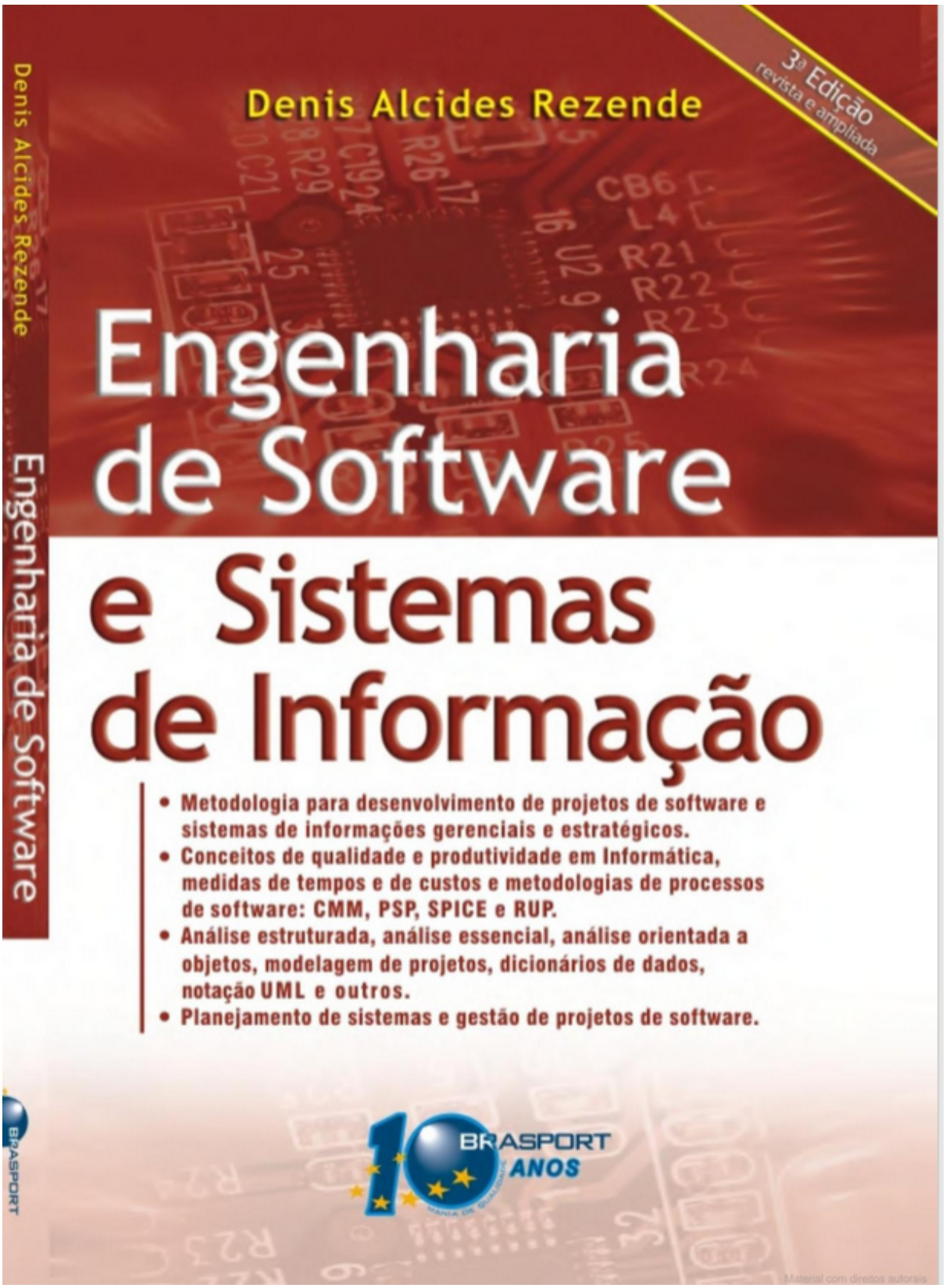
Denis Alcides Rezende
3ª Edição
revista e ampliada
Engenharia de Software
e Sistemas de Informação
• Metodologia para desenvolvimento de projetos de software e sistemas de informações gerenciais e estratégicos.
• Conceitos de qualidade e produtividade em Informática, medidas de tempos e de custos e metodologias de processos de software: CMM, PSP, SPICE e RUP.
• Análise estruturada, análise essencial, análise orientada a objetos, modelagem de projetos, dicionários de dados, notação UML e outros.
• Planejamento de sistemas e gestão de projetos de software.
BRASPORT
10 ANOS
Material com direitos autorais

CICLO DE DESENVOLVIMENTO DE SOFTWARE E SISTEMAS

Dentre as metodologias utilizadas nas organizações, onde há padrões próprios para desenvolvimento de sistemas, é usual a adoção de uma metodologia que aborde o desenvolvimento de *software* por meio de suas etapas. Apresenta de uma maneira bastante abrangente, que poderá ser utilizado em diferentes ambientes, sistemas e metodologias, incluindo sistemas especialistas, sistemas de tempo real, protótipos etc.

O ciclo de vida e desenvolvimento apresentado por YOURDON (1989) é dividido em *etapas*: estudo ou estudo de viabilidade; análise de sistemas; projeto; implementação; geração do teste de aceite; garantia da qualidade; descrição de procedimentos; conversão de banco de dados; instalação.

Nas literaturas atualizadas de Yourdon (YOURDON; CONSTANTINE, 1978; (YOURDON; ARGILA, 1999), a *garantia da qualidade* que era elaborada apenas após a implementação do projeto, agora têm sido feita em todas as fases. Este novo ciclo tem contribuído para as organizações obterem a ISO 9001, ISO 9000-3 e outras certificações de sistemas de informação e de software.

Etapas e subetapas

O ciclo de desenvolvimento pode indicar as seguintes etapas (YOURDON, 1989):

a. Estudo ou estudo de viabilidade

Identificar as deficiências atuais; estabelecer objetivos do novo sistema; gerar cenários aceitáveis; preparar encargos de projeto.

b. Análise de sistemas

Desenvolver o modelo ambiental; desenvolver o modelo comportamental; estabelecer os limites homem-máquina; executar a análise custo-benefício; selecionar opção; restringir o sistema; especificação do pacote.

c. Projeto

Alocar especificações para os processadores; alocar especificações às tarefas; derivar o diagrama estrutural; avaliar diagrama estrutural; projetar módulos; projetar o banco de dados; empacotamento do projeto.

d. Implementação

Solucionar próximo módulo; codificar módulo; testar o esqueleto do sistema.

e. Geração do teste do aceite

Gerar plano de teste; preparar testes de performance; preparar testes de vias normais; preparar testes de vias de erro; empacotar os testes.

f. Garantia da qualidade

Teste final ou teste de aceite, comparando ao projeto de implantação.

g. Descrição de procedimentos

Descrições das atividades operacionais do cliente ou usuário normalmente correspondem ao manual do usuário.

h. Conversão de banco (base) de dados

Pode envolver mais trabalho e planejamento do que desenvolvimento de programas para o novo sistema e, em outros casos, pode não haver uma base de dados para ser convertida.

i. Instalação

Atividade final, suas entradas são o manual do usuário, o banco de dados convertido e o sistema de aceite.

Descrição das subetapas

a. Estudo ou estudo de viabilidade

Tipicamente, ela começa quando o cliente ou usuário requisita que uma ou mais partes do seu trabalho sejam automatizadas. Também chamada de Estudo Preliminar ou Estudo Inicial.

As subatividades desta etapa são: identificar as deficiências atuais (entrevistando-se os clientes ou usuários para identificar os aspectos do ambiente que sejam inadequados); estabelecer objetivos do novo sistema (como sendo uma lista das funções requisitadas, requisitos de rendimento, limitações de custo, requisitos de confiabilidade); gerar cenários aceitáveis (a partir de limitações de prazo, horas-homem, orçamento e limitações opera-

cionais. Sugerem-se alguns cenários aceitáveis, resumidamente, de um novo sistema proposto e um resumo de seu custo-benefício); preparar encargos de projeto (que serão usados para guiar o restante do projeto, por meio de plano de trabalho detalhado, com os custos e benefícios associados a um cenário, identificando a relação custo-benefício).

b. Análise de sistemas

O principal propósito da atividade de análise é transformar suas duas principais entradas, a política do cliente ou usuário e os encargos do projeto, em uma especificação estruturada do projeto. Isso envolve a modelagem do ambiente do cliente ou usuário com os Diagramas de Fluxo de Dados (DFD), Dicionários de Dados, bem como de outras ferramentas. Também chamada de Análise do Sistema Atual e Projeto Lógico ou Anteprojeto (o que fazer).

As subatividades desta etapa são: desenvolver o modelo ambiental (é à parte do modelo essencial do sistema; modelo que mostra como o sistema interage com o ambiente externo; possui duas entradas: os encargos do projeto e a política do cliente ou usuário; produz duas saídas: diagrama de contexto e lista de eventos ou estímulos no ambiente externo a que o sistema deve responder); desenvolver o modelo comportamental (é à parte do modelo essencial do sistema; modelo que indica o que o sistema deve fazer para interagir satisfatoriamente com o ambiente externo; é o desenvolvimento dos diagramas, especificações e dicionários de dados); estabelecer os limites homem-máquina (após o modelo essencial ter sido completado, tem-se como saída um 'conjunto' de novos modelos físicos, então serão definidos quais os processos que serão executados pelo computador e pelo homem); executar a análise custo-benefício (baseadas em estimativas mais precisas de custos); selecionar opção (DFDs nivelados e Dicionário de Dados que se tornaram físicos, distinguindo as partes do sistema que serão automatizadas e aquelas que serão executadas por seres humanos, ou seja, o modelo físico é selecionado, embasado num relatório de custo-benefício); restringir o sistema (contendo um resumo de todas as restrições físicas impostas sobre o novo sistema proposto, por exemplo, não aquisição de máquinas, limitações de linguagens); especificação do pacote (que é uma especificação estruturada, com uma breve narrativa, um índice, uma descrição geral de uso e composição da especificação; num único documento são integradas as diversas subatividades).

c. Projeto

A atividade de projeto refere-se à alocação de partes da especificação aos processadores apropriados e disponíveis (computadores e pessoal envolvi-

do) e as tarefas apropriadas dentro de cada processador. Dentro de cada tarefa, a atividade de projeto preocupa-se com o desenvolvimento de uma hierarquia apropriada de módulos de 'programas' e interfaces entre esses módulos para implementar a especificação criada na fase anterior. Também chamada de Projeto Lógico (o que fazer) e início do Projeto Físico (como fazer).

As subatividades desta etapa são: alocar especificações para os processadores (como resultado tem-se miniespecificações, além de um conjunto de DFDs e DER e outros diagramas, podem-se documentar as interfaces ou protocolo de comunicação); alocar especificações às tarefas (é um conjunto de minimodelos, um para cada tarefa); derivar o diagrama estrutural (produzir um conjunto de diagramas estruturais, para cada um dos grupos de diagrama de fluxo de dados em nível de tarefa; é a representação gráfica de uma hierarquia de módulos, incluindo documentação das interfaces entre módulos); avaliar diagrama estrutural (rejeitando-o, quando deixar dúvidas sobre sua conexão, coesão, controles etc., ou refinando-o, quando apresentar problemas de conexão, coesão etc.); projetar módulos (executando o projeto detalhado de cada módulo no sistema, expresso em pseudocódigo, tabelas de decisão, diagramas ou mesmo fluxogramas); projetar o banco de dados (com base na informação lógica dos componentes do Dicionário de Dados e do DER, (juntamente com as restrições físicas); gerar documento descrevendo o projeto físico do banco de dados; não se utilizando banco de dados, considerar 'bases de dados'; empacotamento do projeto (produz um documento denominado 'especificação do projeto', considerando o impacto do projeto inteiro no ambiente operacional).

d. Implementação

Essa atividade inclui a codificação e a integração dos módulos em um esqueleto, progressivamente mais complexo, do sistema final. Também chamada de Projeto Físico (como fazer) e início do Projeto de Implantação.

As subatividades desta etapa são: solucionar próximo módulo (definindo a seqüência em que os módulos serão implementados, sendo que o sistema será codificado, integrado e testado de forma incremental ou decomposição modular sucessiva); codificar módulo (a partir da escolha de uma linguagem de programação competente); testar o esqueleto do sistema (executando o teste do módulo e a integração do sistema, simultaneamente).

e. Geração do teste do aceite

A especificação estruturada deve conter toda a informação necessária para definir um sistema aceitável do ponto de vista do cliente ou usuário. Assim,

uma vez gerada a especificação, o trabalho pode começar na atividade de geração de um grupo de casos de teste de aceite a partir da especificação estruturada. Também chamada de Projeto de Implantação.

As subatividades desta etapa são: gerar plano de teste (onde se estabelece a pessoa ou grupo responsável por testar o ambiente, com procedimentos padrões, verificando-se os possíveis erros e os resultados reais e os esperados); preparar testes de performance (verificando os requisitos de tempo e volume da transação, em relação ao ambiente disponível); preparar testes de vias normais (confirmando se o sistema realmente faz o desejado para entradas válidas, considerando limites de entrada, de saída e funcionais); preparar testes de vias de erro (normalmente utilizam-se clientes ou usuários para tentar "enganar" o sistema com dados não usuais ou errados); empacotar os testes (produz um documento denominado 'especificação e resultado de testes', considerando as especificações do projeto).

f. Garantia da qualidade

A garantia de qualidade é também conhecida como teste final ou teste de aceite, comparando o Projeto de Implantação. Essa atividade requer, como sua entrada, dados do teste de aceite gerados na atividade anterior e um sistema integrado produzido na atividade de implementação. Observa-se a satisfação do cliente ou usuário no tocante ao seu atendimento de expectativas. Também chamada de Projeto de Implantação.

Bom lembrar que a *Engenharia de Software* defende a garantia da qualidade do *software* já no ato da *construção* do mesmo. Neste sentido, aqui pode residir uma *divergência* na prática, que deve ser ajustada para aplicação deste ciclo nas organizações.

g. Descrição do procedimentos

Descrição formal das partes do novo sistema que serão Manuais, bem como uma descrição de como os clientes ou usuários realmente interagirão com a parte automatizada do novo sistema. São as descrições das atividades operacionais do cliente ou usuário que, normalmente, correspondem ao Manual do Usuário. Também chamada de Projeto de Implantação.

h. Conversão de banco (base) de dados

Num projeto de conversão de Banco de Dados, este pode envolver mais trabalho e planejamento do que desenvolvimento de programas para o novo sistema e, em outros casos, pode não haver uma *Base de Dados* para ser convertida. De um modo geral, a entrada desta atividade é o Banco de Da-

dos Atual do Cliente ou usuário, bem como as especificações de projeto. Esta etapa existe sempre que houver uma migração de ambientes operacionais. Também chamada de Projeto de Implantação.

i. Instalação

Na atividade final, suas entradas são o manual do usuário, o banco de dados convertido e o sistema de aceite. Em alguns casos, no entanto, a instalação pode simplesmente significar uma passagem, da noite para o dia, de um novo sistema e, em outros casos, pode representar um processo gradual, à medida que grupos de clientes ou usuários recebam os Manuais de Usuários, *Hardware* e treinamento para a utilização do novo sistema. Também chamada de Projeto de Implantação.

j. Conclusão

Um ciclo de vida de sistema existe para cada projeto, independentemente da metodologia utilizada. Ainda, todo ciclo deve ser ajustado e aperfeiçoado na medida de sua utilização. No entanto, uma *metodologia* que tenha por base o ciclo de vida deve ser a mais abrangente possível. Também deve contemplar tanto um sistema complexo desenvolvido internamente, um simples sistema ou quanto à aquisição de um pacote de *software*.

Por meio do uso de *matrizes de atividades*, para cada fase de desenvolvimento de sistemas, relacionam-se às pessoas responsáveis em cada fase ou atividade (engenheiros de *software*, analista de sistemas, clientes ou usuários, analistas de banco de dados, administradores de dados, gerenciadores de sistemas ou gestores etc.) e que nível de participação é esperado de cada um (por exemplo: opcional ou obrigatória).

Na prática, o fundamental de fato é a existência de uma metodologia para desenvolvimento sistema e de *software*, onde todos os técnicos possam usar.

Relação com a ISO 9001 e 9000-3

Muitas organizações se utilizam deste ciclo de vida, ajustado, para elaborarem sua certificação ISO 9001 e 9000-3, pois o item *Sistemas da Qualidade – Atividades do Ciclo de Vida* relata as seguintes atividades que devem ser elaboradas: generalidades; análise critica do contrato; especificações dos requisitos do comprador; planejamento do desenvolvimento; planejamento da qualidade; projeto e implementação; ensaios e validação; aceitação; cópia, entrega e instalação; manutenção.

Os detalhes deste tema serão discutidos no capítulo sobre *Qualidade, produtividade e efetividade* em Informática, sistemas e software.

CICLO DE MANUTENÇÃO DE SOFTWARE E SISTEMAS

De maneira geral, todo *software* sofre manutenções, sejam elas para simples ajustes pós-implantação, ou por melhorias substanciais, por força da legislação e, finalmente, por estar gerando erros. A manutenção de *software* é caracterizada como um *iceberg*, ou seja, esperamos que apenas a parte visível esteja carente de manutenção, porém, normalmente uma massa enorme de *problemas e custos* esconde-se sob a superfície (PRESSMAN, 1995).

A manutenção de *software* existente pode ser responsável por mais de *70% (ou mais)* de todo o esforço despendido por uma organização (ou departamento) de *software*, elevando-se à medida que mais *software* é produzido. Neste sentido, uma organização de *software 'baseada na manutenção'* não pode mais produzir novo *software*, porque está gastando todos os seus esforços e recursos disponíveis mantendo um *software* antigo.

As mudanças são inevitáveis quando se constroem sistemas baseados em computador e para pessoas, portanto deve-se *desenvolver mecanismos e processos* para avaliar, controlar e fazer modificações, visando facilitar a acomodação de mudanças e reduzir a quantidade de esforço despendido em manutenção.

Tipos de manutenção de software

O *Ciclo de Manutenção* contempla as 3 formas mais utilizadas de interferência em software e sistemas:

a. Manutenção por legislação

Quando o sistema ou software tem que ser alterado para atender aspectos legais para cumprimento de leis.

b. Manutenção por melhoria ou implementação

Quando o sistema ou software sofre ajustes a fim de otimizar processos, agregar valores, melhorar desempenho, incluir novos requisitos funcionais

etc. Alguns autores não consideram esta forma de manutenção, caracterizando esta atividade como desenvolvimento de projeto.

c. Manutenção por correções de erros

Quando o sistema ou software necessita de alterações para eliminar erros que estão acontecendo. Esta modalidade é considerada problemática e pouco tolerada, uma vez que nenhum sistema ou software deveria estar em funcionamento com erros, que provavelmente foram deixados passar nas fases de testes e de avaliação da qualidade mal elaborada.

Independente das 3 formas de manutenção de software, outras atividades estão presentes nessas manutenções:

a. Manutenção corretiva

Porque a atividade de testes não descobriu todos os erros latentes num grande sistema de software, sendo que o processo inclui diagnóstico e correção de um ou mais erros.

b. Manutenção adaptativa

Por causa das rápidas mudanças e da evolução do hardware, novos sistemas operacionais, novas linguagens ou tecnologias e atualização de periféricos, sendo que o software é modificado para atender ao ambiente mutante.

c. Manutenção perfectiva

À medida que um software bem-sucedido é usado, as recomendações de novas capacidades, de modificações em funções existentes e de ampliações gerais, sendo que o software procura satisfazer ainda mais o cliente ou usuário.

d. Manutenção preventiva ou preditiva

Ocorre quando o software é modificado para melhorar a confiabilidade ou a manutenibilidade futura, ou para oferecer uma base melhor para futuras ampliações. Pode se manifestar quando da certeza de alteração em legislação, pedido antecipado de clientes ou outra possibilidade de antever transformações no software. Também pode ser chamada de engenharia reversa ou reengenharia.

Além dessas atividades, existem outras características da manutenção de software. Esta é a fase mais negligenciada da engenharia de software, rela-

tivamente pouca pesquisa ou dados de produção têm sido compilados sobre o assunto, e poucas *abordagens ou métodos* técnicos tem sido propostos (PRESSMAN, 1995):

a. Manutenção estruturada

Quando existe uma configuração de software completa, a manutenção é iniciada pela avaliação da documentação do projeto, seguindo-se das características estruturais, de desempenho de interface. As modificações, seus impactos, as correções exigidas são avaliadas, planejadas numa abordagem completa e conseqüentemente com revisão. Utiliza-se um roteiro ou uma Especificação de Testes.

b. Custos tangíveis de manutenção

Variam de organização para organização, girando entre 20% até 80% do orçamento de software.

c. Custos intangíveis de manutenção

Oportunidade de desenvolvimento postergada ou perdida, geralmente porque os recursos disponíveis estão canalizados na manutenção. Insatisfação do cliente ou usuário quando solicitações aparentemente legítimas de reparo ou modificações não podem ser encaminhadas oportunamente quanto ao tempo. Redução da qualidade global do software como resultado de mudanças que introduzem erros latentes no software mantido. Sublevações (revoltas) causadas durante esforços de desenvolvimento quando o pessoal precisa ser 'empurrado' para trabalhar numa tarefa de manutenção. Questões comportamentais etc.

Engenharia reversa e reengenharia de software

A *Engenharia reversa* tem como princípio a *desmontagem* das caixas pretas do *software*, de seus segredos, de trás para frente, ou seja, o processo de recuperação do projeto com especificação e documentação procedimental, arquitetural e de dados (PRESSMAN, 1995).

A engenharia reversa também pode ser elaborada na criação de novos sistemas a partir de sistemas antigos, quando normaliza todos os seus depósitos de dados (HEUSER, 2000). A *reengenharia,* também chamada de *renovação* ou *recuperação,* não somente recupera informações de projeto de um *software* existente, mas usa estas informações para alterar ou reconsti-

tuir o sistema existente, num esforço para melhorar sua qualidade global, *reimplementando* a função do sistema, adicionando novas funções ou melhora de desempenho global.

Controles de versão e auditoria em software

Combinam procedimentos e ferramentas para gerenciar diferentes versões de programas (fonte e objetos), de configurações que são criadas durante o processo de engenharia do *software* (desenvolvimento ou manutenção). Podem permitir que o usuário ou cliente especifique configurações alternativas do sistema de *software* por meio da escolha de versões apropriadas. As mudanças descontroladas em *software* geralmente levam ao caos ou à crise de *software*.

O controle de mudanças combina procedimentos humanos e ferramentas automatizadas para proporcionar um mecanismo eficiente, sendo elaborado um *check-in* completo, contemplando (PRESSMAN, 1995): necessidade de mudança reconhecida; pedido do cliente ou usuário e avaliação do desenvolvedor; documentação de acompanhamento; definição de prioridade; testes e auditoria; inclusão e disponibilização de nova versão.

Algumas organizações elaboram esta atividade no próprio *rosto dos programas*, seguido por uma documentação paralela e complementar.

Além do controle de versões, também devem ser trabalhadas as atividades de auditoria de manutenção e revisões técnicas formais nos sistemas. A identificação, o controle de versão e de mudanças e a revisão técnica formal, incluindo fundamentalmente os testes e averiguações, podem ser elaborados pelas áreas ou atividades de *auditoria*.

Atividade formal de garantia de qualidade de *software* executada por profissionais de *Engenharia de Software*, geralmente em um grupo de trabalho, com usuários ou clientes. Seus principais objetivos são: antecipar o descobrimento de erros de função, lógica, implementação etc.; atendimento aos requisitos e padrões especificados; desenvolvimento uniforme, padronizado e metodológico; tornar projetos administráveis; treinamento da equipe e dos novos integrantes.

REUSABILIDADE DE SOFTWARE

É uma característica importante de um componente de *software* de alta qualidade, ou seja, o componente deve ser projetado e implantado de *forma que possa ser usado em muitos programas diferentes* (PRESSMAN, 1995).

Na década de 1960, construíamos *bibliotecas de sub-rotinas* científicas que eram reusáveis num amplo conjunto de aplicações científicas e de engenharia. Essas bibliotecas de sub-rotinas reusavam algoritmos bem definidos efetivamente, mas tinham um domínio de aplicação limitado.

Atualmente, aplica-se a visão do reuso a fim de envolver não somente algoritmos, mas também estruturas de dados. Um componente reusável da década de 90 engloba tanto dados como processamento num único *pacote (às vezes chamado classe ou objeto)*, possibilitando que engenheiro de *software* crie novas aplicações a partir de partes reusáveis. Por exemplo, as interfaces interativas de hoje freqüentemente são construídas utilizando-se componentes reusáveis que possibilitam a criação de janelas gráficas, menus *pull-down* (navegável) e uma ampla variedade de mecanismos de interação.

Estruturas reusáveis

As estruturas de dados e detalhes de processamento exigidos para se construir a interface com os clientes ou usuários estão contidas numa biblioteca de componentes reusáveis para construção de interfaces.

Os componentes de *software* são construídos usando uma linguagem de programação que tem um vocabulário limitado, uma gramática explicitamente definida e regras de sintaxe e semântica bem formadas. Esses atributos são essenciais para a tradução por máquina.

As formas de linguagem em uso são linguagens de máquina, linguagem de alto nível e linguagens não procedimentais. Não obstante centenas de *linguagens de programação estejam em uso atualmente*, pouco mais do que 10 linguagens de programação de alto nível são amplamente usadas na indústria. Linguagens tais como COBOL e FORTRAN continuam tendo um uso generalizado quase 30 anos depois de sua introdução. Linguagens de programação modernas (linguagens que apóiam diretamente práticas de projeto modernas para projeto procedimental e de dados), tais como Pascal, C e Ada, estão sendo amplamente usadas. Linguagens orientadas a objetos, tais como C++, Object Pascal e outras, estão conquistando entusiásticos seguidores.

Linguagens especializadas (projetadas para domínios de aplicação específicos), tais como APL, LISP, OPS5, Prolog e linguagem descritiva para redes neurais artificiais estão conquistando maior aceitação à medida que novas abordagens de aplicação saem do laboratório para o uso prático. As linguagens de máquinas, as linguagens montadoras (*assembly*) e as lingua-

gens de programação de alto nível freqüentemente são citadas como "as três primeiras gerações" das linguagens de computador. Como todas essas linguagens, o programador deve preocupar-se tanto com a especificação da estrutura de informações como com o controle do programa em si. Daí as linguagens das três primeiras gerações serem denominadas *linguagens procedimentais*.

No decorrer da última década, um grupo de *linguagem de quarta geração*, ou não-procedimentais, foi introduzido. Em vez de exigir que o desenvolvedor de *software* especifique detalhes de procedimento, a linguagem não-procedimental subentende um programa "especificando o resultado desejado, em vez de especificar a ação exigida para se conseguir esse resultado".

O *software* de apoio *converte a especificação do resultado num programa executável* em máquina. Até hoje, as linguagens de quarta geração têm sido usadas em aplicações de bancos de dados e em outras áreas de processamento de dados comerciais.

Software reusáveis

Todo e qualquer processo, módulo, objeto, ou seja, partes de software, podem ser reusáveis várias vezes e em vários locais do software. Como exemplo, podem-se citar as atividades de calcular dígito verificador ou de validar datas, rotinas pré-montadas ou esqueletos de programas, tais como, inclusão, alteração, consulta, exclusão e impressão de dados.

A reusabilidade tem como objetivos principais a qualidade, a produtividade e a efetividade no desenvolvimento e manutenção de software. Na medida em que os engenheiros de software utilizam partes de software já avaliados, bem como de partes de softwares já prontos, não será necessário sua digitação, codificação e verificação de erros. Desta forma o tempo de trabalho será reduzido e conseqüentemente a qualidade já garantida.