

Manual do Engenheiro de Software

Módulo Técnico

*Wilson de Pádua Paula Filho
Janeiro de 2000*

Página em branco

Sumário

Sumário	3
Fundamentos técnicos	9
Engenharia de Software	11
1 Natureza	11
1.1 Engenharia de Software e Ciência da Computação	11
1.2 Sistemas de informática	12
2 Produtos.....	13
2.1 Problemas	13
2.2 Produção	14
2.3 Requisitos	15
2.4 Prazos e custos	18
2.5 Qualidade.....	19
Processos	25
1 Visão geral.....	25
1.1 Processos em geral	25
1.2 Processos de software	25
2 Exemplos de processos.....	29
2.1 O Processo Pessoal de Software.....	29
2.2 O Processo de Software para Times	31
2.3 O Processo Orientado a Objetos para Software Extensível	33
2.4 O Processo Unificado	34
3 Praxis.....	35
3.1 Visão geral	35
3.2 Detalhes das fases.....	43
3.3 Artefatos	54
3.4 Procedimentos de controle	57
Métodos técnicos	59
Requisitos	61
1 Princípios.....	61
1.1 Visão geral	61
1.2 A Especificação dos Requisitos do Software	62
1.3 Qualidade dos requisitos	64
2 Atividades.....	67
2.1 Visão geral	67
2.2 Detalhes das atividades	69
3 Técnicas.....	89
3.1 Introdução	89
3.2 Protótipos	89
3.3 Oficinas de requisitos	91
3.4 Relacionamento com os clientes.....	95
Análise.....	99
1 Princípios.....	99
1.1 Objetivos.....	99
1.2 Objetos e classes.....	99
1.3 Uso de notações alternativas	101
2 Atividades.....	102
2.1 Visão geral	102
2.2 Detalhes das atividades	103

3	Técnicas.....	127
3.1	Introdução.....	127
3.2	Oficinas de análise.....	127
3.3	Documentação do Modelo de Análise.....	129
Desenho	131
1	Princípios.....	131
1.1	Objetivos.....	131
1.2	O modelo de desenho.....	131
1.3	Desenho para a testabilidade.....	132
2	Atividades.....	133
2.1	Visão geral.....	133
2.2	Detalhes das atividades.....	134
3	Técnicas.....	150
3.1	Visão geral.....	150
3.2	Desenho de interfaces de usuário.....	150
3.3	Desenho para a reutilização.....	153
3.4	Interfaces com bancos de dados relacionais.....	155
Testes	161
1	Princípios.....	161
1.1	Visão geral.....	161
1.2	Objetivos.....	161
1.3	Métodos.....	161
1.4	Baterias de testes.....	162
2	Atividades.....	163
2.1	Visão geral.....	163
2.2	Detalhes das atividades.....	166
3	Técnicas.....	174
3.1	Visão geral.....	174
3.2	Testes de integração.....	174
3.3	Testes de aceitação.....	177
3.4	Testes de regressão.....	180
Implementação	181
1	Princípios.....	181
1.1	Objetivos.....	181
1.2	Artefatos.....	181
2	Atividades.....	184
2.1	Visão geral.....	184
2.2	Detalhes das atividades.....	185
3	Técnicas.....	194
3.1	Introdução.....	194
3.2	Modularização.....	195
3.3	Documentação de usuário.....	200
Padrões técnicos	205
Proposta de Especificação de Software	207
1	Introdução.....	207
2	Preenchimento da Proposta de Especificação de Software.....	207
2.1	Missão do produto (PESw-1).....	207
2.2	Lista de funções (PESw-2).....	207
2.3	Requisitos de qualidade (PESw-3).....	208
2.4	Metas gerenciais (PESw-4).....	209
2.5	Outros aspectos (PESw-5).....	209
2.6	Estimativa de custos e prazos para a especificação (PESw-6).....	209
Especificação de Requisitos de Software	211
1	Visão geral.....	211
1.1	Introdução.....	211
1.2	Referências.....	211
1.3	Convenções de preenchimento.....	211

1.4	Organização dos requisitos específicos.....	212
2	Preenchimento da Especificação dos Requisitos do Software	213
2.1	Introdução (ERSw-1).....	213
2.2	Descrição geral do produto (ERSw-2).....	216
2.3	Requisitos específicos (ERSw-3)	223
2.4	Informação de suporte (ERSw-4).....	230
3	Revisão da Especificação dos Requisitos do Software.....	230
3.1	Introdução.....	230
3.2	Revisão da seção <u>Página de Título</u>	231
3.3	Revisão da seção <u>Sumário</u>	231
3.4	Revisão da seção <u>Lista de Ilustrações</u>	231
3.5	Revisão do corpo da Especificação dos Requisitos do Software.....	231
3.6	Revisão do Modelo de Análise.....	236
	Revisões de Software	239
1	Visão geral.....	239
1.1	Objetivos.....	239
1.2	Alternativas.....	239
2	Reuniões de revisão.....	240
2.1	Participantes.....	240
2.2	Preparação.....	240
2.3	Condução.....	240
3	Perfil da equipe de revisão.....	241
3.1	Introdução.....	241
3.2	Perfil do líder	241.....

2.2	Desenho das interfaces de usuário (DDSw-2)	275
2.3	Desenho interno (DDSw-3)	286
2.4	Desenho das liberações (DDSw-4)	292
2.5	Informação de suporte (DDSw-5)	293
3	Revisão da Descrição do Desenho do Software	297
3.1	Introdução	297
3.2	Revisão da seção <u>Página de Título</u>	297
3.3	Revisão da seção <u>Sumário</u>	298
3.4	Revisão da seção <u>Lista de Ilustrações</u>	298
3.5	Revisão do corpo da Descrição do Desenho do Software	298
Documentação de Testes de Software		307
1	Visão geral	307
2	Descrição dos Testes do Software	307
2.1	Introdução	307
2.2	Referências	307
2.3	Estrutura	307
2.4	Preenchimento da Descrição dos Testes do Software	308
2.5	Revisão da Descrição dos Testes do Software	317
3	Relatórios dos Testes do Software	321
3.1	Introdução	321
3.2	Registros de testes	322
3.3	Relatórios de incidentes dos testes	324
3.4	Relatórios de resumo de testes	324
Desenho Detalhado e Codificação de Software		327
1	Visão geral	327
1.1	Introdução	327
2	Diretrizes genéricas	327
2.1	Aspectos abordados	327
2.2	Dados	327
2.3	Estruturas de controle	333
2.4	Expressões	338
2.5	Leiaute de programa	338
2.6	Comentários	342
3	Diretrizes para C++	345
3.1	Diretrizes aplicáveis a C	345
3.2	Transição de C a C++	347
3.3	Gestão de memória	349
3.4	Desenho e declaração de classes e funções	351
3.5	Herança	354
3.6	Compilação	359
3.7	Tópicos avançados	359
4	Revisões da Implementação	360
4.1	Lista de conferência para o desenho detalhado	360
4.2	Lista de conferência para código	362
Documentação para Usuários de Software		369
1	Visão geral	369
1.1	Introdução	369
1.2	Referências	369
1.3	Padronização de formato	369
1.4	Requisitos de apresentação	369
2	Preenchimento do Manual do Usuário do Software	370
2.1	Página de título	370
2.2	Garantias e obrigações contratuais	370
2.3	Sumário	370
2.4	Índice de ilustrações	371
2.5	Introdução	371
2.6	Corpo do documento	371
2.7	Mensagens de erro, problemas conhecidos e recuperação de erros	373
2.8	Apêndices	373

2.9	Bibliografia	373
2.10	Glossário	373
2.11	Índice remissivo	373
<i>A notação UML</i>		375
1	Introdução	375
2	Modelagem funcional	375
2.1	Atores	375
2.2	Casos de uso	376
2.3	Diagramas de casos de uso	376
2.4	Fluxos dos casos de uso	380
2.5	Diagramas de estados	381
2.6	Diagramas de atividade	383
3	Modelagem lógica	386
3.1	Objetos e classes	386
3.2	Organização das classes	388
3.3	Relacionamentos	390
3.4	Detalhes das classes	394
3.5	Diagramas de interação	397
4	Modelagem física	404
4.1	Visão de componentes	404
4.2	Visão de desdobramento	406
<i>Glossário</i>		409
<i>Bibliografia</i>		419

Página em branco

Fundamentos técnicos

Página em branco

Engenharia de Software

1 Natureza

1.1 Engenharia de Software e Ciência da Computação

O que é Engenharia de Software? É uma das disciplinas da Informática, ou da Ciência da Computação? É um sinônimo de um destes termos? Ou, falando de modo mais prático: um profissional formado em Informática ou Ciência da Computação é automaticamente um engenheiro de software?

O Dicionário Aurélio Eletrônico V.2.0 assim define:

Informática	Ciência que visa ao tratamento da informação através do uso de equipamentos e procedimentos da área de processamento de dados.
Ciência	Conjunto organizado de conhecimentos relativos a um determinado objeto, especialmente os obtidos mediante a observação, a experiência dos fatos e um método próprio.
Processamento de dados	Tratamento dos dados por meio de máquinas, com o fim de obter resultados da informação representada pelos dados.
Engenharia	Arte de aplicar conhecimentos científicos e empíricos e certas habilitações específicas à criação de estruturas, dispositivos e processos que se utilizam para converter recursos naturais em formas adequadas ao atendimento das necessidades humanas.

Tabela 1 – Informática, ciência e engenharia

Nos verbetes acima, fica a Informática definida como uma ciência, cujo assunto é o processamento de informação através de máquinas. A ciência, por sua vez, tem como foco a acumulação do conhecimento através do método científico, geralmente baseado em experimentos e observações.

A definição de Engenharia é conexa, porém distinta. Analisemos cada uma de suas partes, tentando interpretá-las em termos da Engenharia de Software e reordenando-as para fins explicativos.

- **Arte** – Na acepção aqui usada, a “capacidade que tem o homem de pôr em prática uma idéia, valendo-se da faculdade de dominar a matéria”, ou “a utilização de tal capacidade, com vistas a um resultado que pode ser obtido por meios diferentes”. O produto da engenharia é matéria dominada: idéia que se torna material através do emprego das faculdades humanas. Na Engenharia de Software, a matéria dominada consiste em máquinas de processamento da informação configuradas e programadas.
- **Atendimento das necessidades humanas** – O foco da engenharia é a necessidade humana. Nisto, ela tem escopo bem diverso da ciência. O conhecimento é certamente uma necessidade humana, mas uma entre várias outras de uma hierarquia¹: alimentação, moradia, segurança, afeição, auto-estima... Todo produto de engenharia se justifica através da satisfação de uma dessas necessidades; portanto, da geração de algo que tenha valor para alguém. A Engenharia de Software procura gerar valor através dos recursos de processamento de informação.

¹ Os especialistas em desenvolvimento humano usam a escala de necessidades de Maslow [Hitt85].

- **Conhecimentos científicos** – Parte dos métodos da engenharia provém da ciência; parte dos métodos da Engenharia de Software provém da Ciência da Computação.
- **Conhecimentos empíricos** – Outra parte dos métodos da engenharia provém da experiência prática, e não apenas da pesquisa científica. Em muitos casos, a ciência intervém posteriormente para explicar, modelar e generalizar o que a prática descobriu. Na Engenharia de Software, muitas práticas são adotadas porque funcionam, mesmo quando ainda carecem de fundamentação teórica satisfatória.
- **Habilitações específicas** – Toda engenharia é uma atividade realizada por pessoas. Para isto, estas pessoas têm de ter habilitações específicas. A Engenharia de Software possui um conjunto de habilitações específicas, ou disciplinas, que se relaciona com o conjunto das disciplinas da Ciência da Computação, mas não se confunde com elas.
- **Recursos naturais** – Toda engenharia parte de recursos naturais; algumas ciências, por contraste, como a Lógica e a Matemática, têm base inteiramente abstrata. Os recursos naturais da Engenharia de Software são as máquinas de tratamento da informação. A Ciência da Computação se ocupa de abstrações como os algoritmos e as estruturas de dados; a Engenharia de Software usa estas abstrações, desde que sejam realizáveis na prática, através da tecnologia existente em determinado momento.
- **Formas adequadas** – Para satisfazer as necessidades humanas, os recursos naturais devem ser convertidos em formas adequadas. Na Engenharia de Software, estas formas são os programas de computador. Comparado com outros engenheiros, o engenheiro de software tem liberdade extrema na criação de formas. Entretanto, só uma ínfima fração das formas possíveis atende ao critério de utilidade.
- **Dispositivos e estruturas** – O engenheiro reúne dispositivos em estruturas capazes de satisfazer uma necessidade humana. A criação de estruturas é essencial para que se extraia uma função útil do conjunto de dispositivos. O desafio do engenheiro de software é escolher e montar as estruturas de grande complexidade que a programação dos computadores permite realizar.
- **Processos** – A engenharia segue processos, que são “maneiras pelas quais se realiza uma operação, segundo determinadas normas”. O método da engenharia se baseia na ação sistemática, e não na improvisação. A noção de processo será também a espinha dorsal deste livro.

Em suma, a Engenharia de Software não se confunde com a Ciência da Computação, e nem é uma disciplina desta, tal como a Engenharia Metalúrgica não é uma disciplina da Física dos Metais, nem a Engenharia Elétrica é uma disciplina da Física da Eletricidade. Como toda engenharia, a Engenharia de Software usa resultados da ciência, e fornece problemas para estudo desta; mas são vocações profissionais completamente distintas, tão distintas quanto as vocações do engenheiro e do físico, do médico e do biólogo, do político e do cientista político.

1.2 Sistemas de informática

As máquinas de tratamento de informação são organizadas em estruturas úteis, formando os sistemas de informática. Várias definições de sistema são aqui pertinentes.

1. Conjunto de elementos, materiais ou ideais, entre os quais se possa encontrar ou definir alguma relação.
2. Disposição das partes ou dos elementos de um todo, coordenados entre si, e que funcionam como estrutura organizada.

3. Reunião de elementos naturais da mesma espécie que constituem um conjunto intimamente relacionado.

O software é a parte programável de um sistema de informática. Ele é um elemento central: realiza estruturas complexas e flexíveis que trazem funções, utilidade e valor ao sistema. Mas outros componentes são indispensáveis: as plataformas de hardware, os recursos de comunicação de informação, os documentos de diversas naturezas, as bases de dados e até os procedimentos manuais que se integram aos automatizados.

Este livro trata apenas dos componentes de software, por limitação de escopo. O engenheiro de software deverá ter em mente, no entanto, que o valor de um sistema depende da qualidade de cada um de seus componentes. Um sistema pode ter excelentes algoritmos codificados em seu software e ser de péssimo desempenho por defeito de desenho de seu hardware, rede ou banco de dados. Cada um destes elementos pode pôr a perder a confiabilidade e a usabilidade do sistema.

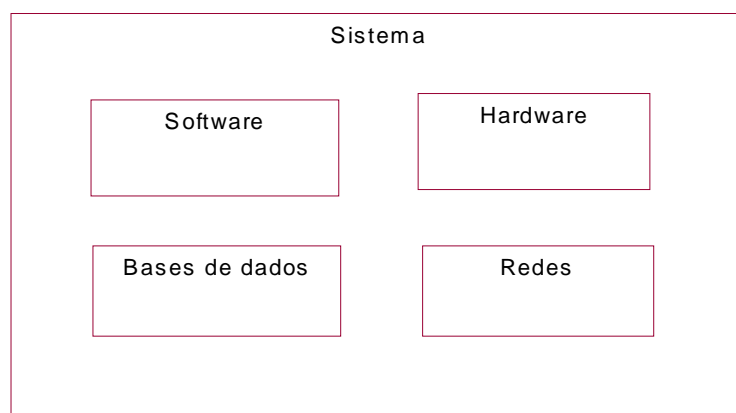


Figura 1 - Sistema de informática e suas partes

Na prática, o engenheiro de software será chamado com frequência a resolver questões pertinentes aos outros componentes do sistema, ou, no mínimo, encontrar quem as resolva. Alguma proficiência nas respectivas disciplinas lhe será necessária. Não trataremos delas neste livro, a não ser tangencialmente, quando necessário.

2 Produtos

2.1 Problemas

Muitas pessoas, inclusive dirigentes de empresa, percebem o computador como problema, e não como solução. Muitos aceitam como fato da vida que os sistemas de informática:

- não façam o que deveriam fazer;
- sejam caros;
- sejam entregues tarde demais;
- sejam de baixa qualidade:
 - sejam cheios de defeitos;
 - sejam difíceis de usar;

- sejam lentos etc.

A tecnologia só resolve problemas quando é usada por pessoas qualificadas, dentro de processos adequados. Os sistemas de informática são os produtos da tecnologia de tratamento da informação. Os problemas que ocorrem com sistemas de informática podem ter várias causas:

- Podem ser fruto de deficiência de qualificação das pessoas que os operam. Isto pode decorrer de falta de treinamento, dificuldade de uso do próprio sistema, ou muitos outros fatores relacionados com pessoas.
- Podem originar-se de processos de negócio inadequados. Por **processo de negócio** entendemos o processo que faz parte da área de aplicação, em que, tipicamente, alguns procedimentos são executados por pessoas e outros são automatizados através do computador. Por exemplo, sacar dinheiro de um banco pode ser feito por dois processos diferentes: diretamente no caixa, ou através do equipamento conhecido como caixa eletrônico. O segundo processo é mais automatizado que o primeiro.
- Podem também ser causados por deficiências de tecnologia, ou seja, do próprio sistema de informática. Neste livro, trataremos apenas dessa classe de problemas.

2.2 Produção

2.2.1 Ciclos de vida

A Engenharia de Software se preocupa com o software como produto. Estão fora de seu escopo programas que são feitos unicamente para diversão do programador. Estão fora de seu escopo também pequenos programas descartáveis, feitos por alguém exclusivamente como meio para resolver um problema, e que não serão utilizados por outros.

Chamaremos de **cliente** a uma pessoa física ou jurídica que contrata a execução de um projeto, ou a um seu representante autorizado, com poder de aceitação de propostas e produtos. A pessoa que efetivamente usará um produto será chamada de **usuário**. Um usuário pode ser o próprio cliente, um funcionário de uma organização cliente, ou mesmo não ser relacionado diretamente com o cliente. Por exemplo, quando se produz software de prateleira, que será vendido no mercado aberto, é útil considerar como cliente, por exemplo, o departamento de marketing da organização produtora.

Como todo produto industrial, o software tem um ciclo de vida:

- ele é concebido a partir da percepção de uma necessidade;
- desenvolvido, transformando-se em um conjunto de itens entregue a um cliente;
- entra em operação, sendo usado dentro de um algum processo de negócio, e sujeito a atividades de manutenção, quando necessário;
- é retirado de operação, ao final de sua vida útil.

Cada fase do ciclo de vida tem divisões e subdivisões, que serão exploradas ao longo deste livro. É interessante observar, na Tabela 1, que a Codificação, que representa a escrita final de um programa em forma inteligível para um computador, é apenas uma pequena parte do ciclo de vida. Para a maioria das pessoas, inclusive muitos profissionais da informática, esta parece ser a única tarefa de um programador, ou seja, um produtor de software.

Ciclo de vida	Percepção da necessidade			
	Desenvolvimento	Concepção		
		Elaboração		
		Construção	Desenho inicial	
			Liberação	Desenho detalhado
				Codificação
				Testes de unidade
			Testes alfa	
		Transição		
	Operação			
Retirada				

Tabela 2 - Esquema simplificado do ciclo de vida do software

2.2.2 Projetos

Normalmente, o desenvolvimento de software é feito dentro de um **projeto**. Todo projeto tem uma data de início, uma data de fim, uma equipe (da qual faz parte um responsável, a que chamaremos de **gerente do projeto**) e outros recursos. Um projeto representa a execução de um **processo**.

Quando um processo é bem definido, ele terá subdivisões que permitam avaliar o progresso de um projeto e corrigir seus rumos quando ocorrerem problemas. Essas subdivisões são chamadas de fases, atividades ou iterações; posteriormente, usaremos estas palavras com significados técnicos específicos.

As subdivisões devem ser terminadas por **marcos**, isto é, pontos que representam estados significativos do projeto. Geralmente os marcos são associados a **resultados** concretos: documentos, modelos ou porções do produto, que podem fazer parte do conjunto prometido aos clientes, ou ter apenas utilização interna ao projeto. O próprio produto é um resultado associado ao marco de conclusão do projeto.

2.3 Requisitos

2.3.1 Características

O valor de um produto vem de suas **características**. Tratando-se de software, costuma-se dividir as características em:

- características **funcionais**, que representam os comportamentos que um programa ou sistema deve apresentar diante de certas ações de seus usuários;
- características **não funcionais**, que quantificam determinados aspectos do comportamento.

Por exemplo, em um terminal de caixa automático, os tipos de transações bancárias suportadas são características funcionais. A facilidade de uso, o tempo de resposta e o tempo médio entre falhas são características não-funcionais.

Os **requisitos** são as características que definem os critérios de aceitação de um produto. A engenharia tem por objetivo colocar nos produtos as características que são requisitos. Outras características podem aparecer acidentalmente, mas os produtos não devem ser desenhados para incluí-las, já que, normalmente, toda característica extra significa um custo adicional de desenho ou de fabricação.

2.3.2 *Especificação dos requisitos*

Os requisitos podem ser dos seguintes tipos:

- Os requisitos **explícitos** são aqueles descritos em um documento que arrola os requisitos de um produto, ou seja, um documento de **especificação de requisitos**.
- Os requisitos **normativos** são aqueles que decorrem de leis, regulamentos, padrões e outros tipos de normas a que o tipo de produto deve obedecer.
- Os requisitos **implícitos** são expectativas dos clientes e usuários, que são cobradas por estes, embora não documentadas.

Requisitos implícitos são indesejáveis porque, não sendo documentados, provavelmente não serão considerados no desenho do produto. O resultado será um produto que, embora satisfazendo os compromissos formais, que são os requisitos explícitos e normativos, não atenderá às necessidades do consumidor.

Mesmo requisitos documentados podem apresentar problemas. Uma especificação de requisitos pode conter requisitos incompletos, inconsistentes ou ambíguos. Alguns desses problemas decorrem da natureza da própria linguagem natural, que normalmente é usada para expressá-los. Outros decorrem de técnicas deficientes de elaboração dos requisitos.

2.3.3 *Engenharia dos requisitos*

Um dos problemas básicos da engenharia de software é o levantamento e documentação dos requisitos dos produtos de software. Quando este levantamento é bem feito, os requisitos implícitos são minimizados. Quando a documentação é bem feita, os requisitos documentados têm maiores chances de serem corretamente entendidos pelos desenvolvedores. Algumas técnicas de análise dos requisitos ajudam a produzir especificações mais precisas e inteligíveis. O conjunto das técnicas de levantamento, documentação e análise forma a **engenharia dos requisitos**, que é uma das disciplinas da Engenharia de Software.

Infelizmente, muitos clientes não entendem a necessidade de especificações de requisitos. Pior ainda, muitos desenvolvedores de software e, pior de tudo, muitos gerentes também não. É uma situação tão absurda quanto querer resolver um problema sem escrever o respectivo enunciado: existe grande risco de resolver-se o problema errado. Por outro lado, é possível também a existência de requisitos que não correspondam a necessidades reais dos clientes e usuários. Essa falha de engenharia de requisitos indica que não foi feita uma análise do valor de cada requisito, do ponto de vista da missão que o produto deve cumprir.

Cabe aos engenheiros de software insistirem sempre na elaboração de uma boa especificação de requisitos. Faz parte do seu trabalho convencer clientes e usuários de que:

- boas especificações de requisitos são indispensáveis;
- elas não representam custos supérfluos, mas investimentos necessários, que se pagam com altos juros;
- a participação dos usuários na engenharia de requisitos é fundamental para que as suas necessidades sejam corretamente atendidas pelo produto;
- uma boa especificação de requisitos custa tempo e dinheiro;
- a ausência de uma boa especificação de requisitos custa muito mais tempo e dinheiro.

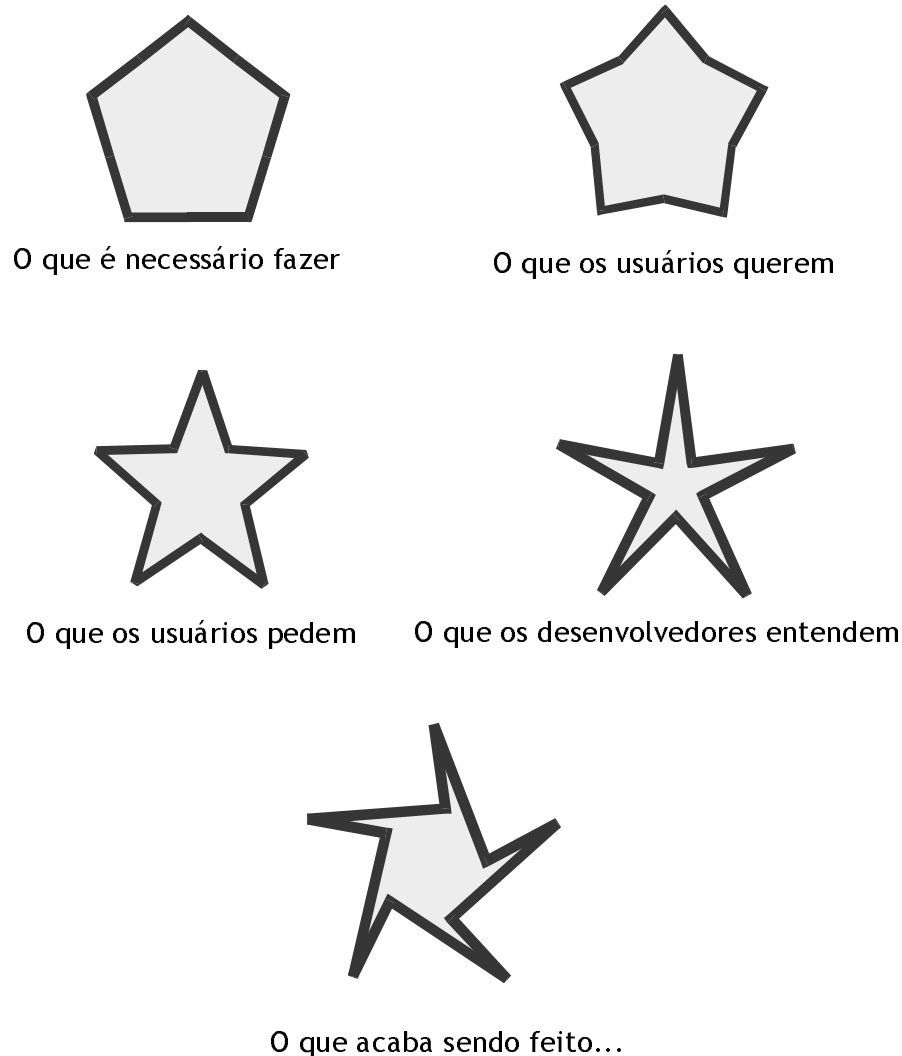


Figura 2 - Como os requisitos evoluem

2.3.4 Gestão dos requisitos

Um problema comum no desenvolvimento de software é a **instabilidade dos requisitos**, que ocorre quando clientes e usuários trazem novos requisitos, ou alterações de requisitos, quando o desenvolvimento já está em fase adiantada. A instabilidade dos requisitos costuma ter um custo muito alto; geralmente significa perder trabalho já feito, desfazer algumas coisas e remendar outras. Na Engenharia de Software, a instabilidade dos requisitos é tão danosa quanto nas outras engenharias. Quando se muda a planta de um edifício durante a construção, geralmente é preciso desfazer parte do que já foi construído, e o remendo raramente é satisfatório.

A boa engenharia de requisitos reduz a instabilidade destes, ajudando a obter os requisitos corretos em um estágio anterior ao desenvolvimento. Entretanto, alterações dos requisitos são às vezes inevitáveis. A engenharia de requisitos é sujeita a limitações humanas, e, mesmo que o levantamento seja perfeito, podem ocorrer alterações de requisitos por causas externas aos projetos. Por exemplo, a legislação pode mudar no meio do projeto, requerendo alterações nos relatórios que o produto deve emitir. A **gestão dos requisitos** é a disciplina da engenharia de software que procura manter sob controle o conjunto dos requisitos de um produto, mesmo diante de algumas alterações inevitáveis.

2.4 Prazos e custos

2.4.1 *Realismo de prazos e custos*

Por que tantos sistemas informatizados são entregues com atraso e custam mais do que o previsto? Estourar cronogramas e orçamentos é parte da rotina da maioria dos profissionais de software. Clientes e gerentes se desesperam com os atrasos dos projetos de software, e às vezes sofrem enormes prejuízos com eles. Entretanto, no contrato seguinte, eles provavelmente escolherão o ofertante que prometer menor prazo e/ou menor custo. Se for um projeto interno da organização, farão todo tipo de pressão para conseguir que os desenvolvedores prometam prazos politicamente agradáveis, embora irreais.

Estimar prazos e custos faz parte da rotina de qualquer ramo da engenharia. Para um produto ser viável, não basta que atenda aos requisitos desejados; tem de ser produzido dentro de certos parâmetros de prazo e custo. Se isso não for possível, o produto pode não ser viável do ponto de vista de mercado, ou pode ser preferível adquirir outro produto, ainda que sacrificando alguns dos requisitos. Ter estimativas de prazos e custos, portanto, é uma expectativa mais que razoável de clientes e gerentes.

O problema é que existem alguns desenvolvedores pouco escrupulosos. E existem muitos que, mesmo sendo honestos, não conhecem métodos técnicos de estimativa de prazos e custos do desenvolvimento de software. E existem ainda os que, mesmo sabendo fazer boas estimativas, trabalham em organizações onde não existe clima para que os desenvolvedores possam apresentar avaliações francas das perspectivas dos projetos. Nestas organizações, existe a política de "matar os mensageiros de más notícias". Esta política foi usada por muitos reis da antiguidade, com resultados geralmente desastrosos.

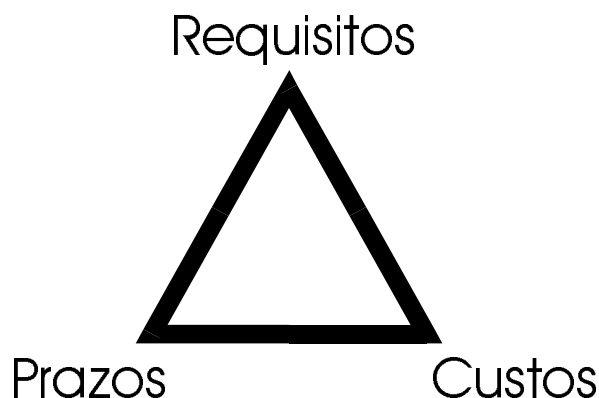


Figura 3 - Um triângulo crítico da Engenharia de Software

Requisitos, prazos e custos formam os vértices de um triângulo crítico. Aumentos de requisitos levam a aumentos de prazos ou de custos, ou de ambos. Reduções de requisitos podem levar a reduções de prazos ou de custos (mas nem sempre).

2.4.2 *Planejamento de projetos*

Uma coisa é exigir dos engenheiros de software estimativas de prazos e cobrar o cumprimento dos prazos prometidos. Clientes e gerentes podem e devem fazê-lo. Outra coisa é pressioná-los para que façam promessas que não podem ser cumpridas. Uma frase comum dessa cultura é: "Não me interessa como você vai fazer, desde que entregue no prazo!". Na realidade, o cliente ou gerente deve, no seu próprio interesse, ter algum meio de checar se o cronograma e o orçamento propostos são realistas; e, se preciso, recorrer aos serviços de uma terceira parte para fazer esta verificação.

A cultura do prazo político é ruim para todos. Para os desenvolvedores, ela significa estresse e má qualidade de vida. Para os gerentes, perda de credibilidade e prejuízos. E, para os clientes, produtos de má qualidade e mais caros do que deveriam. Ainda por cima, entregues fora do prazo.

Para cumprir compromissos de prazos e custos, esses compromissos têm de ser assumidos com base em requisitos bem levantados, analisados e documentados. E os planos dos projetos têm de ser feitos com boas técnicas de estimativa e análise de tamanho, esforços, prazos e riscos. Estas técnicas pertencem à disciplina de **planejamento de projetos**, que faz parte da Engenharia de Software.

2.4.3 *Controle de projetos*

Todo plano comporta incertezas. Por exemplo, o tamanho de certas partes do produto pode ser estimado grosseiramente a partir dos requisitos, mas o desenho detalhado das partes do produto permite refinar as estimativas, e o tamanho correto só é exatamente conhecido no final dos projetos. A produtividade dos desenvolvedores pode ser estimada com base em projetos anteriores da organização, mas é afetada por muitas variações, que dependem de pessoas, processos e tecnologia. E riscos previstos e imprevistos podem se materializar.

Ao longo de um projeto, os gerentes têm de enfrentar problemas e tentar controlar variáveis que afetem o cumprimento de seus compromissos. Algumas vezes, os problemas podem ser resolvidos por meio de contratação e remanejamento de pessoal, ou de uma melhoria de ferramentas. Outras vezes não existe maneira viável de contornar os problemas, e é necessário renegociar requisitos, prazos ou custos. Para renegociar, é preciso replanejar, atualizando as estimativas para levar em conta os novos dados.

A disciplina complementar do planejamento de projetos é o **controle dos projetos**, que compreende:

- o acompanhamento do progresso dos projetos, comparando-se o planejado com o realizado;
- a busca de alternativas para contornar problemas surgidos na execução dos projetos;
- o replanejamento dos projetos, quando não é possível manter os planos anteriores dentro de um grau razoável de variação;
- a renegociação dos compromissos assumidos, envolvendo todas as partes interessadas.

2.5 **Qualidade**

2.5.1 *Conformidade com requisitos*

Entenderemos como **qualidade** de um produto o seu grau de conformidade com os respectivos requisitos. De acordo com essa definição de qualidade, por exemplo, um carro popular pode ser de boa qualidade, e um carro de luxo pode ser de má qualidade. O que decide a qualidade é comparação com os respectivos requisitos: o confronto entre a promessa e a realização de cada produto.

Geralmente a qualidade de um produto decorre diretamente da qualidade do processo utilizado na produção dele. Note-se que importa aqui a qualidade do processo efetivamente utilizado, não do "processo oficial", que pode eventualmente estar descrito nos manuais da organização. Muitas vezes os processos oficiais não são seguidos na prática, por deficiência de ferramentas, por falta de qualificação das pessoas, ou porque pressões de prazo levam os gerentes dos projetos a eliminar etapas relacionadas com controle da qualidade.

Em um produto de software de má qualidade, muitos requisitos não são atendidos completamente. As deficiências de conformidade com os requisitos se manifestam de várias maneiras. Em alguns casos, certas funções não são executadas corretamente sob certas condições, ou para certos valores de entradas. Em outros casos, o produto tem desempenho insuficiente, ou é difícil de usar.

Cada requisito não atendido é um **defeito**. No mundo da informática, criou-se a usança de chamar de "bugs" os defeitos de software. Assim, erros técnicos adquirem conotação menos negativa, que lembra simpáticos insetos de desenho animado. E o nome ajuda a esquecer que esses defeitos foram causados por erro de uma pessoa falível, e que cada defeito tem responsáveis bem precisos.

Note-se que defeitos incluem situações de falta de conformidade com requisitos explícitos, normativos e implícitos. Os defeitos associados a requisitos implícitos são os mais difíceis de tratar. Eles levam a desentendimentos sem solução entre o fornecedor e o cliente do produto. Além disso, como esses requisitos, por definição, não são documentados, é bastante provável que eles não tenham sido levados em conta no desenho do produto, o que tornará a correção dos defeitos particularmente trabalhosa.

2.5.2 *Garantia da qualidade*

Um erro conceitual comum é imaginar que é possível trocar prazo, e talvez custo, por qualidade. Na realidade, é possível, em muitos casos, reduzir prazos e custos através da redução dos requisitos de um produto. A qualidade, por outro lado, é consequência dos processos, das pessoas e da tecnologia. A relação entre a qualidade do produto e cada um desses fatores é complexa. Por isto, é muito mais difícil controlar o grau de qualidade do produto do que controlar os requisitos.

Em todas as fases do desenvolvimento de software, as pessoas introduzem defeitos. Eles decorrem de limitações humanas: erros lógicos, erros de interpretação, desconhecimento de técnicas, falta de atenção, ou falta de motivação. Em todo bom processo, existem atividades de garantia da qualidade, tais como revisões, testes e auditorias. Essas atividades removem parte dos defeitos introduzidos. Quando atividades de controle da qualidade são cortadas, parte dos defeitos deixa de ser removida em um ponto do projeto.

Defeitos que não são removidos precocemente acabam sendo detectados depois. Quanto mais tarde um defeito é corrigido, mais cara é a sua correção, por várias razões que serão discutidas posteriormente. O pior caso acontece quando o defeito chega ao produto final. Neste caso, ele só será removido através de uma operação de **manutenção**. Esta é a forma mais cara de remoção de defeitos. Em certos casos, como acontece em sistemas de missão crítica, defeitos de software podem trazer prejuízos irreparáveis.

A Figura 4 mostra que o tempo de desenvolvimento é geralmente reduzido com o aumento da qualidade do processo. Isso acontece porque um processo melhor é mais eficiente na detecção e eliminação precoce dos defeitos. Em geral, o tempo gasto com a correção precoce é mais do que compensado pela eliminação do tempo que seria gasto com a correção tardia. O prazo aumenta apenas quando se quer reduzir o nível de defeitos do produto final a um parâmetro mais rigoroso em relação ao estado-da-arte. Em certos casos, isso se justifica pelo caráter crítico do sistema: por exemplo, quando defeitos podem colocar pessoas em perigo, ou causar prejuízos materiais vultosos.

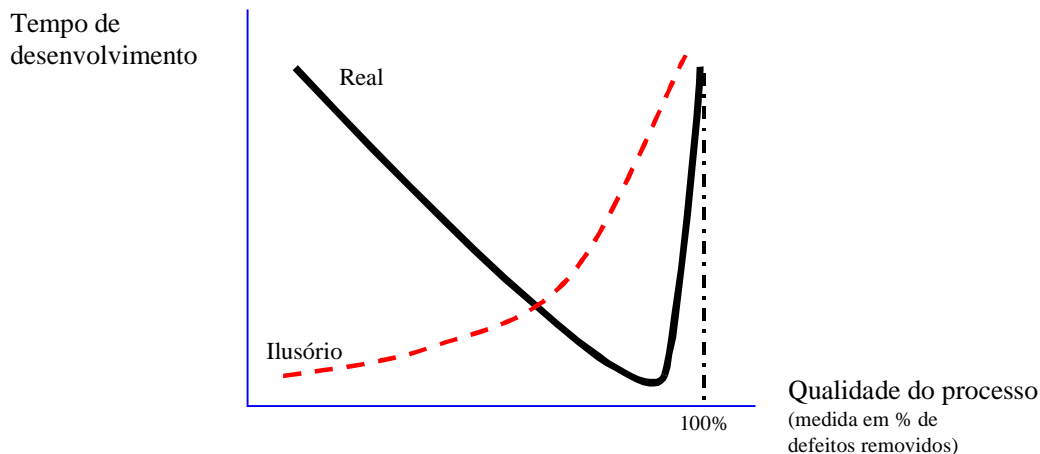


Figura 4 - A curva prazo \times qualidade

Vários métodos de garantia da qualidade levam em conta uma limitação humana: somos mais eficazes para achar os defeitos dos outros do que os nossos próprios defeitos. Por isso, os tipos mais eficazes de revisão são feitos por revisores distintos dos autores. Testes de aceitação de um produto devem ser desenhados e realizados de preferência por testadores independentes. E o controle da qualidade, como

um todo, funciona melhor quando é coordenado por um grupo da organização que é independente dos projetos controlados e que tem acesso direto à alta gerência da organização.

2.5.3 *Gestão de configurações*

Um produto de software é composto por muitos **artefatos**: códigos executáveis, códigos fontes, modelos, relatórios e outros documentos. Alguns desses artefatos são resultados oficiais do projeto; a aprovação dos resultados assinala que um marco do projeto foi cumprido. Outros artefatos têm caráter mais informal; por exemplo, documentos e modelos temporários de trabalho dos desenvolvedores.

A maioria dos artefatos evolui ao longo de um projeto, e até ao longo de toda a vida de um produto. Mesmo depois de terminado um projeto, é importante que os resultados sejam guardados e controlados. Pode ser necessário atualizá-los em caso de manutenção. Documentos e modelos consistentes são indispensáveis para facilitar a manutenção e evitar que esta introduza novos defeitos. A guarda e a atualização de documentos e modelos são rotineiras em todos os ramos da engenharia, e a Engenharia de Software não deveria ser exceção.

Mesmo um pequeno projeto pode gerar mais de uma dezena de resultados diferentes, cada um com mais de uma dezena de versões. Organizar e controlar a proliferação dos artefatos é o objetivo da disciplina de **gestão de configurações**. Sem gestão de configurações é impossível atingir sequer níveis razoáveis de qualidade. Versões corrigidas de artefatos serão perdidas, e versões defeituosas acabarão reaparecendo. Com efeito, o número de itens diferentes produzidos em projetos de software, mesmo pequenos, ultrapassa facilmente os limites da memória e da atenção humanas.

2.5.4 *Gestão de contratos*

Muitas organizações atuais procuram, justificadamente ou não, reduzir sua força de trabalho permanente. Muitas organizações para as quais a produção de software não é uma atividade fim têm preferido contratar externamente o desenvolvimento dos sistemas de que necessitam. E muitos profissionais de informática preferem trabalhar como empresários ou profissionais liberais do que como assalariados. Por causa dessas forças, muitas organizações optam por encomendar a produtores externos o desenvolvimento de seus sistemas informatizados, ou de parte deles.

Terceirizar o desenvolvimento de software é uma boa solução, em muitos casos. Mas não é panacéia. O esforço de uma organização para melhorar a qualidade de seus sistemas informatizados pode ser perdido por causa de falhas dos contratados. Para que isso não aconteça, a organização contratante deve estar capacitada em **gestão de contratos**. Para isso, ela tem de ser capaz, no mínimo, de:

- especificar correta e completamente o produto a ser desenvolvido;
- fazer uma boa seleção entre os candidatos a subcontratado, avaliando o grau de realismo das propostas destes;
- acompanhar o desempenho do subcontratado sem interferir no trabalho destes, mas detectando precocemente sintomas de problemas;
- planejar e executar os procedimentos de aceitação do produto.

2.5.5 *Desenho*

Os defeitos mais grosseiros de um produto de software são os defeitos de requisitos. Felizmente, estes defeitos são relativamente raros, desde que a engenharia de requisitos tenha sido levada a sério tanto por desenvolvedores como por usuários. Defeitos de implementação, por outro lado, são mais comuns. Programadores experientes em uma linguagem de programação, entretanto, não erram com frequência, principalmente quando fazem revisões cuidadosas de seu código.

Entre os requisitos e o código final existe sempre um **desenho**². Ele pode ser explícito, documentado e feito de acordo com determinadas técnicas. Ou pode existir apenas na cabeça do programador, de maneira informal e semiconsciente. Neste último caso, pesam mais as limitações humanas: de raciocínio, de memória e de capacidade de visualização. Por isso, geralmente um desenho de boa qualidade é explícito e documentado.

Os defeitos de desenho geralmente são quase tão graves quanto os de requisitos. Quando os programadores não são competentes em desenho, são quase tão freqüentes quanto os defeitos de implementação. E muitos programadores que têm excelente domínio de uma linguagem de programação nunca tiveram formação em técnicas de desenho. Estas técnicas formam uma das disciplinas mais importantes da Engenharia de Software.

Defeitos de desenho têm geralmente conseqüências graves em todos os ramos da engenharia. Em construções, por exemplo, erros de desenho podem levar a vazamentos, perigo de incêndios, rachaduras e até desabamentos. As conseqüências nos produtos de software são igualmente sérias. Algumas resultados típicos de defeitos de desenho são:

- dificuldade de uso;
- lentidão;
- problemas imprevisíveis e irreprodutíveis;
- perda de dados;
- dificuldade de manutenção;
- dificuldade de adaptação e de expansão.

2.5.6 Modelos de maturidade

A produção industrial de software é quase sempre uma atividade coletiva. Alguns produtos são construídos inicialmente por indivíduos ou por pequenas equipes. Na medida em que se tornam sucesso de mercado, passam a evoluir. A partir daí, um número cada vez maior de pessoas passa a cuidar da sua manutenção e evolução. Por isso, quase todas as atividades de Engenharia de Software são empreendidas por organizações.

A maturidade de uma organização em Engenharia de Software mede o grau de competência, técnica e gerencial, que essa organização possui para produzir software de boa qualidade, dentro de prazos e custos razoáveis e previsíveis. Em organizações com baixa maturidade em software, os processos geralmente são informais. Processos informais são aqueles que existem apenas na cabeça de seus praticantes.

A existência de processos definidos é necessária para a maturidade das organizações produtoras de software. Os processos definidos permitem que a organização tenha um modus operandi padronizado e reproduzível. Isso facilita a capacitação das pessoas e torna o funcionamento da organização menos dependente de determinados indivíduos. Entretanto, não é suficiente que os processos sejam definidos. Processos rigorosamente definidos, mas não alinhados com os objetivos da organização são entaves burocráticos, e não fatores de produção.

Para tornar uma organização mais madura e capacitada, é realmente preciso melhorar a qualidade dos seus processos. Processos não melhoram simplesmente por estarem de acordo com um padrão externo.

² Esta palavra é aqui usada como sinônimo de design, e não na acepção de desenho pictórico (que equivaleria a “drawing” ou “drafting”). Muitas vezes o desenho é chamado de projeto. Usaremos o termo projeto apenas na acepção de unidade gerencial, conforme discutido no capítulo referente à Gestão de Projetos (correspondente a “project”).

O critério de verdadeiro êxito dos processos é a medida de quanto eles contribuem para que os produtos sejam entregues aos clientes e usuários com melhor qualidade, por menor custo e em prazo mais curto.

Diversas organizações do mundo propuseram paradigmas para a melhoria dos processos dos setores produtivos; em particular, algumas desenvolveram paradigmas para a melhoria dos processos de software. Estes paradigmas podem assumir diversas formas. Interessam aqui, especialmente, os paradigmas do tipo **modelos de capacitação**. Um modelo de capacitação serve para avaliar a maturidade dos processos de uma organização. Ele serve de referência para avaliar-se a maturidade destes processos.

Um modelo de capacitação particularmente importante para a área de software é o **CMM** (Capability Maturity Model), do Software Engineering Institute. O CMM é patrocinado pelo Departamento de Defesa americano, que o utiliza para avaliação da capacidade de seus fornecedores de software. Esse modelo teve grande aceitação da indústria americana de software e considerável influência no resto do mundo.

O CMM foi baseado em algumas das idéias mais importantes dos movimentos de qualidade industrial das últimas décadas. Destacam-se entre elas os conceitos de W. E. Deming, que também teve grande influência na filosofia japonesa de qualidade industrial. Esses conceitos foram adaptados para a área de software por Watts Humphrey [Humphrey90]. A primeira versão oficial do CMM foi divulgada no final dos anos 1980, e é descrita em relatórios técnicos do SEI ([Paulk+93], [Paulk+93a]) e um livro ([Paulk+95]).

O CMM focaliza os processos, que considera o fator de produção com maior potencial de melhoria a prazo mais curto. Outros fatores, como tecnologia e pessoas, só são tratados pelo CMM na medida em que interagem com os processos. Para enfatizar que o escopo do CMM se limita aos processos de software, o SEI passou a denominá-lo de SW-CMM, para distingui-lo de outros modelos de capacitação aplicáveis a áreas como desenvolvimento humano, engenharia de sistemas, definição de produtos e aquisição de software. Neste texto, fica entendido que CMM se refere sempre ao SW-CMM. A Tabela 3 resume os níveis do CMM, destacando as características mais marcantes de cada nível.

Número do nível	Nome do nível	Característica da organização	Característica dos processos
Nível 1	Inicial	Não segue rotinas	Processos caóticos
Nível 2	Repetitivo	Segue rotinas	Processos disciplinados
Nível 3	Definido	Escolhe rotinas	Processos padronizados
Nível 4	Gerido	Cria e aperfeiçoa rotinas	Processos previsíveis
Nível 5	Otimizante	Otimiza rotinas	Processos em melhoria contínua

Tabela 3 - Níveis do CMM