Sistema Aberto de Educação

SABE
Sistema Aberto de Educação

Guia de Estudo

Engenharia de Software II

unis
CENTRO UNIVERSITÁRIO DO SUL DE MINAS

Instituição Credenciada pelo MEC
Centro Universitário do Sul de Minas

CREDENCIADO PELO MEC PORTARIA 4.385/05

SABE – Sistema Aberto de Educação

Av. Cel. José Alves, 256 - Vila Pinto
Varginha - MG - 37010-540
Tele: (35) 3219-5204 - Fax - (35) 3219-5223

Instituição Credenciada pelo MEC – Portaria 4.385/05

Centro Universitário do Sul de Minas - UNIS/MG
Unidade de Gestão da Educação a Distância – GEaD

Mantida pela
Fundação de Ensino e Pesquisa do Sul de Minas - FEPESMIG

Varginha/MG

Revisão pelo Novo Acordo Ortográfico
Da Língua Portuguesa

005.1

S719E SOUZA, Rafael Rodrigues de.

Guia de Estudo – Engenharia de Software II. Rafael Rodrigues de Souza. Varginha: GEaD-UNIS/MG, 2009.

48p.

1. Software. 2. Análise de Sistemas. 3. Ciclo de Vida do Software. I. Título.

REITOR
Prof. Ms. Stefano Barra Gazzola

GESTOR
Prof. Ms. Tomás Dias Sant' Ana

Supervisor Técnico
Prof. Ms. Wanderson Gomes de Souza

Coord. do Núcleo de Recursos Tecnológicos
Profª. Simone de Paula Teodoro Moreira

Coord. do Núcleo de Desenvolvimento Pedagógico
Profª. Vera Lúcia Oliveira Pereira

Coord. do Núcleo de Comunicação Relacionamento
Prof. Ms. Renato de Brito

Revisão ortográfica / gramatical
Profª. Ms. Silvana Prado

Design/diagramação
Prof. César dos Santos Pereira

Equipe de Tecnologia Educacional
Prof. Celso Augusto dos Santos Gomes
Profª. Débora Cristina Francisco Barbosa
Jacqueline Aparecida da Silva

Autor(a)
RAFAEL RODRIGUES DE SOUZA

Técnico em Processamento de Dados (1996) pelo Colégio Batista, Bacharel em Ciência da Computação (2001) pelo UNIS-MG, Mestre em Administração e Desenvolvimento Organizacional (2007) pela FACECA, atua como docente em cursos de graduação e pós-graduação no UNIS, FACECA e UNINCOR, é sócio da VSoftware, empresa especializada no desenvolvimento de softwares gerenciais utilizando as linguagens Visual Basic e C#.

TABELA DE ÍCONES

	**REALIZE**. Determina a existência de atividade a ser realizada. Este ícone indica que há um exercício, uma tarefa ou uma prática para ser realizada. Fique atento a ele.
	**PESQUISE**. Indica a exigência de pesquisa a ser realizada na busca por mais informação.
	**PENSE**. Indica que você deve refletir sobre o assunto abordado para responder a um questionamento.
	**CONCLUSÃO**. Todas as conclusões sejam de ideias, partes ou unidades do curso virão precedidas desse ícone.
	**IMPORTANTE**. Aponta uma observação significativa. Pode ser encarado como um sinal de alerta que o orienta para prestar atenção à informação indicada.
	**HIPERLINK**. Indica um link (ligação), seja ele para outra página do módulo impresso ou endereço de Internet.
	**EXEMPLO.** Esse ícone será usado sempre que houver necessidade de exemplificar um caso, uma situação ou conceito que está sendo descrito ou estudado.
	**SUGESTÃO DE LEITURA**. Indica textos de referência utilizados no curso e também faz sugestões para leitura complementar.
	**APLICAÇÃO PROFISSIONAL.** Indica uma aplicação prática de uso profissional ligada ao que está sendo estudado.
	**CHECKLIST ou PROCEDIMENTO.** Indica um conjunto de ações para fins de verificação de uma rotina ou um procedimento (passo a passo) para a realização de uma tarefa.
	**SAIBA MAIS.** Apresenta informações adicionais sobre o tema abordado de forma a possibilitar a obtenção de novas informações ao que já foi referenciado.
	**REVENDO.** Indica a necessidade de rever conceitos estudados anteriormente.

SUMÁRIO

APRESENTAÇÃO

Olá!!!

Espero que esteja animado e empolgado para o início deste novo semestre. Veremos nesta disciplina de Engenharia de Software II conteúdos ainda mais focados na realidade do mercado de trabalho, abordando metodologias que podem ser usadas na condução de projetos, análise e projeto orientado a objetos, UML, dentre outros tópicos.

Para desenvolver esta disciplina trabalharemos com muitos “casos” que espero que ajude vocês em sua atuação profissional. É com o espírito renovado e com vontade de aprender é que espero que comece esta nova jornada. Vamos aos estudos!!!

1. PLANEJAMENTO DO DESENVOLVIMENTO DE SOFTWARE

1.1 Introdução

Um dos focos de nosso conteúdo neste semestre é o projeto de software, vamos começar então falando sobre o planejamento do desenvolvimento de software, tentarei mostrar os principais aspectos e depois entraremos em contato com as metodologias e técnicas propriamente ditas.

Você já percebeu que o tempo todo está planejando???

Em nosso cotidiano o tempo é primordial, planejamos a hora de sair de casa para chegar a tempo no trabalho, você calcula o tempo que irá gastar para realizar as atividades e entregá-las a tempo para o professor e até nas atividades mais corriqueiras planejamos.

Agora uma pergunta, porque alguns prazos apesar de termos calculado mentalmente não são cumpridos?

Se você pensou em falta de técnica ou falta de métodos este realmente é um dos principais fatores. Claro que com o tempo você já calcula melhor atividades já feitas anteriormente. Por exemplo, se eu lhe perguntar quanto tempo você gasta para ir de sua casa até a padaria mais próxima você provavelmente me dará esta informação com uma margem pequena de erro, já se perguntar quanto tempo gasta para ir de sua casa até um ponto nunca visitado terá mais dificuldade em fazer esta previsão.

O ponto principal de nossa discussão é com relação a tempo e custo, na maior parte dos trabalhos de engenharia de software, o tempo é um fator extremamente importante. 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 irrealistas, que desde o início já se sabia que não era possível de cumprir.

Em muitos casos isto acontece porque foram definidos por quem não tem um grau de envolvimento significativo no projeto, ou seja, não tem informações suficientes para calcular este prazo dentro de um limite aceitável.

Outro ponto importante é com relação aos riscos que nem sempre são previstos antes do início do projeto e quando eles acontecem, afetam diretamente o cumprimento dos prazos determinados, vamos então ver algumas de suas características.

1.2 Análise de Riscos

Uma das atividades essenciais para que um projeto caminhe bem é a análise dos riscos. Esta atividade está baseada na realização de quatro tarefas, que devem ser feitas de forma sequencial: a identificação, a projeção, a avaliação e a administração, parecem bem óbvias, mas realmente são, tenha sempre em mente, um projeto bem conduzido não é aquele que usa as técnicas mais complicadas, mas sim as que funcionam.

1.2.1 A Identificação dos Riscos

Nesta primeira tarefa, o objetivo é que seja feito o levantamento, da parte do gerente e dos profissionais envolvidos no projeto, de todos os riscos que podem atingir e impactar negativamente no mesmo. Nesta identificação, nós podemos identificar riscos de diferentes naturezas:

- **riscos de projeto**, são aqueles problemas relacionados ao próprio processo de desenvolvimento como orçamento, cronograma, pessoal e outros.

	Podemos citar como exemplo um colaborador que pediu demissão no meio do projeto, ou a verba prevista para fazer o investimento necessário, mas que teve que ser usada em caráter de urgência para alguma outra finalidade. Não é o correto e nem o adequado, mas sabemos que às vezes acontece e nem sempre se tem a opção de evitar esta situação.

- **riscos técnicos**, que consistem dos problemas de projeto efetivamente como implementação, manutenção, interfaces, plataformas de implementação.

	Como exemplo podemos citar uma interface com determinado hardware que demandou mais tempo que o previsto, ou a implementação de alguma rotina específica foi mais complexa que o planejado.

- **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...).

	Em muitos casos não se faz um estudo prévio do mercado, pode então acontecer de, por exemplo, existir um produto considerado satisfatório pelos clientes a um custo aceitável e isto será uma barreira a novos fornecedores.

Percebam o seguinte, o fato de analisarmos os riscos em categorias, não garante que teremos no final resultados satisfatórios, temos que ter sempre em mente que nem todos os riscos podem ser identificados facilmente. Uma boa técnica para conduzir a identificação dos riscos é a criação 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 na empresa? • 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?

Com base nas respostas a estas perguntas, será possível ao planejador verificar qual será o impacto dos riscos sobre o projeto, não sei se percebeu que a criação de um bom checklist é um bom ponto de partida para analisar os riscos.

1.2.2 Projeção dos Riscos

A projeção ou estimativa de riscos permite que você defina basicamente duas questões:

- Qual a probabilidade de que o risco aconteça durante o projeto?
- Caso o risco venha a acontecer, quais seriam as consequências dos problemas gerados por ele?

Para respondermos estas questões, podemos usar basicamente quatro atividades:

- estabelecimento de uma escala que demonstre a possibilidade de ocorrência de um dos riscos;
- identificação de quais seriam as consequências do risco;
- calcular ou fazer uma previsão do impacto do risco no projeto e no software (produtividade e qualidade);
- calcular ou prever qual a precisão da projeção de riscos, ou seja, o quanto a previsão dos riscos é confiável.

A escala pode ser definida de acordo com várias representações (booleana, qualitativa ou quantitativa). No limite cada pergunta de um checklist pode ser respondida com "sim" ou "não", mas nem sempre esta representação permite representar as incertezas de modo realístico, em alguns momentos você vai ver que pode aparecer um “talvez”, ou seja, colocarmos como resposta as perguntas sim ou não em algumas situações não vai ser a melhor opção.

Uma outra forma de se representar seria utilizar uma escala de probabilidades, onde os valores seriam: altamente improvável, improvável, moderado, provável, altamente provável. A partir destes "valores", seria possível

realizar cálculos para determinar melhor e até matematicamente as probabilidades de que certo risco viesse a ocorrer, depois veremos um exemplo disto.

O próximo passo é então o levantamento do impacto que os problemas associados ao risco terão sobre o projeto e sobre o produto, isto vai permitir dar prioridade a situação que iria gerar maiores problemas. Podemos destacar três fatores que vão influenciar no impacto de um determinado risco: a sua natureza, o seu escopo (abrangência) e o período da sua ocorrência.

A natureza do risco permite indicar os problemas prováveis se ele ocorrer, por exemplo, um risco técnico gerado por uma má definição da comunicação entre o software e o hardware do cliente levará certamente a problemas de teste de integração. O seu escopo vai indicar qual a parcela do projeto que será atingida, ou seja, quanto do projeto vai ser afetado pelos problemas gerados por aquele risco. E por último, o período de ocorrência de um risco permite que você pense sobre quando ele provavelmente pode acontecer e, caso aconteça, quanto tempo este problema provavelmente irá durar.

Os conjuntos formados por estes três fatores vão permitir várias análises e atitudes, veja alguns exemplos:

- Focar a atenção no que é mais grave primeiro;
- Não desconsiderar, mas dar menor importância a um fator de risco de impacto alto se a sua probabilidade de acontecer for baixa;
- Dar maior atenção a riscos com alto impacto e com possibilidade de ocorrência de moderada a alta;
- Prestar atenção a riscos com baixo impacto, mas com grande possibilidade de acontecer, já que fatalmente terão algum impacto no projeto.

1.2.3 Avaliação dos Riscos

Você percebeu como estamos seguindo uma sequência bem lógica e quase que natural? Identificamos o risco, fazemos uma projeção do que pode ocorrer, chega então o momento de avaliar, nesta etapa o objetivo é pensar a respeito das informações que colhemos antes sobre o fator de risco, o impacto do risco e a probabilidade de ocorrência. Nesta avaliação, iremos checar as informações que

tivemos na projeção de riscos e iremos buscar meios ou formas de controlá-los ou então, se for possível, contorná-los, principalmente aqueles com alta probabilidade de acontecer.

Para que a avaliação seja eficiente, devemos primeiro definir qual é o **nível de risco referente**. Para você ter uma ideia do que estou dizendo, exemplos de níveis referentes típicos em projetos de Engenharia de Software são: o custo, o prazo e o desempenho. Isto significa que vamos definir um nível limite, ou seja, uma margem aceitável para o aumento de custo, para a ultrapassagem de prazo e para a queda do desempenho ou qualquer combinação dos três.

Caso os problemas causados pelos riscos causem a ultrapassagem de um ou mais desses níveis, o projeto poderá ser suspenso, quero que você perceba o tamanho de sua responsabilidade na hora de definir este “teto”, já que ele pode determinar se um projeto vai ser abortado em algum momento. Normalmente, é possível estabelecer um limite, denominado de ponto referente (*breakpoint*) onde tanto a decisão de continuar o projeto ou de abandoná-lo pode ser tomada.

Leia com bastante atenção este trecho que ele é chave para que você entenda de forma clara como acontece o breakpoint. A figura 1 mostra uma situação, onde dois níveis de risco referente são considerados (o custo e o prazo). Se uma combinação de riscos gerar a uma situação que ultrapasse os limites de custo e/ou de prazo (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.

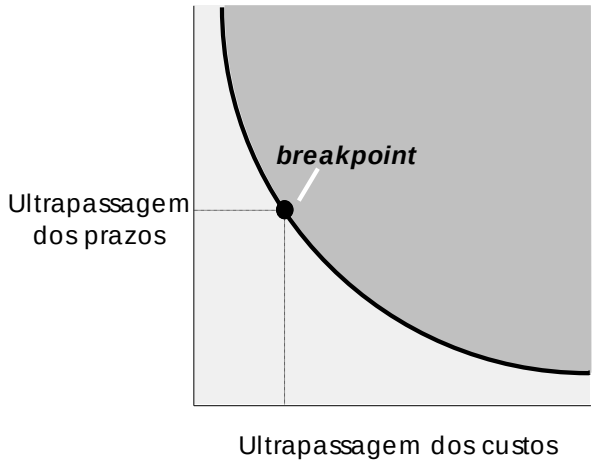


Figura 1 - Ilustração de uma situação de definição de dois níveis de riscos.

1.2.4 Administração e Monitoração dos Riscos

Chegou a hora então de administrar e monitorar os riscos que foram identificados, projetados e avaliados anteriormente, faço então uma pergunta.

O que pode acontecer se a etapa de identificação dos riscos não for bem feita??? Visualizou a sua importância? Se um risco não é identificado, como ele vai ser depois administrado e monitorado?

Neste caso você vai ter que contar com a sorte para conseguir contornar todos os problemas que você não projetou e não avaliou.

Considerando uma situação onde foi tudo feito corretamente, depois de avaliados os riscos de desenvolvimento, é importante tomarmos medidas para evitar a ocorrência do risco. Caso não seja possível contornar o risco, devemos definir quais são as ações “emergenciais” que serão tomadas caso o risco ocorra.

Este é o objetivo da tarefa de Administração e monitoração dos riscos. Para isto, as informações mais importantes são as obtidas na tarefa anterior que são a *descrição*, *probabilidade de ocorrência* e *impacto sobre o processo*, estas informações tem que ser feitas para cada fator de risco existente.

Vejamos um exemplo, considerando a alta rotatividade de pessoal numa equipe, um fator de risco, com base em dados de projetos passados, conseguimos determinar 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 total em 12%.

Sendo assim, podem-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...);
- tomar 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 as equipes do projeto de forma que as informações sobre cada atividade sejam conhecidas por mais de uma pessoa ou difundidas;
- definir padrões de documentação para que outra pessoa ao ler a documentação consiga dar continuidade de forma adequada;
- fazer revisões do trabalho entre os 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.

Este foi só um exemplo, bem prático na verdade, já que isto é um fator de risco que normalmente aparece em grandes projetos.

É importante que você perceba um detalhe, as ações para evitar um risco, como as que coloquei acima, podem afetar também os prazos e o custo global do projeto. Para difundir as informações entre os membros da equipe, fazer reuniões, ter um profissional que possa "tapar" a brecha deixada, tudo irá gerar um custo e levar determinado tempo, claro que isto deve se considerado também.

Vamos imaginar um projeto de grande dimensão, onde 30 fatores de risco foram identificados; se para cada fator de risco, sete ações forem definidas, a administração dos riscos pode tornar-se um projeto por si só. Por esta razão é que precisamos dar prioridade aos riscos para avaliarmos o que realmente é necessário ser tratado.

Todo o trabalho efetuado nesta tarefa é registrado num documento que podemos chamar **Plano de Administração e Monitoração de Riscos**, e que será utilizado depois pelo gerente de projetos (particularmente, para a definição do Plano de Projeto, que é gerado ao final da etapa de planejamento). Uma ilustração sintetizando os passos essenciais desta tarefa é apresentada à figura 2, irei disponibilizar este modelo para uso.

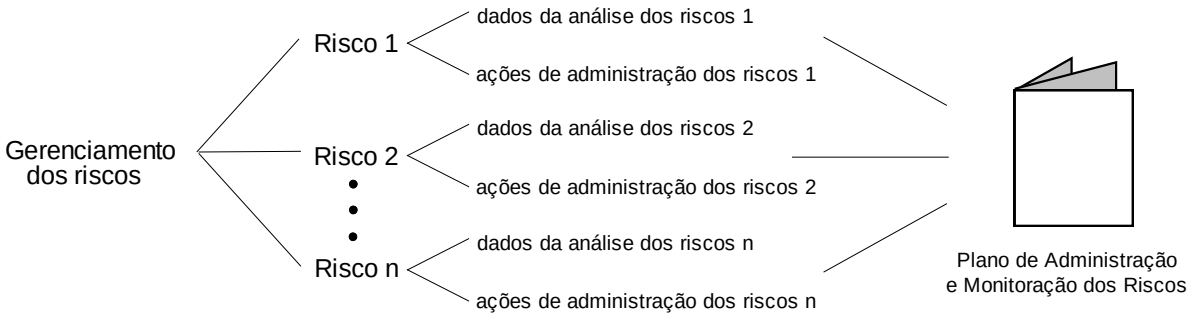


Figura 2 - Síntese da tarefa de Administração e Monitoração dos Riscos

Conclusão	As atividades que acontecem dentre da análise de risco são bem simples e eficientes, consiste basicamente em identificar, medir seu impacto, ver a probabilidade de que ele aconteça e contorná-los. Apesar de simples esta atividade pode ser responsável pelo sucesso de um projeto ou pelo menos garantir que ele permaneça dentro dos limites aceitáveis.

1.3 Cronograma

Obviamente que todo projeto deve seguir um cronograma, mas como fazer? O que deve ter nele? O que deve ser considerado? É isto que iremos definir neste tópico. Para começarmos a definir um cronograma, podemos fazer segundo duas abordagens diferentes:

- a primeira, é quando se tem um prazo pré-definido para entrega do software; neste caso, o planejamento deve ser feito usando este prazo para distribuir os esforços ao longo do tempo estabelecido;
- a segunda, é feita através de uma discussão de limites cronológicos aproximados para cada etapa do desenvolvimento, sendo que o prazo de entrega do software será estabelecido a partir de técnicas de planejamento da Engenharia de Software.

É evidente que a primeira abordagem é a mais encontrada nos projetos de software, você já sabe também que a segunda seria a ideal, mas nem sempre pode ser feito desta forma.

Mesmo na primeira abordagem, um cronograma bem definido pode trazer enormes benefícios a um projeto de software, ele pode ser às vezes mais importante que a própria definição de custos. Numa visão de desenvolvimento de software como produto, o custo adicional 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, dentre outros.

Quando a tarefa é fixar prazos para os projetos de software, podemos pensar em diversas questões:

- como relacionar o tempo cronológico com o esforço humano?
- quais tarefas podem ser executadas ao mesmo tempo?
- 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 podemos usar para determinar os prazos?
- como representar fisicamente o cronograma e como acompanhar o progresso a partir do início do projeto?

Como estas questões são muito importantes, por isto separei cada um destes assuntos em tópicos para discutirmos cada um deles.

1.3.1 As relações pessoas-trabalho

Em primeiro lugar, é preciso dizer que, à medida que um projeto ganha dimensão, um maior número de pessoas provavelmente estarão envolvidas. Além disso, devemos tomar o cuidado de não levar a sério o mito falado na disciplina Engenharia de Software I, *"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 pede esforço adicional e, consequentemente, 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 não forem tão importantes) do que o contrário.

1.3.2 A Definição de Tarefas

Uma das vantagens de se ter uma equipe com mais de uma pessoa são as possibilidades de se fazer várias tarefas ao mesmo tempo. O paralelismo de tarefas é um mecanismo interessante que se for usado de forma correta pode economizar tempo em qualquer trabalho de engenharia. Para isto, basta que um gerenciamento dos recursos que vão ser usados em cada tarefa (recursos humanos, recursos de software, etc...) seja feito de forma eficiente.

Podemos então, se o paralelismo for usado, representar as tarefas que serão feitas durante um projeto de desenvolvimento de software na forma de uma rede (rede de tarefas) que irá representar todas as tarefas do projeto e suas relações de realização (paralelismo, sequência, pontos de encontro, etc...), veja um exemplo na figura 3.

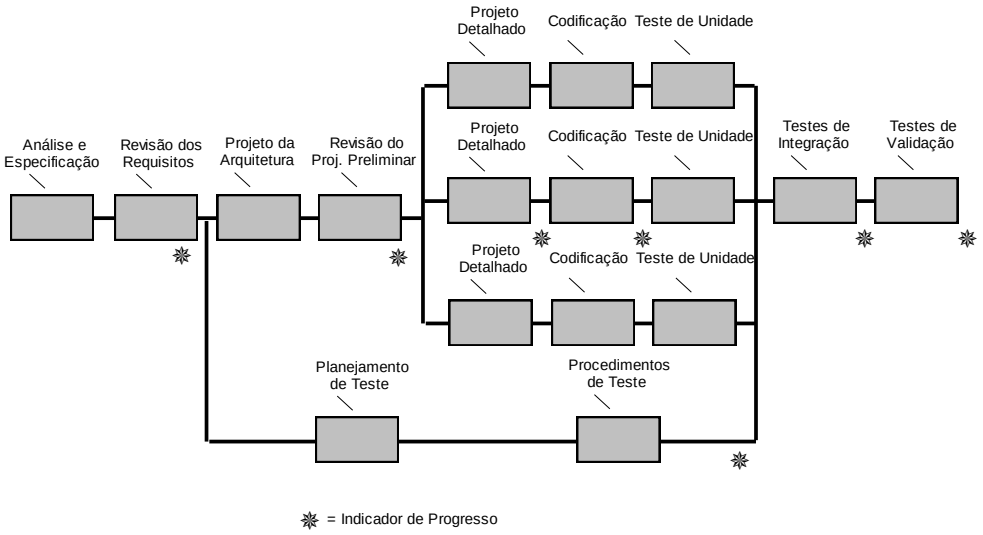


Figura 4.3 - Exemplo de uma Rede de Tarefas.

Nesta figura, podemos ver que existem indicadores de progresso (representados por um símbolo "□") colocados ao longo do processo e que permitem que o gerente faça 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.

1.3.3 A Distribuição do Esforço

As principais técnicas de estimativas para projetos de software usam uma definição do esforço em termos do número de homens-mês ou homens-ano necessários para realizar o projeto. Uma proposta de distribuição de esforço muito utilizada nos projetos é a ilustrada na figura 4, onde se segue a regra denominada *Regra 40-20-40*, que sugere esforço maior, extremos que são os processos análise/projeto e testes, já a codificação deve ser a tarefa que irá envolver a menor concentração de esforço.

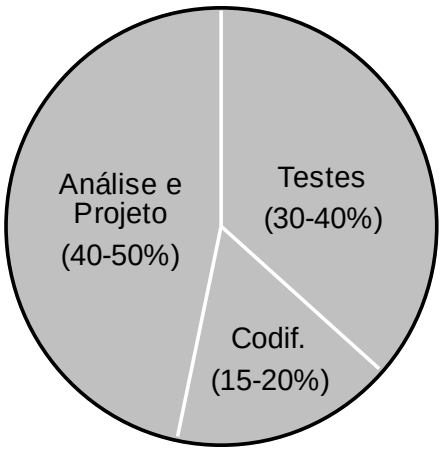


Figura 4 - Proposta de distribuição dos esforços de desenvolvimento.

Esta divisão pode ir contra o grau de importância em termos de esforço que muitos desenvolvedores dão a cada uma destas atividades.

É claro que estes valores não devem ser levados à risca para todos os projetos de software, as características de cada projeto irão influenciar na parcela de esforço que será dedicada a cada etapa.

O importante é que se dedique mais ou menos tempo para cada etapa de forma consciente e não ao acaso.

Na realidade, se a etapa de projeto foi realizada utilizando as boas regras da Engenharia de Software, a etapa de codificação será consequentemente, minimizada em termos de esforço, já que o código a ser produzido é na verdade somente a representação do que já está planejado.

Em processos de desenvolvimento em que as etapas de projeto/análise não são feitas seguindo as boas práticas de engenharia de software, fatalmente a etapa de codificação irá exigir maior esforço, já que atividades de análise e decisões de projeto acabam sendo efetuadas na etapa de desenvolvimento. Veja bem, se você dedicou menos tempo às fases que vem antes da codificação, durante ela você terá que arcar com isto, já que assim não compensa planejar mais e executar menos e com mais qualidade? Além disto, você terá mais tranquilidade já que seu software corresponderá às necessidades do usuário, pois o levantamento foi bem feito.

A etapa de testes, por sua vez, pode representar de 30 a 40% do esforço total do projeto, um detalhe que você deve ter sempre em mente, é que a qualidade do software não está na quantidade de testes feitos, mas sim em todas as outras etapas anteriores, não podemos passar a responsabilidade da qualidade para esta fase, já que nunca conseguiremos encontrar todos os erros existentes, isto é fato.

Na realidade, o que vai determinar o esforço a ser 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).

1.3.4 A Representação do Cronograma

A última atividade, também uma das tarefas mais difíceis esta na etapa de planejamento do software é a elaboração do cronograma. O planejador deve levar em conta diversos aspectos relacionados ao desenvolvimento, como: os recursos que vão estar disponíveis no momento da execução de cada uma das tarefas, 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, dentre outros.

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 5.

As unidades de tempo consideradas no projeto horas, dias, semanas ou meses 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 irá indicar o período que certa atividade será realizada, o tempo necessário será medido em unidades de tempo consideradas. Quando atividades puderem ser realizadas em paralelo, os traços vão ocupar unidades de tempo comuns.

O cronograma deve conter 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) para garantir que aquela etapa foi realmente concluída.

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

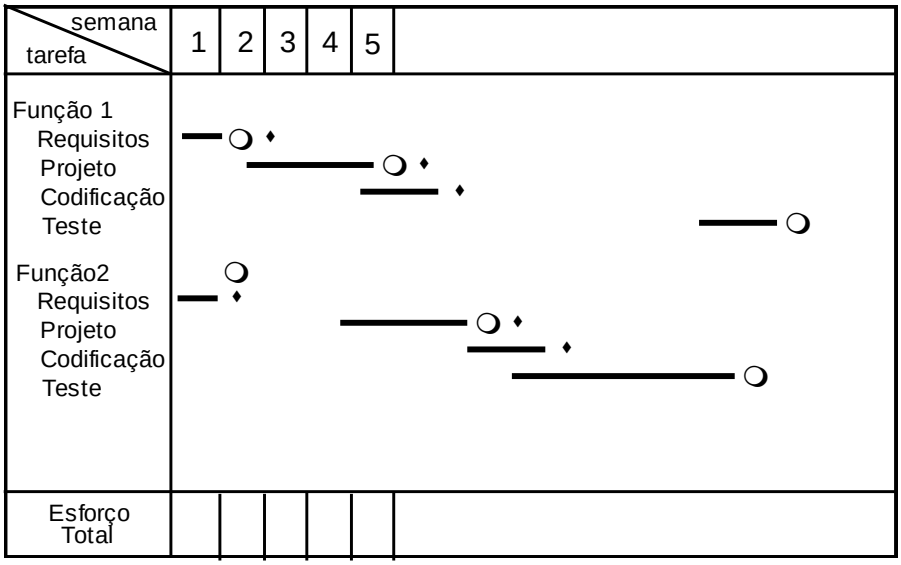


Figura 5 - Representação geral de um cronograma de desenvolvimento.

Conclusão	A definição de um cronograma é importante para que todos os envolvidos se situem dentro do projeto, saibam se estão adiantados e podem se dedicar mais a alguma tarefa ou se estão atrasados e devem fazer um maior esforço para tentar chegar ao que foi proposto. Não podemos enxergar o cronograma como uma atividade burocrática, ele é um documento que deve estar sempre atualizado e correspondendo a realidade para nortear todos os interessados.

1.4 Aquisição de Software

Apesar de a Engenharia de Software sugerir o desenvolvimento de software, em alguns casos pode ser mais interessante adquirir o software do que desenvolvê-lo. Esta possibilidade deve ser considerada, e muitas das vezes é a melhor alternativa. Determinados projetos serão mais bem atendidos com a compra do que com o desenvolvimento em relação a tempo, custo e/ou qualidade, esta é uma decisão que 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 os requisitos estabelecidas;
- adquirir um pacote comercial e customizá-lo de forma que o novo software atenda às especificações de projeto;
- encomendar o software a terceiros para que este crie um produto que atenda às especificações.

Os procedimentos que devem ser feitos para a aquisição de um software vão depender 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, deve ser feita uma análise mais cuidadosa, sendo que os procedimentos podem ser os seguintes:

- desenvolver uma especificação funcional e de desempenho para o software, estabelecendo, quando possível, valores que podem ser medidos para determinar o quanto ele atende o que se precisa;
- realizar a estimativa do custo interno de desenvolvimento para fazer um comparativo entre o que se gastaria para desenvolver e as outras possibilidades;
- 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 pode envolver vários critérios, dentre eles:

- O produto será entregue antes da data de finalização do produto se desenvolvido internamente?
- o custo de aquisição mais o custo de "customização" será menor que o custo de desenvolvimento interno?
- o custo de suporte externo (contrato de manutenção) é menor que o custo do suporte interno?

Conclusão	Em diversas situações a opção de desenvolver o software não é a mais adequada, seja em termos de custo, prazo ou qualidade. Nestes casos a aquisição se torna uma grande opção, o importante é que ela seja feita usando **critérios claros**, para que a compra traga os resultados esperados.

1.5 Reengenharia de software

Um outro problema importante que deve 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 deficientes e com alta taxa de falhas. Estes fatores em conjunto levaram a uma situação onde os custos de manutenção destes sistemas não justificam mais os benefícios que estas alterações podem trazer.

Por outro lado, um processo de reengenharia do software pode aparecer como uma alternativa para redução dos custos de manutenção do software. Para isto, vejamos quais são os procedimentos iniciais para um processo deste tipo:

- 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 depois de 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.

Conclusão	Como você pôde perceber a reengenharia não traz nenhuma receita a ser seguida, isto irá depender do caso em estudo. É uma opção a mais a ser considerada, principalmente quando o impacto do desenvolvimento ou aquisição de um novo software será grande.

1.6 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:

- *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;
- *n* indivíduos são alocados a *m* diferentes tarefas, formando assim equipes informais de desenvolvimento, com um responsável em cada equipe, sendo que a coordenação entre as equipes é da responsabilidade do gerente do projeto;
- *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 terceira opção é a 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" 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 vão variar em função da grandeza do projeto.

O núcleo de uma equipe pode 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 **pessoal 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** que pode envolver várias áreas como telecomunicações, bancos de dados, interface homem-máquina e qualquer outra envolvida;
- **pessoal de apoio** como secretárias, editores técnicos e desenhistas;
- um **bibliotecário**, que será responsável pela organização e catalogação de todos os componentes do produto de software (documentos, listagens, mídia magnética, coleta de dados relativos ao projeto, módulos reutilizáveis, etc...).

Em alguns projetos pequenos pode parecer que esta divisão está fora da realidade, é importante você perceber que esta estrutura é flexível e para projetos menores, uma pessoa provavelmente será a responsável por mais de uma das funções.

Porém a definição do papel de cada um dos envolvidos permite que nenhuma das atividades deixe de ser executada, quando todos são responsáveis por alguma das tarefas, pode acontecer uma falta de coordenação e alguma delas ficar sem ser executada.

1.7 O Plano de software

Ao final desta etapa, um documento de **Plano de Software** deverá ser gerado, ele deverá ser revisto para servir de referência às etapas posteriores. Irá

apresentar as informações iniciais de custo e cronograma que vão nortear o desenvolvimento do software. Ele consiste de um documento relativamente breve, que será encaminhado às diversas pessoas envolvidas no desenvolvimento do software.

Dentre as informações a serem contidas neste documento, podemos citar como mais importantes:

- 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 7 apresenta uma possível estrutura para este documento. A apresentação dos custos e cronograma pode ser diferente dependendo de quem vai ser 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

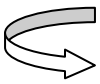
Figura 7 - Proposta de estrutura para o documento de Plano do Software.

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.

2 PROJETO DE SOFTWARE

2.1 Introdução

A etapa de Projeto é o passo inicial de uma nova fase do ciclo de vida do software, a fase de Desenvolvimento. O Projeto é formado pela aplicação de um conjunto de técnicas e princípios, com o objetivo de definir um sistema num nível de detalhe suficiente à sua realização física. A tarefa do projetista nada mais é do que produzir um modelo de representação do software que será implementado. Nesta etapa, os requisitos definidos na etapa anterior devem servir de referência para fazermos à representação do software.



Caso sinta necessidade reveja os conceitos dos ciclos de vida vistos em engenharia de software I para visualizar onde está o projeto no contexto.

Veremos aqui os principais aspectos relativos ao projeto de software, como etapa fundamental para a obtenção de um software de qualidade. Serão apresentadas algumas das técnicas clássicas de projeto e será feita a aplicação de técnicas de projeto a aspectos mais inovadores.

2.2 O processo de Projeto

2.2.1 Projetos Preliminar e Detalhado

A etapa de projeto é caracterizada pelo conjunto de atividades que vai permitir traduzir os requisitos definidos na etapa anterior em uma representação do software a ser construído. Na sua forma mais clássica, o primeiro resultado obtido no projeto é uma visão da estrutura do software em termos de componentes, sendo que, a partir de procedimentos de refinamento, chega-se a um nível de especificação bastante próxima da codificação do programa.

Do ponto de vista do gerenciamento do processo de desenvolvimento, a etapa de projeto é conduzida basicamente em dois principais estágios:

- o ***projeto preliminar***, que nos permite estabelecer, a partir dos requisitos, a arquitetura do software e da informação relacionada;

- o ***projeto detalhado***, que nos permite aperfeiçoar a estrutura do software e definir representações algorítmicas dos seus componentes.

No contexto dos projetos preliminar e detalhado, um conjunto de atividades técnicas de projeto são desenvolvidas. Num ponto de vista mais genérico, pode-se destacar os projetos *arquiteturais*, *procedimentais* e *de dados*. Em projetos mais recentes, e, particularmente, naqueles envolvendo interatividade com o usuário, o *projeto de interfaces* tem assumido um papel de grande importância, sendo considerada uma atividade do mesmo nível das outras atividades.

2.2.2 Características desejáveis num projeto de software

A criação de um software de boa qualidade está relacionada ao sucesso de cada uma das etapas do seu desenvolvimento, ou seja, se cada uma das etapas for concluída de forma adequada, no final teremos uma solução de qualidade. No que diz respeito à etapa de projeto, podemos destacar algumas das características desejáveis:

- deve apresentar um organização hierárquica, utilizando racionalmente todos os elementos de software;
- deve basear-se em princípios de modularidade, ou seja, propor um particionamento do software em elementos que programem funções ou subfunções do software, isto traz uma série de benefícios que serão comentados mais a frente;

- deve conduzir a uma definição em módulos com alto grau de independência, isto irá facilitar as tarefas de manutenção ao longo da vida do software;
- deve ser fiel à especificação dos requisitos definida na etapa anterior, se não for isto poderá gerar futuramente retrabalho.

Estas são apenas algumas das principais características que são desejáveis em um projeto, existem diversas outras que com o tempo iremos discutindo.

2.3 Aspectos fundamentais do projeto

Durante esta etapa, é importante pensarmos e ter em mente alguns conceitos para que possamos criar um projeto contendo as características que já foram comentadas acima. As seções a seguir discutirão alguns destes conceitos de forma um pouco mais detalhada e organizada para que você possa identificar em um projeto o que deve ser considerado. Irei conceituar cada um deles e depois iremos aplicar isto em nosso projeto.

2.3.1 Abstração

O princípio de abstração está muito relacionado com as características de modularidade que um software pode ter. Quando se desenvolve um software que vai apresentar estas características, é comum que suas representações sejam feitas considerando vários níveis de abstração.

Falando da linguagem de representação, nos níveis mais altos de abstração, a linguagem utilizada é bastante orientada à aplicação e ao ambiente onde o software vai ser executado. À medida que se vai "descendo" nos níveis de abstração, a linguagem de representação vai se aproximando de questões de implementação, até que, no nível mais baixo, a solução é representada de modo que possa ser feita diretamente em uma linguagem de implementação.

Na realidade, o próprio processo de Engenharia de Software é formado por um aprimoramento de níveis de abstração. Na Engenharia de Sistemas, por exemplo, partes das tarefas do sistema que será desenvolvido são atribuídas ao software, como elemento de um sistema computacional. Na Análise de Requisitos, a solução em termos de software é apresentada de forma conceitual. Durante o Projeto, partindo do Projeto Preliminar para o Projeto Detalhado, múltiplos níveis de abstração vão sendo definidos, aproximando-se cada vez mais da implementação, que é o nível mais baixo de abstração.

A cada etapa completada mais perto da implementação do software vamos chegando, falando de ferramentas, podemos começar pensando nas entrevistas, passando pelo DFD, dicionário de dados e chegando implementação e finalmente às técnicas de teste.

2.3.2 Refinamento

O refinamento surgiu na forma de uma técnica de projeto (a técnica de Refinamentos Sucessivos, proposta por Wirth em 1971). Esta técnica sugere como ponto de partida a definição da arquitetura do software a ser desenvolvido, sendo que esta vai sendo refinada sucessivamente até atingir níveis onde os procedimentos são detalhados. Este processo dá origem a uma hierarquia de representações, onde uma descrição mais abrangente de cada função vai sendo decomposta passo-a-passo até chegarmos a representações bastante próximas de uma linguagem de implementação, que podemos chamar também de linguagem de programação. Este processo é bastante similar ao processo utilizado em diversas técnicas de Análise de Requisitos como os níveis do DFD, sendo que a diferença fundamental está nos níveis de detalhamento atingidos nesta etapa.

	Qual a ligação entre as ferramentas vistas em Engenharia de Software I e esta etapa que estamos vendo agora?

2.3.3 Modularidade

O conceito de modularidade tem sido utilizado já há bastante tempo, como forma de obtenção de um software que apresente algumas características interessantes como a facilidade de manutenção.

Este conceito apareceu como uma solução aos antigos softwares "monolíticos" feito por um único grande bloco de software, que representavam grandes dificuldades de entendimento e, consequentemente para manutenção que fosse necessária.

A utilização do conceito de modularidade oferece resultados em curto prazo, ao dividir um grande problema em problemas menores, as soluções são encontradas com esforço relativamente menor. Isto significa que, quanto maior o número de módulos definidos num software, menor será o esforço necessário para desenvolvê-lo, já que o esforço de desenvolvimento de cada módulo será menor. Por outro lado, quanto maior o número de módulos, maior será o esforço no desenvolvimento das interfaces, ou seja, devemos equilibrar bem o uso desta regra para que ela mesmo não se torne um problema.

É importante que você diferencie o conceito de modularidade de projeto com o de modularidade de implementação (visto na disciplina POO). Nada impede que um software seja projetado sob a ótica da modularidade e que sua implementação seja monolítica.

Em alguns casos, como forma de evitar desperdício de tempo, de processamento e de memória em chamadas de procedimentos, coloca-se como regra o desenvolvimento de um programa sem a utilização de funções e procedimentos. Ainda assim, uma vez se o projeto seguiu uma filosofia de modularidade, o software deverá apresentar alguns benefícios gerados pela adoção deste princípio.

2.3.4 A Arquitetura de Software

O conceito de arquitetura de software está ligado aos dois principais aspectos do funcionamento de um software: a estrutura hierárquica de seus componentes (ou módulos) e as estruturas de dados.

A arquitetura de software resulta do desenvolvimento de atividades de particionamento de um problema, encaminhadas desde a etapa de Análise de Requisitos. Naquela etapa, é dado o pontapé inicial para a definição das estruturas de dados e dos componentes de software. A solução é encaminhada ao longo do projeto, através da definição de um ou mais elementos de software que solucionarão uma parte do problema global.

É importante lembrar que não existe técnica de projeto que garanta a unicidade de solução a um dado problema. Diferentes soluções em termos de arquitetura podem ser derivadas a partir de um mesmo conjunto de requisitos de software. A grande dificuldade concentra-se em definir qual a melhor opção em termos de solução.

Para definir ou representar uma arquitetura de software, surgiram as Linguagens de descrição de arquitetura (LDAs). Vários diferentes LDAs foram desenvolvidas por diferentes organizações, incluindo Wright (desenvolvido por Carnegie Mellon), Acme (desenvolvido por Carnegie Mellon), xADL (desenvolvido por UCI), Darwin (desenvolvido por Imperial College London), DAOP-ADL (desenvolvido pela University of Málaga). Elementos comuns de uma LDA são componente, conexão e configuração.

2.3.4.1 Visões

A arquitetura de software é normalmente organizada em visões que são pontos de vista diferentes a respeito da mesma informação, algumas possíveis visões são:

Visão funcional/lógica
Visão de código.
Visão de desenvolvimento/estrutural
Visão de concorrência/processo/thread
Visão física/evolutiva
Visão de ação do usuário/feedback

Várias linguagens para descrição da arquitetura de software foram inventadas, mas nenhum consenso foi ainda alcançado em relação a qual conjunto de símbolos ou sistema representação deve ser adotado. Alguns acreditam que a UML, que será vista mais a frente por nós, ira estabelecer um padrão para representação de arquitetura de software. Outros acreditam que os desenvolvimentos efetivos de software devem contar com a compreensão única das restrições de cada problema, e notações tão universais são condenadas a um final infeliz porque cada uma prove um notação diferenciada que necessariamente torna a notação inútil ou perigosa para alguns conjuntos de tarefas.

Há muitas formas comuns de projetar módulos de software de computador e suas comunicações, entre elas:

- Cliente-Servidor
- Computação distribuída
- P2P
- Blackboard
- Criação implícita
- Pipes e filtros
- Plugin
- Sistema Monolítico
- Modelo em três camadas
- Análise de sistema estruturada
- Arquitetura orientada a serviço
- Arquitetura orientada a busca

Como viu somente este tópico já seria o suficiente discutirmos um semestre todo ou mais, a ideia aqui é somente que você tome conhecimento deste universo, para que depois possa explorá-lo, caso necessário, de forma consciente.

2.3.5 Hierarquia de Controle

A hierarquia de controle nada mais é do que a representação, de forma hierárquica da estrutura do software, no que diz respeito aos seus componentes. O objetivo não é apresentar detalhes procedimentais ou de sequenciamento entre processos, mas de estabelecer as relações entre os diferentes componentes do software

O modo mais usual de apresentar a hierarquia de controle utiliza uma linguagem gráfica, normalmente em forma de árvore, como mostra a figura 7. Com relação à estrutura de controle é importante apresentar algumas definições de "medição".

Utilizando a figura 7 como referência, é possível extrair alguns conceitos:

- a ***profundidade***, que está ligada ao número de níveis de abstração definidos para a representação do software;
- a ***largura***, que permite definir a abrangência do controle global do software;
- o ***fan-out***, que permite definir a quantidade de módulos controlados por um dado módulo;
- o ***fan-in***, que indica quantos módulos controlam um dado módulo.

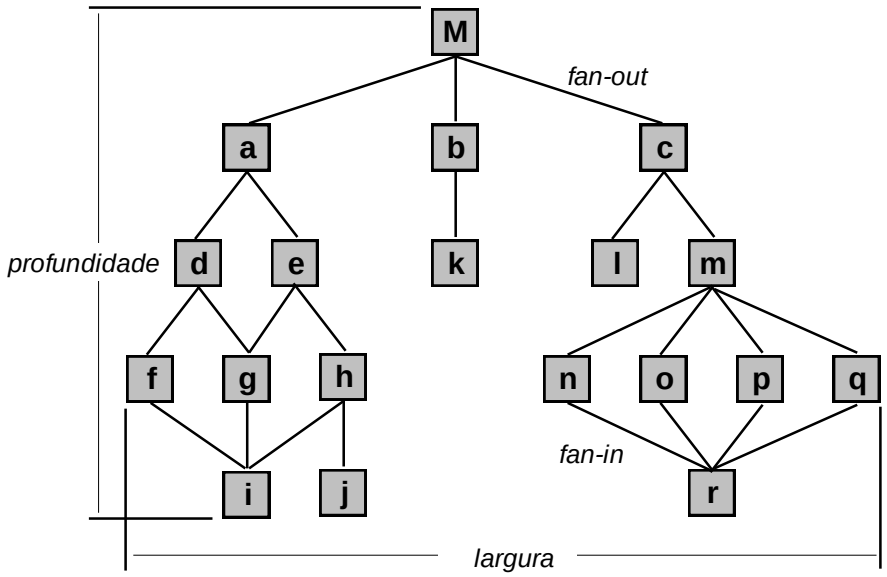


Figura 7 - Representação da hierarquia de controle.

Ainda, é possível estabelecermos as relações de controle entre os módulos. Um módulo que exerce controle sobre outros módulos é chamado de ***superordenado***. Um módulo que é controlado por outro é chamado de módulo ***subordinado*** a ele.

2.3.6 A Estrutura dos Dados

A estrutura dos dados representa os relacionamentos lógicos entre os diferentes elementos de dados. À medida que o projeto se aproxima da implementação, esta representação começa a ser de fundamental importância para que o programador possa organizar as informações da forma determinada para o engenheiro de software para melhor performance, já que a estrutura da informação vai ter um impacto significativo no projeto procedimental final.

A estrutura dos dados define a forma que os dados estão organizados, os métodos de acesso, o grau de associatividade e as alternativas de processamento das informações. Apesar de que a forma de organizar os elementos de dados e a complexidade de cada estrutura dependa, do tipo de aplicação a desenvolver e da criatividade do projetista, existe um número limitado de componentes clássicos que funcionam como blocos de construção (building-blocks) de estruturas mais complexas.

A figura 8 ilustra alguns destes blocos. Os blocos ilustrados podem ser combinados das mais diversas maneiras para obter estruturas de dados com grau de complexidade relativamente complexo.

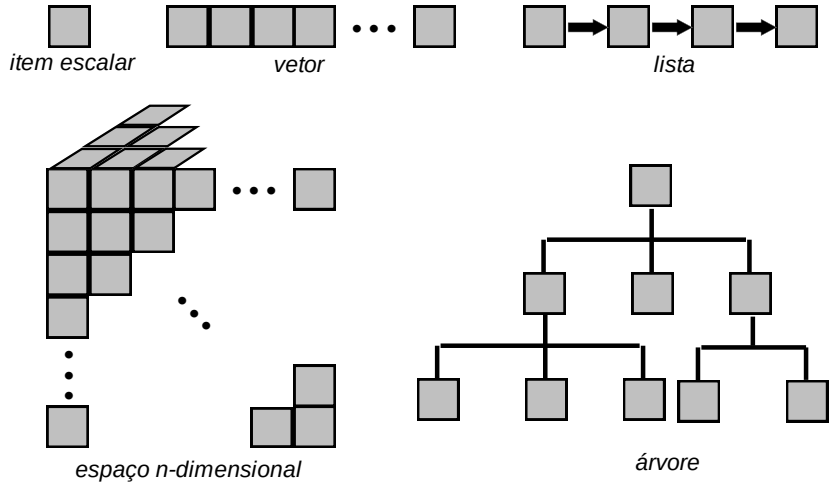


Figura 8 - Alguns construtores clássicos de estruturas de dados.

2.3.7 Ocultação de Informação

O princípio da ocultação de informações propõe um caminho para decompor um problema para obter, de modo eficiente, os diferentes módulos do software a ser construído.

Segundo este princípio, os módulos devem ser decompostos de modo que as informações (os procedimentos e dados) contidas em cada módulo sejam inacessíveis aos módulos que não tenham necessidade destas informações. Ao realizar a decomposição segundo este princípio, o projetista proporciona um grau relativamente alto de independência entre os módulos, o que é altamente desejável num tal projeto.

Os benefícios da adoção deste princípio vão aparecer no momento em que modificações deverão ser encaminhadas a implementação, por exemplo, por consequência de uma atividade de teste do software. Graças à ocultação de informação, erros introduzidos como resultado de alguma modificação num dado módulo, não serão propagados a outros módulos do software.

Com este último item encerramos nossa discussão sobre as características de um projeto, espero que tenha conseguido ter uma visão sobre os diversos aspectos que estão relacionados a ele e também que tenha servido para lhe mostrar como é vasto o campo da Engenharia de Software, não se desespere com a quantidade de informações que você sente necessidade de saber ao ler esta seção, vamos caminhando devagar que chegamos lá!!!

Qual a semelhança entre ocultamento de informação em Engenharia de Software e ocultamento de informação em POO?

2.4 Classes de Projeto

Olhando para os diversos aspectos apresentados na seção anterior, podemos verificar que, de fato, a etapa de projeto de software é composta por um conjunto de projetos, que são feitos paralelamente. Nos itens abaixo vamos discutir estes diferentes projetos.

2.4.1 O Projeto Modular

O primeiro projeto que será comentado trata do princípio da modularidade, ele é sem dúvida, um dos mais difundidos e adotados em qualquer abordagem de desenvolvimento de software conhecida, você mesmo já não ouviu falar várias vezes sobre módulos do sistema?

Durante o projeto, os princípios de abstração e de ocultação de informação são aplicados para obter os módulos que constituem a arquitetura de um programa. A aplicação destes dois princípios vai se refletir em características de operação do módulo, como:

- o ***tempo de incorporação***, que indica o momento da introdução do módulo no código-fonte do software como, por exemplo, um macro de compilação ou uma subrotina;
- o ***mecanismo de ativação***, que indica a forma como o módulo será invocado durante a execução do programa, por exemplo, por meio de uma referência, uma chamada ou por interrupção;
- o ***padrão de controle***, que descreve como o módulo é executado internamente.

Com relação aos tipos de módulos que podem ser definidos na arquitetura de um software, pode-se encontrar basicamente três categorias:

- os ***módulos sequenciais***, que são ativados e sua execução ocorre sem qualquer interrupção;
- os ***módulos incrementais***, que podem ser interrompidos antes da conclusão do aplicativo e terem sua execução retomada a partir do ponto de interrupção;
- os ***módulos paralelos***, que executam simultaneamente a outros módulos em ambientes multiprocessadores concorrentes.

Falando do projeto dos módulos, podemos comentar algumas definições importantes:

- a ***independência funcional***, que surge como consequência da aplicação dos princípios de abstração e ocultação de informação; a independência funcional pode ser obtida a partir da definição de módulos de "propósito único" e evitando muitas interações com outros módulos;
- a ***coesão***, que está fortemente ligada ao princípio de ocultação e que sugere que um módulo pode realizar a sua função com um mínimo de interação com os demais módulos do sistema; é desejável que os módulos num software apresentem um alto grau de coesão;
- o ***acoplamento***, que permite exprimir o grau de conexão entre os módulos; os módulos de um software devem apresentar um baixo coeficiente de acoplamento.

2.4.2 O Projeto dos Dados

O segundo projeto fala sobre o projeto dos dados, à medida que a etapa de projeto evolui, as estruturas de dados vão assumindo um papel mais importante nesta atividade, já que irão definir a complexidade dos componentes ou procedimentos do software. O projeto dos dados nada mais é do que o conjunto de representações lógicas dos objetos de dados identificados na etapa de Análise e Especificação dos Requisitos.

Como forma de obter resultados satisfatórios no que diz respeito ao projeto dos dados no contexto de um software, podemos adotar alguns princípios:

- a realização de uma análise sistemática no que diz respeito aos dados, da mesma forma que é feito com os aspectos funcionais e comportamentais do software;
- identificação exaustiva das estruturas de dados e das operações que vão ser feitas nelas;
- estabelecimento de um dicionário de dados podendo ser o mesmo definido na etapa de Análise de Requisitos, porém refinado;
- representar as estruturas de dados somente nos módulos que as utilizarão;

- estabelecimento de uma biblioteca de estruturas de dados úteis e das operações a serem aplicadas a elas, facilitando a reusabilidade;
- adoção de uma linguagem de programação e projeto que suporte tipos abstratos de dados.

2.4.3 O Projeto Arquitetural

O terceiro projeto é o arquitetural, que visa a criação de uma estrutura modular de programa, dotada de uma representação dos relacionamentos de controle entre os módulos.

Uma questão importante com relação a este projeto é que ele deve ser encaminhado prioritariamente a outros projetos. É importante, antes que outras decisões possam ser tomadas, que se tenha uma visão global da arquitetura do software. O que estiver definido no projeto da arquitetura do software vai ter, sem dúvida, grande impacto nas definições dos demais projetos.

2.4.4 O Projeto Procedimental

O projeto procedimental é encaminhado a partir da definição da estrutura do software. Devido à riqueza de detalhes que pode caracterizar o projeto procedimental, a adoção de notações adequadas ao projeto é uma necessidade, como forma de evitar a indesejável ambiguidade que poderia ser resultante da utilização, por exemplo, da linguagem natural.

2.4.4.1 O pseudocódigo

Esta notação pode ser aplicada tanto para o projeto arquitetural quanto para o projeto detalhado e a qualquer nível de abstração do projeto. O projetista representa os aspectos estruturais e de comportamento utilizando uma linguagem de síntese, com expressões de sua língua (Inglês, Português, etc...), e estruturada através de construções tais como: IF-THEN-ELSE, WHILE-DO, REPEAT-UNTIL, END, etc... Esta política permite uma representação e uma análise dos fluxos de controle que

determinam o comportamento dos componentes do software de forma bastante simples.

O uso do pseudocódigo pode suportar o desenvolvimento do projeto segundo uma política "top-down" ou "bottom-up". No caso do desenvolvimento "top-down", uma frase definida num dado nível de projeto é substituída por sequências de frases que representam o refinamento da frase original.

O pseudocódigo, apesar de não apresentar uma visão gráfica da estrutura do software ou de seu comportamento, é uma técnica bastante interessante pela sua facilidade de uso e pela sua similaridade com algumas das linguagens de implementação conhecidas, como, por exemplo, Pascal, C, etc... A seguir, é apresentada uma listagem exemplo de uso do pseudocódigo para o projeto detalhado de um componente de software.

```
INICIALIZA tabelas e contadores;
ABRE arquivos;
LÊ o primeiro registro de texto;
ENQUANTO houver registros de texto no arquivo FAZER
    ENQUANTO houver palavras no registro de texto FAZER
      LÊ a próxima palavra
      PROCURA na tabela de palavras a palavra lida
        SE a palavra lida é encontrada
          ENTÃO
            INCREMENTA o contador de ocorrências da palavra lida
          SENÃO
            INSERE a palavra lida na tabela de palavras
        INCREMENTA o contador de palavras processadas
    FIM_ENQUANTO
FIM_ENQUANTO
IMPRIME a tabela de palavras e o contador de palavras processadas
FECHA arquivos
FIM do programa
```

Este tipo de documentação deve ser feita para as rotinas mais importantes ou complexas, não é útil ou necessário fazermos isto para todas as rotinas do software.

2.4.4.2 O uso de notações gráficas

Outra contribuição importante às atividades de projeto pode ser adoção de notações gráficas para representar os procedimentos a serem implementados. O fluxograma é uma conhecida técnica para a representação do fluxo de controle de programas, particularmente, os programas sequênciais. Os fluxogramas estruturados são aqueles baseados na existência de um conjunto limitado de fluxogramas básicos que, combinados, compõem uma representação de comportamento de um elemento de software.

O resultado obtido é uma representação gráfica de comportamento que pode, eventualmente, ser representada por um pseudocódigo. Exemplos de fluxogramas básicos para a composição de fluxogramas mais complexos são apresentados na figura 9.

Uma técnica gráfica também definida para a representação de comportamento de componentes de um software é o diagrama de Nassi-Schneiderman. Esta técnica é baseada na representação através de caixas, das estruturas de controle clássicas.

De forma similar aos fluxogramas estruturados, os diagramas de Nassi-Schneiderman são baseados na existência de blocos básicos representando estruturas elementares de controle que serão combinados de modo a compor a representação total do software (ou do componente de software) projetado.

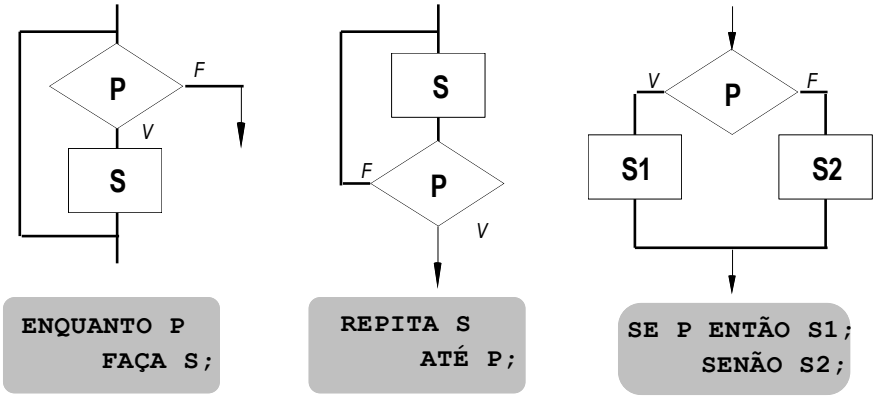


Figura 9 - Exemplos de blocos básicos para fluxogramas estruturados.

A figura 10 apresenta alguns exemplos de blocos básicos definidos nesta técnica.

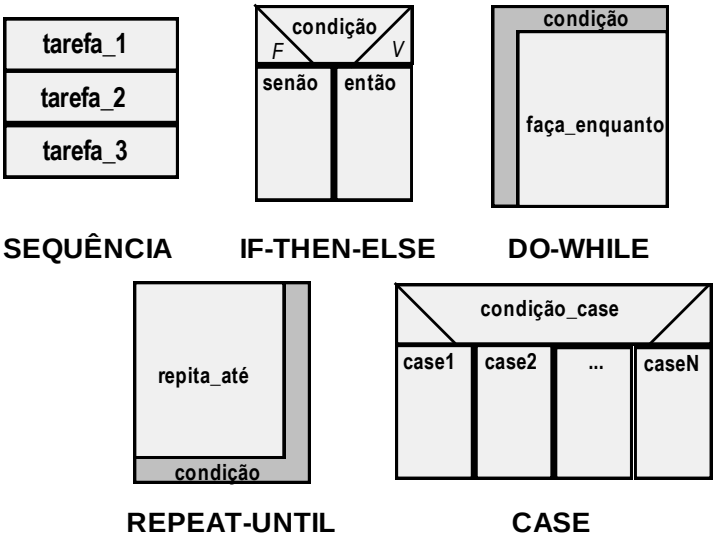


Figura 10 - Exemplos de blocos básicos dos diagramas de Nassi-Schneiderman.

2.5 Documentação

A figura 11 apresenta uma proposta de estrutura para o documento de projeto de software. O objetivo deste documento é apresentar uma descrição completa do software, sendo que as seções vão sendo geradas à medida que o projetista avança no seu trabalho de refinamento da representação do software.

Na seção I, é apresentado o escopo global do software, sendo que grande parte do que vai conter esta seção é derivada diretamente do documento de

Especificação de Requisitos obtido na etapa precedente, assim como de outros documentos gerados na fase de definição do software. A seção II apresenta as referências específicas à documentação de apoio utilizada.

A *Descrição de Projeto*, objeto da seção III, apresenta uma visão de projeto preliminar. Nesta parte do documento, é apresentada a estrutura do software, obtida a partir de diagramas de fluxos de dados ou outras técnicas de representação utilizadas durante a etapa de Análise de Requisitos. Juntamente com a estrutura do software, são apresentadas as diferentes interfaces de cada componente de software.

Nas seções IV e V, são apresentados os resultados das atividades de refinamento, que conduzem aos níveis mais detalhados de projeto. Inicialmente, é apresentada uma narrativa (em linguagem natural) da operação de cada componente de software (módulos, procedimentos, funções, etc...). Esta narrativa deve concentrar-se na especificação da função a ser provida pelo componente, evitando detalhes algorítmicos.

A partir desta narrativa, com o auxílio de uma técnica de projeto procedimental, obtém-se uma descrição estruturada de cada componente. A seção V apresenta uma descrição da organização dos dados (arquivos mantidos em meios de armazenagem, dados globais, referência cruzada, etc...).

I - Escopo
II - Documentos de Referência
III - Descrição do Projeto
- **3.1 - Descrição dos Dados**
- **3.2 - Estrutura de Software**
- **3.3 - Interfaces dos Componentes**

IV - Descrição dos Módulos
- **4.1 - Narrativa de Processamento**
- **4.2 - Descrição da Interface**
- **4.3 - Descrição numa Técnica de Projeto**
- **4.4 - Outros**

V - Estrutura de Arquivos e Dados Globais
VI - Referência Cruzada dos Requisitos
VII - Procedimentos de Teste
VIII - Requisitos Especiais
IX - Observações
X - Apêndices

Figura 11 - Estrutura do documento de Projeto de Software.

Na seção VI, é elaborada feita uma referência cruzada dos requisitos, com o objetivo de determinar que aspectos do projeto desenvolvido estão satisfazendo os requisitos especificados.

De uma forma mais concreta, deve ser feita a indicação de que conjunto de módulos é determinante para a implementação dos requisitos especificados.

Na seção VII é apresentada a especificação dos procedimentos de teste, o que é possível efetuar graças à definição já estabelecida da estrutura do software e das interfaces.

Na seção VIII, são apresentadas as restrições e requisitos especiais para o desenvolvimento do software (necessidade de overlay, gerenciamento de memória virtual, processamento de alta velocidade, interface gráfica específica, etc...).

As seções IX e X apresentam dados complementares, como a descrição de algoritmos, procedimentos alternativos, dados tabulares. Finalmente, pode ser encaminhado o desenvolvimento de um Manual de Instalação/Operações Preliminares, a ser incluído como apêndice ao documento.

	Quais itens você considera que esteja faltando neste documento modelo que foi apresentado? O que você acha obsoleto ou desnecessário neste documento?

3 ANÁLISE E PROJETO ORIENTADOS A OBJETOS

3.1 Introdução

Existem muitas definições para o que se chama, em desenvolvimento de software, de Orientação a Objetos. Estudando a bibliografia da área, você poderá observar que cada autor apresenta a sua visão do que entende por esta abordagem. Aí vão alguns conceitos:

- a orientação a objeto pode ser vista como a abordagem de modelagem e desenvolvimento que facilita a construção de sistemas complexos a partir de componentes individuais;
- o desenvolvimento orientado a objetos é a técnica de construção de software na forma de uma coleção estruturada de implementações de tipos abstratos de dados;
- desenvolvimento de sistemas orientado a objetos é um estilo de desenvolvimento de aplicações onde a encapsulação potencial e real de processos e dados é reconhecida num estagio inicial de desenvolvimento e num alto nível de abstração, com vistas a construir de forma econômica o que imita o mundo real mais fielmente;
- a orientação a objetos é uma forma de organizar o software como uma coleção de objetos discretos que incorporam estrutura de dados e comportamento.

Visando a que cada leitor passe a ter a sua própria definição do que significa Orientação a Objetos, vamos apresentar aqui os principais conceitos relacionados a esta tecnologia.

REFERÊNCIAS

Básica

Rumbaugh, J., Blaha, M. e outros. Modelagem e projetos baseados em objetos. Editora Campus, 1994.

Coad, P. e Yourdon, E.. Análise baseada em objetos.. Editora Campus, Rio de Janeiro, 1992.

Coad, P. e Yourdon, E.. Projeto baseado em objetos.. Editora Campus, Rio de Janeiro, 1993.

Complementar

Pressman, R.S.. Engenharia de Software, Makron Books do Brasil Editora, 1996.