atualizadas; lista de prioridades dos riscos do projeto; riscos agrupados por categoria; lista de riscos que exigem resposta em curto prazo; lista de riscos por análise e respostas adicionais; lista de observações de riscos de baixa prioridade; tendências dos resultados da análise qualitativa de riscos; análise probabilística de riscos; probabilidade de realização dos objetivos de custo e tempo; lista priorizada de riscos quantificados; tendências dos resultados da análise quantitativa de riscos;

• **Reparo de Defeito Aprovado:** É a solicitação autorizada e documentada para corrigir um defeito do produto encontrado durante a inspeção de qualidade ou o processo de auditoria;

• **Reparo de Defeito Implementado:** Implementação de correções aprovadas devido a defeito do produto;

• **Reparo de Defeito Validado:** Notificação que o item reparado foi aceito ou rejeitado na inspeção;

• **Reparos de Defeitos Recomendado:** Recomendações de reparos de defeitos encontrados durante a inspeção de qualidade e o processo de auditoria;

• **Solicitações de Mudanças Aprovadas:** São mudanças autorizadas e documentadas que alteram o escopo, políticas ou planos do projeto. As solicitações de mudança aprovadas são agendadas para serem implementadas;

• **Solicitações de Mudanças Implementadas:** São as solicitações de mudança aprovadas que foram implementadas;

• **Solicitações de Mudanças Rejeitadas:** São mudanças solicitadas que foram rejeitadas e sua documentação de apoio;

• **Termo de Abertura do Projeto:** Este documento apresenta a necessidade de negócios, a descrição de alto nível do projeto ou requisitos do produto. Objetivo ou justificativa do projeto. Gerente de projetos designado e nível de autoridade atribuída. Cronograma de marcos sumarizado. Influência das partes interessadas. Organizações funcionais e sua participação. Premissas e restrições organizacionais, ambientais e externas. Caso de negócios justificando o projeto, incluindo o retorno sobre o investimento. Orçamento sumarizado.

Entrada
Plano de Gerenciamento de Riscos
Registro de Riscos
Solicitações de Mudanças Aprovadas
Informações sobre o Desempenho do Trabalho
Relatórios de Desempenho
Saída
Registros de Riscos
Mudanças Solicitadas
Ações Corretivas Recomendadas
Ações Preventivas Recomendadas
Ativos de Processos Organizacionais
Plano de Gerenciamento do Projeto

Tabela 38. Entradas e Saídas do Processo: Monitoramento e Controle de Riscos.

Entrada
Fatores Ambientais da Empresa
Ativos de Processos Organizacionais
Declaração do Escopo do Projeto
Estrutura Analítica do Projeto
Dicionário da EAP
Plano de Gerenciamento do Projeto
Saída
Plano de Gerenciamento das Aquisições
Declaração do Trabalho do Contrato.
Decisões de Fazer ou Comprar
Mudanças Solicitadas

Tabela 39. Entradas e Saídas do Processo: Planejar Compras e Aquisições.

produtos, serviços e resultados e identificação de possíveis fornecedores. A **Tabela 40** apresenta as entradas e as saídas deste processo.

Entrada
Plano de Gerenciamento das Aquisições
Declaração do Trabalho do Contrato
Decisões de Fazer ou Comprar
Plano de Gerenciamento do Projeto
Saída
Documentos de Aquisição
Critérios de Avaliação
Declaração do Trabalho do Contrato

Tabela 40. Entradas e Saídas do Processo: Planejar Contratações.

Solicitar Respostas de Fornecedores

Processo responsável pela obtenção de informações, cotações, preços, ofertas ou propostas, conforme adequado. A **Tabela 41** apresenta as entradas e as saídas deste processo.

Entrada
Ativos de Processos Organizacionais
Plano de Gerenciamento das Aquisições
Documentos de Aquisição
Saída
Lista de Fornecedores Qualificados
Pacote de Documentos de Aquisição
Propostas

Tabela 41. Entradas e Saídas do Processo: Solicitar Respostas de Fornecedores.

Selecionar Fornecedores

Processo responsável pela análise de ofertas e escolha entre possíveis fornecedores e pela negociação de um contrato por escrito com cada fornecedor. A **Tabela 42** apresenta as entradas e as saídas deste processo.

Entrada
Ativos de Processos Organizacionais
Plano de Gerenciamento das Aquisições
Critérios de Avaliação
Pacote de Documentos de Aquisição
Propostas
Lista de Fornecedores Qualificados
Documentos de Aquisição
Saída
Fornecedores Selecionados
Contrato
Plano de Gerenciamento de Contratos
Disponibilidade de Recursos
Plano de Gerenciamento das Aquisições
Mudanças Solicitadas

Tabela 42. Entradas e Saídas do Processo: Selecionar Fornecedores.

Administração de Contrato

Processo responsável pelo gerenciamento do contrato e da relação entre o comprador e o fornecedor, assim como pela análise e documentação do desempenho atual ou passado de um fornecedor com o fim de estabelecer ações corretivas necessárias e fornecer uma base para futuras relações com o fornecedor.

Este processo também é responsável pelo gerenciamento de mudanças relacionadas ao contrato e, quando adequado, pelo gerenciamento da relação contratual com o comprador externo do projeto. A **Tabela 43** apresenta as entradas e as saídas deste processo.

Encerramento do Contrato

Processo responsável pelo término e liquidação de cada contrato, inclusive a resolução de quaisquer itens em aberto, e pelo encerramento de cada contrato aplicável ao projeto ou a uma fase do projeto. A **Tabela 44** apresenta as entradas e as saídas deste processo.

Entrada
Contrato
Plano de Gerenciamento de Contratos
Fornecedores Selecionados
Relatórios de Desempenho
Solicitações de Mudanças Aprovadas
Informações sobre o Desempenho do Trabalho
Saída
Documentação do Contrato
Mudanças Solicitadas
Ações Corretivas Recomendadas
Ativos de Processos Organizacionais
Plano de Gerenciamento do Projeto

Tabela 43. Entradas e Saídas do Processo: Administração de Contrato.

Entrada
Plano de Gerenciamento das Aquisições
Plano de Gerenciamento de Contratos
Documentação do Contrato
Procedimentos de Encerramento de Contratos
Saída
Contratos Encerrados
Ativos de Processos Organizacionais

Tabela 44. Entradas e Saídas do Processo: Encerramento do Contrato.

Conclusão

O PMBoK é um conjunto de boas práticas de gerenciamento de projeto amplamente aceito e utilizado. Uma boa prática é uma recomendação ou orientação a ser seguida, ou seja, não significa que sempre deva ser seguida.

Por isso, 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.

Além disso, o PMBoK fornece um vocabulário comum para o gerenciamento de projetos, padronizando termos e conceitos.

Finalmente, este artigo foi baseado na terceira edição do PMBoK (versão de 2004). Recentemente foi publicada a quarta edição (versão de 2008). Na quarta edição, houve uma redução do número total de 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 processos em um único processo.

Mesmo com a redução total de processos, ainda houve a adição de dois novos processos:

- Identificar Stakeholers
- Coletar Requisitos

Em futuras edições, devemos apresentar em maiores detalhes as principais mudanças entre o PMBoK 2004 e 2008.

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

Ferramentas de Qualidade de Código

Automação do processo de aderência a padrões de codificação com Checkstyle

De que se trata o artigo?

Este artigo aborda o emprego de ferramentas de qualidade de código, objetivando apresentar aos desenvolvedores como essas ferramentas podem ser úteis na automação da aderência a padrões de codificação estabelecidos.

Para que serve?

Para que serve: Através de exemplos de aplicação desenvolvidos em uma ferramenta típica de qualidade de código, o Checkstyle, são apresentados os benefícios concretos e ganhos de produtividade no desenvolvimento dessas aplicações.

Em que situação o tema é útil?

O emprego de ferramentas de qualidade de código é útil em ambientes de desenvolvimento de software no sentido de automatizar a aderência a padrões de codificação, liberando os programadores de um conjunto de tarefas complexas e enfadonhas de serem realizadas sem suporte computacional específico.

Escrever código de boa qualidade é um requisito importante para os programadores. Mas o que isso de fato significa? Na prática, uma definição clara do que seja código fonte de qualidade pode variar bastante e depender de diversos fatores relacionados ao desenvolvimento de software, entre os quais o leitor pode considerar: ser seguro, rápido, enxuto, de fácil manutenção e possível de estender e de ser entendido por outros desenvolvedores. Além disso, a linguagem de programação utilizada no projeto é fator importante nessa discussão.

O objetivo aqui é apresentar uma ferramenta de qualidade de código que pode ser utilizada em projetos de desenvolvimento em Java. O enfoque do artigo concentra-se no **Checkstyle**, embora o programador possa utilizar outros softwares similares, tais como **PMD**, **FindBugs**, **Dependency Finder** e **SQE**. Dada a importância deste assunto, também será abordado um exemplo de integração da ferramenta com o ambiente de desenvolvimento integrado **Eclipse**, embora os usuários do **NetBeans** e outros ambientes de desenvolvimento integrado (IDE, do inglês *Integrated Development Environment*) também possam contar com esse apoio.

Wander Antunes Gaspar Valente
wander@cesjf.br
É mestrando em Modelagem Computacional pela UFJF, Especialista em Ciência da Computação pela UFV, Engenheiro Eletricista pela UFJF, professor dos cursos de Bacharelado em Sistemas de Informação e Engenharia de Telecomunicações do Centro de Ensino Superior de Juiz de Fora.

Parâmetros para avaliar a qualidade de código

Entre os aspectos que podem ser considerados a fim de avaliar a qualidade de código incluem-se: formatação consistente e facilidade de entendimento (endentação e espacejamento); regras de nomeação consistentes; ausência de erros de compilação; capacidade adequada e consistente de tratamento de erros de execução; aderência a boas práticas de programação e de projeto; e documentação abrangente e de fácil entendimento em todo o código fonte.

Naturalmente que algumas dessas características podem ser consideradas mais fáceis de serem implementadas, mas não todas. Muitas empresas e organizações possuem padrões documentados que os desenvolvedores devem seguir. Na prática, tem sido muito difícil para essas organizações obterem sucesso nessa padronização. A questão é que, para se adequarem, os programadores precisam assimilar os padrões adotados e reverem frequentemente o código produzido para garantir a sua conformidade. Esse processo gasta tempo e é complexo de ser conduzido manualmente.

Uma alternativa mais racional é o emprego de ferramentas capazes de automatizar o processo de avaliação da qualidade de código. Certamente o processo torna-se ainda mais produtivo se tais ferramentas puderem ser utilizadas diretamente a partir de IDEs disponíveis.

Checkstyle

Checkstyle é uma ferramenta de qualidade de código projetada para auxiliar os programadores a detectar violações de estilo de codificação em Java. A versão 5, compatível com Java 5 ou superior, foi lançada em meados de 2008 e encontra-se disponível sob licença LGPL. O software pode ser usado como um aplicativo, como parte de um script Ant ou como um plugin para IDEs como Eclipse, NetBeans e BlueJ.

A ferramenta contém diversos controles – "checks" –, um para cada padrão de codificação ou estilo que é capaz de identificar. O desenvolvedor pode configurar o software habilitando os controles necessários conforme o padrão de codificação adotado no escopo do projeto. A partir da avaliação, é gerado um relatório das violações detectadas. É importante ressaltar, porém, que o **Checkstyle** não é capaz de fazer a refatoração automática do código fonte. Assim, o programador é inteiramente responsável por fazer as correções detectadas durante processo de avaliação do código.

A abrangência dos recursos incorporados ao **Checkstyle** pode ser observada a partir do número de controles atualmente disponíveis para a verificação da qualidade do código, que ultrapassa a casa de uma centena. Com o sentido de organização, os controles são agrupados de acordo com suas respectivas funcionalidades. Para que o leitor possa ter uma noção do escopo da ferramenta, segue a relação dos agrupamentos com uma breve descrição dos principais controles oferecidos:

– **Javadoc Comments:** verifica se os pacotes utilizados estão associados a um correspondente arquivo de documentação e valida as tags Javadoc incluídas;

– **Naming Conventions:** valida as especificações de nomes de identificadores, para cada um dos tipos (classes, métodos, constantes). Vale mencionar que a verificação baseia-se na avaliação de uma expressão regular, como **^[a-z][a-zA-Z0-9]*$**, que, nesse caso, especifica que os identificadores comecem com uma letra minúscula – **[a-z]** – e o restante dos caracteres sejam letras e números – **[a-zA-Z0-9]***, onde o metacaractere * indica a possibilidade de repetição. Os metacaracteres **^** e **$** marcam, respectivamente, o início e o fim da expressão regular;

Header: verifica a existência de um cabeçalho associado ao código fonte e valida seu conteúdo de acordo com um arquivo texto externo contendo as informações requeridas, como dados do projeto e direitos autorais;

Imports: detecta declarações imports com o marcador *, redundâncias, imports não utilizados, ordena imports por grupo – primeiro java, em seguida javax, depois todo o restante – e em ordem alfabética;

Size Violations: valida número de linhas do código fonte, de classes e de métodos, tamanho de linhas e número de parâmetros a partir de valores arbitrados para o projeto. Assim, pode-se limitar o número de linhas do código fonte a 1000 ou o número de caracteres por linha a 80, por exemplo;

Whitespaces: valida o emprego de espaços em branco entre classes, métodos, identificadores, palavras reservadas e após parênteses e vírgulas, dentre outros. Como exemplo, a especificação da padronização de código fonte do projeto pode requerer a inserção de um ou mais espaços em branco após uma vírgula e a não ocorrência de espaço antes de cada vírgula;

Modifiers: verifica a ordem dos modificadores, conforme a *Java Language Especification* (ver seção Links) e detecta redundâncias como, por exemplo, declaração de *public* e *abstract* para métodos declarados em uma *interface*, ou *final* para métodos declarados em classes *final*;

Block Checks: detecta blocos vazios, ausência de chaves de abertura e fechamento de blocos, e blocos aninhados desnecessários.

Coding: detecta diversas violações de codificação, incluindo comandos condicionais *inline*, declarações vazias (;), redundâncias em cláusulas *throw*, nomes idênticos para variáveis locais e de classe; ausência de claúsula *this* para variáveis de instância do objeto corrente, declarações de variáveis em linhas separadas, ausência de cláusula default em comandos *switch*;

Class Design: verifica visibilidade de membros de classes – por exemplo, membros *static final* devem ser *public*, os demais devem ser *private*, *protected* ou *package*. Verifica implementação de interfaces inadequadas – por exemplo, que contêm apenas constantes –e classes utilitárias, ou seja, classes que contêm apenas métodos estáticos, que não devem ser definidas com um construtor *public*;

Duplicate Code: detecta trechos e linhas de código considerados duplicados – o número mínimo de linhas para detecção pode ser configurado. Na avaliação de código duplicado, apenas discrepâncias quanto à endentação são desconsideradas;

Metrics: verifica o número de operadores lógicos em expressões relacionais – o número de operadores &&, ||, &, | e ^ pode ser configurado para a avaliação do Checkstyle –, avalia

a complexidade condicional (ciclomática) do código, avalia as complexidades NPath e *Non Commenting Source Statements* (NCSS). A complexidade condicional é medida pelo número de *if*, *while*, *do*, *for*, *?* (comando condicional ternário), *catch*, *switch* e de operadores relacionais em um método Java. Valores acima de 10 são considerados ruins e, nesse caso, o método é passível de refatoração. A complexidade NPath mede o número de caminhos de execução possíveis em um método Java. Valores acima de 200 são considerados ruins. A complexidade NCSS mede o número de linhas de código, exceto linhas de comentários, contidas em métodos e classes Java. Valores recomendados – e que podem ser configurados – são 50 e 1500, respectivamente, para métodos e classes.

Miscellaneous: verifica endentação, padrão de arrays (colchetes após o tipo de dado ou após o identificador do atributo), detecta métodos *main()* – comumente usados para *debug* – e não excluídos.

Plugin do Checkstyle para o IDE Eclipse

Conforme já foi mencionado, é possível executar o **Checkstyle** a partir de uma linha de comando do sistema operacional ou através de um build **Ant**. Se o leitor tiver interesse em utilizar o **Checkstyle** nesses ambientes, poderá consultar a documentação sobre o assunto na página oficial da ferramenta.

Contudo, o processo de verificação de qualidade de código torna-se mais produtivo quando incorporado a um ambiente de desenvolvimento. Existem diversos plugins para integrar os recursos do **Checkstyle** a IDEs amplamente utilizados pela comunidade de Engenharia de Software.

Um dos trabalhos mais sólidos nesse sentido denomina-se Eclipse-CS, que se encontra na versão 4.4.2. O software recebeu o prêmio "Eclipse Community Award 2007" na categoria "Best Open Source Eclipse-based Developer Tool". O download e a instalação do plugin pode ser feito diretamente a partir do IDE Eclipse. Se você estiver utilizando a versão 3.4 (Eclipse Ganymede), a partir de Help - Software Updates, clique na aba - Available Software e, depois, clique no botão Add site. Informe a URL do recurso: *http://eclipse-cs.sourceforge.net/update*. Após a conexão com o servidor, selecione Eclipse Checkstyle plugin e clique no botão Install. Para a versão 3.3 (Eclipse Europa), há pequenas diferenças, mas ainda assim é possível seguir esse roteiro ou consultar o tutorial disponível a partir da página oficial do projeto Eclipse-CS (ver seção Links).

Em *Windows Preferences Checkstyke Global Check Configurations* são listadas as configurações de controle disponíveis **(Figura 1)**. Observe que a instalação padrão do **Checkstyle** incluiu dois arquivos de configuração, *Sun Checks* e *Sun Checks (Eclipse)*. O primeiro verifica a aderência às convenções de codificação Java sugeridas pela Sun Microsystems (*Sun Code Conventions*). O segundo também, mas com pequenas adaptações para o padrão de codificação do **Eclipse**. Estas configurações abrangem regras para nomeação e organização de arquivos, endentação, comentários, declarações, uso de espaços em branco e boas práticas de programação.

Exemplo de aplicação

Para que o leitor possa perceber mais claramente de que forma o **Checkstyle** pode ser útil no processo de avaliação de qualidade de código, é apresentado a seguir um exemplo prático, onde vários dos controles disponíveis são configurados e executados a partir da integração com o IDE Eclipse.

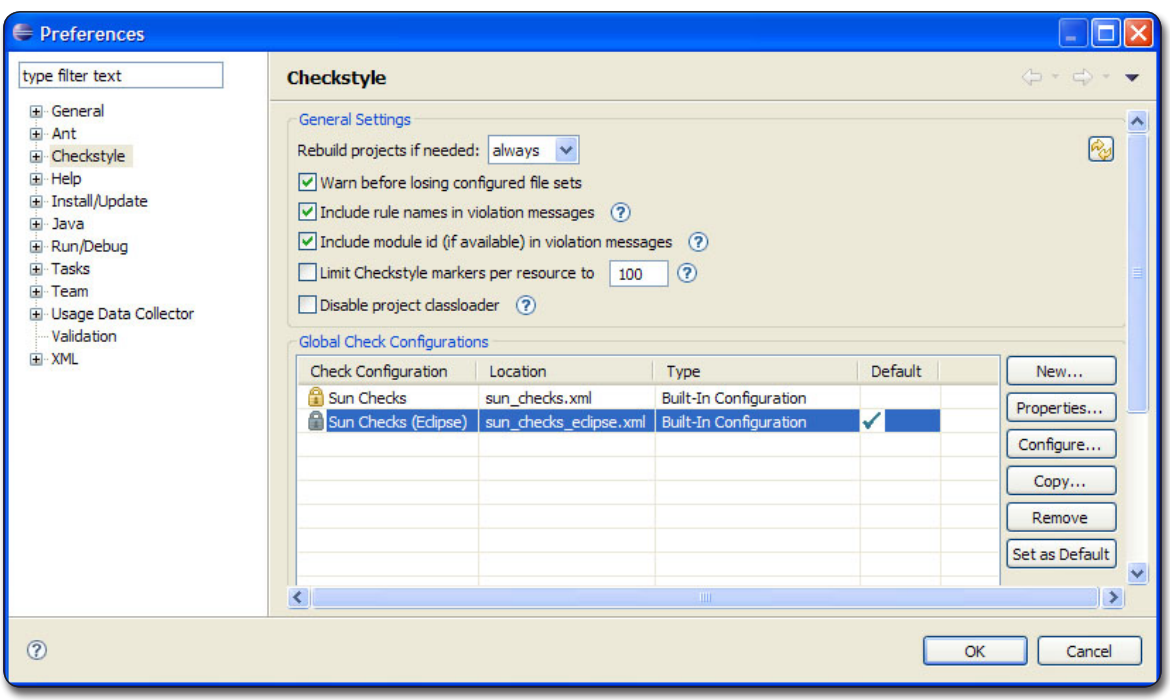


Figura 1. Opções padrão de configuração do Checkstyle

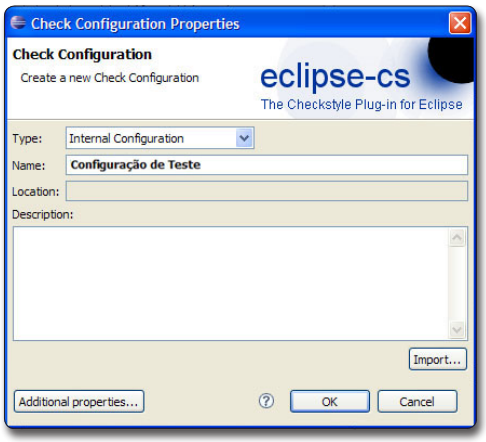


Figura 2. Janela ***Check Configuration Properties***

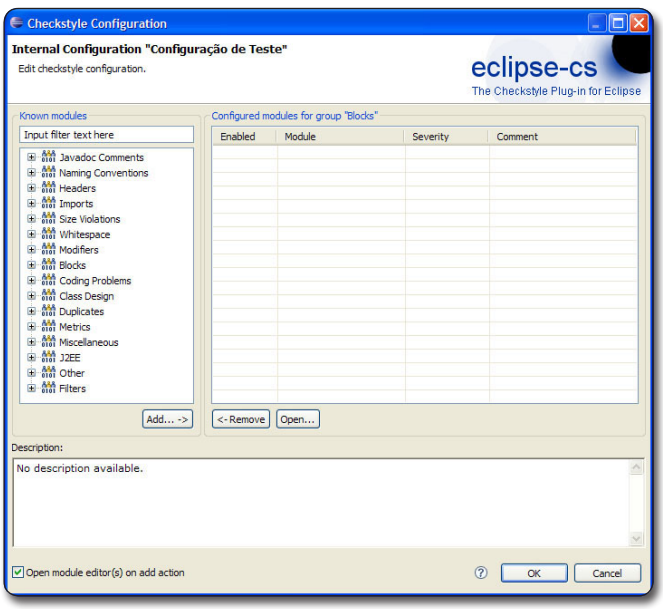


Figura 3. Janela ***Checkstyle Configuration***

Listagem 1. Classe Ponto2D.java

```
import javax.swing.*;

public class Ponto2D {

    public void inicializaPonto2D(double x,double y) {
        this.x = x;
        this.y = y;
    }

    private double x, y;

    public boolean eIgual(Ponto2D outroPonto2D) {
        if ((x == outroPonto2D.x) && (this.y == outroPonto2D.y))
            return true;
        else
            return false;
    }

    public Ponto2D origem() {
        Ponto2D temp = new Ponto2D(); ;
        temp.inicializaPonto2D(0, 0);
        return temp;
    }

    public Ponto2D clona() {
        Ponto2D temp = new Ponto2D();
        {
            temp.inicializaPonto2D(x, y);
        }
        return temp;
    }

    public String toString() {
        String Resultado = "(" + x + "," + y + ")";
        return Resultado;
    }

    public static void main(String[] args) {
        Ponto2D pt1;
        Ponto2D pt2;
        Ponto2D pt3;
        Ponto2D pt4;
        pt1 = new Ponto2D();
        pt2 = new Ponto2D();
        pt1.inicializaPonto2D(-1.34, 9.17);
        System.out.println("Coordenadas de p1: " + pt1);
        System.out.println("Coordenadas de p2: " + pt2);
        pt3 = pt1.clona();
        System.out.println("P3 está na mesma posição de P1?" +
          pt3.eIgual(pt1));
        pt4 = pt1.origem();
        System.out.println("P4 está na mesma posição de P1?" +
          pt4.eIgual(pt1));
    }
}
```

Um dos pontos fortes da integração do **Checkstyle** com o **Eclipse** reside na possibilidade de customização dos arquivos de configuração. Por exemplo, é possível criar uma nova configuração diretamente a partir de *Window Preferences Checkstyle*. Na janela *Check Configuration Properties*, clique no botão *new*, selecione *Internal Configuration* para a opção *type* e especifique um nome para o arquivo na opção *name* **(Figura 2)**. Em seguida, na janela *Checkstyle Configuration*, especifique os parâmetros e habilite cada um dos controles a serem inseridos na nova configuração **(Figura 3)**.

Vamos inserir 10 controles para o nosso arquivo de configuração, a partir da janela *Checkstyle Configuration*: *Avoid Star Imports* e *Unused Imports* (grupo *Imports*); *Local Variable Names* (grupo *Naming Conventions*); *Multiple Variable Declarations*, *Empty Statement* e *Declaration Order Check* (grupo *Coding Problems*); *Whitespace After* (grupo *Whitespaces*); *Maximum Line Length* (grupo *Size Violations*); *Need Braces e Avoid Nested Blocks* (grupo *Blocks*).

Para efeito de teste usaremos no Eclipse a classe Java *Ponto2D.java*, apresentada na **Listagem 1**. Esse código fonte, propositalmente, contém diversas violações de código que deverão ser detectadas ao ser avaliado pelo **Checkstyle**, por meio do nosso arquivo de configuração.

Crie um projeto Java no Eclipse e insira a classe *Ponto2D.java*. A partir do menu principal, selecione *Project Properties Checkstyle*. Habilite a opção *Checkstyle active for this project*. Na janela *Simple – use the following check configuration for all files*, configure para *Configuração de Teste*, o nosso arquivo de configuração, no qual habilitaremos os controles que se seguem.

Controle Avoid Star Imports

Na janela *Checkstyle Configuration*, selecione *Know Modules Imports Avoid Star (Demand) Imports* e clique no botão *Add*. Em *Edit module configuration*, clique em *OK* **(Figura 4)**.

Esse controle pertence ao grupo *Imports*. Seu objetivo consiste em verificar a ocorrência de declarações *import* que usam o *wildcard* *. Embora esse marcador na realidade não importe todas as classes do pacote, é considerada boa prática de programação escrever um *import* para cada classe utilizada, a fim de facilitar o entendimento do código fonte.

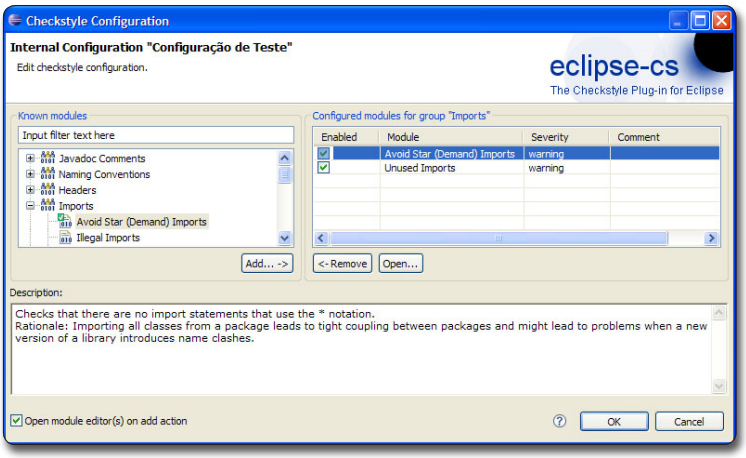


Figura 4. Configuração do módulo Avoid Star Imports

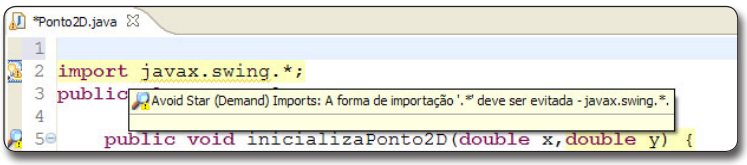


Figura 5. Detecção de violação de código Avoid Star Imports

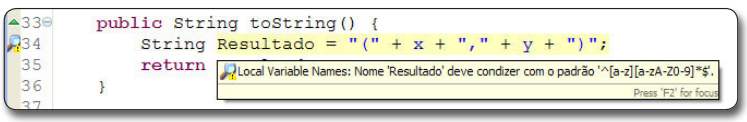


Figura 6. Detecção de violação de código Local Variable Names

A **Figura 5** apresenta a detecção da violação de código *Avoid Star Imports* na linha referente ao comando *import javax.swing.**.

Controle Local Variable Names

Esse controle pertence ao grupo *Naming Conventions* e seu objetivo é verificar a aderência dos nomes das variáveis locais ao padrão definido para o projeto de software. A verificação baseia-se na avaliação de expressões regulares, conforme já mencionado anteriormente. A expressão regular **^[a-z][a-zA-Z0-9]*$** determina que os identificadores devam começar com uma letra minúscula e que o restante dos caracteres sejam constituídos por letras e números. Essa convenção é sugerida pela própria *Java Language Especification*.

A **Figura 6** apresenta a detecção da violação de código *Local Variable Names* na linha 34 do código fonte da classe *Ponto2D.java*.

Controle Multiple Variable Declarations

Esse controle pertence ao grupo *Coding Problems* do **Checkstyle** e tem por objetivo verificar se as variáveis são declaradas uma por linha e em declarações separadas. A justificativa é que a declaração de mais de uma variável por linha dificulta a leitura e a documentação. Além disso, trata-se de uma convenção sugerida para programação Java a partir do documento *Sun Code Conventions* (ver seção Links).

A **Figura 7** apresenta a detecção da violação de código *Multiple Variable Declarations* na linha 11 do código fonte da classe *Ponto2D.java*.

Controle Empty Statement

Esse controle também está incluído no grupo *Coding Problems*. Seu objetivo é detectar ocorrências de declarações vazias (ponto-e-vírgulas "sozinhos"). A questão aqui já não se refere nem mesmo à qualidade de código, mas como o problema não é apontado pelo interpretador Java, podem ser encontradas declarações vazias em códigos fonte Java.

A **Figura 8** apresenta a detecção da violação de código *Empty Statement* na linha 11 do código fonte da classe *Ponto2D.java*.

Controle Declaration Order Check

Mais um controle incluído no grupo *Coding Problems*. De acordo com o *Sun Code Conventions*, a ordem sugerida dos componentes de uma classe ou interface Java é: classe, atributos, construtores e, finalmente, métodos.

A **Figura 9** apresenta a detecção da violação de código *Declaration Order Check* na linha 11 do código fonte da classe *Ponto2D.java*. Pode-se perceber que essa declaração de atributos aparece após um método.

Controle Whitespace After

Esse controle pertence ao grupo *Whitespaces*. Objetivando a clareza do código, segundo o *Sun Code Conventions*, é recomendada a inserção de um espaço em branco após alguns *tokens* da linguagem, como vírgula, ponto-e-vírgula e *typecast* (coerção explícita de tipo de dado).

A **Figura 10** apresenta a detecção da violação de código *Declaration Order* na linha 6 do código fonte da classe *Ponto2D.java*.

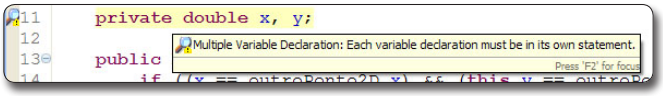


Figura 7. Detecção de violação de código Multiple Variable Declarations

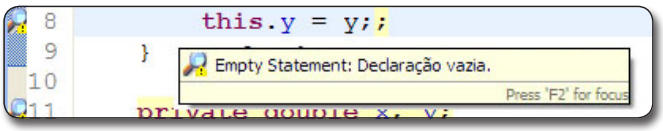


Figura 8. Detecção de violação de código Empty Statement

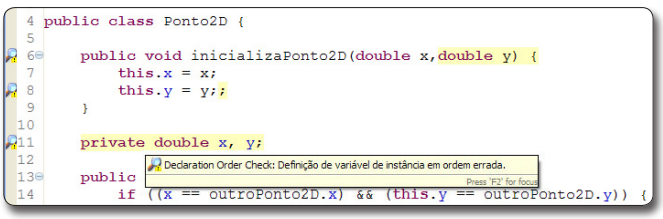


Figura 9. Detecção de violação de código Declaration Order Check

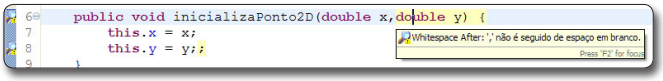


Figura 10. Detecção de violação de código Whitespace After

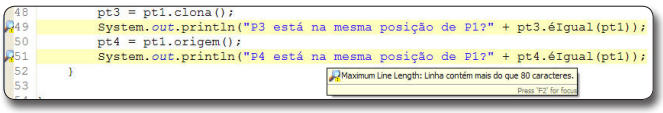


Figura 11. Detecção de violação de código Maximum Line Length

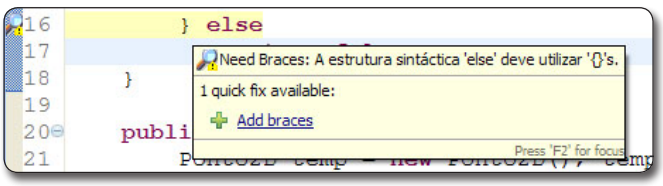


Figura 12. Detecção de violação de código Need Braces

Controle Maximum Line Length

Esse controle pertence ao grupo *Size Violations*. Seu objetivo é detectar linhas de código que ultrapassam o número de caracteres definido no controle. A justificativa para o seu emprego é que linhas longas dificultam a leitura, seja em fontes impressos ou mesmo em IDEs, onde usualmente a janela de código divide espaço com informações adicionais do projeto.

A **Figura 11** apresenta a detecção da violação de código *Maximum Line Length* nas linhas 49 e 51 do código fonte da classe *Ponto2D.java*. Observe que o controle foi configurado com o valor 80 para o número máximo de caracteres por linha.

Controle Need Braces

Esse controle pertence ao grupo *Block*. Seu objetivo é verificar se comandos associados a estruturas de seleção e repetição encontram-se delimitados por chaves de abertura e fechamento.

A **Figura 12** apresenta a detecção da violação de código *Need Braces* na linha 16 do código fonte da classe *Ponto2D.java*, referente a uma cláusula *else*. Observe que, nessa violação de código, o plugin do **Checkstyle** para o **Eclipse** disponibiliza a opção de *quick fix* (correção rápida).

Controle Avoid Nested Blocks

Esse controle pertence ao grupo *Block*. Seu objetivo é detectar a ocorrência de blocos de códigos usados desnecessariamente e que dificultam a compreensão pelo leitor. A **Figura 13** apresenta a detecção da violação de código *Avoid Nested Blocks* na linha 27 do código fonte da classe *Ponto2D.java*, referente ao bloco de código que delimita a instrução *temp.inicializaPonto2D(x, y);*.

Outros recursos do plugin

O plugin **Eclipse-CS** oferece ainda uma série de recursos adicionais que aprimoram a integração do **Checkstyle** ao **Eclipse**. Por exemplo, as mensagens de violação de código podem ser apresentadas na janela Problems **(Figura 14)**.

A qualquer momento é possível alterar um arquivo de configuração do **Checkstyle**, a partir da janela *Checkstyle Configuration*, o que permite adaptar o processo de avaliação de qualidade do código de forma dinâmica, de acordo com as necessidades do ambiente de desenvolvimento.

Além disso, um arquivo de configuração pode ser ajustado para atender às particularidades de um projeto específico de software, sem que os demais projetos em andamento sejam afetados. Para esse fim, selecione o projeto na aba *Package Explorer* e, a partir do menu principal, selecione *Project Properties Checkstyle*. Clique na aba *Local Check Configurations* e defina os controles necessários.

Também é possível importar para o **Eclipse-CS** um arquivo de configuração do **Checkstyle** em formato XML, criado manualmente em um editor de textos simples ou exportado a partir de outro IDE. Para isso, selecione *Window Preferences Checkstyle*. Na janela *Check Configuration Properties*, clique no botão *new*, selecione *Internal Configuration* para a opção *type* e especifique um nome para o arquivo na opção *name*. Em seguida, clique no botão *Import* e selecione o arquivo XML.

Por fim, o **Eclipse-CS** possui um recurso interessante, a janela *Checkstyle violations chart*, que pode ser habilitado a partir de *Window Show View Others Checkstyle*. Esse recurso apresenta, na forma de um gráfico de pizza, as violações de código detectadas, categorizadas pelos agrupamentos **(Figura 15)**, auxiliando na identificação dos tipos de violações mais comuns realizadas.

Conclusão

As equipes de desenvolvimento em Java dispõem de diversas ferramentas livres destinadas à melhoria da qualidade de código em seus projetos de software. Esses utilitários – entre os quais o *Checkstyle* constitui um representante típico – são capazes de verificar o código fonte produzido e detectar as violações encontradas, incluindo-se códigos duplicados, bugs, aderência a boas práticas de programação e projeto, entre outros. Esses recursos trazem benefícios concretos ao processo de desenvolvimento uma vez que fazer manualmente a avaliação

```
25    public Ponto2D clona() {
26        Ponto2D temp = new Ponto2D();
27        {  Avoid Nested Blocks: Avoid nested blocks.
28            temp.inicializaPonto2D(x, y);
29        }
30        return temp;
31    }
```

Figura 13. Detecção de violação de código Avoid Nested Braces

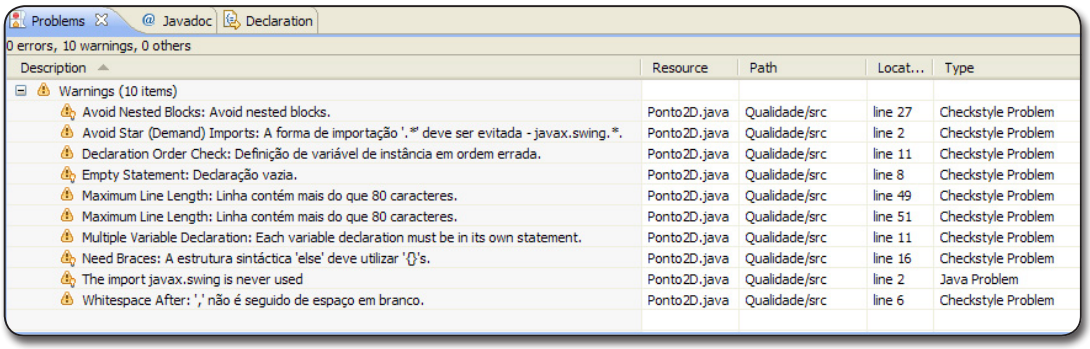


Figura 14. Janela ***Problems*** com mensagens de avaliação do ***Checkstyle***

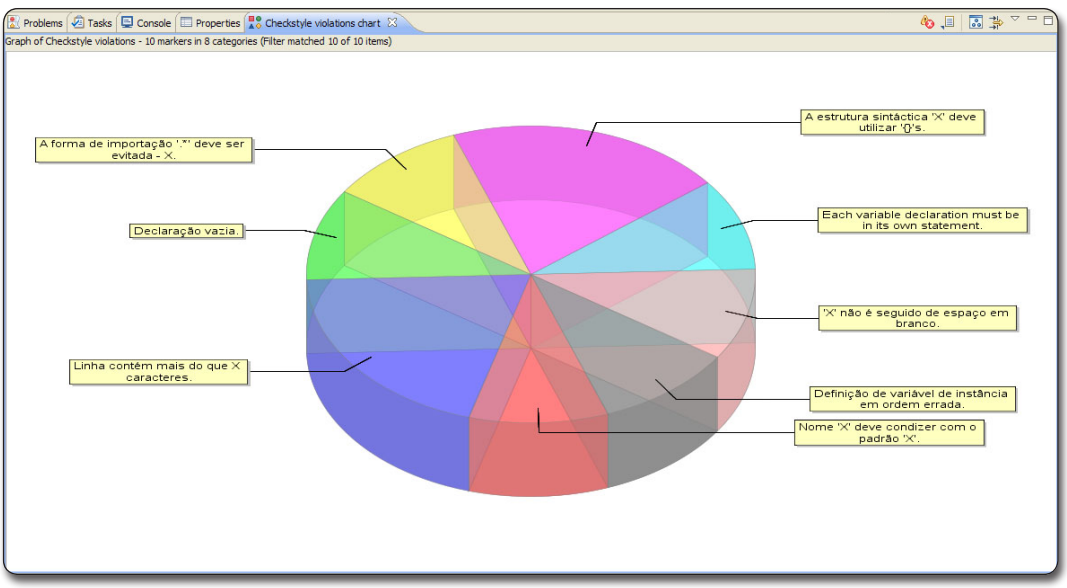


Figura 15. Janela ***Checkstyle violations chart***

de qualidade do código fonte é um trabalho difícil, sujeito a erros e omissões, além de consumir um tempo precioso.

A integração com ambientes de desenvolvimento, conforme apresentado nesse artigo, encoraja e agiliza a adoção sistematizada de ferramentas de qualidade de código. Os ganhos no desenvolvimento de software podem ser observados em códigos mais seguros e coesos, de mais fácil entendimento e manutenção, entre outros aspectos relevantes. ●

Links

Página oficial do projeto Checkstyle
http://checkstyle.sourceforge.net/

The Java Language Specification
http://java.sun.com/docs/books/jls/

Code Conventions for the JavaTM Programming Language:
http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

Página oficial do projeto Eclipse-CS
http://eclipse-cs.sourceforge.net/

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