Capítulo 1

Introdução

A **engenharia de software** é a parte da ciência da computação que tem por objetivo definir metodologias que permitam às pessoas projetar, construir e manter sistemas de software de larga escala. Neste capítulo vamos fazer uma introdução a engenharia de software e cobrir os seguintes tópicos:

- Definir o que é engenharia de software e qual sua importância;
- Discutir os conceitos de produto de software e processo de software;
- Explicar a importância da visibilidade no processo de software;
- Introduzir a noção de responsabilidade profissional.

1.1 O mundo da engenharia de software

O mundo atual é totalmente dependente de sistemas de software em todos os ramos da sociedade. O funcionamento efetivo da economia e da política modernas estão intimamente ligados a nossa capacidade, enquanto desenvolvedores, de criar softwares de qualidade.

Nesta pragmática, a engenharia de software é a área da ciência da computação encarregada em modelar o mundo real em softwares. Diferente de outras áreas da engenharia, a engenharia de software não se baseia em ciências exatas pois, o mundo real está longe de ser modelado matematicamente. Portanto a engenharia de software é a ciência que tenta *modelar o imodelável*.

O conceito de engenharia de software surgiu em meados da década de 1960 com o advento da terceira geração de computadores[1], quando as máquinas passaram a ser várias ordens de grandeza mais avançadas que as máquinas da segunda geração. Este avanço fez com que as pessoas percebessem que, a partir daí, desenvolver programas passava a ser

[1] A terceira geração de computadores caracteriza-se pelo advento dos computadores baseados em transistores. Veja a bibliografia da disciplina de Organização de Comutadores.

algo mais do que simplesmente deduzir fórmulas: envolvia interagir com pessoas e com requisitos multidimensionais.

Foi então que surgiram, de forma absolutamente empírica, uma série infindada de metodologias, fórmulas e macetes para se melhor desenvolver um software. Com o tempo estas técnicas passaram a ser agrupadas e classificadas dentro de uma área a qual chamamos de engenharia de software.

Ao longo do tempo a engenharia de software evoluiu bastante e não temos dúvida de que os programas desenvolvidos atualmente são muito melhores e mais complexos do que aqueles feitos no passado graças, muitas vezes, aos avanços nas metodologias de desenvolvimento. Entretanto ainda falta muito a se desenvolver pois, mesmo hoje em dia, o empirismo é inevitável no desenvolvimento de sistemas de software.

Para demonstrar as características inexatas desta disciplina vejamos um exemplo: se tivermos dois programadores desenvolvendo dois sistemas equivalentes, cada um com um ferramental específico, não podemos determinar qual deles irá desenvolver o melhor software, pois esta é uma questão que depende, antes de mais nada da capacidade individual de cada programador. Ou ainda, não é possível provar matematicamente que um software é mais fácil de utilizar do que um outro software equivalente. Considerando que sistemas cuja a operação é mais simplificada têm maior probabilidade de sucesso, esta informação é fundamental, entretanto não podemos obtê-la senão testando os softwares com usuários humanos.

Observamos que existe aí uma situação conflitante entre a necessidade de desenvolver sistemas de lógica exata (softwares) à serem utilizados pelos indivíduos humanos (desprovidos de lógica). E daí nasce todo um conjunto de conflitos e incertezas que a engenharia de software, enquanto *engenharia* e *ciência não exata* precisa lidar.

1.1.1 Custo do software

Gastos com sistemas informatizados representam uma parcela significante das despesas de um país e os gastos com software normalmente são os mais relevantes no desenvolvimento de um sistema computacional. Em um PC, por exemplo, o custo para se desenvolver e/ou manter funcionando um software é muito maior do que o custo do hardware.

Devemos considerar ainda que o custo de manutenção de um software é maior do que o seu custo de desenvolvimento e/ou implantação. Sistemas com longos tempos de vida útil costumam custar várias vezes mais para serem mantidos do que para serem desenvolvidos.

A engenharia de software, como disciplina da engenharia também preocupa-se em desenvolver software à um custo efetivo.

1.2 Produtos de software

O objetivo da engenharia de software é a produção de produtos de software. Um produto de software nada mais é do que um programa computacional – suficientemente efetivo na

solução de um ou mais problemas determinados e acessível ao usuário a que se destina – acompanhado da respectiva documentação e, evetualmente, de um hardware específico.

Produtos de software dividem-se em dois tipos:

- **Produtos genéricos**, ou **software de pacote**. São sistemas desenvolvidos para serem utilizados sem um grande acompanhamento de seus desenvolvedores. São produzidos por organizações específicas, como as softwarehouses, e vendidos a qualquer usuário.
- **Produtos personalizados** ou **sob-medida**. São sistemas comissionados para um cliente específico e especialmente desenvolvido por uma equipe dedicada a aquele cliente.

Até a década de 1970 a grande maioria dos produtos softwares eram personalizados, pois os computadores eram muito caros sendo utilizados somente por grandes organizações, cada uma com as suas características. Desenvolver software naquela época também era um processo muito demorado. Entretanto com o advento dos computadores pessoais na década de 1980 passou-se a investir mais em soluções genéricas que atendessem ao maior número de possíveis usuários.

Os produtos personalizados também são muito mais caros pois um único usuário precisa arcar com o custo de todo o desenvolvimento, já os softwares genéricos são mais baratos pois os custos são diluídos por todos os consumidores do mesmo produto.

Os problemas que cercam o desenvolvimento de softwares genéricos e personalizados são de mesma natureza. A principal diferença é que em um software personalizado o desenvolvedor analisa os requisitos do sistema entrevistando o usuário final, enquanto no software de prateleira, é necessário prever que tipo de software será efetivo no mercado.

1.2.1 Atributos dos produtos de software

São atributos desejáveis em produtos de software.

- **Manutenibilidade.** Deve ser possível que o software evolua para atender a mudanças nas necessidades do cliente, com o passar do tempo;
- **Dependabilidade.** O software não deve causar danos físicos ou econômicos na ocorrência de falhas;
- **Eficiência.** O software não deve desperdiçar recursos;
- **Usabilidade.** O software deve ter uma interface e uma documentação apropriadas ao seu uso.

A importância relativa destas características são dependentes de cada produto e do ambiente no qual este é utilizado. Em alguns casos, alguns atributos devem predominar, por

exemplo, em sistemas cuja a segurança seja crítica a dependabilidade e a eficiência devem predominar.

Os custos de produção do software tendem a aumentar drasticamente quando um determinado atributo é muito solicitado.

1.3 O processo do software

O processo do software é um conjunto de atividades estruturadas requeridas para se desenvolver um sistema de software. Existem várias formas de se desenvolver um produto de software, entretanto quatro fazes são fundamentais:

- **Especificação.** Determinar o conjunto completo de requisitos do sistema de forma que se saiba exatamente o que se quer desenvolver;
- **Projeto (Design).** Dada a especificação, projetar significa desenhar e implementar o sistema de software.
- **Validação.** Nesta etapa deve-se cumprir uma bateria de testes a todos os aspectos do sistema a fim de assegurar ao máximo a sua qualidade.
- **Evolução.** Depois de pronto e em operação um produto de software pode requerer alterações decorrentes das evoluções do ambiente em que este se encontra.

As atividades variam dependendo de cada organização e de cada sistema a se desenvolvido. O processo do software deve ser explicitamente modelado para que se possa gerenciá-lo.

1.3.1 Características do processo do software

Podemos avaliar um processo de software pelas suas características (abstratas). São algumas delas:

- **Entendibilidade.** O processo do software é bem definido e bem inteligível?
- **Visibilidade.** O progresso do processo de software pode ser acompanhado e compreendido externamente?
- **Suportabilidade.** Pode o processo do software ser desenvolvido utilizando-se uma ferramenta CASE?
- **Aceitabilidade.** O processo é bem aceito por aquelas pessoas envolvidas com ele?
- **Credibilidade.** Os erros no processo são descobertos antes que produzam falhas no produto e conseqüente danos a sistemas físicos e econômicos?
- **Robustez.** Pode o processo do software prosseguir em caso de problemas inesperados?

- **Mantenibilidade.** Pode o processo do software evoluir atendendo a mudanças nas necessidades de uma organização?
- **Rapidez.** Quão rápido é o processo de desenvolvimento do software?

1.3.2 Modelos de processos de software

Os processos de desenvolvimento de projetos de engenharia em geral seguem um modelo baseado em algumas fases. São elas:

- **Especificar.** Definir os requisitos e as restrições do sistema;
- **Projetar.** Colocar no papel o modelo do sistema;
- **Manufaturar.** Construir o sistema;
- **Testar.** Verificar se o sistema atende aos requisitos;
- **Instalar.** Entregar o sistema para o cliente e assegurar-se que este é operacional;
- **Manter.** Reparar falhar no sistema a medida que estas são descobertas.

Esta metodologia genérica para projetos de engenharia em geral também foi aplicada na construção de sistemas de software, entretanto, algumas deficiências deste modelo foram encontradas quando da sua aplicação no mundo do software:

- Normalmente, especificações de requisitos são incompletas e/ou anômalas.
- A distinção entre especificação, projeto e manufatura não é clara.
- Software não é algo tangível o que significa que a manutenção de um software não é feita através da troca de componentes.

Assim, tornaram-se necessários modelos específicos para o desenvolvimento de software. Existe uma quantidade enorme de modelos de processos entretanto podemos agrupa-los em quatro categorias:

- **Modelo em cascata** (*Waterfall Model*). Modelos baseados em fases separadas e distintas de especificação e desenvolvimento.
- **Desenvolvimento evolucionário.** Modelos onde a especificação e o desenvolvimento são integrados.
- **Transformação formal.** Modelos baseados em sistemas matemáticos que são formalmente convertidos em uma implementação.
- **Desenvolvimento baseado em reutilização – COTS** (*Components Off-The-Shelf).* Modelos onde o sistema é montado à partir de componentes já existentes.

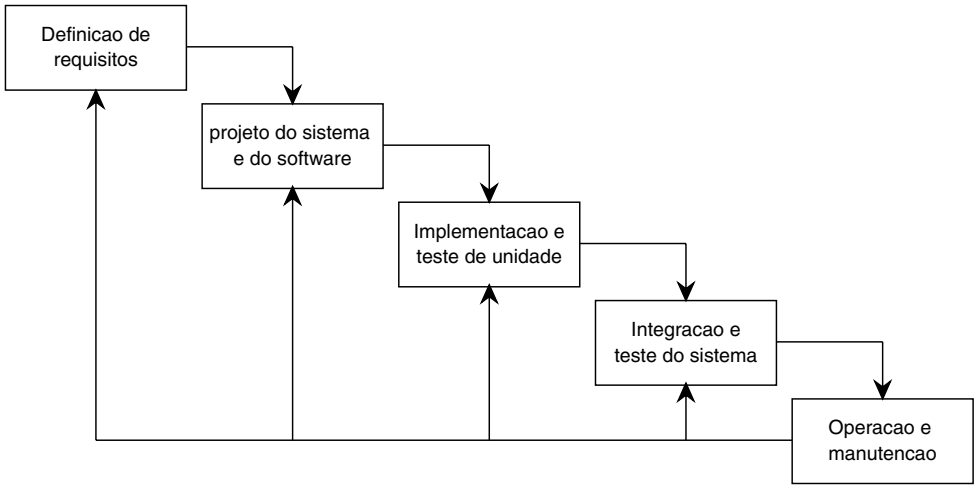

Figura 1.1: Modelo em Cascata.

Modelo em cascata

As metodologias utilizadas em processos de software baseadas no modelo em cascata são oriundas diretas da metodologia tradicional de projetos de engenharia e dividem as etapas do processo em fases bem definidas e o mais desacopladas quanto possível, e que podem ser realizadas em paralelo ou em seqüência.

As fases do modelo em cascata podem variar em número e escopo de uma metodologia para a outra, mas podemos reconhecer pelo menos cinco *grandes* fases principais: a análise e definição de requisitos, o projeto do sistema e do software, a implementação e o teste de unidades, a integração e o teste global do sistema, e a operação e manutenção do sistema.

A grande desvantagem do modelo em cascata é a dificuldade de realizar mudanças no sistema que cresce à medida que o processo se desenrola. Veja na Figura 1.1 um diagrama genérico deste modelo.

Desenvolvimento evolucionário

O desenvolvimento evolucionário baseia-se na unificação das fases de especificação, desenvolvimento e validação do sistema e um único processo isolado. Assim, em um processo que utiliza este modelo, os desenvolvedores recebem uma descrição preliminar do problema a ser desenvolvido e elaboram, o mais rapidamente possível, um sistema parcialmente funcional, ou seja, um **protótipo**, que é implantado no ambiente do usuário.

O protótipo é utilizado para que, observando o seu uso possam ser refinados os requisitos do sistema e levando a criação de um novo protótipo. Este processo se repete até que se consiga uma versão final do sistema atendendo plenamente os requisitos do sistema. A Figura 1.2 mostra como funciona o modelo de desenvolvimento evolucionário.

Existem duas formas de se trabalhar com prototipação:

- **Prototipação exploratória.** Nesta o objetivo é trabalhar com o cliente e fazer

evoluir o sistema final à partir de um protótipo inicial. Este tipo de metodologia só deve ser utilizada quando já se tem uma boa idéia de como é a estrutura geral do sistema.

- **Prototipação descartável (*throw-away*).** Tem por objetivo compreender melhor um sistema. Neste modelo os protótipos são descartados e reconstruídos em cada nova versão e pode ser feita mesmo que, ou especialmente quando, não se tem uma boa idéia dos requisitos do sistema.

O desenvolvimento exploratório, baseado em prototipação, possui algumas desvantagens como:

- **Perda de visibilidade do processo.** A cada nova versão os documentos são reescritos e é difícil perceber em que pé está o projeto pois é sempre difícil determinar *quantos* protótipos ainda precisam ser desenvolvidos para que o projeto seja completado.
- **A pobre estruturação dos sistema.** A cada versão a estrutura do sistema é mexida o que gera o aumento da complexidade de sua lógica interna fazendo com que eventuais falhas sejam mais difíceis de serem previstas.

Este modelo também exige que se disponha de desenvolvedores virtuosos , o que é uma qualidade difícil de mensurar, e especializados em ferramentas de rápida prototipação. Entretanto a prototipação possui aplicabilidade e determinados tipos de sistemas:

- Em sistemas interativos de pequena e média escala;
- Partes isoladas de grandes sistemas (por exemplo, interfaces com o usuário);
- Para sistemas com curto tempo de vida;

1.4 Gerenciamento do risco do processo

Embora os quatro modelos genéricos vistos sintetizem bem todas as variantes de modelos de desenvolvimento de softwares, nenhum deles trata o problema do risco do projeto, ou seja, qual o nível de certeza que se tem de que um projeto vai obter êxito na construção de um sistema efetivo e eficiente dentro das especificações, do prazo e do custo definidos. Em outras palavras, o *risco* inerente de uma atividade é a medida de incerteza do sucesso desta atividade[2].

Risco de um projeto só pode ser minimizado com um bom gerenciamento. Atividades de alto risco causam atrazos de cronograma e aumento de custo de um projeto. O risco está relacionado com a quantidade e a qualidade das informações sobre a atividade desenvolvida. Quanto menos se sabe de um processo, maior o risco de que ele não obtenha êxito.

Os modelos vistos têm diferentes tendências de riscos como vemos:

[2]Não confunda **risco** com **falta de segurança**.

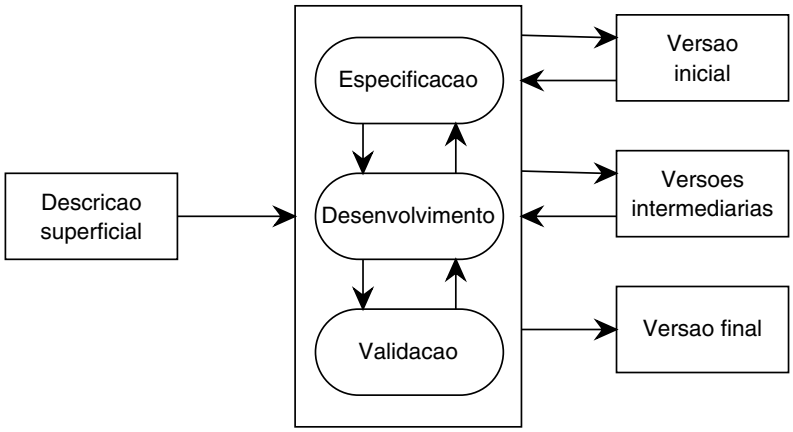

Figura 1.2: Desenvolvimento evolucionário.

- **Modelo em cascata.** Alto risco quando se está desenvolvendo um sistema inédito devido aos problemas de especificação e projeto. Baixo risco em projetos bem compreendidos e que se utilizam de tecnologias estabelecidas.
- **Prototipação.** Tem um risco menor em projetos inéditos pois podem-se identificar os erros de especificação utilizando protótipos. Por outro lado possui um alto risco decorrente da baixa visibilidade.
- **Transformação formal.** Alto risco pois requer um grande controle da tecnologia e requer desenvolvedores altamente capacitados.

A fim de minimizar os riscos é proposta a utilização de modelos híbridos de processos de software. Sistemas de larga escala são feitos de diversos subsistemas e o mesmo processo não precisa ser utilizado em todos estes subsistemas. Por exemplo, pode-se utilizar a prototipação para subsistemas de difícil especificação e o modelo em cascata para subsistemas bem conhecidos.

1.5 Visibilidade do processo

Sistemas de software são entidades intangíveis e assim os gerenciadores dos projetos precisam de documentação para saber como vai o projeto. Entretanto esta necessidade pode causar alguns problemas:

- Se perde tempo documentando um sistema de software, tempo que pode ser precioso;
- A necessidade de documentar o sistema limita a iteração do processo;
- O tempo gasto em revisar e aprovar os documentos é significativo;

Vejamos na Tabela 1.1 alguns documentos que podem ser gerados nas diversas atividades do desenvolvimento em cascata.

Atividade	Documentos gerados
Análise de Requisitos	Estudo de possibilidades, Requisitos gerais
Definição de requisitos	Documentação dos requisitos
Especificação do sistema	Especificação funcional, Plano de aceitação
Projeto arquitetural	Especificação arquitetural, Plano de teste do sistema
Projeto de interface	Especificação das interfaces, Plano de integração
Projeto detalhado	Especificação do projeto, Plano de teste de unidades
Codificação	Código fonte
Teste de unidade	Relatório do teste de unidade
Teste de módulos	Relatório do teste dos módulos
Teste de integração	Relatório do teste de integração, Manual do usuário
Teste do sistema	Relatório do teste do sistema
Teste de aceitação	O sistema pronto e sua respectiva documentação

Tabela 1.1: Documentos do modelo em cascata.

1.6 Responsabilidade profissional

Assim como em outros ramos da engenharia, aqueles profissionais que exercem a função de engenheiro de software não devem apenas preocupar-se com considerações técnicas. Os sistemas de software regulam o mundo moderno e portanto este indivíduo tem uma vasta gama de responsabilidades éticas, sociais e profissionais.

As principais questões éticas que este profissional deve se preocupar são:

- **Confidencialidade.** Engenheiros de software e profissionais de informática em geral devem manter sigilo sobre assuntos que digam respeito a um determinado cliente, mesmo que nenhum termo explícito tenha sido assinado.

- **Competência.** O profissional não deve se candidatar a executar projetos de software que estejam além de suas capacidades e competências.

- **Direito de propriedade intelectual.** Como softwares são simples agregados de lógica, ou seja, produtos intelectuais, estes podem ser facilmente replicáveis desrespeitando-se o direito de propriedade intelectual daqueles que desenvolveram este produto, o que em nosso país é muito corriqueiro. O engenheiro de software deve refletir sobre estas questões e calcular o custo ético, social e econômico deste tipo de ação.

- **Mal uso do computador.** Computadores são sistemas fracamente seguros quando caem na mão de desenvolvedores inteligentes, entretanto eles guardam hoje informações relativas a todas as atividades do ser humano, conseqüentemente podemos concluir que a própria ordem social depende de uma postura profissional do pessoal da área de informática em não utilizar o seu conhecimento de informática em detrimento do próximo.