

Engenharia de Software

Capítulo 4

Planejamento do Desenvolvimento de Software

1. INTRODUÇÃO

Na maior parte dos trabalhos de engenharia (e os trabalhos de Engenharia de Software não fazem exceção), o tempo é um fator preponderante. Isto é consequência de uma questão cultural (todos nós aprendemos a trabalhar sob o efeito da pressão de tempo), resultante de uma "pressa" nem sempre justificada, conduzindo, na maior parte das vezes, a prazos totalmente irrealísticos, definidos por quem não tem um grau de envolvimento significativo no projeto.

Na prática, a definição de cronogramas dos projetos é feita de maneira arbitrária; os riscos do projeto só são considerados quando eles transformaram-se em problemas reais; a organização da equipe nem sempre é clara e feita de forma consciente.

Neste capítulo serão discutidos alguns pontos fundamentais para que o projeto de desenvolvimento de um software seja conduzido de forma a obter resultados satisfatórios em termos de produtividade (do processo) e qualidade (do produto).

2. ANÁLISE DE RISCOS

A análise dos riscos é uma das atividades essenciais para o bom encaminhamento de um projeto de software. Esta atividade está baseada na realização de quatro tarefas, conduzidas de forma sequencial: a identificação, a projeção, a avaliação e a administração.

2.1. A Identificação dos Riscos

Nesta primeira tarefa, o objetivo é que sejam levantados, da parte do gerente e dos profissionais envolvidos no projeto, todos os eventuais riscos aos quais este será submetido. Nesta identificação, riscos de diferentes naturezas podem ser detectados:

- **riscos de projeto**, os quais estão associados a problemas relacionados ao próprio processo de desenvolvimento (orçamento, cronograma, pessoal, etc...);
- **riscos técnicos**, que consistem dos problemas de projeto efetivamente (implementação, manutenção, interfaces, plataformas de implementação, etc...);
- **riscos de produto**, os quais estão mais relacionados aos problemas que vão surgir para a inserção do software como produto no mercado (oferecimento de um produto que ninguém está interessado; oferecer um produto ultrapassado; produto inadequado à venda; etc...).

A categorização dos riscos, apesar de interessante, não garante a obtenção de resultados satisfatórios, uma vez que nem todos os riscos podem ser identificados facilmente. Uma boa técnica para conduzir a identificação dos riscos de forma sistemática é o estabelecimento de um conjunto de questões (*checklist*) relacionado a algum fator de

risco. Por exemplo, com relação aos riscos de composição da equipe de desenvolvimento, as seguintes questões poderiam ser formuladas:

- são os melhores profissionais disponíveis?
- os profissionais apresentam a combinação certa de capacidades?
- há pessoas suficientes na equipe?
- os profissionais estão comprometidos durante toda a duração do projeto?
- algum membro da equipe estará em tempo parcial sobre o projeto?
- os membros da equipe estão adequadamente treinados?

Em função das respostas a estas perguntas, será possível ao planejador estabelecer qual será o impacto dos riscos sobre o projeto.

2.2. **Projeção dos Riscos**

A projeção ou estimativa de riscos permite definir basicamente duas questões:

- Qual a probabilidade de que o risco ocorra durante o projeto?
- Quais as conseqüências dos problemas associados ao risco no caso de ocorrência do mesmo?

As respostas a estas questões podem ser obtidas basicamente a partir de quatro atividades:

- estabelecimento de uma escala que reflita a probabilidade estimada de ocorrência de um risco;
- estabelecimento das conseqüências do risco;
- estimativa do impacto do risco sobre o projeto e sobre o software (produtividade e qualidade);
- anotação da precisão global da projeção de riscos.

A escala pode ser definida segundo várias representações (booleana, qualitativa ou quantitativa). No limite cada pergunta de uma dada checklist pode ser respondida com "sim" ou "não", mas nem sempre esta representação permite representar as incertezas de modo realístico.

Uma outra forma de se representar seria utilizar uma escala de probabilidades qualitativas, onde os valores seriam: altamente improvável, improvável, moderado, provável, altamente provável. A partir destes "valores" qualitativos, seria possível realizar cálculos que melhor expressassem as probabilidades matemáticas de que certo risco viesse a ocorrer.

O próximo passo é então o levantamento do impacto que os problemas associados ao risco terão sobre o projeto e sobre o produto, o que permitirá priorizar os riscos. Pode-se destacar três fatores que influenciam no impacto de um determinado risco: a sua natureza, o seu escopo e o período da sua ocorrência.

A natureza do risco permite indicar os problemas prováveis se ele ocorrer (por exemplo, um risco técnico como uma mal definição de interface entre o software e o hardware do cliente levará certamente a problemas de teste e integração). O seu escopo permite indicar de um lado qual a gravidade dos problemas que serão originados e qual a parcela do projeto que será atingida. O período de ocorrência de um risco permite uma reflexão sobre quando ele poderá ocorrer e por quanto tempo.

Estes três fatores definirão a importância do risco para efeito de priorização...um fator de risco de elevado impacto pode ser pouco considerado a nível do gerenciamento de projeto se a sua probabilidade de ocorrência for baixa. Por outro lado fatores de risco com alto peso de impacto e com probabilidade de ocorrência de moderada a alta e fatores de risco com baixo peso de impacto com elevada probabilidade de ocorrência não devem ser desconsiderados, devendo ser processados por todas atividades de análise de risco.

2.3. Avaliação dos Riscos

O objetivo da atividade de avaliação dos riscos é processar as informações sobre o fator de risco, o impacto do risco e a probabilidade de ocorrência. Nesta avaliação, serão checadas as informações obtidas na projeção de riscos, buscando priorizá-los e definir formas de controle destes ou de evitar a ocorrência daqueles com alta probabilidade de ocorrência.

Para tornar a avaliação eficiente, deve ser definido um **nível de risco referente**. Exemplos de níveis referentes típicos em projetos de Engenharia de Software são: o custo, o prazo e o desempenho. Isto significa que se vai ter um nível para o excesso de custo, para a ultrapassagem de prazo e para a degradação do desempenho ou qualquer combinação dos três. Desta forma, caso os problemas originados por uma combinação de determinados riscos provoquem a ultrapassagem de um ou mais desses níveis, o projeto poderá ser suspenso. Normalmente, é possível estabelecer um limite, denominado de ponto referente (*breakpoint*) onde tanto a decisão de continuar o projeto ou de abandoná-lo podem ser tomadas.

A figura 4.1 ilustra esta situação, na qual dois níveis de risco referente são considerados (o custo e o prazo). Se uma combinação de riscos conduzir a uma situação onde os limites de ultrapassagem de custo e/ou de prazo estejam acima do limite (delimitado pela curva na figura), o projeto será abandonado (área em cinza escuro). No breakpoint, as decisões de continuar o projeto ou abandoná-lo têm igual peso.

2.4. Administração e Monitoração dos Riscos

Uma vez avaliados os riscos de desenvolvimento, é importante que medidas sejam tomadas para evitar a ocorrência dos riscos ou que ações sejam definidas para a eventualidade da ocorrência dos riscos. Este é o objetivo da tarefa de Administração e monitoração dos riscos. Para isto, as informações mais importantes são aquelas obtidas na tarefa anterior, relativa à *descrição*, *probabilidade de ocorrência* e *impacto sobre o processo*, associadas a cada fator de risco.

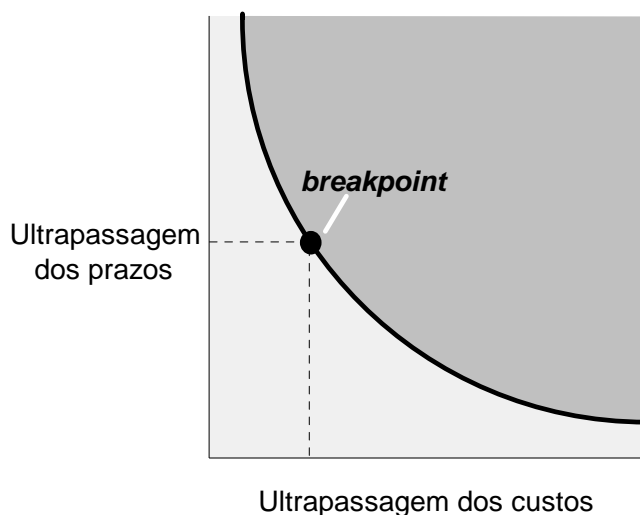


Figura 4.1 - Ilustração de uma situação de definição de dois níveis de riscos.

Por exemplo, considerando a alta rotatividade de pessoal numa equipe um fator de risco, com base em dados de projetos passados, obtém-se que a probabilidade de ocorrência deste risco é de 0,70 (muito elevada) e que a sua ocorrência pode aumentar o prazo do projeto em 15% e o seu custo global em 12%. Sendo assim, pode-se propor as seguintes ações de administração deste fator de risco:

- reuniões com os membros da equipe para determinar as causas da rotatividade de pessoal (más condições de trabalho, baixos salários, mercado de trabalho competitivo, etc...);
- tomada de providências para eliminar ou reduzir as causas "controláveis" antes do início do projeto;
- no início do projeto, pressupor que a rotatividade vai ocorrer e prever a possibilidade de substituição de pessoas quando estas deixarem a equipe;
- organizar equipes de projeto de forma que as informações sobre cada atividade sejam amplamente difundidas;
- definir padrões de documentação para garantir a produção de documentos de forma adequada;
- realizar revisões do trabalho entre colegas de modo que mais de uma pessoa esteja informado sobre as atividades desenvolvidas;
- definir um membro da equipe que possa servir de *backup* para o profissional mais crítico.

É importante observar que a implementação destas ações pode afetar também os prazos e o custo global do projeto. Isto significa que é necessário poder avaliar quando os custos destas ações podem ser ultrapassados pela ocorrência dos fatores de risco.

Num projeto de grande dimensão, de 30 a 40 fatores de risco podem ser identificados; se para cada fator de risco, sete ações forem definidas, a administração dos riscos pode tornar-se um projeto ela mesma. Por esta razão, é necessário que uma priorização dos riscos seja efetuada, atingindo normalmente, a 20% de todos os fatores levantados.

Todo o trabalho efetuado nesta tarefa é registrado num documento denominado **Plano de Administração e Monitoração de Riscos**, o qual será utilizado posteriormente pelo gerente de projetos (particularmente, para a definição do Plano de Projeto, que é gerado ao final da etapa de planeamento). Uma ilustração sintetizando os passos essenciais desta tarefa é apresentada à figura 4.2.

3. DEFINIÇÃO DE UM CRONOGRAMA

A definição de um cronograma pode ser obtida segundo duas diferentes abordagens:

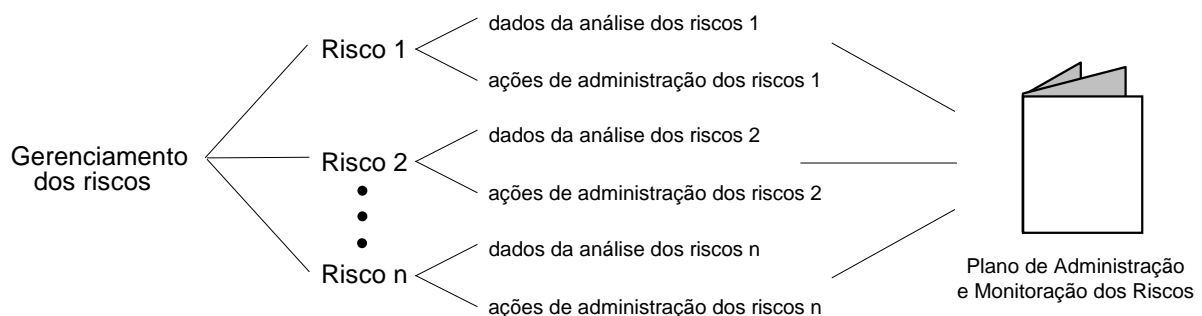


Figura 4.2 - Síntese da tarefa de Administração e Monitoração dos Riscos.

- a primeira, é baseada na definição prévia de um prazo de entrega do software; neste caso, o planeamento deve ser feito de modo a distribuir os esforços ao longo do prazo estabelecido;
- a segunda, está relacionada a uma discussão de limites cronológicos aproximados para cada etapa do desenvolvimento, sendo que o prazo de entrega do software seja estabelecido a partir de técnicas de planeamento da Engenharia de Software.

É evidente que a primeira abordagem é a mais encontrada nos projetos de software.

Um cronograma bem definido pode trazer enormes benefícios a um projeto de software, sendo às vezes mais importante que a própria definição de custos. Numa visão de desenvolvimento de software como produto, um adicional nos custos de produção pode ser absorvido por uma redefinição nos preços ou pela amortização em função de um elevado número de vendas. Já, um acréscimo imprevisto no prazo de entrega de um software pode provocar grandes prejuízos ao produto, como por exemplo: a queda no impacto de mercado, insatisfação dos clientes, elevação de custos internos, etc...

Quando a tarefa é fixar prazos para os projetos de software, diversas questões podem ser formuladas:

- como relacionar o tempo cronológico com o esforço humano?
- que tarefas e que grau de paralelismo podem ser obtidos?
- como medir o progresso do processo de desenvolvimento (indicadores de progresso)?
- como o esforço pode ser distribuído ao longo do processo de Engenharia de Software?
- que métodos estão disponíveis para a determinação de prazos?
- como representar fisicamente o cronograma e como acompanhar o progresso a partir do início do projeto?

As seções que seguem discutirão algumas destas questões.

3.1. As relações pessoas-trabalho

Em primeiro lugar, é preciso dizer que, à medida que um projeto ganha dimensão, um maior número de pessoas deve estar envolvida. Além disso, deve-se tomar o cuidado de não levar a sério o mito, apresentado no capítulo I, de que *"Se o desenvolvimento do software estiver atrasado, basta aumentar a equipe para honrar o prazo de desenvolvimento."*

Deve-se considerar, ainda que, quanto maior o número de pessoas, maior o número de canais de comunicação, o que normalmente requer esforço adicional e, conseqüentemente, tempo adicional.

A experiência tem mostrado que pode ser mais interessante realizar o desenvolvimento de um software com uma equipe com menos pessoas por um período maior de tempo (quando as restrições de prazo de entrega assim o permitem) do que o contrário.

3.2. A Definição de tarefas

Uma das vantagens de se ter uma equipe com mais de uma pessoa são as possibilidades de se realizar determinadas tarefas em paralelo. O paralelismo de tarefas é um mecanismo interessante como forma de economia de tempo na realização de qualquer trabalho de engenharia. Para isto, basta que um gerenciamento dos recursos necessários a cada tarefa (recursos humanos, recursos de software, etc...) seja feito de forma eficiente.

Sendo assim, as tarefas a serem realizadas durante um projeto de desenvolvimento de software podem ser expressas na forma de uma rede (rede de tarefas) a qual apresenta todas as tarefas do projeto, assim como as suas relações em termos de realização (paralelismo, seqüência, pontos de encontro, etc... Um exemplo desta representação é apresentado na figura 4.3.

Nesta figura, é possível verificar que existem indicadores de progresso (representados por um símbolo "※") situados ao longo do processo e que permitem ao gerente uma avaliação do estado de evolução do processo. Um indicador de progresso é considerado atingido ou satisfeito quando a documentação produzida com relação a este indicador é revisada e aprovada.

3.3. A DISTRIBUIÇÃO DO ESFORÇO

Normalmente, as técnicas de estimativas para projetos de software conduzem a uma definição do esforço em termos do número de homens-mês ou homens-ano necessárias para realizar o desenvolvimento do projeto. Uma proposta de distribuição de esforço bastante utilizada nos projetos é aquela ilustrada na figura 4.4, a qual é baseada numa regra anteriormente denominada *Regra 40-20-40*, a qual sugere, como etapas onde o esforço deve ser maior, as dos extremos do processo de desenvolvimento (análise/projeto e testes), a codificação sendo a tarefa que deve envolver a menor concentração de esforço. Isto pode ir contra o grau de importância em termos de esforço que muitos desenvolvedores dão a cada uma destas atividades.

É evidente que estes valores não podem ser levados à risca para todos os projetos de software, sendo que as características de cada projeto vai influenciar na parcela de esforço a ser dedicada a cada etapa.

No entanto, é importante entender a razão pela qual o esforço dedicado à etapa de codificação aparece com menor intensidade que os demais.

Na realidade, se a etapa de projeto foi realizada utilizando as boas regras da Engenharia de Software, a etapa de codificação será, conseqüentemente, minimizada em termos de esforço.

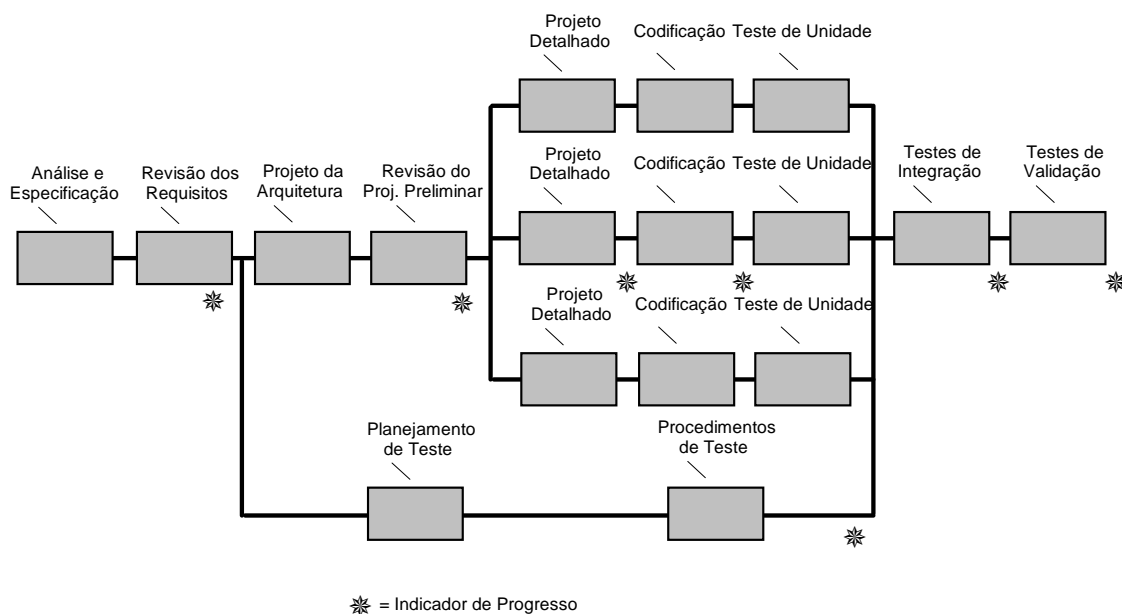


Figura 4.3 - Exemplo de uma Rede de Tarefas.

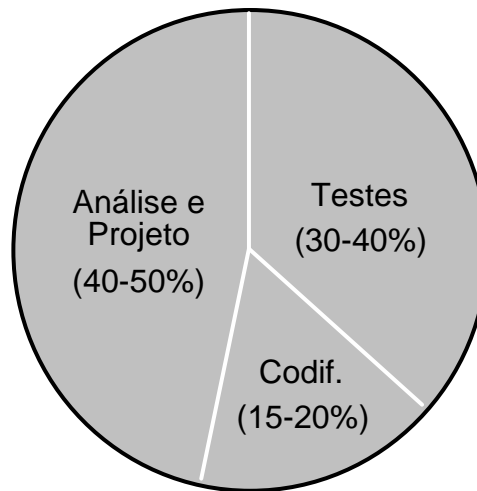


Figura 4.4 - Proposta de distribuição dos esforços de desenvolvimento.

Nos processos de desenvolvimento em que as etapas de projeto/análise não são enfatizadas, existe uma razão porque a etapa de codificação acaba exigindo maior esforço que estas etapas... atividades de análise e decisões de projeto acabam sendo efetuadas na etapa de codificação, o que certamente pode corresponder a um custo de desenvolvimento superior ao de um processo onde isto fosse feito segundo uma metodologia mais condizente com o que prega a Engenharia de Software.

A etapa de testes, por sua vez, pode representar de 30 a 40% do esforço total do projeto. Na realidade, o que vai determinar o esforço a ser dispendido com os testes serão os requisitos do próprio software. Softwares concebidos para aplicações críticas, onde as consequências dos erros podem ser catastróficas (seja em termos de perdas humanas ou materiais), vão exigir um esforço muito maior em termos de teste do que as aplicações mais clássicas (de automação de escritório, por exemplo).

3.3. A Representação do Cronograma

A definição do cronograma é uma das tarefas mais difíceis de se definir na etapa de planejamento do software. O planejador deve levar em conta diversos aspectos relacionados ao desenvolvimento, como: a disponibilidade de recursos no momento da execução de uma dada tarefa, as interdependências das diferentes tarefas, a ocorrência de possíveis estrangulamentos do processo de desenvolvimento e as operações necessárias para agilizar o processo, identificação das principais atividades, revisões e indicadores de progresso do processo de desenvolvimento, etc...

A forma de representação dos cronogramas depende da política de desenvolvimento adotada, mas pode ser apresentada, numa forma geral por um documento do tipo apresentado na figura 4.5.

As unidades de tempo consideradas no projeto (dias, semanas, meses, etc...) são anotadas na linha superior da folha de cronograma. As tarefas do projeto, as atividades e os indicadores de progresso são definidos na coluna da esquerda. O traço horizontal permite indicar o período durante o qual determinada atividade será realizada, o tempo necessário sendo medido em unidades de tempo consideradas. Quando atividades puderem ser realizadas em paralelo, os traços vão ocupar unidades de tempo comuns.

O cronograma deve explicitar as atividades relevantes do desenvolvimento e os indicadores de progresso associados. É importante que os indicadores de progresso sejam representados por resultados concretos (por exemplo, um documento).

A disponibilidade de recursos deve também ser representada ao longo do cronograma. O impacto da indisponibilidade dos recursos no momento em que eles são necessários deve também ser representado, se possível, no cronograma.

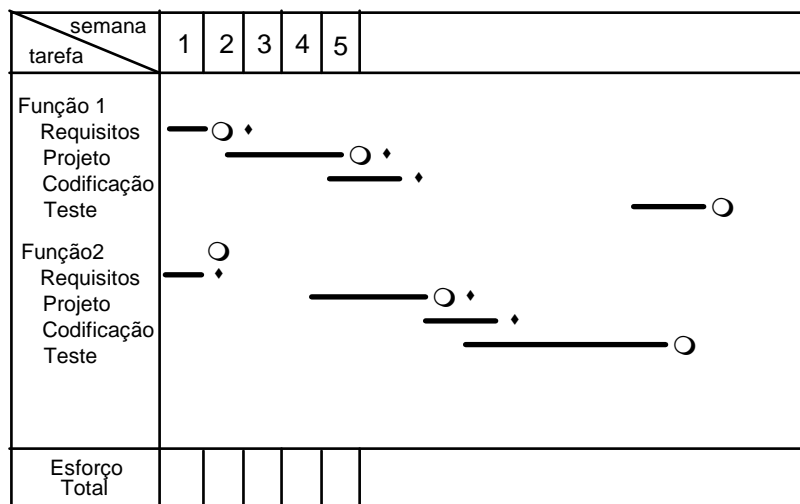


Figura 4.5 - Representação geral de um cronograma de desenvolvimento.

4. AQUISIÇÃO DE SOFTWARE

Embora a Engenharia de Software sugira o desenvolvimento de software, em alguns casos pode ser mais interessante adquirir o software do que desenvolvê-lo. Esta é uma decisão a qual o gerente de um projeto pode ter de tomar no contexto de um projeto. Com relação à aquisição de software, diversas ações podem ser tomadas:

- adquirir (ou licenciar) um pacote comercial que atenda às especificações estabelecidas;
- adquirir um pacote comercial e modificá-lo de forma a que o novo software atenda às especificações de projeto;
- encomendar o software a terceiros para que este atenda às especificações.

Os procedimentos a efetuar para a aquisição de um software vão depender da criticidade das especificações e do seu custo. No caso de um software de custo relativamente baixo (um software para PC, por exemplo), pode ser mais interessante adquiri-lo e fazer uma análise de adequação às especificações de projeto.

No caso de softwares mais caros, uma análise mais cuidadosa se faz necessária, sendo que os procedimentos podem ser os seguintes:

- desenvolver uma especificação funcional e de desempenho para o software a projetar, estabelecendo, quando possível, valores mensuráveis;
- realizar a estimativa do custo interno de desenvolvimento;
- escolher um conjunto de pacotes comerciais que poderiam atender às especificações;
- desenvolver um esquema de análise comparativa que permita confrontar as funções-chave das soluções alternativas;
- avaliar cada pacote com base na qualidade do produto, no suporte do vendedor (garantia), reputação do vendedor e direcionamento do produto;
- entrevistar usuários do software, colhendo suas opiniões sobre os pacotes.

A partir dos resultados obtidos, a decisão entre comprar ou desenvolver o software vai estar baseada nos seguintes critérios:

- a data de entrega do produto precede a data de finalização do produto se desenvolvido internamente?

- o custo de aquisição mais o custo de "customização" será inferior ao custo de desenvolvimento interno?
- o custo de suporte externo (contrato de manutenção) é inferior ao custo do suporte interno?

5. REENGENHARIA

Um outro problema importante a ser tratado pela Engenharia de Software são os antigos pacotes de software que são fundamentais à realização de negócios de uma empresa e que oferecem grandes dificuldades de manutenção. Historicamente, a manutenção destes pacotes foi realizada com base em verdadeiros "remendos", sem nenhuma documentação, resultando muitas vezes em programas ineficientes e com alta taxa de falhas. Estes fatores conduziram a uma situação na qual os custos de manutenção de tais sistemas não justificam mais os benefícios que tais alterações podem trazer.

Por outro lado, um processo de reengenharia do software pode aparecer como uma alternativa de baixo custo à manutenção do software. Para isto, é importante que os seguintes procedimentos sejam encaminhados:

- selecionar os programas que estejam sendo bastante utilizados no momento e que continuarão a ser utilizados nos próximos 5 a 10 anos;
- estimar o custo anual de manutenção dos programas selecionados, incluindo correção de erros, adaptação e melhorias funcionais;
- organizar, por ordem de prioridade, os programas selecionados, registrando os custos levantados no procedimento anterior;
- estimar os custos para realizar a reengenharia dos programas selecionados e estimar os custos de manutenção do programa após realizada a reengenharia;
- realizar uma análise comparativa de custos para cada programa;
- calcular o tempo necessário para o retorno de investimento da reengenharia;
- levar em consideração algumas questões importantes que resultam da reengenharia, como a melhor confiabilidade do sistema, melhor desempenho e melhores interfaces;
- obter a aprovação da empresa para realizar a reengenharia de um programa;
- com base nos resultados obtidos desta experiência, desenvolver uma estratégia de reengenharia dos demais programas.

6. PLANEJAMENTO ORGANIZACIONAL

Existe uma grande diversidade no que diz respeito aos modos de organização das equipes de desenvolvimento de software. A escolha da forma como a equipe de desenvolvimento vai ser organizada é um dos pontos que deve ser definido nesta etapa. Considerando que um processo de desenvolvimento de software vai ocupar uma equipe de n pessoas com uma duração de k anos, é possível comentar algumas opções organizativas:

- ① n indivíduos são alocados a m diferentes tarefas com pequeno grau de interação, sendo que a coordenação da equipe fica a cargo do gerente de projeto;
- ② n indivíduos são alocados a m diferentes tarefas, com $m \leq n$, formando equipes informais de desenvolvimento, com um responsável ad-hoc de cada equipe, sendo que a coordenação entre as equipes é da responsabilidade do gerente do projeto;
- ③ n indivíduos são organizados em k equipes, cada equipe sendo alocada para uma ou mais tarefas; a organização de cada equipe é específica a ela própria, a coordenação ficando a cargo da equipe e do gerente do projeto.

Sem discutir em detalhes os argumentos contrários ou a favor de cada uma das opções, é importante dizer que a constituição em equipes formais de desenvolvimento (como sugere a opção ③) é mais produtiva.

O principal objetivo da formação de equipes é o desenvolvimento de um conceito de projeto como sendo o resultado de uma reunião de esforços. A criação de equipes evita o sentimento de "ego-programação" que pode atingir as pessoas envolvidas num desenvolvimento, transformando o "meu programa" em "nosso programa".

De um ponto de vista geral, pode-se estabelecer uma referência no que diz respeito à organização de uma equipe de desenvolvimento de software, sendo que o número de equipes e o número de membros de cada equipe vai variar em função da grandeza do projeto.

O núcleo de uma equipe vai ser composto dos seguintes elementos:

- um **engenheiro sênior** (ou programador chefe), responsável do planejamento, coordenação e supervisão de todas as atividades relativas ao desenvolvimento do software;
- o **peçoal técnico**, de dois a cinco membros que realizam as atividades de análise e desenvolvimento;
- um **engenheiro substituto**, que atua no apoio ao engenheiro sênior e que pode, eventualmente, substituí-lo sem grandes prejuízos ao desenvolvimento do software.

Além deste núcleo, a equipe de desenvolvimento pode ainda receber a colaboração dos seguintes elementos:

- um **conjunto de especialistas** (telecomunicações, bancos de dados, interface homem-máquina, etc...);
- **peçoal de apoio** (secretárias, editores técnicos, desenhistas, etc...);
- um **bibliotecário**, o qual será responsável da organização e catalogação de todos os componentes do produto de software (documentos, listagens, mídia magnética, coleta de dados relativos ao projeto, módulos reutilizáveis, etc...).

7. MÉTRICA DO SOFTWARE

7.1. Importância da Métrica

Em primeiro lugar, é importante estar ciente que as medidas são uma forma clara de avaliação da produtividade no desenvolvimento de software.

Sem informações quantitativas a respeito do processo de desenvolvimento de softwares por parte de uma empresa ou equipe de desenvolvedores de produto, é impossível tirar qualquer conclusão sobre de que forma está evoluindo a produtividade (se é que está evoluindo!?!).

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 importantes fatores da Engenharia de Software.

Particularmente no que diz respeito à qualidade, é possível, com uma avaliação quantitativa deste parâmetro, promover-se pequenos ajustes no processo de desenvolvimento como forma de eliminar ou reduzir as causas de problemas que afetam de forma significativa o projeto de software.

No que diz respeito aos desenvolvedores, a obtenção de medidas podem auxiliar a responder com precisão uma série de perguntas que estes elementos se fazem a cada projeto:

- que tipos de requisitos são os mais passíveis de mudanças?
- quais módulos do sistema são os mais propensos a erros?
- quanto de teste deve ser planejado para cada módulo?

- quantos erros (de tipos específicos) pode-se esperar quando o teste se iniciar?

7.2. MEDIDAS DE SOFTWARE

Na área de engenharia, a medição tem sido um aspecto de grande importância, sendo que poderíamos desfilar uma fila interminável de grandezas as quais sofrem este tipo de tratamento: dimensões físicas, peso (ou massa), temperatura, tensões e correntes elétrica, etc... No mundo dos computadores, alguns parâmetros são quantificados como forma de expressar as potencialidades de determinadas máquinas, tais como a capacidade de um processador de executar um certo número de instruções por segundo (MIPS), as capacidades de armazenamento (Mbytes), a frequência do clock do processador (MHz), etc...

No caso particular do software, existem diversas razões para que a realização de medições seja um item de importância:

- quantizar a qualidade do software como produto;
- avaliar a produtividade dos elementos envolvidos no desenvolvimento do produto;
- avaliar os benefícios de métodos e ferramentas para o desenvolvimento de software;
- formar uma base de dados para as estimativas;
- justificar o pleito e aquisição de novas ferramentas e/ou treinamento adicional para membros da equipe de desenvolvimento.

De forma análoga a outras grandezas do mundo físico, as medições de software podem ser classificadas em duas categorias principais:

- as **medições diretas**, por exemplo, o número de linhas de código (LOC) produzidas, o tamanho de memória ocupado, a velocidade de execução, o número de erros registrados num dado período de tempo, etc...
- as **medições indiretas**, as quais permitem quantizar aspectos como a funcionalidade, complexidade, eficiência, manutenibilidade, etc...

As medições diretas, tais quais aquelas exemplificadas acima, são de obtenção relativamente simples, desde que estabelecidas as convenções específicas para isto. Por outro lado, aspectos como funcionalidade, complexidade, eficiência, etc..., são bastante difíceis a quantizar.

As medições de software podem ser organizadas em outras classes, as quais serão definidas a seguir:

- **métricas da produtividade**, baseadas na saída do processo de desenvolvimento do software com o objetivo de avaliar o próprio processo;
- **métricas da qualidade**, que permitem indicar o nível de resposta do software às exigências explícitas e implícitas do cliente;
- **métricas técnicas**, nas quais encaixam-se aspectos como funcionalidade, modularidade, manutenibilidade, etc...

Sob uma outra ótica, é possível definir uma nova classificação das medições:

- **métricas orientadas ao tamanho**, baseadas nas medições diretas da Engenharia de Software;
- **métricas orientadas à função**, que oferecem medidas indiretas;
- **métricas orientadas às pessoas**, as quais dão indicações sobre a forma como as pessoas desenvolvem os programas de computador.

7.3. Métricas orientadas ao tamanho

Esta classe abrange todas as possíveis medidas obtidas diretamente do software. Um exemplo de tal classe de medidas é mostrado na tabela a seguir. Como pode ser verificado na tabela desta figura, cada projeto é caracterizado por um conjunto de parâmetros que permite obter diversas informações tanto sobre a produtividade do processo quanto sobre a qualidade do software obtido. Pode-se, então definir algumas grandezas tais como:

$$\text{Produtividade} = \text{KLOC/pessoa-mês}$$

$$\text{Qualidade} = \text{erros/KLOC}$$

$$\text{Custo} = \$/\text{KLOC}$$

$$\text{Documentação} = \text{págs/KLOC}$$

A despeito da facilidade em sua obtenção, existe muita discussão em torno das métricas orientadas ao tamanho. Um caso bastante discutido é o da utilização do número de linhas de código como medida de dimensão. Os que defendem o uso desta métrica, afirmam que é um aspecto de fácil obtenção e, portanto, de fácil aplicação a qualquer projeto de software, além de destacar a quantidade de informação existente com base nesta métrica.

Projeto	Esforço	Custo	KLOC	Págs	Erros	Pessoas
AAA-01	24	168	12,1	365	29	3
CCC-04	62	440	27,2	1224	86	5
FFF-03	43	314	20,2	1050	64	6

Os que discutem o uso desta informação, alertam para a forte dependência da linguagem no uso desta técnica. Além disso, a métrica orientada ao tamanho tende a penalizar os programas bem estruturados e que tenham feito economia de software. É uma métrica que, na verdade, deixa de ser precisa principalmente por causa dos diversos novos fatores introduzidos pelas linguagens de programação mais recentes, além de ser de difícil aplicação para efeito de estimativas, uma vez que é necessário descer a um nível de detalhe que não é desejável na etapa de planejamento do projeto.

7.4. Métricas orientadas à função

Esta classe é baseada em medidas indiretas do software e do processo utilizado para obtê-lo. Em lugar da contagem do número de linhas de código, esta métrica leva em conta aspectos como a funcionalidade e a utilidade do programa.

Uma abordagem proposta nesta classe é a do **ponto-por-função** (*function point*), a qual é baseada em medidas indiretas sobre a complexidade do software. A figura 2.2 mostra uma tabela que deve ser preenchida para que se possa obter uma medida sobre a complexidade do software. Os valores a serem computados são os seguintes: número de entradas do usuário, número de saídas do usuário, número de consultas do usuário, número de arquivos e número de interfaces externas.

Uma vez computados todos os dados, um valor de complexidade é associado a cada contagem. Feito isto, obtém-se o valor de pontos-por-função utilizando-se a seguinte fórmula:

$$FP = \text{Contagem Total} * [0,65 + 0,01 * \text{SOMA}(Fi)]$$

Na fórmula acima, além da *Contagem Total*, obtida diretamente da tabela da figura 4.6, F_i (para $i = 1$ a 14) corresponde a um conjunto de valores de ajuste de complexidade.

Parâmetro	Contagem		Fator de ponderação			
			Simple	Médio	Complexo	
Ent. usuário	<input type="text"/>	x	3	4	5	<input type="text"/>
Saídas usuário	<input type="text"/>	x	4	5	7	<input type="text"/>
Consult. usuário	<input type="text"/>	x	3	4	6	<input type="text"/>
Arquivos	<input type="text"/>	x	7	10	15	<input type="text"/>
Interfaces ext.	<input type="text"/>	x	5	7	10	<input type="text"/>
Contagem total =						<input type="text"/>

Figura 4.6 - Computação da métrica *ponto-por-função*.

Estes valores são obtidos a partir de respostas dadas a perguntas feitas a respeito da complexidade do software. Os parâmetros da equação acima, assim como os pesos apresentados na figura 4.6 (relativos aos níveis de complexidade) são obtidos empiricamente.

Uma vez computados, os pontos por função podem ser utilizados de forma análoga ao número de linhas de código (LOC) para a obtenção de medidas de qualidade, produtividade e outros atributos:

$$\text{Produtividade} = FP/\text{pessoa-mês}$$

$$\text{Qualidade} = \text{erros}/FP$$

$$\text{Custo} = \$/FP$$

$$\text{Documentação} = \text{págs}/FP$$

8. O PLANO DE SOFTWARE

Ao final desta etapa, um documento de **Plano de Software** deverá ser gerado, o qual deverá ser revisto para servir de referência às etapas posteriores. Ele vai apresentar as informações iniciais de custo e cronograma que vão nortear o desenvolvimento do software. Ele consiste de um documento relativamente breve, o qual é encaminhado às diversas pessoas envolvidas no desenvolvimento do software.

Dentre as informações a serem contidas neste documento, é possível destacar:

- o contexto e os recursos necessários ao gerenciamento do projeto, à equipe técnica e ao cliente;
- a definição de custos e cronograma que serão acompanhados para efeito de gerenciamento;
- dá uma visão global do processo de desenvolvimento do software a todos os envolvidos.

A figura 4.7 apresenta uma possível estrutura para este documento. A apresentação dos custos e cronograma pode diferir dependendo de quem será o leitor do documento.

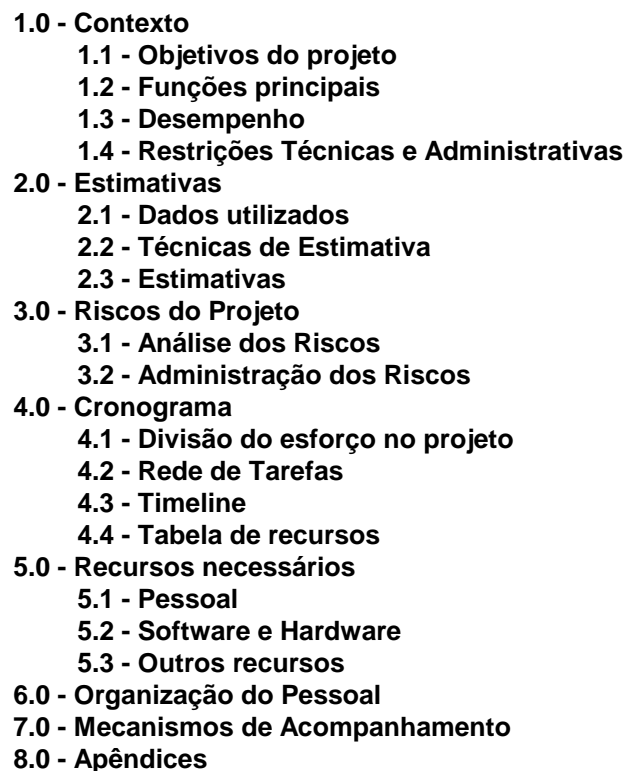
- 
- 1.0 - Contexto**
 - 1.1 - Objetivos do projeto
 - 1.2 - Funções principais
 - 1.3 - Desempenho
 - 1.4 - Restrições Técnicas e Administrativas
 - 2.0 - Estimativas**
 - 2.1 - Dados utilizados
 - 2.2 - Técnicas de Estimativa
 - 2.3 - Estimativas
 - 3.0 - Riscos do Projeto**
 - 3.1 - Análise dos Riscos
 - 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 4.7 - Proposta de estrutura para o documento de Plano do Software.

O Plano de Software não necessita ser um documento extenso e complexo. O seu objetivo é auxiliar a análise de viabilidade dos esforços de desenvolvimento. Este documento está associado aos conceitos de "*o que*", "*quanto*" e "*quão longo*" associados ao desenvolvimento. As etapas mais à frente vão estar associadas ao conceito de "*como*".