



**UNIVERSIDADE DO OESTE DE SANTA CATARINA
CAMPUS – XANXERÊ**

Curso: Tecnologia em Informática

Disciplina: Engenharia de Software

Professor: André Luiz Forchesatto

APOSTILA ENGENHARIA DE SOFTWARE

SUMÁRIO

1. Introdução

A engenharia de software surgiu da necessidade de se construir software com mais qualidade em menor tempo, antigamente se produzia software de uma maneira muito desordenada sem preocupação com o que realmente o software deveria fazer ou se era possível construir um software para executar tal tarefa, com isto surgiu a famosa crise do software que fez com que as empresas ou fábricas de software pensassem em uma maneira de como desenvolver os softwares de maneira confiável e rápida.

O software teve uma grande evolução no decorrer de sua existência ocasionada principalmente pelo barateamento do hardware e a evolução das técnicas de desenvolvimento, como podemos observar no quadro 1 abaixo.

Período	Evolução
1950 – 1960	Orientação a Batch
	Software totalmente customizados
	Distribuição limitada
1960 – 1970	Multiusuários
	Tempo Real
	Banco de Dados
	Produto de Software
1980 – 1990	Sistemas distribuídos
	Inteligência embutida
	Hardware de baixo custo
	Impacto de consumo
1990 – 2000	Sistemas de desktop poderosos
	Tecnologia orientada a objeto
	Sistemas Especialistas
	Redes neurais artificiais
	Computação Paralela

Quadro 1 – Evolução do software(fonte Pressmam, pg.5).

Como podemos observar houve uma grande evolução nas tecnologias de software, hoje podemos dizer que produzimos software muito mais confiável do que no tempo que se trabalhava com sistema em batch, porém alguns problemas que tínhamos naquela época ainda temos hoje, como por exemplo, os problemas de comunicação, e alguns mitos levantados por (Pressman, pg.26) que serão demonstrados agora.

1.1. Mitos e Realidade

Segundo Pressman, existem três grandes mitos no desenvolvimento de software, que surgiram nos primórdios do desenvolvimento de software e continuam até hoje propagando desinformação e confusão.

Mitos Administrativos: Envolve os gerentes que tem como responsabilidade de gerenciar cronogramas e custos.

Mito	Realidade
Se a equipe dispõe de um manual de padrões e procedimentos de desenvolvimento de software, então a equipe está apta a encaminhar bem o desenvolvimento.	É preciso que a equipe aplique efetivamente os conhecimentos apresentados no manual... é necessário que o que conste no dado manual reflita a prática de desenvolvimento de software e que esta prática seja verificada com frequência, para confirmar o uso do conhecimento.
A equipe tem ferramentas de	O fato da empresa ter hardware de última geração não garante

desenvolvimento de software de última geração, uma vez que eles dispõem de computadores de última geração.	que em nada a qualidade do software desenvolvido. Mais importante que ter um hardware de última geração é ter ferramentas para a automatização do desenvolvimento de software. As chamadas ferramentas CASE.
Se o desenvolvimento do software estiver atrasado, basta aumentar a equipe para honrar o prazo de desenvolvimento.	Isto não condiz com a realidade. Quanto mais pessoas pegarem “o barco andando”, pior. Por que os novos profissionais deverão ser treinados e isto será feito por membros da equipe, o que vai implicar em maiores atrasos no cronograma.

Quadro 2 – Mitos Administrativos

Mitos do Cliente: Envolve todos os clientes que podem ser deste de um atendente até o dono da empresa.

Mito	Realidade
Uma descrição breve e geral dos requisitos do software é o suficiente para iniciar o seu projeto... maiores detalhes podem ser definidos posteriormente.	Este é um dos problemas que pode levar um projeto ao fracasso. O cliente deve ser questionado a fim de definir o mais precisamente possível os requisitos importantes para o software: funções, desempenho, interfaces, restrições de projeto e critérios de validação. Estes são pontos fundamentais para o sucesso do projeto.
Os requisitos de projeto mudam continuamente durante o seu desenvolvimento, mas isto não representa um problema, uma vez que o software é flexível e poderá suportar facilmente as alterações.	O software é mais flexível que a maioria dos produtos manufaturados, mas não existe software que suporte uma alteração significativa em seu escopo sem influenciar no custo e no prazo de entrega. O fator de multiplicação dos custos no desenvolvimento de software pode ser observado na figura abaixo.

Quadro 3 – Mitos Cliente

Mitos do Profissional: Envolve os profissionais que desenvolvem software é que vem com uma cultura desde os primórdios da programação.

Mito	Realidade
Após a edição do programa e a sua colocação em funcionamento, o trabalho está terminado.	A realidade é justamente o contrário. Segundo estatísticas, após a implantação ocorrerão 50% a 70% do esforço do desenvolvimento de software (manutenção).
Enquanto o programa não entrar em funcionamento, é impossível avaliar a sua qualidade.	A preocupação com a qualidade do software deve ocorrer durante todo o processo de desenvolvimento, ao final de cada etapa deve ser feita uma verificação dos critérios de qualidade.
O produto a ser entregue no final do projeto é o programa funcionando.	O produto final não é apenas o software funcionando, mas também um conjunto importante de documentos, que acompanham o mesmo.

Quadro 4 – Mitos Profissional

1.2. Perguntas freqüentes sobre engenharia de software

A engenharia de software concentra seus esforços em uma grande gama de assuntos, gerando com isto muitas duvidas, através do livro “Engenharia de Software do Ian Sommerville”, onde o mesmo propõe uma sessão de perguntas e respostas relativas a engenharia de software, será possível esclarecer as duvidas fundamentais da engenharia de software.

Pergunta	Resposta
O que é Software?	São os programas de computador e a documentação associada. Produtos de software podem ser desenvolvidos para um cliente específico ou para o mercado.
O que é engenharia de software?	Engenharia de software é uma disciplina da engenharia que se ocupa de todos os aspectos da produção de software. Pressman. “O estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que seja confiável e que funcione eficientemente em máquina reais.”
Qual é a diferença entre engenharia de software e ciência da computação?	A ciência da computação se ocupa de todos os aspectos relacionados ao desenvolvimento de sistemas com base em computadores, incluindo hardware, software e engenharia de processos. A engenharia de software é parte desse processo.
O que é um processo de software?	É um conjunto de atividades, cuja meta é desenvolvimento ou evolução do software.
O que é um modelo de processo de software?	É uma representação simplificada de um processo de software, apresentada a partir de uma perspectiva específica.
Quais são os custos da engenharia de software?	Aproximadamente 60 por cento dos custos são relacionados ao desenvolvimento e 40 por cento são custos referentes aos testes. Para o software personalizado, os custos de evolução freqüentemente excedem os custos de desenvolvimento.
O que são métodos de engenharia de software?	São abordagens estruturadas para o desenvolvimento de software, que incluem modelos de sistemas, notações, regras, recomendações de projetos e diretrizes de processos.
O que é CASE (computer-aided software engineering – engenharia de software com auxílio de computador)?	São sistemas de software destinados a proporcionar apoio automatizado às atividades de processo de software. Os sistemas CASE são freqüentemente utilizados para proporcionar apoio aos métodos.
Quais são os atributos de um bom software?	O software deve proporcionar ao usuário a funcionalidade e o desempenho requeridos e deve ser passível de manutenção, confiável e de fácil uso.
Quais são os principais desafios enfrentados pela engenharia de software?	Lidar com sistemas legados, atender à crescente diversidade e atender às exigências quanto a prazos de entrega reduzidos.

Quadro 5 – Perguntas freqüentes sobre engenharia

1.3. Paradigmas de Desenvolvimento de Software (Processo)

Segundo Pressman

“Não Existe uma abordagem em particular que seja a melhor para a solução da aflição de software. Entretanto, ao combinarmos métodos abrangentes para todas as fases de desenvolvimento do software, melhores ferramentas para automatizar esses métodos, blocos de construção mas poderosos para a implementação do software, melhores técnicas para garantia da qualidade do software e uma filosofia de coordenação predominante, controle e administração, podemos conseguir uma disciplina para o desenvolvimento do software.”

Como podemos analisar no relato acima Pressman afirma que uma boa combinação de métodos e uma boa disciplina podemos elaborar um projeto de software com qualidade.

Embora existam varias abordagem para o desenvolvimento de software a atividades que são inerentes a todos os processos de desenvolvimento de software como relata (Sommerville, pg. 36):

1. **Especificação de software:** É preciso definir a funcionalidade do software e as restrições em sua operação;
2. **Projeto e implementação de software:** Deve ser produzido o software de modo que cumpra suas especificações. Normalmente está faze é dividida em 3 partes: Projeto de software, codificação e testes;
3. **Validação do software:** O software precisa ser validado para garantir que ele faz o que o cliente deseja;
4. **Evolução de software:** O software precisa evoluir par atender às necessidades mutáveis do cliente.

Com isto foi apresentada uma visão geral do que envolve um processo ou paradigma de desenvolvimento de software, mas existem diferentes abordagens onde as mesmas serão discutidas abaixo.

1.3.1. Modelo Cascata

A figura 1 representa o modelo cascata ou ciclo de vida clássico que segundo (Pressman, pg.32) “[...] o ciclo de vida clássico requer uma abordagem sistemática, seqüencial ao desenvolvimento de software, que se inicia no nível do sistema e avança ao longo da análise, projeto, codificação, teste e manutenção.”. O Modelo Cascata envolve as seguintes atividades:

Planejamento/Engenharia de Sistemas: preparar o plano de desenvolvimento do sistema com base nas conversas com o usuário. Ter uma visão global do sistema, incluindo hardware, software e as pessoas envolvidas.

Análise de Requisitos: definição dos requisitos de software. O resultado será utilizado nas etapas posteriores de Projeto, Construção, Testes e Manutenção. É recomendado o uso de ferramentas CASE.

Projeto: transformar os requisitos da análise, o modelo lógico em modelo físico. Utilizando ferramenta CASE, banco de dados e linguagem de programação.

Construção/Codificação: fazer os programas e relatórios dos sistemas com base nas especificações das fases anteriores.

Teste e Integração: realizar os testes no sistema e realizar as integrações com eventuais sistemas atuais.

Operação e Manutenção: Fazer o acompanhamento do software durante um período. Liberar o uso para o usuário e efetuar alterações no sistema que visem a solução de erros, melhorias ou inclusão de novas funcionalidades.

Após concluída uma etapa ou fase, deve ser feita uma homologação a fim de verificar se os documentos gerados condizem com os requisitos do sistema. Sendo necessário, devem ser refeitas as tarefas com problemas. Uma vez aprovada a fase, ela não deve ser alterada, pois implica em alteração na fase seguinte, e conseqüentemente aumento dos custos.

Problemas: O modelo em cascata possui alguns problemas em virtude dele impor que seja seguida a ordem proposta, muitas vezes isto não é possível, outro problema é que o usuário geralmente não consegue relatar todos seus problemas na fase de análise como o modelo exige, outro problema é que o usuário só ira conseguir visualizar um resultado após um longo tempo de andamento do projeto.

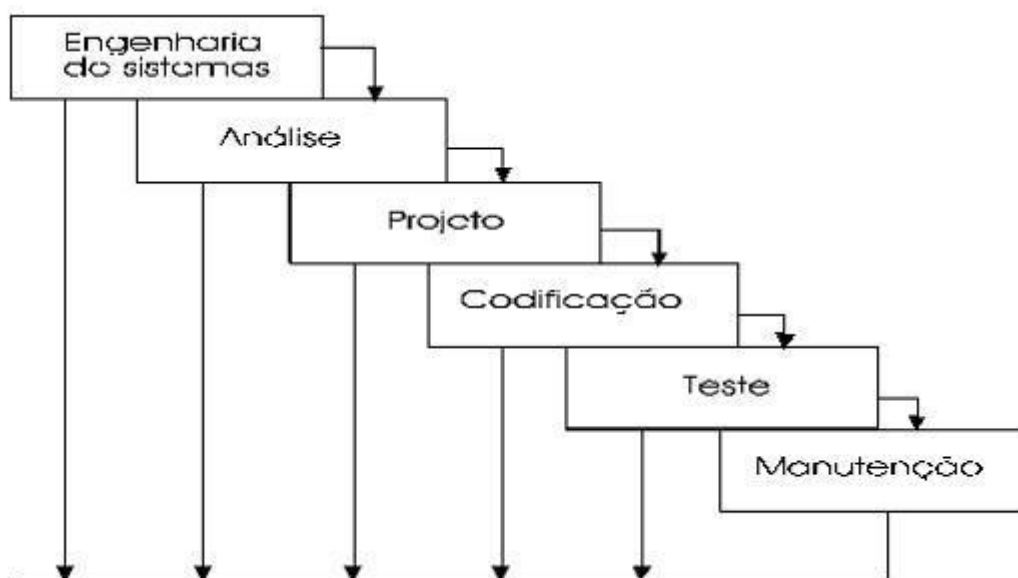


Figura 1 – Ciclo de Vida clássico

1.4. Desenvolvimento Evolucionário

O desenvolvimento evolucionário segundo (Sommerville, pg.39) caracteriza por desenvolver uma implementação inicial e expor o resultado ao comentário do usuário que devera solicitar as alterações e com isto será gerada uma nova implementação. Existem dois tipos de desenvolvimento evolucionário onde os mesmos são descritos abaixo:

Desenvolvimento exploratório: Neste paradigma o cliente e o desenvolver trabalham junto sendo que o processo é fazer o levantamento de requisitos básicos desenvolver um sistema inicial e após isto ir incrementando novas funcionalidades ao sistema.

Desenvolvimento de protótipos descartáveis: O objetivo deste é compreender melhor os requisitos do cliente que estejam mal compreendidos através do desenvolvimento de protótipos que depois serão descartados.

O desenvolvimento evolucionário possui algumas vantagens em relação ao modelo cascata, pois o mesmo possibilita que sejam desenvolvidos gradativamente os requisitos levantados pelo usuário, possibilitando assim uma melhor compreensão sobre o sistema desejado. Na figura 2 pode-se observar como funciona o desenvolvimento evolucionário.

Este modelo proposto também possui algumas desvantagens, que entre elas podemos citar:

- O processo não é visível pelos seus gerentes, pelo fato de parecer que o processo se concentra só em desenvolvimento;
- Os sistemas se tornam mal estruturados pelo fato de serem desenvolvidos como uma cocha de retalho;
- Podem ser exigidas ferramentas e técnicas especiais para desenvolvimento rápido.

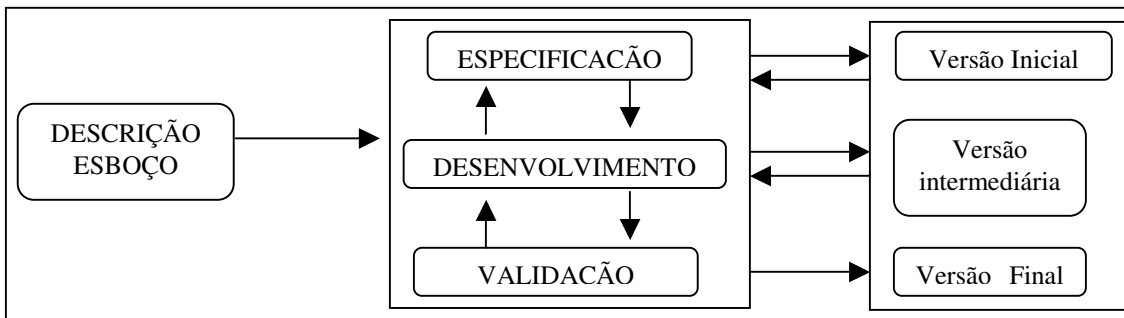


Figura 2 – Modelo de desenvolvimento evolucionário.(fonte sommerville, pg.39).

1.5. Desenvolvimento formal de sistema

O desenvolvimento formal tem como base o modelo cascata, mas a grande característica dele é que o mesmo faz uma transformação matemática formal nos requisitos do software para depois gerar o executável, isto proporciona muito mais segurança no comprimento dos requisitos do cliente só que torna o processo de análise muito mais demorado. Geralmente o modelo de desenvolvimento formal de sistema é aplicado em situações que se exige uma grande segurança no comprimento de requisitos, como em sistema que mexe com a vida humana.

As principais diferenças entre as abordagens modelo cascata e desenvolvimento formal de sistema segundo (Sommerville, pg.40) são:

1. A especificação de requisitos de software é redefinida em uma especificação formal detalhada, que é expressa em uma notação matemática.
2. Os processos de desenvolvimento de projeto, implementação e testes de unidade são substituídos por um processo de desenvolvimento transformacional, em que a especificação é refinada por meio de uma série de transformações, em um programa.

Como podemos observar na citação de Sommerville o modelo é extremamente formal, pois pega os requisitos e transforma em uma especificação formal como demonstrada na figura 3 abaixo.

Mas o modelo possui algumas desvantagens claras que são:

- É necessária uma perícia muito especializada para a transformação dos requisitos levantados pelo cliente em formas normal;
- Não oferece vantagens em relação aos outros modelos nos requisitos custo e qualidade;
- Consome um esforço muito grande em desenvolvimento para a maioria dos sistemas;

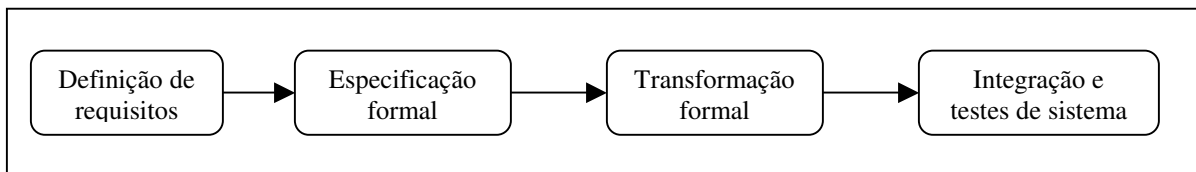


Figura 3 – Desenvolvimento formal de sistemas.(fonte sommerville, pg.40).

1.6. Desenvolvimento orientado a reuso

Pode-se dizer que na maioria dos softwares a reuso, por exemplo, quando desenvolvemos alguma aplicação em delphi estamos reutilizando uma série de bibliotecas que o próprio delphi nos traz isto já é reuso. Só que o desenvolvimento orientado a reuso não é só isto ele se preocupa em muito mais do que usar os componentes existentes, para ele todo componente tem que ser estudado em qual escopo ele pode ser aplicado.

A abordagem de desenvolvimento orientada não se utiliza simplesmente de componentes ela pode-se utilizar de sistema propriamente ditos, como por exemplo os sistemas comerciais de prateleira, como explica (Sommerville, pg.41)

Essa abordagem orientada a reuso conta com uma ampla base de componentes de software reutilizáveis, que podem ser acessados, e com alguma infra-estrutura de integração para esses componentes. Às vezes, esses componentes são sistemas propriamente ditos (os já mencionados COTS, sistemas comerciais de prateleira), que podem ser utilizados para proporcionar funcionalidade específica, como formatação de textos, cálculo numérico, entre outros.

O modelo desenvolvimento orientado a reuso pode ser dividido em:

- **Análise de componentes:** é feita uma busca de componentes para suprir os requisitos;
- **Modificação de requisitos:** Requisitos são analisados utilizando informações sobre os componentes;
- **Projeto de sistema com reuso:** Desenvolvimento da infra-estrutura do sistema utilizando os componentes;
- **Desenvolvimento e integração:** Desenvolver interfaces de comunicação entre os componentes escolhidos para o sistema.

Na figura 4 abaixo podemos observar o modelo orientado a reuso.

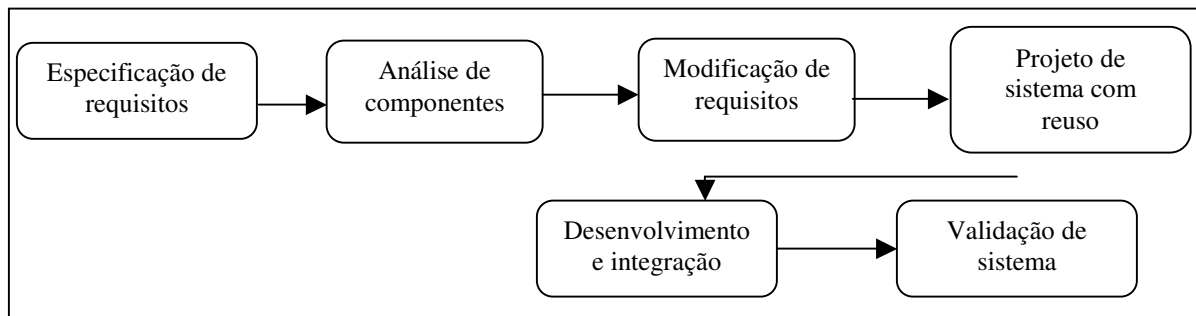


Figura 4 – Desenvolvimento orientado a reuso.(fonte sommerville, pg.42).

1.7. Modelos Híbridos

Os modelos híbridos surgiram da necessidade de se utilizar diferentes tipos de modelos conhecidos em um só sistema, surgiu também em virtude de os modelos já conhecidos serem fracos no que se diz respeito ao desenvolvimento iterativo, ou seja, quando algum pedaço do processo deve ser refeito. Existem dois modelos que surgiram para tentar solucionar estes problemas onde os mesmo são descritos abaixo.

1.7.1. Desenvolvimento Incremental ou iterativo

A idéia principal deste modelo de desenvolvimento é a de que um sistema deve ser desenvolvido de forma incremental, sendo que cada incremento vai adicionando ao sistema novas capacidades funcionais, até a obtenção do sistema final, sendo que, a cada passo realizado, novas modificações podem ser introduzidas. Uma vantagem desta abordagem é a facilidade em testar o sistema, uma vez que a realização de testes em cada nível de desenvolvimento é, sem dúvida, mais fácil do que testar o sistema final. O importante é que mesmo sendo a implementação por funcionalidade, a arquitetura global deve estar pronta antes da construção do primeiro módulo.

Esta abordagem traz várias vantagens ao cliente pois o mesmo pode adiar as decisões sobre os requisitos mais importantes para uma etapa posterior, pois o mesmo estará recebendo varias versões do sistema onde o mesmo poderá ir trabalhando até todo o sistema ficar pronto, abaixo na figura 5 podemos observar como deve funcionar este desenvolvimento.

Contudo este tipo de desenvolvimento também traz desvantagens para os desenvolvedores como, por exemplo, é difícil definir o tamanho exato de cada incremento, torna-se difícil também prever a independência dos módulos do sistema como desenvolver um sistema que não dependa de outras partes que ainda não foram desenvolvidas.

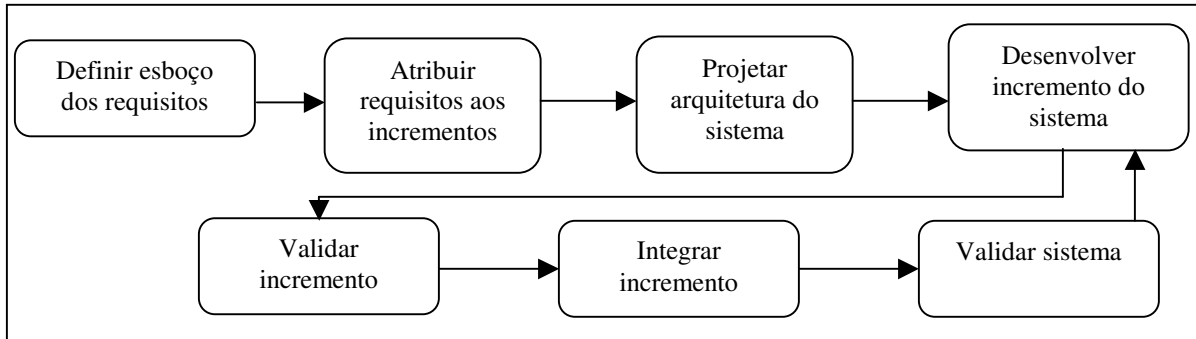


Figura 5- Desenvolvimento incremental (fonte sommerville, pg.42).

1.7.2. Modelo Espiral

O modelo espiral para a engenharia de software foi desenvolvido para abranger as melhores características tanto do ciclo de vida clássico como da prototipação, acrescentando, ao mesmo tempo, um novo elemento – a análise de riscos – que falta a esses paradigmas. O modelo define quatro atividades representadas pelos quatro quadrantes da figura 7:

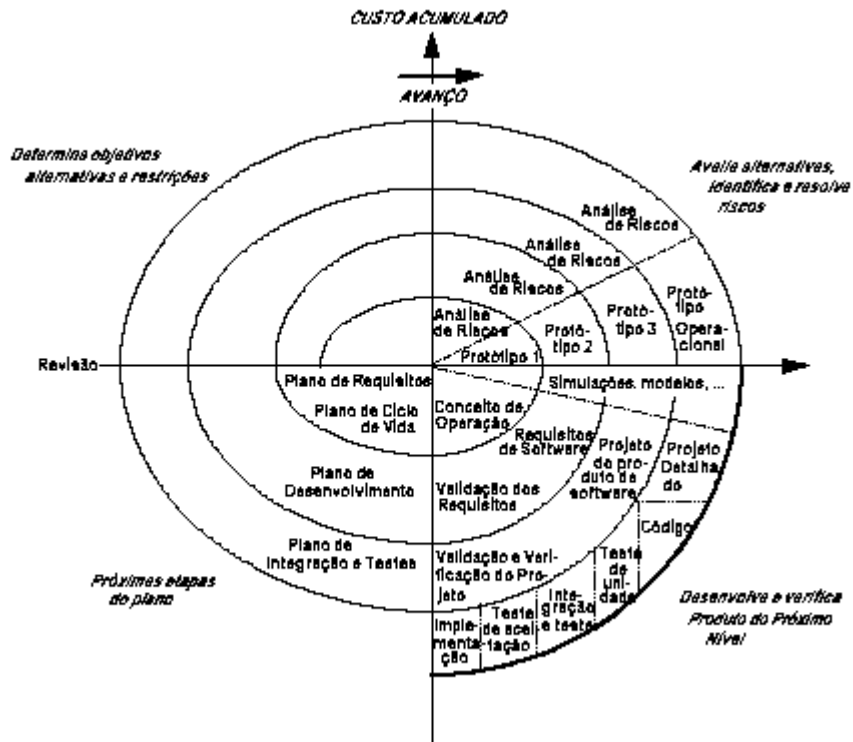


Figura 6- Modelo espiral

Planejamento: determinação dos objetivos, alternativas e restrições.

Análise de riscos: análise de alternativas e identificação/resolução dos riscos.

Engenharia: desenvolvimento do produto no “nível seguinte”.

Avaliação do cliente: avaliação dos resultados da engenharia.

2. Projeto de software

Segundo (Pressmam, pg.415) “O projeto é o primeiro passo da fase de desenvolvimento de qualquer produto ou sistema de engenharia.” Como pode-se observar nas palavras de Pressmam o projeto é o início de qualquer coisa ligada a engenharia, por exemplo, um engenheiro civil não manda construir um prédio sem antes elaborar um planta, verificar o terreno e outras coisas que denominamos de projeto, assim também é no desenvolvimento de software você não pode construir nada sem antes projetar, mas como todo engenheiro, antes mesmo de projetar alguma coisa deve-se ser desenvolvido um estudo para verificar se o desenvolver do projeto é viável, cria-se também mecanismos de rastreamento deste projeto para ver se tudo o que ele projetou esta acontecendo conforme o previsto, isto na engenharia de software é chamado de Gerência de Projeto que é o primeiro aspecto do projeto de software que estará sendo discutido agora.

2.1. Gerência de Projetos

Este tópico trata de conceitos fundamentais para uma administração efetiva de projetos de software levando em consideração as métricas de software utilizadas e o impacto que estas métricas causam sobre a gerência de projetos.

O que significa Gerenciamento de Projeto? Gerenciar um projeto requer a realização das seguintes atividades básicas:

- Planejamento:
 - Identificar e planejar as atividades e tarefas, preparando planos de trabalho, estimativas e cronogramas viáveis.
 - Identificar, planejar ações e eliminar fontes de risco, antes que aconteçam
 - Planejar as necessidades e a obtenção dos recursos humanos e técnicos
 - Definir os produtos intermediários e planejar a entrega destes.
- Execução:
 - Criar condições para que as atividades previstas possam ser realizadas.
 - Atuar para que o grupo realize as tarefas com sinergia e supere as dificuldades previstas e imprevistas, visando atingir os objetivos, orçamentos e cronogramas, com o menor desvio possível.
 - Servir de “técnico” entre o grupo do projeto e o mundo externo a este.
 - Estabelecer regras para as solicitações de mudanças nas especificações do projeto e negociar as condições para a aceitação destas.
 - Conduzir o processo para a obtenção de máxima qualidade do produto.
- Controle:
 - Medir e avaliar o progresso do projeto (físico e financeiro) em relação ao planejado, tomar ações corretivas e comunicar o andamento do projeto.

O exercício de todas essas atividades combinadas, realizadas por pessoas com grande energia e habilidade, se transforma num gerenciamento eficaz de projeto.

Gerência de projetos é a primeira camada da engenharia de software, abrangendo todo o processo de desenvolvimento do início ao fim. Para a condução de um projeto de software bem sucedido é necessário compreender:

- a) O escopo do trabalho;
- b) Os riscos;
- c) Os recursos exigidos;
- d) As tarefas a serem executadas;
- e) Os marcos de referência a serem acompanhados;
- f) O esforço/custo despendido;
- g) E a programação a ser seguida.

A gerência começa antes do trabalho técnico e prossegue durante o desenvolvimento do software e encerra somente quando o software se torna obsoleto, ao final da fase de gerencia de projeto um plano de projeto como o apresentado na figura 7 deverá ser gerado.

1.0 - Contexto
1.1 - Objetivos do projeto
1.2 - Funções principais
1.3 - Desempenho
1.4 - Restrições Técnicas e Administrativas
2.0 - Estimativas
2.1 - Dados utilizados
2.2 - Técnicas de Estimativa
2.3 - Estimativas
3.0 - Riscos do Projeto
3.1 - Análise dos Riscos
3.2 - Administração dos Riscos
4.0 - Cronograma
4.1 - Divisão do esforço no projeto
4.2 - Rede de Tarefas
4.3 - Timeline
4.4 - Tabela de recursos
5.0 - Recursos necessários
5.1 - Pessoal
5.2 - Software e Hardware
5.3 - Outros recursos
6.0 - Organização do Pessoal
7.0 - Mecanismos de Acompanhamento
8.0 - Apêndices

Figura 7 – Plano de projeto(fonte Pressman, pg.168)

2.1.1. Iniciando um projeto de software (Contexto)

Segundo (Pressman, pg55),

Antes de iniciar o planejamento de um projeto de um software, o objetivo e escopo devem estar bem estabelecidos, soluções alternativas devem ser consideradas e as restrições administrativas e técnicas identificadas. Sem essas informações é impossível definir estimativas de custo precisas, dividir de forma realista as tarefas de um projeto de tal forma que ofereça sinais significativos de progresso.

Como pode-se observar nas palavras de Pressmam um bom inicio de projeto deve-se concentrar-se em definição do escopo do software. O escopo de um software consiste em

definir quais são as funções primárias que o software deve realizar e procura delimitar a quantidade de funções, geralmente o escopo de um software é definido em conjunto com o cliente e se divide em algumas etapas:

- **Definição das funções:** consistem em extrair do cliente quais as reais funções que o software devesse ter;
- **Questões de desempenho:** extrair do documento ou do cliente quais são os fatores que influenciarão no desempenho do software, como por exemplo a quantidade de dados, a não existência de servidores, problemas de rede, comunicação remota, etc..
- **Restrições de hardware:** definir com o cliente que tipo de máquina está disponível para o sistema rodar.
- **Restrições de interface:** Verificar se haverá a necessidade de interface de comunicação com periféricos como, por exemplo: impressoras fiscais, leitores de código de barras, impressoras, etc...

2.1.2. Medidas e Métricas (Estimativas)

Medições e métricas auxiliam a entender o processo usado para se desenvolver um produto. O processo é medido a fim de melhorá-lo e o produto é medido para aumentar sua qualidade. Entretanto, medições às vezes levam à argumentações:

- a) Quais são as métricas apropriadas para o processo e o produto ?
- b) É justo usar medições para se comparar pessoas, processos e produtos ?

Essas e outras perguntas sempre vêm à tona quando é feita uma tentativa de se medir alguma coisa que não tenha sido medida no passado. Isso é o que acontece com a engenharia de software (o processo) e com o software (o produto).

As medidas são uma forma clara de avaliação de produtividade no desenvolvimento de software. Através da obtenção de medidas relativas à produtividade e à qualidade, é possível que metas de melhorias no processo de desenvolvimento sejam estabelecidas como forma de incrementar estes dois fatores.

Métricas de software referem-se a uma ampla variedade de medidas de software. No contexto de gerenciamento de projeto de software, estamos preocupados com métricas de produtividade e de qualidade - medidas do resultado do desenvolvimento de software como uma função do esforço aplicado e medidas da "adequação ao uso" do resultado que é produzido.

O software é medido por muitas razões:

- a) Indicar a qualidade do produto.
- b) Avaliar a produtividade das pessoas que produzem o produto.
- c) Avaliar os benefícios em termos de produtividade e qualidade derivados de novos métodos e ferramentas de software.
- d) Formar uma linha básica de estimativas.
- e) Ajudar a justificar os pedidos de novas ferramentas ou treinamento adicional.

As medições do mundo físico podem ser divididas em duas categorias:

1. **Diretas:** como o comprimento de algum componente de computador.
2. **Indiretas:** a qualidade do componente através da quantidade de componentes rejeitados.

Entre as medidas diretas do processo de engenharia de software incluem-se o custo e o esforço aplicados.

Entre as medidas diretas do produto as linhas de código (LOC) produzidas, velocidade de execução, tamanho de memória e defeitos registrados ao longo de certos espaço de tempo.

As medidas indiretas do produto incluem funcionalidade, qualidade, complexidade, eficiência, confiabilidade, manutenibilidade e muitas outros aspectos do produto.

Medidas diretas são fáceis de serem medidas dado que um padrão de medição seja adotado antecipadamente.

Medidas indiretas como funcionalidade e qualidade são mais difíceis de serem medidas.

Podemos dividir o domínio das métricas de software em:

a) **Métricas de produtividade:** concentram-se na saída do processo de engenharia de software com o objetivo de avaliar o próprio processo.

b) **Métricas de qualidade:** oferecem uma indicação de quão estreitamente o software conforma-se às exigências implícitas e explícitas do cliente.

c) **Métricas técnicas:** concentram-se nas características do software (por exemplo: complexidade lógica e modularidade).

Ou ainda podemos dividir o domínio das métricas de uma outra forma:

a) **Métricas orientadas ao tamanho:** medidas são derivadas a partir de atributos de tamanho do software como linhas de código, esforço, custo, quantidade de documentação, etc...

b) **Métricas orientadas para a função:** oferecem medidas indiretas.

c) **Métricas orientadas às pessoas:** usam informações sobre a maneira pela qual as pessoas desenvolvem software e percepções humanas sobre a efetividade das ferramentas e métodos.

2.1.2.1. Métricas Orientadas ao Tamanho

São medidas diretas do software e do processo por meio do qual ele é desenvolvido.

Uma tabela pode ser mantida sobre dados de projetos do últimos anos, como mostra a tabela abaixo.

Projeto	Esforço (pessoas-mês)	\$	KLOC	Pags doc.	Erros	Pessoas
AAA-1	24	168	12,1	365	29	3
CCC-4	62	440	27,2	1224	86	5
FFF-3	43	314	20,2	1050	64	6

Tabela 1- Métricas Orientadas ao Tamanho

A partir dos dados descritos para cada projeto da tabela um conjunto de métricas de qualidade e de produtividade orientadas ao tamanho pode ser desenvolvido para cada projeto. Médias podem ser computadas levando-se em conta todos os projetos.

Exemplos de algumas métricas:

Produtividade = KLOC/pessoa-mês.

Qualidade = defeitos/KLOC

Custo = \$/LOC

Documentação = Pg.Doc/KLOC

Repare que estas métricas orientadas ao tamanho giram em torno do número de linhas de código como a medida principal. Esta abordagem gera algumas discussões. As pessoas que são a favor afirmam LOC estão presentes em todos os projetos e são fáceis de serem contadas, que muitos modelos existentes utilizam esta medida como chave e que já existe um grande volume de literatura e de dados baseados em linhas de código.

Por outro lado, opositores afirmam que as medidas LOC são dependentes da linguagem de programação, que penalizam programas bem projetados porém mais curtos, que elas não podem acomodar facilmente linguagens não procedimentais e que seu uso em estimativas requer um nível de detalhes que pode ser difícil de se conseguir (ou seja, como estimar o número de linhas de código de um programa muito antes de análise e projeto terem sido concluídos?).

2.1.2.2.Métricas orientadas à função

São medidas indiretas do software e do processo através do qual o software é desenvolvido. Métrica orientada à função concentra-se na funcionalidade ou utilidade do software e não no número de linhas de código.

O método Ponto por Função (FP- function point) é uma métrica orientada à função e possibilita a medição de produtividade. Os pontos por função são computados completando-se a tabela representada na figura 8 abaixo:

Parâmetro de medida	Contagem	Fator de ponderação			Pontos
		Simples	Médio	Complexo	
Número de entradas do usuário		X 3	X 4	X 6	
Número de saídas do usuário		X 4	X 5	X 7	
Número de consultas do usuário		X 3	X 4	X 6	
Número de arquivos		X 7	X 10	X 15	
Número de interfaces externas		X 5	X 7	X 10	
Contagem - total					

Figura 8 – Métricas orientada a função

Onde:

Número de entradas do usuário: cada entrada do usuário que proporcione dados distintos orientados à aplicação é contada.

Número de saída do usuário: Cada saída do usuário que proporcione informações orientadas à aplicação é contada. Relatórios, telas, mensagens de erro, etc. Os itens de dados individuais dentro de um relatório não são contados separadamente.

Número de arquivos: Cada agrupamento lógico de dados que pode ser uma parte do banco de dados ou um arquivo convencional é contado.

Número de interfaces externas: todas as interfaces legíveis por máquina (por exemplo, arquivos de dados em fita ou disco) que sejam usadas para transmitir informações a outro sistema são contadas.

A cada contagem, um valor de complexidade é associado. Para computar os pontos por função, a relação do quadro 6 é usada:

$$FP = \text{contagem total} \times [0,65 + 0,01 \times \text{SOMA}(Fi)].$$

Quadro 6- Formula para calculo do FP

Onde

Contagem total: Representa a soma dos fatores da figura8;

Fi: São valores de ajustes de complexidade baseados nas respostas às perguntas do quadro 7 abaixo, seguindo uma escala de pontuação conforme a tabela 2 abaixo:

Pontos	Resposta
0	Sem influência
1	Incidental
2	Moderado
3	Médio
4	Significativo
5	Essencial

Tabela 2 – Pontos para os fatores

Fatores
O sistema requer backup e recuperação confiáveis?
São exigidas comunicações de dados?
Há funções de processamento distribuídas?
O desempenho é crítico?
O sistema funcionará num ambiente operacional existente, intensivamente utilizado?
O sistema requer entrada de dados <i>on-line</i> ?
A entrada de dados <i>on-line</i> exige que a transação de entrada seja elaborada em múltiplas telas ou operações?
Os arquivos-mestres são utilizados <i>on-line</i> ?
As entradas, saídas, arquivos e consultas são complexos?
O processo interno é complexo?
O código foi projetado de forma a ser reusado?
A conversão e a instalação estão incluídas no projeto?
O sistema é projetado para múltiplas instalações em diferentes organizações?
A aplicação é projetada de forma a facilitar mudanças e o uso pelo usuário?

Quadro 7 – Fatores de ajuste de complexidade

Os valores constantes da equação e os fatores de peso que são aplicados às contagens do domínio de informações são determinados empiricamente.

Assim que forem calculados os pontos por função serão usados de maneira análoga às LOC como medida de produtividade, qualidade e outros atributos de software:

Produtividade = FP/pessoa-mês.

Qualidade = defeitos/FP.

Custo = \$/FP.

Documentação = páginas de documentação/FP.

Uma outra medida chamada de *pontos de particularidades* foi acrescentada a fim de acomodar aplicações em que a complexidade algorítmica é elevada. Os pontos de particularidades são computados usando a figura 9 abaixo:

Parâmetro de medida	Contagem	Pontos	Total
Número de entradas do usuário		X 3	
Número de saídas do usuário		X 4	
Número de consultas do usuário		X 3	
Número de arquivos		X 7	
Número de interfaces externas		X 5	
Algoritmos		X 5	
Contagem - total			

Figura – 9 pontos de particularidade

O valor do ponto de particularidade é global e calculado pela mesma fórmula do quadro 6, pontos de particularidades e pontos por função representam a mesma coisa - a funcionalidade do software.

Assim como a métrica de LOC, a de ponto por função ou a de particularidades é controversa. Os proponentes afirmam que FP independe da linguagem de programação tornando-o ideal para aplicações que usam linguagens convencionais e não procedimentais. Além disso, afirmam que se baseia em dados que têm maior probabilidade de ser conhecidos logo no começo da evolução do projeto.

Os opositores afirmam que o método se baseia parcialmente em dados subjetivos e que o FP não tem significado físico direto - é apenas um número.

2.1.2.3.Reconciliação entre diferentes abordagens de métricas

A relação entre linhas de código e pontos por função depende da linguagem de programação utilizada e da qualidade do projeto.

A tabela 3 abaixo apresenta estimativas do número médio de linhas de código exigido para construir um ponto por função em várias linguagens de programação:

Linguagem	Quantidade de Linhas
Assembly	300
Cobol	100
Fortran	100
Pascal	90
Linguagens orientadas a objeto	30
Geradores de código	15

Tabela 3- Reconciliação entre diferentes abordagens de métricas

Questões que devem ser consideradas com relação ao uso de medidas de produtividade de software:

- a) As LOC/pessoa-mês ou FP/pessoa-mês de um grupo devem ser comparadas com dados similares de outros ?
 - b) Os gerentes devem avaliar o desempenho de pessoas usando essas métricas ?
- NÃO ! Muitos fatores influenciam a produtividade:

Fatores humanos: tamanho e experiência da organização de desenvolvimento.

Fatores do problema: complexidade do problema a ser resolvido e o número de mudanças nos requisitos ou restrições de projeto.

Fatores do processo: técnicas de análise e projeto que são usadas, linguagens e ferramentas CASE disponíveis e técnicas de revisão.

Fatores do produto: Confiabilidade e desempenho do sistema baseado em computador.

Fatores relacionados a recursos: disponibilidade de ferramentas CASE, recursos de hardware e software.

Os pontos por função e linhas de código foram considerados previsões relativamente precisas do esforço e custo do desenvolvimento. Porém, para que se possa usar estas métricas, uma linha básica de informação histórica deve ser estabelecida.

Por que é tão importante medir o processo de engenharia de software e o produto que ele produz? Se não medirmos não haverá nenhuma maneira real de sabermos se estamos ou não melhorando. A medição oferece benefícios em nível estratégico, em nível de projeto e em nível técnico.

Para estabelecer metas de melhoria, a situação atual do desenvolvimento do software deve ser entendida. Portanto, a medição é usada para estabelecer uma linha básica de processo a partir da qual as melhorias possam ser avaliadas.

Ao usar medições para estabelecer uma linha básica de projeto, questões como qualidade, prazo e estimativas tornam-se mais fáceis de serem administradas.

2.1.2.4. Estabelecimento de uma linha básica (Baseline)

A linha básica consiste de dados compilados de projetos passados de desenvolvimento de software. Esta linha básica pode conter medidas orientadas ao tamanho, à função e de qualidade. Os dados da linha básica devem ter os seguintes atributos:

- a) Os dados devem ser razoavelmente precisos.
- b) Os dados devem ser obtidos de tantos projetos quanto for possível.
- c) As medições devem ser consistentes, por exemplo, a linha de código deve ser interpretada consistentemente ao longo de todos os projetos para os quais são compilados dados.
- d) As aplicações devem ser idênticas ao trabalho que será estimado.

2.2. Estimativa

O planejamento é uma das atividades fundamentais do processo de gerenciamento de software. No planejamento são colocados estimativas de esforço humano (pessoas-mês), duração cronológica (em tempo de calendário) e custo (em moeda).

Mas como é feito um planejamento?

Muitas vezes, estimativas são feitas usando-se a experiência passada como único guia. Mas se um novo projeto for totalmente diferente dos projetos realizados até o momento? Assim, apenas a experiência passada talvez não seja suficiente.

Várias técnicas de estimativa estão disponíveis. Embora cada uma tenha suas particularidades, todas têm os seguintes atributos:

- a) O escopo do projeto deve estar estabelecido.
- b) Métricas de software são utilizadas e o histórico de aferições passadas é usado como uma base a partir da qual estimativas são feitas.
- c) O projeto é dividido em pequenas partes que são estimadas individualmente.

O processo de administração de projetos de software inicia-se com um conjunto de atividades que são denominadas *planejamento de projetos*. A primeira dessas atividades é a realização de estimativas. Quando queremos que estimativas sejam feitas, olhamos para o futuro e aceitamos certo grau de incerteza como coisa esperada.

Estimativas são baseadas em métricas históricas e empíricas.

Métricas históricas

Obtidas a partir de experiências anteriores da equipe.

Métricas empíricas

Dados estatísticos de diferentes equipes.

Planejar - estimar

Para estimar:

Experiência.

Acesso a boas informações históricas.

Riscos Inerentes.

Fatores que aumentam os riscos

Complexidade do projeto

Tamanho do Projeto

Estrutura do Projeto

Objetivos do Planejamento

O objetivo do planejamento de projetos de software é fornecer uma estrutura que possibilite ao gerente fazer estimativas razoáveis de recursos, custos e prazos. Essas estimativas são realizadas dentro de um plano de tempo limitado ao início de um projeto de software e devem ser regularmente atualizadas à medida que o projeto progride. Para se obter uma boa estimativa algumas atividades devem ser cumpridas, abaixo estão algumas delas:

Escopo do Software

A função e o desempenho atribuídos ao software durante o trabalho de engenharia de sistema computadorizado devem ser avaliados para que se possa estabelecer um escopo de projeto que seja claro e compreensível tanto em nível técnico como administrativo.

O escopo do software descreve a função, o desempenho, as restrições, as interfaces e a confiabilidade. As *funções* descritas na declaração do escopo são avaliadas e, em certos casos, refinadas para oferecer mais detalhes antes do início da estimativa. O *desempenho* abrange os requisitos de processamento e de tempo de resposta. As *restrições* identificam os limites impostos ao software pelo hardware externo, memória disponível e outros sistemas existentes.

Recursos

Cada recurso é especificado segundo quatro características: descrição do recurso, uma declaração da disponibilidade, tempo cronológico em que o recurso será exigido e por quanto tempo o recurso será aplicado. A disponibilidade do recurso para uma atividade específica deve ser estabelecida o mais cedo possível.

Recursos humanos

O planejador começa a avaliar o escopo e a selecionar as habilidades exigidas para concluir o desenvolvimento. Tanto os postos organizacionais (por exemplo, gerente,

engenheiro de software, etc.) como as especialidades (por exemplo, telecomunicações, banco de dados, etc.) são especificados. Para projetos relativamente pequenos, uma única pessoa pode executar todas as etapas de engenharia de software consultando especialistas quando necessário.

O número de pessoas pode ser determinado somente depois de uma estimativa de esforço de desenvolvimento.

Recursos de Hardware

Três categorias de hardware devem ser consideradas durante o planejamento do projeto: o hardware de desenvolvimento, o hardware de produção e outros elementos de hardware do novo sistema.

O *hardware de desenvolvimento* é um computador e os periféricos relacionados que serão usados durante o desenvolvimento do software.

O *hardware de produção* é onde o software será executado.

Outros elementos de hardware do sistema podem ser especificados com recursos para o desenvolvimento.

Recursos de Software

Da mesma forma que usamos hardware para construir um novo hardware, usamos software para auxiliar no desenvolvimento de um novo software.

Atualmente, os engenheiros de software usam um conjunto de ferramentas denominado *engenharia de software auxiliada por computador* (CASE- Computer-Aided Software engineering).

Principais categorias:

- ferramentas de planejamento de sistemas de informação;
- ferramentas de gerenciamento de projetos;
- ferramentas de apoio;
- ferramentas de análise e projeto;
- ferramentas de programação;
- ferramentas de integração e testes;
- ferramentas de construção de protótipos e simulação;
- ferramentas de manutenção;
- ferramentas de *framework*.

Reusabilidade

Reuso dos blocos de construção do software.

Duas regras qdo. o software reusável for utilizado como recurso:

Se o software existente cumprir os requisitos, adquira-o. O custo de aquisição de um software existente quase sempre será menor do que o custo para desenvolver um software equivalente.

Se o software existente exigir “alguma modificação” antes que possa ser adequadamente integrado ao sistema, proceda cautelosamente. O custo para modificar um software existente às vezes pode ser maior do que o custo para desenvolver um software equivalente.

Estimativas para Projetos de Software

Software - Elemento mais caro.

Erros de estimativa – diferença entre lucro e prejuízo.

Estimativa não é exata: variáveis podem afetar o custo final do software - humanas, técnicas, ambientais, políticas...

Usar técnicas oferece estimativas com riscos aceitáveis
 Atrasar a estimativa até um ponto tardio do desenvolvimento
 Usar técnicas de decomposição
 Desenvolver modelo empírico
 Adquirir ferramentas de estimativas automatizadas.

2.2.1. Estimativas Utilizando COCOMO (Constructive Cost Model)

O método COCOMO foi desenvolvido por Barry Boehm, para estimar esforço, prazo, custo e tamanho da equipe para um projeto de software.

O método foi derivado de um *data set* que compreendia 63 projetos cobrindo áreas como: negócios, controle, científica, suporte e sistema operacional.

Existem três modelos neste método:

COCOMO Básico: é um modelo estático de valor simples que computa o esforço (e custo) de desenvolvimento de software como uma função do tamanho de programa expresso em linha de código estimadas.

COCOMO Intermediário: computa o esforço de desenvolvimento de software como uma função do tamanho do programa e de um conjunto de “direcionadores de custo” que incluem avaliações subjetivas do produto, do hardware, do pessoal e dos atributos do projeto.

COCOMO Avançado: incorpora todas as características da versão intermediária, com uma avaliação do impacto dos direcionadores de custo sobre cada passo (análise, projeto, etc.) do processo de engenharia de software.

O COCOMO classifica os projetos em três tipos :

Modelo Orgânico (ou convencional): projetos de software simples, relativamente pequenos, nos quais pequenas equipes com boa experiência em aplicações trabalham num conjunto de requisitos não tão rígidos. Outras características: ambiente estável de desenvolvimento, algoritmos simples, prêmio relativamente baixo para término antes do prazo, tamanho relativamente pequeno, projetos na faixa de 50.000 linhas de código.

Modelo Semidestacado (ou difuso): projeto de software intermediário (em tamanho e complexidade) onde a equipe mescla grande e pouca experiência com aplicações, grande e pouca experiência com a tecnologia, o tamanho dos software varia até 300.000 linhas de código.

Modelo embutido (ou restrito): um projeto que deve ser desenvolvido dentro de um conjunto rígido de restrições operacionais, de hardware e de software.

COCOMO Básico

Estimativa de Esforço

1º) Determinar o Modo do Projeto (Orgânico, Difuso ou Restrito)

2º) Determinar o Número de Linhas de código

Esta é uma das limitações do método, pois como, no início do projeto, saberemos quantas linhas de código serão produzidas? Uma alternativa viável é a utilização combinada do método FPA (ou FP) e COCOMO. Uma vez que o FP pode transformar pontos de função em linhas de código, poderíamos usar o resultado da transformação para a aplicação das equações de esforço do COCOMO.

3º) Aplicar a Estimativa de LOC na equação do Esforço.

O COCOMO propicia três equações para determinar o esforço previsto para o projeto, conforme o modelo do mesmo.

Modelo	Esforço
Orgânico	$P/M=2,4 (KLOC)^{1,05}$
Difuso	$P/M=3,0 (KLOC)^{1,12}$
Restrito	$P/M=3,6 (KLOC)^{1,20}$

Supondo: 50 KLOC para um software orgânico.

100 KLOC para um software difuso.

200 KLOC para um software restrito.

Aplicando a fórmula:

- Para o software orgânico: $P/M=2,4 (50)^{1,05} = 146$
- Para o software difuso: $P/M=3,0 (100)^{1,12} = 521$
- Para o software restrito: $P/M=3,6 (200)^{1,20} = 2077$

Estimativa do Prazo

O COCOMO também provê equações para a determinação do prazo do projeto.

Modelo	Prazo
Orgânico	$\text{Prazo}=2,5 (P/M)^{0,38}$
Difuso	$\text{Prazo}=2,5 (P/M)^{0,35}$
Restrito	$\text{Prazo}=2,5 (P/M)^{0,32}$

Aplicando as estimativas de esforço do exemplo anterior, os prazos estimados seriam:

- Para o software orgânico: $\text{Prazo}=2,5 (146)^{0,38} = 16,61$ meses
- Para o software difuso: $\text{Prazo}=2,5 (521)^{0,35} = 22,33$ meses
- Para o software restrito: $\text{Prazo}=2,5 (2077)^{0,32} = 28,81$ meses

Estimativa de quantidade de Pessoal

É preciso dividir o esforço estimado pelo prazo estimado. Em nosso exemplo, os resultados seriam:

- Para o software orgânico: $\text{Equipe} = 146/16,61 = 9$.
- Para o software difuso: $\text{Equipe} = 521/22,33 = 23$.
- Para o software restrito: $\text{Equipe} = 2077/28,81 = 72$.

Diversos fatores não-previstos podem alterar o planejamento.

Um risco é a ocorrência de um evento que possa comprometer o andamento do projeto (cronograma, orçamento, qualidade...).

2.3. Análise de Riscos

Segundo Robert Charette.

[...] o risco preocupa-se com acontecimentos futuros. Hoje e amanhã estão além da preocupação ativa, uma vez que já estamos colhendo aquilo que foi anteriormente plantado por nossas ações passadas. A questão é, portanto: ao mudarmos nossas ações hoje, podemos criar oportunidade para uma situação diferente e esperançosamente melhor para nós mesmos amanhã? Isso significa,

em que o risco envolve mudanças, tais como mudanças de mentalidade, de opinião, de ações ou de lugares [...] o risco envolve escolha e a incerteza que a própria escolha acarreta[...]

Quando o risco é considerado no contexto da engenharia de software, três pilares conceituais de Charette estão em evidência:

a) O futuro é nossa preocupação - quais riscos poderiam fazer com que o projeto de software saísse fora do destino traçado?

b) A mudança é nossa preocupação - mudanças nos requisitos do cliente, nas tecnologias de desenvolvimento, nos computadores de destino e em todas as demais entidades ligadas ao projeto afetarão o sucesso global e o cumprimento do cronograma?

c) Escolhas - quais métodos e ferramentas devem usar, quantas pessoas devem ser envolvidas, quanta ênfase sobre a qualidade é suficiente, ...?

A análise de riscos é composta por quatro atividades: Identificação, projeção, avaliação e administração dos riscos.

2.3.1. Identificação dos riscos

Peter Drucker diz "Embora seja fútil tentar eliminar o risco e questionável tentar minimizá-lo, é fundamental que os riscos assumidos sejam os riscos certos."

Antes de identificar os riscos certos, é importante levantar todos os riscos que sejam óbvios tanto aos gerentes quanto aos demais profissionais envolvidos no desenvolvimento.

Os riscos de um projeto podem ser divididos em três categorias:

- **Riscos de projeto:** identificam problemas orçamentários, de cronograma, de pessoal, de recursos, de clientes e de requisitos e o impacto dos mesmos sobre o projeto.
- **Riscos técnicos:** identificam potenciais problemas de projeto, implementação, interface e manutenção. Ambigüidade de especificação, incerteza técnica, obsolescência de técnica são fatores de risco.
- **Riscos de negócio:** podem destruir os resultados até mesmo dos melhores projetos de software. Alguns riscos de negócio: (1) construir um excelente produto que ninguém realmente quer (risco de mercado), (2) construir um produto que não mais se encaixe na estratégia global de produtos da empresa, (3) construir um produto que a equipe de vendas não sabe como vender, (4) perder o apoio da alta administração devido à mudança de enfoque ou mudança de pessoas (risco administrativo); (5) perder o compromisso orçamentário ou de pessoal (risco orçamentário).

Nem todos os riscos são impossíveis de ser prognosticados antecipadamente. A identificação dos riscos envolve a relação dos riscos específicos de projeto dentro das amplas categorias acima esboçadas. Um dos melhores métodos para se entender cada um dos riscos é usar um conjunto de perguntas que ajude o planejador do projeto a compreender os riscos em termos técnicos ou de projeto.

Por exemplo, o planejador poderia atingir uma compreensão dos riscos de composição do pessoal ao responder às seguintes perguntas:

1. São as melhores pessoas disponíveis?
2. As pessoas têm a combinação certa de habilidades?
3. Há pessoas suficientes à disposição?
4. O pessoal está comprometido com toda a duração do projeto?

5. Algum membro do pessoal do projeto estará trabalhando somente em tempo parcial nesse projeto?
6. O pessoal tem as expectativas certas sobre o trabalho que tem à mão?
7. Os membros do pessoal receberam o necessário treinamento?
8. A rotatividade entre os membros do pessoal será baixa o bastante para permitir continuidade?

As repostas a essas perguntas permitirá que o planejador estime o impacto dos riscos na composição da equipe.

2.3.2. Projeção dos riscos

A projeção ou estimativa dos riscos tenta classificar cada risco de duas maneiras - probabilidade de que o risco seja real e as conseqüências dos problemas associados ao risco, caso ele ocorra.

Algumas tarefas são desempenhadas para a projeção dos riscos. O primeiro passo é definir uma escala em termos booleanos, qualitativos ou quantitativos. Ao extremo, cada pergunta poderia ser respondida com um "sim" ou "não". Mas isso é altamente irreal. Como avaliar os riscos em termos tão absolutos?

Uma abordagem melhor seria adotar uma escala de probabilidades qualitativas que tivesse os seguintes valores: altamente improvável, improvável, moderado, provável, altamente provável. Alternativamente, o planejador poderia calcular a probabilidade matemática de que o risco venha a ocorrer (por exemplo, uma probabilidade de 0.90 implica um risco altamente provável).

Probabilidades numéricas podem ser estimadas usando-se a análise estatísticas das métricas compiladas de experiências passadas, da intuição e de outras informações. Por exemplo, se as medidas compiladas de 45 projetos indicarem que 37 projetos experimentaram o dobro de mudanças feitas pelo cliente do que as originalmente previstas, a probabilidade de que um novo projeto experimente números de mudança excessivos é de $37/45 = 0,82 =$ muito provável.

Finalmente, os riscos são ponderados em função do possível impacto percebido sobre o projeto e depois colocados em ordem de prioridade. Três fatores afetam o impacto: sua natureza, seu escopo e seu tempo de ocorrência.

A natureza do risco indica os problemas prováveis se ele ocorrer. Por exemplo, uma interface externa com o hardware do cliente mal definida (risco técnico) frustrará a realização do projeto e dos testes e provavelmente levará a problemas na integração do sistema posteriormente.

O escopo de um risco combina a gravidade (quão ele é sério) com sua distribuição global (quanto do projeto será afetado ou quantos clientes serão prejudicados?).

O tempo de ocorrência de um risco considera quando e por quanto tempo o impacto será sentido.

2.3.3. Avaliação dos riscos

Neste momento do processo de análise dos riscos é possível estabelecer um conjunto de termos que tem a forma $[ri, li, xi]$ onde ri é o risco, li a probabilidade do risco e xi o impacto do risco.

Durante a avaliação dos riscos examinaremos mais detalhadamente a precisão das estimativas que foram feitas durante a projeção dos riscos, tentaremos determinar uma

ordem de prioridade e começaremos a pensar em maneiras de controlar e/ou evitar riscos que têm probabilidade de ocorrer.

A fim de que a avaliação seja útil, um nível de risco referente deve ser definido. Para a maioria dos projetos de software, o custo, os prazos e o desempenho representam três níveis de risco referentes típicos.

Um nível de risco referente tem um ponto simples ou também chamado de break-point. A figura 10 abaixo representa esta situação:

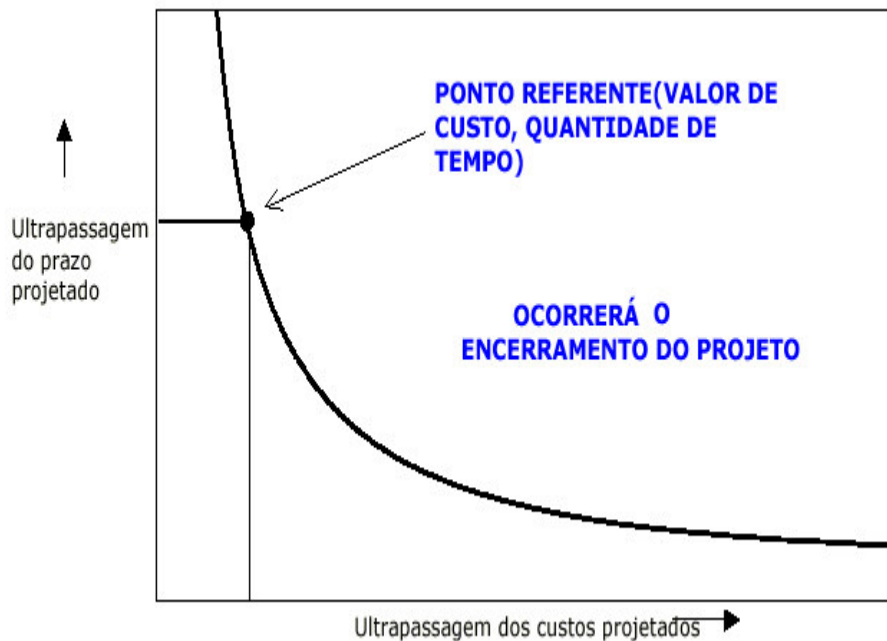


Figura 10 – Break-Point

Se o acontecimento de um risco levar a problemas que façam com que o custo ou os prazos seja ultrapassado haverá um nível, representado pela curva da figura, que provocará o encerramento do projeto. No ponto simples, a decisão de prosseguir ou encerrar possuem mesmo peso. Qualquer outro ponto alcançado na curva provoca o término do projeto, representado pela parte acima da curva.

Então, durante a avaliação dos riscos:

1. Definir níveis de risco referentes.
2. Procurar desenvolver um relacionamento entre cada $[r_i, l_i, x_i]$ e cada um dos níveis referentes.
3. Determinar o conjunto de pontos referentes que define uma região de encerramento.
4. Tentar levantar como associações combinadas dos riscos afetarão um nível referente.

2.3.4. Gerenciamento e monitoração dos riscos

Para a atividade de gerenciamento e monitoração dos riscos, o trio (descrição, probabilidade e impacto dos riscos) associado a cada risco é usado como uma base a partir da qual os passos de gerenciamento dos riscos são desenvolvidos.

Por exemplo, se a alta rotatividade de pessoal é tomada como um risco r_1 , com uma probabilidade estimada $l_1 = 70\%$ (bastante elevada) e o impacto x_1 é projetado para aumentar a duração do projeto em 15% e o custo global em 12%, os seguintes passos de administração dos riscos são propostos:

1. Reunir-se com o pessoal atual para determinar as causas da rotatividade e pessoal.
2. Tomar providências para suavizar aquelas causas que estejam sob nosso controle antes que o projeto se inicie.
3. Assim que o projeto iniciar, pressupor que haverá rotatividade de pessoa e desenvolver técnicas para garantir a continuidade quando as pessoas saírem.
4. Organizar equipes de projeto de forma que as informações a respeito de cada atividade sejam amplamente difundidas.
5. Definir padrões de documentação e estabelecer mecanismos para se certificar de que os documentos sejam desenvolvidos de maneira oportuna.
6. Realizar revisões de todo o trabalho com os colegas de forma que mais de uma pessoa esteja a par das atividades desenvolvidas.
7. Definir um membro do pessoal que sirva de backup para cada profissional mais crítico.

Estes passos de administração dos riscos acarretam custos - por exemplo, manter um profissional backup para cada profissional crítico. Se as providências para evitar os riscos da alta rotatividade de pessoal aumentarem o custo e a duração do projeto em 15% e o fator de custo predominante for o backup, a administração poderá decidir não implementar esse passo. Por outro lado, se os passos de administração dos riscos forem projetados para aumentar os custos em 5% e a duração em apenas 3%, a administração provavelmente os colocará em prática.

Ou seja, parte da administração dos riscos significa avaliar quando os benefícios advindos das atividades tomadas para evitá-los são ultrapassados pelos custos associados à implementação dos mesmos.

O gerenciamento pode ser representado graficamente pela figura 11 abaixo:

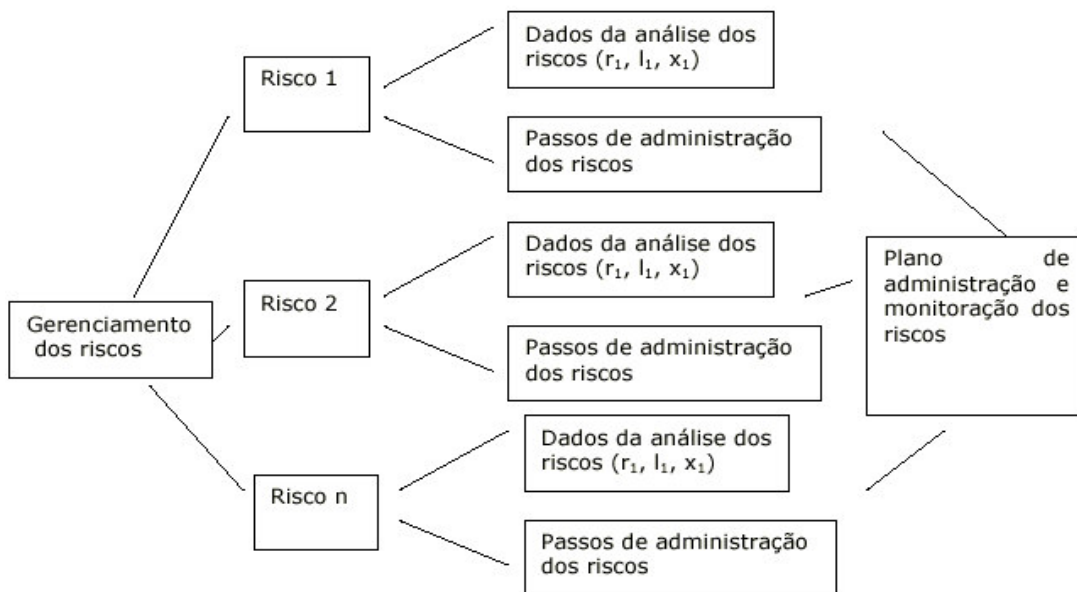


Figura 11 – Gerenciamento dos riscos

Para um grande projeto, muitos riscos podem ser identificados. Se entre três e sete passos de administração dos riscos forem identificadas para cada um, a administração dos riscos pode tornar-se um projeto em si mesmas! Entretanto, a experiência indica que 80% dos riscos globais do projeto podem ser correspondentes a apenas 20% dos riscos identificados. Durante os primeiros passos de análise dos riscos é possível determinar quais riscos residem nesses 20%. Por esse motivo, alguns dos riscos identificados, avaliados e projetados podem não se encaixar no plano de administração dos riscos – eles não correspondem aos 20% críticos.

2.4. Definição do cronograma

A determinação de um cronograma para desenvolvimento de software pode ser vista sob duas perspectivas diferentes:

- Uma data final para a entrega já foi estabelecida. A organização de software vê-se compelida a distribuir o esforço dentro do espaço de tempo previsto.
- Limites cronológicos aproximados foram discutidos, mas que a data final será estabelecida pela engenharia de software. O esforço é distribuído para que se possa tirar o melhor proveito dos recursos e uma data final é definida após cuidadosa análise.

A precisão na determinação de um cronograma às vezes pode ser bem mais importante do que a precisão na determinação dos custos. Os custos adicionados podem ser absorvidos por nova determinação de preços ou por amortização após o longo de um grande número de vendas. Um cronograma não cumprido, porém, pode reduzir o impacto de mercado, criar clientes insatisfeitos e elevar os custos internos.

Quando abordamos a determinação de prazos para um projeto de software, uma série de perguntas pode ser feita:

- Como relacionamos o tempo cronológico com o esforço humano?
- Quais tarefas e paralelismos devem ser esperados?
- Quais marcos de referência podem ser usados para mostrar o progresso?
- Como representaremos fisicamente o cronograma e como rastreamos o progresso quando o projeto se iniciar?

2.4.1. Relações pessoas-trabalho

Num projeto de desenvolvimento de software pequeno, uma única pessoa pode analisar os requisitos, executar o projeto, gerar o código e realizar testes. À medida que o tamanho do projeto aumenta, mais pessoas devem ser envolvidas. Raramente podemos abordar um esforço de 10 pessoas-ano com uma pessoa trabalhando durante 10 anos.

Mito: se nos atrasarmos, sempre poderemos acrescentar mais programadores e posteriormente sairmos do atraso no projeto. Infelizmente, acrescentar pessoas tardiamente num projeto muitas vezes tem um efeito desintegrador fazendo com que o cronograma fuja ainda mais do controle:

1. Tempo deverá ser despendido para treinar o novo pessoal.
2. Um número maior de pessoas aumenta o número de canais de comunicação e a complexidade desta ao longo de um projeto. Cada canal de comunicação adicional requer esforço e tempo adicionais.

Por exemplo:

Sejam quatro engenheiros de software, cada um capaz de produzir 5000 LOC/ANO trabalhando individualmente num projeto. Se estes quatro engenheiros são inseridos num projeto em equipe, seis potenciais canais de comunicação são criados. Cada canal requer tempo que, de outra forma, seria usado para o desenvolvimento.

Vamos supor que a produtividade da equipe seja reduzida em 250 LOC/ano para cada canal de comunicação devido ao consumo de recursos gerais associados à comunicação. Assim a produtividade da equipe será igual a $(20.000 - (250 \times 6)) = 18.500$ LOC/ano = 7,5% menos do que aquilo que poderíamos esperar. Supondo que o projeto de um ano em que a equipe está trabalhando tem o seu cronograma atrasado e restando apenas dois meses, mais duas pessoas são acrescentadas à equipe. O número de canais de comunicação sobe para 14. A entrada de produtividade do novo pessoal é equivalente a $840 \times 2 = 1680$ LOC para os dois meses até a entrega. A produtividade da equipe agora é igual a $20.000 + 1680 - (250 \times 14) = 18.180$ LOC/ano.

2.4.2. Definição de tarefa e paralelismo

Quando mais de uma pessoa está envolvida num projeto de software, é provável que as atividades de desenvolvimento sejam executadas em paralelo.

Marcos de referência ao longo do processo de engenharia de software oferece ao gerente uma indicação do progresso. Um marco de referência é atingido assim que a documentação produzida como parte de uma tarefa de engenharia de software é revisada e aprovada. A natureza concorrente das atividades de engenharia de software leva a uma série de importantes exigências de cronograma. Uma vez que as tarefas paralelas ocorrem de maneira assíncrona, devem ser determinadas as dependências intertarefas para garantir o contínuo progresso rumo à conclusão.

2.4.3. Distribuição do esforço

Muitas técnicas de estimativa de projetos produzem a estimativa de pessoas-mês exigidas para o desenvolvimento do software.

A figura 12 abaixo ilustra uma distribuição do esforço exigido entre algumas macro-atividades do desenvolvimento do software. Esta distribuição enfatiza as tarefas de análise e projeto realizadas inicialmente e as tarefas de teste realizadas próximo do término do projeto.

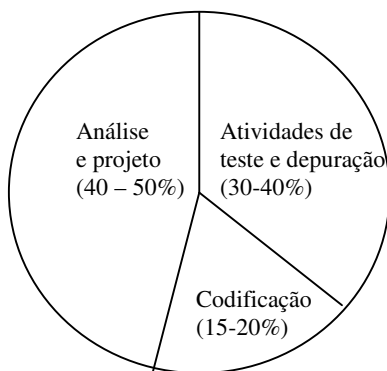


Figura – 12 Distribuição do esforço

Esta divisão do esforço proposta deve ser usada como uma diretriz. As características de cada projeto é que devem ditar qual distribuição acomoda melhor para que os objetivos do projeto sejam alcançados dentro do custo e prazo estimados.

O esforço gasto com planejamento de projeto raramente é responsável por mais do que 2 a 3% do esforço total, a menos que o plano envolva grandes gastos com riscos.

A análise de requisitos pode envolver de 10 a 25% do esforço de projeto. O esforço dependido em análise e construção de protótipos deve elevar-se em proporção direta com o tamanho e complexidade do projeto. Em geral de 20 a 25% do esforço total é gasto com projeto.

Em geral, devido ao esforço empregado em análise e projeto espera-se que da codificação seja exigido um esforço menor. Os testes podem consumir de 15 a 20%. A condição crítica do software dita a quantidade de testes exigida. Por exemplo, se uma falha no software pode causar a perda de uma vida humana, então a fase de testes terá um esforço maior.

2.4.4. Métodos de determinação de cronograma

Na especificação de um cronograma devemos estabelecer prazos para o esforço de desenvolvimento.

Dois métodos de determinação de cronograma são:

- PERT – Program Evaluation and Review Technique (método de avaliação e revisão de programa).
- CPM – Critical Path Method (método do caminho crítico).

Em ambas as técnicas existem uma rede de tarefas a serem desenvolvidas desde o começo até o final do projeto.

A rede é definida ao se desenvolver uma lista de todas as tarefas associadas a um projeto específico e uma lista de disposições que indica em que ordem as tarefas devem ser executadas.

As duas técnicas permitem ao planejador de software:

- Determinar o caminho crítico – cadeia de tarefas que determina a duração do projeto.
- Estabelecer as estimativas de tempo mais prováveis para tarefas individuais ao aplicar modelos estatísticos.
- Calcular limites de tempo que definam uma janela de tempo para uma tarefa em particular.

Alguns limites de tempo são definidos tanto para o PERT como para o CPM:

- O limite mais cedo em que uma tarefa pode iniciar-se quando todas as tarefas precedentes foram completadas no tempo mais curto possível.
- O limite mais tarde para o início da tarefa antes que o tempo de conclusão mínimo do projeto seja atrasado.
- O término mais breve – a soma do início mais cedo com a duração da tarefa.
- O término mais tardio – a soma do início mais tardio com a duração da tarefa.
- A flutuação total – a quantidade de superávit de tempo ou de margem de segurança permitida nas tarefas de determinação de prazos de forma que o caminho crítico da rede seja mantido dentro do prazo.

Os cálculos do tempo-limite levam à determinação do caminho crítico e proporcionam ao gerente um método quantitativo para avaliar o progresso a medida em que as tarefas são concluídas.

O planejador deve reconhecer que o esforço despendido no software não se encerra ao final do desenvolvimento. O esforço de manutenção, ainda que não seja fácil de programar neste estágio, tornar-se-á, por fim, o maior fator de custo. Uma meta primordial da engenharia de software é ajudar a reduzir esse custo.

2.4.5. Gráfico de Gantt

O gráfico de Gantt constitui-se de uma ótima ferramenta para representar o cronograma do PLANO DO PROJETO. Permite uma comparação entre o previsto e o realizado, mostrando o andamento do projeto no tempo.

Ele deve aparecer em dois níveis distintos, no mínimo:

Cronograma Mestre - representando as atividades sumárias. Este cronograma dá uma visão geral dos prazos do projeto, mostra sua duração total e interessa principalmente ao usuário ou superiores nos níveis estratégicos ou tático de decisões.

Cronograma Parcial - representando o detalhamento das atividades sumárias. Este cronograma relaciona as atividades e usa uma escala de tempo menor do que no cronograma mestre, ele interessa mais aos componentes do projeto para controle operacional.

O gráfico de Gantt mantém o Gerente de Projeto ciente do progresso realizado na execução do plano. A comparação da execução com o previsto converte-se simplesmente, em um trabalho de menor nível e a direção dispõe de mais tempo para estudar as tendências indicadas pelo gráfico e agir de maneira mais conveniente.

Este tipo de gráfico evidencia a quantidade de trabalho executado, se esta quantidade é inferior a prevista, indica a quem se deve a responsabilidade pelo êxito ou fracasso de uma determinada tarefa ou projeto.

O gráfico de Gantt é muito sintético e muito fácil de traçar e de ler, não apresenta linhas que se cruzam e, todas as anotações avançam com o tempo, da esquerda para a direita da folha. Apresenta, normalmente, tal continuidade de traçado que qualquer interrupção de registro ou qualquer lacuna de informação, relativa ao que se passou, torna-se visível, imediatamente. Torna visível o transcurso do tempo e, por isso ajuda a reduzir a ociosidade e a perda de tempo.

As linhas traçadas horizontalmente no gráfico representam a relação entre a quantidade de trabalho realizado durante um tempo dado e a quantidade prevista. Este é o detalhe característico que distingue o gráfico de Gantt de todos os demais gráficos de controle: cada coluna e a barra nela traçada representam, simultaneamente:

- Quantidades iguais de tempo
- Quantidades variáveis de trabalho previsto
- Quantidades variáveis de trabalho realizado

As vantagens do gráfico de Gantt são que ele permite razoavelmente efetuar o planejamento, a programação, o controle, forçar a definição de objetivos e responsabilidades; prevenir a omissão de tarefas e apropriar tempo, quantidade de trabalho realizados e as relações entre tempos e quantidades.

Como desvantagens temos a dificuldade de controlar um grande número de atividades e em período longo (muitas linhas e colunas).

As etapas envolvidas na aquisição do software dependem de quão crítico é o software e o custo final. Se o custo não é elevado, em alguns casos é válida a aquisição e a experiência do software para saber se atende às necessidades do cliente.

Para pacotes mais caros, as seguintes diretrizes podem ser consideradas:

1. Desenvolva uma especificação funcional e de desempenho do software desejado.
2. Estime um custo interno para desenvolver e a data de entrega.
3. Escolha mais de um pacote de software candidatos.
4. Desenvolva uma matriz de comparação entre os softwares candidatos que confronte entre as funções chaves.
5. Avalie cada pacote de software baseado na qualidade do produto, suporte do vendedor, reputação do fornecedor.
6. Contate outros usuários do software e peça opiniões.

Na análise final, a decisão de produzir ou comprar é tomada em função das seguintes condições:

- A data de entrega do produto precede a data do software desenvolvido internamente?
- O custo de aquisição e customização será menor do que o custo para se desenvolver internamente?
- O custo do suporte externo será menor do que o custo do suporte interno?

Outros critérios:

- A entrega e disponibilidade.
- A conformidade aos requisitos.
- A probabilidade de mudanças.

3. Análise de requisitos

O completo entendimento dos requisitos de software é um ponto fundamental para o sucesso de um projeto de software. Independentemente da precisão com a qual um software venha a ser projetado e implementado, ele certamente trará problemas ao cliente/usuário se a sua análise de requisitos foi mal realizada.

A análise de requisitos é uma tarefa que envolve, antes de tudo, um trabalho de descoberta, refinamento, modelagem e especificação das necessidades e desejos relativos ao software que deverá ser desenvolvido. Nesta tarefa, tanto o cliente quanto o desenvolvedor vão desempenhar um papel de grande importância, uma vez que caberá ao primeiro a formulação (de modo concreto) das necessidades em termos de funções e desempenho, enquanto o segundo atua como indagador, consultor e solucionador de problemas.

Esta etapa é de suma importância no processo de desenvolvimento de um software, principalmente porque ela estabelece o elo de ligação entre a alocação do software em nível de sistema (realizada na etapa de Engenharia de Sistema) e o projeto do software.

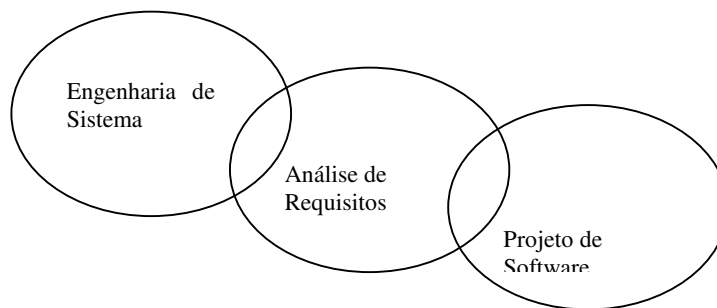


Figura 14 – Representação do vínculo dos requisitos

Desta forma, esta etapa permite:

Ao engenheiro de software:

- especificar as necessidades do software em termos de funções e de desempenho;
- estabelecer as interfaces do software com os demais elementos do sistema e
- especificar as restrições de projeto.

Ao engenheiro de software (analista):

- uma alocação mais precisa do software no sistema e
- a construção de modelos do processo, dos dados e dos aspectos comportamentais que serão tratados pelo software.

Ao projetista:

- a obtenção de uma representação da informação e das funções que poderá ser traduzida em projeto procedimental, arquitetônico e de dados.

Além disso, é possível definir os critérios de avaliação da qualidade do software a serem verificados uma vez que o software esteja concluído.

Abaixo na figura 15 um fluxograma das funções que a análise de requisitos deve cumprir:

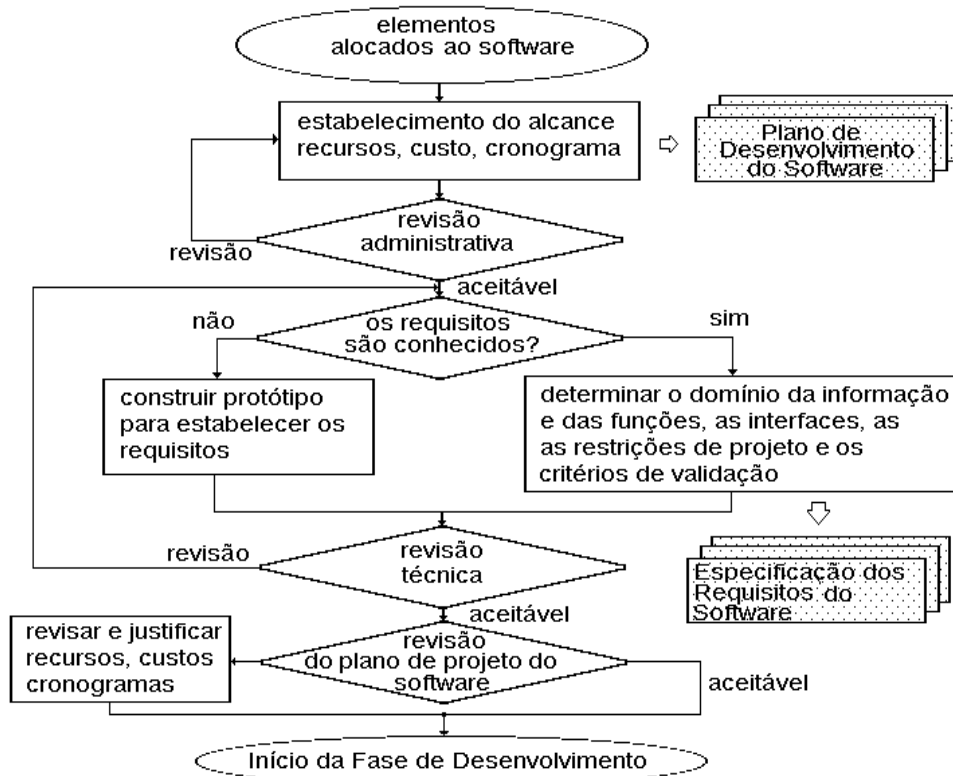


Figura 15 – Fluxograma das funções da análise de requisitos.

3.1. Atividades da Análise de Requisitos

3.1.1. Análise do Problema

Nesta fase inicial, o analista estuda os documentos de Especificação do Sistema e o Plano de Software, como forma de entender o posicionamento do software no sistema e revisar o escopo do software utilizado para definir as estimativas de projeto.

Um elo de comunicação entre o analista e o cliente deve ser estabelecido.

A meta do analista nesta fase é identificar os principais fatores do problema a ser resolvido, pela ótica do cliente.

3.1.2. Avaliação e Síntese

Esta fase envolve a análise dos fluxos de informação e a elaboração de todas as funções de tratamento e os aspectos de comportamento do software. Ainda, é importante que uma definição de todas as questões relacionadas à interface com o sistema, além de uma especificação das restrições do projeto.

Terminada a análise, o analista pode iniciar a síntese de uma ou mais soluções para o problema. Na síntese das eventuais soluções, o analista deve levar em conta as estimativas e as restrições de projeto. Este processo de avaliação e síntese prossegue até que o analista e o cliente estejam de acordo sobre a adequação das especificações realizadas para a continuidade do processo.

3.1.3. Modelagem

A partir da tarefa de avaliação e síntese, o analista pode estabelecer um modelo do sistema, o qual permitirá uma melhor compreensão dos fluxos de informação e de controle, assim como dos aspectos funcionais e de comportamento. Este modelo, ainda distante de um projeto detalhado, servirá de referência às atividades de projeto, assim como para a criação da especificação de requisitos.

Em muitas situações como forma de reforçar o conhecimento sobre a viabilidade de software a ser desenvolvido, pode ser necessário o desenvolvimento de um protótipo de software como alternativa ou como trabalho paralelo à análise de requisitos.

3.1.4. Especificação de Requisitos e Revisão

A etapa de Análise de requisitos culmina com a produção de um documento de **Especificação de Requisitos de Software**, que registra os resultados das tarefas realizadas. Eventualmente, pode ser produzido como documento adicional um Manual Preliminar do Usuário. Embora pareça estranho, a produção deste manual permite que o analista passe a olhar para o software da ótica do cliente/usuário, o que pode ser bastante interessante, principalmente em sistemas interativos. Além disso, a posse de um Manual do Usuário, mesmo em estágio preliminar permite ao cliente uma revisão dos requisitos (de interface, pelo menos) ainda num estágio bastante prematuro do desenvolvimento de software. Desta forma, algumas decepções resultantes de uma má definição de alguns aspectos do software podem ser evitadas.

3.2. Processos de comunicação

O desenvolvimento de um software é, na maior parte das vezes, motivado pelas necessidades de um cliente que deseje automatizar um sistema existente ou obter um novo sistema completamente automatizado. O software, porém, é desenvolvido por um desenvolvedor ou por uma equipe de desenvolvedores. Uma vez desenvolvido, o software será provavelmente utilizado por usuários finais, os quais não são necessariamente os clientes que originaram o sistema.

De fato, existem três partes envolvidas no processo de desenvolvimento de um produto de software: o cliente, o desenvolvedor e os usuários. Para que o processo de desenvolvimento seja conduzido com sucesso, é necessário que os desejos do cliente e as expectativas dos eventuais usuários finais do sistema sejam precisamente transmitidos ao desenvolvedor.

Este processo de transmissão de requisitos, do cliente e dos usuários ao desenvolvedor, invariavelmente apresenta relativa dificuldade, considerando alguns aspectos:

- geralmente, os clientes não entendem de software ou do processo de desenvolvimento de um programa;
- o desenvolvedor, usualmente, não entende do sistema no qual o software cao executar.

A boa comunicação entre a equipe de desenvolvimento e o cliente é fundamental para a realização de sucesso da atividade de análise de requisitos. Nem sempre a boa comunicação é alcançada devido aos motivos já citados.

No início do processo, em geral, uma entrevista é realizada entre o cliente e o desenvolvedor. Durante as primeiras entrevistas com o cliente, o analista pode começar fazendo perguntas que levem a uma compreensão básica dos seguintes pontos:

- a) O problema.
- b) Pessoas que querem uma solução.
- c) Natureza da solução desejada.

O primeiro conjunto de perguntas concentra-se no cliente, nas metas globais e nos benefícios. Por exemplo:

- a) Quem está por trás do pedido deste trabalho?
- b) Quem usará a solução?
- c) Qual o benefício econômico de uma solução bem-sucedida?
- d) Há outra fonte para a solução exigida?

O próximo conjunto de perguntas possibilita ao analista compreender melhor o problema e ao cliente expressar sua percepção sobre uma solução:

- a) Como você caracterizaria um bom resultado que seria gerado por uma solução bem sucedida?
- b) Qual(is) problema(s) essa solução resolverá?
- c) Você poderia mostrar-me o ambiente em que a solução será usada?
- d) Existem questões de desempenho ou restrições especiais que afetarão a maneira pela qual a solução é abordada?

Outro conjunto de perguntas concentra-se na efetividade da entrevista:

- a) Você é a pessoa certa para responder estas perguntas? Suas respostas são oficiais?
- b) Minhas perguntas são pertinentes ao problema que você tem?
- c) Estou fazendo perguntas demais?
- d) Há mais alguém que possa fornecer informações adicionais?
- e) Existe algo que eu deva perguntar-lhe?

Estas perguntas podem auxiliar a introduzir uma comunicação para conduzir uma análise de requisitos bem sucedida. Entretanto, esta forma de encontro onde o analista faz uma série de perguntas e o cliente responde não deve persistir nas demais reuniões que acontecerão.

Elementos que combinam solução, negociação e especificação devem ser abordados.

3.2.1. Técnicas facilitadas para especificação de aplicações.

Realizar um levantamento de requisitos tendo em mente um trabalho dividido em duas equipes – clientes e desenvolvedores - tem demonstrado não ser uma abordagem de sucesso. Se cada pessoa define seu território e se comunica com a outra através de documentos, sessões de perguntas e respostas,... ocasiona mal-entendidos, omissão de informações importantes e isso contribui para a falta do sucesso em um trabalho.

A fim de solucionar estes problemas, pesquisadores desenvolveram uma abordagem à coleta de exigências orientada às equipes a qual é aplicada durante as primeiras etapas da análise e especificação.

Denominada **Técnica Facilitada para Especificação de Aplicações (Facilitated Application Specification Techniques - FAST)**, esta abordagem estimula a criação de uma equipe conjunta de clientes e desenvolvedores a trabalharem juntos para:

- a) identificar o problema,
- b) propor elementos de solução,
- c) negociar diferentes abordagens e
- e) especificar um conjunto preliminar de requisitos de solução.

Outras abordagens são propostas como, por exemplo, JAD (Joint Application Development – Desenvolvimento Conjunto de Aplicação), desenvolvido pela IBM e The Method – Performance Resources, Inc.

Independente da técnica de entrevista entre as equipes, todas seguem estas diretrizes:

- a) Encontro realizado em local neutro e contando com a participação de clientes e desenvolvedores.
- b) Regras para preparação e participação são estabelecidas.
- c) Uma agenda que seja formal o bastante para cobrir todos os pontos importantes, mas informal o suficiente para encorajar o livre fluxo de idéias.
- d) Designar um moderador para controlar a reunião.
- e) Meta: identificar o problema, propor elementos de solução, negociar diferentes abordagens e especificar um conjunto preliminar de requisitos de solução.

Então, inicialmente desenvolvedores e clientes se reúnem com o objetivo de perceber o escopo do problema e uma solução. Uma “requisição de produto” é elaborada. Uma data e um lugar são marcados para a realização da FAST. Um moderador é designado. Clientes e desenvolvedores são convidados e uma cópia da requisição do produto é distribuída para cada participante antes da data da reunião.

Cada participante é solicitado a fazer uma lista de objetos que:

- a) fazem parte do ambiente que interage com o sistema,
- b) são produzidos pelo sistema,
- c) são usados para que o sistema execute suas funções.

3.3. Princípios da análise

Pesquisadores desenvolveram vários métodos de análise e especificação de software identificando problemas e suas causas.

Cada método tem suas características que são únicas mas todos compartilham alguns princípios fundamentais:

- 1) O **domínio da informação** deve ser representado e compreendido. -> a função pode ser entendida melhor.
- 2) **Modelos** que descrevam a informação, função e comportamento devem ser desenvolvidos. -> comunicação de forma resumida.
- 3) Modelos devem ser divididos em **partições** de tal forma que revele detalhes em camadas. -> redução da complexidade.

4) O processo de análise deve mover-se da **informação essencial para os detalhes de implementação**.

3.3.1. O domínio da informação

Composto por dados e eventos. Para entender o domínio da informação cada um destes três pontos de vista deve ser considerado para dados e eventos:

a) **fluxo da informação**: representa a maneira pela qual os dados e eventos se modificam à medida que cada um se movimentam pelo sistema. Ao longo deste caminho, novas informações podem ser introduzidas a partir de um depósito. Uma entrada pode ser transformada em informações intermediárias até alcançar a saída.

b) **conteúdo da informação**: representa os dados e os itens de controle que compõem um determinado item de informação mais amplo.

c) **estrutura da informação**: representa a organização interna dos dados que compõe um item de informação.

3.3.2. Modelagem

Obter maior compreensão do que deve ser construído. Os modelos devem ser capazes de modelar a informação que o software transforma, as funções que possibilitam as transformações e o comportamento do sistema quando a transformação está ocorrendo.

Durante a análise de requisitos construímos modelos que se concentram naquilo que o software deve fazer e não como ele deve fazer. Papéis dos modelos criados durante a análise dos requisitos:

a) Ajuda o analista a entender a informação, função e o comportamento.

b) Torna-se ponto principal para revisão.

c) Torna-se base para o projeto a qual pode ser “mapeada” para um contexto de implementação.

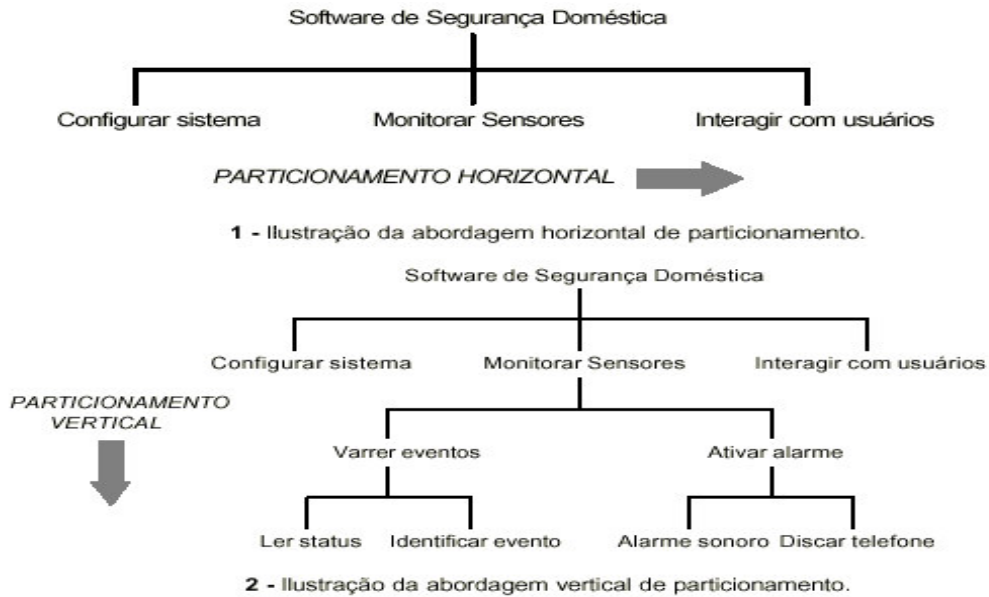
3.3.3. Particionamento

Muitas vezes, os problemas a serem resolvidos são excessivamente complexos, apresentando grande dificuldade para a sua compreensão (e conseqüente resolução) como um todo. Para dominar de forma completa os problemas sob análise, um princípio fundamental é a decomposição em partes menores – particionamento.

A partir do particionamento de um problema e a partir da análise de cada parte estabelecida, o entendimento fica mais facilitado. Desta forma, é possível estabelecer as interfaces de cada parte do problema de modo que a função global do software seja realizada.

O procedimento básico de particionamento é o estabelecimento de uma estrutura hierarquizada de representação da função ou da informação, dividindo em partições o elemento superior, podendo esta divisão ser efetuada segundo uma abordagem vertical (deslocando-se verticalmente na hierarquia) ou horizontal. As figuras a seguir representam a aplicação sobre um exemplo das abordagens horizontal e vertical, respectivamente.

Neste o particionamento é realizado sobre o aspecto funcional do software, mas poderia ser aplicado segundo uma abordagem sobre os aspectos informacionais ou comportamentais, por exemplo.



3.3.4. Concepções essenciais e de implementação

A concepção essencial dos requisitos apresenta funções a serem executadas e as informações a serem processadas sem levar em consideração detalhes de implementação. Considerando o exemplo da figura anterior, a concepção essencial da função **ler status** ignora o formato dos dados de status ou o tipo de sensor que será utilizado. A vantagem de realizar a concepção essencial é de deixar em aberto as alternativas de implementação possíveis, o que é adequado para as atividades iniciais do desenvolvimento.

A concepção de implementação dos requisitos de software apresenta a manifestação de funções de processamento e estruturas de informação do sistema real.

Em alguns casos, uma representação física é o ponto de partida da etapa de projeto do software mas, na maioria dos casos, grande parte dos sistemas computacionais é especificada acomodando alguns detalhes de implementação. O conhecimento de alguns destes detalhes pode auxiliar o analista (e, em seguida, o projetista) na definição de restrições impostas ao software pelo sistema.

3.4. Prototipação de software

Em certas situações um modelo do software baseado na especificação de análise pode ser construído.

Outros casos, um protótipo precisa ser usado no início da análise para derivar as exigências do software.

Um panorama de prototipação:

Assim que um pedido para a construção de um software é realizado, alguns passos podem ser aplicados para se a fim de utilizar a prototipação de software:

Passo 1)

Avaliar o pedido de software verificando se é bom candidato à prototipação:

- a) área de aplicação.
 - b) Características do cliente.
 - c) Características do projeto.
- Interação homem-máquina muito forte.
Muito processamento.

Complexidade pode fazer com que o software não seja forte candidato a prototipação. Idéia: dividir a complexidade em partições.

Uma vez que o cliente deve interagir com o protótipo em etapas posteriores é necessário que recursos do cliente sejam comprometidos com avaliação e aprimoramento do protótipo.

Outra questão:

- Experiência com métodos de prototipação ?
- Ferramentas de prototipação estão disponíveis ?

Passo 2)

Dado um projeto aceitável, o analista desenvolve uma representação abreviada dos requisitos – gerando um modelo de requisitos.

Passo 3)

Revisar o modelo de requisitos e criar uma especificação de projeto abreviada.

Passo 4)

Criar, testar e aprimorar o software protótipo.

Passo 5)

Apresentar o protótipo ao cliente. Deixar que o cliente dirija o teste da aplicação e faça sugestões.

Passo 6)

Repetir os passos 4 e 5 até que todas as exigências sejam contempladas.

3.4.1. Métodos e ferramentas de prototipação

Para executar a prototipação de forma efetiva, um protótipo deve ser construído rapidamente para a avaliação do cliente e futuras modificações. Três classes genéricas de métodos e ferramentas estão disponíveis:

a) Técnicas de quarta geração – abrange linguagens de geração de relatórios e de consulta ao banco de dados, geradores de aplicações e programas, linguagens não procedimentais...

b) Componentes reusáveis – os quais permitem a “montagem” do protótipo a partir de “blocos de construção”, dos quais não se conhece necessariamente o seu funcionamento

interno, mas as suas funções e interfaces são dominadas pelo analista. O uso desta técnica só é possível a partir da construção (ou existência) de uma biblioteca de componentes reusáveis, catalogados para posterior recuperação; em alguns casos um software existente pode ser adotados como “protótipo” para a construção de um novo produto, mais competitivo e eficiente, o que define uma forma de reusabilidade de software.

c) Especificações formais e ambientes de prototipação, que surgiram em substituição às técnicas baseadas em linguagem natural. As vantagens da utilização destas técnicas são, basicamente, a representação dos requisitos de software numa linguagem padrão, a geração de código executável a partir da especificação e a possibilidade de validação (da parte do cliente) de modo a refinar os requisitos do software.

3.5. Especificação

O modo de especificação reflete na qualidade da solução. Ela representa o que deve ser implementado.

Princípios da especificação:

- 1) Separe funcionalidade de implementação.
- 2) Uma linguagem de especificação de sistemas orientada ao processo é exigida – expressar o comportamento de certas partes do sistema.
- 3) Deve levar em conta o sistema do qual o software faz parte e o ambiente no qual o sistema vai operar.
- 4) Deve ser um modelo cognitivo. Deve descrever o sistema da forma como ele é percebido pela comunidade de usuários. As entidades que manipula devem pertencer ao domínio do problema. Pessoas, equipamentos, ... que fazem parte do domínio devem ser modelados.
- 6) Deve ser operacional, no sentido de que ela deve permitir, mesmo num nível de abstração elevado, algum tipo de validação;
- 7) Deve ser localizada e fracamente acoplada. O que são requisitos fundamentais para permitir a realização de modificações durante a análise de requisitos.

A figura 16 a seguir apresenta uma proposta de estrutura para o documento de Especificação dos Requisitos do Software.

1.0 - Introdução
2.0 - Descrição da Informação
2.1 - Diagramas de Fluxos de Dados
2.2 - Representação da Estrutura dos Dados
2.3 - Dicionários de Dados
2.4 - Descrição das Interfaces do Sistema
2.5 - Interfaces Internas
3.0 - Descrição Funcional
3.1 - Funções
3.2 - Descrição do Processamento
3.3 - Restrições de Projeto
4.0 - Critérios de Validação
4.1 - Limites de Validação
4.2 - Classes de Testes
4.3 - Expectativas de Resposta do Software
4.4 - Considerações Especiais
5.0 - Bibliografia
6.0 - Apêndices

Figura 16 – Documento da especificação de requisitos

O documento inicia com uma introdução, que apresenta as principais metas do software, descrevendo o seu papel no contexto do sistema global. As informações prestadas nesta seção podem ser, inclusive, inteiramente extraídas do documento de plano de software.

A **Descrição da Informação** deve apresentar informações sobre o problema que o software deve resolver. Os fluxos e a estrutura das informações devem ser descritos nesta seção (utilizando um formalismo adequado, como, por exemplo, os DFDs e os Dicionários de Dados), assim como os elementos relacionados às interfaces internas e externas do software (incluindo hardware, elementos humanos, outros softwares, etc.).

A **Descrição Funcional** apresenta as informações relativas às funções a serem providas pelo software, incluindo uma descrição dos processamentos envolvidos no funcionamento de projeto, detectadas nesta etapa.

Crítérios de Validação, vai estabelecer os parâmetros que permitirão avaliar se a implementação do software vai corresponder à solução desejada para o problema. Definir os critérios de validação significa ter um entendimento completo dos requisitos funcionais e de processamento de informação do software. Uma definição importante a ser encaminhada nesta seção é o conjunto de testes que deverá ser aplicado à implementação para que se tenha uma garantia do funcionamento do software nos moldes estabelecidos pela especificação de requisitos.

Finalmente devem ser registrados neste documento a lista de documentos relacionados ao software a ser desenvolvido (Plano de Software, por exemplo) e uma seção de Apêndices, onde podem ser apresentadas informações mais detalhadas sobre a especificação do software (descrição detalhada de algoritmos, diagramas, gráficos, etc.).

Um outro aspecto interessante, não apresentado na figura, refere-se a quando um software terá características de interatividade com usuários. Neste caso, é interessante que um Manual do Usuário (em versão preliminar) seja elaborado, o qual vai conduzir a duas conseqüências positivas: forçar o analista a enxergar o software do ponto de vista do usuário e permitir ao cliente uma avaliação do que será o software nesta etapa de desenvolvimento.

3.6. Análise estruturada

A Análise Estruturada ou SSA (para *Structured System Analysis*) foi desenvolvida em meados dos anos 70 por Gane, Sarson e De Marco. A técnica SSA é baseada na utilização de uma linguagem gráfica para construir modelos de um sistema, incorporando também conceitos relacionados às estruturas de dados.