

29

MANUTENÇÃO E REENGENHARIA

CONCEITOS-CHAVE

análise de inventário.....	669
engenharia direta.....	675
engenharia reversa.....	670
dados.....	674
interface de usuário.....	673
processamento.....	672
manutenção de software.....	663
manutenibilidade.....	664

Independentemente do domínio de aplicação, tamanho ou complexidade, o software continuará a evoluir com o tempo. As mudanças dirigem esse processo. No âmbito do software, ocorrem alterações quando são corrigidos erros, quando há adaptação a um novo ambiente, quando o cliente solicita novas características ou funções e quando a aplicação passa por um processo de reengenharia para proporcionar benefício em um contexto moderno. Durante os últimos 30 anos, Manny Lehman [por exemplo, Leh97a] e seus colegas executaram análises detalhadas do software industrial e sistemas para desenvolver a *teoria unificada para evolução do software* (*unified theory for software evolution*). Os detalhes desse trabalho estão além do escopo deste livro, mas vale a pena conhecer as leis que daí se originaram [Leh97b]:

A Lei da Mudança Contínua (1974): Softwares que foram implementados em um contexto de computação do mundo real e, portanto, irão evoluir com o tempo (chamados de *sistemas tipo E*) devem ser adaptados continuamente ou se tornarão cada vez menos satisfatórios.

A Lei da Complexidade Crescente (1974): À medida que um sistema tipo E evolui, sua complexidade aumenta, a menos que seja feito um trabalho para mantê-la ou reduzi-la.

PANORAMA

O que é? Considere qualquer produto de tecnologia que tenha lhe servido bem. Você o utiliza regularmente, mas ele está ficando obsoleto. Apresenta problemas com muita frequência, leva mais tempo para ser reparado do que você aceitaria e já não representa a tecnologia mais atualizada. O que fazer? Por um tempo, você tenta consertá-lo, remendá-lo ou até ampliar sua funcionalidade. Isso se chama manutenção. Mas a manutenção se torna cada vez mais difícil à medida que os anos passam. Chega então um momento em que precisa reformá-lo. Você criará um produto com mais funcionalidades, melhor desempenho e confiabilidade e de manutenção mais fácil. Isso é o que chamamos de reengenharia.

Quem realiza? No nível organizacional, a manutenção é executada por pessoal de suporte que faz parte da organização de engenharia de software. A reengenharia é executada por especialistas de negócio (muitas vezes empresas de consultoria), e no nível de software, a reengenharia é executada por engenheiros de software.

Por que é importante? Vivemos em um mundo que muda rapidamente. As demandas por funções de negócio e a tecnologia de informação que as suporta estão mudando em um ritmo que impõe enorme pressão competitiva em todas as organizações comerciais. É por essa razão que o software deve ser mantido continuamente. No momento apropriado, deve passar pela reengenharia para manter o ritmo.

Quais são as etapas envolvidas? A manutenção corrige defeitos, adapta o software para atender a um ambiente em mutação e melhora a funcionalidade para atender às necessidades dos clientes. Em um nível estratégico, a reengenharia de processos de negócio (*business process reengineering – BPR*) define objetivos comerciais, identifica e avalia processos comerciais existentes e cria processos de negócio revisados que melhor atendem aos objetivos atuais. A reengenharia de software abrange análise de inventário, reestruturação de documentos, engenharia reversa, reestruturação de programas e dados e engenharia direta. O objetivo dessas atividades é criar versões dos programas que apresentam uma qualidade mais alta e melhorar a manutenibilidade.

Qual é o artefato? É produzida uma variedade de produtos de manutenção e reengenharia (por exemplo, casos de uso, modelos de análise e projeto, procedimentos de teste). O resultado final é um software atualizado.

Como garantir que o trabalho foi realizado corretamente? Use as mesmas práticas de SQA aplicadas em todos os processos de engenharia de software — revisões técnicas avaliam os modelos de análise e projeto; revisões especializadas consideram a aplicabilidade e compatibilidade de negócio; e são aplicados testes para descobrir erros em conteúdo, funcionalidade e interoperabilidade.

reengenharia de processos de negócio (BPR)	665
reestruturação dos documentos	669
reestruturação ... código	674
dados	674
reengenharia de software	667
suportabilidade ...	664



Como os sistemas legados evoluem com o tempo?

A Lei da Autorregulação (1974): O processo de evolução do sistema tipo E é autorregulado com distribuição do produto e medidas de processo próximo do normal.

A Lei da Conservação da Estabilidade Organizacional (1980): A taxa de atividade efetiva média em um sistema tipo E em evolução é invariante durante o tempo de vida do produto.

A Lei da Conservação da Familiaridade (1980): Conforme um sistema tipo E evolui, tudo o que está associado a ele, por exemplo, desenvolvedores, pessoal de vendas, usuários, deve manter o domínio de seu conteúdo e comportamento para uma evolução satisfatória. Um crescimento excessivo diminui esse domínio. Portanto, o crescimento incremental médio permanece invariante à medida que o sistema evolui.

A Lei do Crescimento Contínuo (1980): O conteúdo funcional dos sistemas tipo E deve ser continuamente ampliado durante toda a sua existência, para manter a satisfação do usuário.

A Lei da Qualidade em Declínio (1996): A qualidade dos sistemas tipo E irá parecer que está diminuindo a menos que sejam rigorosamente mantidos e adaptados às mudanças do ambiente operacional.

A Lei do Sistema de Realimentação (1996): Processos tipo E em evolução constituem sistemas de realimentação multinível, multilaço, multiagente e devem ser tratados como tais para que se obtenha uma melhora significativa em qualquer base razoável.

As leis que Lehman e seus colegas definiram são parte da realidade do engenheiro de software. Neste capítulo, discutiremos o desafio das atividades de manutenção e reengenharia de software necessárias para ampliar a vida efetiva dos sistemas legados.

29.1 MANUTENÇÃO DE SOFTWARE

A manutenção começa quase imediatamente. O software é liberado para os usuários finais, e em alguns dias, os relatos de bugs começam a chegar à organização de engenharia de software. Em algumas semanas, uma classe de usuários indica que o software deve ser mudado para se adaptar às necessidades especiais de seus ambientes. E em alguns meses, outro grupo corporativo, ainda não interessado no software quando foi lançado, agora reconhece que pode lhes trazer alguns benefícios. Eles precisarão de algumas melhorias para fazer o software funcionar em seu mundo.

O desafio da manutenção do software começou. Enfrentamos uma fila crescente de correções de bugs, solicitações de adaptação e melhorias que devem ser planejadas, programadas e, por fim, executadas. Logo, a fila já cresceu muito e o trabalho ameaça devorar os recursos disponíveis. Com o passar do tempo, sua organização descobre que está gastando mais tempo e dinheiro com a manutenção dos programas do que criando novas aplicações. De fato, não é raro uma organização de software despende de 60% a 70% de todos os recursos com manutenção de software.

Você talvez pergunte por que é necessário tanta manutenção e por que tanto esforço deve ser despendido. Osborne e Chikofsky [Osb90] fornecem uma resposta parcial:

Muitos softwares dos quais dependemos hoje têm em média de 10 a 15 anos. Mesmo quando esses programas foram criados, usando as melhores técnicas de projeto e codificação conhecidas na época (e muitos não foram), o tamanho do programa e o espaço de armazenamento eram as preocupações principais. Eles então migraram para novas plataformas, foram ajustados para mudanças nas máquinas e na tecnologia dos sistemas operacionais e aperfeiçoados para atender a novas necessidades dos usuários – tudo isso sem grande atenção na arquitetura geral. O resultado são estruturas mal projetadas, mal codificadas, de lógica pobre e mal documentadas em relação aos sistemas de software, para os quais somos chamados a fim de mantê-los rodando...

Outra razão para o problema de manutenção do software é a mobilidade dos profissionais. É bem provável que a equipe (ou profissional) responsável pelo trabalho original não esteja mais por perto. Pior ainda, outras gerações de profissionais de software podem ter modificado o sistema e já se foram. E hoje, pode não ter restado ninguém que tenha conhecimento direto do sistema legado.

"Manutenibilidade e clareza de programa são conceitos paralelos: quanto mais difícil for entender um programa, mais difícil será mantê-lo.

Gerald Berns

Conforme notamos no Capítulo 22, a natureza ubíqua das alterações permeia todo o trabalho de software. Mudanças são inevitáveis quando sistemas baseados em computadores são criados; portanto, você deve desenvolver mecanismos para avaliar, controlar e fazer modificações.

Em todo este livro, destacamos a importância de entender o problema (análise) e desenvolver uma solução bem estruturada (projeto). De fato, a Parte 2 deste livro é dedicada aos mecanismos dessas ações de engenharia de software e a Parte 3 concentra-se nas técnicas para garantir que você as tenha feito corretamente. Tanto a análise quanto o projeto levam a uma importante característica do software que chamamos de manutenibilidade, que, essencialmente, é uma indicação qualitativa¹ da facilidade com que o software pode ser corrigido, adaptado ou melhorado. Grande parte das funções da engenharia de software é criar sistemas que apresentem alta manutenibilidade.

Mas o que é manutenibilidade? Software "manutenível" apresenta uma modularidade eficaz (Capítulo 8). Utiliza padrões de projeto (Capítulo 12) que permitem entendê-lo facilmente. Foi construído usando padrões e convenções de codificação bem definidos, levando a um código-fonte autodocumentado e inteligível. Passou por uma variedade de técnicas de garantia de qualidade (Parte 3 deste livro) que descobriu potenciais problemas de manutenção antes que o software fosse lançado. Foi criado por engenheiros de software que reconhecem que não estarão por perto quando as alterações tiverem de ser feitas. Portanto, o projeto e a implementação do software deve "ajudar" a pessoa que for fazer a alteração.

29.2 SUPORTABILIDADE DO SOFTWARE

Para suportar efetivamente software de classe industrial, a organização (ou seus projetistas) deve ser capaz de fazer correções, adaptações e melhorias inerentes à atividade de manutenção. Além disso, a organização deve executar outras atividades importantes que incluem suporte operacional continuado, suporte ao usuário final e atividades de reengenharia durante toda a vida útil do software. Uma definição razoável da *suportabilidade do software* é

... a capacidade de suportar um sistema de software durante toda a vida útil do produto. Isso implica satisfazer quaisquer necessidades ou requisitos, mas também a provisão do equipamento, infraestrutura de suporte, software adicional, serviços de conveniências, mão de obra ou qualquer outro recurso necessário para manter o software operacional e capaz de satisfazer suas funções [SSO08].

Essencialmente, a suportabilidade é um dos muitos fatores de qualidade que devem ser considerados durante a análise e projeto na gestão da qualidade. Ela deve ser tratada como parte do modelo de requisitos (ou especificações) e considerada conforme o projeto evolui e a construção inicia.

Por exemplo, a necessidade de software "antidefeito" em nível de componente e código foi discutida anteriormente neste livro. O software deve conter facilidades para ajudar o pessoal de suporte quando encontrado um defeito no ambiente operacional (e não se iluda, eles *serão* encontrados). Além disso, o pessoal de suporte deve ter acesso a uma base de dados que contém os registros de todos os defeitos já detectados – suas características, causa e solução. Isso permitirá que o pessoal de suporte examine defeitos "similares" e possibilitará diagnóstico e correção mais rápidos.

Embora os erros encontrados em uma aplicação sejam um problema crítico de suporte, a suportabilidade também exige que sejam providenciados recursos para resolver os problemas diários dos usuários finais. A função do pessoal de suporte é responder às dúvidas dos usuários sobre instalação, operação e uso da aplicação.

WebRef

Uma grande quantidade de documentos sobre suportabilidade de software pode ser encontrada no site www.software-supportability.org/Downloads.html.

¹ Há algumas medidas quantitativas que fornecem uma indicação indireta da manutenibilidade (por exemplo, [Sch99], [SEI02]).

29.3 REENGENHARIA

Em um artigo seminal escrito para a *Harvard Business Review*, Michael Hammer [Ham90] lançou os fundamentos para uma revolução no modo de pensar dos gerentes sobre processos de negócio e computação:

Está na hora de parar de seguir as trilhas das vacas. Em vez de acrescentar processos ultrapassados em hardware e software, devemos rejeitá-los e começar de novo. Devemos fazer uma "reengenharia" em nossos negócios: usar o poder da moderna tecnologia de informação para redesenhar radicalmente nossos processos de negócio para obter melhoras significativas no desempenho.

Toda empresa opera de acordo com muitas regras não articuladas... A reengenharia luta para romper com as velhas regras sobre como organizamos e conduzimos nossos negócios.

"Encorar o amanhã pensando em usar métodos de ontem é ver a vida estagnada."

James Bell

Como ocorre em todas as revoluções, a agitação provocada por Hammer resultou em mudanças tanto positivas quanto negativas. Durante a década de 1990, algumas empresas fizeram um legítimo esforço para a reengenharia, e os resultados levaram a uma melhor competitividade. Outras ficaram apenas no downsizing e na terceirização (em vez da reengenharia) para melhorar sua sobrevivência. Isso resultou no aparecimento de organizações "medianas" com pouco potencial para crescimento futuro [DeM95a].

No fim da primeira década do século 21, a moda associada à reengenharia desapareceu, mas o processo continua por si só em grandes e pequenas empresas. A ligação entre reengenharia nos negócios e reengenharia no software baseia-se em uma "visão de sistema".

O software é muitas vezes a realização das regras de negócio discutidas por Hammer. Hoje, grandes empresas possuem dezenas de milhares de programas de computador que suportam as "velhas regras de negócio". Enquanto os gerentes trabalham para modificar as regras para obter maior eficiência e competitividade, o software deve acompanhar. Em alguns casos, isso significa a criação de grandes sistemas baseados em computadores.² Mas em alguns outros, significa a modificação ou reforma de aplicações existentes.

Nas seções a seguir, examinaremos a reengenharia de um modo descendente, começando com uma rápida visão geral da reengenharia de processos de negócio passando para uma discussão mais detalhada das atividades técnicas que ocorrem quando o software sofre uma reengenharia.

29.4 REENGENHARIA DE PROCESSO DE NEGÓCIO

PONTO-CHAVE

A reengenharia de processos de negócio (BPR) muitas vezes resulta em nova funcionalidade de software, enquanto a reengenharia de software trabalha para substituir funcionalidade existente por um software melhor, mais fácil de manter.

A reengenharia de processos de negócio (*business process reengineering* — BPR) se estende muito além do escopo das tecnologias de informação e da engenharia de software. Entre as muitas definições (muitas um tanto abstratas) sugeridas para a BPR, destaca-se a publicada na revista *Fortune Magazine* [Ste93]: "a busca para, e a implementação de, mudanças radicais nos processos de negócio para obter resultados excelentes". Mas como é feita a busca e como é obtida a implementação? Mais importante ainda, como podemos assegurar que a "mudança radical" sugerida levará de fato a "resultados inovadores" e não ao caos organizacional?

29.4.1 Processos de negócio

Um processo de negócio é "um conjunto de tarefas logicamente relacionadas executadas para obter um resultado de negócio definido" [Dav90]. No processo de negócio, pessoas, equipamentos, recursos materiais e procedimentos são combinados para produzir um resultado específico. Exemplos de processos de negócio incluem projeto de um novo produto, compra de serviços e suprimentos, contratação de funcionários e pagamento dos fornecedores. Cada um desses demanda uma série de tarefas e cada um utiliza diversos recursos na empresa.

² A explosão de aplicações e sistemas baseados na Web é uma indicação dessa tendência.



Como engenheiro de software, seu trabalho ocorre na base dessa hierarquia. Porém, você deve se certificar de que alguém tenha dado séria atenção aos níveis acima. Se isso não foi feito, seu trabalho está em risco.

Todo processo de negócio tem um cliente definido — uma pessoa ou grupo que recebe o resultado (por exemplo, uma ideia, um relatório, um projeto, um serviço, um produto). Além disso, os processos de negócio cruzam as fronteiras da organização. Requerem que diferentes grupos organizacionais participem das “tarefas logicamente relacionadas” que definem o processo.

Cada sistema é na realidade uma hierarquia de subsistemas. Um negócio não é uma exceção; de forma geral é segmentado da seguinte maneira:

O negócio → sistemas de negócio → processos de negócio → subprocessos de negócio

Cada sistema de negócio (também chamado de *funções de negócio*) é composto de um ou mais processos, e cada processo de negócio é definido por uma série de subprocessos.

A BPR pode ser aplicada a qualquer nível da hierarquia, à medida que o escopo da BPR se amplia (conforme subimos na hierarquia), os riscos associados a ela crescem significativamente. Por essa razão, muitos esforços de BPR concentram-se em processos individuais e subprocessos.

29.4.2 Um modelo de BPR

Assim como a maioria das atividades de engenharia, a reengenharia dos processos de negócio é iterativa. As metas de negócio e os processos para atingi-las devem ser adaptados a um ambiente de negócio em mutação. Por essa razão, não há um início e fim para a BPR — é um processo evolucionário. Um modelo para reengenharia de processo de negócio é apresentado na Figura 29.1. O modelo define seis atividades:

Definição de negócio. As metas dos negócios são identificadas em um contexto dos quatro motivadores principais: redução de custo, redução do tempo, melhoria da qualidade e desenvolvimento pessoal. As metas podem ser definidas no nível do negócio ou para um componente específico do negócio.

Identificação do processo. São identificados os processos críticos para atingir as metas estabelecidas na definição do negócio. Eles podem então ser classificados por importância, por necessidade de mudança ou de qualquer outra forma apropriada para a atividade de reengenharia.

Avaliação de processo. O processo existente é amplamente analisado e medido. Identificam-se as tarefas do processo, registram-se os custos e o tempo consumido pelas tarefas do processo, e são isolados os problemas de qualidade/desempenho.

FIGURA 29.1
Um modelo de BPR



"Tão logo identificamos algo conhecido em uma coisa nova, nos tranquilizamos."

F. W. Nietzsche

Especificação e projeto do processo. Com base nas informações obtidas durante as três primeiras atividades da BPR, são preparados casos de uso (Capítulos 5 e 6) para cada processo que deve ser reprojetoado. No contexto da BPR, os casos de uso identificam um cenário que fornece algum resultado a um cliente. Com o caso de uso como especificação do processo, um novo conjunto de tarefas é projetado para o processo.

Protótipo. Um processo de negócio reprojetoado deve passar por um protótipo antes de ser totalmente integrado nos negócios. Essa atividade "testa" o processo de modo que possam ser feitos os refinamentos.

Refinamento e instanciação. Com base nas informações obtidas do protótipo, o processo de negócio é refinado e instanciado em um sistema de negócio.

As atividades de BPR são às vezes utilizadas em conjunto com ferramentas de análise de fluxo de trabalho. A finalidade das ferramentas é criar um modelo do fluxo de trabalho existente, em um esforço para melhor analisar os processos.

29.5 REENGENHARIA DE SOFTWARE

O cenário é muito comum. Um aplicativo atendeu às necessidades de negócio de uma empresa por 10 ou 15 anos. Durante esse tempo, ele foi corrigido, adaptado e aperfeiçoado muitas vezes. Profissionais realizaram esse trabalho com as melhores intenções, mas as boas práticas de engenharia de software foram sempre deixadas de lado (devido à pressão por outros aspectos). Agora o aplicativo está instável. Ainda funciona, mas sempre que se tenta fazer uma alteração, ocorrem efeitos colaterais sérios e inesperados. No entanto, o aplicativo deve continuar evoluindo. O que fazer?

Software que não pode ser mantido não é novidade. Na verdade, a ênfase cada vez mais maior sobre a reengenharia de software foi motivada pelos problemas de manutenção criados por mais de quatro décadas.

FERRAMENTAS DO SOFTWARE



Reengenharia de processos de negócio (business process reengineering-BPR)

Objetivo: o objetivo das ferramentas de BPR é suportar a análise e avaliação de processos de negócio existentes e a especificação de projetos de novos.

Mecânica: a mecânica das ferramentas varia. Em geral, as ferramentas de BPR permitem que um analista modele os processos de negócio existentes em um esforço para avaliar as ineficiências do fluxo de trabalho ou problemas funcionais. Uma vez identificados os problemas, existem ferramentas que permitem a análise do protótipo e/ou simulação de processos de negócio revisados.

Ferramentas representativas:³

Extend, desenvolvida pela ImagineThat, Inc. (www.imagine-that-inc.com), é uma ferramenta de simulação para modelar processos existentes e explorar novos processos. A ferramenta Extend proporciona um recurso amplo do tipo "o

que-se" que permite a um analista de negócio explorar diferentes cenários de processo.

e-Work, desenvolvida pela Metastorm (www.metastorm.com), proporciona suporte de gerenciamento de processos de negócio tanto para processos manuais quanto automáticos.

IceTools, desenvolvida pela Blue Ice (www.blueice.com), é uma coleção de modelos de BPR para o Microsoft Office e Microsoft Project.

SpeedDev, desenvolvida pela SpeedDev Inc. (www.speeddev.com), é uma das muitas ferramentas que permitem a uma organização modelar o fluxo de trabalho de processo (neste caso, fluxo de trabalho de TI).

Workflow tool suite, desenvolvida pela MetaSoftware (www.metasoftware.com), incorpora um conjunto de ferramentas para modelagem de fluxo de trabalho, simulação e cronogramas.

Uma boa lista de links de ferramentas de BPR pode ser encontrada no site www.opfro.org/index.html?Components/Producers/Tools/BusinessProcessReengineeringTools.html~Contents.

³ As ferramentas aqui apresentadas não significam um aval, mas sim uma amostra dessa categoria. Na maioria dos casos, seus nomes são marcas registradas pelos respectivos desenvolvedores.

29.5.1 Um modelo de processo de reengenharia de software

Reengenharia toma tempo, tem um custo significativo em dinheiro e absorve recursos que poderiam de outra forma ser usados em necessidades mais imediatas. Por todas essas razões, a reengenharia não é realizada em alguns meses ou mesmo anos. A reengenharia dos sistemas de informação é uma atividade que absorverá recursos da tecnologia de informação por muitos anos. Todas as organizações precisam de uma estratégia pragmática para reengenharia de software.

Uma estratégia prática faz parte de um modelo de processo de reengenharia. Discutiremos o modelo mais tarde nesta seção, mas primeiro, vejamos alguns princípios básicos.

A reengenharia é um trabalho de reforma. Para melhor entendê-la, considere uma atividade análoga: a reforma de uma casa. Imagine a seguinte situação. Você acaba de comprar uma casa em outro Estado, mas nunca viu o imóvel. Comprou-o por um preço extremamente baixo, tendo sido alertado de que a casa poderia necessitar de reforma completa. Como você faria?

- Antes de iniciar a reforma, seria razoável inspecionar a casa. Para determinar se precisa de reforma, você (ou um profissional de construção) criaria uma lista de critérios para que a inspeção fosse sistemática.
- Antes de começar a reconstrução, verifique como está a estrutura. Se a casa estiver com a estrutura em bom estado, pode ser possível “remodelar” sem reformar (a um custo bem mais baixo e em menos tempo).
- Antes de começar a reforma, procure entender como a casa original foi construída. Dê uma olhada atrás das paredes. Verifique a fiação elétrica, a tubulação hidráulica e as partes internas da estrutura. Mesmo que você resolva descartar tudo, as informações obtidas terão utilidade quando iniciar a reconstrução.
- Na reforma, use somente os materiais mais modernos e mais duráveis. Isso pode custar um pouco mais agora, mas ajudará a evitar uma manutenção cara e demorada mais tarde.
- Se você decide reformar, seja disciplinado. Use práticas que resultarão na mais alta qualidade — hoje e no futuro.

Embora esses princípios concentrem-se na reforma de uma casa, eles se aplicam igualmente bem à reengenharia de sistemas e aplicações baseados em computador.

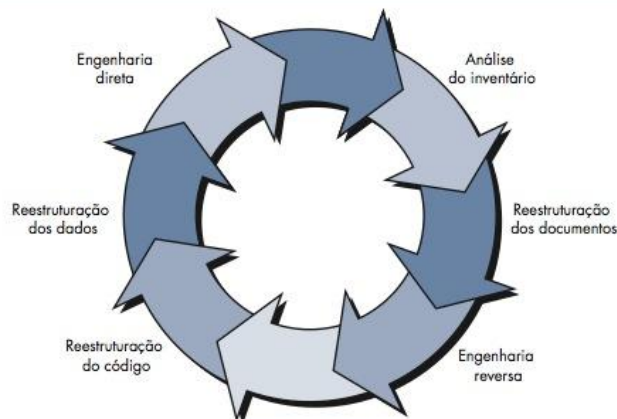
Para implementarmos esses princípios, podemos usar um modelo de processo de reengenharia de software, apresentado na Figura 29.2. Em alguns casos, essas atividades ocorrem em

WebRef

Uma excelente fonte de informações sobre reengenharia de software pode ser encontrada no site reengineer.org.

FIGURA 29.2

Um modelo de processo de reengenharia de software



seqüência linear, mas nem sempre é o caso. Por exemplo, pode acontecer de a engenharia reversa (entendimento do funcionamento interno de um programa) ter de ocorrer antes do início da reestruturação dos documentos.



Se o tempo e os recursos estiverem escassos, você pode aplicar o princípio de Pareto ao software que vai passar por reengenharia. Aplique o processo de reengenharia aos 20% do software responsáveis por 80% dos problemas.



Crie apenas a documentação necessária para melhor entender o software, nem uma página a mais.

29.5.2 Atividades de reengenharia de software

O paradigma da reengenharia mostrado na Figura 29.2 é um modelo cíclico. Isso significa que cada uma das atividades apresentadas como parte do paradigma pode ser revisitada. Para qualquer ciclo em particular, o processo pode terminar após qualquer uma dessas atividades.

Análise de inventário. Toda organização de software deve ter um inventário de todos os aplicativos. O inventário pode ser nada mais do que uma planilha com informações detalhadas (por exemplo, tamanho, idade, criticidade nos negócios) para cada aplicativo ativo. Ordenando essas informações de acordo com a criticidade de negócio, longevidade, manutenibilidade atual e suportabilidade, e outros critérios localmente importantes, surgem os candidatos à reengenharia. Recursos podem então ser alocados aos aplicativos candidatos ao trabalho de reengenharia.

É importante observar que o inventário deverá ser revisto regularmente. O status dos aplicativos (por exemplo, criticidade de negócio) pode mudar em função do tempo, e, como resultado, as prioridades para a reengenharia também podem mudar.

Reestruturação dos documentos. Documentação pobre é a marca registrada de muitos sistemas legados. O que se pode fazer quanto a isso? Quais são as suas opções?

1. *Criar documentação consome muito tempo.* Se o sistema funciona, você pode optar por conviver com aquilo que tem. Em alguns casos, essa é a atitude correta. Não é possível recriar documentação para centenas de programas de computador. Se um programa for relativamente estático, está chegando ao fim de sua vida útil, e provavelmente não terá uma modificação significativa, deixe-o como está!
2. *A documentação tem de ser atualizada, mas sua organização apresenta recursos limitados.* Você empregará uma abordagem "documentar quando usar". Pode não ser necessário redocumentar totalmente o aplicativo. Em vez disso, aquelas partes do sistema que estão passando por alteração são completamente documentadas. Com o tempo, você terá uma coleção de documentos úteis e importantes.
3. *O sistema é crítico para os negócios e deve ser totalmente documentado.* Mesmo neste caso, uma abordagem inteligente é limitar a documentação a um mínimo essencial.

Cada uma dessas opções é viável. Sua organização de software deve escolher a mais apropriada para cada caso.

Engenharia reversa. O termo *engenharia reversa* tem suas origens no mundo do hardware. Uma empresa desmonta um produto de hardware competitivo na tentativa de conhecer os "segredos" de projeto e fabricação do concorrente. Os segredos poderiam ser facilmente entendidos se fosse possível obter as especificações de projeto e fabricação do concorrente. Mas esses documentos são de propriedade privada e não estão disponíveis para a empresa que está fazendo a engenharia reversa. Essencialmente, uma engenharia reversa bem-sucedida resulta em uma ou mais especificações de projeto e fabricação para um produto pelo exame de amostras atuais do produto.

A engenharia reversa para o software é bem similar. Em muitos casos, no entanto, o programa a ser submetido a uma engenharia reversa não é o programa de um concorrente. Em vez disso, é o próprio trabalho da empresa (em geral, feito há muitos anos). Os "segredos" a ser entendidos são obscuros porque nenhuma especificação jamais foi desenvolvida. Portanto, a engenharia reversa para software é o processo para analisar um programa na tentativa de criar uma representação do programa em um nível mais alto de abstração do que o código-fonte.

WebRef

Um conjunto de recursos para a comunidade de reengenharia pode ser obtido no site www.comp.lancs.ac.uk/projects/RenaissanceWeb/.

A engenharia reversa é um processo de *recuperação do projeto*. As ferramentas de engenharia reversa extraem informações do projeto de dados, da arquitetura e procedural com base em um programa existente.

Reestruturação do código. O tipo mais comum de reengenharia (atualmente, o uso do termo reengenharia é questionável neste caso) é a *reestruturação de código*.⁴ Alguns sistemas legados têm uma arquitetura de programa razoavelmente sólida, mas os módulos individuais foram codificados de um modo que se torna difícil de entendê-los, testá-los e mantê-los. Nesses casos, o código dentro dos módulos suspeitos pode ser reestruturado.

Para executar essa atividade, o código-fonte é analisado por meio de uma ferramenta de reestruturação. As violações das construções de programação estruturada são registradas, e o código é então reestruturado (isso pode ser feito automaticamente) ou mesmo reescrito em uma linguagem de programação mais moderna. O código reestruturado resultante é revisado e testado para garantir que não tenham sido introduzidas anomalias. A documentação interna do código é atualizada.

Reestruturação dos dados. Um programa com uma arquitetura de dados fraca será difícil de adaptar e melhorar. Na verdade, para muitas aplicações, a arquitetura das informações tem mais a ver com a viabilidade de longo prazo de um programa do que o próprio código-fonte.

Diferentemente da reestruturação de código, que ocorre em um nível relativamente baixo de abstração, a reestruturação de dados é uma atividade de reengenharia em escala completa. Em muitos casos, a reestruturação dos dados começa com uma atividade de engenharia reversa. A arquitetura de dados atual é dissecada, e os modelos de dados necessários são definidos (Capítulos 6 e 9). Identificam-se os objetos de dados e atributos, e as estruturas de dados existentes são revisadas quanto à qualidade.

Quando a estrutura de dados é fraca (por exemplo, estão implementados em arquivos de texto, quando uma abordagem relacional simplificaria muito o processamento), os dados passam por uma reengenharia.

Devido à arquitetura de dados ter uma forte influência sobre a arquitetura do programa e os algoritmos que a constituem, mudanças nos dados resultarão invariavelmente em mudanças de arquitetura ou de nível de código.

Engenharia direta. Em um mundo ideal, os aplicativos seriam recriados por meio de um “motor de reengenharia” automatizado. O programa antigo seria colocado nesse motor, analisado, reestruturado e regenerado em uma forma que apresentasse os melhores aspectos da qualidade do software. Em poucas palavras, é improvável que tal “motor” algum dia apareça, mas os fabricantes de software introduziram ferramentas que fornecem um subconjunto limitado desses recursos que se destinam a domínios de aplicação específicos (por exemplo, aplicações implementadas por meio de um sistema de base de dados específico). Mais importante, essas ferramentas de reengenharia estão se tornando cada vez mais sofisticadas.

A engenharia direta não apenas recupera as informações do projeto do software existente, mas também usa as informações para alterar ou reconstituir o sistema existente em um esforço para melhorar sua qualidade geral. Em muitos casos, o software que passou pela reengenharia reimplementa a função do sistema existente e também acrescenta novas funções e/ou melhora o desempenho geral.

29.6 ENGENHARIA REVERSA

A engenharia reversa evoca uma imagem da “fenda mágica”. Você coloca na fenda uma listagem de código não documentada, criada de qualquer jeito, e do outro lado sai uma descrição completa de

⁴ A reestruturação de código tem alguns dos elementos de “refabricação”, um conceito de redesenho introduzido no Capítulo 8 e discutido em outras partes deste livro.

projeto (e documentação completa) para o programa do computador. Infelizmente, a fenda mágica não existe. A engenharia reversa pode extrair informações do projeto do código-fonte, mas o nível de abstração, a completeza da documentação, o grau segundo o qual as ferramentas e um analista humano trabalham juntos e a direcionalidade do processo são altamente variáveis.

O *nível de abstração* de um processo de engenharia reversa e as ferramentas utilizadas para realizá-lo referem-se à sofisticação das informações de projeto que podem ser extraídas do código-fonte. Idealmente, o nível de abstração deverá ser tão alto quanto possível. Isto é, o processo de engenharia reversa deverá ser capaz de derivar representações de projeto procedural (em uma abstração de baixo nível), informações de programa e de estrutura de dados (em um nível de abstração um tanto mais alto), modelos de objeto, dados e/ou modelos de fluxo de controle (em um nível de abstração relativamente alto) e modelos de entidade-relacionamento (em um nível alto de abstração). À medida que o nível de abstração aumenta, você recebe informações que lhe permitirão entender o programa mais facilmente.

A *completeza* de um processo de engenharia reversa refere-se ao nível de detalhe fornecido em um nível de abstração. Em muitos casos, a completeza diminui conforme o nível de abstração aumenta. Por exemplo, dada uma listagem de código-fonte, é relativamente fácil desenvolver uma representação completa do projeto procedural. Podem ser derivadas também representações simples de projeto arquitetônico, mas é muito mais difícil desenvolver um conjunto completo de diagramas UML ou modelos.

A completeza melhora em proporção direta com a quantidade de análise executada pela pessoa que está fazendo a engenharia reversa. *Interatividade* refere-se ao grau segundo o qual as pessoas são "integradas" com as ferramentas automáticas para criar um processo eficaz de engenharia reversa. Em muitos casos, na medida em que o nível de abstração aumenta, a interatividade deve aumentar ou a completeza será prejudicada.

Se a *direcionalidade* do processo de engenharia reversa for um único sentido, todas as informações extraídas do código-fonte serão fornecidas ao engenheiro de software que pode então usá-las durante qualquer atividade de manutenção. Se a direcionalidade for nos dois sentidos, as informações serão colocadas em uma ferramenta de reengenharia que tenta reestruturar ou regenerar o sistema antigo.

O processo de engenharia reversa está representado na Figura 29.3. Antes de começar as atividades de engenharia reversa, o código-fonte não estruturado ("ruim") é reestruturado (Seção 29.5.1)

WebRef

Recursos úteis para "recuperação de design e entendimento de programas" podem ser encontrados no site www.sdl.it.nrc.ca/projects/dr/dr.html.

FIGURA 29.3
O processo de engenharia reversa



para que contenha somente as construções de programação estruturada.⁵ Isso torna o código-fonte mais fácil de ler e proporciona a base para todas as atividades subsequentes de engenharia reversa.

O núcleo da engenharia reversa é uma atividade chamada de *extração de abstrações*. Você deve avaliar o programa antigo e, com base no código-fonte (muitas vezes não documentado), desenvolver uma especificação significativa do processamento executado, da interface de usuário aplicada e das estruturas de dados de programas ou base de dados utilizada.

29.6.1 Engenharia reversa para entender os dados

A engenharia reversa dos dados ocorre em diferentes níveis de abstração e frequentemente é a primeira tarefa da reengenharia. No nível de programa, as estruturas internas de dados de programa devem muitas vezes passar por uma engenharia reversa como parte de um trabalho de reengenharia total. Em nível de sistema, estruturas de dados globais (por exemplo, arquivos, bases de dados) passam muitas vezes por uma reengenharia para acomodar novos paradigmas de gerenciamento de base de dados (por exemplo, a mudança de arquivos de texto para sistemas de bases de dados relacionais ou orientados a objeto). A engenharia reversa das estruturas de dados globais atuais define o cenário para a introdução de uma nova base de dados para todo o sistema.

Estruturas internas de dados. As técnicas de engenharia reversa para dados internos de programa focalizam a definição de classes de objetos. Isso é feito examinando-se o código do programa para agrupar variáveis de programa relacionadas. Em muitos casos, a organização de dados dentro do código identifica tipos de dados abstratos. Por exemplo, estruturas de registro, arquivos, listas e outras estruturas de dados muitas vezes fornecem um indicador inicial das classes.

Estrutura de base de dados. Independentemente de sua organização lógica e estrutura física, uma base de dados permite a definição de objetos de dados e suporta algum método para estabelecer relações entre os objetos. Portanto, para fazer a reengenharia de um esquema de base de dados para outro é necessário entender os objetos e suas relações.

Podem ser usados os passos a seguir [Pre94] para definir o modelo de dados existente como um precursor para a reengenharia de um novo modelo de base de dados: (1) criar um modelo do objeto inicial, (2) determinar chaves candidatas (são examinados os atributos para determinar se elas são usadas para apontar para outro registro ou tabela; aquelas que servem como ponteiros tornam-se chaves candidatas), (3) refinar as classes provisórias, (4) definir generalizações, e (5) descobrir associações usando técnicas análogas à abordagem CRC. Uma vez conhecidas as informações definidas nos passos anteriores, pode ser aplicada uma série de transformações [Pre94] para mapear a estrutura antiga de base de dados em uma nova estrutura de base de dados.

29.6.2 Engenharia reversa para entender o processamento

A engenharia reversa para entender o processamento começa com uma tentativa de entender e extrair abstrações procedurais representadas pelo código-fonte. Para entender as abstrações procedurais, o código é analisado em vários níveis de abstrações: sistema, programa, componente, padrão e instruções.

A funcionalidade global de todo o sistema da aplicação deve ser entendida antes do trabalho detalhado de engenharia reversa. Isso estabelece um contexto para análise posterior e proporciona uma visão sobre os problemas de interoperabilidade entre os aplicativos no sistema. Cada um dos programas que formam o sistema de um aplicativo representa uma abstração funcional em um alto nível de detalhe. É criado um diagrama de blocos, representando a interação entre essas abstrações funcionais. Cada componente executa alguma subfunção e representa uma abstração procedural definida. É desenvolvida uma narrativa de processamento para cada componente. Em algumas situações, já existem especificações de sistema, programa e componente.



Em alguns casos, a primeira atividade de reengenharia tenta construir um diagrama de classe UML.



A abordagem para a engenharia reversa de dados para software convencional segue um caminho análogo: (1) criar um modelo de dados, (2) identificar atributos dos objetos dados, e (3) definir relações.

“Existe uma paixão pela compreensão, assim como uma paixão pela música. Essa paixão é muito comum nas crianças, mas em muitas pessoas acaba se perdendo mais tarde.”

Albert Einstein

⁵ O código pode ser reestruturado usando um *motor de reestruturação* — uma ferramenta que reestrutura código-fonte.

Quando esse é o caso, as especificações são revistas para verificar a conformidade com o código existente.⁶

A situação torna-se mais complexa quando é considerado o código dentro de um componente. Você deverá procurar seções de código que representam padrões procedurais genéricos. Em quase todos os componentes, uma seção do código prepara dados para processamento (dentro do módulo), outra seção faz o processamento e outra prepara os resultados do processamento para exportação do componente. Em cada uma dessas seções, podem-se encontrar padrões menores; por exemplo, validação de dados e verificações de limites, que muitas vezes ocorrem na seção do código que prepara os dados para processamento.

Para sistemas grandes, a engenharia reversa em geral é feita usando-se uma abordagem semiautomática. Podem ser empregadas ferramentas automatizadas para ajudá-lo a entender a semântica do código existente. O resultado desse processo é então passado para as ferramentas de reestruturação e engenharia direta para completar o processo de reengenharia.


29.6.3 Engenharia reversa das interfaces de usuário

As GUIs sofisticadas tornaram-se uma exigência para produtos e sistemas de todos os tipos baseados em computadores. Portanto, o desenvolvimento de interfaces de usuário tornou-se um dos tipos mais comuns de atividade de reengenharia. Mas antes de recriar uma interface de usuário, deverá ocorrer a engenharia reversa.

Para entender completamente uma interface de usuário, deve ser especificada a estrutura e o comportamento da interface. Merlo e seus colegas [Mer93] sugerem três questões básicas que devem ser respondidas quando se começa a engenharia reversa da interface de usuário (UI):

- Quais são as ações básicas (por exemplo, teclas e cliques de mouse) que a interface deve processar?
- Qual é a descrição compacta da resposta comportamental do sistema para essas ações?
- O que significa “substituição” ou, mais precisamente, que conceito de equivalência de interfaces é relevante aqui?

A notação de modelagem comportamental (Capítulo 7) pode proporcionar um meio para desenvolver respostas para as duas primeiras questões. Grande parte das informações necessárias para criar um modelo comportamental pode ser obtida observando-se a manifestação externa da interface. Mas as informações adicionais necessárias para criar o modelo comportamental devem ser extraídas do código.

 Como podemos entender o funcionamento de uma interface de usuário existente?



Engenharia reversa

Objetivo: ajudar os engenheiros de software a entender a estrutura de projeto interna de programas complexos.

Mecânica: em muitos casos, as ferramentas de engenharia reversa aceitam código-fonte como entrada e produzem uma variedade de representações de projeto estrutural, procedural, dados e comportamental.

Ferramentas representativas:⁷

Imagix 4D, desenvolvida pela Imagix (www.imagix.com), “ajuda os desenvolvedores de software a entender software

em C e C++ complexo ou legado” pela engenharia reversa e documentação do código-fonte.

Understand, desenvolvida pela Scientific Toolworks, Inc. (www.scitools.com), analisa Ada, Fortran, C, C++ e Java “para fazer a engenharia reversa, documentar automaticamente, calcular métricas de código e ajudá-lo a entender, navegar e manter o código-fonte”.

Pode ser encontrada uma lista abrangente de ferramentas para engenharia reversa no site <http://scgwiki.iam.unibe.ch:8080/SCG/370>.

FERRAMENTAS DO SOFTWARE

⁶ Frequentemente, especificações escritas no início da vida do programa nunca são atualizadas. À medida que são feitas as atualizações, o código não está mais em conformidade com a especificação.

⁷ As ferramentas aqui apresentadas não significam um aval, mas sim uma amostra dessa categoria. Na maioria dos casos, seus nomes são marcas registradas pelos respectivos desenvolvedores.

É importante notar que uma GUI substituta pode não refletir exatamente a interface antiga (na verdade, pode ser radicalmente diferente). Muitas vezes compensa desenvolver uma nova metáfora de interação. Por exemplo, uma interface de usuário antiga requer que o usuário forneça um fator de escala (variando de 1 a 10) para reduzir ou ampliar uma imagem gráfica. Uma GUI que passou pela reengenharia pode usar uma barra de rolagem e o mouse para executar a mesma função.

29.7 REESTRUTURAÇÃO



Embora a reestruturação de código possa aliviar problemas imediatos associados com o debugging ou pequenas alterações, ela não é a reengenharia. O verdadeiro benefício é conseguido somente quando os dados e a arquitetura são reestruturados.

A reestruturação de software modifica o código-fonte e/ou os dados para torná-lo mais amigável para futuras alterações. Em geral, a reestruturação não modifica a arquitetura geral do programa. Ela tende a concentrar-se nos detalhes de projeto dos módulos individuais e nas estruturas de dados locais definidas nos módulos. Se o trabalho de reestruturação se estende além dos limites de módulo e abrange a arquitetura do software, a reestruturação passa a ser engenharia direta (Seção 29.7).

A reestruturação ocorre quando a arquitetura básica de um aplicativo é sólida, mesmo que as partes técnicas internas necessitem de um retrabalho. Ela ocorre quando partes importantes do software são reparáveis e somente um subconjunto de todos os módulos e dados necessita de uma modificação mais extensa.⁸

29.7.1 Reestruturação de código

A *reestruturação de código* é feita para obter um projeto que produz a mesma função, mas com mais qualidade do que o programa original. Em geral, as técnicas de reestruturação de código (por exemplo, as técnicas de simplificação lógica de Warnier [War74]) modelam a lógica de programação usando álgebra booleana e aplicam uma série de regras de transformação que resulta na lógica reestruturada. O objetivo é pegar um código “emaranhado” e obter um projeto procedural que esteja em conformidade com a filosofia de programação estruturada (Capítulo 10).

Outras técnicas de reestruturação também foram propostas para uso com as ferramentas de reengenharia. Um diagrama de intercâmbio de recursos mapeia cada módulo de programa e os recursos (tipos de dados, procedimentos e variáveis) permutados entre ele e outros módulos. Criando representações do fluxo de recursos, a arquitetura do programa pode ser reestruturada para obter o mínimo acoplamento entre os módulos.

29.7.2 Reestruturação de dados

Antes de iniciar a reestruturação de dados, deve ser feita uma atividade de engenharia reversa chamada de *análise do código-fonte*. São avaliadas todas as instruções da linguagem de programação que contenham definições de dados, descrições de arquivo, I/O e descrições de interface. A finalidade é extrair itens de dados e objetos, para obter informações sobre fluxo de dados e para entender as estruturas de dados existentes que precisam ser implementadas. Essa atividade às vezes é chamada de *análise de dados*.

Uma vez completada a análise de dados, inicia-se o *reprojeto dos dados*. Em sua forma mais simples, uma etapa de *padronização de registro de dados* esclarece as definições de dados para obter consistência entre nomes de itens de dados ou formatos de registros físicos em uma estrutura de dados ou formato de arquivo existentes. Outra forma de reprojeto, chamada de *racionalização de nomes de dados*, garante que todas as convenções de nomes de dados estejam de acordo com os padrões locais e que os pseudônimos (alias) sejam eliminados à medida que os dados fluem através do sistema.

Quando a reestruturação vai além da padronização e racionalização, são feitas modificações físicas nas estruturas de dados existentes para tornar mais eficaz o projeto de dados. Isso pode

⁸ Às vezes é difícil fazer a distinção entre reestruturação extensiva e redesevolvimento. Ambos os casos são reengenharia.

significar a transformação de um formato de arquivo para outro ou, em alguns casos, a transformação de um tipo de base de dados para outro.



Reestruturação de software

Objetivo: o objetivo das ferramentas de reestruturação é transformar software não estruturado e mais antigo em linguagens de programação e estruturas de projeto modernas.

Mecânica: em geral, o código-fonte é o elemento de entrada e ele é transformado em um programa mais bem estruturado. Em alguns casos, a transformação ocorre na mesma linguagem de programação. Em outros, uma linguagem de programação mais antiga é transformada em uma mais moderna.

Ferramentas representativas:

DMS Software Reengineering Toolkit, desenvolvida pela Semantic Design (www.semdesigns.com), fornece uma variedade de recursos de reestruturação para COBOL, C++, Java, Fortran 90 e VHDL.

Clone Doctor, desenvolvida pela Semantic Designs, Inc. (www.semdesigns.com), analisa e transforma programas escritos em C, C++, Java ou COBOL ou qualquer outra linguagem de programação de computador baseada em texto.

plusFORT, desenvolvida pela Polyhedron (www.polyhedron.com), é um conjunto de ferramentas FORTRAN contendo recursos para reestruturar programas FORTRAN mal projetados em programas modernos padrão FORTRAN ou C.

Links para uma variedade de ferramentas de engenharia e engenharia reversa podem ser encontrados em www.csse.monash.edu/~ipeake/reeng/free-swre-tools.html e www.cs.ualberta.ca/~kenw/toolsdir/all.html.

29.8 ENGENHARIA DIRETA

? Que opções existem quando encontramos um programa mal projetado e mal implementado?

Um programa com um fluxo de controle, que graficamente equivale a um emaranhado, com módulos de 2 mil instruções, algumas poucas linhas de comentários em 290 mil instruções de código-fonte e nenhuma outra documentação deve ser modificado para acomodar alterações e requisitos de usuário. Temos as seguintes opções:

1. Podemos trabalhar arduamente fazendo modificação após modificação, enfrentando os problemas do projeto *ad hoc* e do código-fonte confuso para implementar as mudanças necessárias.
2. Podemos tentar entender os detalhes internos do programa em um esforço para tornar as modificações mais eficazes.
3. Podemos reprojeter, recodificar e testar as partes do software que requerem modificação, aplicando uma abordagem de engenharia de software a todos os segmentos revisados.
4. Podemos reprojeter completamente, recodificar e testar o programa, usando ferramentas de reengenharia para ajudar a entendermos o projeto atual.



AVISO
A reengenharia é bem semelhante à limpeza de seus dentes. Você pode pensar em mil razões para adiar, e conseguirá um jeito de retardar por um tempo. Mas, eventualmente, sua técnica de adiamento se voltará contra você, causando-lhe dor.

Não há uma única opção "correta". As circunstâncias podem recomendar a primeira opção mesmo que as outras sejam mais desejáveis.

Em vez de esperar até que seja solicitada uma manutenção, a organização de desenvolvimento ou suporte usa os resultados da análise de inventário para selecionar um programa que (1) permanecerá em uso por certo número de anos, (2) está sendo utilizado no momento com sucesso e (3) pode passar por modificações importantes ou melhoramentos em futuro próximo. Então é aplicada a opção 2, 3 ou 4.

Em uma primeira impressão, a sugestão para que você redesenvolva um grande programa quando uma versão ainda funcione pode ser bastante extravagante. Mas antes de fazer um julgamento, considere os seguintes pontos:

9 As ferramentas aqui apresentadas não significam um aval, mas sim uma amostra dessa categoria. Na maioria dos casos, seus nomes são marcas registradas pelos respectivos desenvolvedores.

1. O custo para manter uma linha de código-fonte pode ser 20 a 40 vezes o do desenvolvimento inicial daquela linha.
2. O reprojeto da arquitetura de software (programa e/ou estrutura de dados), usando conceitos modernos de projeto, pode facilitar muito a manutenção futura.
3. Como já existe um protótipo do software, a produtividade do desenvolvimento deverá ser muito mais alta do que a média.
4. O usuário agora tem experiência com o software. Portanto, novos requisitos e a direção das mudanças podem ser definidos com grande facilidade.
5. Ferramentas automatizadas para reengenharia facilitarão algumas partes do trabalho.
6. Existirá uma configuração de software completa (documentos, programas e dados) depois de uma manutenção preventiva.

Um grande departamento de desenvolvimento de software interno (por exemplo, um grupo de desenvolvimento de sistemas comerciais para uma grande empresa de produtos de consumo) pode ter de 500 a 2 mil programas em produção sob sua responsabilidade. Esses programas podem ser classificados por importância e examinados como candidatos para engenharia direta.

O processo de engenharia direta aplica os princípios, conceitos e métodos de engenharia de software, para recriar um aplicativo. Em muitos casos, a engenharia direta não cria só um equivalente moderno de um programa antigo. Em vez disso, novos requisitos de usuário e tecnologia são integrados ao trabalho de reengenharia. O programa redesenvolvido amplia a capacidade do aplicativo antigo.

29.8.1 Engenharia direta para arquiteturas cliente-servidor

Durante as últimas décadas, muitos aplicativos para mainframes passaram por operações de reengenharia para acomodar arquiteturas cliente-servidor (incluindo WebApps). Essencialmente, recursos de computação centralizados (incluindo software) são distribuídos entre muitas plataformas cliente. Embora possa ser projetada uma variedade de diferentes ambientes distribuídos, a aplicação para mainframes típica que passa por uma reengenharia, transformando-se em uma arquitetura cliente-servidor, tem as seguintes características:

- A funcionalidade da aplicação migra para cada computador cliente.
- Novas interfaces GUI são implementadas nas instalações do cliente.
- Funções de base de dados são atribuídas ao servidor.
- Funcionalidade especializada (por exemplo, análise que demanda muitos cálculos) podem permanecer no servidor.
- Novos requisitos de comunicação, segurança, arquivamento e controle podem ser estabelecidos tanto no lado do cliente quanto no do servidor.

É importante notar que a migração de computação mainframe para cliente-servidor requer reengenharia tanto de negócio quanto de software. Além disso, deverá ser estabelecida uma "infraestrutura de rede corporativa" [Jay94].

A reengenharia para aplicações cliente-servidor começa com uma análise completa do ambiente comercial que abrange o mainframe existente. Três leis de abstração podem ser identificadas. A *base de dados* estabelece a fundação de uma arquitetura cliente-servidor e gerencia transações e consultas dos aplicativos do servidor. No entanto, essas transações e consultas devem ser controladas com base em um conjunto de regras de negócio (definidas por um processo de negócio existente ou que passou por uma reengenharia). Aplicações cliente fornecem funcionalidade destinada à comunidade de usuários.

As funções do sistema de gerenciamento de base de dados e a arquitetura de dados da base de dados existentes devem passar por uma engenharia reversa como precursoras para o reprojeto da camada fundamental da base de dados. Em alguns casos, é criado um modelo de dados



Em alguns casos, a migração para uma arquitetura cliente-servidor deverá ser considerada não como uma reengenharia, mas como um novo trabalho de desenvolvimento. A reengenharia entra em cena apenas quando funcionalidade específica do sistema antigo deve ser integrada em uma nova arquitetura.

(Capítulo 6). De qualquer forma, a base de dados cliente-servidor passa por uma reengenharia para garantir que as transações sejam executadas de uma maneira consistente, que todas as atualizações sejam executadas somente por usuários autorizados, que as regras de negócio básicas sejam seguidas (por exemplo, antes que o registro de um fornecedor seja deletado, o servidor assegura que não existem contas a pagar, contratos, ou comunicações para aquele fornecedor), que as consultas possam ser acomodadas eficientemente e que seja estabelecida uma capacidade total de arquivamento.

A camada das regras de negócio representa software residente tanto no cliente quanto no servidor. Esse software executa tarefas de controle e coordenação para assegurar que as transações e consultas entre o aplicativo cliente e a base de dados estejam em conformidade com processos de negócio estabelecidos.

A camada dos aplicativos cliente implementa funções de negócio requeridas por grupos específicos de usuários finais. Em muitas ocasiões, um aplicativo para mainframe é segmentado em uma série de aplicativos para desktop menores e modificados por meio da reengenharia. A comunicação entre os aplicativos para desktop (quando necessária) é controlada pela camada de regras de negócio.

Uma discussão ampla sobre projeto e reengenharia de software cliente-servidor pode ser encontrada em livros dedicados ao assunto. Se você tiver interesse, veja [Van02], [Cou00], ou [Orf99].

29.8.2 Engenharia direta para arquiteturas orientadas a objeto

A reengenharia de software orientada a objeto tornou-se o paradigma de desenvolvimento preferido por muitas organizações. Mas o que dizer sobre os aplicativos desenvolvidos usando-se métodos convencionais? Em alguns casos, a resposta é deixar esses aplicativos “como estão”. Em outros, mais antigos, devem passar por uma reengenharia para que possam ser facilmente integrados em sistemas orientados a objeto de grande porte.

A reengenharia de software convencional para uma implementação orientada a objeto usa muitas das mesmas técnicas discutidas na Parte 2 deste livro. Primeiro, o software passa por uma engenharia reversa de modo que possam ser criados modelos de dados, funcional e comportamental apropriados. Se o sistema ao qual se aplica a reengenharia amplia a funcionalidade ou comportamento do aplicativo original, são criados casos de uso (Capítulos 5 e 6). Os modelos de dados desenvolvidos durante a engenharia reversa são então utilizados em conjunto com a modelagem CRC (Capítulo 6) para estabelecer a base para a definição de classes. Hierarquias de classe, modelos de relacionamenmto de objetos, modelos de comportamento de objeto e subsistemas são definidos, e inicia-se o projeto orientado a objeto.

Na medida em que a engenharia direta orientada a objeto avança da análise para o projeto, pode ser invocado um modelo de processo CBSE (Capítulo 10). Se o aplicativo existente reside dentro de um domínio que já seja constituído por muitos aplicativos orientados a objeto, é provável que exista uma biblioteca de componentes robusta e que possa ser empregada durante a engenharia direta.

Para aquelas classes que devem ser criadas desde o início, pode ser possível reutilizar algoritmos e estruturas de dados do aplicativo convencional existente. No entanto, estes devem ser reprojatados para ficar em conformidade com a estrutura orientada a objeto.

29.9 A ECONOMIA DA REENGENHARIA

Em um mundo perfeito, todo programa não manutenível seria aposentado imediatamente, para ser substituído por aplicativos de alta qualidade, desenvolvidos usando modernas práticas de engenharia de software. Mas vivemos em um mundo de recursos limitados. A reengenharia consome recursos que podem ser utilizados para outras finalidades de negócio. Portanto, antes de pensar em fazer a reengenharia de um aplicativo existente, deve-se executar uma análise de custo-benefício.

"Você pode gastar um pouco agora ou gastar muito mais tarde."

Cartaz em uma revendedora de automóveis sugerindo manutenção no carro

Um modelo de análise custo-benefício para reengenharia foi proposto por Sneed [Sne95]. São definidos nove parâmetros:

- P_1 = custo anual corrente de manutenção para um aplicativo
- P_2 = custo anual corrente das operações para um aplicativo
- P_3 = valor de negócio anual corrente de um aplicativo
- P_4 = custo de manutenção anual previsto após a reengenharia
- P_5 = custo anual previsto das operações após a reengenharia
- P_6 = valor de negócio anual previsto após a reengenharia
- P_7 = custos estimados da reengenharia
- P_8 = prazo estimado para fazer a reengenharia
- P_9 = fator de risco da reengenharia ($P_9 = 1,0$ é nominal)
- L = expectativa de vida do sistema

O custo associado com a manutenção continuada de um aplicativo candidato à reengenharia (a reengenharia ainda não foi feita) pode ser definido como

$$C_{\text{manut}} = [P_3 - (P_1 + P_2)] \times L \quad (29.1)$$

Os custos associados com a reengenharia são definidos usando a seguinte relação:

$$C_{\text{reeng}} = P_6 - (P_4 + P_5) \times (L - P_8) - (P_7 \times P_9) \quad (29.2)$$

Usando os custos apresentados nas Equações (29.1) e (29.2), o benefício total da reengenharia pode ser calculado como

$$\text{Custo-benefício} = C_{\text{reeng}} - C_{\text{manut}} \quad (29.3)$$

A análise custo-benefício apresentada nessas equações pode ser feita para todos os aplicativos de alta prioridade identificados durante a análise de inventário (Seção 29.4.2). Aqueles aplicativos que apresentam o custo-benefício mais alto podem ser identificados para reengenharia, enquanto o trabalho com os outros aplicativos pode ser adiado até que os recursos estejam disponíveis.

29.10 RESUMO

A manutenção e o suporte de software são atividades contínuas que ocorrem por todo o ciclo de vida de um aplicativo. Durante essas atividades, defeitos são corrigidos, aplicativos são adaptados a um ambiente operacional ou de negócio em mutação, melhorias são implementadas por solicitação dos interessados e é fornecido suporte aos usuários quando integram um aplicativo em seu fluxo de trabalho pessoal ou corporativo.

A reengenharia ocorre em dois níveis de abstração. No nível de negócio, ela concentra-se nos processos de negócio para melhorar a competitividade em alguma área do negócio. No nível de software, a reengenharia examina os sistemas de informação e os aplicativos, com a finalidade de reestruturá-los para que tenham uma melhor qualidade.

A reengenharia de processos de negócio (*business process reengineering* - BPR) define metas de negócio; identifica e avalia processos de negócio existentes (no contexto das metas definidas); especifica e projeta processos revisados; cria protótipos, refina e os viabiliza em um negócio. A BPR tem um foco que se estende além do software. O resultado da BPR é muitas vezes a definição de maneiras pelas quais a tecnologia de informação pode suportar melhor o negócio.

A reengenharia de software abrange uma série de atividades que incluem análise de inventário, reestruturação da documentação, engenharia reversa, reestruturação de programas e dados e engenharia direta. A finalidade dessas atividades é criar versões dos programas existentes que tenham melhor qualidade e melhor manutenibilidade — programas que serão viáveis para o século 21.

O custo-benefício da reengenharia pode ser determinado quantitativamente. O custo do estado atual, isto é, o custo associado ao suporte e manutenção continuados de um aplicativo existente, é comparado aos custos projetados da reengenharia e a redução resultante nos custos de manutenção e suporte. Em quase todos os casos nos quais um programa tem uma vida longa e apresenta no momento uma manutenibilidade ou suportabilidade ruim, a reengenharia representa uma estratégia de negócio eficiente em termos de custo.

PROBLEMAS E PONTOS A PONDERAR

- 29.1.** Considere um trabalho que tenha realizado nos últimos cinco anos. Descreva os processos de negócio nos quais você tomou parte. Use o modelo BPR descrito na Seção 29.4.2 para recomendar alterações no processo em um esforço para torná-lo mais eficiente.
- 29.2.** Faça alguma pesquisa sobre a eficiência da reengenharia dos processos de negócio. Apresente argumentos a favor e contra essa abordagem.
- 29.3.** O seu professor selecionará um dos programas que todos na classe tenham desenvolvido durante o curso. Troque o seu programa de forma aleatória com algum outro aluno na classe. Não explique nem detalhe o programa. Agora, implemente uma melhoria (especificada pelo seu professor) no programa que você recebeu.
- Execute todas as tarefas de engenharia de software incluindo um breve detalhamento (mas não com o autor do programa).
 - Mantenha um controle preciso de todos os erros encontrados durante o teste.
 - Discuta sua experiência na classe.
- 29.4.** Explore a checklist da análise de inventário apresentada no site da SEPA e tente desenvolver um sistema de classificação quantitativa de software que poderia ser aplicado a programas existentes, em um trabalho para escolher programas candidatos para a reengenharia. O seu sistema deve se estender além da análise econômica apresentada na Seção 29.9.
- 29.5.** Sugira alternativas em documentação impressa ou eletrônica que poderia servir de base para a reestruturação da documentação. (Sugestão: pense em novas tecnologias descritivas que poderiam ser usadas para mostrar a finalidade do software.)
- 29.6.** Algumas pessoas acreditam que a tecnologia de inteligência artificial aumentará o nível de abstração do processo de engenharia reversa. Faça alguma pesquisa sobre esse assunto (isto é, o uso de inteligência artificial (*artificial intelligence* - AI) para a engenharia reversa) e escreva um pequeno artigo que destaca esse ponto.
- 29.7.** Por que a completeza é difícil de ser atingida na medida em que o nível de abstração aumenta?
- 29.8.** Por que a interatividade deve aumentar se a completeza tiver de aumentar?
- 29.9.** Usando informações obtidas na Web, apresente características de três ferramentas de engenharia reversa para a sua classe.
- 29.10.** Há uma diferença sutil entre reestruturação e engenharia direta. Qual é?
- 29.11.** Pesquise a literatura e/ou fontes na Internet para encontrar um ou mais artigos que discutem estudos de casos de reengenharia de mainframe para cliente-servidor. Apresente um resumo.
- 29.12.** Como você determinaria P_4 até P_7 no modelo de custo-benefício apresentado na Seção 29.9?

LEITURAS E FONTE DE INFORMAÇÃO COMPLEMENTARES

É irônico que a manutenção e suporte de software representem as atividades mais custosas na vida útil de um aplicativo e, no entanto, foram escritos menos livros sobre manutenção e suporte do que sobre qualquer outro tópico importante de engenharia de software. Entre as recentes adições à literatura estão os livros de Jarzabek (*Effective Software Maintenance and Evolution*, Auerbach, 2007), Grubb e Takang (*Software Maintenance: Concepts and Practice*, World

Scientific Publishing Co., 2d ed., 2003), e Pigoski (*Practical Software Maintenance*, Wiley, 1996). Esses livros abrangem as práticas básicas de manutenção e suporte e apresentam diretrizes úteis de gerenciamento. As técnicas de manutenção que focalizam ambientes cliente-servidor são discutidas por Schneberger (*Client/Server Software Maintenance*, McGraw-Hill, 1997). A pesquisa atual em "evolução de software" é apresentada em uma antologia editada por Mens e Demeyer (*Software Evolution*, Springer, 2008).

Como muitos tópicos atuais na área de negócios, a euforia que envolve a reengenharia de processos de negócio deu origem a uma visão mais pragmática do assunto. Hammer e Champy (*Reengineering the Corporation*, HarperBusiness, revised edition, 2003) fomentaram logo o interesse com sua obra que se tornou best-seller. Outros livros de Smith e Fingar [*Business Process Management (BPM): The Third Wave*, Meghan-Kiffer Press, 2003], Jacka e Keller (*Business Process Mapping: Improving Customer Satisfaction*, Wiley, 2001), Sharp e McDermott (*Workflow Modeling*, Artech House, 2001), Andersen (*Business Process Improvement Toolbox*, American Society for Quality, 1999), e Harrington et al. (*Business Process Improvement Workbook*, McGraw-Hill, 1997) apresentam estudos de casos e diretrizes detalhadas para BPR.

Fong (*Information Systems Reengineering and Integration*, Springer, 2006) descreve técnicas de conversão de base de dados, engenharia reversa e engenharia direta da forma como são aplicadas para a maioria dos sistemas de informação. Demeyer e seus colegas (*Object Oriented Reengineering Patterns*, Morgan Kaufmann, 2002) proporcionam uma visão baseada em padrões sobre como refazer e/ou recriar sistemas orientados a objeto. Secord e seus colegas (*Modernizing Legacy Systems*, Addison-Wesley, 2003), Ulrich (*Legacy Systems: Transformation Strategies*, Prentice Hall, 2002), Valenti (*Successful Software Reengineering*, IRM Press, 2002), e Rada (*Reengineering Software: How to Reuse Programming to Build New, State-of-the-Art Software*, Fitzroy Dearborn Publishers, 1999) concentram-se em estratégias e práticas para reengenharia em nível técnico. Miller (*Reengineering Legacy Software Systems*, Digital Press, 1998) "proporciona uma estrutura para manter sistemas de aplicativos sincronizados com as estratégias de negócio e com as mudanças da tecnologia".

Cameron (*Reengineering Business for Success in the Internet Age*, Computer Technology Research, 2000) e Umar [*Application (Re)Engineering: Building Web-Based Applications and Dealing with Legacies*, Prentice Hall, 1997] fornecem orientações para organizações que querem transformar sistemas legados em um ambiente baseado na Web. Cook (*Building Enterprise Information Architectures: Reengineering Information Systems*, Prentice Hall, 1996) aborda a ligação entre BPR e tecnologia da informação. Aiken (*Data Reverse Engineering*, McGraw-Hill, 1996) discute como reunir, reorganizar e reutilizar dados organizacionais. Arnold (*Software Reengineering*, IEEE Computer Society Press, 1993) montou uma excelente antologia dos primeiros artigos que focalizavam as tecnologias de reengenharia de software.

Uma grande variedade de fontes de informação sobre reengenharia de software está disponível na Internet. Uma lista atualizada das referências da Web relevantes para a manutenção e reengenharia de software pode ser encontrada no site www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm.