CAPÍTULO

25 MÉTRICAS DE PROCESSO E PROJETO

CONCEITOS-CHAVE

A medição nos permite obter o entendimento do processo e projeto, dando-nos um mecanismo para uma avaliação objetiva. Lord Kelvin disse certa vez:

> Quando você pode medir o que você está falando e expressar em números, você sabe alguma coisa sobre aquilo; mas quando você não pode medir, quando você não pode expressar em números, o seu conhecimento é algo escasso e insatisfatório: pode ser o começo do conhecimento, mas você avançou muito pouco, em seu raciocínio, para o estágio de uma ciência.

A comunidade de engenharia de software levou a sério as palavras de Lord Kelvin. Mas não sem frustração e mais do que um pouco de controvérsia!

Medições podem ser aplicadas ao processo de software com a intenção de melhoria contínua. As medições podem ser usadas durante um projeto para ajudar nas estimativas, controle de qualidade, produtividade, e controle de projeto. Finalmente, a medição pode ser usada pelos engenheiros de software para ajudar a avaliar a qualidade dos artefatos e auxiliar na tomada de decisões táticas na medida em que o projeto evolui (Capítulo 23).

Dentro do contexto do processo de software e dos projetos que são executados usando o processo, uma equipe de software se preocupa primariamente com as métricas de produtividade e qualidade – medidas do "resultado" do desenvolvimento de software em função do esforço e do tempo aplicados e medidas da "adequação para uso" dos artefatos que são produzidos. Para fins de planejamento e estimativa, o seu interesse é histórico. Qual foi a produtividade do desenvolvimento de software nos projetos anteriores? Qual foi a qualidade do software produzido? Como a produtividade passada e os dados de qualidade podem ser extrapolados para o presente? Como isso pode ajudá-lo a fazer planos e estimativas mais precisos?

PANORAMA

O que é? As métricas de projeto e processo de software são medidas quantitativas que permitem que você tenha o discernimento sobre a eficácia do processo de software e os projetos que são executados usando o processo como uma estrutura. São coletados dados básicos de qualidade e produtividade. Esses dados são então analisados, comparados com médias passadas, e avaliados para determinar se ocorreram melhorias de qualidade e produtividade. As métricas são usadas também para apontar áreas com problemas, de modo que as correções possam ser desenvolvidas e o processo de software melhorado.

Quem realiza? Métricas de software são analisadas e avaliadas por gerentes de software. As medidas muitas vezes são coletadas por engenheiros de software.

Por que é importante? Se você não medir, o julgamento só pode ser baseado em uma avaliação subjetiva. Com a medição, tendências (tanto boas quanto ruins) podem ser detectadas, podem ser feitas melhores estimativas e melhorias significativas podem ser obtidas ao longo do tempo.

Quais são as etapas envolvidas? Comece definindo um conjunto limitado de medidas de processo, projeto e produto que sejam fáceis de coletar. Essas medidas são muitas vezes normalizadas usando-se métricas orientadas a tamanho ou função. O resultado é analisado e comparado com médias anteriores para projetos similares executados na organização. São avaliadas as tendências e são obtidas as conclusões.

Qual é o artefato? Uma série de métricas de software que proporcionam discernimento sobre o processo e possibilitam entender o projeto.

Como garantir que o trabalho foi realizado corretamente? Aplicando um esquema de medições consistente, embora simples que nunca deve ser usado para avaliar, recompensar, ou punir indivíduos por seu desempenho pessoal.

Em seu guia sobre medições de software, Park, Goethert, e Florac [Par96b] observam as razões por que medimos: (1) *caracterizar,* em um esforço para obter um conhecimento "de processos, produtos, recursos, e ambientes, e para estabelecer linhas de base para comparações com futuras avaliações "; (2) *avaliar* "para determinar o status com relação aos planos"; (3) *prever* "entendendo as relações entre os processos e produtos e criando modelos dessas relações"; e (4) *melhorar* "identificando etapas, causas raiz de problemas, ineficiências e outras oportunidades de melhoria da qualidade do produto e desempenho do processo".

A medição é uma ferramenta de gerenciamento. Se for usada adequadamente, ela proporciona discernimento ao gerente de projeto. E, consequentemente, ela ajuda o gerente de projeto e a equipe de software a tomar decisões que levarão a um projeto bem-sucedido.

25.1 Métricas no Domínio de Processo e Projeto

PONTO-CHAVE

As métricas de processo têm impacto de longo prazo. Seu objetivo é melhorar o próprio processo. As métricas de projeto muitas vezes contribuem para o desenvolvimento de métricas de processo.

Métricas de processo são coletadas através de todos os projetos e sobre longos períodos de tempo. Sua finalidade é proporcionar uma série de indicadores de processo que levam à melhoria do processo de software no longo prazo. *Métricas de projeto* permitem ao gerente de projeto de software (1) avaliar o estado de um projeto em andamento, (2) rastrear os riscos em potencial, (3) descobrir áreas problemáticas antes que elas se tornem "críticas", (4) ajustar o fluxo de trabalho ou tarefas, e (5) avaliar a habilidade da equipe de projeto para controlar a qualidade dos produtos finais de software.

Medidas que são coletadas por uma equipe de projeto e convertidas em métricas para uso durante um projeto podem também ser transmitidas para aqueles que têm responsabilidade pelo aperfeiçoamento do processo de software (Capítulo 30). Por essa razão, as mesmas métricas são usadas, muitas delas, tanto nos domínios de processo quanto de projeto.

25.1.1 Métricas de processo e aperfeiçoamento do processo de software

A única maneira racional de melhorar qualquer processo é medir atributos específicos do processo, desenvolver uma série de métricas significativas com base nesses atributos, e então usar as métricas para fornecer indicadores que levarão a uma estratégia de aperfeiçoamento (Capítulo 30). Mas antes de discutirmos as métricas de software e seu impacto sobre a melhoria do processo de software, é importante notar que o processo é apenas um item dentre uma série de "fatores controláveis na melhoria da qualidade do software e do desempenho organizacional" [Pau94].

De acordo com a Figura 25.1, o processo está no centro do triângulo que conecta três fatores que possuem uma profunda influência sobre a qualidade do software e desempenho organizacional. Já foi mostrado [Boe81] que a habilidade e a motivação das pessoas representam o fator individual mais influente na qualidade e no desempenho. A complexidade do produto pode ter um impacto substancial sobre a qualidade e desempenho da equipe. A tecnologia, (isto é, os métodos e ferramentas de engenharia de software) que preenche o processo tem também um impacto.

PONTO-CHAVE

A habilidade e motivação dos profissionais que fazem o trabalho são os fatores mais importantes que influenciam na qualidade do software.

Além disso, o triângulo do processo encontra-se dentro de um círculo de condições ambientais que inclui o ambiente de desenvolvimento (por exemplo, ferramentas de software integradas), condições de negócios (por exemplo, prazos de entrega, regras de negócio) e características do cliente (por exemplo, facilidade de comunicação e colaboração).

Você só pode medir a eficácia de um processo de software indiretamente. Isto é, você cria uma série de métricas baseadas nos resultados que podem ser obtidos do processo. Os resultados incluem medidas de erros descobertos antes da entrega do software, defeitos relatados pelos usuários finais, produtos acabados fornecidos (produtividade), esforço humano gasto, tempo despendido, conformidade com o cronograma, e outras medidas. Você pode também obter métricas de processo medindo as características de tarefas específicas de engenharia de software. Por exemplo, você pode medir o esforço e o tempo despendidos executando as atividades genéricas de engenharia de software descritas no Capítulo 2.

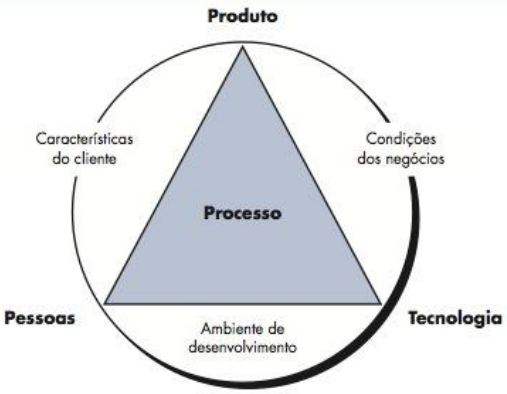


FIGURA 25.1 **Determinantes para a qualidade do software e eficácia organizacional** ***Fonte:*** **Adaptado de (Pau94)**

"As métricas de software permitem que você saiba quando deve rir e quando deve chorar."

Tom Gilb

Grady [Gra92] afirma que há usos "privados e públicos" para diferentes tipos de dados de processo. Como é natural que os engenheiros de software individualmente podem ser sensíveis ao uso de métricas coletadas individualmente, esses dados devem ser privados e servir apenas como um indicador individual. Exemplos de *métricas privadas* incluem taxas de defeito (por indivíduo), taxas de defeito (por componente) e erros encontrados durante o desenvolvimento.

A filosofia de "dados privados do processo" combina bem com a abordagem Personal Software Process (Capítulo 2) proposta por Humphrey [Hum97]. Ele reconhece que o aperfeiçoamento do processo de software pode e deve começar em nível individual. Dados privados do processo podem servir como um motivador importante quando você trabalha para melhorar a sua abordagem de engenharia de software.

Qual é a diferença entre usos público e privado para as métricas de software?

Algumas métricas de processo são privadas para a equipe de projeto de software, mas são públicas para todos os membros da equipe. Exemplos incluem defeitos relatados para funções principais do software (que foram desenvolvidas por vários profissionais), erros encontrados durante revisões técnicas, e linhas de código ou pontos de função por componente ou função.[1] A equipe examina esses dados para descobrir indicadores que podem melhorar o desempenho da equipe.

Métricas públicas geralmente assimilam informações que originalmente eram privadas aos indivíduos e equipes. Taxas de defeito em nível de projeto (absolutamente não atribuídas aos indivíduos), esforço, prazos agendados, e dados relacionados são coletados e avaliados tentando descobrir indicadores que podem melhorar o desempenho do processo organizacional.

Métricas de processo de software podem produzir benefícios significativos quando uma organização trabalha para melhorar seu nível geral de maturidade de processo. No entanto, assim como todas as métricas, essas podem ser mal utilizadas, criando mais problemas do que elas podem resolver. Grady [Gra92] sugere uma "etiqueta de métricas de software " apropriada para os gerentes e para os profissionais quando instituem um programa de métricas de processo:

Que diretrizes devem ser aplicadas quando coletamos métricas de software?

- Use bom senso e sensibilidade organizacional ao interpretar dados de métricas.
- Forneça regularmente feedback aos indivíduos e equipes que coletam medidas e métricas.
- Não use métricas para avaliar indivíduos.
- Trabalhe com profissionais e equipes para definir objetivos claros e métricas que serão usadas para alcançá-las.

1 Métricas de linhas de código e pontos de função são discutidas nas Seções 25.2.1 e 25.2.2.

- Nunca use métricas para ameaçar indivíduos ou equipes.
- Dados de métricas que indicam uma área com problema não deverão ser considerados "negativos". Esses dados são simplesmente um indicador para melhoria do processo.
- Não seja obsessivo sobre uma única métrica excluindo as outras métricas importantes.

À medida em que uma organização se sente mais à vontade, coletando e usando métricas de processo, a derivação de indicadores simples dá margem a uma abordagem mais rigorosa chamada *melhoria estatística de processo de software* (*statistical software process improvement* – SSPI). Essencialmente, a SSPI usa a análise de falhas de software para coletar informações sobre todos os erros e defeitos[2] encontrados quando uma aplicação, sistema, ou produto é desenvolvido e usado.

25.1.2 Métricas de projeto

Diferentemente das métricas de processo de software que são usadas para fins estratégicos, as medidas de projeto de software são táticas. Isto é, métricas de projeto e os indicadores derivados delas são usados por um gerente de projeto e uma equipe de software para adaptar o fluxo de trabalho do projeto e as atividades técnicas.

A primeira aplicação das métricas de projeto na maioria dos projetos de software ocorre durante as estimativas. Métricas coletadas de projetos passados são usadas como base a partir da qual são feitas as estimativas de esforços e tempo para o trabalho atual de software. Na medida em que um projeto progride, medidas de esforço e tempo despendidos são comparadas com as estimativas originais (e com o cronograma do projeto). O gerente de projeto usa esses dados para monitorar e controlar o progresso.

Quando o trabalho técnico começa, outras métricas de projeto começam a ter significado. São medidas as taxas de produção representadas em termos de modelos criados, revisões, pontos de função e linhas de código-fonte fornecidas. Além disso, são rastreados os erros descobertos durante cada tarefa de engenharia de software. Na medida em que o software evolui dos requisitos para o projeto, métricas técnicas (Capítulo 23) são coletadas para avaliar a qualidade do projeto e fornecer indicadores que terão influência na abordagem adotada para geração de código e teste.

Como devemos usar as métricas durante o próprio projeto?

O objetivo das métricas de projeto é duplo. Primeiro, essas métricas são usadas para minimizar o cronograma de desenvolvimento fazendo os ajustes necessários para evitar atrasos e mitigar problemas e riscos em potencial. Segundo, as métricas de projeto são usadas para avaliar a qualidade do produto de forma contínua e, quando necessário, modificar a abordagem técnica para melhorar a qualidade.

À medida que a qualidade melhora, os defeitos são minimizados, e à medida que a contagem de defeitos diminui, a quantidade de retrabalho necessário durante o projeto também é reduzida. Isso leva a uma redução no custo total do projeto.

25.2 Medidas de Software

"Nem tudo o que pode ser contado importa, e nem tudo o que importa pode ser contado."

Albert Einstein

No Capítulo 23, observamos que as medidas no mundo físico podem ser classificadas de duas maneiras: medidas diretas (por exemplo, o comprimento de um parafuso) e medidas indiretas (por exemplo, a "qualidade" dos parafusos produzidos, medida contando os rejeitos). As métricas de software podem ser classificadas de maneira similar.

Medidas diretas do processo de software incluem custos e trabalho aplicado. Medidas diretas do produto incluem linhas de código (lines of code - LOC) produzidas, velocidade de execução, tamanho de memória e defeitos relatados durante um determinado período de tempo. Medidas

2 Neste livro, um *erro* é definido como alguma falha em um artefato que é descoberta *antes* que o software seja fornecido ao usuário final. Um *defeito* é uma falha descoberta *depois* que o software é fornecido ao usuário final. Devemos destacar que muitas pessoas não fazem essa distinção.

CasaSegura

Estabelecendo uma abordagem de métricas

A cena: Escritório de Doug Miller, quando o projeto de software *CasaSegura* está para começar.

Os atores: Doug Miller (gerente da equipe de engenharia de software *CasaSegura*) e Vinod Raman e Jamie Lazar, membros da equipe de engenharia de artefatos.

Conversa:

Doug: Antes de começarmos a trabalhar neste projeto, eu gostaria que vocês definissem e coletassem uma série de métricas simples. Para começar, vocês terão que definir nossas metas.

Vinod (com cara de contrariado): Nós nunca fizemos isso antes, e...

Jamie (interrompendo): E com base na administração do tempo sobre a qual temos conversado, nós nunca teremos tempo. Afinal, essas métricas são mesmo boas?

Doug (erguendo a mão para interromper a discussão): Calma, respirem fundo rapazes. O fato de nunca termos feito isso antes é a melhor razão para começar a fazer agora, e o trabalho com as métricas das quais eu estava falando não deve tomar muito tempo... Na verdade, elas podem até nos poupar tempo.

Vinod: Como?

Doug: Olhe, faremos muito mais trabalho interno de engenharia de software à medida que nossos produtos se tornarem mais inteligentes, mais voltados à Web, tudo isso... E precisaremos entender o processo que usamos para criar o software... E melhorá-lo para criarmos um software melhor. A única maneira de fazer isso é por meio das medições.

Jamie: Mas estamos sendo pressionados no prazo, Doug. Não sou a favor de gerar mais papel... Precisamos de tempo para fazer nosso trabalho, não para coletar dados.

Doug (calmamente): Jamie, o trabalho de um engenheiro envolve coleta de dados, avaliação desses dados, e o uso do resultado para melhorar o produto e o processo. Estou errado?

Jamie: Não, mas...

Doug: E que tal se mantivermos o número de medidas que coletamos em não mais de cinco ou seis e focarmos a qualidade?

Vinod: Ninguém pode argumentar contra a alta qualidade...

Jamie: Certo... Mas, eu não sei. Eu ainda acho que não é necessário.

Doug: Peço que sejam tolerantes comigo neste ponto. O que vocês sabem sobre métricas?

Jamie (olhando para Vinod): Não muito.

Doug: Aqui estão algumas referências da Web... Dediquem algumas horas para se informar.

Jamie (sorrindo): Eu achei que você tinha falado que isso não tomaria muito tempo.

Doug: O tempo que você gasta aprendendo nunca é perdido... Façam isso e então estabeleceremos nossas metas, façam algumas perguntas, e definam as métricas que precisamos coletar.

indiretas do produto incluem funcionalidade, qualidade, complexidade, eficiência, confiabilidade, manutenibilidade, e muitas outras "ades" que são discutidas no Capítulo 14.

O custo e trabalho requerido para criar o software, o número de linhas de código produzidas e outras medidas diretas são relativamente fáceis de coletar, desde que sejam estabelecidas antecipadamente convenções para as medições. No entanto, a qualidade e a funcionalidade do software ou sua eficiência ou manutenibilidade são mais difíceis de avaliar e podem ser medidas somente de forma indireta.

Como muitos fatores afetam o trabalho de software, não use métricas para comparar indivíduos ou equipes.

Nós dividimos o domínio de métricas de software em processo, projeto, e métricas de produto e observamos que as métricas de produto que são privadas a um indivíduo são muitas vezes combinadas para desenvolver métricas de projeto que são públicas para a equipe de software. Métricas de projeto são então consolidadas para criar métricas de processo que são públicas à organização de software como um todo. Mas como uma organização combina métricas que vieram de diferentes indivíduos ou projetos?

Para ilustrar, considere um exemplo simples. Indivíduos em duas equipes de projeto diferentes registram e classificam todos os erros que eles encontram durante o processo de software. As medidas individuais são então combinadas para criar medidas de equipe. A Equipe A encontrou 342 erros durante o processo de software antes da entrega. A Equipe B encontrou 184 erros. Sendo iguais todas as outras coisas, qual equipe é mais eficaz na descoberta de erros em todo o processo? Como você não sabe qual é o tamanho ou complexidade dos projetos, você não pode responder a essa pergunta. No entanto, se as medidas forem normalizadas, é possível criar métricas de software que possibilitam a comparação com médias organizacionais mais amplas.

FIGURA 25.2
Métricas orientadas a tamanho

Projeto	Linhas de código	Esforço	$(000)	Pág. doc.	Erros	Defeitos	Pessoas
alfa	12.100	24	168	365	134	29	3
beta	27.200	62	440	1224	321	86	5
gama	20.200	43	314	1050	256	64	6
⋮	⋮	⋮	⋮	⋮	⋮		

25.2.1 Métricas orientadas a tamanho

Métricas de software orientadas a tamanho são criadas normalizando-se as medidas de qualidade e/ou produtividade considerando o *tamanho* do software que foi produzido. Se uma organização de software mantém registros simples, pode ser criada uma tabela de medidas orientadas a tamanho, como aquela mostrada na Figura 25.2. A tabela lista cada projeto de desenvolvimento de software que foi completado durante alguns anos passados e medidas correspondentes para aquele projeto. Olhando a linha da tabela (Figura 25.2) para o projeto alfa: 12.100 linhas de código foram criadas com 24 pessoas-mês de trabalho a um custo de $168.000. Deve-se observar que todo o trabalho e custo registrados na tabela representa todas as atividades de engenharia de software (análise, projeto, código e teste), não apenas codificação. Outras informações para o projeto alfa indicam que foram criadas 365 páginas de documentação, foram registrados 134 erros antes da entrega do software e foram encontrados 29 defeitos após a entrega para o cliente durante o primeiro ano de operação. Três pessoas trabalharam no desenvolvimento do software para o projeto alfa.

Para desenvolver métricas que podem ser assimiladas com métricas similares de outros projetos, você pode escolher número de linhas de código como um valor de normalização. A partir dos dados rudimentares contidos na tabela, pode ser desenvolvido um conjunto de métricas simples orientadas a tamanho para cada projeto:

- Erros por KLOC (mil linhas de código)
- Defeitos por KLOC
- $ por KLOC
- Páginas de documentação por KLOC

Além disso, podem ser computadas outras métricas interessantes:

- Erros por pessoa-mês
- KLOC por pessoa-mês
- $ por página de documentação

Métricas orientadas por tamanho não são aceitas universalmente como a melhor maneira de medir os processos de software. A maior parte da controvérsia gira em torno do uso de linhas de código como medida principal. Os proponentes da medida LOC (linhas de código) argumentam que LOC é um "item" de todos os projetos de desenvolvimento de software que pode ser facilmente contado, que muitos modelos de estimativa de software existentes usam LOC ou KLOC como dado de entrada principal, e que já existe uma grande quantidade de literatura e

PONTO-CHAVE

Métricas orientadas a tamanho são amplamente usadas, mas continua o debate sobre sua validade e aplicabilidade.

dados baseados em LOC. Por outro lado, os oponentes argumentam que as medidas LOC são dependentes da linguagem de programação, que quando é considerada a produtividade, elas penalizam programas bem projetados, mas menores; que elas não podem facilmente acomodar linguagens não procedurais; e que seu uso nas estimativas requer um nível de detalhe que pode ser difícil de alcançar (isto é, o planejador deve estimar a LOC para ser produzida bem antes que a análise e o projeto estejam completados).

25.2.2 Métricas orientadas a função

Métricas de software orientadas a função usam uma medida da funcionalidade fornecida pela aplicação como um valor de normalização. A métrica orientada a função mais amplamente usada é a pontos de função (function point – FP). O cálculo de pontos de função é baseada nas características de domínio de informação e complexidade do software. O mecanismo de cálculo de FP foi discutido no Capítulo 23.[3]

O ponto de função, assim como a medida LOC, é controverso. Os proponentes argumentam que essa função é independente da linguagem de programação, tornando-a ideal para aplicações que usam linguagens convencionais e não procedurais, e que é baseada em dados que têm maior probabilidade de ser conhecidos na evolução de um projeto, tornando a FP mais atraente como abordagem de estimativa. Os oponentes argumentam que o método requer um pouco de "jeitinho", porque o cálculo é baseado em dados subjetivos ao invés de objetivos, que as contagens do domínio de informações (e outras dimensões) podem ser difíceis de coletar após o fato, e que a FP não tem um significado físico direto – é apenas um número.

25.2.3 Reconciliando métricas LOC e FP

A relação entre linhas de código e pontos de função depende da linguagem de programação que é usada para implementar o software e a qualidade do projeto. Muitos estudos já tentaram relacionar medidas FP e LOC. A tabela a seguir[4] [QSM02] fornece estimativas aproximadas da média do número de linhas de código necessárias para criar um ponto de função em várias linguagens de programação:

Uma revisão desses dados indica que uma LOC de C++ fornece aproximadamente 2,4 vezes a "funcionalidade" (em média) de uma LOC de C. Além disso, uma LOC de Smalltalk fornece pelo menos quatro vezes a funcionalidade de uma LOC para uma linguagem de programação convencional como Ada, COBOL, ou C. Usando as informações da tabela, é possível "retroagir" [Jon98] o software existente para estimar o número de pontos de função, uma vez conhecido o número total de instruções da linguagem de programação.

Medidas de LOC e FP muitas vezes são usadas para derivar métricas de produtividade. Isso leva invariavelmente a um debate sobre o uso de tais dados. Deveria o valor LOC/pessoa-mês (ou FP/pessoa-mês) de um grupo ser comparado com dados similares de outro grupo? Os gerentes deveriam avaliar o desempenho dos indivíduos usando essas métricas? A resposta a essas questões é um enfático "Não!" A razão para essa resposta é que muitos fatores influenciam a produtividade, gerando comparações do tipo "maçãs com laranjas", que podem facilmente ser mal interpretadas.

Descobriu-se que métricas baseadas em pontos de função e LOC são indicadores relativamente precisos do trabalho e custo do desenvolvimento de software. No entanto, para usar LOC e FP para estimativas (Capítulo 26), deve ser estabelecida uma referência histórica de informações.

Dentro do contexto de métricas de processo e projeto, você deve se preocupar em primeiro lugar com a produtividade e a qualidade – medidas de "resultado" de desenvolvimento de software em função do esforço e tempo aplicados e as medidas da "adequação para uso " dos produtos que são criados. Para fins de melhoria de processo e planejamento de projeto, seu interesse é histórico. Qual foi a produtividade do desenvolvimento de software nos projetos passados?

3 Veja na Seção 23.2.1 uma discussão detalhada da computação FP.

4 Usado com permissão de Quantitative Software Management (www.qsm.com), copyright 2002.

	LOC por Ponto de Função			
Linguagem de Programação	**Média**	**Mediana**	**Baixa**	**Alta**
Access	35	38	15	47
Ada	154	—	104	205
APS	86	83	20	184
ASP 69	62	—	32	127
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
Clipper	38	39	27	70
COBOL	77	77	14	400
Cool:Gen/IEF	38	31	10	180
Culprit	51	—	—	—
DBase IV	52	—	—	—
Easytrieve+	33	34	25	41
Excel 47	46	—	31	63
Focus	43	42	32	56
FORTRAN	—	—	—	—
FoxPro	32	35	25	35
Ideal	66	52	34	203
IEF/Cool:Gen	38	31	10	180
Informix	42	31	24	57
Java	63	53	77	—
JavaScript	58	63	42	75
JCL	91	123	26	150
JSP	59	—	—	—
Lotus Notes	21	22	15	25
Mantis	71	27	22	250
Mapper	118	81	16	245
Natural	60	52	22	141
Oracle	30	35	4	217
PeopleSoft	33	32	30	40
Perl	60	—	—	—
PL/1	78	67	22	263
Powerbuilder	32	31	11	105
REXX	67	—	—	—
RPG II/III	61	49	24	155
SAS	40	41	33	49
Smalltalk	26	19	10	55
SQL	40	37	7	110
VBScript 36	34	27	50	—
Visual Basic	47	42	16	158

Qual foi a qualidade do software produzido? Como os dados de produtividade e qualidade do passado podem ser extrapolados para o presente? Como isso pode nos ajudar a melhorar o processo e planejar novos projetos mais precisamente?

25.2.4 Métricas orientadas a objeto

Métricas de projeto de software convencional (LOC ou FP) podem ser usadas para estimar projetos de software orientados a objeto. No entanto, essas métricas não fornecem granularidade suficiente para os ajustes de cronograma e esforço que são necessários na medida em que você passa por iterações por meio de um processo evolucionário ou incremental. Lorenz e Kidd [Lor94] sugerem o seguinte conjunto de métricas para projetos orientados a objeto:

Número de scripts de cenário (*Number of scenario scripts*). Um script de cenário (análogo aos casos de uso discutidos na Parte 2 deste livro) é uma sequência detalhada de passos que descrevem a interação entre o usuário e a aplicação. Cada script é organizado em trios da forma

{**iniciador,** *ação,* **participante**}

onde **iniciador** é o objeto que solicita algum serviço (que inicia uma mensagem), *ação* é o resultado da solicitação, e **participante** é o objeto servidor que satisfaz a solicitação. O número de scripts de cenário está correlacionado diretamente com o tamanho da aplicação e com o número de casos de testes que devem ser desenvolvidos para exercitar o sistema depois que ele é construído.

Não é raro ocorrer que scripts de múltiplos cenários mencionem a mesma funcionalidade ou objetos dados. Portanto, tenha cuidado ao usar contagem de scripts. Muitos scripts podem às vezes ser reduzidos a uma única classe ou conjunto de código.

Número de classes-chave (*Number of key classes*). *Classes-chave* (*Key classes*) são os "componentes altamente independentes " [Lor94] que são definidos logo no início em análise orientada a objeto (Capítulo 6).[5] Como as classes-chave são essenciais ao domínio do problema, a quantidade dessas classes é uma indicação da quantidade de esforço necessário para desenvolver o software e também uma indicação do potencial de reutilização a ser aplicado durante o desenvolvimento do sistema.

Número de classes de apoio (*Number of support classes*). *Classes de apoio* (*Support classes*) são necessárias para implementar o sistema, mas não estão imediatamente relacionadas com o domínio do problema. Como exemplos podemos citar as classes de interface de usuário (GUI), classes de acesso e manipulação de bases de dados, e classes de cálculo. Além disso, podem ser desenvolvidas classes de apoio para cada uma das classes-chave. As *classes de apoio* são definidas iterativamente durante um processo evolucionário. O número de classes de apoio é uma indicação da quantidade de esforço necessário para desenvolver o software e também uma indicação do potencial de reutilização a ser aplicado durante o desenvolvimento do sistema.

As classes podem variar em tamanho e complexidade. Portanto, pense em classificar as contagens de classes por tamanho e complexidade.

Número médio de classes de apoio para cada classe-chave (*Average number of support classes per key class*). Em geral, as classes-chave são conhecidas logo no início do projeto. As classes de apoio são definidas durante o projeto. Se o número médio de classes de apoio para cada classe-chave fosse conhecido para um dado domínio de problema, a estimativa (baseada no número total de classes) seria muito simplificada. Lorenz e Kidd sugerem que as aplicações com uma GUI tenham de duas a três vezes a quantidade de classes suporte como classes-chave. Aplicações não GUI têm de uma a duas vezes a quantidade de classes suporte como classes-chave.

Número de subsistemas (*Number of subsystems*). Um *subsistema* é uma agregação de classes que apoia uma função que é visível ao usuário final de um sistema. Uma vez identificados os subsistemas, é mais fácil elaborar um cronograma razoável no qual o trabalho nos subsistemas é dividido entre o pessoal de projeto.

---

5 Nós nos referimos às classes-chave como classes de análise na Parte 2 deste livro.

Para serem usadas eficazmente em um ambiente de engenharia de software orientado a objeto, métricas similares àquelas mencionadas acima deverão ser coletadas juntamente com as medidas de projeto, tais como o esforço gasto, erros e defeitos descobertos, e modelos ou páginas de documentação produzidas. Na medida em que a base de dados cresce (após completar um grupo de projetos), as relações entre as medidas orientadas a objeto e as medidas de projeto fornecerão métricas que podem ajudar nas estimativas do projeto.

25.2.5 Métricas orientadas a casos de uso

Os casos de uso[6] são amplamente usados como método para descrever requisitos no nível dos clientes ou domínio de negócio que sugerem características e funções de software. Seria considerado razoável usar o caso de uso como uma medida de normalização similar a LOC ou FP. Assim como a FP, o caso de uso é definido no início no processo de software, permitindo que ele seja usado para estimativas antes de iniciar atividades significativas de modelagem e construção. Os casos de uso descrevem (indiretamente, pelo menos) funções e características visíveis ao usuário que são requisitos básicos para um sistema. O caso de uso é independente da linguagem de programação. Além disso, o número de casos de uso é diretamente proporcional ao tamanho do aplicativo em LOC e ao número de casos de testes que terão de ser projetados para exercitar completamente o aplicativo.

Como os casos de uso podem ser criados em níveis muito diferentes de abstração, não há um "tamanho" padrão para um caso de uso. Sem uma medida padronizada do que é um caso de uso, sua aplicação como medida de normalização (por exemplo, esforço gasto por cada caso de uso) é suspeita.

Os pesquisadores têm sugerido os *pontos de casos de uso* (UCPs) como um mecanismo para estimar trabalho de projeto e outras características. O UCP é uma função do número de atores e transações deduzidas pelos modelos de casos de uso e é análogo ao FP em alguns aspectos. Se você tiver mais interesse, veja [Cle06].

25.2.6 Métricas de projeto WebApp

O objetivo de todos os projetos WebApp é fornecer uma combinação de conteúdo e funcionalidade para o usuário final. Medidas e métricas usadas para projetos tradicionais de engenharia de software são difíceis de traduzir diretamente para WebApps. No entanto, é possível desenvolver uma base de dados que permite acesso a medidas internas de produtividade e qualidade obtidas em uma série de projetos. Entre as medidas que podem ser coletadas estão:

Número de páginas Web estáticas. Páginas Web com conteúdo estático (isto é, o usuário final não tem controle sobre o conteúdo mostrado na página) são as mais comuns de todas as características WebApp. Essas páginas representam baixa complexidade relativa e geralmente requerem menos esforços para ser criadas do que as páginas dinâmicas. Essa medida fornece uma indicação do tamanho global da aplicação e do esforço necessário para desenvolvê-la.

Número de páginas Web dinâmicas. Páginas Web com conteúdo dinâmico (isto é, as ações do usuário final ou outros fatores externos resultam em conteúdo personalizado apresentado na página) são essenciais em todos os aplicativos e-commerce, recursos de pesquisa, aplicações financeiras e muitas outras categorias WebApp. Essas páginas representam uma complexidade relativa maior e requerem mais esforço para ser construídas do que as páginas estáticas. Essa medida fornece uma indicação do tamanho geral do aplicativo e o esforço necessário para desenvolvê-lo.

Número de links de páginas internos. Links de páginas internos são ponteiros que fornecem um hyperlink para alguma outra página Web dentro do WebApp. Essa medida fornece uma indicação do grau de acoplamento arquitetônico dentro do WebApp. Na medida em que o número de links de páginas aumenta, aumenta também o esforço gasto no projeto navegacional e na construção.

---

6 Casos de uso são apresentados nos Capítulos 5 e 6.

Número de objetos dados persistentes. Um ou mais objetos dados persistentes (por exemplo, uma base de dados ou um arquivo de dados) pode ser acessado por uma WebApp. Na medida em que cresce o número de objetos dados persistentes, a complexidade do WebApp também cresce e o esforço para implementá-la aumenta proporcionalmente.

Número de sistemas externos interfaceados. Os WebApps muitas vezes devem se interfacear com aplicações de negócio de "retaguarda". Na medida em que cresce o requisito para interfaceamento, a complexidade do sistema e esforço de desenvolvimento também aumenta.

Número de objetos de conteúdo estático. Objetos de conteúdo estático abrangem objetos estáticos baseados em texto, objetos gráficos, vídeo, animação e informações de áudio que são incorporadas dentro do WebApp. Objetos de conteúdo múltiplo podem aparecer em uma única página Web.

Número de objetos de conteúdo dinâmico. Objetos de conteúdo dinâmico são gerados baseados nas ações do usuário final e abrangem informações de texto geradas internamente, informações gráficas, vídeo, animação, e áudio, que são incorporadas dentro do WebApp. Objetos de conteúdo múltiplo podem aparecer em uma única página Web.

Número de funções executáveis. Uma função executável (por exemplo, um script ou applet) proporciona algum serviço de cálculo ao usuário final. Na medida em que o número de funções executáveis aumenta, os esforços de modelagem e construção também aumentam.

Cada uma das medidas que acabamos de mencionar pode ser determinada em um estágio relativamente antecipado. Por exemplo, você pode definir uma métrica que reflete o grau de personalização de usuário final requerida para o WebApp e correlacionar isso com o esforço gasto no projeto e/ou os erros descobertos na medida em que são feitas as revisões e testes. Para isso, você define

$$N_{sp} = \text{número de Páginas Web Estáticas}$$
$$N_{dp} = \text{número de Páginas Web Dinâmicas}$$

Então,

$$\text{Índice de personalização, } C = \frac{N_{dp}}{N_{dp} + N_{sp}}$$

O valor de C varia de 0 a 1. Na medida em que C se torna maior, o nível de personalização do WebApp se torna um aspecto técnico significativo.

Métricas WebApp similares podem ser calculadas e correlacionadas com medidas de projeto, tais como esforço gasto, erros e defeitos descobertos e modelos ou páginas de documentação produzidos. Na medida em que a base de dados cresce (após completar certo número de projetos), relações entre as medidas WebApp e medidas de projeto proporcionarão indicadores que podem ajudar na estimativa do projeto.

FERRAMENTAS DO SOFTWARE

Métricas de projeto e processo

Objetivo: auxiliar na definição, coleta, avaliação e relatórios de medidas e métricas de software.

Mecanismos: cada ferramenta varia em sua aplicação, mas todas elas fornecem mecanismos para coletar e avaliar dados que levam à computação de métricas de software.

Ferramentas Representativas:[7]

Function Point WORKBENCH, desenvolvida pela Charismatek (**www.charismatek.com.au**), oferece uma ampla gama de métricas orientadas para FP.

MetricCenter, desenvolvida pela Distributive Software (**www.distributive.com**), suporta coleta automática de dados, análise, formatação de gráficos, geração de relatórios e outras tarefas de medição.

PSM Insight, desenvolvida pela Practical Software and Systems Measurement (**www.psmsc.com**), ajuda na criação e subsequente análise de uma base de dados de medições de projeto.

SLIM tool set, desenvolvida pela QSM (**www.qsm.com**), contém um conjunto abrangente de métricas e ferramentas de estimativas.

SPR tool set, desenvolvida pela Software Productivity Research (**www.spr.com**), oferece uma coleção abrangente de ferramentas orientadas a FP.

TychoMetrics, desenvolvida pela Predicate Logic, Inc. (**www.predicate.com**), é um conjunto de ferramentas para coleta e relato de métricas de gerenciamento.

7 A citação de ferramentas aqui não representa um endosso, mas sim uma amostragem das ferramentas nessa categoria. Na maioria dos casos, os nomes das ferramentas são marcas registradas pelos seus respectivos desenvolvedores.

25.3 MÉTRICAS PARA QUALIDADE DE SOFTWARE

Software é uma entidade complexa. Portanto, deve-se esperar que ocorram erros na medida em que o produto é desenvolvido. As métricas de processo são destinadas a melhorar o processo de software para que os erros sejam descobertos pela maneira mais eficiente.

O principal objetivo da engenharia de software é produzir um sistema, aplicação ou produto, de alta qualidade, dentro de um prazo que satisfaça as necessidades do mercado. Para atingir esse objetivo, devem-se aplicar métodos eficazes, combinados com modernas ferramentas dentro do contexto de um processo de software bem desenvolvido. Além disso, um bom engenheiro de software (e bons gerentes de engenharia de software) devem medir se a alta qualidade será obtida.

A qualidade de um sistema, aplicação, ou produto, é apenas tão boa quanto os requisitos que descrevem o problema, o projeto que modela a solução, o código que leva ao programa executável, e os testes que exercitam o software para descobrir os erros. Você pode usar medidas para avaliar a qualidade dos requisitos e modelos de projeto, o código-fonte, e os casos de testes que foram criados enquanto o software é desenvolvido. Para conseguir essa avaliação em tempo real, você aplica métricas de produto (Capítulo 23) para avaliar a qualidade dos artefatos de software de maneira objetiva, e não subjetiva.

Um gerente de projeto deve também avaliar a qualidade enquanto o projeto avança. Métricas privadas coletadas por engenheiros de software individuais são combinadas para fornecer os resultados no nível de projeto. Embora muitas medidas de qualidade possam ser coletadas, a principal tendência no nível de projeto é medir erros e defeitos. Métricas derivadas dessas medidas proporcionam uma indicação da efetividade da garantia de qualidade de software individual e de grupo e das atividades de controle.

Métricas como erros de produto por ponto de função, erros descobertos por horas de revisão, e erros descobertos por horas de teste proporcionam informações sobre a eficácia de cada uma das atividades sugeridas pela métrica. Os dados sobre os erros também podem ser usados para calcular a *eficiência de remoção de defeitos* (*defect removal efficiency* – DRE) para cada atividade da estrutura de processo. A DRE é discutida na Seção 25.3.3.

25.3.1 Medição da qualidade

Embora existam muitas medidas de qualidade de software,[8] a correção, a manutenibilidade, integridade e usabilidade fornecem indicadores úteis para a equipe de projeto. Gilb [Gil88] sugere definições e medidas para cada uma delas.

Uma excelente fonte de informações sobre qualidade de software e tópicos relacionados (incluindo métricas) pode ser encontrada em **www.quality world.com**.

Correção. Um programa deve operar corretamente ou terá pouca utilidade para seus usuários. A correção é o grau com o qual o software executa sua função. A medida mais comum da exatidão é o número de defeitos por KLOC, onde um defeito é definido como uma ocorrência de falta de conformidade com os requisitos. Ao considerar a qualidade geral de um artefato, os defeitos são aqueles problemas relatados por um usuário do programa depois que o programa foi liberado para uso geral. Para fins de avaliação de qualidade, os defeitos são contados durante um período de tempo padrão, em geral um ano.

Manutenibilidade. A manutenção do software e o suporte exigem maiores esforços do que qualquer outra atividade de engenharia. A manutenibilidade é a facilidade com a qual um programa pode ser corrigido se for encontrado um erro, adaptado se o ambiente mudar, ou melhorado se o cliente desejar uma alteração nos requisitos. Não há uma maneira de medir a manutenibilidade diretamente, portanto, é preciso usar medidas indiretas. Uma métrica simples orientada por tempo é o *tempo médio de alteração* (*mean-time-to-change* – MTTC), o tempo necessário para analisar a solicitação de alteração, projetar uma modificação apropriada, implementar a alteração, testá-la, e distribuí-la a alteração para todos os usuários. Na média, os programas manuteníveis terão um MTTC mais baixo (para tipos equivalentes de alteração) do que programas que não são manuteníveis.

8 Uma discussão detalhada dos fatores que influenciam na qualidade do software e as métricas que podem ser usadas para avaliar a qualidade do software foi apresentada no Capítulo 23.

Integridade. A integridade do software vem se tornando cada vez mais importante na era dos terroristas e hackers cibernéticos. Esse atributo mede a habilidade de um sistema em resistir aos ataques (tanto acidentais quanto intencionais) à sua segurança. Os ataques podem ser feitos em todos os três componentes do software: programas, dados e documentos.

Para medir a integridade, devem ser definidos dois atributos adicionais: ameaça e segurança. *Ameaça* é a probabilidade (que pode ser estimada ou derivada de evidência empírica) de que um ataque de um tipo específico ocorrerá em um dado intervalo de tempo. *Segurança* é a probabilidade (que pode ser estimada ou derivada de evidência empírica) de que um ataque de um tipo específico será repelido. A integridade de um sistema pode então ser definida como:

$$\text{Integridade} = \Sigma\ [1 - (\text{ameaça} \times (1 - \text{segurança}))]$$

Por exemplo, se a ameaça (probabilidade de que um ataque possa ocorrer) for 0,25 e a segurança (a possibilidade de repelir o ataque) for 0,95, a integridade do sistema é 0,99 (muito alta). Por outro lado, se a probabilidade de ameaça for 0,50 e a possibilidade de repelir um ataque for de apenas 0,25, a integridade do sistema é 0,63 (muito baixa e inaceitável).

Usabilidade. Se um programa não for fácil de usar, muitas vezes ele está destinado ao fracasso, mesmo que as funções que ele executa sejam valiosas. A usabilidade é uma tentativa de quantificar a facilidade de uso e pode ser medida em termos das características apresentadas no Capítulo 11.

Os quatro fatores que acabamos de descrever são apenas alguns exemplos daqueles que foram propostos como medidas para a qualidade do software. O Capítulo 23 considera este tópico com mais detalhes.

25.3.2 Eficiência na remoção de defeitos

Uma métrica de qualidade que traz benefícios tanto para o projeto quanto para o processo é a eficiência na remoção de defeitos (*defect removal efficiency* – DRE). Essencialmente, a DRE é uma medida da habilidade de filtragem das ações de garantia de qualidade e controle quando elas são aplicadas através de todas as atividades da estrutura de processo.

Quando considerada para um projeto como um todo, a DRE é definida da seguinte maneira:

$$\text{DRE} = \frac{E}{E + D}$$

Se a DRE for baixa quando você faz a análise e projeto, dedique algum tempo melhorando a maneira como você executa as revisões técnicas.

onde E é o número de erros encontrados antes que o software seja fornecido ao usuário final e D é o número de defeitos depois que o software é entregue.

O valor ideal para DRE é 1. Isto é, nenhum defeito é encontrado no software. Realisticamente, D será maior do que 0, mas o valor de DRE pode ainda se aproximar de 1 à medida em que E aumenta para um dado valor de D. De fato, na medida em que E aumenta, é provável que o valor final de D diminua (os erros são filtrados antes de se tornarem defeitos). Se for usada como uma métrica que fornece um indicador da habilidade de filtragem das atividades de controle de qualidade e segurança, a DRE estimula a equipe de projeto de software a instituir técnicas para encontrar o maior número possível de erros antes da entrega do software.

A DRE também pode ser usada no projeto para avaliar a habilidade de uma equipe para encontrar erros antes que eles passem para a próxima atividade na estrutura do software ou para a próxima ação da engenharia de software. Por exemplo, a análise de requisitos produz um modelo de requisitos que pode ser examinado para encontrar e corrigir erros. Aqueles erros que não são detectados durante a revisão do modelo de requisitos passam adiante para o projeto (onde eles podem ser ou não encontrados). Quando usada nesse contexto, redefinimos a DRE como

$$\text{DRE}_i = \frac{E_i}{E_i + E_{i+1}}$$

onde *Ei* é o número de erros encontrados durante a ação de engenharia de software *i* e E_{i+1} é o número de erros encontrados durante a ação de engenharia de software $i + 1$ que estão ligados a erros que não foram descobertos na ação de engenharia de software *i*.

Um objetivo de qualidade para uma equipe de software (ou um engenheiro de software individual) é conseguir que uma DRE_i que se aproxime de 1. Isto é, os erros deverão ser filtrados antes que eles passem para a próxima atividade ou ação.

CASASEGURA

Estabelecendo uma abordagem de métricas

A cena: Escritório de Doug Miller, dois dias após a reunião inicial sobre métricas de software.

Os atores: Doug Miller (gerente da equipe de engenharia de software *CasaSegura*), Vinod Raman e Jamie Lazar, membros da equipe de engenharia de artefatos de software.

Conversa:

Doug: Vocês dois tiveram oportunidade de aprender um pouco sobre métricas de processo e projeto?

Vinod and Jamie: [Ambos acenam a cabeça afirmativamente]

Doug: É sempre uma boa ideia estabelecer metas quando você adota qualquer métrica. Quais são as suas?

Vinod: Nossas métricas deverão focalizar a qualidade. Na verdade, nossa meta geral é manter em um valor mínimo absoluto o número de erros que passamos de uma atividade de engenharia de software para a próxima.

Doug: E ter certeza absoluta de manter o número de defeitos do produto liberado tão próximo de zero quanto possível.

Vinod (acenando afirmativamente): Naturalmente.

Jamie: Eu gosto da DRE como métrica e acho que podemos usá-la para o projeto inteiro, mas também quando passamos de uma atividade estrutural para a próxima. Ela nos estimulará a encontrar erros em cada etapa.

Vinod: Eu gostaria também de coletar o número de horas que gastamos em revisões.

Jamie: E o esforço total gasto em cada tarefa de engenharia de software.

Doug: Você pode calcular uma relação entre a revisão e o desenvolvimento... Pode ser interessante.

Jamie: Eu gostaria também de rastrear alguns dados de casos de uso. Como, por exemplo, o esforço necessário para desenvolver um caso de uso, o esforço necessário para criar software para implementar um caso de uso, e...

Doug (sorrindo): Eu pensava que iríamos manter tudo simples.

Vinod: Deveríamos, mas quando se entra nesse negócio de métricas, há muitas coisas interessantes para ver.

Doug: Eu concordo, mas vamos sem correria e vamos manter nosso objetivo. Limitem os dados a ser coletados em cinco ou seis itens, e estaremos prontos para começar.

25.4 INTEGRANDO MÉTRICAS DENTRO DO PROCESSO DE SOFTWARE

A maioria dos desenvolvedores de software ainda não mede nada, e, infelizmente, muitos têm pouca vontade de começar. Conforme afirmamos anteriormente neste capítulo, o problema é cultural. Tentar coletar medidas onde nenhuma medida foi coletada no passado muitas vezes causa uma resistência. "Por que precisamos fazer isso?", pergunta o gerente de projeto apressado. "Não vejo razão para isso", concorda um programador muito atarefado.

Nesta seção, vamos considerar alguns argumentos para métricas de software e apresentar uma abordagem para instituir um programa de coleta de métricas em uma organização de engenharia de software. Mas antes de começar, algumas palavras de prudência (não mais de vinte anos atrás) são sugeridas por Grady e Caswell [Gra87]:

> Algumas das coisas que descrevemos aqui parecerão muito fáceis. Na realidade, no entanto, estabelecer um programa bem-sucedido de métricas de software para a empresa inteira é um trabalho difícil. Quando dizemos que você deve esperar pelo menos três anos para conhecer as tendências organizacionais, você tem uma ideia da grandeza desse esforço.

Uma recomendação de cautela sugerida pelos autores é sempre bem-vinda, mas os benefícios das medidas são tão atraentes que compensa o trabalho duro.

25.4.1 Argumentos favoráveis a métricas de software

Por que é tão importante medir o processo de engenharia de software e o artefato que ele produz? A resposta é relativamente óbvia. Se você não medir, não haverá uma maneira real de determinar se está melhorando. E se você não estiver melhorando, está perdido.

"Nós controlamos as coisas por números em muitos aspectos da nossa vida... Esses números nos dão uma visão e ajudam a direcionar nossas ações."

Michael Mah e Larry Putnam

Solicitando e avaliando medidas de produtividade e qualidade, uma equipe de software (e seu gerente) pode estabelecer objetivos claros para melhorias do processo de software. Anteriormente neste livro, já destacamos que o software é um assunto estratégico do negócio para muitas empresas. Se o processo através do qual ele é desenvolvido pode ser melhorado, disso pode resultar um impacto direto sobre a base. Mas para estabelecer metas de melhoria, deve ser entendido o status atual do desenvolvimento de software. Consequentemente, são usadas medidas para estabelecer uma linha de base do processo a partir da qual podem ser avaliadas as melhorias.

O rigor do dia a dia do projeto de software deixa pouco tempo para pensamento estratégico. Os gerentes de projeto de software estão preocupados com problemas mais corriqueiros (mas igualmente importantes): desenvolver estimativas claras de projeto, produzir sistemas de qualidade mais alta, entrega do produto dentro do prazo. Usando as medidas para estabelecer uma linha de base de projeto, cada um desses aspectos se torna mais controlável. Já mencionamos que a linha de base serve como um referencial para estimativas. Além disso, a coleção de métricas de qualidade permite que a organização "sintonize" seu processo de software para remover as "poucas causas vitais" de defeitos que têm o maior impacto sobre o desenvolvimento do software.[9]

25.4.2 Estabelecendo uma linha de base

O que é uma linha de base de métricas e quais os benefícios que ela proporciona a um engenheiro de software?

Estabelecendo-se uma linha de base de métricas, podem ser obtidos benefícios nos níveis (técnicos) de processo, projeto e produto. No entanto, as informações que são coletadas não precisam ser fundamentalmente diferentes. As mesmas métricas podem servir a muitos mestres. As linhas de base das métricas consistem em dados coletados de projetos de desenvolvimento de software do passado e pode ser tão simples quanto a tabela apresentada na Figura 25.2 ou tão complexas quanto uma abrangente base de dados contendo dezenas de medidas de projeto e métricas delas derivadas.

Para ser um auxílio eficaz na melhoria do processo e/ou estimativas de custo e esforços, os dados da linha de base devem ter os seguintes atributos: (1) os dados devem ser razoavelmente precisos – "chutes" sobre projetos passados devem ser evitados, (2) devem ser coletados dados de tantos projetos quantos for possível, (3) as medidas devem ser consistentes (Por exemplo, uma linha de código deve ser interpretada consistentemente através de todos os projetos dos quais são coletados os dados), (4) as aplicações devem ser similares ao trabalho que deve ser estimado – faz pouco sentido usar uma linha de base para trabalho com sistemas de informação em lote para estimar uma aplicação em tempo real.

25.4.3 Coleta, cálculo e avaliação de métricas

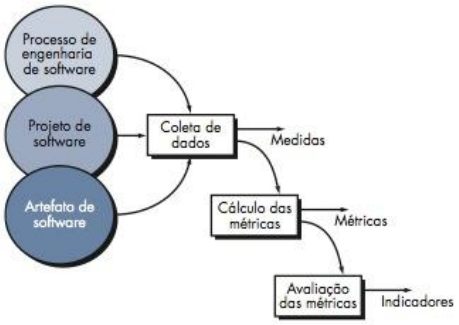
PONTO-CHAVE

Dados das linhas de base de métricas devem ser coletados de uma grande amostragem representativa de projetos de software do passado.

O processo para estabelecer uma linha de base de métricas está ilustrado na Figura 25.3. Em uma situação ideal, os dados necessários para estabelecer uma referência seriam coletados dinamicamente. Infelizmente, isso raramente acontece. Portanto, a coleta de dados requer uma investigação histórica de projetos passados para reconstruir os dados necessários. Uma vez coletadas as medidas (sem dúvida é a parte mais difícil), é possível o cálculo das métricas. Dependendo da amplitude das medidas coletadas, as métricas podem se estender sobre um amplo intervalo de métricas orientadas a aplicação (por exemplo, LOC, FP, orientada a objeto, WebApp), bem como outras métricas orientadas à qualidade e ao projeto. Finalmente, as métricas devem ser avaliadas e aplicadas durante a estimativa, trabalho técnico, controle de projeto e melhoria de processo. A avaliação de métricas focaliza as razões subjacentes para os resultados obtidos e produz um conjunto de indicadores que guiam o projeto ou o processo.

9 Essas ideias foram formalizadas em uma abordagem chamada garantia estatística de qualidade de software (*statistical software quality assurance*).

FIGURA 25.3
Processo de coleta de métricas de software



25.5 MÉTRICAS PARA PEQUENAS ORGANIZAÇÕES

AVISO

Se você estiver apenas começando a coletar dados de métricas, lembre-se de mantê-las simples. Se você começar a se afundar nos dados, o seu trabalho com métricas não terá sucesso.

A grande maioria das organizações de desenvolvimento de software têm menos de 20 profissionais de software. Não é razoável, e na maioria dos casos não é viável, esperar que essas organizações desenvolvam programas abrangentes de métricas de software. No entanto, é razoável sugerir que organizações de software de todos os tamanhos meçam e depois usem as métricas resultantes para ajudar a melhorar seu processo local de software e a qualidade e prazo dos produtos que elas produzem.

Um abordagem de bom senso para a implementação de qualquer atividade relacionada com processo de software é: mantenha simples, ajuste para satisfazer as necessidades locais, e certifique-se de que tudo agregue valor. Nos próximos parágrafos, examinaremos como essas diretrizes se relacionam com as métricas para pequenas empresas.[10]

Como devemos derivar um conjunto de métricas de software "simples"?

"Manter simples " é uma diretriz que funciona razoavelmente bem em muitas atividades. Mas como você poderá derivar um conjunto de métricas de software "simples" que ainda assim tenha valor, e como você pode ter certeza de que essas métricas simples atenderão às necessidades específicas da sua organização de software? Você pode começar focalizando não na medição, mas sim os resultados. O grupo de software é consultado para definir um objetivo único que requer melhorias. Por exemplo, "reduzir o tempo necessário para avaliar e implementar solicitações de alterações". Uma pequena organização pode escolher o seguinte conjunto de medidas que podem ser obtidas facilmente:

- Tempo (horas ou dias) decorridos desde o instante em que é feita uma solicitação até que a avaliação esteja completa, t_{fila}.
- Esforço (homem-hora) para executar a avaliação, $W_{avaliação}$.
- Tempo (horas ou dias) decorridos desde o término da avaliação até a atribuição da ordem de alteração para o pessoal, $t_{avaliação}$.
- Esforço (homem-hora) necessário para fazer a alteração, $W_{alteração}$.
- Tempo (horas ou dias) para fazer a alteração, $t_{alteração}$.
- Erros descobertos durante o trabalho para fazer a alteração, $E_{alteração}$.
- Defeitos descobertos depois que a alteração é liberada para o cliente, $D_{alteração}$.

Uma vez coletadas essas medidas para um conjunto de solicitações de alteração, é possível calcular o tempo total decorrido desde a solicitação de alteração até a implementação da alteração e a porcentagem de tempo decorrido gasto na classificação inicial, avaliação e atribuição

10 Essa discussão é igualmente relevante para equipes de software que tenham adotado um processo ágil de desenvolvimento de software (Capítulo 3).

da mudança, e implementação da alteração. De forma semelhante, pode ser determinada a porcentagem do trabalho necessário para a avaliação e implementação. Essas métricas podem ser analisadas no contexto de dados de qualidade, $E_{alteração}$ e $D_{alteração}$. A porcentagem permite visualizar onde o processo de solicitação de alteração se retarda e permite conduzir às etapas de melhoria de processo para reduzir t_{fila}, $W_{avaliação}$, $t_{avaliação}$, $W_{alteração}$, e/ou $E_{alteração}$. Além disso, a eficiência na remoção de defeitos pode ser computada como

$$\text{DRE} = \frac{E_{alteração}}{E_{alteração} + D_{alteração}}$$

A DRE pode ser comparada com o tempo decorrido e o trabalho total para determinar o impacto das atividades de garantia de qualidade sobre o tempo e trabalho necessários para fazer uma alteração.

Para pequenos grupos, o custo da coleta de medidas e cálculo das métricas varia de 3 a 8 por cento do orçamento do projeto durante a fase de aprendizado e depois cai para menos de 1 por cento do orçamento do projeto depois que os engenheiros de software e gerentes de projeto se familiarizaram com o programa de métricas [Gra99]. Esses custos podem apresentar um substancial retorno sobre o investimento se as informações derivadas dos dados das métricas levarem a melhorias significativas do processo para a organização de software.

25.6 Estabelecendo um Programa de Métricas de Software

O Software Engineering Institute desenvolveu um guia abrangente [Par96b] para estabelecer um programa de métricas de software "orientado a metas". O livro sugere as seguintes etapas:

1. Identifique as suas metas de negócio.
2. Identifique o que você quer saber ou aprender.
3. Identifique as suas submetas.
4. Identifique as entidades e atributos relacionados com as suas submetas.
5. Formalize as suas metas de medição.
6. Identifique questões quantificáveis e os indicadores relacionados que você usará para ajudá-lo a atingir as suas metas de medição.
7. Identifique os elementos de dados que você coletará para construir os indicadores que ajudam a responder às suas questões.
8. Defina as medidas a ser usadas e torne essas definições operacionais.
9. Identifique as ações que você tomará para implementar as medidas.
10. Prepare um plano para implementar as medidas.

WebRef
Um Guidebook for Goal Driven Software Measurement pode ser baixado do endereço **www.sei.cmu.edu**.

Uma discussão detalhada dessas etapas se encontra no guidebook da SEI. No entanto, temos aqui uma visualização rápida dos pontos principais.

Devido ao fato de que o software suporta as funções de negócio, diferencia sistemas ou produtos baseados em computadores, ou age como um produto por si mesmo, os objetivos definidos para os negócios podem quase sempre ser ligados a metas específicas em nível de engenharia de software. Por exemplo, considere o produto *CasaSegura*. Trabalhando como uma equipe, a engenharia de software e os gerentes de negócio desenvolvem uma lista de metas comerciais com prioridades:

PONTO-CHAVE
As métricas de software que você escolhe deverão ser motivadas pelas metas de negócio e técnicas que você deseja atingir.

1. Melhorar a satisfação dos seus clientes com seus produtos.
2. Tornar os seus produtos mais fáceis de usar.
3. Reduzir o tempo necessário para ter um novo produto no mercado.
4. Tornar mais fácil o suporte ao nosso produto.
5. Melhorar nossa lucratividade geral.

A organização de software examina cada meta de negócio e pergunta: "Quais as atividades que nós gerenciamos, executamos, ou suportamos e o que nós queremos melhorar nessas atividades?" Para responder a essas perguntas a SEI recomenda a criação de uma "lista entidade-questão" na qual todas as coisas (entidades) dentro do processo de software que são gerenciadas ou influenciadas pela organização de software são listadas. Exemplos de entidades incluem recursos de desenvolvimento, produtos acabados, código fonte, test cases, solicitações de alteração, tarefas de engenharia de software, e cronogramas. Para cada entidade listada, os profissionais do software desenvolvem uma série de questões que investigam características quantitativas da entidade (por exemplo, tamanho, custo, tempo para desenvolver). As questões originadas em consequência da criação de uma lista entidade-questão levam à criação de uma série de submetas que se relacionam diretamente com as entidades criadas e com as atividades executadas como parte do processo de software.

Considere a quarta meta: "Tornar mais fácil o suporte ao nosso produto." Para essa meta, pode ser criada a seguinte lista de questões [Par96b]:

- As solicitações de alterações do cliente contém as informações que precisamos para avaliar adequadamente a alteração e então implementá-la dentro do prazo normal?
- Qual é o acúmulo de solicitações de alteração?
- O nosso tempo de resposta para corrigir os bugs é aceitável com base nas necessidades do cliente?
- O nosso processo de controle de alterações (Capítulo 22) é seguido?
- As alterações de alta prioridade são implementadas dentro do prazo normal?

Com base nessas questões, a organização de software pode derivar a seguinte submeta: *Melhorar o desempenho do processo de gerenciamento de alterações.* As entidades e atributos do

INFORMAÇÕES

Estabelecendo um programa de métricas

O Software Productivity Center (**www.spc.ca**) sugere uma abordagem de oito passos para estabelecer um programa de métricas em uma organização de software que pode ser usada como alternativa para a abordagem SEI descrita na Seção 25.6. A abordagem se resume em:

1. Entender o processo de software existente.
   São identificadas as atividades da estrutura (Capítulo 2).
   São descritas as informações para cada atividade.
   São definidas as tarefas associadas com cada atividade.
   São descritas as funções de garantia de qualidade.
   São listados os produtos acabados que são produzidos.
2. Definir as metas a ser atingidas estabelecendo um programa de métricas.
   Exemplos:
   melhorar a precisão da estimativa,
   melhorar a qualidade do produto.
3. Identificar métricas necessárias para atingir as metas.
   São definidas as questões a ser respondidas; por exemplo, *Quantos erros encontrados em uma atividade estrutural podem estar ligados com a atividade estrutural precedente?*
   Criar medidas e métricas que ajudarão a responder a essas questões.
4. Identificar as medidas e métricas a ser coletadas e computadas.
5. Estabelecer um processo de coleta de medidas respondendo às seguintes questões:
   Qual é a fonte das medições?
   Podem ser usadas ferramentas para coletar dados?
   Quem é responsável pela coleta de dados?
   Quando os dados são coletados e registrados?
   Como os dados são armazenados?
   Que mecanismos de validação são usados para garantir que os dados estão corretos?
6. Adquirir as ferramentas apropriadas para ajudar na coleta e avaliação.
7. Estabelecer uma base de dados de métricas.
   É definida a sofisticação relativa da base de dados.
   É explorado o uso de ferramentas relacionadas (por exemplo, um repositório SCM, Capítulo 26).
   São avaliados os produtos existentes da base de dados.
8. Definir mecanismos de feedback apropriados.
   Quem solicita informações das métricas em andamento?
   Para quem as informações devem ser fornecidas?
   Qual é o formato das informações?

Uma descrição consideravelmente mais detalhada dessas oito etapas pode ser baixada do site **www.spc.ca/resources/metrics/**.

processo de software que são relevantes à submeta são identificadas, e são delineadas as metas de medição associadas com elas.

A SEI [Par96b] fornece instruções detalhadas para as etapas 6 a 10 de sua abordagem de medição motivada por meta. Essencialmente, você refina as metas de medição, transformando-as em questões que são mais refinadas tornando-se entidades e atributos que são então refinados tornando-se métricas.

25.7 Resumo

As medidas permitem aos gerentes e profissionais melhorar o processo de software; ajudam no planejamento, acompanhamento e controle dos projetos de software; e avaliam a qualidade do artefato de software que é produzido. Medidas de atributos específicos de processo, projeto e produto são usadas para computar as métricas de software. Essas métricas podem ser analisadas para fornecer indicadores que guiam a gerência e as ações técnicas.

As métricas de processo permitem que uma organização tenha uma visão estratégica, fornecendo informações sobre a eficácia de um processo de software. Métricas de projeto são táticas. Elas permitem que o gerente de projeto adapte o workflow de projeto e a abordagem técnica de uma maneira em tempo real.

Na indústria, são usadas métricas orientadas a tamanho e função. Métricas orientadas a tamanho usam a quantidade de linhas de código como um fator de normalização para outras medidas como pessoa-mês ou defeitos. O ponto de função é derivado de medidas de domínio de informações e de uma avaliação subjetiva da complexidade do problema. Além disso, podem ser usadas métricas orientadas a objeto e métricas orientadas a aplicação Web.

Métricas de qualidade de software, como as métricas de produtividade, focalizam o processo, o projeto e o produto. Desenvolvendo e analisando uma linha de base de métricas para qualidade, uma organização pode corrigir aquelas áreas do processo de software que são a causa dos defeitos de software.

Medidas resultam em mudanças culturais. Coleta de dados, cálculo das métricas, e análise das métricas são as três etapas que devem ser implementadas para começar um programa de métricas. Em geral, uma abordagem motivada por meta ajuda uma organização a focalizar as métricas corretas para seus negócios. Criando uma linha de base de métricas – uma base de dados contendo medidas de processo e produto – os engenheiros de software e seus gerentes podem ter uma visão melhor do trabalho que eles executam e do produto que eles produzem.

Problemas e Pontos a Ponderar

25.1. Descreva a diferença entre métricas de processo e de projeto com suas próprias palavras.

25.2. Por que algumas métricas devem ser mantidas "privadas"? Forneça exemplos de três métricas que deverão ser privadas. Forneça exemplos de três métricas que deverão ser públicas.

25.3. O que é uma medida indireta, e por que essas medidas são comuns no trabalho com métricas de software?

25.4. Grady sugere uma etiqueta para métricas de software. Você pode acrescentar mais três regras àquelas mencionadas na Seção 25.1.1?

25.5. A Equipe A encontrou 342 erros durante um processo de engenharia de software antes do lançamento. A Equipe B encontrou 184 erros. Que medidas adicionais terão de ser feitas para os projetos A e B para determinar qual das equipes eliminou erros com mais eficiência? Que métricas você poderia propor para ajudar nessa determinação? Que dados históricos podem ser úteis?

25.6. Apresente um argumento contra a adoção de número de linhas de código como medida de produtividade de software. O seu argumento poderá se manter quando forem consideradas dezenas ou centenas de projetos?

25.7. Calcule o valor de pontos de função para um projeto com as seguintes características no domínio de informações:

Número de entradas de usuário: 32
Número de saídas de usuário: 60
Número de consultas de usuário: 24
Número de arquivos: 8
Número interfaces externas: 2

Suponha que todos os valores de ajuste de complexidade são médios. Use o algoritmo do Capítulo 23.

25.8. Usando a tabela apresentada na Seção 25.2.3, apresente um argumento contra o uso de linguagem assembler com base na funcionalidade fornecida pelas instruções de código. Novamente referindo-se à tabela, discuta porque C++ seria uma alternativa melhor do que C.

25.9. O software usado para controlar uma fotocopiadora requer 32.000 linhas de C e 4.200 linhas de Smalltalk. Estime o número de pontos de função do software que está na copiadora.

25.10. Uma equipe de engenharia Web criou um WebApp e-commerce contendo 145 páginas individuais. Dessas páginas, 65 são dinâmicas, ou seja, elas são geradas internamente com base em entradas do usuário. Qual é o índice de personalização dessa aplicação?

25.11. Um WebApp e seu ambiente de suporte não foram totalmente protegidos contra ataques. Os engenheiros Web estimam que a probabilidade de repelir um ataque é de apenas 30%. O sistema não contém informações sensíveis ou comprometedoras, portanto, a probabilidade de ataque é 25%. Qual é a integridade do WebApp?

25.12. Na conclusão de um projeto, foi determinado que foram encontrados 30 erros durante a atividade de modelagem e foram encontrados 12 erros durante a atividade de construção que estavam ligados a erros que não foram descobertos na atividade de modelagem. Qual é a DRE para a atividade de modelagem?

25.13. Uma equipe de software fornece um incremento de software para usuários finais. Os usuários descobrem 8 defeitos durante o primeiro mês de uso. Antes da entrega, a equipe de software encontrou 242 erros durante as revisões técnicas formais e em todas as tarefas de teste. Qual é a DRE geral do projeto após um mês de uso?

Leituras e Fontes de Informação Complementares

A melhoria do processo de software (software process improvement – SPI) tem recebido uma atenção significativa durante as últimas duas décadas. Como as medições e as métricas de software são fundamentais para uma melhora bem-sucedida do processo de software, muitos livros sobre SPI também discutem métricas. Rico (*ROI of Software Process Improvement,* J. Ross Publishing, 2004) fornece uma discussão aprofundada da SPI e as métricas que podem ajudar uma organização a atingi-la. Ebert e seus colegas (*Best Practices in Software Measurement,* Springer, 2004) trata do uso das medições dentro do contexto das normas ISO e CMMI. Kan (*Metrics and Models in Software Quality Engineering,* 2d ed., Addison-Wesley, 2002) apresenta uma coleção de métricas relevantes.

Ebert e Dumke (*Software Measurement,* Springer, 2007) proporcionam um bom tratamento sobre medidas e métricas de como elas devem ser aplicadas para projetos TI. McGarry e seus colegas (*Practical Software Measurement,* Addison-Wesley, 2001) apresenta uma profunda consultoria para avaliação de processo de software. Uma boa coleção de artigos foi editada por Haug e seus colegas (*Software Process Improvement: Metrics, Measurement, and Process Modeling,* Springer-Verlag, 2001). Florac e Carlton (*Measuring the Software Process,* Addison-Wesley, 1999) e Fenton e Pfleeger (*Software Metrics: A Rigorous and Practical Approach,* Revised, Brooks/Cole Publishers, 1998) discutem como as métricas de software podem ser usadas para proporcionar os indicadores necessários para melhorar o processo de software.

Laird e Brennan (*Software Measurement and Estimation*, Wiley-IEEE Computer Society Press, 2006) e Goodman (*Software Metrics: Best Practices for Successful IT Management*, Rothstein Associates, Inc., 2004) discutem o uso das métricas de software para gerenciamento de projeto e estimativas. Putnam e Myers (*Five Core Metrics*, Dorset House, 2003) usam uma base de dados de mais de 6.000 projetos de software para demonstrar como cinco métricas fundamentais – tempo, trabalho, tamanho, confiabilidade e produtividade de processo – podem ser usadas para controlar projetos de software. Maxwell (*Applied Statistics for Software Managers*, Prentice-Hall, 2003) apresenta técnicas para analisar dados de projeto de software. Munson (*Software Engineering Measurement*, Auerbach, 2003) discute um amplo conjunto de aspectos de medidas em engenharia de software. Jones (*Software Assessments, Benchmarks and Best Practices*, Addison-Wesley, 2000) descreve fatores quantitativos e qualitativos que ajudam uma organização a avaliar seu processo e práticas de software.

A medida de pontos de função tornou-se uma técnica amplamente utilizada em muitas áreas da engenharia de software. Parthasarathy (*Practical Software Estimation: Function Point Methods for Insourced and Outsourced Projects*, Addison-Wesley, 2007) fornece um guia abrangente. Garmus e Herron (*Function Point Analysis: Measurement Practices for Successful Software Projects*, Addison-Wesley, 2000) discutem métricas de processo com ênfase na análise de pontos de função.

Relativamente pouco se tem publicado sobre métricas em trabalhos de engenharia Web. Kaushik *(Web Analytics: An Hour a Day*, Sybex, 2007), Stern (*Web Metrics: Proven Methods for Measuring Web Site Success*, Wiley, 2002), Inan e Kean (*Measuring the Success of Your Website*, Longman, 2002), e Nobles e Grady (*Web Site Analysis and Reporting*, Premier Press, 2001) tratam das métricas Web a partir de uma perspectiva de negócios e marketing.

A mais recente pesquisa na área de métricas está resumida pelo IEEE (*Symposium on Software Metrics*, publicado anualmente). Uma grande variedade de fontes de informações sobre métricas de processo e projeto está disponível na Internet. Uma lista atualizada das referências da World Wide Web relevantes a métricas de processo e projeto pode ser encontrada no site **www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm**.