Apostila

Análise e Gerência de Requisitos

ANÁLISE E GERÊNCIA DE REQUISITOS

Análise e Gerência de Requisitos

1- A Engenharia de Requisitos de Software

I- Introdução

“Nosso mundo é um lugar no qual as pessoas não sabem aquilo que querem e estão dispostas a passar pelo inferno para consegui-lo.” (Don Marques)

O inferno dos requisitos é aquele círculo particular do Inferno em que Sisyphus está empurrando uma pedra montanha acima, somente para vê-la rolar de volta para baixo. Freqüentemente mal interpretado esse mito possui raízes na realidade. Sisyphus, na verdade, alcançou o topo da montanha muitas vezes; só que ele fica se perguntando se já terminou, com o infeliz resultado de ser forçado a recomeçar.

Talvez a resposta seja não perguntar se os requisitos ficaram corretos. Por outro lado, suspeito que a resposta seja apenas que você precisa seguir em frente, rolando a pedra dos requisitos montanha acima. Este curso prepara o terreno de modo que você pelo menos não escorregue e role montanha acima.

As necessidades do usuário são, ou pelo menos deveriam ser, o ponto de partida para o início de um projeto. A ambigüidade e sua resolução em requisitos declarados e validados de maneira compreensível são a plataforma da qual você prosseguirá para o projeto. Dar prioridade aos requisitos permite que você desenvolva um plano de projeto significativo, adiando itens de menor prioridade para projetos posteriores.

Finalmente, compreender o escopo de seus requisitos permite que você compreenda de que tipo de arquitetura terá o seu projeto.

1.1- O desafio do levantamento dos requisitos

Determinar e gerenciar requisitos, embora seja uma das mais importantes atividades do processo de desenvolvimento de um projeto é, muitas vezes, tratada com pouco ou até nenhum esmero.

Muitas situações, tais como uma solicitação de um cliente para desenvolver um produto ou realizar um serviço; uma solicitação de um amigo para realizar um favor; ou uma solicitação de um gerente para realizar uma tarefa requer uma definição de requisitos.

Determinar os requisitos é, em última análise, entender exatamente **o que** deve ser feito e **o que** se espera receber como resultado. Neste ponto ainda não se questiona **como** o trabalho será realizado.

ANÁLISE E GERÊNCIA DE REQUISITOS

1.1.1- As Features de um Sistema

Ambigüidade e Persistência

A coleta de requisitos faz parte de todo projeto, e as técnicas se aplicam a qualquer sistema. Esta seção resume alguns conselhos gerais com relação a requisitos e se aprofunda em requisitos relacionados a banco de dados, mas que podem ser utilizados para qualquer projeto.

Ambigüidade

A ambigüidade pode tornar a vida interessante. Entretanto, a menos que você goste do vaivém de usuários e programadores irritados, sua meta ao coletar requisitos é reduzir a ambigüidade até o ponto em que você possa produzir um projeto de banco de dados útil, que faça o que as pessoas querem.

Como exemplo, considere o livro de notas. É uma coleção de materiais de consulta que Sherlock Holmes construiu para complementar sua prodigiosa memória para fatos. Algumas das citações relevantes de Holmes ilustram os requisitos básicos.

Essa passagem resume a natureza do livro de notas:

"Procure-a, por gentileza, em meu índice, doutor", murmurou Holmes, sem abrir seus olhos. Por muitos anos, ele havia adotado um sistema para resumir todos os parágrafos relativos a homens e coisas, de modo que era difícil citar um sujeito ou uma pessoa sobre a qual ele não pudesse rapidamente fornecer informações. Neste caso, encontrei sua biografia espremida entre a de um rabino judeu e a de um comandante que havia escrito uma monografia sobre peixe do fundo do mar.
"Deixe-me ver", disse Holmes. "Ah! Nascida em New Jersey no ano de 1858. Contralto – Ah! La Scala – Ah! Prima donna da Ópera Imperial de Varsóvia – sim! Aposentada do palco da ópera – Ah! Vivendo em Londres – exatamente! Sua Majestade, pelo que sei, se envolveu com essa jovem criatura, escreveu algumas cartas comprometedoras e agora deseja conseguir essas cartas de volta".

Nem toda tentativa de encontrar informações é bem sucedida:

Meu amigo havia escutado com grande surpresa essa longa fala, que foi expressa com vigor e honestidade extraordinários, cada ponto sendo esclarecido pelo bater de uma mão forte no joelho do orador. Quando nosso visitante ficou em silêncio, Holmes esticou sua mão e foi até a letra 'S' de seu livro de notas. Dessa única vez, ele consultou em vão aquela mina de informações variadas.

"Há um Arthur H. Stauton, o jovem falsificador emergente", disse ele, "e houve um Henry Stauton, que eu ajudei a enforcar, mas Godfrey Stauton é um novo nome para mim"

Foi a vez de nosso visitante parecer surpreso.

ANÁLISE E GERÊNCIA DE REQUISITOS

A passagem a seguir ilustra as entradas biográficas de pessoas e sua relação com organizações criminosas.

“Apenas pegue para mim meu índice de biografias da estante.” Ele virava as páginas preguiçosamente, reclinando-se em sua cadeira e soprando grandes nuvens de seu charuto. “Minha coleção de “Emes” é ótima, disse ele. “Moriarty sozinho é suficiente para tornar qualquer carta ilustre, e aqui temos Morgan, o envenenador, e Merridew, de abominável memória, e Mathews, que arrancou meu canino esquerdo na sala de espera da Cruz Vermelha e finalmente, aqui temos nosso amigo de hoje à noite.”

Ele me passou o livro e li:

Moran, Sebastian, coronel. Desempregado. Pertenceu ao grupo dos Primeiros Pioneiros de Bangalore. Nascido em Londres, 1840. Filho de Sir Augustus Moran, C. B., já foi ministro britânico da Pérsia. Educado em Eton e Oxford. Serviu na Campanha de Jowaki, Campanha Afegã, de Charasiab (despachos), Sherpur e Cabul. Autor de Heavy Game of the Western Himalayas (1881), Three Months in the Jungle (1884). Endereço: Rua Conduit. Clubes The Anglo-Indian, Tankerville, Bagatelle Card Club.
Na margem estava escrito, com a precisa mão de Holmes:
O segundo homem mais perigoso de Londres.

Aqui temos um exemplo do uso prático do livro de notas na investigação criminal:

Ambos estamos em silêncio por algum tempo depois de escutar essa extraordinária narrativa. Então Sherlock Holmes tirou da prateleira um dos pesados livros de notas em que ele colocava seus recortes.

“Aqui está um anúncio que irá interessá-lo, disse ele. “Apareceu em todos os jornais há aproximadamente um ano. Escute só: “

“Desaparecido no dia nove do presente mês, Sr. Jeremiah Hayling, 26 anos, engenheiro hidráulico. Deixou seu alojamento às dez horas e não mais se ouviu falar dele. Estava vestindo..Etc ... etc... Ah! Essa representa a última vez que o coronel precisou mandar sua máquina ser revisada, suponho.”

E aqui temos outro exemplo, mostrando a maneira como Holmes adicionava notas marginais ao item original:

“Assim que nossa visitante deixou a sala bamboleando – nenhum outro verbo pode descrever melhor o método de movimentação da sra, Merrilow – Sherlock Holmes se jogou uma energia violenta sobre a pilha de livros de notas no canto. Por alguns minutos, houve um constante zunido das folhas e, então, com um grunhido de satisfação, ele encontrou o que procurava, Estava tão excitado que não levantou, mas sentou-se no chão como algum estanho Buda, com as pernas cruzadas, os enormes livros ao seu redor, e um aberto sobre seus joelhos.”
“O caso me preocupou na época, Watson. Aqui estão minhas notas marginais para provar. Confesso que não consegui nada. E, contudo, estava convencido de que o médico legista estava errado. Você não se lembra da tragédia do Abbas Parva?”

ANÁLISE E GERÊNCIA DE REQUISITOS

“não, Holmes. E você ainda estava comigo naquela época. Mas certamente minha impressão foi muito superficial. Pois não havia nada que se pudesse seguir e nenhuma das partes havia contratado meus serviços. Você se incomodaria de ler os jornais?”

Holmes também utiliza o livro de notas para rastrear casos que sejam paralelos a casos pelos quais um cliente se interesse:

“Um estudo muito interessante, aquela senhorita”, ele observou. “Achei-a mais interessante do que seu pequeno problema que, a propósito, é um tanto trivial. Você encontrará casos paralelos, se consultar meu índice, em Andover em 77, e houve algo do tipo em Haia no ano passado. Por mais que a idéia fosse velha, houve um ou dois detalhes que forma novos para mim. Mas a senhorita em si foi a mais instrutiva”.

Esse uso faz limite com outro tipo de consulta, o livro de casos:

“Todo o rumo dos eventos”, disse Holmes, “do ponto de vista do homem que se chamava Stapleton, era simples e direto, apesar de para nós, que não tínhamos meios, no início, de saber os motivos de seus atos e só tínhamos acesso a parte dos fatos, tudo parecia excessivamente completo. Tive a vantagem de duas conversas com a sra. Stapleton, e o caso agora foi tão inteiramente esclarecido que não tenho conhecimento de nada que tenha permanecido um grande segredo para nós. Você encontrará algumas notas sobre o assunto sob o título B em minha lista indexada de casos”.

Certas limitações do instrumento do livro de notas claramente limitavam as maneiras pelas quais Holmes ordenava seus casos:

"Matilda Briggs não era o nome de uma jovem senhorita, Watson", disse Holmes com uma voz rememorativa. "Era um navio que é associado com o rato gigante de Sumatra, uma história para qual o mundo ainda não está preparado. Mas o que sabemos sobre vampiros? Ou isso é de nossa competência? Qualquer coisa é melhor do que a estagnação, mas nós parecemos realmente ter sido ligados em um conto de fadas dos Grimm. Estique o braço, Watson, e veja o que tem a dizer."

Inclinei-me para trás e peguei o grande volume de índice a que ele se referia. Holmes o equilibrou sobre o joelho e seus olhos se moviam lenta e carinhosamente pelos registros de casos antigos, misturados com as informações acumuladas de toda uma vida.

"Viagem de Gloria Scott", ele leu. "Esse foi um mau negócio. Lembro-me de você ter feito um registro dele, Watson, apesar de eu não ter sido capaz de congratulá-lo pelo resultado. Victor Lynch, o falsificador. Lagarto ou monstro de Gila venenoso. Caso notável, esse! Vittoria, a bela do circo. Vanderbilt e o arrombador. Víboras. Vigor, a maravilha do martelo. Olá! Olá! Bom e velho índice. Insuperável. Escute isso, Watson, Vampirismo na Hungria. E, novamente, Vampiros na Transilvânia". Ele virou as páginas com avidez, mas depois de uma curta leitura atenta ele largou o grande livro com uma rosnada de descontentamento.

ANÁLISE E GERÊNCIA DE REQUISITOS

"Droga, Watson, droga! O que podemos fazer com cadáveres que andam e que só podem ser mantidos em seus túmulos por estacas atravessadas em seu coração? É pura insanidade!"

Essas citações mostram como Holmes construiu seus livros de notas:

Ele pegou o grande livro em que, dia após dia, ele arquivava as colunas de pedido de ajuda de vários jornais de Londres. "Meu Deus!", disse ele, virando as páginas, "que repetição de queixas, lamentações e lamúrias! Que conjunto de acontecimentos singulares! Mas certamente o terreno de caça mais valioso que já foi dado a um estudioso do incomum/ Essa pessoa está sozinha e não pode ser acessada por carta sem a quebra do sigilo absoluto que é desejado. Como alguma novidade ou mensagem pode alcançá-la?

Obviamente por anúncios em um jornal. Parece não haver nenhuma outra maneira, e felizmente precisamos nos preocupar apenas com o jornal.

Em uma noite de inverno, enquanto sentávamos perto da lareira, atrevi-me a sugerir a ele que, quando tivesse terminado decolar os recortes em seu livro de notas, ele deveria passar as duas horas seguintes tornando nosso quarto um pouco mais habitável.

O primeiro dia, Holmes havia passado fazendo um indexamento cruzado de seu enorme livro de consultas.

Quando a noite caiu, a tempestade foi ficando cada vez mais barulhenta e o vento gemia e soluçava como uma criança na chaminé. Sherlock Holmes sentou-se taciturnamente em um dos lados da lareira fazendo um indexamento cruzado de seus registros de crimes...

Essas passagens dos livros de Holmes ilustram a natureza meticulosa da abordagem de Holmes da investigação e a natureza da ambigüidade ao se coletar requisitos. Holmes, apesar de seu enorme esforço, era incapaz de conceber seu sistema de livros de notas em termos ambíguos. "Viagem de Gloria Scott", por exemplo, como entrada no volume 'V' parece um pouco misterioso.

A ambigüidade é um estado em que você encontra interpretações múltiplas e contraditórias em um texto ou situação. Você cria ambigüidades quando declara os requisitos para seu sistema ou banco de dados que alguém possa interpretar de maneiras diferentes.

Gause e Weinberg citam três tipos específicos de ambigüidade que são importantes ao se coletar requisitos:

- ***Requisitos ausentes****:* Necessidades que são uma parte principal da resolução de diferenças de

interpretação.

- ***Palavras ambíguas****:* Palavras na descrição do requisito que são capazes de múltiplas interpretações.
- ***Elementos introduzidos****:* Palavras que você coloca "na boca" dos interessados.

Ao projetar o sistema de livros de notas, talvez tenhamos um pouco menos de acesso a Holmes do que normalmente gostaríamos, considerando seu horário cheio e concisão geral. Essa situação certamente levará a requisitos ausentes e elementos introduzidos. Com alguma persistência, podemos acertar os detalhes.

Por exemplo, nas citações acima, o sistema parece requerer vários tipos de fato:

- ***Fatos biográficos****:* Fatos relacionados à história de pessoas individuais.
- ***Histórias de casos:*** Fatos relacionados à história de casos criminais do trabalho de Holmes e casos de interesse de todo o mundo.
- ***Artigos de enciclopédia****:* Fatos relacionados a coisas de interesse do detetive que estiver fazendo a consulta (vampiros, por exemplo, ou monstros de Gila).
- ***Anúncios classificados****:* As colunas de pedido de ajuda, nas quais as pessoas anunciam membros da família desaparecidos ou se comunicam umas com as outras.
- ***Artigos periódicos****:* Textos de jornais e revistas que contribuam com fatos interessantes relacionados a pessoas, lugares e eventos de interesse para o detetive que estiver fazendo a consulta.

Pensando um pouco, você pode ver que talvez possa haver muito mais fontes de informações que sejam críticas para o investigador. Holmes as excluiu por algum motivo ou elas simplesmente não apareceram na conversa? E quanto a artigos da Web? E quanto a dados do governo? E quanto a informações sobre corporações, parcerias limitadas e outras entidades de negócios? E comerciais informativos de televisão (relativos a fraudes de consumidores, por exemplo, possivelmente em grande escala)?

Além disso, temos que inferir, a partir das citações, que as informações do livro de notas não são aleatórias ou universais. Mas parece difícil fazer a afirmação direta de que o livro contém somente fragmentos de fatos relativos a coisas de interesse do detetive. Isso é ambíguo porque os termos são ambíguos (o que é de interesse?) e porque estamos colocando palavras na boca de Holmes. Como podemos solucionar essas questões?

Observando e fazendo as perguntas certas.

Em uma palavra: **exploração.**

Você precisa explorar os requisitos para esclarecer, o máximo possível, que problema você precisa resolver.

A primeira pergunta é simples: pode existir solução para o problema? Isto é, se você declarou o problema corretamente, existe, de fato, uma maneira de resolvê-lo?

Qual é o problema essencial que o sistema do livro de notas resolve? No trabalho de um detetive privado, é crítico raciocinar sobre uma situação com base em dados

factuais, tanto sobre a situação quanto sobre o contexto da situação. Nas citações, Holmes utiliza seu livro de notas escrito para conseguir os fatos que precisa para interpretar eventos e situações.
A declaração do problema, assim, talvez possa ser a seguinte:

Holmes PLC precisa de uma maneira de disponibilizar os fatos relevantes às suas consultas da prática de investigação para o detetive privado que estiver consultando.

Essa declaração lhe fornece o sabor do problema, mas não possui detalhes suficientes para declarar uma solução. Neste caso, porém, estamos começando com uma norma: uma solução existente. Com esse tipo de ponto de partida, você quer refinar a definição de seu problema utilizando o sistema existente como guia. No processo, você identifica as limitações do sistema e talvez os problemas subjacentes de que o sistema tenta tratar.

O passo seguinte na exploração é fazer perguntas livres de contexto, normalmente começando com os pronomes familiares *quem, o que, por que, quando, onde* e *como.* Perguntas sobre o processo indagam sobre a natureza do processo do projeto, perguntas sobre o produto indagam sobre a natureza do produto do projeto e meta perguntas fazem perguntas sobre as perguntas. Por exemplo, temos aqui alguns assuntos que talvez você possa levantar no primeiro conjunto de perguntas sobre o livro de notas:

*O cliente do livro de notas é o detetive privado que está fazendo a consulta no Holmes PLC. Esse sistema é proprietário e é parte da propriedade intelectual de Holmes PLC.
*O valor desse sistema para o cliente é de pelo menos 20 milhões de libras esterlinas ao longo de cinco anos de lucros ampliados. Esses lucros vem da melhor identificação de oportunidades de lucro e melhores resultados de investigações, levando a uma maior capacidade dos serviços de investigação serem colocados no mercado.
*Existe uma pressão cada vez maior por uma solução automatizada para esse problema por parte dos clientes, que querem acesso mais rápido a melhores informações do que o sistema atual fornece. Houve Godfrey Stauntons demais ultimamente.

*O conteúdo do sistema varia de informações críticas sobre criminosos e eventos criminosos a informações úteis sobre vampiros e víboras. As informações críticas incluem dados biográficos sobre criminosos e pessoas e organizações relevantes, histórias de casos do Holmes PLC e arquivos policiais e entradas das colunas de pedido de ajuda nos jornais (apesar dessa boa e velha instituição para anúncios e auxílio em papel agora estar sendo substituída I por salas de bate papo na internet).

*Nem todas as informações têm que estar disponíveis de imediato. Informações recentes são mais importantes, e você pode adicionar informações históricas quando o momento permitir. As informações biográficas de todos os criminosos conhecidos pela agência precisam estar presentes no primeiro lançamento. As informações no sistema atual em papel são mais importantes do que outras informações.

ANÁLISE E GERÊNCIA DE REQUISITOS

*Os clientes gostariam de ter o sistema existente em papel acessível por meio de busca via computador dentro de um ano. O sistema antigo deve continuar a existir até que o novo sistema esteja funcionando com informações equivalentes.

*Neste caso, as informações são mais importantes do que o tempo. Seria melhor ter informações mais completas do que fazer o sistema funcionar no prazo de um ano.

*As informações servem à necessidade dos detetives de informações factuais sobre criminosos e outros envolvidos em casos. Esses fatos fornecem a base para o trabalho investigativo e dedutivo e indutivo; sem eles, você não consegue raciocinar de modo eficaz. O sistema de computador resolve o problema de acessar uma enorme quantidade de informações rapidamente. Também resolve o problema de fazer indexações cruzadas das informações para o acesso mediante os conceitos. Existe aqui um potencial para conflitos, porque os mapas cognitivos de diferentes clientes diferem muito: Holmes talvez tivesse interesse em 'Viagens', enquanto Watson talvez quisesse procurar 'Gloria Scott'.

*A qualidade dos dados é vital. Informações erradas não são apenas sem valor; na verdade, elas impedem o pensamento dedutivo e indutivo. Isso significa que as ferramentas de validação de dados são um componente essencial do sistema, assim como os processos de realização da validação. Também significa que o banco de dados deve ser protegido de danos, ou por acidente ou por intenção maliciosa.

*A segurança durante o acesso ao sistema não somente é importante para a validação dos dados, ela também garante que o cliente mantenha a confidencialidade e sigilo durante investigações em andamento. Sujeitos de investigações não devem ser capazes de determinar se o sistema possui informações sobre eles ou se os clientes estão acessando tais informações.

*O problema mais importante com esse sistema é uma combinação de assuntos de invasão de privacidade e de propriedade intelectual. Grande parte do material a partir do qual Holmes PLC reúne informações é público. Com o passar do tempo, Holmes PLC utilizará cada vez mais informações reunidas a partir de informantes individuais, agências policiais e outras organizações privadas. Isso poderia causar problemas com as leis e regulamentações de privacidade, particularmente leis de grampeamento de cabos e de bisbilhotagem eletrônica em jurisdições variáveis em todo o mundo. Além disso, o material reunido de fontes públicas pode ser sujeito a direitos autorais ou outras restrições de propriedade intelectual.

Considerando tudo isso, você pode estar pensando onde está a ambigüidade.

Todas as declarações acima possuem ambigüidade; a questão é se existe ambigüidade suficiente para criar o risco de fazer a coisa errada em um nível intolerável.
Por exemplo, um requisito diz que ser, completo é vital, mesmo às custas de se aprontar o sistema em um ano. Você poderia não prosseguir com o requisito nesta forma.

Separe-o. O que significa 'completo'?
Poderia significar muitas coisas, dependendo do conjunto de dados em questão. Por exemplo, para biografias criminais, completo talvez possa significar todos os registros policiais relativos a uma pessoa, ou talvez possa significar registros policiais, artigos de jornal, relatórios do informante e inúmeras outras fontes de informação.

E quando o relatório do informante é 'completo'? Esses são os tipos de questão que você deve resolver voltando aos clientes, apresentando situações e sondando o significado mais profundamente. Você presumivelmente sairia desse processo com uma compreensão de como avaliar a adequação das informações sobre um sujeito, potencialmente com algum tipo de medida indicando o quanto a informação é adequada, ou uma medida de confiabilidade para informantes.
Esse tipo de métrica de metadados transmite informações reais ao cliente sobre o quanto eles podem confiar nos dados do banco de dados. É um erro primário fazer teorias antes de seus dados, e saber quais realmente são os seus dados torna-se vital para julgar a relevância de suas teorias.

Persistindo

Persistir aqui significa duas coisas: continuar rolando aquela pedra e descobrir o que os requisitos significam para seus dados persistentes.
Você pode passar sua vida inteira coletando requisitos em vez de desenvolver sistemas.

Apesar da aquisição interminável de conhecimento ser útil, a menos que alguém lhe pague somente para isso, em algum ponto você precisa começar a desenvolver seu software. Saber quando persistir ao reunir e interpretar requisitos e quando ir adiante vem com a experiência. Você pode utiliza medidas sofisticadas para julgar a adequação de seus requisitos, ou pode usar sua intuição.

Ambos exigem uma boa quantidade de experiência, ou para reunir dados comparativos ou apurar o senso a ponto de ser capaz de julgar que você já fez o suficiente. É importante sondar o nível de resolução de ambigüidades adequado às necessidades do projeto.

A persistência também se estende aos próprios dados. Em todos esses requisitos existem suposições sob dados persistentes. Examinar alguns dos requisitos lhe mostrará o que isso significa.

*O *cliente do livro de notas é o detetive privado que está fazendo a consulta no Holmes PLC. E, sistema é proprietário e é parte da propriedade intelectual de Holmes PLC.*

Nada muito relativo a banco de dados aqui.

*O *valor desse sistema para o cliente é de pelo menos 20 milhões de libras esterlinas ao longo cinco anos de lucros ampliados. Esses lucros vêm da melhor identificação*

de oportunidades lucro e melhores resultados de investigações, levando a uma maior capacidade dos serviços investigação serem colocados no mercado.

Nem aqui.

**Existe uma pressão cada vez maior por uma solução automatizada para esse problema por parte dos clientes, que querem acesso mais rápido a melhores informações do que o sistema atual fornece. Houve Godfrey Stauntons demais ultimamente.*

Aqui aparece a primeira suposição sobre a tecnologia subjacente. 'Acesso mais rápido a melhores I informações' implica o armazenamento dessas informações e a existência de caminhos de acesso a elas. A única qualidade é 'mais rápido', que poderia significar qualquer coisa. Neste caso, 'mais rápido' refere-se ao sistema atual em papel.

Forçar um maior aprofundamento sobre o requisito provavelmente traria à tona alguns detalhes como a necessidade de acesso a partir de instalações móveis, a necessidade de resposta imediata em uma rede de vasta amplitude, e a distribuição dos dados a locais do Holmes PLC ao redor do mundo. 'Mais rápido' torna-se, então, relativo ao sistema atual, em que os detetives precisam ligar para um escritório central para obter informações, em que os pesquisadores procuram arquivos em sistemas de armazenamento em papel. O termo 'melhor' provavelmente refere-se ao último requisito sobre dados 'completos', significando que não devemos somente passar o banco de dados para a forma eletrônica, mas também melhorar seu conteúdo.

*O conteúdo do sistema varia de informações críticas sobre criminosos e eventos criminosos a informações úteis sobre vampiros e víboras. As informações críticas incluem dados biográficos sobre criminosos e pessoas e organizações relevantes, histórias de casos do Holmes PLC e arquivos policiais e entradas das colunas de pedido de ajuda nos jornais.

Aqui temos alguns dados específicos que devem persistir: informações biográficas e informações policiais sobre criminosos. Também há informações sobre outras pessoas relevantes, informações sobre organizações (criminais, corporativas, não lucrativas, e assim por diante), dados de histórias de casos e publicações da mídia que refletem potencial criminoso ou atividade 'interessante'.

*Nem todas as informações devem estar disponíveis imediatamente. Informações recentes são mais importantes, e você pode adicionar informações históricas quando o momento permitir. As informações biográficas de todos os criminosos conhecidos pela agência precisam estar presentes no primeiro lançamento. As informações no sistema atual em papel são mais importantes do que outras informações.

Esse requisito é similar ao requisito de 'acesso mais rápido' que veio anteriormente.

ANÁLISE E GERÊNCIA DE REQUISITOS

*Os clientes gostariam de ter o sistema existente em papel acessível por meio de busca via computador dentro de um ano. O sistema antigo deve continuar a existir até que o novo sistema esteja funcionando com informações equivalentes.

Este requisito nos ajuda a priorizar os requisitos de dados. Veja a próxima seção para obter mais detalhes.

* Neste caso, as informações são mais importantes do que o tempo. Seria melhor ter informações mais completas do que fazer o sistema funcionar no prazo de um ano.

Esses são requisitos amplos relativos ao conteúdo dos dados. A última seção discutiu o enorme grau de ambigüidade nesse requisito, então será necessário um pouco de s9ndagem para refiná-lo em algo útil. Ele é, entretanto, um requisito crítico para os dados persistentes. 'Completo', neste sentido, certamente se aplica aos dados persistentes, não ao software que os acessam. Você deve perceber, porém, que esse requisito é um tanto incomum; o tempo é, freqüentemente, tão ou mais importante quanto as informações serem completas. Às vezes é melhor ter dados incompletos agora do que dados completos na terça-feira, depois do réu acusado ser absolvido pelo júri por falta de evidências. Esse banco de dados se concentra em uma perspectiva a mais longo prazo, que é vital para compreender como estruturar o aplicativo e o banco de dados.

*As informações servem à necessidade dos detetives de informações factuais sobre criminosos e outros envolvidos em casos. Esses fatos fornecem a base para o trabalho investigativo dedutivo e indutivo; sem eles, você não consegue raciocinar de modo eficaz. Um sistema computacional resolve o problema de acessar uma enorme quantidade de informações rapidamente. Também resolve o problema de fazer referências cruzadas das informações para o acesso por meio de conceitos. Existe aqui um potencial para conflitos, porque os mapas cognitivos de diferentes clientes diferem muito: Holmes talvez tivesse interesse em 'Viagens', enquanto Watson talvez quisesse procurar 'Gloria Scott'.

Aqui temos dois elementos que são relativos a dados persistentes: métodos de acesso e metadados. A 'enorme quantidade' implica que haverá terabytes de dados, o que exige um software especial de gerenciamento de banco de dados, extensas análises de hardware e requisitos específicos para caminhos de acesso como indexação, aglomeração e particionamento. Tudo isso será parte do projeto do banco de dados físico. A 'referência cruzada' também exige dados sobre os dados - dados que descrevam os dados de forma que permitam o rápido acesso aos dados que você queira ver.

Existem várias maneiras de organizar metadados. Você pode confiar na busca de texto, o que não é particularmente bom para encontrar informações relevantes. Você pode desenvolver um sistema de palavras-chave, que exige muito trabalho, mas paga os dividendos com velocidade de acesso: foi isso que Holmes fez com seus 'índices' em papel. Você pode utilizar um simples esquema enumerativo como o sistema decimal de Dewey que as bibliotecas utilizam ou pode desenvolver sistemas mais elaborados *utilizando facetas,* dimensões do domínio do problema em várias

combinações. Esse é o estilo clássico do armazenamento de dados, por exemplo. Sondar um pouco pode elucidar requisitos, neste caso, para indexações extensas, deixando o cliente adicionar facetas ou classificações ao esquema, à medida que for necessário, em vez de depender de uma representação estática.

*A qualidade dos dados é vital. Informações erradas não são apenas sem valor; na verdade, elas impedem o pensamento dedutivo e indutivo. Isso significa que as ferramentas de validação de dados são um componente essencial do sistema, assim como os processos de realização da validação. Também significa que o banco de dados deve ser tolerante a falhas, ou por acidente ou por más intenções.

Esta definitivamente causa impacto sobre dados persistentes. Ela significa várias coisas. Primeiro, você deve ter alguma maneira de saber se as informações contraditórias são parte da realidade que seu banco de dados representa ou simplesmente estão erradas. As pessoas discordam, por exemplo, na recordação de eventos e coisas. Esse não é um problema de qualidade dos dados, mas uma questão de informação. Você não precisa se preocupar com nada além da qualidade dos dados. Segundo, você deve estabelecer um processo para entrada e validação de dados. Terceiro, você deve errar -por precaução -ao estabelecer restrições de reforço ou integridade referencial e outras regras de negócios. Quarto, você deve submeter-se ao banco de dados em tal reforço. Quinto, você deve impor segurança sobre os dados persistentes. Você deve proteger os dados que entram e saem.

*A segurança durante o acesso ao sistema não somente é importante para a validação dos dados, ela também garante que o cliente mantenha a confidencialidade e o sigilo durante investigações em andamento. Sujeitos de investigações não devem ser capazes de determinar se o sistema possui informações sobre eles ou se os clientes estão acessando tais informações.

Apesar de você talvez achar que esse requisito afeta o software em vez de os dados persistentes, ele também pode afetar os dados. Por exemplo, se você quer a máxima segurança de um esquema de segurança de acesso obrigatório, precisa associar rótulos de segurança com os dados em todos os níveis do sistema, inclusive o banco de dados e o sistema operacional. Isso restringe rapidamente suas escolhas de tecnologia, pois não existem tantos sistemas operacionais e gerenciadores de banco de dados que suportam esse nível de segurança. Também há implicações para autenticação e acesso aos dados persistentes, que se sobrepõem ao requisito anterior, relativo à integridade dos dados. Além disso, a espionagem está hoje mais fácil do que nunca com o acesso a dados via internet. Holmes PLC depende da confidencialidade de seus segredos comerciais e outros tipos, de propriedade intelectual no banco de dados. A segurança no nível do banco de dados protege esses segredos do acesso mediante outros mecanismos que não sejam aplicativos.

*O problema mais importante com esse sistema é uma combinação de assuntos de invasão de privacidade e de propriedade intelectual. Grande parte do material a partir do qual Holmes PLC reúne informações é público. Com o passar do tempo, Holmes PLC utilizará cada vez mais informações reunidas a partir de informantes individuais,

agências policiais e outras organizações privadas. Isso poderia causar problemas com as leis e regulamentações de privacidade, particularmente leis de grampeamento de cabos e de bisbilhotagem eletrônica em jurisdições variáveis em todo o mundo. Além disso, o material reunido de fontes públicas pode estar sujeito a direitos autorais ou outras restrições de propriedade intelectual.

A maior implicação desse requisito é uma necessidade de armazenar licenças e permissões relativas a dados proprietários e restritos. Esse é um outro tipo de dados de segunda ordem. Também é relativo ao requisito de segurança, pois manter propriedade intelectual segura é freqüentemente necessário, assim como a privacidade dos dados, quando se trata de indivíduos.

Acertando suas prioridades

Coletar os requisitos não é o fim de seu trabalho. Nem todo requisito possui a mesma importância dos outros. A tarefa seguinte é priorizar os requisitos compreendendo-os, relacionando-os uns aos outros e categorizando-os. É um processo iterativo que freqüentemente exige uma análise mais profunda por meio de casos de uso, o assunto do próximo capítulo.

Até você chegar a um nível razoável de detalhes, pode haver dificuldade para perceber quais são as verdadeiras prioridades e que coisas dependem de outras coisas. Quando você chegar ao projeto, geralmente achará que existem dependências tecnológicas que também deve levar em consideração, forçando-o a revisar novamente suas prioridades. Porém, quando você possuir seus requisitos básicos, pode iniciar a criação de uma estrutura para o desenvolvimento do sistema e do banco de dados.

Compreendendo os requisitos

O fato de você possuir uma lista do que as pessoas esperam que seu sistema faça, não significa que você compreende os requisitos. A lógica do erro afirma que todo sistema resiste aos esforços dos seres humanos em mudá-lo.

A primeira coisa que você deve fazer é parar sua caçada para reunir esforços e pensar neles. E pense não apenas nos requisitos que você reuniu, mas também nos que você não reuniu, e em seus efeitos colaterais. Envolva nesse esforço pessoas que compreendam o domínio e possuam uma extensa experiência na resolução de problemas nesse domínio. Faça treinamentos em resolução de problemas.

Considere as seguintes recomendações da psicologia cognitiva:

- Declare metas de maneira clara.
- Compreenda onde você deve acomodar metas contraditórias.
- Estabeleça prioridades e, então, modifique-as quando necessário.
- Modele o sistema, incluindo os efeitos colaterais e modificações a longo prazo.
- Compreenda de quantos dados você precisa e consiga-os.

- Não seja excessivamente abstrato.
- Não reduza tudo a um resumo, uma única causa ou fato.
- Escute os outros quando eles lhe disserem que você está indo pelo caminho errado.
- Aplique métodos quando for útil e evite-os quando eles atrapalharem.
- Estude sua história e aplique as lições aprendidas ao novo sistema.
- Pense em termos de sistemas, não em termos de simples requisitos.

Quanto mais você fizer, e quanto mais você aprender com seus erros, melhor você ficará em reunir requisitos.
Categorizando requisitos
No nível mais fundamental, você pode categorizar requisitos em objetivos operacionais, propriedades de objetos, regras e preferências.

Objetivos operacionais
Os *objetivos operacionais* expressam o propósito existente por trás do sistema e a abordagem que você pretende assumir para alcançar esse propósito. Metas, funções, comportamentos, operações são todos sinônimos de objetivos operacionais.

Objetivos são o tipo mais importante de requisito, porque eles expressam o significado por trás do sistema, o porquê do que você está fazendo.
Por exemplo, considere o seguinte requisito da última seção:
*O *conteúdo do sistema varia de informações críticas sobre criminosos e eventos criminosos a informações úteis sobre vampiros e víboras. As informações críticas incluem dados biográficos sobre criminosos e pessoas e organizações relevantes, histórias de casos do Holmes PLC e arquivos policiais e entradas das colunas de pedido de ajuda nos jornais.*

Esse requisito expressa vários objetivos operacionais do sistema do livro de notas relativo a necessidades de informações que se sobrepõem:

- Fornecer aos detetives as informações de que eles precisam para conduzir seu trabalho por meio de uma interface de computador de banco de dados adequada.
- Fornecer informações sobre criminosos conhecidos mediante dados biográficos e de histórias de casos.
- Fornecer informações sobre pessoas de interesse dos detetives em seu trabalho com gados biográficos e anúncios pessoais em várias mídias.
- Fornecer informações sobre casos criminais de interesse por meio de dados de histórias de casos do Holmes PLC e investigações policiais e através de notícias de várias mídias.
- Fornecer informações sobre fatos de uso em investigações criminais mediante entradas de enciclopédia (vampiros e víboras).

- Permitir a integração rápida e fácil de informações atualizadas através de tecnologias de interface flexíveis que vinculem o sistema a outros sistemas similares de propriedade da polícia e fontes de mídia para obter grande parte das informações exigidas pelos clientes.

Esses requisitos expressam um propósito (fazer algo) e uma maneira de alcançar esse propósito (através de dados ou comportamento). Alguns desses objetivos, como os acima citados, são relativos ao usuário do sistema. Outros objetivos podem ser escondidos do usuário, existindo somente para capacitar algum outro objetivo ou para tornar o sistema mais flexível ou mais fácil de ser utilizado. Neste caso, o cão que não latiu à noite (a pista que disse a Holmes que o rapto de Silver Blaze foi um negócio interno) é o requisito de interface. O sistema precisa possuir uma interface de dados flexível que deixe Holmes integrar facilmente novas informações ao banco de dados. O detetive não se importa realmente com a maneira com que as informações entram no sistema, apenas que elas estejam disponíveis quando necessário.

Propriedades de objetos

Quando você começar a desenvolver os requisitos de dados para o sistema, começará a ver os objetos (entidades, tabelas, classes, e assim por diante) que existirão no sistema. Quando você pensar neles, começará a ver suas propriedades. Propriedades de objetos são a segunda categoria de requisito, e provavelmente a mais importante para o projetista do banco de dados. As propriedades podem ser colunas de tabelas, atributos de objetos ou membros de dados de classe.

Por exemplo, um objeto-chave no requisito de informações da última seção é a história de caso. Você pode pensar imediatamente em certas propriedades básicas deste objeto:

- Um identificador para o caso.
- Um conjunto de jurisdições com interesse no caso.
- Um conjunto de pessoas associadas ao caso em vários papéis (criminoso, cúmplice, investigador, criminalista, médico examinador, testemunha, vítima, juiz, jurado, entre outros).
- Um conjunto de fatos comprobatórios (procure o caso do assassinato de Nicole Simpson para ver as minúcias que podem resultar).
- O valor da propriedade envolvida no caso.
- .As despesas da agência para o caso.

Estes são os elementos clássicos dos dados. Nem todas as propriedades de objetos estão diretamente relacionadas aos elementos do banco de dados, é claro: confiabilidade, capacidade de ser utilizável, não toxicidade são todas propriedades de objetos que se aplicam ao sistema de software em geral. Em sua maioria, boas propriedades de objetos são muito específicas: o sistema deve estar disponível 24 horas por dia, ou as informações devem refletir todas as informações relativas a morte ou danos, ou a resposta do sistema deve levar dois segundos ou menos para

qualquer informação determinada. Muita ambigüidade se esconde nas propriedades de objetos: solucione-as.

Regras

Regras são requisitos condicionais sobre propriedades de objetos. Para o valor de uma propriedade ser aceitável, ele deve satisfazer a condição.
A maioria de restrições de integridade referencial se enquadra nessa categoria de requisito. Por exemplo, se uma história de caso se refere a determinado criminoso sob um pseudônimo, criminoso e pseudônimo devem existir no banco de dados.

As regras de regulação de tempo também se enquadram nesta categoria. Por exemplo, você pode possuir um objetivo operacional de que o sistema responda uma solicitação de informações dentro de dois segundos. Isso se traduz em uma regra de que todas as informações do banco de dados devem possuir características de acesso que permitam recuperação em dois segundos ou menos.

Uma terceira categoria de regra é a regra de qualidade. Você pode especificar tolerâncias de erro que se tornem parte de seus requisitos. Por exemplo, se você armazenar evidências de DNA, erros de identificação da pessoa individual devem ser menores do que 0,0001, o que significa que os resultados são estatisticamente significativos, com probabilidade 0,9999. Esses tipos de regras estabelecem tolerâncias ou limites além dos quais os dados não podem passar.

Pense um pouco antes de criar uma regra. As regras têm a tendência a se tornarem leis, particularmente em um projeto de software com prazo de entrega. Esteja razoavelmente seguro de que sua regra é necessária antes de impô-la a seu projeto. Você pode ter que conviver com ela por muito tempo.

Preferências

Uma *preferência* é uma condição sobre um objeto que expressa um estado preferido. Por exemplo, um detetive preferiria conhecer o criminoso por trás de cada crime. De maneira realista, você deve ter histórias de casos não solucionados no sistema - histórias de casos que não tenham criminosos. Entretanto, você pode expressar uma preferência de que as histórias de casos tenham um criminoso com um requisito de que a fonte de informações faça o máximo esforço para especificar essa informação. Alternativamente, é possível qualificar 'suspeitos' ou algo similar com algum tipo de valor de probabilidade para permitir que os detetives prossigam.

Você também pode utilizar valores iniciais ou valores padrão - valores que o banco de dados atribuirá se você não especificar nada - para representar preferências.

Normalmente, as preferências assumem a forma do 'máximo possível', 'ótimo', 'em equilíbrio' ou algo similar. Você precisa compreender a diferença entre regras e preferências. Uma regra diz 'deverá', 'irá' ou 'poderá'. É uma questão de integridade *versus* uma questão de desejo (assim como tantos outros aspectos da vida).

ANÁLISE E GERÊNCIA DE REQUISITOS

Relacionando os requisitos

Os requisitos não são pérolas de conhecimento isoladas que existem independentemente uns dos outros. O próximo passo na compreensão de requisitos é tratá-los como um sistema, relacionando-os uns aos outros. A combinação sinérgica de requisitos geralmente pode mudar a natureza da sua abordagem do projeto.

Com *requisitos de dados*, você normalmente expressa relacionamentos diretamente por meio de modelos de dados que contenham os objetos e seus relacionamentos. Requisitos mais gerais possuem relacionamentos mais gerais.

Por exemplo, armazenar uma história de caso relacionada a um criminoso exige relacionar um criminoso a algum tipo de evento. O criminoso é, de alguma forma, uma propriedade do evento. A existência do criminoso como objeto separado torna-o um objeto no sistema que se relaciona com outros objetos como crimes.

De maneira mais geral, o sistema do livro de notas relaciona histórias de casos por analogia. Isto é, alguns casos são 'parecidos' com outros casos. Os detetives adorariam (amariam, na verdade) recuperar todos os casos que são parecidos com o que eles estão investigando. Esse é um problema difícil e improvável de ser transformado nos sérios objetivos para o livro de notas. Ainda assim, existem relacionamentos entre casos que exigem regras adicionais no banco de dados. Por exemplo, uma linguagem de. classificação consistente para crimes poderia expressar boa parte da similaridade entre os crimes, inclusive motivo, meio e oportunidade. Na verdade, é bem difícil expressar isso em um formato de banco de dados (Muller, 1982], mas você poderia tentar.

É importante compreender como os requisitos se aglomeram. As conexões que você pode estabelecer entre os requisitos normalmente lhe dirão como a arquitetura de seu sistema pode organizar seus subsistemas. Se em determinado conjunto os requisitos se relacionam fortemente uns com os outros, mas são relativamente desconexos de outros requisitos, você possui um candidato a um subsistema. Por outro lado, se você sentir intuitivamente que vários requisitos pertencem uns aos outros, mas não encontrar nenhum relacionamento entre eles, pense mais. Você deixou escapar alguma coisa.

No sistema do livro de notas, por exemplo, a história de caso é um requisito central. Você pode vincular uma história de caso a criminosos, crimes e outros eventos, pessoas, organizações e investigações. Vampiros, por outro lado, vinculam-se a muito pouco. Os primeiros objetos estão no coração do subsistema relativo ao principal propósito do livro de notas: registrar informações sobre eventos passados e pessoas para ajudar a solucionar investigações atuais. As histórias de casos reúnem todos os caminhos diferentes em um único. A entrada de enciclopédia vampiro, entretanto, é uma informação quase aleatória que, por acaso, foi de interesse em um único caso. Esse item, e itens como esse, provavelmente pertencem a um subsistema de enciclopédia separado, fora do escopo do sistema de histórias de casos.
Também existe um íntimo relacionamento interno entre pessoas, criminosos e organizações.

Esses requisitos provavelmente também pertencem a um subsistema, mostrando que os sistemas podem conter outros sistemas, ou poder se sobrepor a eles. A organização criminosa da qual Moran fazia parte era um alvo-chave dos esforços investigativos de Holmes. Isso sugere que ela merece um subsistema de dados separado para rastrear a rede complexa de relacionamentos entre pessoas, organizações e eventos. Esse sistema se sobrepõe ao sistema biográfico e ao sistema de histórias de casos. Você pode ver o despertar do dragão arquitetônico mesmo na etapa inicial dos requisitos.

Relacionamentos como esses estão no domínio do analista. Geralmente, parte de um sistema como esse incluirá data mining, análise estatística e outras ferramentas desse tipo para auxiliar o analista a identificar e quantificar os relacionamentos.
O crescente impacto de sua compreensão dos requisitos sobre o escopo e a direção de seu projeto leva naturalmente ao próximo assunto: priorizando o que você compreendeu.

Priorizando os requisitos

Os requisitos assumem muitos aspectos. Alguns são vitais para o projeto; outros, são menos importantes. Infelizmente, você precisa passar por muitos requisitos insubstanciais para treinar sua mente *a distinguir entre algo insubstancial e a coisa verdadeira.* Um aspecto disso é a relatividade das prioridades. Elas dependem da natureza das pessoas envolvidas no projeto.

Essa configuração de crenças e atitudes pode mudar radicalmente de projeto para projeto. De muitas formas, você deve ver as prioridades como completamente novas em cada projeto. Cada um dos diferentes tipos de requisito possui um diferente conjunto de prioridades.

Devem-se priorizar os objetivos operacionais por sua contribuição à missão básica do sistema. Se o sucesso do sistema exige o alcance bem sucedido do objetivo, este *é crítico*. Se você pode pular o objetivo com apenas pequenos problemas, ele será marginal. Se o objetivo não possui relação com a missão, ele é um *requisito ruim.*

Devem-se priorizar as propriedades de objetos em categorias similares: obrigatória, desejável e ignorável. Uma propriedade 'obrigatória' é uma propriedade sem a qual o sistema não realizará sua missão. Uma propriedade 'desejável' é algo bom de ter. Uma propriedade 'ignorável' é algo que talvez seja útil, mas provavelmente não vale ou não vale o custo de se construí-la.

Devem-se priorizar as regras e preferências por sua natureza. As regras se aplicam a situações 'obrigatórias'. Se você expressou uma regra, precisa construí-la. As preferências, por outro lado, vêm em tonalidades de cinza, não em preto-e-branco. Definir 'possível' ou 'ótimo' é uma das coisas mais difíceis a fazer na coleta de requisitos, porém, trata-se de algo necessário.

ANÁLISE E GERÊNCIA DE REQUISITOS

1.1.2- Técnicas de Levantamento de Requisitos

1.1.2.1-Entrevistas

A entrevista é um instrumento fundamental que pode ter uma grande variedade de objetivos, como no caso do jornalista, chefe de empresa, diretor de escola, professor, juiz, etc. Aqui nos interessa a entrevista, entendida como aquela na qual se busca a investigação dos requisitos.

A entrevista se baseia em uma técnica com seus procedimentos ou regras empíricas com os quais não só se amplia e se verifica como também, ao mesmo tempo, se aplica o conhecimento científico. A técnica é o ponto de interação entre a ciência e as necessidades práticas; é assim que a entrevista alcança a aplicação de conhecimentos científicos e, ao mesmo tempo, obtém ou possibilita levar a vida diária do ser humano ao nível do conhecimento e da elaboração científica. E tudo isso em um processo ininterrupto de interação.

A entrevista é um instrumento muito difundido e devemos delimitar o seu alcance em função de suas regras ou indicações práticas de sua execução.

A entrevista pode ser de dois tipos fundamentais: **aberta** e **fechada**.

- Na segunda as perguntas já estão previstas, assim como a ordem e a maneira de formulá-las, e o entrevistador não pode alterar nenhuma destas disposições.
- Na entrevista aberta, pelo contrário, o entrevistador tem uma liberdade para as perguntas ou para suas intervenções, permitindo-se toda a flexibilidade necessária em cada caso particular.

A entrevista fechada é, na realidade, um questionário que passa a ter uma relação estreita com a entrevista, na medida em que uma manipulação de certos princípios e regras facilita e possibilita a aplicação do questionário.

Contudo, a entrevista aberta não se caracteriza essencialmente pela liberdade de colocar perguntas, pois há uma flexibilidade suficiente para permitir, na medida do possível, que o entrevistado configure o campo da entrevista segundo sua estrutura psicológica particular, ou - dito de outra maneira – que o campo da entrevista se configure, o máximo possível, pelas variáveis que dependem da personalidade do entrevistado.

De outro ponto de vista, considerando o número de participantes, distingue-se a entrevista em individual e grupal, segundo sejam um ou mais os entrevistadores e/ou os entrevistados. A realidade é que, em todos os casos, a entrevista é sempre um fenômeno grupal, já que mesmo com a participação de um só entrevistado sua relação com o entrevistador deve ser considerada.

Podem-se diferenciar também as entrevistas segundo o beneficiário do resultado: assim, podemos distinguir:

- a) entrevista que se realiza em benefício do entrevistado - que é o caso do levantamento de requisitos;

- b) a entrevista cujo objetivo é a pesquisa, na qual importam os resultados científicos;
- c) a entrevista que se realiza para um terceiro (uma instituição).

Cada uma delas implica variáveis distintas a serem levadas em conta, já que modificam ou atuam sobre a atitude do entrevistador, assim como do entrevistado, e sobre o campo total da entrevista.
Um ponto fundamental é que o entrevistador desperte interesse e participação, que "motive" o entrevistado.
O certo é que não há possibilidade de uma entrevista correta e frutífera se não se incluir a investigação. Em outros termos, a entrevista é um campo de trabalho no qual se investiga os requisitos.
Uma utilização correta da entrevista integra na mesma pessoa e no mesmo ato o profissional e o pesquisador.

A **chave fundamental** da entrevista está na investigação que se realiza durante o seu transcurso. As observações são sempre registradas em função das hipóteses que o observador vai emitindo.

Afirma-se, geralmente de maneira muito formal, que a investigação consta de etapas nítidas e sucessivas que se escalonam, uma após a outra, na seguinte ordem: primeiro intervém a observação, depois a hipótese e posteriormente a verificação. O certo, contudo, é que a observação se realiza sempre em função de certos pressupostos e que, quando estes são conscientes e utilizados como tais, a observação se enriquece.

Assim, a forma de observar bem é ir formulando hipóteses enquanto se observa, e durante a entrevista verificar e retificar as hipóteses no momento mesmo em que ocorrem em função das observações subseqüentes, que por sua vez se enriquecem com as hipóteses prévias. Observar, pensar e imaginar coincidem totalmente e formam parte de um só e único processo dialético. Quem não utiliza a sua fantasia poderá ser um bom verificador de dados, porém nunca um investigador.

Quanto ao tipo de comunicação que se estabelece na relação entrevistador-entrevistado, é altamente significativo da personalidade do entrevistado, especialmente do caráter de suas relações interpessoais, ou seja, da modalidade do seu relacionamento com seus semelhantes.

Nesse processo que se produz na entrevista, o entrevistador observa como e através do que o entrevistado condiciona, sem o saber, efeitos dos quais ele mesmo não tem consciência. Interessam particularmente os momentos de mudança na comunicação e as situações e temas ante os quais ocorrem, assim como as inibições, interceptações e bloqueios.

O tipo de comunicação não é importante apenas por oferecer dados de observação direta que, inclusive, podem ser registrados, mas porque é o fenômeno-chave de toda a relação interpessoal, que, por sua vez, pode ser manipulado pelo entrevistador e, assim, graduar ou orientar a entrevista.

1.1.2.2- Workshops

Preparação do Workshop

Conduzir um workshop de requisitos implica reunir todos os envolvidos durante um período intensivo, concentrado. Um Analista de Sistemas atua como um facilitador da reunião. Todos os participantes deverão contribuir ativamente e os resultados da sessão deverão ser disponibilizados imediatamente para eles.

O workshop de requisitos fornece um framework para aplicar as outras técnicas de identificação como, por exemplo, brainstorming, encenação, interpretação de papéis e revisão dos requisitos existentes. Essas técnicas poderão ser usadas isoladamente ou combinadas. Todas poderão ser combinadas ao método de caso de uso. Por exemplo, você poderá criar uma ou algumas encenações para cada caso de uso previsto para o sistema. Você poderá usar a interpretação de papéis como uma maneira de compreender como os atores usarão o sistema e para ajudá-lo a definir os casos de uso.

O facilitador de um workshop de requisitos precisará estar preparado para as seguintes dificuldades:

- É possível que os envolvidos saibam o que desejam, mas não consigam expressar isso.
- É possível que os envolvidos não saibam o que desejam.
- Os envolvidos poderão achar que sabem o que desejam até que você lhes dê o que eles disseram que desejavam.
- Os analistas poderão achar que compreendem os problemas dos usuários melhor do que eles próprios.
- As pessoas poderão achar que as demais pessoas estão politicamente motivadas.

Os resultados dos workshops de requisitos serão documentados em um ou mais artefatos de solicitações dos principais envolvidos. Desde que você tenha boas ferramentas de suporte, geralmente é aconselhável permitir que os envolvidos relatem informações desse tipo.

Se tiver optado por discutir o sistema em termos de atores (Um ator define um conjunto coerente de papéis que os usuários do sistema podem desempenhar ao interagir com ele). Uma instância de ator pode ser desempenhada tanto por um indivíduo quanto por um sistema externo e casos de uso (Um caso de uso define um conjunto de instâncias de casos de uso, no qual cada instância é uma seqüência de ações realizada por um sistema que produz um resultado de valor observável para determinado ator.), você também poderá ter um esquema de modelo de casos de uso (o modelo de casos de uso é um modelo das funções pretendidas do sistema e seu ambiente, e serve como um contrato estabelecido entre o cliente e os desenvolvedores. O modelo de casos de uso é usado como fonte de informações essencial para atividades de análise, design e teste).

Antes do Workshop

O facilitador precisará "vender" o workshop para os envolvidos que deverão estar presentes e estabelecer o grupo que participará dele. Os futuros participantes deverão receber material de estudo de "aquecimento" antes do workshop. O facilitador é o responsável pelas atividades logísticas relacionadas ao workshop como, por exemplo, enviar convites, encontrar um local adequado com o equipamento necessário para a sessão, assim como distribuir uma agenda referente ao workshop.

Conduzir a Sessão

O facilitador presidirá a sessão, o que inclui:

- Dar a todos a oportunidade de falar.
- Manter a sessão sob controle.
- Reunir informações para Atributos de Requisitos aplicáveis.
- Registrar as descobertas.
- Resumir a sessão e elaborar conclusões.

Consolidar Resultados

Após o workshop de requisitos, o facilitador (juntamente com os colegas analistas de sistemas) precisará dedicar um tempo à síntese das descobertas e ao resumo das informações em um formato apresentável.

Truques do Negócio

A tabela abaixo lista um conjunto de problemas e as soluções sugeridas que poderão atender às necessidades do facilitador. As soluções referem-se a um conjunto de "cartões" que podem parecer desnecessários, mas que, na maior parte dos casos, se mostram muito eficientes:

Problema	**Solução**
Dificuldade de retomar o workshop após os intervalos.	Qualquer pessoa atrasada leva um cartão "Atrasado". Use um timer de cozinha para chamar a atenção das pessoas, utilize uma caixa semelhante às utilizadas nas arrecadações de caridade (estabeleça R$1,00 para cada cartão usado).
Críticas severas - preconceitos mesquinhos, disputas de território, politicagens e comentários vulgares.	Cartão "1 Comentário Vulgar Gratuito", cartão "É uma Excelente Idéia!!" .
Exibicionismos, pontos de vista autoritários, informações irregulares dos participantes.	Use um facilitador treinado, limite o tempo do discurso a uma "Expressão de Opinião de Cinco Minutos".
Baixa de rendimento após o	Sirva almoços, lanches, café, refrigerantes,

almoço.	balas e biscoitos leves, reorganize a sala, altere a temperatura.

1.1.2.3- Brainstorm ou Tempestade Cerebral

Freqüentemente a tempestade cerebral permite um desbloqueio, um aquecimento para a conversação entre o analista e o cliente. Tem como propostas o desenvolvimento da originalidade e da desinibição, bem como a produção de um grande número de idéias em prazo curto; numa palavra, à criatividade.

Seu funcionamento geral é o seguinte: dado um tema, o cliente(s) expressa oralmente, em uma palavra ou em frases bem curtas, tudo o que lhe vem à cabeça, sugerido por aquele tema, sem se preocupar em censurar essas idéias. O analista vai anotando tudo e a seguir faz a seleção das idéias, segundo algum critério prévio, seja agrupando-as por alguma semelhança, seja eliminando as que não podem ser postas em prática, etc.

1.1.2.4- Teatralização

Filmes, desenhos animados, recursos animados, todos começam com encenações que informam quem são os atores, o que acontece a eles e como acontece.

- Ajuda a reunir e restringir os requisitos dos clientes de uma maneira amigável para os usuários.
- Estimula soluções de design mais criativas e inovadoras.
- Estimulam a revisão em equipe e evita os recursos indesejáveis.
- Assegura que os recursos sejam implementados de maneira intuitiva e acessível.
- Facilita o processo de entrevista - evitando a síndrome da página em branco.

De uma maneira simples, a encenação implica usar uma ferramenta para ilustrar (e algumas vezes animar) para os usuários (atores) como o sistema se ajustará à organização e também indicar como ele se comportará. Um facilitador mostra uma encenação inicial para o grupo e este último faz comentários. A encenação se desenrola então em "tempo real" durante o workshop. Sendo assim, será necessária uma ferramenta de desenho gráfico que permita alterar facilmente a encenação. Para evitar distrações, geralmente é mais inteligente usar ferramentas simples como, por exemplo, flip charts, um quadro branco ou o PowerPoint.

Há dois grupos distintos de ferramentas a serem usados na encenação: ferramentas passivas e ferramentas ativas. As ferramentas passivas indicam que serão mostradas figuras não animadas, enquanto as ferramentas passivas têm recursos internos mais sofisticados.

São exemplos de ferramentas passivas de encenação:

- Papel e lápis;
- Blocos de anotações autocolantes;
- Construtores GUI;
- Diferentes tipos de gerenciadores de apresentação.

São exemplos de ferramentas ativas de encenação:

- HyperCard, SuperCard ;
- Demo-It(tm) II da Bricklin ;
- Macromedia Director e outras ferramentas de animação;
- PowerPoint;

Avisos e comentários:

- É necessário que as encenações possam ser criadas e alteradas com facilidade. Se você não alterou nada é sinal de que não aprendeu nada.
- Não faça uma encenação muito boa. Não é nem um protótipo nem uma demonstração do que é real (percepção realista).

Interpretação de Papéis

Será atribuído a cada membro do grupo um papel de interesse para o sistema. Os papéis são os usuários, o sistema propriamente dito, outros sistemas e, algumas vezes, as entidades mantidas pelo sistema. O grupo inspecionará então como o sistema é usado. Ao longo do caminho, haverá discussões sobre quem é responsável por o que. Anote as responsabilidades de cada papel. Fazer com que o analista de sistemas interprete o papel do usuário ou do cliente ajuda a obter um discernimento real do domínio do problema.

Como um framework da interpretação de papéis, você poderá efetuar inspeções técnicas, seguindo uma espécie de script, para detectar como o sistema é usado. Se tiver alguns casos de uso descritos, você poderá usá-los como uma base para o script. A inspeção técnica também deve ser efetuada no nível do negócio, usando os casos de uso de negócios como uma base para o script.

Outra técnica freqüentemente combinada à interpretação de papéis é o uso de cartões de Colaboração de Responsabilidades da Classe (CRC).

Revisar Requisitos Existentes

Você poderá consultar especificações de requisitos de sistemas anteriores ou de outros sistemas relacionados - o que poderá ser útil. Com o grupo, inspecione cada requisito para detectar comportamentos de aplicativo ou atributos comportamentais. Em geral, durante a inspeção técnica, você deverá ignorar informações explicativas como introduções e descrições gerais do sistema.

Mantenha uma lista de todos os problemas identificados e certifique-se de encarregar alguém de resolver cada problema. Talvez você tenha que fazer algumas suposições

se um requisito não estiver claro. Anote essas suposições para que possa verificá-las com os envolvidos.

Lembre-se de quem escreveu os requisitos. Procure possíveis "requisitos que não se aplicam", no sentido de estarem fora do escopo do projeto. Se você não souber se algum item é um requisito, pergunte aos envolvidos.

É muito eficiente realizar esse tipo de inspeção técnica usando quaisquer esquemas de caso de uso existentes como um framework. É necessário que cada requisito esteja relacionado à pelo menos uma funcionalidade do sistema. Se não houver nenhuma funcionalidade com a qual se possa estabelecer uma relação, é sinal de que falta uma funcionalidade ou de que o requisito não se aplica.

2. O Componente Humano (Participantes do Processo)

O desenvolvimento de software é uma tarefa altamente cooperativa. Tecnologias complexas demandam especialistas em áreas específicas. Uma equipe de desenvolvimento de sistemas de software pode envolver vários especialistas, como por exemplo, profissionais de informática para fornecer o conhecimento técnico necessário ao desenvolvimento do sistema de software e especialistas do domínio para o qual o sistema de software deve ser desenvolvido.

Uma equipe de desenvolvimento de software típica consiste de um gerente, analistas, projetistas, programadores, clientes e grupos de avaliação de qualidade. Esses participantes do processo de desenvolvimento são descritos a seguir. Contudo, é importante notar que: a descrição dos participantes do processo tem mais um fim didático. Na prática, a mesma pessoa desempenha diferentes funções e, por outro lado, uma mesma função é normalmente desempenhada por várias pessoas.

Em virtude de seu tamanho e complexidade, o desenvolvimento de sistemas de software é um empreendimento realizado em equipe.

2.1 Gerentes de projeto

Como o próprio nome diz, o gerente de projetos é o profissional responsável pela gerência ou coordenação das atividades necessárias à construção do sistema. Esse profissional também é responsável por fazer o orçamento do projeto de desenvolvimento, como por exemplo, estimar o tempo necessário para o desenvolvimento do sistema, definir qual o processo de desenvolvimento, o cronograma de execução das atividades, a mão-de-obra especializada, os recursos de hardware e software etc.

O acompanhamento das atividades realizadas durante o desenvolvimento do sistema também é tarefa do gerente do projeto. É sua função verificar se os diversos recursos alocados estão sendo gastos na taxa esperada e, caso contrário, tomar providências para adequação dos gastos.

Questões tais como identificar se o sistema é factível, escalonar a equipe de desenvolvimento e definir qual o processo de desenvolvimento a ser utilizado também devem ser cuidadosamente estudadas pelo gerente do projeto.

2.2 Analistas

O analista de sistemas é o profissional que deve ter conhecimento do *domínio* do negócio. Esse profissional deve entender os problemas do domínio do negócio

ANÁLISE E GERÊNCIA DE REQUISITOS

Para que possa definir os requisitos do sistema a ser desenvolvido. Analistas devem estar aptos a se comunicar com *especialistas* do domínio para obter conhecimento acerca dos problemas e das necessidades envolvidas da organização empresarial. O analista não precisa ser um especialista do domínio. Contudo, ele deve ter suficiente domínio do vocabulário da área de conhecimento na qual o sistema será implantado para que, ao se comunicar com o especialista de domínio, este não precise ser interrompido a todo o momento para explicar conceitos básicos da área.

Tipicamente, o analista de sistemas é o profissional responsável por entender as necessidades dos clientes em relação ao sistema a ser desenvolvido e repassar esse entendimento aos demais desenvolvedores do sistema. Neste sentido, o analista de sistemas representa uma ponte de comunicação entre duas "facções": a dos profissionais de computação e a dos profissionais do negócio.

Para realizar suas funções, o analista de sistemas deve entender não só do domínio do negócio da organização, mas também ter conhecimento dos aspectos de alto nível de computação. Neste sentido, o analista de sistemas funciona como um tradutor, que mapeia informações entre duas "linguagens" diferentes: a dos especialistas de domínio e a dos profissionais técnicos da equipe de desenvolvimento.

Com a experiência adquirida através da participação no desenvolvimento de diversos projetos, alguns analistas se tomam gerentes de projetos. Na verdade, as possibilidades de evolução na carreira de um analista são bastante grandes. Isso se deve ao fato de que, durante a fase de levantamento de requisitos de um sistema, o analista se toma quase um especialista no domínio do negócio da organização. Para essa organização, é bastante interessante ter em seu quadro um profissional que entenda ao mesmo tempo de técnicas de desenvolvimento de sistemas e do processo de negócio da empresa. Por essa razão, não é rara a situação em que uma organização oferece um contrato de trabalho ao analista de sistemas ao final do desenvolvimento do sistema.

Uma característica importante que um analista de sistemas deve ter é a capacidade de comunicação, tanto escrita quanto falada, pois ele é um agente facilitador da comunicação entre os clientes e a equipe técnica. Muitas vezes, as capacidades de se comunicar agilmente e de ter um bom relacionamento interpessoal são mais importantes para o analista do que o conhecimento tecnológico.

Uma outra característica necessária a um analista é a ética profissional. Muitas vezes, o analista de sistemas está em contato com informações sigilosas e estratégicas dentro da organização na qual está trabalhando. Os analistas de sistemas têm acesso a informações como preços de custo de produtos, margens de lucro aplicadas, algo ritmos proprietários etc. Certamente, pode ser desastroso para a organização se informações de caráter confidencial como essas caírem em mãos erradas. Portanto, a ética profissional do analista de sistemas na manipulação de informações como essas é fundamental.

2.3 Projetistas

O projetista de sistemas é o componente da equipe de desenvolvimento cujas funções são (1) avaliar as alternativas de solução (da definição) do problema resultante da análise e (2) gerar a especificação de uma solução computacional detalhada. A tarefa do projetista de sistemas é muitas vezes chamada de *projeto físico.*

Na prática, existem diversos tipos de projetistas. Pode-se falar em projetistas de interface (especializados nos padrões de uma interface gráfica, como o *Windows* ou o *MacOS),* projetista de redes (especializados no projeto de redes de comunicação), projetista de bancos de dados (especializados no projeto de bancos de dados) e assim por diante. O ponto comum a todos esses tipos é que eles trabalham nos modelos resultantes da análise para adicionar os aspectos tecnológicos a tais modelos.

2.4 Arquitetos de software

Um profissional encontrado principalmente em grandes equipes reunidas para desenvolver sistemas complexos é o arquiteto de software. O objetivo desse profissional é elaborar a arquitetura do sistema como um todo. É ele quem toma decisões sobre quais são os subsistemas que compõem o sistema como um todo e quais são as interfaces entre esses subsistemas.

Além de tomar decisões globais, o arquiteto também deve ser capaz de tomar decisões técnicas detalhadas (por exemplo, decisões que têm influência sobre o desempenho do sistema).

Esse profissional também trabalha em conjunto com o gerente de projeto para priorizar e organizar o plano de projeto.

2.5 Programadores

Este profissional é o responsável pela *implementação* do *sistema.* Normalmente há vários programadores em uma equipe de desenvolvimento. Um programador pode ser proficiente em uma ou mais linguagens de programação, além de ter conhecimento sobre bancos de dados e poder ler os modelos resultantes do trabalho do projetista.

Na verdade, a maioria das equipes de desenvolvimento possui analistas que realizam alguma programação, e programadores que realizam alguma análise. No entanto, para fins didáticos, pode-se enumerar algumas diferenças entre as atividades desempenhadas por esses profissionais. Em primeiro lugar, o analista de sistemas está envolvido em todas as etapas do desenvolvimento, diferentemente do programador, que participa unicamente das fases finais (implementação e testes). Outra diferença é que analistas de sistemas devem entender tanto de tecnologia de

informação quanto do processo de negócio; programadores tendem a se preocupar somente com os aspectos tecnológicos do desenvolvimento.

Muitas vezes, bons programadores são "promovidos" a analistas de sistemas. Essa é uma prática comum nas empresas de desenvolvimento de software e é baseada na falsa lógica de que bons programadores serão bons analistas de sistemas. A verdade é que não se pode ter certeza de que um bom programador será um bom analista. Além disso, um programador não tão talentoso pode se tornar um ótimo analista. De fato, uma crescente percentagem de analistas sendo formados tem uma formação anterior diferente de computação. Por outro lado, um analista de sistemas deve ter algum conhecimento de programação para que produza especificações técnicas de um processo de negócio para serem passadas a um programador.

2.6 Clientes

Um outro componente da equipe de desenvolvimento é o *cliente.* O cliente é o indivíduo, ou grupo de indivíduos, para o qual o sistema é construído. Podem-se distinguir dois tipos de clientes: o *cliente usuário* e o *cliente contratante.* O cliente usuário é o indivíduo que efetivamente utilizará o sistema. O cliente usuário normalmente é um especialista no domínio do negócio. É com esse tipo de cliente que o analista de sistemas interage para levantar os requisitos do sistema. O cliente contratante é o indivíduo que solicita o desenvolvimento do sistema. Ou seja, é esse usuário quem encomenda e patrocina os custos de desenvolvimento e manutenção.

Em pequenas organizações, o cliente usuário e o cliente proprietário são a mesma pessoa. Já em grandes organizações, o cliente proprietário faz parte da gerência e é responsável por tomadas de decisões estratégicas dentro da empresa, enquanto que o cliente usuário é o responsável pela realização de processos operacionais.

Há casos em que o produto de software não é encomendado por um cliente. Em vez disso, o produto é desenvolvido para posteriormente ser comercializado. Esses produtos de software são normalmente direcionados para o mercado de massa. Exemplos de produtos de software nessa categoria são: processadores de texto, editores gráficos, jogos eletrônicos etc. Nesses casos, a equipe de desenvolvimento trabalha com o pessoal de marketing como se estes fossem os clientes reais.

Em qualquer caso, a participação do cliente usuário no processo de desenvolvimento é de suma importância. O distanciamento de usuários do desenvolvimento do sistema se manifesta, sobretudo, em projetos que excedem o seu orçamento, estão atrasados ou que não correspondam às reais necessidades. Mesmo que o analista entenda exatamente o que o usuário necessitava inicialmente, se o sistema construído não contemplar as necessidades atuais desse cliente, ainda será um sistema inútil. A única maneira de se ter um usuário satisfeito com o sistema de informações é torná-lo um legítimo participante no desenvolvimento do sistema.

ANÁLISE E GERÊNCIA DE REQUISITOS

Não importa qual seja o processo de desenvolvimento utilizado; o envolvimento do usuário final no desenvolvimento de um sistema de software é de fundamental importância.

3. O Processo de Requisitos de Software

3.1-Requisitos de software

Objetivos
O objetivo deste capítulo é apresentar uma introdução sobre os requisitos de sistema de software e explicar diferentes maneiras de expressar esses requisitos. Depois de ler este capítulo, você poderá:

- compreender os conceitos dos requisitos do usuário e dos requisitos de sistema e por que esses requisitos podem ser expressos utilizando-se diferentes notações;
- compreender as diferenças entre os requisitos funcionais e não funcionais;
- compreender duas técnicas para descrever os requisitos de sistema, como a descrição em linguagem natural estruturada e a descrição com base em linguagem de programação, e
- compreender como os requisitos podem ser organizados em um documento de requisitos de software.

Os problemas que os engenheiros de software têm para solucionar são, muitas vezes, imensamente complexos. Compreender a natureza dos problemas pode ser muito difícil, especialmente se o sistema for novo. Conseqüentemente, é difícil estabelecer com exatidão o que o sistema deve fazer. As descrições das funções e das restrições são os requisitos para o sistema; e o processo de descobrir, analisar, documentar e verificar essas funções e restrições é chamado de engenharia de requisitos.

O termo *requisito* não é utilizado pela indústria de software de modo consistente. Em alguns casos, um requisito é visto como uma declaração abstrata, de alto nível, de uma função que o sistema deve fornecer ou de uma restrição do sistema. No outro extremo, ele é uma definição detalhada, matematicamente formal, de uma função do sistema. Davis (1993) explica por que essas diferenças existem:

Se uma empresa deseja estabelecer um contrato para o desenvolvimento de um grande projeto de software, ela tem de definir suas necessidades de maneira suficientemente abstrata para que uma solução não seja predefinida. Os requisitos devem ser redigidos de modo que os diversos fornecedores possam apresentar propostas, oferecendo, talvez, diferentes maneiras de atender às necessidades organizacionais do cliente. Uma vez estabelecido um contrato, o fornecedor precisa preparar uma definição de sistema para o cliente, com mais detalhes, de modo que o cliente compreenda e possa validar o que o software fará. Esses dois documentos podem ser chamados de *documentos de requisitos* do sistema.

ANÁLISE E GERÊNCIA DE REQUISITOS

Alguns dos problemas que surgem durante o processo de engenharia de requisitos são resultantes da falta de uma nítida separação entre esses diferentes níveis descrição. Faço essa distinção utilizando o termo requisitos do usuário para designar os requisitos abstratos de alto nível, e o termo requisitos de sistema para indicar a descrição detalhada do que o sistema deverá fazer. Além desses dois níveis de detalhes, uma descrição mais detalhada (uma especificação de projeto de software) pode ser produzida para associar a engenharia de requisitos e as atividades de projeto. Os requisitos do usuário, os requisitos de sistema e a especificação de projeto de software podem ser definidos como se segue:

1. *Requisitos do usuário* são declarações, em linguagem natural e também em diagramas, sobre as funções que o sistema deve fornecer e as restrições sob as quais deve operar.

2. *Requisitos de sistema* estabelecem detalhadamente as funções e as restrições de sistema. O documento de requisitos de sistema, algumas vezes chamado de especificação funcional, deve ser preciso. Ele pode servir como um contrato entre o comprador do sistema e o desenvolvedor do software.

3. *Especificação de projeto de software* é uma descrição abstrata do projeto de software; que é uma base para o projeto e a implementação mais detalhados.

Diferentes níveis de especificação de sistema são úteis porque comunicam informações sobre o sistema para diferentes tipos de leitores. A figura 3.1 ilustra a distinção entre os requisitos de usuários e os de sistemas. Ela mostra como um requisito de usuário pode ser expandido em diversos requisitos de sistema.

Os requisitos do usuário devem ser escritos para gerentes do cliente e dos fornecedores, que não tenham um documento técnico detalhado do sistema (Figura 3.2). A especificação de requisitos de sistema dever ter como alvo os profissionais técnicos de nível sênior e os gerentes de projeto. Novamente, ela será utilizada pelo gerente do cliente e do fornecedor. Os usuários finais de sistemas podem ler ambos os documentos. Por fim, a especificação de projeto de software é um documento orientado à implementação. Ele deve ser escrito para os engenheiros de software que desenvolverão o sistema.

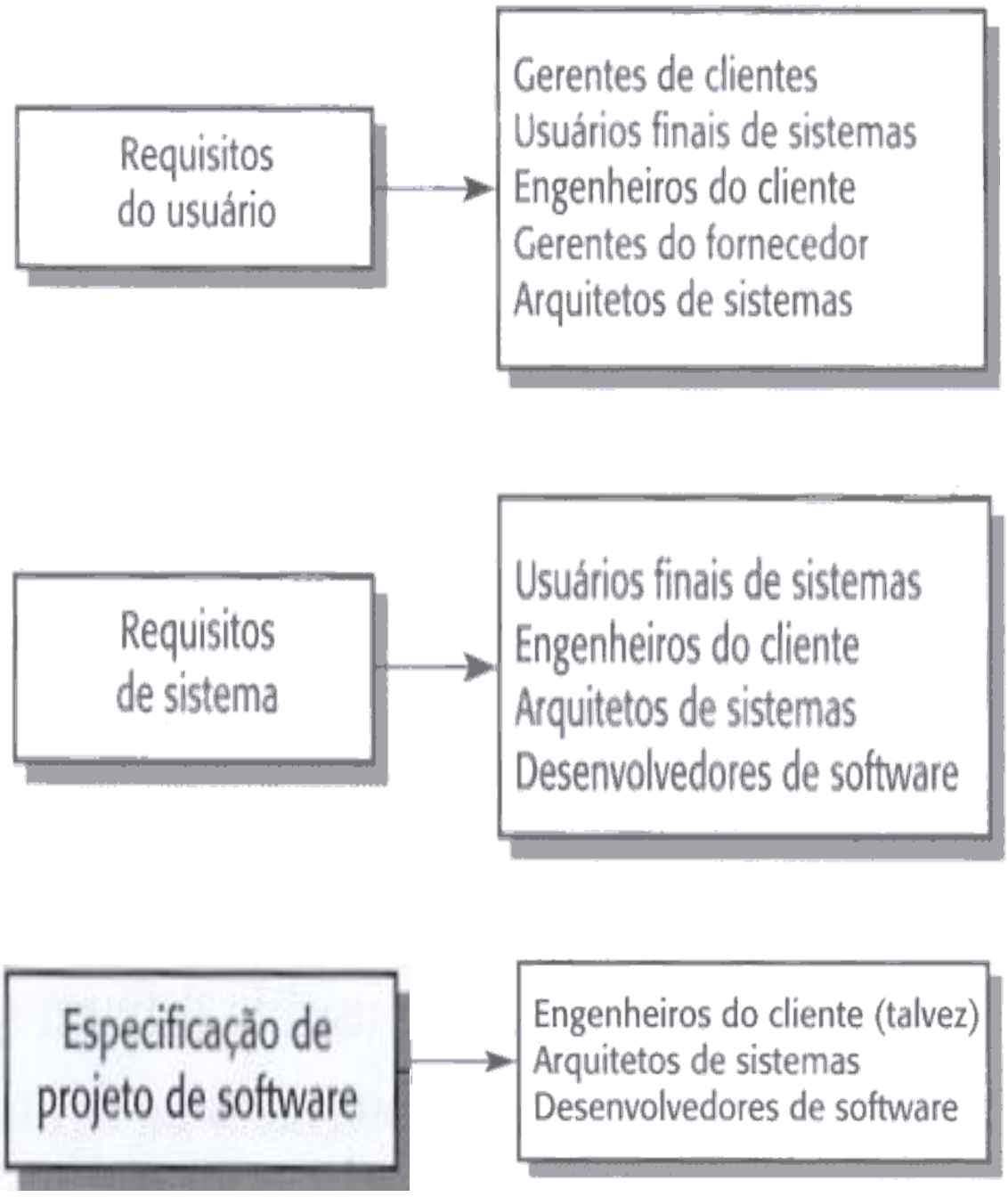
Figura 3.1: Requisitos do usuário e do sistema
Definição dos requisitos do usuário

1. O software deve oferecer um meio de representar e acessar arquivos externos criados por outras ferramentas.

Especificação dos requisitos de sistema

1.1 O usuário deve dispor de recursos para definir o tipo dos arquivos externos.
1.2 Cada tipo de arquivo externo pode ter uma ferramenta associada que pode ser aplicada a ele.
1.3 Cada tipo de arquivo externo pode ser representado como um ícone específico na tela do usuário.
1.4 Devem ser fornecidos recursos para o ícone que representa um arquivo externo, a ser definido pelo usuário.
1.5 Quando um usuário seleciona um ícone que representa um arquivo externo, o efeito dessa seleção é aplicar a ferramenta associada com o tipo de arquivo externo ao arquivo representado pelo ícone selecionado

Figura 3.2 Leitores de Diferentes tipos de especificação



ANÁLISE E GERÊNCIA DE REQUISITOS

3.1.1- Requisitos Funcionais e Não-Funcionais

Os requisitos de sistema de software são, freqüentemente, classificados como funcionais ou não-funcionais ou como requisitos de domínio:

1- **Requisitos Funcionais**. São declarações de funções que o sistema deve fornecer, como o sistema deve reagir a entradas específicas e como deve se comportar em determinadas situações. Em alguns casos, os requisitos funcionais podem também explicitamente declarar o que o sistema não deve fazer.

2- **Requisitos Não-funcionais**. São restrições sobre os serviços ou as funções oferecidas pelo sistema. Entre eles destacam-se restrições de tempo, restrições sobre o processo de desenvolvimento, padrões, entre outros.

3- **Requisitos de Domínio**. São requisitos que se originam do domínio de aplicação do sistema e que refletem características desse domínio. Podem ser requisitos funcionais ou não funcionais.

Na realidade, a distinção entre esses diferentes tipos de requisitos não é tão clara como sugerem definições simples. Um requisito de usuário relacionado à proteção, digamos, parece ser um requisito não funcional. Contudo, quando desenvolvido com mais detalhes, pode levar aos outros requisitos que são claramente funcionais, como a necessidade de incluir recursos de autorização de usuários no sistema. Portanto, embora seja útil classificar os requisitos dessa maneira quando os discutimos, devemos lembrar que essa é, na verdade, uma distinção artificial.

3.1.1.1-Requisitos funcionais

Os requisitos funcionais para um sistema descrevem a funcionalidade ou os serviços que se espera que o sistema forneça. Eles dependem do tipo de software que está sendo desenvolvido, dos usuários de software que se espera verificar e do tipo de sistema que está sendo desenvolvido. Quando expressos como requisitos de usuário, eles são normalmente descritos de um modo bastante geral, mas os requisitos funcionais de sistema descrevem a função de sistema detalhadamente, suas entradas e saídas, exceções, etc.

ANÁLISE E GERÊNCIA DE REQUISITOS

Os requisitos funcionais de um sistema de software podem ser expressos de diversas maneiras. Aqui estão vários requisitos funcionais do sistema de biblioteca de universidade (Kotonya e Sommerville, 1998) para que os estudantes e a faculdade possam pedir livros e documentos de outras bibliotecas:

1. O usuário deverá ser capaz de buscar todo o conjunto inicial de banco de dados ou selecionar um subconjunto a partir dele.

2. O sistema fornecerá telas apropriadas para o usuário ler documentos no repositório de documentos.

3. Cada pedido será alocado a um único identificador (ORDER_ID), que o usuário poderá copiar para a área de armazenagem permanente da conta.

Esses requisitos funcionais de usuário definem recursos específicos que devem ser fornecidos pelo sistema. Eles foram observados no documento de requisitos de usuário do sistema e ilustram que os requisitos funcionais podem ser escritos em diferentes níveis de detalhes (compare os requisitos 1 e 3).

Muitos problemas de engenharia de software se originam da imprecisão na especificação de requisitos. É natural para um desenvolvedor de sistemas interpretar um requisito ambíguo para simplificar sua implementação. Muitas vezes, contudo, isso não é o que o cliente quer. Novos requisitos devem ser estabelecidos e é necessário realizar mudanças no sistema. Naturalmente, isso atrasa a entrega do sistema e aumenta os custos.

Considere o requisito do segundo exemplo para o sistema de biblioteca, na lista de requisitos mostrada anteriormente, que se refere a 'telas apropriadas' fornecidas pelo sistema. O sistema de biblioteca pode fornecer documentos em uma série de formatos, e a intenção desse requisito é que as telas para todos esses formatos estejam disponíveis. Contudo, ele está redigido de maneira ambígua, uma vez que não toma claro que as telas para cada formato de documento devem ser disponibilizadas. Um desenvolvedor que está sob pressão quanto à programação pode simplesmente fornecer uma tela de texto e argumentar que os requisitos foram cumpridos.

Em princípio, a especificação de requisitos funcionais de um sistema deve ser completa e consistente. A completeza significa que todas as funções requeridas pelo usuário devem estar definidas. A consistência significa que os requisitos não devem ter definições contraditórias. Na prática, para sistemas complexos e grandes, é quase impossível atingir a consistência e a completeza dos requisitos. A razão disso é, em parte, por causa da complexidade inerente ao sistema e, em parte, porque diferentes pontos de vista apresentam necessidades inconsistentes.

Essas inconsistências podem não ser óbvias quando os requisitos são especificados primeiramente. Os problemas somente emergem depois de uma análise mais profunda. À medida que os problemas são descobertos durante as revisões ou em fases posteriores do ciclo de vida, os problemas no documento de requisitos devem ser corrigidos.

3.1.2 Requisitos Não Funcionais

Os requisitos não funcionais, como o nome sugere, são aqueles que não dizem respeito diretamente às funções específicas fornecidas pelo sistema. Eles podem estar relacionados a propriedades de sistema emergentes, como confiabilidade, tempo de resposta e espaço em disco.

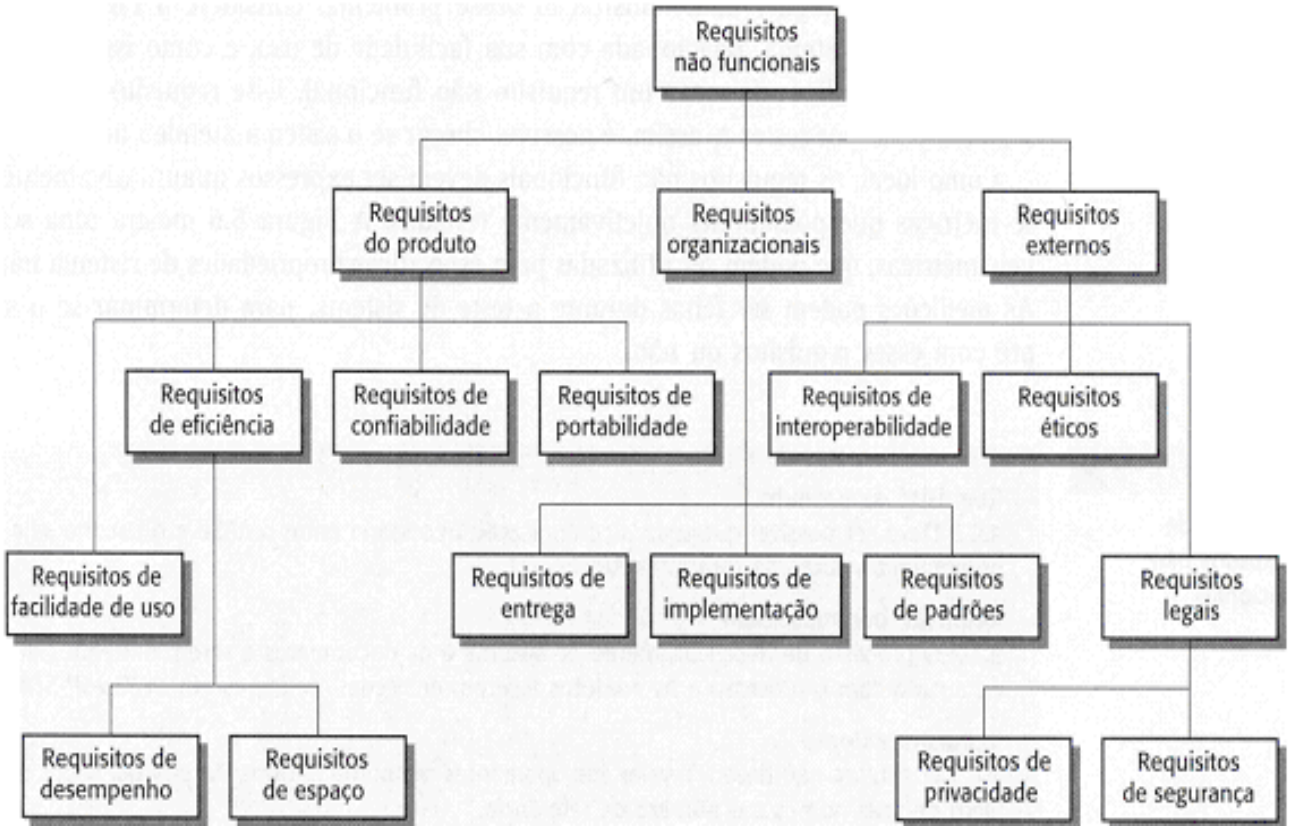
Como alternativa, eles podem definir restrições para o sistema, como a capacidade dos dispositivos de E/S (entrada/saída) e as representações de dados utilizadas nas interfaces de sistema.

Muitos requisitos não funcionais dizem respeito ao sistema como um todo, e não a características individuais do sistema. Isso significa que eles são, freqüentemente, mais importantes do que os requisitos funcionais individuais. Enquanto a falha em cumprir com um requisito funcional individual pode degradar o sistema, a falha em cumprir um requisito não funcional de sistema pode tomar todo o sistema inútil. Por exemplo, se um sistema de aviação não atender a seus requisitos de confiabilidade, ele não será atestado como seguro para operação; se um sistema de controle em tempo real falhar em cumprir com seus requisitos de desempenho, as funções de controle não operarão corretamente.

Contudo, os requisitos não funcionais nem sempre dizem respeito ao sistema de software a ser desenvolvido. Alguns requisitos não funcionais podem restringir o processo que pode ser utilizado para desenvolver o sistema. São exemplos de requisitos de processo uma especificação dos padrões de qualidade, que deve ser utilizada no processo, uma especificação de que o projeto deve ser produzido com um conjunto especificado de ferramentas CASE e uma descrição de processo a ser seguido.

Os requisitos não funcionais surgem conforme a necessidade dos usuários, em razão de restrições de orçamento, de políticas organizacionais, pela necessidade de interoperabilidade com outros sistemas de software ou hardware ou devido a fatores externos, como por exemplo regulamentos de segurança e legislação sobre privacidade. A Figura 3.3 é uma classificação dos diferentes tipos de requisitos não funcionais que podem surgir.

Figura 3.3 Tipos de Requisitos Não Funcionais



Classificação dos diferentes tipos de requisitos não funcionais, mostrados na Figura 3.3, de acordo com sua procedência:

1. ***Requisitos de produtos*** São os requisitos que especificam o comportamento do produto. Entre os exemplos estão os requisitos de desempenho sobre com que rapidez o sistema deve operar e quanta memória ele requer, os requisitos de confiabilidade, que estabelecem a taxa aceitável de falhas, os requisitos de portabilidade e os requisitos de facilidade de uso.

2. ***Requisitos organizacionais*** São procedentes de políticas e procedimentos nas organizações do cliente e do desenvolvedor. Entre os exemplos estão os padrões de processo que devem ser utilizados, os requisitos de implementação, como a linguagem de programação ou o método de projeto utilizado, e os requisitos de fornecimento, que especificam quando o produto e seus documentos devem ser entregues.

3. ***Requisitos externos*** Esse amplo tópico abrange todos os requisitos procedentes de fatores externos ao sistema e a seu processo de desenvolvimento. Dentre eles destacam-se os requisitos de interoperabilidade, que definem como o sistema interage com sistemas em outras organizações, os requisitos legais, que devem ser seguidos para assegurar que o sistema opera de acordo com a lei, e os requisitos éticos. Os requisitos éticos são definidos em um sistema para garantir que este será aceitável para seus usuários e o público em geral.

A Figura 3.4 mostra exemplos de requisitos de produto, organizacionais e externos. O requisito de produto diz respeito a um ambiente de apoio à programação para a linguagem Ada (um APSE). Ele restringe a liberdade dos projetistas do APSE na sua escolha de símbolos utilizados na interface com o usuário, do APSE; não diz nada sobre a funcionalidade do APSE, e claramente identifica uma restrição do sistema, em vez de uma função. O requisito organizacional especifica que o sistema deve ser desenvolvido de acordo com um processo-padrão da empresa, definido como XYZCo-SP-STAN-95. O requisito externo é derivado da necessidade de o sistema atender à legislação de privacidade. Ele especifica que os operadores de sistema não devem ter acesso a qualquer dado de que não necessitem.

Um problema comum com os requisitos não funcionais é que eles são, às vezes, de difícil verificação; eles podem ser escritos para refletir os objetivos gerais do cliente, como a facilidade de uso, a habilidade de o sistema se recuperar de uma falha ou a rapidez de resposta ao usuário. Esses requisitos causam problemas para os desenvolvedores de sistema, à medida que eles deixam o enfoque aberto à interpretação e à conseqüente discussão, quando o sistema é entregue. Como ilustração desse problema, considere a Figura 3.5. Ela mostra uma meta do sistema, relacionada com sua facilidade de uso, e como isso pode ser expresso de maneira verificável, como um requisito não funcional. Esse requisito não funcional pode ser verificado por testes e, assim, é possível checar se o sistema atendeu ao objetivo do cliente.

Como ideal, os requisitos não funcionais devem ser expressos quantitativamente, utilizando-se métricas que possam ser objetivamente testadas. A Figura 3.6 mostra uma série de possíveis métricas, que podem ser utilizadas para especificar propriedades de sistema não funcionais. As medições podem ser feitas durante o teste de sistema, para determinar se o sistema cumpre com esses requisitos ou não.

Figura 3.4 Exemplos de Requisitos não funcionais

Requisito de produto
4.C.8 Deve ser possível que toda a comunicação necessária ente o APSE e o usuário seja expressa no conjunto-padrão de caracteres Ada.
Requisito Organizacional
9.3.2 O processo de desenvolvimento de sistema e os documentos a serem entregues deverão estar de acordo com o processo e os produtos a serem entregues, definidos em XYZCo-SP-STAN-95.
Requisito Externo
7.6.5 O sistema não deverá revelar aos operadores nenhuma informação pessoal sobre os clientes, além de seus nomes e o número de referência.

Figura 3.5 Metas do Sistema e Requisitos Verificáveis

Uma meta de sistema
O sistema deve ser fácil de utilizar por controladores e deve ser organizado de modo que os erros dos usuários sejam minimizados.

Um requisito não funcional verificável

Controladores experientes devem ser capazes de utilizar todas as funções do sistema depois de um total de duas horas de treinamento. Depois desse treinamento, o número médio de erros feitos por usuários experientes não deve exceder a dois por dia.

Figura 3.6 Métricas para especificar requisitos não funcionais

Propriedade	**Métrica**
Velocidade	Transações processadas/segundo Tempo de resposta ao usuário/evento Tempo de refresh da tela
Tamanho	K bytes Número de chips de RAM
Facilidade de Uso	Tempo de Treinamento Número de frames de ajuda
Confiabilidade	Tempo médio para falhar Probabilidade de indisponibilidade Taxa de ocorrência de falhas Disponibilidade
Robustez	Tempo de reinício depois de uma falha Porcentagem de eventos que causam falhas Probabilidade de que dados sejam corrompidos por falhas
Portabilidade	Porcentagem de declarações dependentes de sistemas-alvo Número de sistemas-alvo

Na prática, em geral, a especificação quantitativa de requisitos é difícil. Os clientes podem não ser capazes de traduzir suas metas em requisitos quantitativos; para algumas metas, como as metas de facilidade de manutenção, não há métricas que possam ser utilizadas; o custo de verificar objetivamente requisitos quantitativos não funcionais pode ser muito alto. Portanto, os documentos de requisitos, muitas vezes, incluem declarações de metas misturadas com requisitos. Essas metas podem ser úteis para os desenvolvedores porque fornecem algumas pistas sobre as prioridades do cliente. Contudo, os clientes devem saber que essas metas estão sujeitas às más interpretações e não podem ser objetivamente verificadas.

Os requisitos não funcionais freqüentemente entram em conflito e interagem com outros requisitos funcionais do sistema. Por exemplo, pode ser requerido que o máximo de espaço ocupado por um sistema seja de 4 Mbytes, porque todo o sistema deve ser adequado à memória do tipo *read-only* (memória 'somente para leitura') instalado em uma espaçonave. Um requisito adicional pode ser que o sistema seja escrito utilizando-se Ada, uma linguagem de programação projetada para o desenvolvimento de software critico e em tempo real. Contudo, pode não ser possível compilar um programa Ada com a funcionalidade exigida, com menos de 4 Mbytes. É preciso encontrar certa 'compensação' entre esses requisitos. Uma linguagem alternativa de desenvolvimento pode ser utilizada, ou também é possível acrescentar memória ao sistema.

Em princípio, os requisitos funcionais e não funcionais devem ser diferenciados em um documento de requisitos; na prática, isso é difícil. Se os requisitos não funcionais forem definidos separadamente dos requisitos funcionais, algumas vezes será difícil ver o relacionamento entre eles. Se eles forem definidos com os requisitos funcionais, poderá ser difícil separar considerações funcionais e não funcionais e identificar os requisitos que correspondem ao sistema como um todo. É preciso encontrar um equilíbrio adequado e isso depende do tipo de sistema que está sendo especificado. Contudo, requisitos claramente relacionados às propriedades emergentes do sistema devem ser explicitamente destacados. Isso pode ser feito colocando-os em uma seção separada do documento de requisitos ou distinguindo-os, de alguma maneira, dos outros requisitos de sistema.

3.1.3 Requisitos de Domínio

Os requisitos de domínio são derivados do domínio da aplicação do sistema, em vez de serem obtidos a partir das necessidades específicas dos usuários do sistema. Eles podem ser novos requisitos funcionais em si, podem restringir os requisitos funcionais existentes ou estabelecer como devem ser realizados cálculos específicos. Os requisitos de domínio são importantes porque, muitas vezes, refletem fundamentos do domínio da aplicação. Se esses requisitos não forem satisfeitos, poderá ser impossível fazer o sistema operar satisfatoriamente.

Para ilustrar os requisitos de domínio, considere os seguintes requisitos do sistema de biblioteca: .

1. Deve haver uma interface-padrão com o usuário para todos os bancos de dados, que terá como base o padrão *Z39.50.* .
2. Em razão das restrições referentes a direitos autorais, alguns documentos devem ser excluídos imediatamente ao serem fornecidos. Dependendo dos requisitos dos usuários, esses documentos serão impressos localmente no servidor do sistema para serem encaminhados manualmente ao usuário ou direcionados para uma impressora de rede.

O primeiro desses requisitos é uma restrição sobre uma exigência funcional de sistema. Ele especifica que a interface com o usuário para o banco de dados deve ser implementada de acordo com um padrão específico da biblioteca. O segundo requisito foi introduzido devido às leis de direitos autorais, que se aplicam aos materiais utilizados em bibliotecas. Ele especifica que o sistema deve incluir um recurso automático de excluir impressão, para algumas classes de documentos.

Para ilustrar os requisitos de domínio que especificam como um cálculo é realizado, considere a Figura 3.7, que tem como base a especificação de um

sistema auto matizado de proteção de trens. Esse sistema parará automaticamente um trem se ele atravessar um sinal vermelho. Esse requisito define como a desaceleração do trem é calculada pelo sistema e utiliza terminologia específica do domínio. Para compreendê-lo, o engenheiro precisa ter algum conhecimento de operação de sistemas de estrada de ferro e das características dos trens.

Os requisitos para o sistema de trens ilustram o problema principal com os requisitos de domínio. Esses requisitos são expressos com o uso de uma linguagem que é específica do domínio da aplicação e, muitas vezes, é difícil para os engenheiros de software compreender essa linguagem. Os especialistas em um domínio podem deixar de fornecer informações em um requisito, simplesmente porque para eles essas informações são muito óbvias. Contudo, elas podem não ser óbvias para os desenvolvedores de sistemas, que, como resultado disso, talvez venham a implementar o requisito de maneira insatisfatória.

Figura 3.7 Um requisito de um domínio de um sistema de proteção de trens.

A desaceleração do trem será computada como: D trem=D controle+D gradiente Onde D gradiente é 9,8 ms2 * gradiente compensado/alfa e onde os valores de 9,81 ms2/alfa são conhecidos para diferentes tipos de trens.

3.2 Requisitos de Usuário

Os requisitos de usuários para um sistema devem descrever os requisitos funcionais e não funcionais de modo compreensível pelos usuários do sistema que não têm conhecimentos técnicos detalhados. Eles devem especificar somente o comportamento externo do sistema, evitando tanto quanto possível as características do projeto de sistema.

Conseqüentemente, não devem ser definidos utilizando-se um modelo de implementação. Eles podem ser escritos com o uso de linguagem natural, formulários e diagramas intuitivos simples.

Contudo, vários problemas podem surgir quando os requisitos são escritos em linguagem natural:

1. *Falta de clareza* Às vezes, é difícil utilizar a linguagem de maneira precisa e sem ambigüidade, sem produzir um documento de difícil leitura.
2. *Confissão de requisitos* Os requisitos funcionais e os não funcionais, os objetivos do sistema e as informações sobre o projeto podem não estar claramente definidos.
3. *Fusão de requisitos* Vários requisitos diferentes podem ser expressos juntos como um único requisito.

Como ilustração de alguns desses problemas, considere um dos requisitos para um ambiente de programação em linguagem Ada, mostrado na Figura 3.8.

Esse requisito inclui informações conceituais e detalhadas. Ele expressa o conceito de que deve haver recursos de controle de configuração fornecidos como uma parte inerente do APSE. Contudo, ele também inclui o detalhe de que os recursos de controle de configuração devem permitir o acesso aos objetos em um grupo de versão, sem especificar seu nome completo.
Teria sido melhor deixar esse detalhe para a especificação de requisitos de sistema. É uma boa prática separar os requisitos de usuário dos requisitos mais detalhados do sistema, em um documento de requisitos. Do contrário, leitores que não têm conhecimentos técnicos dos requisitos de usuário podem ser sobrecarregados com detalhes que, na verdade, somente são relevantes para os técnicos. A Figura 2.9 ilustra essa confusão. Esse exemplo tem como base o documento de requisitos para uma ferramenta CASE destinada a editar modelos de projeto de software.

O usuário pode especificar que uma grade deve ser exibida de modo que entidades possam ser precisamente posicionadas em um diagrama. A primeira sentença mistura três diferentes tipos de requisitos:

1. Um requisito funcional conceitual especifica que o sistema de edição deve fornecer uma grade e apresenta uma base lógica para isso.

Figura 3.8 Um requisito sobre um banco de dados para um ambiente de programação.

4.A.5. o banco de dados deve aceita a geração e o controle de objetos de configuração; ou seja, os objetos que são agrupamentos de outros objetos no banco de dados. Os recursos de controle de configuração devem permitir o acesso aos objetos em um grupo de versão, pelo uso de um nome incompleto.

Figura 3.9 Um requisito de usuário para uma grade de editor.

2.6. **Recursos de Grade** Para ajudar no posicionamento de entidades em um diagrama, o usuário pode acionar uma grade em centímetros ou em polegadas, por meio de uma opção no painel de controle. Inicialmente, a grade está desativada. Ela pode ser ligada e desligada a qualquer momento durante uma sessão de edição e pode ser alternada entre polegadas e centímetros a qualquer momento. Uma opção de grade será fornecida na visão reduzida do diagrama, mas o número de linhas da grade mostrado diminuirá, para evitar preencher o diagrama menor com linhas de grade.

2. Um requisito não funcional, que fornece informações detalhadas sobre as unidades da grade (centímetros ou polegadas).

3. Um requisito não funcional de interface com o usuário, que define como essa grade é ligada e desligada pelo usuário.

O requisito na Figura 3.9 também apresenta algumas informações, mas não todas as de inicialização. Ele especifica que a grade está inicialmente desligada, contudo não define suas unidades quando está ligada. Ele fornece informações detalhadas, por exemplo, que o usuário pode alternar entre unidades, mas não o espaçamento entre as linhas da grade.

Quando os requisitos do usuário apresentam muitas informações, isso restringe a liberdade do desenvolvedor do sistema para oferecer soluções inovadoras aos problemas do usuário e faz com que os requisitos sejam de difícil compreensão. Os requisitos do usuário devem simplesmente mostrar os recursos principais a serem fornecidos. Isso é ilustrado na Figura 3.10, em que reescrevi o requisito para a grade de editor, focalizando apenas as características essenciais do sistema.

A lógica associada com os requisitos é importante. Ela ajuda os desenvolvedores de sistema e os responsáveis pela sua manutenção a compreender por que os requisitos foram incluídos e a avaliar o impacto da mudança nos requisitos. Por exemplo, na Figura 3.10, a lógica reconhece que pode ser útil uma grade ativa, em que objetos posicionados automaticamente saltam para uma linha da grade. Contudo, ela rejeita deliberadamente essa opção em favor de um posicionamento manual. Se uma mudança nessa opção for proposta em algum estágio posterior, ficará claro que o uso de uma grade passiva foi deliberado, em vez de ser uma decisão de implementação.

Outro exemplo de um requisito de usuário mais específico, que também define parte do sistema de edição, é mostrado na Figura 3.11. Essa é uma especificação

mais detalhada de uma função. Nesse caso, a definição inclui uma lista de ações do usuário. Essa opção é, algumas vezes, necessária, de modo que todas as noções possam ser fornecidas de maneira consistente. Os detalhes de implementação não devem ser incluídos nessas informações adicionais.

Portanto, a definição não estabelece como o cursor e os símbolos são movimentados ou como o tipo é selecionado.

Para minimizar possíveis divergências quando estiver redigindo os requisitos do usuário, recomendo que o leitor siga algumas diretrizes simples:

1. Invente um formato-padrão e certifique-se de que todas as definições de requisitos estejam conforme esse formato. Padronizar o formato significa que as omissões podem ser menos freqüentes e faz com que os requisitos sejam verificados com mais facilidade. "O' formato que utilizo inclui 'reforçar' o requisito inicial, considerando uma declaração da lógica, com cada requisito de usuário e uma referência à especificação mais detalhada de requisitos de sistema."

2. Utilize a linguagem de modo consistente. Em particular, faça uma distinção entre os requisitos obrigatórios e os que são desejáveis. É prática usual definir os requisitos obrigatórios utilizando-se o verbo 'deve' e os requisitos desejáveis, utilizando-se o verbo 'deveria', como podemos ver na Figura 3.11. Portanto, é obrigatório que o sistema inclua um recurso para acrescentar nós em um desenho. É desejável que a seqüência de ações seja feita como foi especificado, mas isso não será absolutamente essencial, se houver boas razões para fazê-la de outra maneira.

3. Utilize destaque no texto (negrito ou itálico) para ressaltar partes importantes dos requisitos.

4. Evite, tanto quanto possível, o uso de jargões de informática. Contudo, inevitavelmente, ocorrerá que termos técnicos detalhados, utilizados no domínio da aplicação do sistema, sejam incluídos nos requisitos do usuário.

Figura 3.10 Uma definição de um recurso de grade de editor.

2.6 Recursos de Grade

2.6.1 o editor deverá fornece um recurso de grade, em que uma matriz de linhas horizontais e verticais constitua um fundo da janela do editor. Essa grade deverá ser uma tela passiva em que o alinhamento de entidades é de responsabilidade do usuário.

Lógica: uma grade ajuda o usuário a criar um diagrama `limpo`, com entidades bem espaçadas. Embora uma grade ativa, em que as entidades saltam as linhas de grade, possa ser útil, o posicionamento é impreciso. O usuário é a melhor pessoa para decidir onde as entidades devem ser posicionadas.

*Especificação:*ECLIPSE/WS/Ferramentas/DE/FS.

Figura 3.11 Requisitos de usuário para criação de nó.

3.5.1 Adicionando nó a um desenho

3.5.1.1 **O editor deve fornecer um recurso aos usuários para adicionar nós de um tipo especificado a seu desenho.**

3.5.1.2 A seqüência de ações para acrescentar um nó deve se como se segue:

1- o usuário deve selecionar o tipo de nó a se acrescentado.

2- o usuário deve mover o cursor para a posição aproximada do nó no diagrama e indicar que o símbolo do nó deve ser adicionado naquele ponto.

3- o usuário deve então arrastar o símbolo do nó para sua posição final. Lógica: o usuário é a melhor pessoa para decidir onde posicionar um nó no diagrama. Essa abordagem dá ao usuário o controle direto sobre a seleção do tipo de nó e seu posicionamento.

Especificação: ECLIPSE/WS/Ferramentas/DE/FS.

3.3 Requisitos de Sistema

Os requisitos de sistema são descrições mais detalhadas dos requisitos do usuário. Eles podem servir como base para um contrato destinado à implementação do sistema e, portanto, devem ser uma especificação completa e consistente de todo o sistema. Eles são utilizados pelos engenheiros de software como ponto de partida para o projeto de sistema.

ANÁLISE E GERÊNCIA DE REQUISITOS

A especificação de requisitos de sistema pode incluir diferentes modelos do sistema, como um modelo de objeto ou um modelo de fluxo de dados.
Em princípio, os requisitos de sistema deveriam definir o que o sistema deveria fazer, e não como ele teria de ser implementado. Contudo, no que se refere aos detalhes exigidos para especificar o sistema completamente, é quase impossível excluir todas as informações de projeto. Há várias razões para isso:

1. Uma arquitetura inicial do sistema pode ser definida para ajudar a estruturar a especificação de requisitos. Os requisitos de sistema são organizados de acordo com os diferentes subsistemas que constituem o sistema.

2. Na maioria dos casos, os sistemas devem interoperar com outros sistemas existentes. Isso restringe o projeto, e essas restrições geram requisitos para o novo sistema.

3. O uso de um projeto específico pode ser um requisito externo de sistema.

A linguagem natural é utilizada freqüentemente para escrever especificações de requisitos de sistema. Contudo, problemas com a linguagem natural podem surgir, quando ela é utilizada para uma especificação mais detalhada:

1- A compreensão da linguagem natural depende do uso das mesmas palavras para o mesmo conceito, pelos leitores e por quem escreve as especificações. Isso leva às divergências, devido à ambigüidade da linguagem natural. Jackson (1995) apresenta um excelente exemplo disso quando discute os sinais exibidos em uma escada rolante. Eles diziam "É preciso estar calçado" e "Os cachorros devem ser carregados". Deixo a seu encargo analisar a interpretação conflitante dessas frases.

2. Uma especificação de requisitos em linguagem natural é muito flexível. Pode-se dizer a mesma coisa de modos completamente diferentes. Fica por conta do leitor descobrir quando os requisitos são os mesmos e quando são diferentes.

3- Não existe nenhum meio fácil de padronizar os requisitos de linguagem natural. Pode ser difícil encontrar todos os requisitos relacionados. Para descobrir as conseqüências de uma mudança, talvez seja preciso examinar cada requisito, em vez de simplesmente um grupo de requisitos relacionados.

Devido a esses problemas, as especificações de requisitos escritas em linguagem natural estão sujeitas às divergências. Muitas vezes, elas somente são descobertas em fases posteriores do processo de software e, então, a solução para elas pode ser muito dispendiosa.

Existem várias alternativas para o uso da linguagem natural, que adicionam estrutura à especificação e que ajudam a reduzir a ambigüidade. Elas são mostradas na Figura 3.12.
Outras abordagens, como linguagens especializadas de requisitos (Teichrow e Hershey, 1977; Alford, 1977; Bell *et ai.,* 1977; Alford, 1985), também foram desenvolvidas, mas atualmente elas são pouco utilizadas. Davis (1990) resume e compara algumas dessas diferentes abordagens da especificação de requisitos.

Figura 3.12 Notações para a especificação de requisitos.

Notação	**Descrição**
Linguagem natural estruturada	Essa abordagem depende da definição de formulários-padrão ou *templates* para expressar a especificação de requisitos.
Linguagem de descrição de projeto	Essa abordagem utiliza uma linguagem como uma linguagem de programação, mas com recursos mais abstratos para especificar os requisitos pela definição de um modelo operacional do sistema.
Notações gráficas	Uma linguagem gráfica, complementada com anotações de texto, é utilizada para definir os requisitos funcionais do sistema. Um exemplo anterior dessa linguagem gráfica foi SADT (Ross, 1977; Schoman e Ross, 1977). Mais recentemente, as descrições de use-case (Jacobson *et ai.*, 1993) foram utilizadas. Essas linguagens serão discutidas no próximo capítulo.
Especificações matemáticas	São notações com base em conceitos matemáticos, como uma máquina de estados finitos e conjuntos. Essas especificações não ambíguas reduzem as discussões entre cliente e fornecedor sobre a funcionalidade do sistema. Contudo, a maioria dos clientes não compreende as especificações formais e reluta em aceitá-las no momento de uma contratação de sistema.

3.3.1Especificações em linguagem estruturada

A linguagem natural estruturada é uma forma restrita da linguagem natural, que se destina a escrever requisitos de sistema. A vantagem dessa abordagem é que ela mantém a maior parte da facilidade de expressão e compreensão da linguagem natural, mas garante que algum grau de uniformidade seja imposto à especificação. As notações de linguagem estruturada podem limitar a terminologia utilizada e utilizar templates para especificar os requisitos de sistema. Elas também podem incorporar princípios de controle procedentes de linguagens de programação e destaques gráficos para dividir a especificação.

Um projeto que utilizou linguagem natural estruturada para especificar requisitos de sistema é descrito por Heninger (1980). Formulários com propósito especial foram projetados para descrever a entrada, a saída e as funções do sistema de software de uma aeronave. Os requisitos de sistema foram especificados com o uso desses formulários.

Para utilizar uma abordagem com base em formulários para especificar os requisitos de sistema, é preciso definir um ou mais formulários ou templates-padrão para expressar os requisitos. A especificação pode ser estruturada em tomo dos objetos manipulados pelo sistema, das funções realizadas ou dos eventos processados por ele. Um exemplo dessa especificação com base em formulários é mostrado na Figura 3.13. Essa é uma definição mais detalhada da função de nó Add para o sistema de projeto-edição, definido na Figura 3.11.

Quando um formulário-padrão é utilizado para especificar os requisitos funcionais, as seguintes informações devem ser incluídas:

1- uma descrição da função ou entidade que está sendo especificada;

2- uma descrição de suas entradas e de onde elas se originam;

3- uma descrição de suas saídas e para onde elas prosseguirão;

4- uma indicação de que outras entidades são utilizadas (a parte referente a 'requer');

5- se uma abordagem funcional for utilizada, será 'chamada' uma pré-condição estabelecendo o que deve ser verdadeiro antes da função; e também será 'chamada' uma pós-condição especificando o que é verdadeiro depois da função, e

6- uma descrição dos efeitos colaterais (se existirem) da operação.

Figura 3.13. Especificação de requisitos de sistema utilizando-se um formulário-padrão.

Eclipse/Workstation/Ferramentas/DE/FS
Função Adicionar nós.

Descrição Adiciona um nó em um desenho existente. O usuário seleciona o tipo de nó e seu posicionamento. Quando adicionado ao desenho, o nó se torna a seleção atual. O usuário escolhe a posição do nó movimentando o cursor para a área em que o nó será adicionado.

Entradas Tipo de nó, Posição do nó, Identificador do desenho.

Origem Tipo de nó e Posição do nó são entradas fornecidas pelo usuário; Identificador de desenho se origina da base de dados.

Saídas Identificador do desenho.

Destino O banco de dados do desenho. O desenho é designado para a base de dados, no término da operação.

Requer Gráfico de desenho associado ao identificador de desenho de entrada. **Pré-condição** O desenho é aberto e exibido na tabela do usuário.

Pós-condição O desenho é imutável, a não ser pela adição de um nó do tipo especificado em dada posição.

Efeitos colaterais Nenhum.

Definição: ECLlPSE;Workstation/Ferramentas/DE/RD/

Utilizar especificações formatadas elimina alguns dos problemas de especificações em linguagem natural, uma vez que há menos variação na especificação e os requisitos são agrupados de modo mais eficaz. Contudo, pode permanecer alguma ambigüidade na especificação. Métodos alternativos, que utilizam notações mais estruturadas, como uma PDL (descrita a seguir), avançam um pouco em direção à resolução dos problemas de ambigüidade da especificação. Contudo, quem não é especialista geralmente a considera de difícil compreensão.

3.3.2 Especificação de requisitos com o uso de uma PDl

Ao contrário das ambigüidades inerentes à especificação em linguagem natural, é possível descrever os requisitos operacionalmente com o uso de uma PDL *(program description language* -linguagem de descrição de programa). Uma PDL é uma linguagem derivada de uma linguagem de programação como lava ou Ada. Ela pode conter princípios mais abstratos, adicionais, para aumentar seu poder de expressão. A vantagem de utilizar uma PDL é que ela pode ser verificada sintática ou semanticamente por ferramentas de software. Omissões e inconsistências de requisitos podem ser inferidas a partir dos resultados dessas verificações.

As PDLs resultam em especificações muito detalhadas e, algumas vezes, estão muito perto da implementação para sua inclusão em um documento de requisitos. Contudo, recomendo sua utilização em duas situações:

1. Quando uma operação é especificada como uma seqüência de ações mais simples e a ordem de execução é importante. As descrições dessas seqüências em linguagem natural são, muitas vezes, confusas, particularmente se condicionais aninhadas e loops estiverem envolvidos. Isso é ilustrado na Figura 2.14, que define parte da especificação de uma operação de um caixa automático *(automatic teller machine* -ATM). Utilizei lava como a PDL, mas deliberadamente excluí partes da descrição, para poupar espaço. Uma descrição completa em lava da ATM pode ser obtida por download a partir das páginas Web deste livro.

2. Quando interfaces de hardware e software tiverem de ser especificadas. Em muitos casos, as interfaces entre subsistemas são definidas na especificação de requisitos de sistema. O uso de uma PDL permite que tipos e objetos de interface sejam especificados.

Se o leitor dos requisitos de sistema estiver familiarizado como a PDL utilizada, especificar os requisitos dessa maneira pode torná-los menos ambíguos e mais fácil de ser compreendidos. Se a PDL tiver como base a linguagem de implementação, existirá uma transição natural dos requisitos para o projeto. A possibilidade de má interpretação fica reduzida. Os especificadores não precisam ser treinados em outra linguagem descritiva.
Há desvantagens nessa abordagem de especificação de requisitos:

1. A linguagem utilizada para escrever a especificação pode não ser suficientemente expressiva para descrever a funcionalidade do sistema.

2. A notação só é compreensível para pessoas que tenham algum conhecimento de linguagem de programação.

3. O requisito pode ser considerado um esboço de especificação de projeto, em vez de um modelo para ajudar o usuário a compreender o sistema.

Uma maneira efetiva de utilizar essa abordagem de especificação é combiná-la com o uso da linguagem natural estruturada. Uma abordagem com base em formulários pode ser utilizada para especificar o sistema completo. Uma PDL pode, então, ser utilizada para definir seqüências de controle ou interfaces mais detalhadamente.

3.3.3 Especificação de interface.

A grande maioria dos sistemas de software deve operar com outros sistemas que já foram implementados e instalados em um ambiente. Se o novo sistema e os sistemas existentes devem trabalhar em conjunto, as interfaces de sistemas existentes precisam ser especificadas com precisão. Essas especificações devem ser definidas no início do processo e incluídas (talvez como um apêndice) no documento de requisitos.

Existem três diferentes tipos de interfaces que podem precisar ser definidas:

1. Interfaces de procedimento, em que subsistemas existentes oferecem uma gama de funções, que são acessadas ao chamar procedimentos de interface.

2. Estruturas de dados, que são transmitidas de um subsistema para outro. Uma **PDL** baseada em lava pode ser utilizada para isso, com a' estrutura de dados sendo descrita com o uso de uma definição de classe com atributos representando campos da estrutura. Contudo, acho que os diagramas de relacionamento de entidades são melhores para esse tipo de descrição.

3. Representações de dados (como a ordenação de bits) que foram estabelecidas para um subsistema existente. Java não é compatível com essa especificação de representação detalhada; assim, não recomendo o uso da **PDL** baseada em Java, nesse caso.

As notações formais permitem que as interfaces sejam definidas de maneira não ambígua, mas sua natureza especializada significa que elas não são compreensíveis sem um treinamento especial. Elas raramente são utilizadas na prática para a especificação de interfaces, embora, a meu ver, são perfeitamente adequadas para esse propósito. Menos formais, as descrições de interfaces em **PDL** são uma conciliação entre a facilidade de compreensão e a precisão; mas, geralmente, elas são mais precisas do que a especificação de interfaces em linguagem natural.

A Figura 3.15 é um exemplo de uma definição do primeiro desses tipos de interfaces. Nesse caso, trata-se da interface procedural, oferecida por um servidor de impressão. Ela gerencia uma fila de pedidos de impressão de arquivos em diferentes impressoras. Os usuários podem examinar a fila associada a uma impressora e remover seus serviços de impressão dessa fila. Eles podem também transferir o serviço de uma impressora para outra.

A especificação na Figura 3.15 é um modelo abstrato do servidor de impressão, sem revelar quaisquer detalhes da interface. Essa funcionalidade das operações de interface pode ser definida utilizando-se linguagem natural estruturada (Figura 3.10), **PDL** com base em Java (Figura 3.14) ou uma notação formal.

3.4 O Documento de requisito de software.

O documento de requisitos de software - às vezes, chamado de SRS *(software requirements specification)* ou especificação de requisitos de software é a declaração oficial do que é exigido dos desenvolvedores de sistema. Ele deve incluir os requisitos de usuário para um sistema e uma especificação detalhada dos requisitos de sistema. Em alguns casos, os requisitos de usuário e de sistema podem ser integrados em uma única descrição. Em outros casos, os requisitos de usuário são definidos em uma introdução à especificação dos requisitos de sistema. Se houver um grande número de requisitos, os requisitos detalhados de sistema poderão ser apresentados como documentos separados.

O documento de requisitos tem um conjunto diversificado de usuários, abrangendo desde a alta gerência da organização, que está pagando pelo sistema, até os engenheiros responsáveis pelo desenvolvimento do software. A Figura 3.16 (Kotonya e Sommerville, 1998) ilustra os possíveis usuários do documento e como eles o utilizam.
Heninger (1980) sugere que existem seis requisitos aos quais um documento de requisito de software deveria satisfazer:
.Deveria especificar somente o comportamento externo do sistema.

Figura 3.14. Uma descrição em PDL da operação de um ATM.

```
class ATM {
/ / declarations here
public statíc void maín (String args[]) throws InvalidCard { try (
thísCard.read () ; / / may throw InvalidCard exception pin = Keypad.readPin () ;
attempts = 1 ;
while ( !thisCard.pin.equals (pin) & attempts < 4 )
{ pin = KeyPad.readPin () ; attempts = attempts + 1
}
if (!thisCard.pin.equals (pin))
throw new InvalidCard ("Bad PIN") ; thisBalance = thisCard.getBalance () ;
do { Screen.prompt (" Please select a service ") ; service = Screen.touchKey () ;
switch (service) {
case Services.withdrawaIWithReceipt: receiptRequired = true ;
case Services.withdrawaINoReceipt:
amount = KeyPad.readAmount () ; if (amount > thisBalance)
{ Screen. printmsg ("Balance insufficient") ;
break;
}
Dispenser.deliver (amount) ;
newbalance = thisBalance -amount ; if (receiptRequired)
Receipt.print (amount, newBalance) ;
break;
/ / other service descriptions here
default: break ;
}
while (service != Services.quit) ; thisCard.returnToUser ("Please take your card") ;
}
catch (InvalidCard e)
{ Screen.printmsg (Ulnvalid card ar PINU)
}
/ / other exception handling here
} //main () } / /ATM
```

Figura 3.15 A descrição em PDL baseada em Java de uma interface de servidor de impressão (PrintServer).

```
interface PrintServer {
/ / define um servidor de impressora abstrato
/ / requer: impressora de interface Printer, interface PrintDoc
/ / fornece: inicialize, print, displayPrintQueue, cancelPrinUob, switchPrinter
void initialize ( Printer p ) ;
void print ( Printer p, PrintDoc d ) ;
void displayPrintQueue ( Printer p ) ;
void cancelPrintJob (Printer p, PrintDoc d) ;
void switchPrinter (Printer p 1, Printer p2, PrintDoc d) ; } / /PrintServer
```

Figura 3.16 Usuários de um documento de requisitos.

Clientes de Sistema	Especificam os requisitos e os lêem para verificar se eles atendem as suas necessidades. requisitos. Especificam as mudanças nos requisitos.
Analista de Sistema	Utilizam o documento de requisitos para planejar um pedido de proposta para o sistema e para planejar o processo de desenvolvimento do sistema.
Engenheiros de Sistema	Utilizam os requisitos para compreender que sistema deve ser desenvolvido
Engenheiros de Teste de Sistema	Utilizam os requisitos para desenvolver testes de validação para o sistema.
Engenheiros de Manutenção de Sistema	Utilizam os requisitos para ajuda a compreende o sistema e as relações entre suas partes.

.Deveria especificar as restrições à implementação.
.Deveria ser fácil de ser modificado.
.Deveria servir como uma ferramenta de referência para os responsáveis pela manutenção do sistema.
.Deveria registrar a estratégia sobre o ciclo de vida do sistema.
.Deveria caracterizar respostas aceitáveis para eventos indesejáveis.

ANÁLISE E GERÊNCIA DE REQUISITOS

Embora tenham sido sugeridos há mais de 20 anos, esses são bons conselhos. Contudo, algumas vezes é difícil especificar sistemas em termos do que eles farão (seu comportamento externo). Inevitavelmente, devido às restrições originárias de sistemas existentes, o projeto de sistema é restringido, e isso deve ser refletido no documento de requisitos. Outro conselho, como a necessidade de registrar uma estratégia sobre o ciclo de vida do sistema, é muito bem aceito, mas não amplamente seguido, quando são escritos os documentos de requisitos.

Uma série de diferentes organizações de grande porte, como o Departamento de Defesa dos Estados Unidos e o IEEE, definiram padrões para os documentos de requisitos. Davis (1993) discute alguns desses padrões e compara seu conteúdo. O padrão mais amplamente conhecido é o padrão IEEE/ANSI 830-1993 (IEEE, 1993). Thayer e Dorfman (1997), em sua excelente coleção de artigos sobre engenharia de requisitos, incluem uma especificação completa desse padrão. Esse padrão do IEEE sugere a seguinte estrutura para os documentos de requisitos:

1- Introdução

- 1.1- Propósito do documento de requisito
- 1.2- Escopo do produto
- 1.3- Definições, acrônimos e abreviações
- 1.4- Referências
- 1.5- Visão geral do restante do documento

2. Descrição geral

- 2.1 Perspectiva do produto
- 2.2 Funções do produto
- 2.3 Características do usuário
- 2.4 Restrições gerais
- 2.5 Suposições e dependências

3. Requisitos específicos que abrangem os requisitos funcionais, não funcionais e de inter-face.

> Essa é, obviamente, a parte mais substancial do documento, mas, devido à ampla variabilidade na prática organizacional, não é apropriado definir uma estrutura-padrão para essa seção. Os requisitos podem documentar interfaces externas, descrever funcionalidade e desempenho de sistema, especificar requisitos lógicos de banco de dados, restrições de projeto, propriedades emergentes do sistema e características de qualidade.

4. Apêndices

5. Índice

Embora o padrão do IEEE não seja ideal, ele contém uma grande quantidade de boas orientações sobre como escrever os requisitos e como evitar problemas. Ele é muito geral para ser considerado um padrão organizacional por si só. Contudo, ele pode ser adaptado para definir um padrão dirigido às necessidades de uma organização em particular. A Figura 3.17 ilustra uma possível organização para um documento de requisitos, que se baseia no padrão IEEE. Contudo, fiz uma ampliação para incluir informações sobre a evolução prevista para o sistema, como foi recomendado por Heninger.

Naturalmente, as informações incluídas em um documento de requisitos devem depender do tipo de software que está sendo desenvolvido e da abordagem de desenvolvimento que é utilizada. Se uma abordagem evolucionária for adotada para um produto de software (digamos), o documento de requisitos se omitirá em relação a muitos dos capítulos detalhados, sugeridos anteriormente. Nesse caso, os projetistas e os programadores utilizam seu julgamento para decidir como atender aos requisitos dos usuários para o sistema.

Por outro lado, quando o software é parte de um grande projeto de engenharia de sistemas, que inclui sistemas de hardware e software que interagem entre si, freqüentemente, é essencial definir os requisitos com um refinado nível de detalhes. Isso significa que os documentos de requisitos provavelmente serão muito extensos e que eles devem incluir a maioria dos capítulos mostrados na Figura 3.17. Para documentos extensos, é particularmente importante incluir uma tabela abrangente de conteúdo e um índice de documentos, de modo que os leitores possam encontrar as informações de que precisam.

Figura 3.17 A estrutura de um documento de requisitos.

Capítulo	Descrição
Prefácio	Ele deve definir o público a que se destina o documento e descrever seu histórico da versão, incluindo uma lógica para a criação de uma nova versão e um sumário das mudanças feitas em cada versão.
Introdução	Deve descrever a necessidade do sistema. Deve descrever brevemente suas funções e explicar como ele deverá operar com outros sistemas. Deve descrever como o sistema se ajusta aos negócios em geral ou aos objetivos estratégicos da organização que está encomendado o software.
Glossário	Deve definir os termos técnicos utilizados

	no documento. Não se deve fazer suposições sobre a experiência ou o conhecimento apurado do leitor.
Definição de Requisitos do usuário	Os serviços fornecidos para o usuário e os requisitos não funcionais do sistema dever ser descritos nesta seção. Essa descrição pode utilizar linguagem natural, diagramas ou outras notações que sejam compreensíveis pelos clientes. Padrões de produtos e de processos a serem seguidos devem ser especificados.
Arquitetura de sistemas	Esse capítulo deve apresentar uma visão geral de alto nível do arquiteto de sistema prevista, mostrando a distribuição de funções por meio de módulos de sistemas. Os componentes de arquitetura que estão sendo reutilizados devem ser destacados.
Especificação de requisitos do sistema	Deve descrever os requisitos funcionais e não funcionais com mais detalhes requisitos do sistema Se for necessário, outros detalhes podem também ser adicionados aos requisitos não funcionais; por exemplo, podem ser definidas interfaces com outros sistemas.
Modelos de sistemas	Devem ser estabelecidos um ou mais modelos de sistemas, mostrando o relacionamento entre os componentes de sistema e o sistema e seu ambiente. Esses podem ser os modelos de objeto, os modelos de fluxo de dados e os modelos semânticos de dados.
Evolução de sistema	Devem ser descritas as suposições fundamentais nas quais o sistema se baseia e as mudanças previstas, devidas à evolução de hardware, mudanças nas necessidades de usuário etc.
Apêndices	São fornecidos detalhes e informações específicas relacionadas à aplicação que está sendo desenvolvida. Exemplos de apêndices que podem ser incluídos são descrições de hardware e bases de dados.

	Os requisitos de hardware definem as configurações mínima e ótima para o sistema. Os requisitos de base de dados definem a organização lógica dos dados utilizados pelo sistema e os relacionamentos entre esses dados.
Índice	Podem ser incluídos vários índices no documento. Da mesma maneira que um índice normal alfabético, pode haver um índice de diagramas, um índice de funções, entre outros.

Pontos-chave

Os requisitos para um sistema de software estabelecem o que o sistema deve fazer e definem restrições sobre sua operação e implementação.

Os requisitos funcionais são declarações de funções que o sistema deve fornecer ou são descrições de como alguns cálculos devem ser realizados. Os requisitos de domínio são requisitos funcionais, provenientes de características do domínio de aplicação.

Os requisitos não funcionais são os requisitos do produto, que restringem o sistema a ser desenvolvido, os requisitos de processo que se aplicam ao processo de desenvolvimento e os requisitos externos. Eles freqüentemente se relacionam às propriedades emergentes do sistema e, portanto, se aplicam ao sistema como um todo.

Os requisitos de usuário se destinam às pessoas envolvidas no uso e na aquisição do sistema. Eles devem ser escritos utilizando-se linguagem natural, tabelas e diagramas, de modo que sejam compreensíveis.

Os requisitos de sistema se destinam a comunicar, de modo preciso, as funções que o sistema tem de fornecer. Para reduzir a ambigüidade, eles podem ser escritos em uma linguagem estruturada de algum tipo, que pode ser um formulário estruturado de linguagem natural, uma linguagem com base em uma linguagem de programação de alto nível ou uma linguagem especial para a especificação de requisitos.

O documento de requisitos de software é a declaração estabelecida dos requisitos do sistema. Ele deve ser organizado de modo que possa ser utilizado pelos clientes de sistema e pelos desenvolvedores de software.

Exercícios

ANÁLISE E GERÊNCIA DE REQUISITOS

1- Discuta os problemas da utilização de linguagem natural para definir os requisitos do usuário e do sistema e mostre, utilizando pequenos exemplos, como a linguagem natural estruturada em formulários pode ajudar a evitar algumas dessas dificuldades.

2- Descubra ambigüidades ou omissões no seguinte documento de requisitos de uma parte de um sistema de emissão de passagens.
Um sistema automático de emissão de passagens vende passagens de trem. Os usuários escolhem seu destino e apresentam um cartão de crédito e um número de identificação pessoal. A passagem é emitida e o custo dessa passagem é incluído em sua conta do cartão de crédito. Quando o usuário pressiona o botão para iniciar, uma tela de menu com os possíveis destinos é ativada, juntamente com uma mensagem para que o usuário selecione um destino. Uma vez selecionado um destino, pede-se que os usuários insiram seu cartão de crédito. A validade do cartão é checada e o usuário, então, deve fornecer um número de identificação pessoal. Quando a transação de crédito é validada, a passagem é emitida.

3- Reescreva a descrição mostrada anteriormente utilizando a abordagem estruturada, descrita neste capítulo. Resolva as ambigüidades identifica das de maneira apropriada.

4- Escreva requisitos de sistema para o sistema em questão, utilizando uma notação com base em lava. Você pode fazer quaisquer suposições razoáveis sobre o sistema. Preste atenção, em particular, à especificação de erros do usuário.

5- Utilizando a técnica sugerida aqui, em que a linguagem natural é apresentada de maneira-padrão, escreva requisitos plausíveis do usuário para as seguintes funções:
.Um sistema de bomba de gasolina para auto-atendimento, que inclui uma leitora de cartão de crédito. O cliente passa o cartão pela leitora e registra a quantidade de combustível solicitada. O combustível é fornecido e a conta do cliente é debitada.
.A função de fornecer dinheiro em um caixa automático de banco.
.A função de verificar e corrigir a ortografia, em um processador de textos.

6- Descreva três tipos diferentes de requisitos não funcionais que possam ser colocados em um sistema. Dê exemplos de cada um desses três tipos de requisitos.

7- Escreva um conjunto de requisitos não funcionais para o sistema de emissão de passagens, descrito anteriormente, estabelecendo sua confiabilidade e seu tempo de resposta.

8- Quais são os requisitos para uma linguagem de programação, de modo que ela seja adequada para definir as especificações de interface? Comente sobre a adequação das linguagens C, lava e Ada para esse propósito.

ANÁLISE E GERÊNCIA DE REQUISITOS

9- Sugira como um engenheiro responsável pelo esboço de uma especificação de requisitos poderia manter o acompanhamento dos relacionamentos entre os requisitos funcionais e não funcionais.

10- Você conseguiu um emprego com um usuário de software que contratou seu funcionário anterior para desenvolver um sistema para eles. Você descobre que a interpretação de sua empresa dos requisitos é diferente da interpretação dada pelo seu funcionário anterior. Discuta o que você deveria fazer nessa situação. Você sabe que os custos para seu atual funcionário deverão aumentar, se as ambigüidades não forem resolvidas, e tem também uma responsabilidade de confidencialidade para com seu funcionário anterior.

4. Gerenciamento de Projetos

Objetivos

Proporcionar uma visão geral do gerenciamento de projetos de software. Depois de ler este capítulo, você poderá:

- compreender as diferenças entre o gerenciamento de projetos de software e outros tipos de gerenciamento de projetos de engenharia;

- conhecer as principais tarefas dos gerentes de projeto de software;

- compreender por que o planejamento de projetos é essencial em todos os projetos de software;

- compreender como representações gráficas (diagramas de barras, diagramas de atividades) são utilizadas pelos gerentes de projeto para representar programações de projeto, e

- compreender o processo de gerenciamento de riscos e alguns dos riscos que podem surgir em projetos de software.

O fracasso de muitos grandes projetos de software, na década de 60 e no início da década de 70, foi a primeira indicação das dificuldades de gerenciamento de software. O software era entregue com atraso, não era confiável, custava várias vezes mais do que previam as estimativas originais e, muitas vezes, exibia características precárias de desempenho (Brooks, 1975).
Esses projetos não fracassaram porque os gerentes ou os programadores eram incompetentes.

Ao contrário, esses projetos grandes e desafiadores atraíam pessoas de capacitação acima da média. A falha residia na abordagem de gerenciamento utilizada. Técnicas de gerenciamento provenientes de outras disciplinas da engenharia eram aplicadas e mostravam-se ineficazes para o desenvolvimento de software.

A necessidade de gerenciamento é uma importante distinção entre o desenvolvimento profissional de software e a programação em nível amador. Necessitamos do gerenciamento de projetos de software porque a engenharia de software profissional está sempre sujeita às restrições de orçamento e de prazo. Essas restrições são estabelecidas pela organização que desenvolve o software. O trabalho do gerente de projeto de software é garantir que o projeto de software cumpra essas restrições e entregar um produto de software que contribua para as metas da empresa.

ANÁLISE E GERÊNCIA DE REQUISITOS

Os gerentes de software são responsáveis por planejar e programar o desenvolvimento do projeto. Eles supervisionam o trabalho para assegurar que ele seja realizado em conformidade com os padrões requeridos e monitoram o progresso para verificar se o desenvolvimento está dentro do prazo e do orçamento. O bom gerenciamento não pode garantir o sucesso do projeto. Contudo, o mau gerenciamento geralmente resulta no fracasso do projeto. O software é entregue com atraso, custa mais do que originalmente foi estimado e apresenta falha no cumprimento de seus requisitos.

Os gerentes de software fazem o mesmo tipo de trabalho que outros gerentes de projeto de engenharia. Contudo, a engenharia de software é distinta de outros tipos de engenharia, em uma variedade de modos que podem tomar o gerenciamento de software particularmente difícil. Algumas dessas diferenças são:

1. O *produto* é *intangível* O gerente do projeto de construção de um navio ou de um projeto de engenharia civil pode ver o produto sendo desenvolvido. Se há um atraso na programação, o efeito no produto é visível. Partes da estrutura estão obviamente inacabadas. O software é intangível; não pode ser visto ou tocado. Os gerentes de projeto de software não podem ver o progresso, eles dependem de outras pessoas para produzir a documentação necessária, a fim de examinar o progresso.

2. *Não há processo de software-padrão* Não temos uma compreensão clara das relações entre o processo de software e os tipos de produto. Nas disciplinas de engenharia com longo histórico, o processo é experimentado e testado. O processo de engenharia para determinados tipos de sistema, como uma ponte, é bem compreendido. Nossa compreensão do processo de software se desenvolveu significativamente nos últimos anos. Contudo, ainda não podemos prever com certeza quando um processo de software específico poderá causar problemas de desenvolvimento.

3. Grandes projetos de software são, freqüentemente, projetos únicos Os grandes projetos de software são normalmente diferentes de projetos anteriores. Os gerentes, portanto, na verdade, têm ampla experiência anterior, que pode ser utilizada para reduzir a incerteza nos planos. Conseqüentemente, é mais difícil prever problemas. Além disso, as rápidas mudanças tecnológicas em computadores e nas comunicações desatualizam as experiências prévias. As lições aprendidas com essas experiências podem não ser transferíveis para novos projetos.

Por causa desses problemas, não é de surpreender que alguns projetos de software sofram atrasos e ultrapassem o orçamento previsto. Os sistemas de software são, freqüentemente, novos e tecnicamente inovadores. Os projetos de engenharia (como novos sistemas de transporte), muitas vezes, têm problemas relacionados ao cronograma.

ANÁLISE E GERÊNCIA DE REQUISITOS

Em face das dificuldades envolvidas, talvez seja até notável que tantos projetos de software sejam entregues dentro do prazo e do orçamento.

4.1 Atividades de Gerenciamento

É impossível dar uma descrição do trabalho-padrão de um gerente de software. O trabalho varia muito, dependendo da organização e do produto a ser desenvolvido. Contudo, a maioria dos gerentes assume a responsabilidade, em algum estágio, por algumas das seguintes atividades ou por todas elas:

- elaboração de propostas
- planejamento e programação de projetos custo do projeto
- monitoramento e revisões de projetos
- seleção e avaliação de pessoal
- elaboração de relatórios e apresentações

O primeiro estágio de um projeto de software pode envolver a elaboração de uma proposta para executar o projeto. A proposta descreve os objetivos do projeto e como ele será realizado. Em geral, inclui também estimativas de custo e programação. Isso pode justificar por que o contrato do projeto deve ser delegado a uma organização ou equipe específica.

A elaboração da proposta é uma tarefa crucial, uma vez que a existência de muitas empresas de software depende de um número suficiente de propostas aceitas e contratos firmados. Pode não haver diretrizes definidas para essa tarefa; a elaboração de propostas é uma habilidade adquirida com a experiência. Aron (1983) apresenta uma argumentação sobre esse aspecto do trabalho de um gerente de projeto que ainda é relevante atualmente.

O planejamento de projeto se ocupa de identificar as atividades, os marcos e os documentos a serem produzidos em um projeto. Um plano deve, então, ser traçado para guiar o desenvolvimento em direção aos objetivos do projeto. A estimativa de custos é uma atividade que se ocupa de estimar os recursos requeridos para realizar o projeto.

O monitoramento de projeto é uma atividade continua. O gerente deve manter o acompanhamento do andamento do projeto e comparar os progressos e custos reais com os que foram planejados. Embora a maioria das organizações tenha mecanismos formais para o monitoramento, um gerente hábil pode formar um quadro mais nítido do que está acontecendo mediante discussões informais com a equipe de projetos.

ANÁLISE E GERÊNCIA DE REQUISITOS

O monitoramento informal pode prever problemas em potencial no projeto, no sentido de que pode revelar dificuldades à medida que elas ocorrem. Por exemplo, discussões diárias com a equipe de projetos podem revelar um problema específico ao encontrar uma falha de software. Em vez de esperar que um atraso na programação seja relatado, o gerente de software pode designar um especialista para resolver o problema ou pode decidir que o projeto seja convenientemente reprogramado.

Durante um projeto, é normal ocorrer uma série de revisões formais pelo gerenciamento de projetos. Essas revisões se ocupam de examinar o progresso geral, o desenvolvimento técnico do projeto, considerando o status do projeto em relação aos objetivos da organização que autoriza a operação do software.

O prazo para o desenvolvimento de um grande projeto de software pode ser de vários anos. Durante esse tempo, os objetivos organizacionais muito provavelmente serão modificados. Essas mudanças podem significar que o software não será mais necessário ou que os requisitos iniciais do projeto são inadequados. O gerenciamento pode decidir interromper o desenvolvimento do projeto ou modificá-lo a fim de acomodar as mudanças aos objetivos da organização.

Os gerentes de projeto, geralmente, precisam selecionar pessoal para trabalhar em seu projeto. O ideal é que uma equipe hábil e com experiência apropriada esteja disponível para trabalhar no projeto. Contudo, na maioria dos casos, os gerentes têm de se contentar com menos do que uma equipe ideal de projeto. As razões disso são:

1- orçamento do projeto pode não ser suficiente para contratar uma equipe bem paga, e pode ser necessário utilizar uma equipe menos experiente e que exija menor remuneração.

2- Uma equipe com a experiência apropriada pode não estar disponível, dentro ou fora da organização. Pode ser impossível recrutar uma equipe nova para o projeto. Dentro da organização, as melhores pessoas talvez já estejam alocadas em outros projetos.

3- A organização pode querer desenvolver as habilidades de seus funcionários, por isso pode ser designada uma equipe inexperiente para um projeto, a fim de que essa equipe aprenda e ganhe experiência.

O gerente de software precisa trabalhar dentro dessas limitações quando seleciona a equipe de projeto. Contudo, é bem provável que ocorram problemas, a não ser que pelo menos um membro do projeto tenha alguma experiência no tipo

de sistema a ser desenvolvido. Sem essa experiência, muitos erros simples podem ocorrer.

O gerente de projeto geralmente é o responsável pela preparação dos relatórios sobre o projeto, tanto para a organização do cliente como para as organizações dos fornecedores. Os gerentes de projeto devem redigir documentos concisos e coerentes, que resumam as informações fundamentais a partir de relatórios de projeto detalhados. Eles devem ser capazes de apresentar essas informações durante as revisões de andamento. Conseqüentemente, a capacidade de se comunicar de modo eficaz, tanto verbalmente como por escrito, é uma habilidade essencial para um gerente de projeto.

4.2 Planejamento de Projeto

O gerenciamento eficaz de um projeto de software depende de um planejamento acurado do andamento do projeto. O gerente de projeto deve prever os problemas que podem surgir e preparar soluções experimentais para esses problemas. Um plano traçado no início do projeto deve ser utilizado como guia para esse projeto. O plano inicial deve ser o melhor possível, em face das informações disponíveis. Ele deve evoluir à medida que o projeto seja desenvolvido e melhores informações se tomem disponíveis.

Uma estrutura para um plano de desenvolvimento de software é descrita na Seção 4.2.1. Assim como um plano de projeto, os gerentes podem também ter de traçar outros tipos de planos. Esses planos estão brevemente descritos na Figura 4.1 e são abordados com mais detalhes no capítulo referente ao assunto, em outra parte do livro.

Figura 4.1 Tipos de Plano.

Plano	Descrição
Plano de Qualidade	Descreve os procedimentos para teste de qualidade que serão em um projeto.
Plano de Validação	Descreve a abordagem, os recursos e o método utilizados para a validação do sistema.
Plano de Gerenciamento de Configuração	Descreve os procedimentos de gerenciamento e as estruturas a serem utilizadas.
Plano de Manutenção	Prevê os requisitos de manutenção dos sistema, os custos de manutenção e o esforço necessário.
Plano de Desenvolvimento em Equipe	Descreve como as habilidades e a experiência dos membros da equipe de projeto serão desenvolvidas.

O pseudocódigo, mostrado na Figura 4.2, descreve o processo de planejamento do projeto para o desenvolvimento de software. Ele mostra que o planejamento é um processo interativo, que só termina quando o próprio projeto for concluído. À medida que as informações sobre o projeto se tomam disponíveis durante seu desenvolvimento, o plano deve ser regularmente revisado. Os objetivos gerais da empresa são um importante fator que deve ser considerado quando se formula o plano de projeto. Quando esses objetivos são modificados, são necessárias mudanças no plano do projeto.

O processo de planejamento se inicia com uma avaliação das restrições (a data de entrega estabelecida, o pessoal disponível, o orçamento total, entre outras) que afetam o projeto. Essa avaliação é realizada em conjunto com uma estimativa dos parâmetros para o projeto, como sua estrutura, seu tamanho e sua distribuição de funções. Os marcos de progresso e os produtos a serem entregues são então definidos. O processo entra, então, em um loop. Uma programação para o projeto é traçada e as atividades definidas na programação são iniciadas ou liberadas para prosseguimento. Depois de algum tempo (normalmente, depois de duas a três semanas), o andamento é examinado e as discrepâncias são observadas. Como as estimativas dos parâmetros de projeto são experimentais, o plano sempre precisará ser modificado.

Os gerentes de projeto revisam as suposições sobre o projeto, à medida que mais informações se tomam disponíveis, refazendo a programação do projeto. Se o projeto estiver atrasado, talvez tenham de renegociar com o cliente as restrições do projeto e os produtos a serem entregues. Se essa renegociação' não for bem-sucedida e a programação não puder ser cumprida, poderá ser considerada uma revisão técnica do projeto. O objetivo dessa revisão é encontrar uma abordagem de desenvolvimento alternativa, que se limite às restrições do projeto e cumpra a programação.

Figura 4.2 Planejamento de projeto.

Estabeleça as restrições do projeto
Faça a avaliação inicial dos parâmetros do projeto
Defina os marcos do projeto e os produtos a serem entregues
while projeto não estiver terminado ou cancelado **loop**
Faça a programação do projeto
Inicie as atividades de acordo com a programação
Aguarde (por um período)
Examine o progresso do projeto
Revise as estimativas de parâmetros do projeto
Atualize a programação do projeto
Reanalise as restrições do projeto e os produtos a serem entregues
if surgirem problemas, **then**
Inicie revisão técnica
end if
end loop

Naturalmente, gerentes de projeto experientes não supõem que tudo correrá bem, pois sempre surgirão problemas de algum tipo durante um projeto. As hipóteses e as programações assumidas inicialmente devem ser pessimistas, em vez de otimistas. É preciso tentar prever todas as possíveis eventualidades no plano de projeto, para que as restrições e os marcos do projeto não tenham de ser renegociados a todo tempo no loop de planejamento.

4.2.1. O plano de projeto

O plano de projeto define os recursos disponíveis para o projeto, a estrutura analítica do trabalho e uma programação para realizar o trabalho. Em algumas organizações, o plano de projeto é um documento único que incorpora todos os diferentes tipos de plano apresentados anteriormente. Em outros casos, o plano de projeto se ocupa exclusivamente do processo de desenvolvimento. São incluídas referências a outros planos, mas os planos são separados entre si.

A estrutura do plano que descrevo aqui é adequada a esse último tipo de plano. Os detalhes do plano de projeto variam, dependendo do tipo de projeto e da empresa. Contudo, a maioria dos planos deve incluir as seguintes seções:

1. *Introdução* Descreve com brevidade os objetivos do projeto e define as restrições (por exemplo, orçamento e prazo) que afetam o gerenciamento do projeto.

2. *Organização de projeto* Descreve o modo como a equipe de desenvolvimento é organizada, as pessoas envolvidas e seus papéis na equipe.

3. *Análise de riscos* Descreve possíveis riscos de projeto, a probabilidade de surgir esses riscos e as estratégias propostas para a redução deles.

4. *Requisitos necessários de hardware* e *software* Descreve o hardware e o software de apoio exigidos para realizar o desenvolvimento. Se o hardware tiver de ser comprado, deverão ser incluídos os prazos de entrega e as estimativas de preço.

5. *Estrutura analítica* Descreve a divisão do trabalho em atividades e identifica os marcos e os produtos a serem entregues com cada atividade.

6. *Programação de projeto* Descreve as dependências entre atividades, o tempo estimado requerido para atingir cada marco e a alocação de pessoas nas atividades.

7. Mecanismos de monitoramento e *de elaboração de relatórios* Descreve os relatórios de gerenciamento que devem ser produzidos, quando eles devem ser produzidos e quais mecanismos de monitoramento são utilizados.

O plano de projeto deve ser revisado regularmente durante o projeto. Algumas partes, como a programação do projeto, serão freqüentemente modificadas; outras partes serão mais estáveis. Deve ser utilizada uma organização de documentos que permita a substituição direta de seções.

4.2.2 Marcos e produtos a serem entregues

Os gerentes precisam de informações. Como o software é intangível, essas informações somente podem ser fornecidas como documentos que descrevem o estado do software que está sendo desenvolvido. Sem essas informações, é impossível julgar o progresso e as estimativas de custos e as programações não podem ser atualizadas.

Quando se planeja um projeto, uma série de *marcos* deve ser estabelecida; um marco é o ponto final de uma atividade de processo de software. A cada marco deve haver uma saída formal, como um relatório, que possa ser apresentado para a gerência. Os relatórios de marcos não precisam ser documentos extensos, eles podem simplesmente ser um breve relatório das realizações em uma atividade de projeto. Os marcos devem representar o final de um estágio distinto e lógico de um projeto. Marcos indefinidos, por exemplo, 'codificação 80 por cento concluída', que são de impossível validação, são inúteis para o gerenciamento do projeto.

Um *produto a ser entregue* é o resultado do projeto entregue ao cliente. Em geral, ele é entregue no término de alguma fase importante do projeto, como a especificação, o projeto etc.

Normalmente, os produtos a serem entregues são marcos, mas os marcos não precisam ser produtos a serem entregues. Os marcos podem ser resultados internos do projeto, que são utilizados pelo gerente de projeto na verificação do andamento do projeto, sem serem entregues ao cliente.

Para estabelecer marcos, o processo de software deve ser dividido em atividades básicas com saídas associadas. Por exemplo, a Figura 4.3 mostra atividades envolvidas na especificação de requisitos, quando se utiliza a prototipação para ajudar a validar os requisitos. As principais saídas para cada atividade (os marcos de projeto) são mostradas. Os produtos a serem entregues do projeto são a definição e a especificação de requisitos.

4.3 Programação de Projeto

A programação de projeto é uma tarefa particularmente necessária para os engenheiros de projeto. Os gerentes estimam o tempo e os recursos exigidos para completar as atividades e os organizam em uma seqüência coerente. A menos que o projeto em programação seja similar a um projeto anterior, estimativas precedentes são uma base incerta para uma nova programação de projeto. A estimativa de programação é ainda mais complicada pelo fato de que diferentes projetos podem utilizar diferentes métodos de projeto e linguagens de implementação.

Se o projeto for tecnicamente avançado, as estimativas iniciais certamente serão otimistas, mesmo quando os gerentes tentam considerar todas as eventualidades. A esse respeito, a programação de software não é diferente da programação de qualquer outro tipo de grandes projetos avançados. O desenvolvimento de novas aeronaves, pontes e mesmo novos modelos de automóveis freqüentemente sofre algum atraso devido a problemas que não foram previstos. As programações, portanto, devem ser continuamente atualizadas, à medida que melhores informações sobre o progresso se tomam disponíveis. '

A programação de projeto (Figura 4.4) envolve dividir o trabalho total de um projeto em atividades distintas e avaliar o tempo necessário para completar essas atividades. De modo geral, algumas dessas atividades são realizadas em paralelo.

Figura 4.3 Marcos no processo de requisitos.

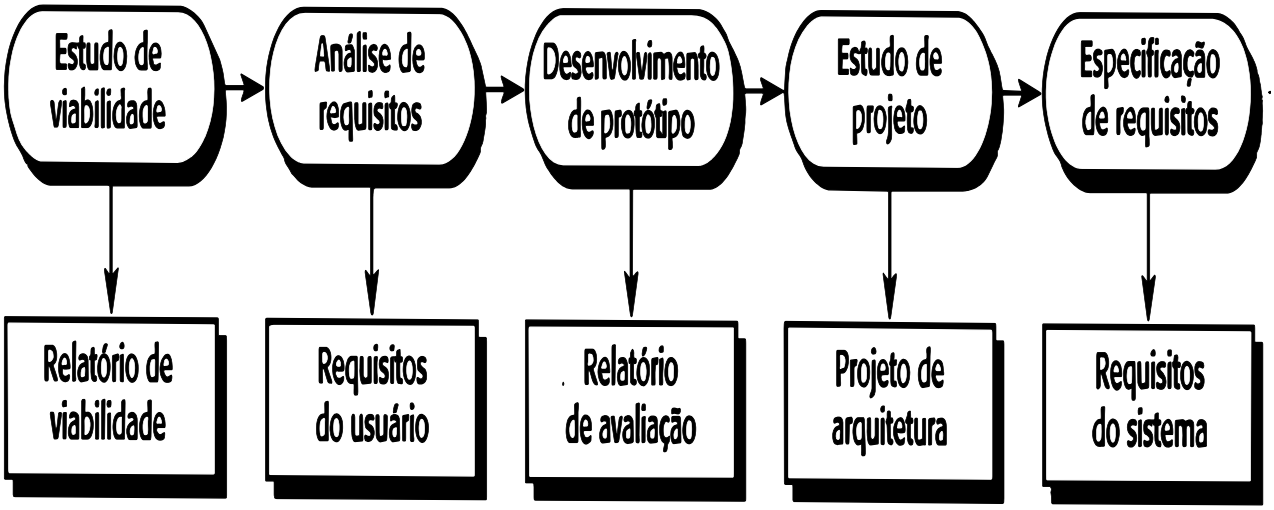
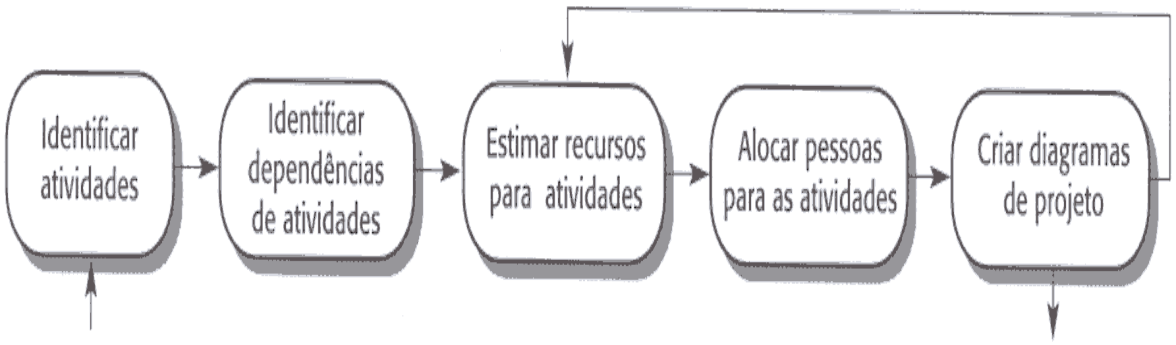


Figura 4.4 O processo de programação de projeto.



Os programadores de projeto devem coordenar essas atividades paralelas e organizar o trabalho, de modo que a força de trabalho seja otimizada. Eles devem evitar uma situação em que todo o projeto é atrasado porque uma tarefa importante não foi concluída.

As atividades de projeto devem normalmente durar pelo menos uma semana. Subdivisões menores significam que um tempo desproporcional deve ser dedicado para a estimativa e a revisão do diagrama. Também é útil estabelecer um tempo máximo de duração para qualquer atividade, que vai de oito a dez semanas. Se uma atividade demorar mais tempo do que isso, ela deverá ser subdividida para a programação e o planejamento do projeto.

Ao estimar as programações, os gerentes não devem presumir que todos os estágios do projeto estarão livres de problemas. As pessoas que trabalham no projeto podem ficar doentes ou pedir demissão; algum hardware pode apresentar defeito, e o software ou hardware de suporte, que são essenciais, podem ser entregues com atraso. Se o projeto for novo e tecnicamente avançado, determinadas partes dele poderão ser mais difíceis e exigirem mais tempo do que foi originalmente previsto.

Os gerentes também devem estimar os recursos necessários para completar cada tarefa, além dos prazos de execução. O recurso principal é o esforço humano requerido. Outros recursos podem ser o espaço em disco exigido em um computador, o tempo de uso necessário de um hardware especializado, como um simulador, e o orçamento de viagens para o pessoal da equipe de projetos.

Uma boa regra prática geral é fazer a estimativa como se nada fosse acontecer de errado e, então, aumentar a estimativa, a fim de poder resolver possíveis problemas. Outras eventualidades destinadas a resolver problemas não previstos também podem ser acrescentadas à estimativa. Essas eventualidades extras dependem do tipo de projeto, dos parâmetros do processo (prazo final, padrões etc.) e do nível de experiência dos engenheiros de software que trabalham no projeto. Como regra geral, sempre acrescento 30 por cento à minha estimativa original, para a solução de possíveis problemas, e mais 20 por cento, para lidar com outros aspectos que não pudemos prever.

A programação de projeto é normalmente representada como um conjunto de diagramas, mostrando a estrutura analítica do trabalho, dependências das atividades e alocação de pessoal. Esses tópicos serão discutidos na seção seguinte. Ferramentas de gerenciamento de software, como o Microsoft Project, são, hoje em dia, comumente utilizadas para automatizar a produção de diagramas.

4.3.1 Diagramas de barras e redes de atividades

Os diagramas de barras e as redes de atividades são notações gráficas utilizadas para ilustrar a programação de projeto. Os diagramas de barras mostram quem é responsável por cada atividade e para quando está programado o início e o

término da atividade. As redes de atividades mostram a dependência entre as diferentes atividades que constituem o projeto. Os diagramas de barras e as redes de atividades podem ser gerados automaticamente, a partir de um banco de dados de informações do projeto, com a utilização de uma ferramenta de gerenciamento de projeto.

Considere o conjunto de atividades indicado na Figura 4.5. Essa tabela mostra atividades, suas durações e suas interdependências. De acordo com a Figura 4.5, pode-se ver que a tarefa T3 depende da tarefa TI. Isso significa que TI deve ser concluída antes que T3 inicie. Por exemplo, TI pode ser a preparação do projeto de um componente e T3, a implementação desse projeto. Antes de iniciar a implementação, a preparação precisa estar terminada.

Considerando as dependências e as durações estimadas para as atividades, pode ser produzida uma rede de atividades que mostre as seqüências dessas atividades (Figura 4.6). Essa rede mostra quais atividades podem ser realizadas em paralelo e quais devem ser executadas em seqüência, devido à dependência em relação a uma atividade anterior. As atividades são representadas como retângulos. Os marcos e os produtos a serem entregues são indicados com cantos arredondados. As datas nesse diagrama mostram a data de início da atividade. O diagrama deve ser lido da esquerda para a direita e de cima para baixo..

Na ferramenta de gerenciamento de projeto utilizada para produzir esse diagrama, todas as atividades devem terminar em marcos. Uma atividade pode iniciar quando seu marco precedente (que pode depender de diversas atividades) tiver sido atingido. Portanto, na terceira coluna da Figura 4.5, mostrei também o marco correspondente (por exemplo, M5), que é atingido quando as atividades naquela coluna são concluídas (veja a Figura 4.6).

Antes de prosseguir de um marco para outro, todos os caminhos que levam até ele precisam estar completos. Por exemplo, a tarefa T9, mostrada na Figura 4.6, não pode ser iniciada até que as tarefas T3 e T6 estejam concluídas. A chegada ao marco M4 indica que essas tarefas foram completadas. .

O tempo mínimo necessário para terminar o projeto pode ser estimado considerando o caminho mais longo no diagrama de atividades (o caminho principal). Nesse caso, o tempo é de 11 semanas ou 55 dias trabalhados. Na Figura 6.6, o caminho critico é mostrado como uma seqüência de blocos com contorno em negrito. A programação total do projeto depende do caminho principal. Qualquer atraso na execução de qualquer atividade importante ocasiona o atraso do projeto.

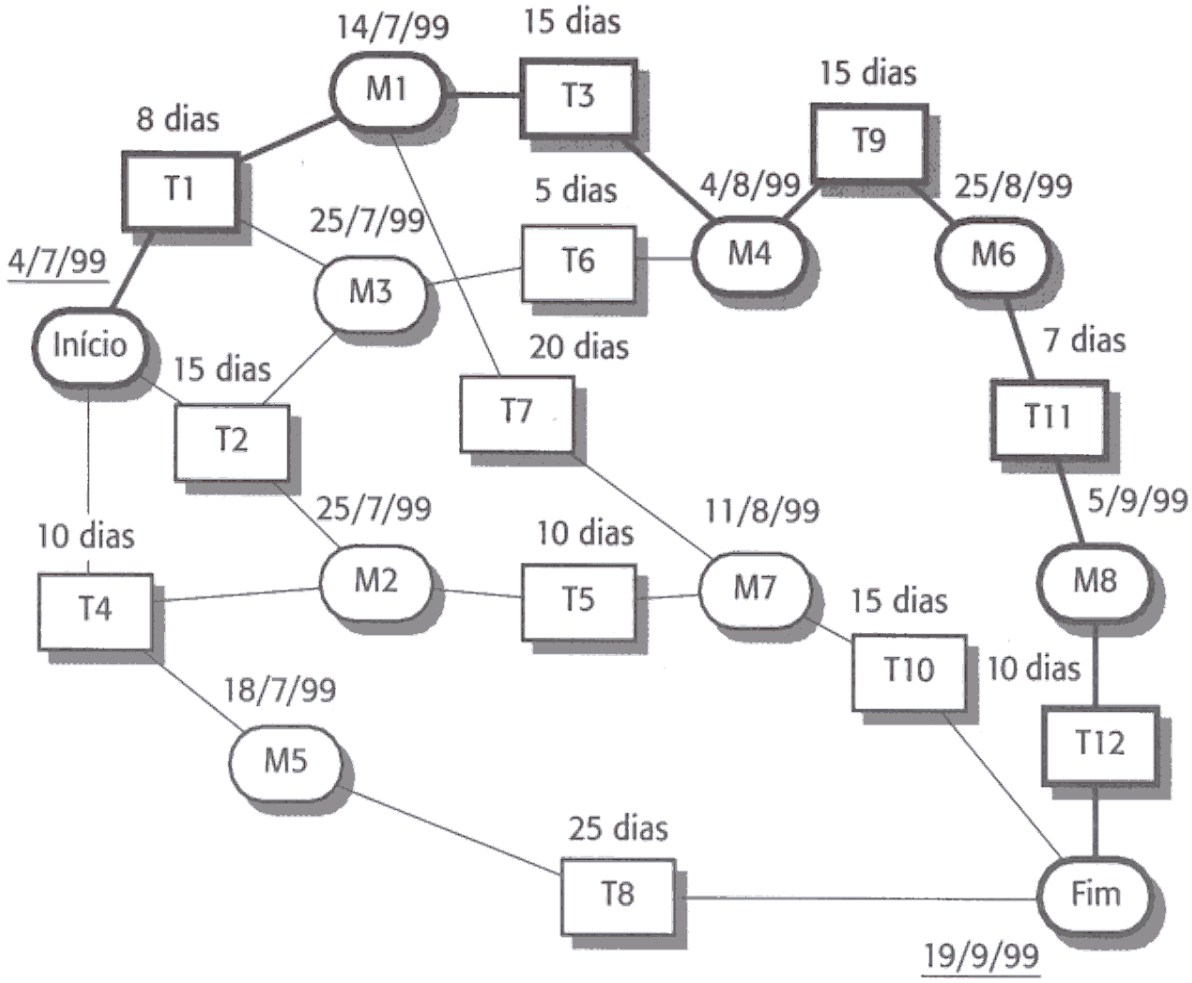
Todavia, os atrasos ocorridos nas atividades que não estão no caminho principal não precisam causar um atraso geral da programação. Contanto que esses atrasos não estendam as atividades a ponto de o tempo total exceder o tempo no

caminho principal, a programação do projeto não será afetada. Por exemplo, se a tarefa Ta se atrasar, talvez isso não afete a data final de término do projeto, uma vez que ela não está no caminho principal. O diagrama de barras do projeto (Figura 4.7) mostra a amplitude de atraso possível com uma barra sombreada.

Figura 4.5 Duração e dependências das tarefas.

Tarefa	Duração (dias)	Dependências
T1	8	
T2	15	
T3	15	T1 (M1)
T4	10	
T5	10	T2, T4 (M2)
T6	5	T1, T2 (M3)
T7	20	T1 (M1)
T8	25	T4 (M5)
T9	15	T3, T6 (M4)
T10	15	T5, T7 (M7)
T11	7	T9 (M6)
T12	10	T11 (M8)

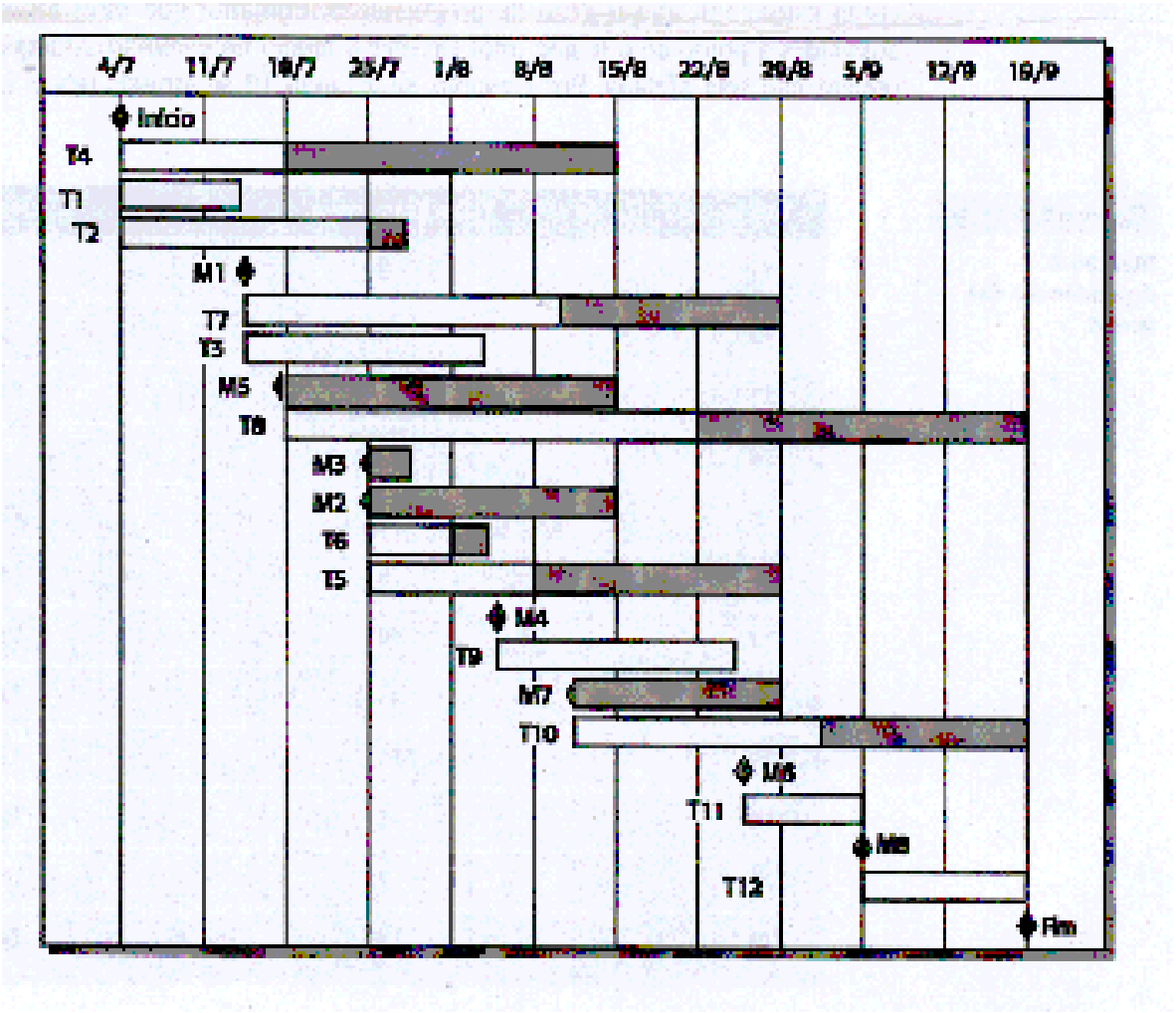
Figura 4.6 Uma rede de atividades.



Os gerentes também utilizam as redes de atividades quando alocam o trabalho de projeto. Elas podem fornecer esclarecimentos sobre dependências que não são intuitivamente óbvias. Pode ser possível modificar o projeto de sistema, de modo que o caminho principal seja encurtado. O período de programação do projeto pode ser diminuído pela redução do tempo gasto na espera pela finalização de atividades.

A Figura 4.7 é uma maneira alternativa de representar as informações sobre a programação do projeto. Trata-se de um diagrama de barras (algumas vezes, chamado de diagrama de Gantt, em homenagem ao seu inventor) mostrando um calendário de projeto e a data de início e término das atividades.

Figura 4.7 Diagrama de Barras de atividades.



Algumas das atividades na Figura 4.7 são seguidas por uma barra sombreada, cuja duração é calculada pela ferramenta de programação. Isso mostra que existe alguma flexibilidade na data de conclusão dessas atividades. Se uma atividade não for concluída a tempo, o caminho principal não será afetado até o fim do período marcado pela barra sombreada. As atividades que estão no caminho principal não têm nenhuma margem de erro e podem ser identificadas porque não existe nenhuma barra sombreada associada a elas.

Além de considerar as programações, os gerentes de projeto devem também considerar a alocação de recursos e, em particular, a alocação de pessoal para as atividades de projeto. A Figura 4.8 sugere uma alocação de pessoal para as atividades ilustradas na Figura 4.7.

A Figura 4.8 também pode ser processada por ferramentas de apoio ao gerenciamento de projeto, e um diagrama de barras produzido, que mostra os períodos nos quais o pessoal está designado para o projeto (Figura 4.9). A equipe não precisa estar designada para um projeto durante o tempo todo. Em alguns períodos de intervalo, as pessoas podem estar de férias, trabalhando em outros projetos, assistindo a cursos de treinamento ou realizando alguma outra atividade.

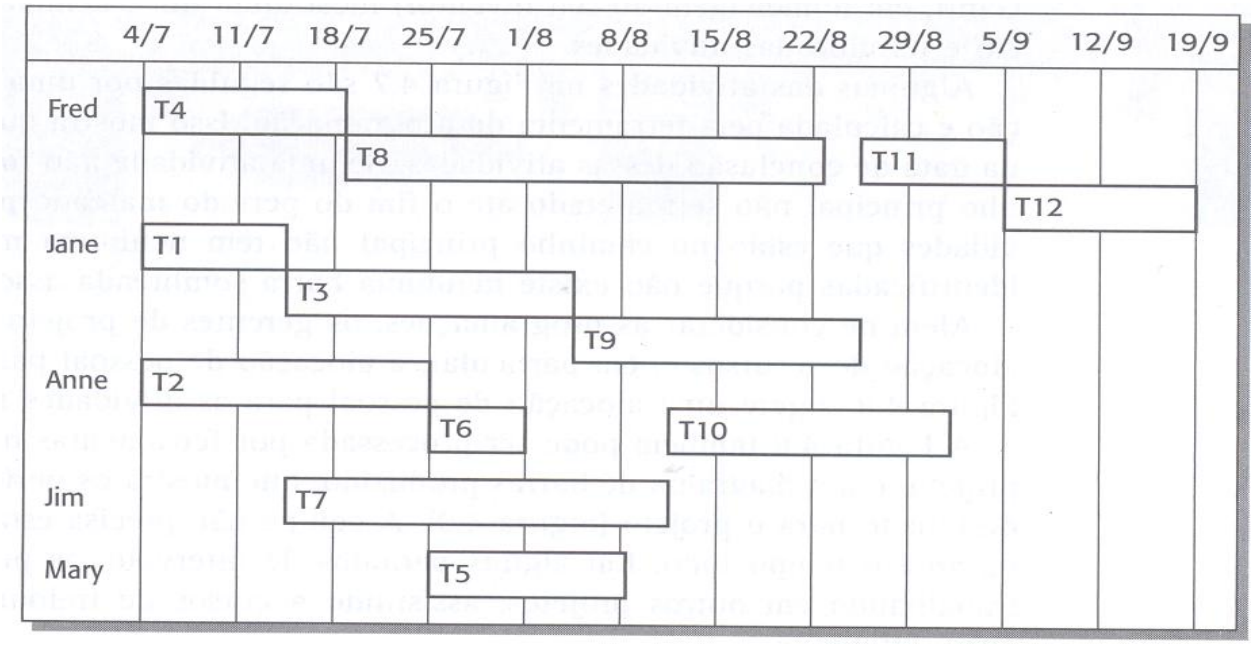
As grandes empresas geralmente empregam uma série de especialistas que trabalham em um projeto, conforme necessário. Isso pode causar problemas de programação. Se um projeto se atrasar enquanto um especialista estiver trabalhando nele, isso poderá ter um efeito prejudicial sobre os outros projetos, que também poderão se atrasar, porque o especialista não está disponível. Inevitavelmente, as programações iniciais de projetos estarão incorretas.

À medida que um projeto se desenvolve, as estimativas devem ser comparadas com o tempo real empregado. Essa comparação pode ser utilizada como base para revisar a programação das partes posteriores do projeto. Quando os números reais são conhecidos, o diagrama de atividades deve ser reexaminado. As atividades de projeto seguintes podem então ser reorganizadas, a fim de reduzir a extensão do caminho principal.

Figura 4.8 Alocação de Pessoas para atividades.

Tarefa	Engenheiro
T1	Jane
T2	Anne
T3	Jane
T4	Fred
T5	Mary
T6	Anne
T7	Jim
T8	Fred
T9	Jane
T10	Anne
T11	Fred
T12	Fred

Figura 4.9 Alocação de Equipes versus diagrama de tempo.



4.4 Gerenciamento de riscos

Uma importante tarefa de um gerente de projeto é prever os riscos que podem afetar a programação do projeto ou a qualidade do software em desenvolvimento e tomar as medidas necessárias para evitar esses riscos.
Os resultados da análise de riscos devem ser documentados no plano de projeto, juntamente com uma análise das conseqüências da ocorrência de algum fator de risco. Identificar riscos e traçar planos para minimizar seus efeitos sobre o projeto é o que se chama de gerenciamento de riscos (Hall, 1998; Ould, 1999).

De modo simplificado, podemos pensar no risco como uma probabilidade de que alguma circunstância adversa realmente venha a ocorrer. Os riscos podem ameaçar o projeto, o software que está sendo desenvolvido ou a organização. Essas categorias de riscos podem ser definidas como se segue:

1- ***Riscos relacionados ao projeto*** são os riscos que afetam a programação ou os recursos do projeto.
2- ***Riscos relacionados ao produto*** são os riscos que afetam a qualidade ou o desempenho do software que está em desenvolvimento.
3- ***Riscos para*** **os** ***negócios*** são os riscos que afetam a organização que está desenvolvendo ou adquirindo o software.

Naturalmente, essa não é uma classificação exclusiva. Se um programador experiente abandonar um projeto, isso poderá representar um risco de projeto (porque a entrega do sistema pode atrasar), um risco relacionado ao produto (porque um substituto pode não ser tão experiente e, portanto, pode cometer erros) e um risco para os negócios (porque aquela experiência não está disponível para elaborar propostas para negócios futuros).

O gerenciamento de riscos é particularmente importante para projetos de software, devido às incertezas inerentes que a maioria dos projetos enfrenta. Os riscos podem surgir como decorrência de requisitos mal definidos, de dificuldades em estimar o prazo e os recursos necessários para o desenvolvimento de software, da dependência de habilidades individuais e de mudanças nos requisitos, em razão de modificações nas necessidades do cliente. O gerente de projeto deve prever riscos, compreender o impacto desses riscos no projeto, no produto e nos negócios e tomar providências para evitar esses riscos. Planos de contingência podem ser traçados para que, se os riscos realmente ocorrerem, seja possível uma ação imediata que vise à recuperação.

Os tipos de risco que podem afetar um projeto dependem do projeto e do ambiente organizacional em que o software está sendo desenvolvido. Contudo, muitos riscos são considerados universais e alguns deles são descritos na Figura 4.10.

O processo de gerenciamento de riscos está ilustrado na Figura 4.11 e envolve vários estágios:

> *1- Identificação de riscos.* São identificados os possíveis riscos de projeto, produto e negócios.
>
> *2- Análise de riscos.* São avaliadas as possibilidades e as conseqüências da ocorrência desses riscos.
>
> *3- Planejamento de riscos.* São traçados planos para enfrentar os riscos, seja evitando-os seja minimizando seus efeitos sobre o projeto.
>
> *4- Monitoramento de riscos.* O risco é constantemente avaliado e os planos para a diminuição de riscos são revisados, à medida que mais informações sobre eles se tomam disponíveis.

O processo de gerenciamento de riscos, como todos os outros planejamentos de projeto, é um processo iterativo, que continua ao longo do projeto. Uma vez traçado um conjunto inicial de planos, a situação é monitorada. À medida que mais informações sobre os riscos se tomam disponíveis, eles têm de ser reavaliados e novas prioridades devem ser estabelecidas. Os planos para evitar riscos e os

planos de contingência podem ser modificados com o surgimento de novas informações sobre os riscos.

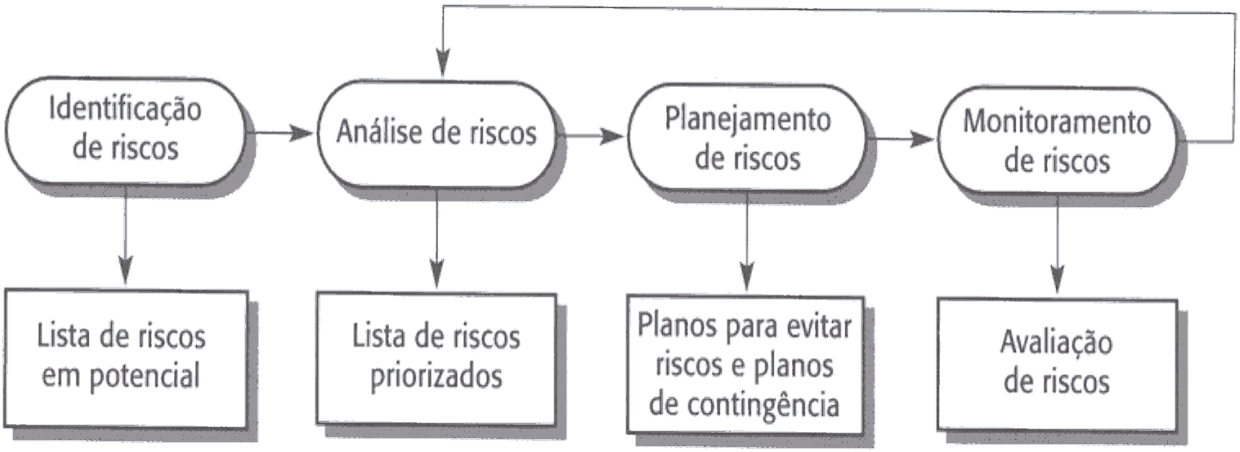
Os resultados do processo de gerenciamento de riscos devem ser documentados em um plano de gerenciamento de riscos. Essa fase deve incluir uma discussão sobre os riscos apresentados pelo projeto, uma análise desses riscos e os planos que são necessários para gerenciá-los.

Quando for apropriado, pode também incluir alguns resultados do gerenciamento de riscos, isto é, planos específicos de contingência a serem ativados caso o risco venha a ocorrer.

Figura 4.10 Possíveis riscos de software.

Risco	Tipo de risco	Descrição
Rotatividade de pessoal	Projeto	O pessoal experiente deixará o projeto antes do término.
Mudança de gerenciamento	Projeto	Haverá uma mudança no gerenciamento organizacional, com a definição de prioridades diferentes.
Indisponibilidade de hardware	Projeto	O hardware essencial ao projeto não será entregue dentro do prazo.
Alteração nos requisitos	Projeto e produto	Haverá maior número de mudanças nos requisitos do que o previsto.
Atrasos na especificação	Projeto e produto	As especificações de interfaces essenciais não estavam disponíveis dentro dos prazos.
Tamanho subestimado	Projeto e produto	O tamanho do sistema foi subestimado.
Baixo desempenho de ferramentas CASE	Produto	As ferramentas CASE que apóiam o projeto não apresentam desempenho conforme o previsto.
Mudanças na tecnologia	Negócios	A tecnologia básica sobre a qual o sistema está sendo construído foi superada por nova tecnologia.
Concorrência com o produto	Negócios	Um produto concorrente foi lançado no mercado, antes que o sistema fosse concluído.

Figura 4.11 O processo de gerenciamento de riscos



4.6.1. Identificação de riscos

A identificação de riscos é o primeiro estágio do gerenciamento de riscos; ela se preocupa em descobrir os possíveis riscos existentes no projeto. Em princípio, esses riscos não devem ser avaliados ou priorizados nesse estágio, embora, na prática, os riscos com conseqüências muito pequenas ou com muito pouca probabilidade não sejam normalmente levados em consideração.

A identificação de riscos pode ser realizada como um processo em equipe, utilizando-se uma abordagem de brainstorming (intensiva troca de idéias), ou pode simplesmente ter como base a experiência de um gerente. Para ajudar nesse processo, uma lista de possíveis tipos de riscos pode ser utilizada. Dentre esses tipos destacam-se:

1. *Riscos quanto à tecnologia* São os riscos que se originam de tecnologias de software ou de hardware, que são utilizadas como parte do sistema em desenvolvimento.

2. *Riscos quanto ao pessoal* São os riscos associados às pessoas da equipe de desenvolvimento.

3. *Riscos organizacionais* São os riscos que derivam do ambiente organizacional em que o software está sendo desenvolvido.

4. *Riscos quanto às ferramentas* São os riscos que derivam de ferramentas CASE e de outros tipos de software de apoio, utilizados para desenvolver o sistema.

5. *Riscos quanto aos requisitos* São os riscos que derivam de modificações nos requisitos do cliente e do processo de gerenciamento da modificação de requisitos.

6.*Riscos quanto à estimativa* São os riscos que derivam das estimativas feitas pelo gerenciamento sobre as características do sistema e os recursos necessários para construí-lo.

A Figura 4.12 apresenta alguns exemplos de possíveis riscos em cada uma dessas categorias. O resultado desse processo deve ser uma longa lista dos riscos que podem ocorrer e que podem afetar o produto, o processo e os negócios.

4.4.2 Análise de riscos

Durante o processo de análise de riscos, cada risco identificado é considerado individualmente e é feito um julgamento sobre a probabilidade e a seriedade desse risco. Não existe uma maneira fácil de fazer isso tudo depende do julgamento e da experiência do gerente
de projeto. Em geral, não é preciso uma avaliação numérica exata, mas essa análise deve ter como base uma análise com utilização de intervalos:

1- A probabilidade do risco pode ser determinada como muito baixa (menor do que 10 por cento), baixa (10-25 por cento), moderada (25-50 por cento), alta (50-75 por cento) ou muito alta (maior do que 75 por cento).

2- Os efeitos do risco podem ser determinados como catastróficos, sérios, toleráveis ou insignificantes.

Os resultados desse processo de análise devem, então, ser apresentados na tabela ordenada de acordo com a seriedade do risco. A Figura 4.13 ilustra esse aspecto par os riscos identificados na Figura 4.12. Obviamente, nesse caso, a avaliação de probabilidade e seriedade é arbitrária. Na prática, você precisa de informações detalhadas sobre o projeto, o processo, a equipe de desenvolvimento e a organização, para poder fazer essa avaliação.

Naturalmente, tanto a probabilidade quanto a avaliação dos efeitos de um risco podem se modificar, à medida que mais informações sobre o risco se tomam disponíveis e os planos de gerenciamento de risco são implementados. Portanto, essa tabela deve ser atualizada durante cada iteração do processo de risco.

Uma vez que os riscos tenham sido analisados e priorizados, deve ser feito um julgamento sobre quais são os riscos mais importantes a serem considerados

durante o projeto. Esse julgamento deve depender de uma combinação da probabilidade de o risco surgir e dos efeitos daquele risco.

Em geral, todos os riscos catastróficos devem sempre ser considerados, como também todos os riscos sérios que tenham mais do que uma probabilidade moderada de vir a ocorrer.
Boehm (1988) recomenda identificar e monitorar os 'dez maiores riscos', mas esse número é um tanto arbitrário. O número certo de riscos a serem monitorados deve depender do projeto; e eles podem ser um total de cinco ou quinze. Contudo, o número de riscos escolhido para monitorar deve ser gerenciável. Um número muito grande de riscos simplesmente poderia exigir que muitas informações fossem coletadas.

Figura 4.12 Riscos e Tipos de Risco

Tipos de risco	Riscos possíveis
Tecnologia	O banco de dados utilizado no sistema não pode processar tantas transações por segundo, como esperado. Componentes do software que deviam ser reutilizados contêm defeitos que limitam sua funcionalidade.
Pessoal	É impossível recrutar pessoal com a habilidade requerida. Pessoas importantes estão doentes e não disponíveis em períodos cruciais. O treinamento necessário para o pessoal não está disponível.
Organizacional	A organização está estruturada de maneira que diferentes gerências são responsáveis pelo projeto. Problemas financeiros organizacionais forçam reduções no orçamento do projeto.
Ferramentas	O código gerado pelas ferramentas CASE é ineficiente. As ferramentas CASE não podem ser integradas.
Requisitos	São propostas mudanças nos requisitos, que exigem significativo retrabalho. Os clientes não compreendem o impacto das mudanças nos requisitos.
Estimativa	O tempo requerido para desenvolver o software é subestimado. A taxa de solução de defeitos é subestimada. O tamanho do software é subestimado.

Figura 4.13 Análise de Riscos.

Risco	Probabilidade	Efeitos
Problemas financeiros organizacionais forçam reduções no orçamento do projeto.	Baixa	Catastróficos
É impossível recrutar pessoal com as habilidades requeridas para o projeto.	Alta	Catastróficos
Pessoas-chave estão doentes em períodos cruciais do projeto.	Moderada	Sérios
Componentes de software que deviam ser reutilizados contêm defeitos que limitam sua funcionalidade.	Moderada	Sérios
São propostas mudanças nos requisitos, que exigem significativo trabalho.	Moderada	Sérios
A organização está estruturada de maneira que diferentes gerências são responsáveis pelo projeto.	Alta	Sérios
O banco de dados utilizado no sistema não pode processar tantas transformações por segundo, como esperado.	Moderada	Sérios
O tempo requerido para desenvolver o software é subestimado.	Alta	Sérios
As ferramentas CASE não podem ser integradas.	Alta	Toleráveis
Os clientes não compreendem o impacto das mudanças nos requisitos.	Moderada	Toleráveis
O treinamento necessário para o pessoal não está disponível.	Moderada	Toleráveis
A taxa de solução de efeitos	Moderada	Toleráveis

é subestimada.		
O tamanho do software é subestimado.	Alta	Toleráveis
O código gerado pelas ferramentas CASE é ineficiente.	Moderada	Insignificante

A partir dos riscos identificados na Figura 4.13, é apropriado considerar todos os oito riscos que podem ter conseqüências catastróficas ou sérias.

4.4.3 Planejamento de riscos

O processo de planejamento de riscos considera cada um dos riscos mais importantes que foram identificados e define estratégias para gerenciá-lo. Mais uma vez, não há nenhum processo simples que possa ser seguido para estabelecer planos de gerenciamento de riscos. Isso depende do julgamento e da experiência do gerente de projeto. A Figura 4.14 mostra possíveis estratégias que foram identificadas para os principais riscos, mostrados na Figura 4.13.

As estratégias classificam-se em três categorias:

1-Estratégias preventivas. Seguir essas estratégias significa que a probabilidade de o risco surgir será reduzida. Um exemplo de estratégia preventiva é a utilizada para lidar com componentes defeituosos, mostrada na Figura 4.14.

2-Estratégias de minimização. Seguir essas estratégias significa que o impacto do risco será reduzido. Um exemplo de estratégia de minimização de riscos é a utilizada quando alguém está doente, mostrada na Figura 4.14.

Figura 4.14 Estratégias de Gerenciamento de Riscos.

Risco	Estratégia
Problemas financeiros organizacionais	Prepare um documento informativo para a alta gerência, mostrando como o projeto presta uma contribuição muito importante para os objetivos da empresa.
Problemas de recrutamento	Alerte o cliente sobre as dificuldades em potencial e a possibilidade de atrasos; investigue a compra de componentes.
Doença de pessoas da equipe	Reorganize a equipe de maneira que haja mais sobreposição de trabalho e, portanto, as pessoas compreendam as tarefas umas das outras.
Componentes defeituosos	Substitua componentes potencialmente defeituosos por componentes comprados e que tenham confiabilidade reconhecida.
Alterações nos requisitos	Extraia informações que podem ser rastreadas, para avaliar o impacto das mudanças nos requisitos, maximize a inclusão de informações no projeto.
Reestruturação organizacional	Prepare um documento informativo para a alta gerência, mostrando como o projeto presta uma contribuição muito importante para os objetivos da empresa.
Desempenho do banco de dados	Investigue a possibilidade de comprar um banco de dados com maior desempenho.
Prazo de desenvolvimento subestimado	Investigue a compra de componentes e verifique o uso de um gerador de programas.

3.*Planos de contingência* Seguir essas estratégias significa que, se o pior acontecer, você está preparado e tem uma estratégia pronta para lidar com o caso. Um exemplo de estratégia de contingência é a estratégia aplicada a problemas financeiros organizacionais, mostrada da Figura 4.14.

4.4.4 Monitoramento de riscos

O monitoramento de riscos envolve avaliar regularmente cada um dos riscos individuais, a fim de decidir se esse risco está se tomando mais ou menos provável e se seus efeitos decorrentes se modificaram. É claro que, em geral, isso não pode ser observado de modo direto; portanto, devem ser examinados outros fatores, para serem levantados indícios sobre a probabilidade de risco e seus efeitos. Esses fatores são, obviamente, dependentes dos tipos de risco. A Figura 4.15 apresenta alguns exemplos de fatores que podem ser úteis para avaliar esses tipos de risco.

Figura 4.15 Fatores de Risco

Tipo de risco	Indicadores em potencial
Tecnologia	Atraso na entrega de hardware ou software de apoio, muitos problemas de tecnologia são relatados.
Pessoal	Pessoal pouco motivado, relacionamento insatisfatório entre os membros da equipe, disponibilidade de trabalho.
Organizacional	Fofocas na empresa, falta de iniciativa por parte da alta gerência.
Ferramentas	Relutância de membros da equipe em utilizar ferramentas, reclamações sobre ferramentas CASE, solicitações de estações de trabalho com maior capacidade.
Requisitos	Muitos pedidos de modificações nos requisitos, reclamações do cliente.
Estimativa	Falha no cumprimento do programa estabelecido, falha em eliminar defeitos registrados.

ANÁLISE E GERÊNCIA DE REQUISITOS

O monitoramento de riscos deve ser um processo contínuo e, a cada análise de progresso feita pela gerência, cada um dos riscos principais deve ser considerado separadamente e discutido em uma reunião.

PONTOS-CHAVE

.Um bom gerenciamento de projeto de software é essencial para que os projetos de engenharia de software sejam desenvolvidos dentro do prazo e do orçamento.

.O gerenciamento de software é diferente dos outros gerenciamentos de engenharia. O software é intangível. Os projetos podem ser novos ou inovadores, de modo que não haja nenhuma experiência prévia para orientar seu gerenciamento. Os processos de software não são bem compreendidos.

.Os gerentes de software têm diversos papéis. Suas atividades mais significativas são: planejamento de projeto, estimativa e programação. O planejamento e a estimativa são processos iterativos, que continuam ao longo do projeto. À medida que mais informações se tornam disponíveis, os planos e as programações precisam ser revistos.

.Um marco de projeto é o resultado previsto de uma atividade em que algum relatório formal de progresso deve ser apresentado à gerência. Os marcos devem ocorrer regularmente ao longo de um projeto de software. Um produto a ser entregue é um marco que é disponibilizado ao cliente do projeto.

.A programação de projeto envolve a criação de várias representações gráficas de parte do plano de projeto. Isso inclui diagramas de atividades, que mostram o inter-relacionamento de atividades de projeto, e diagramas de barras, que mostram a duração das atividades.

.Os principais riscos de projeto devem ser identificados e avaliados, a fim de estabelecer sua probabilidade e suas conseqüências para o projeto. Para os riscos prováveis e potencialmente sérios, devem ser feitos planos preventivos, de gerenciamento de riscos, ou planos para administrar esse risco quando ele surgir. Os riscos devem ser explicitamente discutidos em cada reunião sobre o progresso do projeto.

Figura 4.16 Durações e Dependências de Tarefas.

Tarefa	Duração (dias)	Dependências
T1	10	
T2	15	T1
T3	10	T1, T2
T4	20	
T5	10	
T6	15	T3, T4
T7	20	T3
T8	35	T7
T9	15	T6
T10	5	T5, T9
T11	10	T9
T12	20	T10
T13	35	T3, T4
T14	10	T8, T9
T15	20	T12, T14
T16	10	T15

Exercícios

1 Explique por que a intangibilidade dos sistemas de software gera problemas especiais para o gerenciamento de projetos de software.

2 Explique por que os melhores programadores nem sempre se tornam os melhores gerentes de software.

3 Explique por que o processo de planejamento de projetos é iterativo e por que um plano deve ser revisto continuamente, durante um projeto de software.

4 Explique brevemente o propósito de cada uma das seções em um plano de projeto de software.

5 Qual é a principal distinção entre um marco e um produto a ser entregue?

6 A Figura 4.16 discrimina uma série de atividades, durações e dependências. Faça um diagrama de
atividades mostrando a programação de projeto.

7 A Figura 4.5 fornece a duração de tarefas para atividades de projeto de software. Suponha que um problema sério e não previsto ocorra e, em vez de levar dez dias para ser realizada, a tarefa T5 leve 45 dias. Revise a rede de atividades de acordo com essa ocorrência, evidenciando o novo caminho principal. Desenhe novos diagramas de barras mostrando como o projeto pode ser reorganizado.

8 Utilizando exemplos relatados de problemas com projetos, na literatura disponível, faça uma lista das dificuldades de gerenciamento que ocorreram nesses projetos de programação que fracassaram.

9 Além dos riscos mostrados na Figura 4.12, identifique seis outros riscos que, provavelmente, podem surgir em projetos de software.

10 Seu gerente pede que seja entregue um software dentro de uma programação que você sabe que somente será cumprida caso sua equipe trabalhe durante um período de horas extras, sem remuneração. Todos os membros da equipe têm filhos pequenos. Discuta se você deve aceitar essa solicitação de seu gerente e se deve persuadir a equipe a se dedicar desse modo à organização, em vez de se dedicar a seus familiares. Que fatores podem ser importantes em sua decisão?

11 Como programador, você é promovido ao cargo de gerente de projeto, mas você sente que pode prestar melhor contribuição em uma posição técnica do que em um cargo administrativo. Discuta se você deve aceitar essa promoção.

5. Gerenciamento de Requisitos - O quê e como Gerenciar

O gerenciamento de requisitos é um modelo sistemático para encontrar, documentar, organizar e rastrear os requisitos variáveis de um sistema.
*Gerência de Requisito*s estabelece as diretrizes para um comum entendimento entre o cliente e os requisitos de seu projeto de software, que serão conduzidos pela equipe do desenvolvimento.
Um requisito é definido como: Uma condição ou uma capacidade com a qual o sistema deve estar de acordo.

Nossa definição formal de gerenciamento de requisitos trata-se de um modelo sistemático para:

- Identificar, organizar e documentar os requisitos do sistema, e;
- Estabelecer e manter acordo entre o cliente e a equipe do projeto nos requisitos variáveis do sistema.

Os principais itens para o gerenciamento eficiente de requisitos incluem manter uma declaração clara dos requisitos, juntamente com atributos aplicáveis para cada tipo de requisito e rastreabilidade (rastreabilidade é a capacidade de rastrear um elemento do projeto a outros elementos correlatos, especialmente aqueles relacionados a requisitos. Os elementos do projeto envolvidos em rastreabilidade são chamados de itens de rastreabilidade. Os itens típicos de rastreabilidade incluem diferentes tipos de requisitos.

A finalidade de estabelecer **rastreabilidade** é ajudar a: compreender a origem dos requisitos, gerenciar o escopo do projeto, gerenciar mudanças nos requisitos, avaliar o impacto no projeto da mudança em um requisito, avaliar o impacto da falha de um teste nos requisitos (isto é, se o teste falhar, talvez o requisito não seja atendido), verificar se todos os requisitos do sistema são desempenhados pela implementação, verificar se o aplicativo faz apenas o que era esperado que ele fizesse) para outros requisitos e outros artefatos do projeto.

A coleta de requisitos pode parecer uma tarefa bem precisa. Nos projetos reais, contudo, você encontrará dificuldades porque:

- Nem sempre os requisitos são óbvios e podem vir de várias fontes.
- Nem sempre é fácil expressar os requisitos claramente em palavras.
- Existem diversos tipos de requisitos em diferentes níveis de detalhe.
- O número de requisitos poderá impossibilitar a gerência se não for controlado.
- Os requisitos estão relacionados uns com os outros, e também com o produto liberado do processo de engenharia do software.

- Os requisitos têm propriedades exclusivas ou valores de propriedade. Por exemplo, eles não são igualmente importantes nem igualmente fáceis de cumprir.
- Há várias partes interessadas, o que significa que os requisitos precisam ser gerenciados por grupos de pessoas de diferentes funções.
- Os requisitos são alterados.

Então, que habilidades você precisa desenvolver em sua organização para ajudá-lo a gerenciar essas dificuldades? Nós aprendemos que é importante dominar as seguintes habilidades:

5.1 Análise do Problema

A análise do problema é feita para compreender os problemas e as necessidades iniciais dos envolvidos, e propor soluções de alto nível. É um ato de ponderação e análise encontrar "o problema por trás do problema". Durante a análise do problema, são reconhecidos os problemas reais e quais são os envolvidos. Além disso, você define quais são, de uma perspectiva de negócios, as fronteiras da solução e as restrições de negócios da solução. Você também deverá ter analisado o caso de negócio para o projeto, para que haja uma boa compreensão de qual é o retorno esperado do investimento feito do sistema que está sendo construído.

5.2 Noções Básicas sobre as Necessidades dos Envolvidos

Os requisitos vêm de várias fontes, como clientes, parceiros, usuários finais e peritos do domínio. Você precisa saber o melhor modo de determinar quais devem ser as fontes, como obter acesso a essas fontes e qual a melhor forma de levantar as informações delas. Os indivíduos que constituem as fontes primárias para essas informações são conhecidos como os envolvidos no projeto. Se estiver desenvolvendo um sistema de informações para ser usado internamente na sua empresa, você deverá incluir na sua equipe de desenvolvimento pessoas com experiência de usuário final e com experiência no domínio do negócio. Com bastante freqüência, você começará os debates no nível de um modelo de negócio em vez de no nível de um sistema. Se estiver desenvolvendo um produto para ser vendido para um estabelecimento comercial, você deverá fazer uso extensivo do seu pessoal de marketing para entender melhor as necessidades dos clientes daquele mercado.

ANÁLISE E GERÊNCIA DE REQUISITOS

O levantamento de informações pode ocorrer através de técnicas como entrevistas, discussão de idéias, protótipos conceituais, questionários e análise competitiva. O resultado do levantamento seria uma lista das solicitações ou necessidades que foram descritas textual e graficamente, e que receberam prioridade uma em relação à outra.

5.3 Definição do Sistema

Definir o sistema significa traduzir e organizar as necessidades dos envolvidos em descrições significativas do sistema a ser construído. No início da definição do sistema, ocorre o seguinte: as decisões sobre o que constitui um requisito, o formato de documentação, a formalidade do idioma, o grau de especificidade dos requisitos (quantos e com que detalhe), a prioridade das solicitações e o esforço estimado (duas avaliações bem diferentes em geral atribuídas por pessoas diferentes em testes separados), os riscos técnicos e de gerenciamento, e o escopo inicial. Parte dessa atividade pode incluir modelos de design e protótipos iniciais diretamente relacionados aos mais importantes requisitos dos envolvidos. O resultado da definição do sistema é uma descrição do sistema que esteja em idioma natural e também seja gráfica.

5.4- Gerenciamento do Escopo de um Projeto

Para gerenciar com eficiência um projeto, é necessário priorizar os requisitos, com base em retorno dado por todos os envolvidos, e gerenciar o seu escopo. Vários projetos têm seus desenvolvedores trabalhando nos chamados "ovos de Páscoa" (características que o desenvolvedor acha interessantes e desafiadoras), em vez de estarem concentrados desde o início em tarefas que aliviam algum risco no projeto ou estabilizam a arquitetura do aplicativo.

Para assegurar que os riscos de um projeto sejam resolvidos ou aliviados o mais cedo possível, você deve desenvolver seu sistema de modo incremental, escolhendo cuidadosamente os requisitos para cada incremento que alivia os riscos conhecidos do projeto. Para fazê-lo, você precisa negociar o escopo (de cada iteração) com os envolvidos no projeto. Normalmente, isso requer boas habilidades no gerenciamento de expectativas dos resultados do projeto em suas diferentes fases. Você também precisa ter controle das origens dos requisitos, da aparência dos produtos liberados pelo projeto e do processo de desenvolvimento propriamente dito.

ANÁLISE E GERÊNCIA DE REQUISITOS

5.5 Refinamento da Definição do Sistema

A definição detalhada do sistema precisa ser apresentada de maneira que os envolvidos possam entendê-la, concordar com ela e sair dela. Ela precisa abordar não apenas a funcionalidade, mas também a compatibilidade com os requisitos legais ou reguladores, a usabilidade, a confiabilidade, o desempenho, a capacidade de suporte e de manutenção.

Um erro comum é acreditar que o que você sente é complexo para estabelecer necessidades que tenham uma definição complexa. Isso cria dificuldades para explicar a finalidade do projeto e do sistema. As pessoas podem ficar impressionadas, mas elas não darão bons retornos por falta de compreensão. Você deve se esforçar para compreender o público destinado aos documentos que você produz para descrever o sistema.

Você sempre poderá produzir vários tipos de descrição para públicos diferentes.
Já vimos que a metodologia do caso de uso, muitas vezes em combinação com protótipos visuais simples, é um modo bem eficiente de comunicar a finalidade do sistema e definir os detalhes do sistema. Os casos de uso ajudam a colocar os requisitos em um contexto, eles contam uma história de como o sistema será usado.

Outro componente da definição detalhada do sistema é estabelecer como o sistema deverá ser testado. Planos de teste e definições dos testes a serem realizados nos dizem quais capacidades do sistema serão verificadas.

5.6- Gerenciamento dos Requisitos Variáveis

Não importa o quão cuidadoso você seja sobre a definição dos seus requisitos, sempre haverá mudanças. O que torna complexo o gerenciamento dos requisitos variáveis não é apenas que um requisito mudado implicará mais ou menos tempo gasto na implementação de uma determinada característica nova, mas também que a mudança em um requisito terá impacto em outros requisitos.

Você precisa certificar-se de compor uma estrutura de requisitos que seja adaptável a mudanças, e de usar vínculos de rastreabilidade para representar as dependências entre os requisitos e outros artefatos do ciclo de vida do desenvolvimento. O gerenciamento de mudança inclui atividades como estabelecer uma linha de base, determinar quais dependências são importantes de serem rastreadas, estabelecer a rastreabilidade entre itens correlatos e o controle de mudança.

6- Gerenciamento de Requisitos baseado em Use Cases

6.1- Documentação associada ao modelo de casos de uso

O modelo de casos de uso captura os requisitos funcionais e força o desenvolvedor a pensar em como os agentes externos interagem como o sistema. No entanto, esse modelo corresponde somente aos requisitos funcionais. Outros tipos de requisitos (desempenho, interface, segurança, regras do negócio etc.) que fazem parte do *documento de requisitos* de um sistema não são considerados pelo modelo de casos de uso. Está fora do escopo deste texto introdutório dar uma descrição detalhada sobre como documentar todos os possíveis tipos de requisitos de um sistema.

Entretanto, esta seção descreve como alguns dos demais itens da especificação de requisitos podem estar relacionados com o modelo de casos de uso. Os itens aqui considerados são os seguintes:

- .Regras do negócio
- .Requisitos de interface
- .Requisitos de desempenho

6.2 Regras do negócio

Regras do negócio são políticas, condições ou restrições que devem ser consideradas na execução dos processos existentes em uma organização (Gottesdiener,1999). As regras do negócio constituem uma parte importante dos processos organizacionais porque descrevem a maneira como a organização funciona. O termo "regra de negócio" é utilizado mesmo em organizações não caracterizadas como empresariais.

Cada organização pode ter várias regras do negócio. As regras do negócio de uma organização são normalmente identificadas nas fases de levantamento de requisitos de análise. Essas regras são documentadas no chamado *modelo de regras do negócio.* Nesse modelo, as regras são categorizadas. Cada regra normal mente recebe um identificador (assim como acontece para cada caso de uso). Esse identificador permite que a regra seja facilmente referenciada nos demais artefatos do processo de desenvolvimento.

A descrição do modelo de regras do negócio pode ser feita utilizando-se texto informal, ou alguma forma de estruturação. Alguns exemplos de regras do negócio (não pertencentes a uma mesma organização) são apresentados aqui:

.O *valor total de um pedido* é *igual à soma dos totais dos itens do pedido acrescido de 10% de taxa de entrega.*

.Um professor só pode estar lecionando disciplinas para as quais esteja habilitado.
.Um cliente do banco não pode retirar mais de R$ *1.000 por dia de sua conta.* .Os *pedidos para um cliente não-especial devem ser pagos antecipadamente. .Para alugar um carro,* o *proponente deve estar com a carteira de motorista válida.*

.O *número máximo de alunos por turma* é *igual a 30.*
.Um aluno deve ter a matricula cancelada se *obtiver dois conceitos* D no *curso.*
.Uma vez que um professor confirma as notas de uma turma, estas não podem ser modificadas.

.*Senhas devem ter,* no *mínimo,* seis *caracteres, entre números* e *letras,* e *devem ser atualizadas a cada três meses.*

As regras do negócio normalmente têm influência sobre a lógica de execução de um ou mais casos de uso. Por exemplo, considere a última regra ilustrada anteriormente. Essa regra implica em que deve haver uma maneira de informar ao usuário quando uma atualização é necessária e um modo pelo qual o usuário possa atualizar a sua senha, o que tem influência no modelo de casos de uso do sistema.

Para conectar uma regra a um caso de uso no qual ela é relevante deve ser utilizado o identificador da regra do negócio que influenciar no caso de uso em questão.

Para finalizar a discussão sobre regras do negócio, a Tabela 6.1 fornece um formulário que pode ser utilizado para construir o modelo de regras do negócio.

Este formulário apresenta o nome da regra de negócio, o seu identificador, a descrição da regra, a fonte de informação que permitiu definir a regra (normalmente um especialista do domínio do negócio) e um histórico de evolução da regra (por exemplo, data de identificação, data de última atualização etc.).

Tabela 6.1 Possível formato para documentação de uma regra de negócio.

Nome	**Quantidade de inscrições possíveis (RN01)**
Descrição	Um aluno não pode se inscrever em mais de seis disciplinas por semestre letivo.
Fonte	Coordenador de escola de informática.
Histórico	Data de identificação: 12/07/2002.

6.3 Requisitos de desempenho

O modelo de casos de uso também não considera *requisitos de desempenho.* Um requisito de desempenho define características relacionadas à operação do sistema. Exemplos: número esperado de transações por unidade de tempo, tempo máximo esperado para uma operação, volume de dados que deve ser tratado etc. Alistair Cockburn recomenda em seu livro (Cockburn, 2000) utilizar uma tabela para ilustrar os requisitos de desempenho e suas associações com os casos de uso do sistema.

Na Tabela 6.2 apresentam-se os identificadores de casos de uso na primeira coluna e requisitos de desempenho nas demais colunas.

Tabela 6.2 conectando casos de uso a requisitos de desempenho.

Identificador do caso de uso	**Freqüência da utilização**	**Tempo máximo esperado**
CSU01	5/mês	Interativo
CSU02	15/dia	1 segundo
CSU03	60/dia	Interativo
CSU04	180/dia	3 segundos
CSU05	600/mês	10 segundos
CSU07	500/dia durante 10 dias seguidos	10 segundos

6.4 Requisitos de interface

A especificação dos requisitos de um sistema pode também conter uma seção que descreva os requisitos de interface do sistema. Por exemplo, o cliente pode ter definido restrições específicas com respeito à interface do sistema: cor, estilo, interatividade etc.
Os requisitos de interface podem estar relacionados a um ou mais casos de uso do sistema. Esse relacionamento pode ser feito de uma forma semelhante à da Tabela 6.2.

6.5 Modelo de casos de uso no processo de desenvolvimento

Considerando que o processo de desenvolvimento incremental é utilizado, a identificação da maioria dos atores e casos de uso é feita pelos analistas na fase de concepção. A descrição dos casos de uso considerados mais críticos começa - já nessa fase, que termina com 10% a 20% do modelo de casos de uso completo.

ANÁLISE E GERÊNCIA DE REQUISITOS

Na fase de elaboração, a construção do modelo continua de tal forma que, ao seu término, 80% do modelo de casos de uso esteja construído. Note que a descrição dos casos de uso feita até este momento tem um nível de abstração *essencial*.

Na fase de construção, casos de uso formam uma base natural através da qual podem-se realizar as iterações do desenvolvimento. Um grupo de casos é alocado a cada interação.

Então, o desenvolvimento do sistema segue a alocação realizada: em cada iteração, um grupo de casos de uso é detalhado (utilizando um nível de abstração *real)* e desenvolvido. O processo continua até que todos os casos de uso tenham sido desenvolvidos e o sistema esteja completamente construído. Esse tipo de desenvolvimento é também chamado de desenvolvi*mento dirigido a casos de uso.* Conforme mencionado há pouco nesta seção, um fator importante para o sucesso do desenvolvimento do sistema é considerar os casos de uso mais importantes primeiramente. Murray Cantor (Cantor, 1998) propõe uma classificação dos casos de uso identificados para um sistema em função de dois parâmetros: *risco de desenvolvimento* e *prioridades estabelecidas pelo usuário.* Dessa forma, cada caso de uso se encaixa em uma das categorias a seguir:

> **1**. **Risco *alto* e *prioridade alta***: casos de uso nesta categoria são os mais críticos. Devem ser considerados o quanto antes.
>
> **2. Risco *alto* e *prioridade baixa***: embora os casos de uso nesta categoria tenham risco alto, é necessário, antes de começar a considerá-los, negociar com o cliente em relação a sua verdadeira necessidade.
>
> **3**. **Risco *baixo* e *prioridade alta:*** embora os casos de uso tenham prioridade alta, é necessário ter em mente que os casos de uso de mais alto risco devem ser considerados primeiro.
>
> **4. Risco *baixo* e *prioridade baixa***: em situações em que o desenvolvimento do sistema está atrasado, estes casos de uso são os primeiros a serem "cortados".

Através da atribuição de importância segundo a categorização de Cantor, no caso de uso não tão importante não será contemplado nas iterações iniciais.

Se o requisito correspondente a esse caso de uso for modificado ou não mais precisar ser considerado, os analistas não terão desperdiçado tempo com ele.

Note também que a descrição expandida de um determinado caso de uso é deixada para a iteração na qual este deve ser implementado. Isso evita que se perca tempo inicialmente no seu detalhamento. Além disso, essa estratégia é ais adaptável aos *requisitos voláteis.*

Se todos os casos de uso forem detalhados inicialmente, e se um ou mais requisitos são modificados durante o desenvolvimento, toda a modelagem correspondente a esses casos de uso sofrerá modificações. Por outro lado, se a descrição detalhada é deixada para a iteração à qual o ISO de uso foi alocado, uma eventual mudança dos requisitos associados a esse ISO de uso não afetará tão profundamente o desenvolvimento.

A construção do modelo de casos de uso deve se adequar ao processo de desenvolvimento sendo utilizado. Os casos de uso mais arriscados devem ser considerados primeiramente.

6.6 Casos de uso nas atividades de análise e projeto

Alguns desenvolvedores escrevem as descrições iniciais de casos de uso mencionando detalhes de interface gráfica com o usuário (considerando atores humanos).

A justificativa é que a narrativa dos casos de uso segundo essa abordagem fornece uma idéia mais concreta de como se apresentará uma determinada funcionalidade do sistema. Contudo, as desvantagens dessa abordagem se sobrepõem às vantagens.

Considere a situação de uma parte da interface estar sendo continuamente modificada, por alguma razão.

Nessa situação, a desvantagem de utilização de casos de uso reais se toma nítida, pois o fato de a interface ser modificada possivelmente resultará na modificação da narrativa do caso de uso.

Além disso, lembre-se do objetivo principal do modelo de casos de uso, a saber, modelar os requisitos do sistema.

Portanto, casos de uso devem ser independentes do desenho da interface pelo fato de que os requisitos do sistema não devem estar associados a detalhes de interface.

Uma melhor abordagem é utilizar inicialmente casos de uso essenciais para não acoplar os detalhes da interface da aplicação na especificação narrativa das interações de um caso de uso.

Nessa especificação, a atenção do modelador deve recair sobre a essência das interações entre atores e o sistema, em vez de como cada interação é realizada fisicamente.

Especificações de casos de uso feitas dessa forma ficam mais imunes a futuras mudanças na interface com o usuário, além de permitir que o analista de sistemas se concentre no que é realmente importante em uma narrativa de caso de uso: as interações entre ator(es) e sistema. Por exemplo, considere o termo "envia uma requisição" em contraposição com o termo "duplo clique sobre o botão de envio de requisições".

Em resumo, casos de uso que mencionam detalhes de interface gráfica são indesejáveis durante a análise. O mais adequado é utilizar casos de uso essenciais e, posteriormente, na etapa de projeto, transformá-los em casos de uso reais adicionando mais detalhes.

Na fase de análise, descrições de casos de uso devem capturar os requisitos funcionais do sistema e ignorar aspectos de projeto, como a interface gráfica com o usuário.

Pode-se, agora, descrever um procedimento a ser utilizado na construção do modelo de casos de uso quando da utilização de um processo de desenvolvimento incremental:

1. Identifique os atores e casos de uso na fase de concepção. Alguns atores e casos de uso só serão identificados posteriormente, mas a grande maioria deve ser descoberta nesta fase.

2. Na fase de elaboração:

a. Desenhe o(s) diagrama(s) de casos de uso;

b. Escreva os casos de uso em um formato de alto nível e essencial.

c. Ordene a lista de casos de uso de acordo com prioridade e risco. Cada partição corresponde a um grupo de casos de uso que será implementado em um dos ciclos de desenvolvimento do sistema.

3. Associe cada grupo de casos de uso a uma iteração da fase de construção. Os grupos mais prioritários e arriscados devem ser alocados às iterações iniciais.

4. Na i-ésima iteração da fase de construção:

a. Detalhe os casos de uso do grupo associado a esta iteração (se necessário, utilize o nível de abstração real).
b. Implemente estes casos de uso.

6.7 Casos de uso e outras atividades do desenvolvimento

O modelo de casos de uso direciona a realização de várias outras atividades do

desenvolvimento.

6.8 Planejamento e gerenciamento do projeto

O modelo de casos de uso é uma ferramenta fundamental para o gerente de um projeto no planejamento e controle de um processo de desenvolvimento incremental e iterativo. Ao final de cada iteração, o gerente pode avaliar a produtividade na realização das tarefas. Essa avaliação serve como massa de dados para que esse profissional realize a alocação das tarefas e recursos para as próximas interações.

6.9 Testes do sistema

Em um processo de desenvolvimento incremental e iterativo, não há uma fase I de testes propriamente dita. Ao contrário, os testes do software são realizados II continuamente durante todo o desenvolvimento. Os profissionais responsáveis pelos testes utilizam o modelo de casos de uso pata planejar as atividades de teste. Os casos de uso e seus cenários oferecem *casos de teste.* Quando o sistema está sendo testado, os cenários sobre o sistema podem ser verificados para identificar a existência de erros.

6.10 Documentação do usuário

Os manuais e guias do usuário também podem ser construídos com base no modelo de casos de uso. Na verdade, se o modelo de casos de uso foi bem construído, deve haver uma correspondência clara entre cada caso de uso do sistema e uma seção do manual do usuário. Isso porque esse modelo está baseado na noção de que o sistema é construído para se adequar à perspectiva de seus usuários.

6.11 Estudo de caso

A partir desta seção, um estudo de caso começa a ser desenvolvido. Esse estudo tem o objetivo de consolidar os principais conceitos teóricos descritos e oferecer uma visão prática sobre como os modelos apresentados são desenvolvidos.

O desenvolvimento do estudo de caso é feito de forma incremental: em cada capítulo deste livro em que houver uma seção denominada "Estudo de caso", uma parte do desenvolvimento é apresentada.
É importante notar que, para manter a descrição em um nível de simplicidade e clareza aceitáveis para um livro didático, muitos detalhes do desenvolvimento são deliberadamente ignorados.

6.12 Descrição da situação

O estudo de caso é sobre uma faculdade que precisa de uma aplicação para controlar alguns processos acadêmicos, como inscrições em disciplinas, lançamento de notas, alocação de recursos para turmas etc.

Após o levantamento de requisitos inicial desse sistema, os analistas chegaram à seguinte lista de requisitos funcionais:

R1. O sistema deve permitir que alunos visualizem as notas obtidas por se mestre letivo.

R2. O sistema deve permitir o lançamento das notas das disciplinas lecionadas em um semestre letivo e controlar os prazos e atrasos neste lançamento.

R3. O sistema deve manter informações cadastrais sobre disciplinas no currículo escolar.

R4. O sistema deve permitir a abertura de turmas para uma disciplina, assim como a definição de salas e laboratórios a serem utilizadas e dos horários e dias da semana em que haverá aulas de tal turma.

R5. O sistema deve permitir que os alunos realizem a inscrição em disciplinas de um semestre letivo.

R6. O sistema deve permitir o controle do andamento das inscrições em disciplinas feitas por alunos.

R7. O sistema deve se comunicar com o *Sistema de Recursos Humanos* para obter dados cadastrais sobre os professores.

R8. O sistema deve se comunicar com o *Sistema de Faturamento* para informar as inscrições realizadas pelos alunos.

R9. O sistema deve manter informações cadastrais sobre os alunos e sobre seus históricos escolares.

6.13 Regras do negócio

Algumas regras do negócio iniciais também foram identificadas para o sistema. Essas regras são descritas a seguir:

Quantidade máxima de inscrições por semestre letivo (RNOl)
Descrição: Em um *semestre* letivo, um *aluno não pode* se inscrever em *uma quantidade de disciplinas cuja soma de créditos ultrapasse 20.*

Quantidade de alunos possíveis (RNO2)

Descrição: Uma oferta de disciplina não pode ter mais de 40 alunos inscritos.
Pré-requisitos para uma disciplina (RN03)
Descrição: Um aluno não pode se inscrever em uma disciplina para a qual não possua os pré-requisitos necessários.

Habilitação para lecionar disciplina (RN04)
Descrição Um professor só pode estar lecionando disciplinas para as quais esteja habilitado.

Cancelamento de matrícula (RN05)
Descrição: *Um aluno* deve *ter a matricula cancelada* se *for reprovado mais de duas vezes na mesma disciplina.*

Política de Avaliação de Alunos (RN06)
Descrição: A nota de um aluno em uma disciplina (um valor de O a 10) é obtida pela média de duas avaliações durante o semestre, A1 e A2, ou pela freqüência nas aulas.
Se o aluno obtém nota maior ou igual a 7.0 (sete), está aprovado. Se o aluno obtém nota maior ou igual 5.0 (cinco) e menor que 7.0 (sete), deve fazer a avaliação final.
.Se o aluno obtém nota menor que 5.0 (cinco) está reprovado.
.Se o aluno tiver uma freqüência menor que 75% em uma turma, está automaticamente reprovado.

6.14 Documentação do modelo de casos de uso

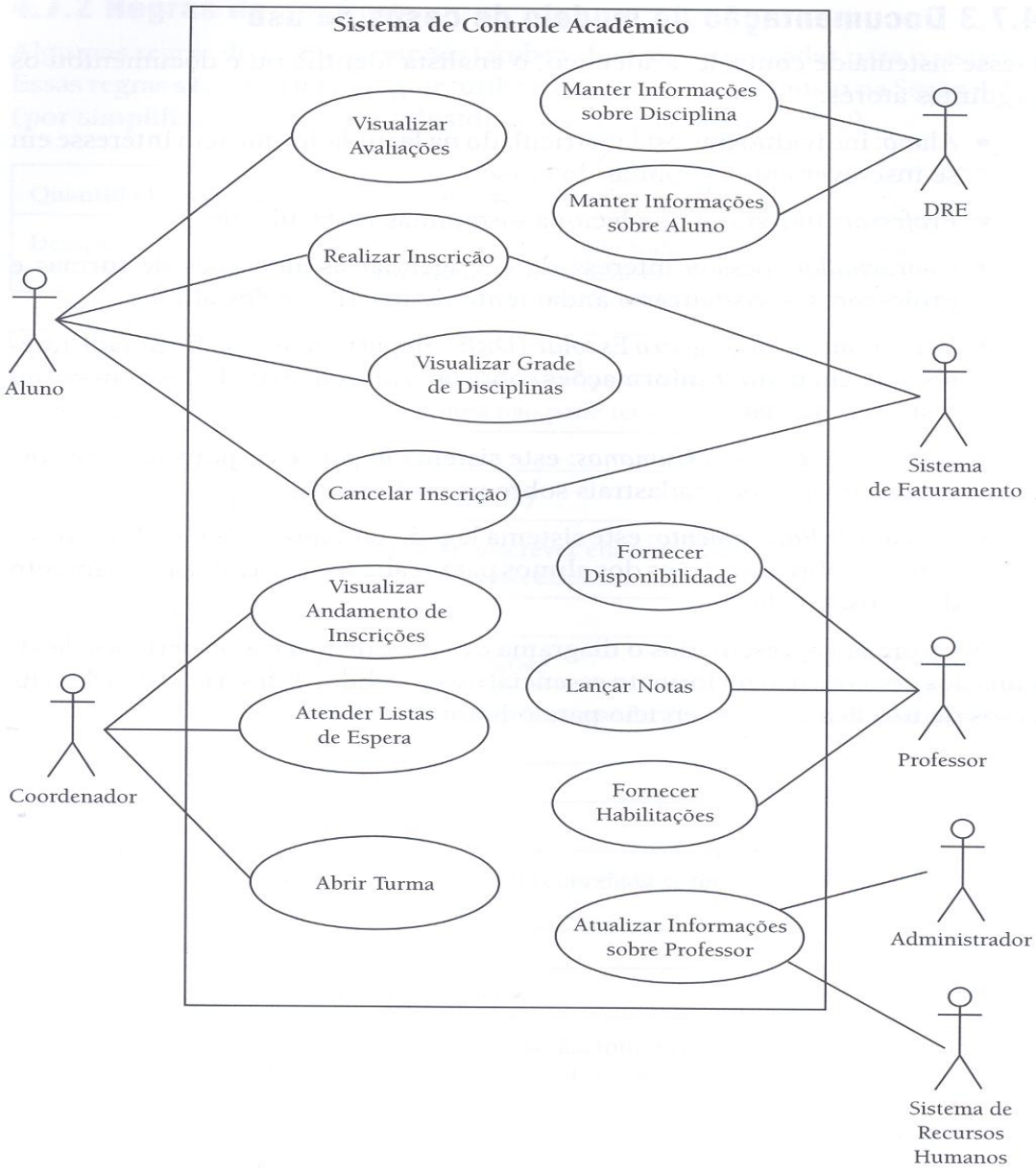
Nesse sistema de controle acadêmico, o analista identificou e documentou os seguintes atores:

.Aluno: indivíduo que está matriculado na faculdade, que tem interesse em se inscrever em disciplinas do curso.
.***Professor****:* indivíduo que leciona disciplinas na faculdade.
Coordenado*r:* pessoa interessada em agenda r as alocações de turmas e professores, e visualizar o andamento de inscrições dos alunos.
Departamento de Registro Escolar (DRE): departamento da faculdade interessado em manter informações sobre os alunos matriculados e sobre seu histórico escolar.
Sistema de Recursos Humanos*:* este sistema legado é responsável por fornecer informações cadastrais sobre os professores.
Sistema de Faturamento*:* este sistema legado tem interesse em obter informações sobre inscrições dos alunos para realizar o controle de pagamento de mensalidades.

ANÁLISE E GERÊNCIA DE REQUISITOS

A seguir são apresentados o diagrama de casos de uso e as descrições de alguns dos casos de uso no formato essencial e expandido. A descrição dos demais casos de uso fica como exercício para o leitor.

Diagrama de Casos de Uso para o SCA.



Realizar Inscrição (CSUO1)

Sumário: Aluno usa o sistema para realizar inscrição em disciplinas. Ator Primário: Aluno

Atores Secundários: Sistema de Faturamento

Precondições: O Aluno está identificado pelo sistema. Fluxo Principal

1. O Aluno solicita a realização de inscrição.

2: O sistema apresenta as disciplinas disponíveis para o semestre corrente e para as quais o aluno tem pré-requisitos.

3. O Aluno seleciona as disciplinas desejadas e as submete para inscrição.

4. Para cada disciplina selecionada, o sistema aloca o aluno em uma turma que apresente uma oferta para tal disciplina.

5. O sistema informa as turmas nas quais o Aluno foi alocado. Para cada alocação, o sistema informa o professor, os horários e os respectivos locais das aulas de cada disciplina.

6. O Aluno confere as informações fornecidas.

7. O sistema envia os dados sobre a inscrição do aluno para o Sistema de Faturamento e o caso de uso termina.

Fluxo Alternativo (4): Inclusão em lista de espera

a. Se não há oferta disponível para alguma disciplina selecionada pelo aluno, o sistema reporta o fato e fornece a possibilidade de inserir o Aluno em uma lista de espera.

b. Se o Aluno aceitar, o sistema o insere na lista de espera e apresenta a posição na qual o aluno foi inserido na lista. O caso de uso retoma ao passo 4.

c. Se o Aluno não aceitar, o caso de uso prossegue a partir do passo 4.

Fluxo de Exceção (4): Violação de RNO1

a. Se o Aluno atingiu a quantidade máxima de inscrições (RNO1), o sistema informa ao aluno a quantidade de disciplinas que ele pode selecionar, e o caso de uso retoma ao passo 2.

Pós-condições: O aluno foi inscrito em uma das turmas de cada uma das disciplinas desejadas, ou foi adicionado a uma ou mais listas de espera.

Regras de Negócio: RNO1, RNO2, RNO3

Visualizar Avaliações (CSUO2)

Sumário: Aluno visualiza avaliação que recebeu (notas e freqüência) nas turmas de um semestre letivo.

Ator Primário: Aluno

Precondições: O Aluno está identificado pelo sistema. Fluxo Principal

1. O Aluno solicita a visualização das avaliações para as ofertas de disciplina em que participou.

2. O sistema exibe os semestres letivos nos quais o Aluno se inscreveu em pelo menos uma oferta de disciplina.

3. O Aluno seleciona os semestres letivos cujas avaliações deseja visualizar.

4. O sistema exibe uma lista de avaliações agrupadas por semestres letivos selecionados e por turma.

5. O aluno visualiza as avaliações e o caso de uso termina.

Fluxo de Exceção (2): Aluno sem inscrição

a. Não há semestre letivo no qual o Aluno tenha participado em alguma oferta de disciplina: o sistema reporta o fato e o caso de uso termina.

Pós-condições: O Aluno obteve as avaliações que desejava visualizar.

ANÁLISE E GERÊNCIA DE REQUISITOS

Fornecer Disponibilidades (CSUO3)

Sumário: Professor fornece a sua grade de disponibilidade (dias e horários) e disciplinas que deseja lecionar no próximo semestre letivo.

Ator Primário: Professor

Pré-condições: O Professor está identificado pelo sistema. Fluxo Principal

1. O Professor indica o desejo de fornecer sua disponibilidade para o próximo semestre letivo.

2. O sistema apresenta a lista de disciplinas disponíveis para as quais o professor está habilitado e uma grade de dias e horários da semana em branco. 3.0 Professor seleciona as disciplinas que deseja lecionar. 4. O Professor preenche a grade com sua disponibilidade. 5. O sistema registra as informações fornecidas.

Fluxo Alternativo (3): Modificação na grade atual

a. O Professor solicita que o sistema apresente a mesma configuração de disponibilidade do semestre atual.

b. O sistema apresenta a configuração requisitada.

c. O professor realiza as modificações que deseja na configuração.

d. O sistema registra a informação e o caso de uso termina.

Fluxo de Exceção (4): Disciplinas não selecionadas

a. Se o Professor não selecionou disciplina alguma ou não preencheu a grade de disponibilidade: o sistema reporta o fato e o caso de uso continua a partir do passo 2.

Pós-condições: o sistema registrou a disponibilidade do Professor para o próximo semestre letivo.

ANÁLISE E GERÊNCIA DE REQUISITOS

Lançar Notas (CSUO4)

Sumário: Professor realiza o lançamento de notas das ofertas de disciplinas lecionadas por ele no semestre corrente.

Ator Primário: Professor

Precondições: O Professor está identificado pelo sistema. Fluxo Principal

1. O Professor solicita o lançamento de notas.

2. O sistema exibe a lista de turmas e disciplinas correspondentes do semestre corrente nas quais o Professor lecionou.

3. O Professor seleciona a turma e, dentro desta, a oferta de disciplina para a qual deseja realizar o lançamento de notas.

4. O sistema exibe a lista de alunos da oferta de disciplina selecionada e requisita a primeira nota (AI), a segunda nota (A2) e a quantidade de faltas para cada aluno. S. O Professor fornece as notas de AI e de A2 e a quantidade de faltas de cada aluno.

6. O sistema exibe o resultado da avaliação de cada aluno, conforme regra de negócio RNO6, para verificação pelo Professor.

7. O Professor confere os dados e confirma a avaliação.

8. O sistema registra as avaliações e o caso de uso termina. Fluxo Alternativo (7): Erro no lançamento

a. O professor detecta que lançou uma avaliação errada para algum aluno.

b. O professor corrige a avaliação do aluno.

c. O sistema aceita a correção e o caso de uso continua a partir do passo 7.

Fluxos de Exceção (4): Avaliação em branco ou errada

a. Se o Professor não fornece alguma nota, ou freqüência, ou fornece dados inválidos: o sistema reporta o fato e o caso de uso retoma ao passo 4.

Pós-condições: as notas de uma ou mais disciplinas lecionadas pelo professor foram lançadas no sistema.

Regras de Negócio: RNO6

ANÁLISE E GERÊNCIA DE REQUISITOS

Manter Informações sobre Disciplina (CSUO5)

Sumário: DRE realiza o cadastro (inclusão, remoção, alteração e consulta) dos dados sobre disciplinas.

Ator Primário: DRE

Precondições: O Professor está identificado pelo sistema. Fluxo Principal

1. DRE requisita a manutenção de disciplinas.
2. O sistema apresenta as operações que podem ser realizadas: a inclusão de uma nova disciplina, a alteração dos dados de uma disciplina, a exclusão de uma disciplina e a consulta pelos dados de uma disciplina.
3. O DRE indica a opção a realizar ou opta por finalizar o caso de uso.
4. O DRE seleciona a operação desejada: Inclusão, Exclusão, Alteração ou Consulta.
5. Se o DRE deseja continuar com a manutenção, o caso de uso retoma ao passo 2; caso contrário, o caso de uso termina.

Fluxo Alternativo (4): Inclusão

a. O DRE requisita a inclusão de uma disciplina.

b. O sistema apresenta um formulário em branco para que os detalhes da disciplina (código, nome e quantidade de créditos) sejam incluídos.

c. O DRE fornece os detalhes da nova disciplina.

d. O sistema apresenta uma lista de disciplinas para que o DRE selecione as que são pré-requisitos para a disciplina a ser criada.

e. O DRE define zero ou mais disciplinas como pré-requisitos.

f. O sistema verifica a validade dos dados. Se os dados forem válidos, inclui a nova disciplina; caso contrário, o sistema reporta o fato, solicita novos dados e repete a verificação.

Fluxo Alternativo (4): Remoção

a. O DRE seleciona uma disciplina e requisita ao sistema que a remova.

b. Se a disciplina pode ser removida, o sistema realiza a remoção; caso contrário, o sistema reporta o fato.

Fluxo Alternativo (4): Alteração

a. O DRE altera um ou mais dos detalhes sobre uma disciplina e requisita a sua atualização.

b. O sistema verifica a validade dos dados e, se eles forem válidos, altera os dados na lista de disciplinas da faculdade.

Fluxo Alternativo (4): Consulta

a. O DRE solicita a realização de uma consulta sobre a lista de disciplinas.

b. O sistema apresenta uma lista com os códigos de todas as disciplinas, permitindo que o usuário selecione a disciplina desejada.

c. O DRE seleciona uma disciplina.

d. O sistema apresenta os detalhes da disciplina e seus pré-requisitos (se existirem) no formulário de disciplinas.

Pós-condições: uma disciplina foi inserida ou removida, ou seus detalhes foram alterados.

Visualizar Andamento de Inscrições (CSUO6)

Sumário: O Coordenador usa o sistema para visualizar o andamento de inscrições sendo realizadas pelos alunos em disciplinas ofertadas para o próximo semestre letivo.

Ator Primário: Coordenador

Precondições: O Coordenador está identificado pelo sistema. Fluxo Principal

1. O Coordenador solicita a visualização do andamento de inscrições.
2. O sistema exibe a lista de disciplinas para as quais existe pelo menos uma oferta para o próximo semestre.
3. O Coordenador seleciona a disciplina para a qual deseja visualizar o andamento de inscrições.
4. O sistema exibe a lista de turmas nas quais existe oferta para a disciplina selecionada, juntamente com as alocações correspondentes: professor, horários, locais e situação (aberta ou fechada).
5. O Coordenador seleciona uma das turmas.
6. O sistema apresenta a lista de alunos inscritos na turma para a disciplina selecionada, ordenados por data de inscrição.
7. O Coordenador visualiza as informações.
8. Se o Coordenador deseja continuar a visualização., o caso de uso retoma ao passo 5; caso contrário, o caso de uso termina.

Atualizar Informações sobre Professor (CSUO7)

Sumário: Administrador do sistema usa o sistema para atualizar as informações cadastrais sobre professores a partir do SRH.

Ator Primário: Administrador

Ator Secundário: Sistema de Recursos Humanos (SRH)

Precondições: O Administrador está identificado pelo sistema. Fluxo Principal

1. O Administrador solicita ao sistema que obtenha os dados atualizados sobre professores.
2. O sistema se comunica com o SRH e obtém os dados a partir deste.
3. O sistema apresenta os dados obtidos e solicita a confirmação do Administrador para realizar a atualização.
4. O Administrador confirma a atualização.
5. O sistema atualiza os dados cadastrais dos professores.

Fluxo de Exceção (2): Houve uma falha na obtenção de dados

a. O sistema não consegue obter os dados a partir do SRH.

b. O sistema reporta o fato e o caso de uso termina.

Fluxo Alternativo (4): Desistência de atualização

O Administrador declina da atualização e o caso de uso termina.

Atender Listas de Espera (CSUO8)
Sumário: Coordenador atende às demandas representadas pelas listas de espera por vagas para uma disciplina.

Ator Primário: Coordenador

Atores Secundários: Sistema de Faturamento

Precondições: O Coordenador está identificado pelo sistema. Fluxo Principal
1. O sistema apresenta as listas de espera existentes para as disciplinas.
2. O Coordenador seleciona uma das listas de espera.
3. O sistema apresenta os detalhes da lista de espera selecionada (data de criação e quantidade de alunos).
4. O Coordenador fornece os dias e respectivos horários da oferta de disciplina que deseja criar.
5. O sistema apresenta as salas e professores disponíveis nos dias e horários fornecidos pelo Coordenador.
6. O Coordenador seleciona um professor e uma ou mais salas.
7. O Coordenador fornece a quantidade de alunos a serem alocados na nova oferta de disciplina.
8. A partir de uma lista fornecida pelo sistema, o Coordenador seleciona a turma na qual deseja alocar a oferta de disciplina.
9. O sistema cria a oferta de disciplina e transfere a quantidade de alunos fornecida no passo 7 da lista de espera para a oferta recém-criada, de acordo com a ordem dos alunos nessa lista.
10. Se o Coordenador deseja continuar o atendimento das listas de espera, o caso de uso retoma ao passo 1; caso contrário, o sistema envia os dados de inscrições de alunos para o sistema de faturamento e o caso de uso termina.

Fluxo de Exceção(7): Violação de RN02
a. Se a quantidade de alunos que o Coordenador fornecer for inválida (violar a regra do negócio RN02, ou for maior que o tamanho da lista de espera), o sistema reporta o fato e solicita um novo valor.
b. O Coordenador corrige o valor e o caso de uso prossegue a partir do passo 8.

Pós-condições:
Zero ou mais ofertas de disciplinas foram criadas. Zero ou mais listas de espera foram atendidas.

Regras de Negócio: RN02

7. Prototipação de software

Objetivos

Explicar como a prototipação de software é utilizada no processo de software e descrever diferentes abordagens para o desenvolvimento de protótipos. Depois de ler este capítulo, você poderá:

- compreender o papel da prototipação em diferentes tipos de projetos de desenvolvimento;
- compreender a diferença entre a prototipação evolucionária e a prototipação descartável;
- conhecer três diferentes técnicas de prototipação, que são: o desenvolvimento de linguagem de nível muito elevado, a programação de banco de dados e o reuso de componentes e aplicações, e
- compreender por que a prototipação é a única técnica viável para o projeto e o desenvolvimento de interface com o usuário.

Os clientes e os usuários finais de software acham muito difícil expressar seus reais requisitos. É quase impossível prever como um sistema afetará práticas de trabalho, como interagirá com outros sistemas e que operações dos usuários devem ser auto matizadas.

A análise cuidadosa e as revisões sistemáticas de requisitos ajudam a reduzir as incertezas sobre o que o sistema deve fazer.

Contudo, não há um substituto real para experimentar um requisito antes de concordar com ele. Isso será possível se um protótipo de sistema estiver disponível.

Um protótipo é uma versão inicial de um sistema de software, que é utilizada para mostrar conceitos, experimentar opções de projeto e, em geral, para conhecer mais sobre os problemas e suas possíveis soluções.

O desenvolvimento rápido de um protótipo é essencial para que os custos sejam controlados e os usuários possam fazer experiências com o protótipo no início do processo de software.

ANÁLISE E GERÊNCIA DE REQUISITOS

Um protótipo de software apóia duas atividades do processo de engenharia de requisitos:

> **1. *Levantamento de requisitos*** Os protótipos de sistema permitem que os usuários realizem experiências para ver como o sistema apóia seu trabalho. Eles obtêm novas idéias para os requisitos e podem identificar pontos positivos e negativos do software. Eles podem, então, propor novos requisitos de sistema.
>
> **2. *Validação de requisitos*** O protótipo pode revelar erros e omissões nos requisitos propostos. Uma função descrita em uma especificação pode parecer útil e bem-definida. Contudo, quando essa função é utilizada com outras, os usuários muitas vezes acham que sua visão inicial era incorreta e incompleta. A especificação de sistema pode então ser modificada para refletir sua compreensão alterada dos requisitos.

A prototipação pode ser utilizada como uma técnica de análise e redução de riscos. Um risco significativo no desenvolvimento de software são os erros e as omissões de requisitos.

Os custos da correção de erros de requisitos em fases posteriores do processo podem ser muito altos. As experiências mostraram (Boehm *et al.,* 1984) que a prototipação reduz o número de problemas com a especificação de requisitos. Além disso, os custos totais de desenvolvimento poderão ser mais baixos se um protótipo for desenvolvido.

Portanto, a prototipação é parte do processo de engenharia de requisitos.

Contudo, a distinção entre a prototipação como uma atividade separada e o desenvolvimento massivo de software ficou obscurecida nos últimos anos.

Muitos sistemas são, atualmente, desenvolvida com o uso de uma abordagem evolucionária, em que uma versão inicial é criada rapidamente e modificada para produzir um sistema final.

Um modelo de processo iterativo, como o desenvolvimento incremental, pode ser utilizado em conjunto com uma linguagem projetada para o rápido desenvolvimento de aplicações.

Portanto, as técnicas utilizadas para desenvolver um protótipo para a validação de requisitos podem também ser utilizadas para desenvolver o próprio sistema de software.

ANÁLISE E GERÊNCIA DE REQUISITOS

Além de permitir que os usuários melhorem a especificação de requisitos, desenvolver um protótipo de sistema pode ter outros benefícios:

1. Possíveis equívocos entre desenvolvedores de software e usuários podem ser identificados à medida que as funções do sistema são apresentadas.

2. A equipe de desenvolvimento de software pode encontrar requisitos incompletos e/ou inconsistentes quando o protótipo é desenvolvido.

3. Um sistema operante, embora limitado, se toma rapidamente disponível, a fim de mostrar a viabilidade e a utilidade da aplicação para a gerência.

4. O protótipo pode ser utilizado como uma base para escrever a especificação para um sistema com qualidade de produção.

Desenvolver um protótipo normalmente leva às melhorias na especificação do sistema. Uma vez disponível um protótipo, ele pode também ser utilizado para outros propósitos (Ince e Hekmatpour, 1987):

1- Treinamento de usuário. Um sistema protótipo pode ser utilizado para treinar os usuários antes que o sistema final seja entregue.

2-Teste de sistema. Os protótipos podem executar testes *back-to-back.* Os mesmos casos de testes são submetidos ao protótipo e ao sistema que está em teste. Se ambos os sistemas apresentarem o mesmo resultado, o caso de teste não terá detectado um defeito. Se os resultados forem diferentes, isso poderá significar que há um defeito no sistema e as razões dessa diferença deverão ser investigadas.

Em um estudo de 39 diferentes projetos de prototipação, Gordon e Bieman (1995) constataram que os benefícios do uso da prototipação no processo de software eram:

1-melhoria na facilidade de uso do sistema;
2-maior aproximação do sistema com as necessidades dos usuários;
3-melhoria da qualidade do projeto;
4-melhoria na facilidade de manutenção, e
5-esforço de desenvolvimento reduzido.

O estudo sugere, portanto, que as melhorias na facilidade de uso e melhores requisitos de usuário, provenientes do uso de protótipo, não significam necessariamente um aumento geral dos custos de desenvolvimento de sistema.

ANÁLISE E GERÊNCIA DE REQUISITOS

A prototipação geralmente aumenta os custos nos primeiros estágios do processo de software, mas reduz os custos posteriores. A razão principal disso é que o retrabalho durante o desenvolvimento é evitado, quando os clientes solicitam menos mudanças no sistema.

Contudo, eles constataram que uma conseqüência negativa da prototipação é que o desempenho geral do sistema é, por vezes, degradado quando um código de protótipo ineficiente é reutilizado.

Um modelo de processo para o desenvolvimento de protótipo é mostrado na Figura 7.1. Os objetivos da prototipação devem ficar explícitos desde o princípio do processo. Esses objetivos podem ser: desenvolver um sistema para o protótipo de interface com o usuário, desenvolver um sistema para validar os requisitos funcionais do sistema ou desenvolver um sistema para mostrar a viabilidade da aplicação para a gerência.

O mesmo protótipo não pode atender a todos os objetivos. Se os objetivos ficam implícitos, a gerência ou os usuários finais podem entender mal a função do protótipo. Conseqüentemente, eles podem não receber os benefícios que esperavam do desenvolvimento do protótipo.

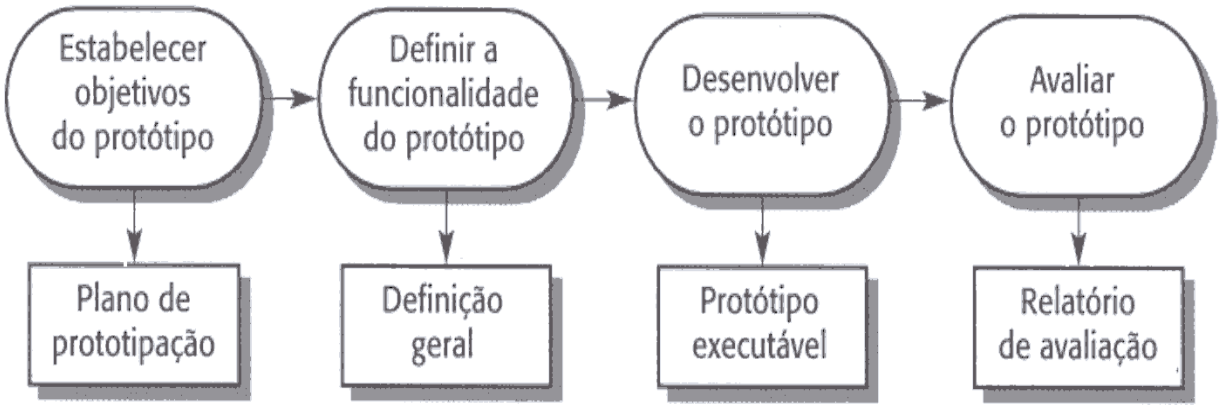
O estágio seguinte no processo é decidir o que incluir no sistema protótipo e, talvez o mais importante, o que deve ser excluído do sistema. Para reduzir os custos de prototipação e acelerar a programação da entrega, pode-se excluir alguma funcionalidade do protótipo.

É possível decidir diminuir os requisitos não funcionais, como o tempo de resposta e o uso de memória. O gerenciamento e o tratamento de erros podem ser ignorados ou ser rudimentares, a menos que o objetivo do protótipo seja estabelecer uma interface com o usuário. Os padrões de confiabilidade e qualidade de programa podem ser reduzidos.

O estágio final do processo é a avaliação do protótipo. Ince e Hekmatpour sugerem que esse é o estágio mais importante da prototipação. Deve ser feita uma provisão durante esse estágio para o treinamento de usuário, e os objetivos do protótipo devem ser utilizados na preparação de um plano para avaliação.

Os usuários precisam de algum tempo para se sentir à vontade com um sistema novo e para estabelecerem um padrão normal de uso. Quando estiverem utilizando o sistema normalmente, eles descobrirão erros e omissões nos requisitos.

Figura 7.1 Processo de desenvolvimento de protótipo.



7.1 Prototipação no processo de software

Como já foi discutido anteriormente, é difícil para os usuários finais prever como vão utilizar novos sistemas de software para dar apoio ao seu trabalho diário. Se esses sistemas forem grandes e complexos, provavelmente será impossível fazer essa avaliação antes de o sistema ser construído e colocado em operação.

Uma maneira de lidar com essa dificuldade é utilizar uma abordagem evolucionária para o desenvolvimento do sistema. Isso significa fornecer ao usuário um sistema que é incompleto e, então, modificá-lo e aumentá-Io até que os requisitos do usuário se tornem claros.

Como alternativa, é possível decidir deliberadamente construir um protótipo descartável para ajudar na análise e na validação de requisitos. Depois da avaliação, o protótipo é descartado e um sistema com qualidade de produção é construído. A Figura 7.2 ilustra ambas as abordagens de desenvolvimento de protótipos.

A prototipação evolucionária começa com um sistema relativamente simples, que implementa os requisitos mais importantes do usuário.

Esse sistema é ampliado e alterado à medida que novos requisitos vão sendo descobertos. Por fim, ele se torna o sistema que foi requerido. Não existe nenhuma especificação detalhada de sistema e, em muitos casos, pode não haver um documento formal de requisitos.

ANÁLISE E GERÊNCIA DE REQUISITOS

A prototipação evolucionária é, atualmente, a técnica utilizada para desenvolvimento de sites Web e aplicações de comércio eletrônico.

Por outro lado, a abordagem de prototipação descartável se destina a ajudar no aprimoramento e na classificação da especificação do sistema. O protótipo é escrito, avaliado e modificado.

A avaliação do protótipo informa o desenvolvimento da especificação detalhada do sistema, que é incluída no documento de requisitos de sistema.

Uma vez escrita a especificação, o protótipo não tem mais utilidade e é descartado.

Existe uma importante diferença entre os objetivos da programação evolucionária e da programação descartável:

1. O objetivo da prototipação evolucionária é fornecer um sistema funcional aos usuários finais. Isso significa que, normalmente, o processo se inicia com os requisitos do usuário que são mais bem compreendidos e com os que têm maior prioridade, Os requisitos de menor prioridade e os mais vagos são implementados quando e se eles forem solicitados pelos usuários.

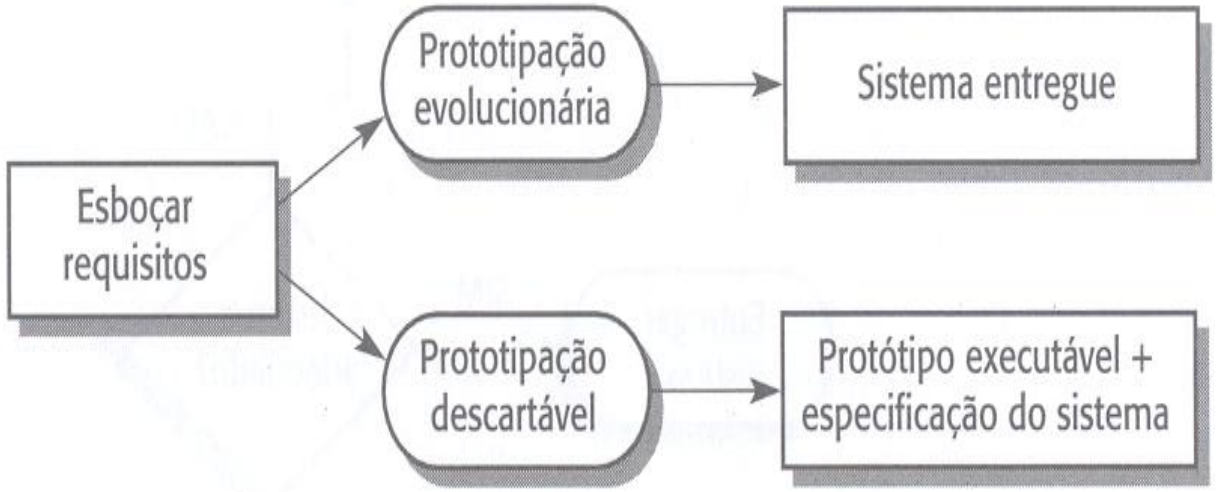
2. O objetivo da prototipação descartável é validar ou derivar os requisitos do sistema. É preciso começar com aqueles requisitos que não são bem compreendidos, porque é preciso saber mais sobre eles. Os requisitos que são diretos podem nunca ser prototipados.

Outra distinção importante entre essas abordagens é quanto ao gerenciamento da qualidade dos sistemas.

Por definição, os protótipos descartáveis têm duração muito curta. Deve ser possível modificá-los muito rapidamente durante o desenvolvimento, mas a facilidade de manutenção a longo prazo não é exigida.

O baixo nível de desempenho e confiabilidade pode ser aceitável em um protótipo descartável, desde que ele cumpra com sua função principal de ajudar no entendimento dos requisitos.

Figura 7.2 A prototipação evolucionária e a prototipação descartável.



Por outro lado, os protótipos que evoluem para o sistema final devem ser desenvolvidos com os mesmos padrões de qualidade organizacional de qualquer outro software.

Eles devem ter sólida estrutura, de modo que possam receber manutenção por muitos anos.

Eles devem ser confiáveis e eficientes e se adequar a padrões organizacionais relevantes.

7.1.1 Prototipação evolucionária

A prototipação evolucionária se baseia na idéia de desenvolver uma implementação inicial, expondo-a aos comentários dos usuários e aperfeiçoando-a ao longo de muitos estágios, até que um sistema adequado tenha sido desenvolvido (Figura 7.3).

Essa abordagem de desenvolvimento foi inicialmente utilizada para os sistemas (por exemplo, os sistemas de AI -*artificial intelligence* ou inteligência artificial) que são difíceis ou impossíveis de especificar.

Contudo, atualmente essa se tomou uma técnica importante de desenvolvimento de software.

A prototipação evolucionária faz parte das técnicas de RAD *(rapid application development* -desenvolvimento rápido de aplicações) e de JAD *(joint application development* -desenvolvimento conjunto de aplicações) (Millington e Stapleton, 1995; Wood e Silver, 1995; Stapleton, 1997) ou tem muito em comum com essas técnicas.

Existem duas vantagens principais em adotar essa abordagem para o desenvolvimento de software:

1. *Rápido fornecimento do sistema*. Como discuti na introdução do livro, o ritmo das mudanças nos negócios significa que é essencial que o apoio ao software seja disponibilizado rapidamente. Em alguns casos, o rápido fornecimento e a facilidade de uso são mais importantes do que os detalhes de funcionalidade ou a facilidade de manutenção de software a longo prazo.

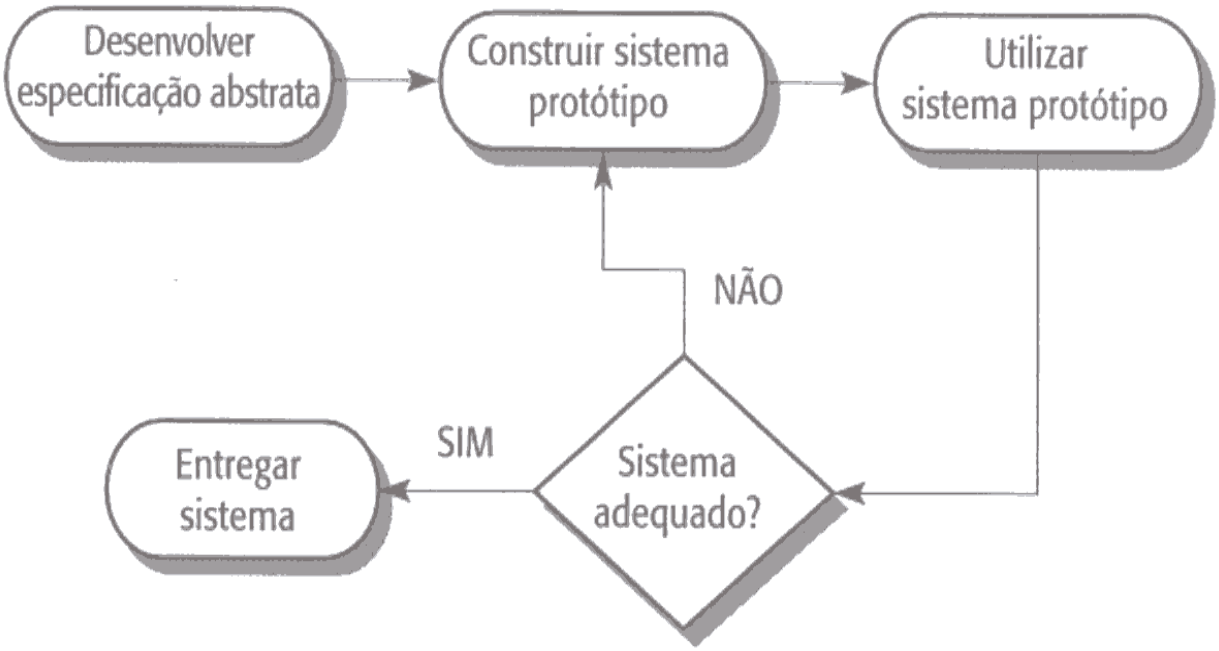
2. ***Compromisso do usuário com* o *sistema*** O envolvimento de usuários com o processo de desenvolvimento não quer dizer apenas que o sistema tem maior possibilidade de atender a seus requisitos, mas também significa que os usuários finais do sistema assumiram um compromisso e provavelmente vão querer contribuir para que ele funcione.

Há diferenças de detalhes entre os métodos particulares de rápido desenvolvimento de software, mas todos compartilham algumas características fundamentais:

1. O processo de especificação, projeto e implementação são intercalados. Não existe especificação detalhada de sistema e a documentação de projeto produzida depende normalmente das ferramentas utilizadas para implementar o sistema. O documento de requisitos do usuário somente define as características mais importantes do sistema.

2. O sistema é desenvolvido em uma série de estágios. Os usuários finais e outros *stake-holders* são envolvidos no projeto e na avaliação, a cada estágio. Eles podem propor mudanças no software e novos requisitos, que devem ser implementados em uma versão posterior do sistema.

Figura78.3 Prototipação evolucionária.



3. As técnicas para o desenvolvimento rápido de sistema são utilizadas. Elas podem conter ferramentas CASE e linguagens de quarta geração (4GL).

4. As interfaces com o usuário do sistema são, geralmente, desenvolvidas utilizando-se um sistema de desenvolvimento interativo, que permite que o projeto de interface seja criado rapidamente, mediante o desenho e a colocação de ícones nas inter-faces.

As abordagens com base na prototipação evolucionária e na especificação para o desenvolvimento de software diferem em sua visão da verificação e da validação.

A verificação é o processo de conferir se um programa atende a suas especificações. Como não existe especificação detalhada para o protótipo, a verificação é, por conseguinte, impossível.

A validação deve mostrar que o programa é adequado ao propósito previsto, em vez de sua conformidade com uma especificação. Isso também é difícil sem uma especificação detalhada, uma vez que não há nenhuma declaração explicita de propósito.

Os usuários finais envolvidos no processo podem estar felizes com o sistema, mas outros *stakeholders* podem se sentir excluídos e insatisfeitos com o sistema quanto ao atendimento de seus propósitos.

Portanto, a verificação e a validação de um sistema que foi desenvolvido com o uso de prototipação evolucionária podem somente verificar se o sistema é adequado, isto é, se ele é suficientemente bom para o propósito previsto.

A adequação, naturalmente, não é mensurável de imediato, e somente podem ser feitos julgamentos subjetivos da adequação de um programa. Isso não invalida sua utilidade; o desempenho humano não pode ser garantido como correto, mas ficamos satisfeitos se o desempenho for adequado à tarefa em questão.

Contudo, como será discutido a seguir, isso não causa problemas nos casos em que os sistemas são desenvolvidos para clientes por um fornecedor externo de desenvolvimento de software.

Existem três problemas principais com a prototipação evolucionária, que são particularmente importantes quando são desenvolvidos sistemas grandes, com longo tempo de duração:

1. ***Problemas de gerenciamento*** As estruturas de gerenciamento de software para sistemas de grande porte são estabelecidas para lidar com um modelo de processo de software que gera produtos regulares, a fim de avaliar o progresso. Os protótipos evoluem tão rapidamente que não é eficaz, sob o ponto de vista do custo, produzir um grande volume de documentação para o sistema. Além disso, o desenvolvimento rápido de protótipos pode requerer o uso de tecnologias que não são familiares. Os gerentes podem achar difícil utilizar os funcionários já existentes, porque eles não apresentam essas habilidades.

2. ***Problemas de manutenção*** A continuidade das mudanças tende a corromper a estrutura do protótipo de sistema. Isso significa que qualquer um que não seja do grupo de desenvolvedores originais provavelmente achará difícil compreendê-Ia. Além disso, se uma tecnologia especializada for utilizada para apoiar o desenvolvimento rápido de um protótipo, ela pode se tomar obsoleta. Portanto, encontrar pessoas que tenham o conhecimento necessário para fazer a manutenção do sistema pode ser difícil.

3. ***Problemas contratuais*** O modelo normal de contrato entre um cliente e um desenvolvedor de software se baseia em uma especificação de sistema. Quando não houver essa especificação, pode ser difícil estabelecer um contrato para o desenvolvimento do sistema. Os clientes podem não se

sentir à vontade com um contrato que simplesmente paga os desenvolvedores pelo tempo gasto no projeto, uma vez que isso pode levar a serviços prolongados e ultrapassar o orçamento definido; os desenvolvedores certamente não estarão dispostos a aceitar um contrato com preço fixo, uma vez que não podem controlar as mudanças requeridas pelos usuários finais.

Essas dificuldades significam que os clientes devem estar cientes do uso da prototipação evolucionária como uma técnica de desenvolvimento.

Ela possibilita o rápido desenvolvimento e o fornecimento de sistemas pequenos e de porte médio. Os custos de desenvolvimento de sistema podem ser reduzidos e a facilidade de uso é melhorada.

Se os usuários estiverem envolvidos no desenvolvimento, o sistema provavelmente deverá estar adequado a suas verdadeiras necessidades.

Contudo, as organizações que utilizam essa abordagem devem aceitar que o tempo de duração do sistema deverá ser relativamente curto. À medida que os problemas de manutenção aumentam, o sistema terá de ser substituído ou completamente reescrito.

Para grandes sistemas, que podem envolver uma série de diferentes fornecedores subcontratados, os problemas de gerenciamento da prototipação evolucionária se tomam intratáveis.

Quando são desenvolvidos protótipos para partes desses grandes sistemas, eles devem ser protótipos descartáveis.

O desenvolvimento incremental (Figura 7.4) evita alguns problemas da mudança constante que caracterizam a prototipação evolucionária. Uma arquitetura geral de sistema é estabelecida no início do processo, a fim de atuar como um *framework.*

Os componentes de sistema são desenvolvidos de maneira incremental e entregues dentro desse *framework.*

Uma vez validados e entregues, nem o *framework* nem os componentes são modificados, a menos que sejam descobertos erros.

O feedback dos usuários sobre os componentes fornecidos, contudo, pode influenciar no projeto de componentes programados para entrega posterior.

O desenvolvimento incremental é mais fácil de ser gerenciado do que a prototipação evolucionária, quando os padrões normais de processo de software são seguidos.

Planos e documentação devem ser produzidos para cada novo estágio do sistema. Esse processo permite algum feedback dos usuários no início do processo e limita os erros de sistema, quando a equipe de desenvolvimento não está preocupada com interações entre partes inteiramente diferentes do sistema de software.

Uma vez concluído um novo estágio do sistema, suas inter- faces são ‘congeladas’. Os estágios posteriores têm de se adaptar a essas interfaces e podem ser testados em relação a elas.

7.1.2 Prototipação Descartável

Um modelo de processo de software com base em um estágio inicial de prototipação descartável está ilustrado na Figura 7.5.

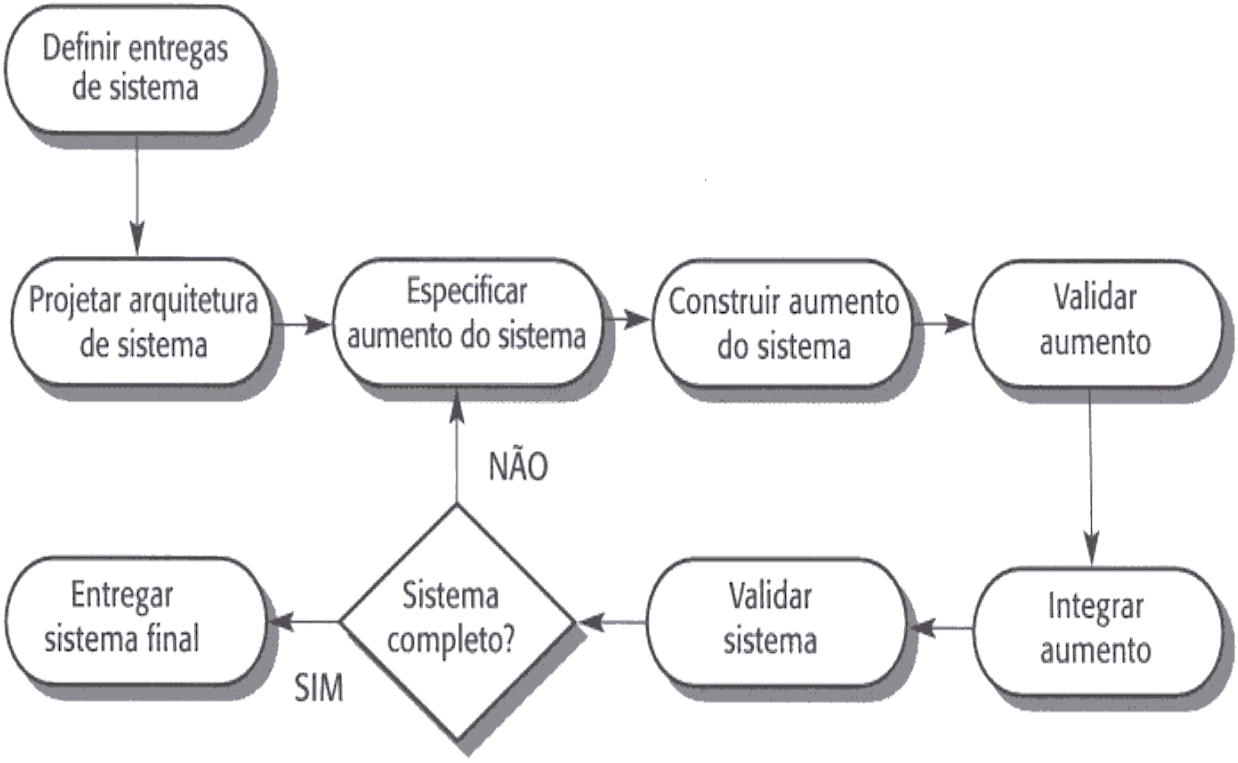
Essa abordagem amplia o processo de análise de requisitos, com a intenção de reduzir os custos totais do ciclo de vida.

A função principal do protótipo é esclarecer os requisitos e fornecer informações adicionais para os gerentes avaliarem riscos de processo. Depois da avaliação, o protótipo é descartado. Ele não é utilizado como base para o desenvolvimento posterior de sistema.

Essa abordagem para a prototipação de sistemas é comumente utilizada para os sistemas de hardware. O protótipo é utilizado para verificar o projeto, antes que dispendiosos encargos referentes à fabricação do sistema tenham sido considerados.

Um protótipo de sistema eletrônico pode ser desenvolvido utilizando-se componentes de prateleira, antes que sejam feitos investimentos em circuitos integrados de aplicação especial para implementar a versão de produção do sistema.

Figura 7.4 Um processo de desenvolvimento incremental



Contudo, o protótipo descartável de software não é normalmente utilizado para a validação de projeto, mas para ajudar a desenvolver os requisitos do sistema.

O projeto de protótipo, freqüentemente, é bem diferente daquele do sistema final.

O sistema deve ser desenvolvido tão rapidamente quanto possível, de modo que os usuários possam fornecer o feedback de sua experiência com o protótipo, para que este seja utilizado no desenvolvimento da especificação do sistema.

A funcionalidade pode ser retirada do protótipo descartável quando essas funções são bem compreendidas, os padrões de qualidade podem ser reduzidos e os critérios de desempenho, ignorados.

A linguagem de prototipação, muitas vezes, será diferente da linguagem final de implementação do sistema.

ANÁLISE E GERÊNCIA DE REQUISITOS

O modelo de processo, na Figura 7.5, considera que o protótipo é desenvolvido a partir de um esboço da especificação de sistema, fornecido para experimentação e, em seguida, modificado, até que o cliente esteja satisfeito com sua funcionalidade.

Nesse estágio, um modelo de processo de software por fases é iniciado, uma especificação é derivada do protótipo e o sistema é reimplementado em uma versão final de produção. Os componentes do protótipo podem ser reutilizados no sistema com qualidade de produção e, assim, os custos de produção podem ser reduzidos.

Em vez de derivar uma especificação do protótipo, algumas vezes, sugere-se que a especificação do sistema possa ser a própria implementação do protótipo. A instrução dada ao fornecedor do software poderia ser simplesmente 'escreva um sistema como este'. Existem vários problemas com essa abordagem:

1. Características importantes podem ter sido excluídas do protótipo, para simplificar uma implementação rápida. Na verdade, pode não ser possível fazer o protótipo de algumas das partes mais importantes do sistema, como as funções cruciais de segurança.

2. Uma implementação não tem apoio legal como um contrato entre o cliente e o fornecedor.

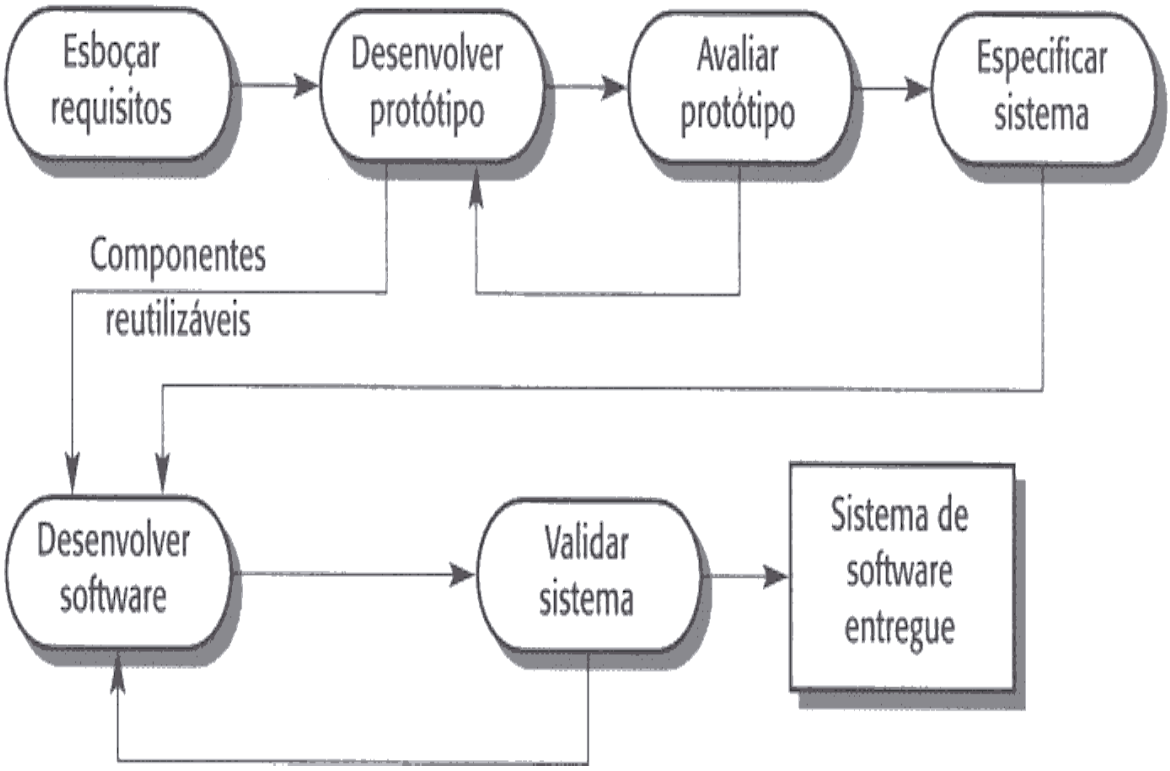
.3. Os requisitos não funcionais, como os relativos à confiabilidade, à capacidade e à segurança, não podem ser adequadamente testados em uma implementação de protótipo.

Um problema geral da prototipação descartável executável é que o modo de utilização do protótipo pode não corresponder à maneira como o sistema final entregue é utilizado.

O responsável pelo teste do protótipo pode estar particularmente interessado no sistema e pode não ser um típico representante de usuário do sistema. O tempo de treinamento durante a avaliação do protótipo pode ser insuficiente.

Se o protótipo for lento, os avaliadores poderão ajustar seu modo de trabalho e evitar essas características de sistema que tenham tempos de resposta demorados. Quando é fornecido com melhor tempo de resposta no sistema final, os avaliadores podem utilizá-lo de modo diferente.

Figura 7.5 Um processo de software com prototipação descartável.



Às vezes, os desenvolvedores sofrem pressões dos gerentes para que entreguem protótipos descartáveis para uso, particularmente quando há atrasos na entrega da versão final do software. Contudo, de modo geral isso não é aconselhável pelas seguintes razões:

1. Pode ser impossível ajustar o protótipo para atender aos requisitos não funcionais, como os requisitos de desempenho, proteção, robustez e confiabilidade, que foram ignorados durante o desenvolvimento do protótipo.

2. Mudanças rápidas durante a prototipação inevitavelmente significam que o protótipo não está documentado. A única especificação de projeto é o código do protótipo. Isso não é suficientemente bom para a manutenção a longo prazo.

3. As mudanças feitas durante o desenvolvimento do protótipo provavelmente terão degradado a estrutura do sistema. O sistema terá uma manutenção difícil e dispendiosa.

4. Os padrões de qualidade organizacional são, normalmente, diminuídos para o desenvolvimento do protótipo.

Os protótipos descartáveis não precisam ser protótipos executáveis de software para que sejam úteis no processo de engenharia de requisitos. Os modelos com base em papel, da interface com o usuário de sistema (Rettig, 1994), se mostraram efetivos em ajudar usuários a aprimorar um projeto de interface e trabalhar em cenários de uso comum.

Seu desenvolvimento é muito barato e eles podem ser construídos em poucos dias. Uma extensão dessa técnica é um protótipo do tipo 'Wizard of Oz' ('Mágico de Oz'), em que somente a interface com o usuário é desenvolvida.

Os usuários interagem com essa interface, mas seus pedidos são transmitidos para uma pessoa que os interpreta e fornece a resposta apropriada. Essas abordagens de prototipação são discutidas em Sommerville e Sawyer (1997).

7.2 Técnicas de prototipação rápida

Técnicas de prototipação rápida são técnicas de desenvolvimento que enfatizam a rapidez do fornecimento, em vez de outras características do sistema, como o desempenho, a facilidade de manutenção ou a confiabilidade. Existem três técnicas de desenvolvimento rápido que são práticas para a prototipação de nível industrial:

1. desenvolvimento com linguagem dinâmica de alto nível;

2. programação de banco de dados, e

3. montagem de componentes e aplicações.

Na prática, contudo, todas são freqüentemente utilizadas no desenvolvimento de um protótipo de sistema.

Por exemplo, uma linguagem de programação de banco de dados pode ser utilizada para extrair dados, com processamento detalhado que empregue componentes reutilizáveis.

A interface com o usuário de sistema pode ser definida utilizando-se programação visual. Luqi (1992) descreve como essa abordagem mista foi utilizada para criar um protótipo de sistema de comando e controle.

O desenvolvimento de protótipos é, hoje em dia, normalmente apoiado por um conjunto de ferramentas que contêm, pelo menos, duas dessas técnicas.

Por exemplo, o sistema Smalltalk Visual Works aceita uma linguagem de nível muito alto e fornece muitos componentes reutilizáveis que podem ser incluídos em aplicações. O Lotus Notes inclui apoio à programação de banco de dados utilizando uma linguagem de nível muito elevado e componentes reutilizáveis, que podem ser vinculados a operações de banco de dados.

Atualmente, a maioria dos sistemas de prototipação é compatível com uma abordagem de programação visual, na qual alguma parte, ou todo o protótipo, é desenvolvida interativamente.

Em vez de escrever um programa seqüencial, o desenvolvedor do protótipo manipula ícones gráficos que representam funções, dados ou componentes de interface com o usuário e associa scripts de processamento com esses ícones. Um programa executável é gerado automaticamente a partir da representação visual do sistema. Isso simplifica o desenvolvimento do programa e reduz custos de prototipação.

7.2.1 Desenvolvimento com linguagem dinâmica de alto nível

As linguagens dinâmicas de alto nível são linguagens de programação que incluem poderosos recursos de gerenciamento de dados em *run-time.* Elas simplificam o desenvolvimento de programas porque reduzem muitos problemas de alocação de armazenamento e gerenciamento. O sistema de linguagens inclui recursos que normalmente precisam ser construídos a partir de construções mais primitivas, em linguagens como Ada ou C. Exemplos de linguagem de nível muito alto são Lisp (com base em estruturas de lista), Prolog (com base em lógica) e Smalltalk (com base em objetos).

Até relativamente pouco tempo atrás, as linguagens dinâmicas de nível muito alto não eram largamente utilizadas para o desenvolvimento de grandes sistemas, porque elas necessitavam de um grande sistema de suporte de *run-time.* Esse suporte de *run-time* aumenta as necessidades de armazenamento e reduz as velocidades de execução de programas escritos nessa linguagem. Contudo, o aumento da capacidade e o custo reduzido de hardware de computador fizeram com que esses fatores passassem a ter menos importância.

Isso significa que, para muitas aplicações de negócios, essas linguagens podem substituir linguagens de programação imperativas, como C, Cobol e Ada. Java é claramente uma linguagem de programação importante, com raízes em C++, e ainda incorpora muitas caracteristicas de Smalltalk, como a independência de plataforma e o gerenciamento automático de armazenamento. Java proporciona muitas vantagens das linguagens de nível muito alto, com o rigor e as

oportunidades de otimização de desempenho, oferecidos pelas linguagens convencionais de terceira geração. Muitos componentes Java reutilizáveis estão disponíveis e, assim, ela é claramente uma linguagem muito adequada para a prototipação evolucionária.

A Figura 7.6 mostra as linguagens dinâmicas que são mais comumente utilizadas para a prototipação. Ao escolher uma linguagem de prototipação, é preciso ter em mente uma série de questões:

1- Qual é o *domínio de aplicação do problema?* Como mostra a Figura 8.6, diferentes linguagens são mais adequadas para diferentes domínios de aplicação. Se você deseja fazer protótipos de aplicações que envolvam o processamento de linguagem natural (digamos), uma linguagem como Lisp ou Prolog é mais adequada do que lava ou Smalltalk.

2- Que interação com o *usuário* é *exigida?* Diferentes linguagens fornecem diferentes níveis de apoio para a interação com o usuário. Algumas linguagens, como SmalltaIk e lava, são bem integradas com Web browsers, enquanto outras, como Prolog, são mais adequadas a interfaces com base em texto.

3- Que ambiente de apoio é *fornecido com a linguagem?* Ambientes de apoio bem desenvolvidos, com muitas ferramentas e fácil acesso a componentes reutilizáveis, simplificam o processo de desenvolvimento de protótipos.

As linguagens dinâmicas de alto nível podem ser utilizadas em combinação, a fim de criar um protótipo de sistema. Diferentes partes do sistema podem ser programadas em diferentes linguagens e *umframework* de comunicação pode ser estabelecido entre essas partes. Zave (1989)

Figura 7.6 Linguagens de Alto nível para prototipação.

Linguagem	**Tipo**	**Domínio de Aplicação**
Smalltalk	Orientada a objetos	Sistemas interativos
Java	Orientada a objetos	Sistemas interativos
Prolog	Lógica	Processamento simbólico
Lisp	com base em listas	Processamento simbólico

descreve essa abordagem de desenvolvimento na prototipação de um sistema de rede de tele- fonia. Quatro diferentes linguagens foram utilizadas: Prolog, para prototipação de banco de dados; Awk (Abo *et ai.,* 1988), para faturamento; CSP (Hoare, 1985), para especificação de protocolo, e PAISLey (Zave e Schell, 1986), para simulação de desempenho.

Não existe uma linguagem ideal para a prototipação de grandes sistemas quando as diferentes partes do sistema são tão diversificadas. A vantagem de uma abordagem com linguagens combinadas é que a linguagem mais apropriada para uma parte lógica da aplicação pode ser escolhida, apressando, assim, o desenvolvimento do protótipo.
A desvantagem é que pode ser difícil estabelecer um *framework* de comunicação que permita que várias linguagens se comuniquem. As entidades utilizadas nas diferentes linguagens são muito diferentes. Conseqüentemente, seções longas de código podem ser necessárias para traduzir uma entidade de uma linguagem para outra.

7.2.2 Programação de banco de dados

O desenvolvimento evolucionário é, atualmente, uma técnica-padrão para implementar aplicações pequenas e de porte médio no domínio de sistemas de negócios. A maioria das aplicações de negócios envolve a manipulação de dados a partir de um banco de dados e a produção de saídas que implicam organizar e formatar esses dados.

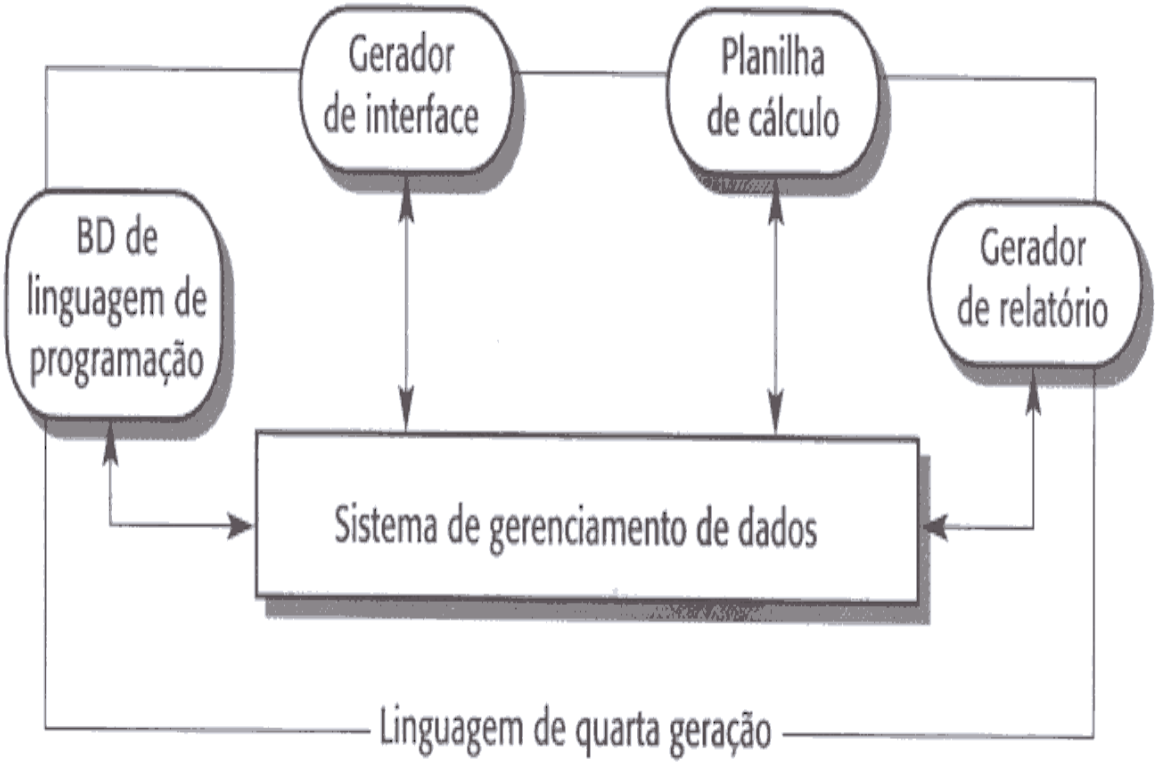
Para oferecer suporte ao desenvolvimento dessas aplicações, todos os sistemas de gerenciamento de bancos de dados comerciais, atualmente, aceitam a programação de bancos de dados.

A programação de bancos de dados é realizada utilizando-se uma linguagem especial, que envolve o conhecimento desse banco de dados e inclui operações que são ajustadas à manipulação de banco de dados.

O ambiente de apoio à linguagem fornece ferramentas para proporcionar compatibilidade com a definição de interface com o usuário, a computação numérica e a geração de relatórios. O termo *linguagem de quarta geração* (4GL) é utilizado para se referir à linguagem de programação de bancos de dados, como também a seu ambiente de apoio.

As linguagens de quarta geração são bem-sucedidas, porque há grande quantidade de aspectos comuns nas aplicações de processamento de dados. Essencialmente, essas aplicações se ocupam de atualizar bancos de dados e de produzir relatórios a partir das informações em bancos de dados. Formulários-padrão são utilizados para entradas e saídas. As 4GLs são ajustadas para produzirem aplicações interativas, que são compatíveis para abstrair informações obtidas em um banco de dados organizacional, apresentando o resultado a usuários finais em seu terminal ou estação de trabalho e depois atualizando o banco de dados com mudanças introduzidas pelos usuários. A interface com o usuário geralmente consiste em um conjunto de formulários-padrão ou planilhas de cálculo.

Figura 7.7 Componentes de linguagem de quarta geração.



As ferramentas incluídas em um ambiente 4GL (Figura 7.7) são:

1. Uma linguagem de consulta de banco de dados que, hoje em dia, é geralmente SQL (Date e Darwen, 1997). Ela pode ser fornecida diretamente ou gerada automaticamente a partir de formulários preenchidos por um usuário final.

2. Um gerador de interface, que é utilizado para criar formulários para a entrada e a exibição de dados.

3. Uma planilha de cálculos, para a análise e a manipulação de informações numéricas.

4. Um gerador de relatórios, que é utilizado para definir e criar relatórios a partir de informações em bancos de dados.

A maioria das aplicações de negócios utiliza formulários estruturados para entrada e saída, de modo que as 4GLs fornecem poderosos recursos para a definição de

tela e a geração de relatórios. As telas são, freqüentemente, definidas como uma série de relatórios vinculados (em uma aplicação que estudamos, havia 137 diferentes definições de formulários); portanto, o sistema de geração de telas deve oferecer:

1. ***definição interativa de formulários****,* em que o desenvolvedor define os campos a serem exibidos e como eles devem ser organizados;

2. ***vinculação de formulários****,* em que o desenvolvedor pode especificar que determinadas entradas acarretam a exibição de outros formulários, e

3. ***verificação de campo****,* em que o desenvolvedor define a variação permitida para a inserção de valores em campos de formulários.

A maioria das 4GLs, atualmente, aceita o desenvolvimento de interfaces de banco de dados com base em Web browsers. Isso permite que o banco de dados seja acessado a partir de qualquer lugar, com uma conexão válida de Internet, reduzindo os custos de treinamento e os custos de software e permitindo que os usuários externos tenham acesso a um banco de dados.

Contudo, as limitações inerentes aos Web browsers e aos protocolos da Internet significam que essa abordagem pode ser inadequada para sistemas em que seja necessário obter respostas muito rápidas e interativas.

O desenvolvimento com base nas 4GLs pode ser utilizado para a prototipação evolucionária ou em conjunto com uma análise com base em métodos, na qual modelos de sistema são utilizados para gerar o protótipo de sistema.

A estrutura que as ferramentas CASE impõem na aplicação e na documentação associada significa que os protótipos evolucionários desenvolvidos com o uso dessa abordagem devem ser de manutenção mais fácil do que os sistemas desenvolvidos manualmente. As ferramentas CASE podem gerar SQL ou código em uma linguagem de nível mais inferior, como Cobol. Forte (1992) descreve uma série de ferramentas desse tipo em uma breve análise sobre as linguagens de quarta geração.

Embora as 4GLs sejam muito adequadas para o desenvolvimento de protótipos, existem algumas desvantagens em utilizá-las para sistemas de produção. Os programas escritos em uma 4GL são, em geral, mais lentos do que programas similares em linguagens convencionais de programação e normalmente requerem muito mais memória.

Por exemplo, estive envolvido em um experimento no qual reescrever um programa 4GL em C ++ resultou em uma redução de 50 por cento nos requisitos de memória. O programa em C também foi executado dez vezes mais rápido do que o sistema em 4GL.

Embora elas claramente reduzam os custos de desenvolvimento de sistemas, o efeito das 4GLs nos custos totais do ciclo de vida de grandes sistemas de processamento de dados ainda não está claro.

Os programas tendem a ser desestruturados e de difícil manutenção, e as 4GLs não são padronizadas. Os custos pós-ímplantação podem, portanto, ser altos. Problemas específicos podem surgir quando sistemas que utilizem 4GL precisam passar pelo processo de reengenharia.

Não existe padronização ou uniformidade entre as linguagens de quarta geração. Isso significa que os usuários podem ter de reescrever programas, porque a linguagem em que eles foram escritos originalmente está obsoleta.

7.2.3 Montagem de componentes e aplicações

O tempo necessário para desenvolver um sistema poderá ser reduzido, se muitas partes desse sistema puderem ser reutilizadas, em vez de serem projetadas e implementadas.

Protótipos poderão ser construídos rapidamente, se tivermos um conjunto de componentes reutilizáveis e um mecanismo para compor esses componentes em sistemas. O mecanismo de composição deve conter recursos de controle e um mecanismo para a comunicação de componentes.

Essa abordagem é ilustrada na Figura 7.8.

A prototipação com componentes reutilizáveis envolve desenvolver uma especificação de sistema, levando em conta quais componentes reutilizáveis estão disponíveis, e isso pode significar que é preciso obter alguma conciliação entre os requisitos.

A funcionalidade dos componentes disponíveis pode não ter uma adequação precisa com os requisitos dos usuários. Contudo, os requisitos de usuário são freqüentemente bastante flexíveis e, assim, na maioria dos casos, essa abordagem pode ser utilizada para o desenvolvimento de protótipos.

O desenvolvimento de protótipos que emprega o reuso pode ser aceito em dois níveis:

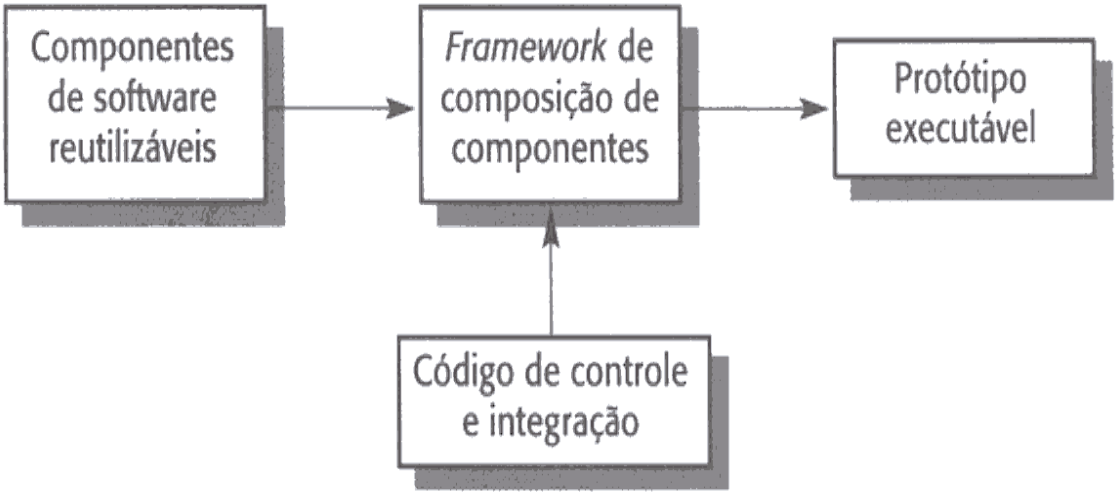
1. No nível de aplicações, em que sistemas inteiros de aplicações são integrados com o protótipo, de modo que sua funcionalidade possa ser compartilhada. Por exemplo, se o protótipo requer uma capacidade de processamento de textos, isso pode ser fornecido pela integração de um sistema-padrão de processamento de textos. Aplicações como o Microsoft Office suportam ligações de aplicativos.

2. No nível de componentes, em que componentes individuais são integrados dentro de um *framework-padrão* a fim de implementar o sistema. Esse *framework-padrão* pode ser uma linguagem de *scripting,* que é projetada para o desenvolvimento evolucionário, como Visual Basic, *TCL/TK* (Ousterhout, 1994), Python (Lutz, 1996) ou Perl (Wall *et ai.,* 1996). Como alternativa, pode ser um *framework* de integração de componentes mais genérico, com base em CORBA, DCOM ou JavaBeans (Sessions, 1997; Orfali e Harkey, 1998; Pope, 1998).

O reuso de aplicações permite o acesso a toda a funcionalidade de uma aplicação. Se a aplicação também fornece recursos de *scripting* ou de composição (por exemplo, macros do Excel), esses recursos podem ser utilizados para desenvolver alguma funcionalidade de protótipo. Uma metáfora de documento composto é útil para compreender essa abordagem para o desenvolvimento de protótipos.

Os dados processados pelo sistema do protótipo são organizados em um documento composto, que atua como um contêiner para vários objetos diferentes. Esses objetos contêm diferentes tipos de dados (como uma tabela, um diagrama e um formulário), que podem ser processados por diferentes aplicações. Eles são vinculados e tipificados, de maneira que o acesso a um objeto resulta em que a aplicação associada seja iniciada.

Figura 7.8 Composição de componente reutilizável.



A Figura 7.9 ilustra a noção de um protótipo de sistema constituído de um documento composto, que inclui elementos de texto, elementos de planilha de cálculo e arquivos de som.

Os elementos de texto são processados pelo processador de textos, as tabelas, por aplicativos de planilhas de cálculo, e os arquivos de som, por um reprodutor de áudio. Quando um usuário do sistema acessa um objeto de um tipo particular, o aplicativo associado é chamado para fornecer funcionalidade ao usuário. Por exemplo, quando objetos do tipo som são acessados, o programa de reprodução de áudio é chamado para processar esses objetos.

Um sistema de apoio ao gerenciamento de requisitos necessita de um meio de captar requisitos, armazenar esses requisitos, produzir relatórios, descobrir relações entre os requisitos e gerenciar essas relações como tabelas de rastreamento. Isso pode ser desenvolvido em protótipos, por meio da vinculação de um banco de dados (para armazenar os requisitos), um processador de textos (para obter os requisitos e formatar os relatórios), uma planilha de cálculos (para gerenciar as tabelas de facilidade de rastreamento) e um código especialmente escrito para encontrar relações entre os requisitos.

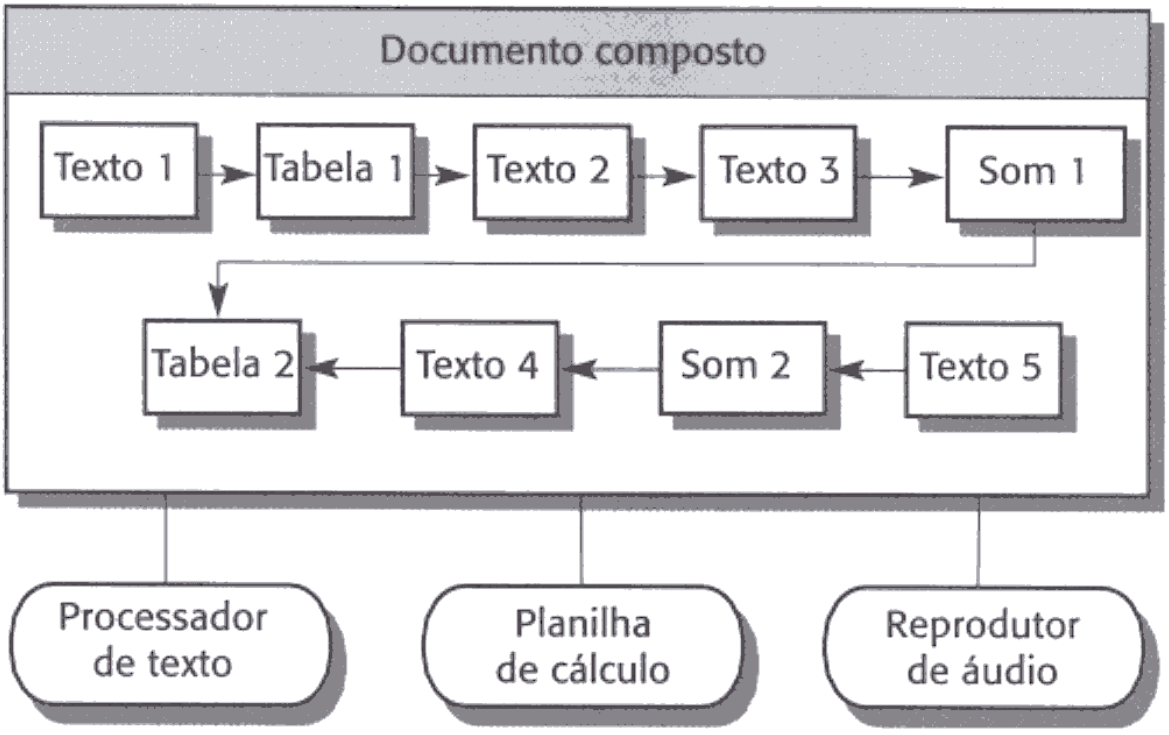
A principal vantagem dessa abordagem é que uma porção da funcionalidade do protótipo pode ser implementada rapidamente, a um custo muito baixo. Se os usuários do protótipo já estiverem familiarizados com as aplicações que o constituem, eles não precisarão aprender como utilizar esses novos recursos. Contudo, se eles não souberem utilizar as aplicações, o aprendizado poderá ser difícil, especialmente quando eles puderem ser confundidos com uma

funcionalidade que não seja necessária. É possível que haja problemas de desempenho com o protótipo, por causa da necessidade de trocar de um sistema de aplicações para outro. Isso depende do apoio fornecido ao sistema operacional. O apoio mais amplamente utilizado para essa abordagem é o mecanismo OLE *(object linking and embedding* - vinculação e inclusão de objetos) da Microsoft (Sessions, 1997).

Nem sempre é possível ou sensato reutilizar aplicações inteiras. O desenvolvimento com reuso depende de sofisticados componentes reutilizáveis, que podem ser funções ou objetos que realizam operações específicas, como classificação, busca e exibição.

O sistema de protótipos é desenvolvido com a definição de uma estrutura geral de controle e, então, com a integração dos componentes com essa estrutura. Se os componentes que realizam a função requerida não estiverem disponíveis, será desenvolvido um código de propósito especial; se for apropriado, ele estará disponível para futuro reuso.

Figura 7.9 Vinculação de aplicativos



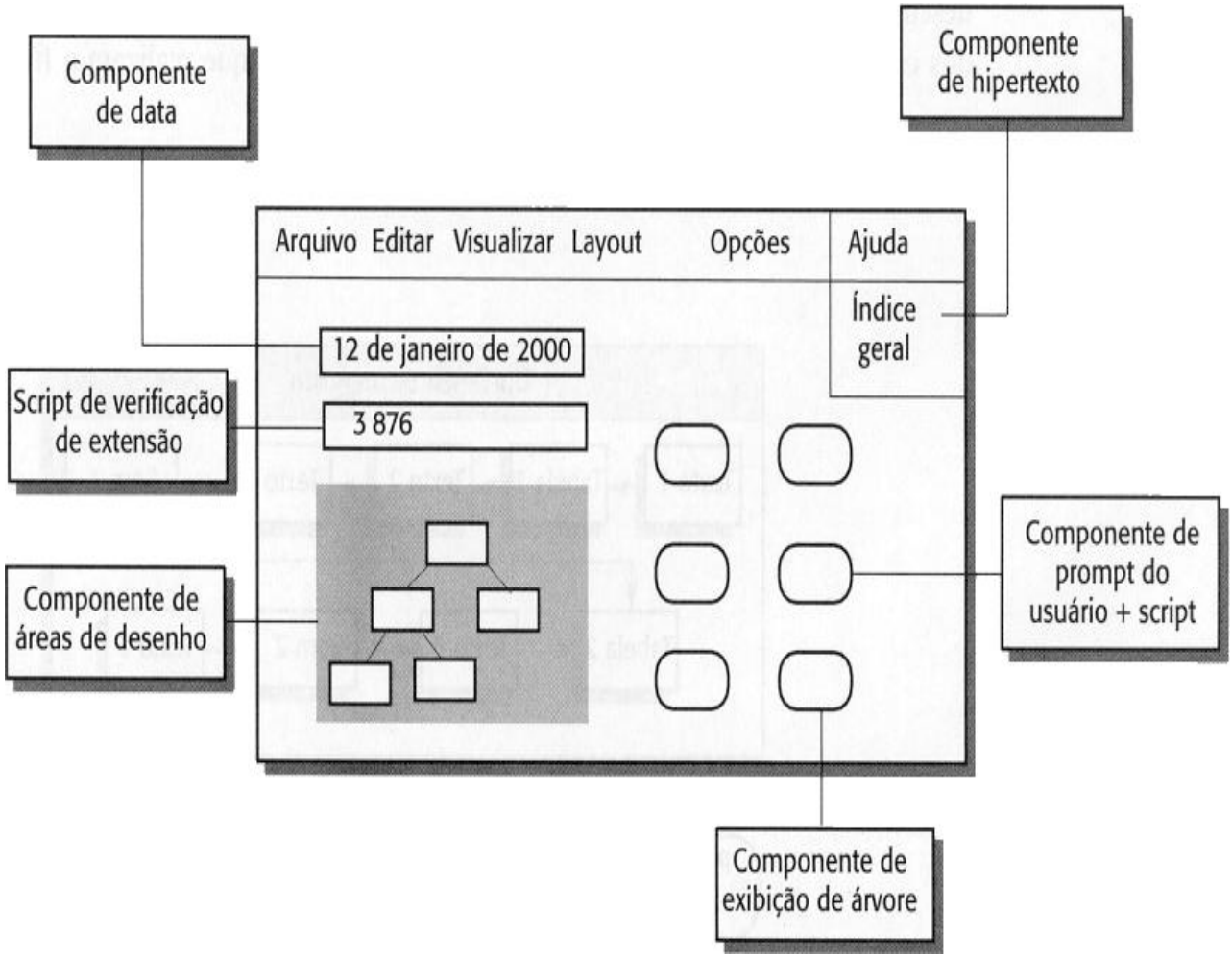
Os sistemas de desenvolvimento visual, como o Visual Basic, são compatíveis com essa abordagem baseada no reuso, para o desenvolvimento de aplicações. Os programadores de aplicações constroem o sistema interativamente, pela definição da interface em termos de telas, campos, botões e menus. Esses itens são denominados e scripts de processamento são associados com partes individuais da interface (por exemplo, um botão denominado 'Simular'). Esses scripts podem ser chamados de componentes reutilizáveis, código de propósito especial ou uma mistura de ambos.

Ilustro essa abordagem na Figura 7.10, que mostra uma tela da aplicação com menus na parte superior, campos de entrada (os campos brancos à esquerda do display), campos de saída (o campo sombreado à esquerda do display) e botões, que são os retângulos com cantos arredondados, à direita do display. Quando essas entidades são posicionadas no display pelo sistema de construção de protótipos, o desenvolvedor define que componente reutilizável deve ser associado a eles ou escreve um fragmento de programa para realizar o processamento necessário. Na Figura 7.10, mostro o componente associado a alguns elementos do display.

O Visual Basic é um exemplo de uma família de linguagens chamada de linguagens de *scrip- ting* (Ousterhout, 1998). As linguagens de *scripting* são linguagens *typeless* de alto nível, que são projetadas para integrar componentes e criar sistemas. Um exemplo inicial de uma linguagem de *scripting* foi o shell do Unix (Boume, 1978) e, desde seu desenvolvimento, uma série de outras linguagens de *scripting* mais poderosas foram criadas (Ousterhout, 1994; Lutz, 1996; Wall *et al.,* 1996). As linguagens de *scripting* incluem estruturas de controle e caixas de ferramentas gráficas, e Ousterhout (1998) ilustra que elas reduzem drasticamente o tempo necessário para o desenvolvimento de sistema.

Essa abordagem para o desenvolvimento de sistemas permite o rápido desenvolvimento de aplicações relativamente simples e pequenas, que podem ser construídas por uma pessoa ou por uma pequena equipe. Para grandes sistemas, que devem ser desenvolvidos por equipes maiores, a organização é mais difícil. Não existe uma arquitetura explícita de sistema e, freqüentemente, ocorrem complexas dependências entre as diferentes partes do sistema. Isso causa dificuldades, quando são necessárias mudanças. Essa abordagem também é limitada a um conjunto específico de objetos de interação e, assim, pode ser difícil implementar interfaces com o usuário que não sejam padronizadas. O desenvolvimento mais genérico com base em componentes em tomo de uma arquitetura de *framework.,* é mais apropriado para grandes sistemas.

Figura 7.10 Programação visual com reuso.



7.3 Prototipação de interface com o usuário

As interfaces gráficas com o usuário se tomaram, atualmente, a norma para sistemas interativos. O esforço envolvido na especificação, no projeto e na implementação de uma interface com o usuário representa parte significativa dos custos de desenvolvimento de aplicações.

Os projetistas não devem impor seus pontos de vista a respeito de uma interface com o usuário que seja aceitável pelos usuários. O usuário deve tomar parte do processo de projeto de interface. Essa compreensão leva a uma abordagem de projeto chamada de projeto centrado no usuário (Norman e Draper, 1986), que depende da prototipação de interface e do envolvimento do usuário com o processo de projeto de interface.

ANÁLISE E GERÊNCIA DE REQUISITOS

Do ponto de vista da engenharia de software, a prototipação é parte essencial do processo de projeto de interface. Devido à natureza dinâmica das interfaces com o usuário, as descrições textuais e os diagramas não são suficientemente bons para exprimir os requisitos da interface com o usuário.

Portanto, a prototipação evolucionária, com o envolvimento do usuário final, é a única maneira sensata de desenvolver interfaces gráficas com o usuário, destinadas a sistemas de software.

Os geradores de interface são sistemas de projeto de telas gráficas, em que componentes de interface, como menus, campos, ícones e botões, são relacionados a partir de um menu e posicionados em uma interface.

Como já foi discutido, os sistemas desse tipo são parte essencial de um sistema de programação de bancos de dados, e o Visual Basic baseou sua técnica-padrão de desenvolvimento nesse sistema. Shneiderman (1998) discute uma série desses sistemas.

Os geradores de interface criam um programa bem estruturado, gerado a partir de uma especificação de interface. As iterações, que são parte inerente do desenvolvimento evolucionário, não degradam a estrutura do software, e a reimplementação não é necessária.

Milhões de pessoas têm acesso aos Web browsers. Eles aceitam uma linguagem de definição de páginas (HTML), que foi estendida a partir de uma simples linguagem de marcação de texto para uma notação abrangente, destinada à especificação de interface com o usuário. Botões, campos, formulários e tabelas podem ser incluídos nas páginas Web, assim como objetos de multimídia, que permitem o acesso a sons, vídeo e exibições de realidade virtual.

Os scripts de processamento podem ser associados com objetos de interface com o usuário, com processamento realizado no cliente Web ou, de modo centralizado, no servidor Web. Devido à grande disponibilidade de Web browsers e à eficiência do HTML e de suas capacidades de processamento associadas, cada vez mais as interfaces com o usuário estão sendo construídas como interfaces com base na Web. Essas interfaces não estão simplesmente limitadas a novos sistemas.

As interfaces com base na Web estão substituindo as interfaces com base em formulários, para uma grande variedade de sistemas legados.

As interfaces com o usuário, com base na Web, podem ser prototipadas utilizando-se um editor-padrão de site Web, que é essencialmente um construtor de interface com o usuário. As entidades de página Web são definidas e posicionadas e ações são relacionadas com elas, seja com a utilização de

recursos de HTML instalados (por exemplo, vinculação com outra página), seja com a utilização de scripts lava ou CGI.

Pontos-chaves

Um protótipo de sistema pode ser desenvolvido para dar aos usuários finais de sistema uma impressão concreta das capacidades desse sistema.

O protótipo pode, portanto, ajudar a estabelecer e validar os requisitos de sistema.

.À medida que aumenta a necessidade de um rápido fornecimento de software, a prototipação está se tornando cada vez mais comum, como a técnica-padrão de desenvolvimento para sistemas pequenos e de porte médio, especialmente no domínio de negócios.

A prototipação descartável envolve o desenvolvimento de um protótipo para compreender os requisitos do sistema. Na prototipação evolucionária, um protótipo evolui de várias versões até o sistema final.

Ao implementar um protótipo descartável, primeiramente são desenvolvidas as partes menos compreendidas do sistema; em um protótipo evolucionário, são desenvolvidas as partes do sistema que são mais bem compreendidas.

O desenvolvimento rápido é importante para os protótipos de sistemas. A fim de entregar um protótipo rapidamente, talvez seja necessário excluir algumas funcionalidades de sistema ou diminuir os requisitos não funcionais, como a rapidez de resposta e a confiabilidade.

.Entre as técnicas de prototipação estão o uso de linguagens de nível muito elevado, a programação de bancos de dados e a construção de um protótipo a partir de componentes reutilizáveis. Muitos ambientes de prototipação são compatíveis com uma abordagem de programação visual de desenvolvimento.

.As interfaces com o usuário deveriam sempre ser desenvolvidas com a prototipação, quando não é possível especificar essas interfaces efetivamente utilizando um modelo estático. Os usuários deveriam estar envolvidos na avaliação e na evolução do protótipo.

ANÁLISE E GERÊNCIA DE REQUISITOS

Exercícios

1- Foi solicitado que você investigasse a viabilidade da prototipação no processo de desenvolvimento de software em sua organização. Escreva um relatório para seu gerente, discutindo as classes de projeto em que a prototipação deve ser utilizada e apresentando os custos e benefícios esperados a partir do uso da prototipação.

2- Explique por que, para o desenvolvimento de grandes sistemas, é recomendado que os protótipos sejam descartáveis.

3- Que características de linguagens, como Smalltalk e Lisp, contribuem para o apoio à prototipação rápida?

4- Sob quais circunstâncias você recomendaria que a prototipação deva ser utilizada como meio de validação dos requisitos de sistema?

5- Sugira as dificuldades que podem surgir quando se utiliza a prototipação em sistemas integrados de computador em tempo real.

6- Um gerente de software está envolvido no desenvolvimento de projeto de um sistema de apoio ao projeto de software, que aceita a tradução de requisitos de software para uma especificação formal de software. Comente as vantagens e as desvantagens das seguintes estratégias de desenvolvimento:

a) Desenvolva um protótipo descartável utilizando uma linguagem de prototipação, como Smalltalk. Avalie esse protótipo e revise os requisitos. Desenvolva o sistema final utilizando a linguagem C.

b) Desenvolva o sistema a partir dos requisitos existentes utilizando Java e modifique-o a fim de adaptá-lo a quaisquer requisitos modificados do usuário.

c) Desenvolva o sistema utilizando a prototipação evolucionária e empregando uma linguagem de prototipação como Smalltalk. Modifique o sistema de acordo com os requisitos do usuário e entregue o protótipo modificado.

7- Discuta a prototipação com a utilização de componentes reutilizáveis e sugira os problemas que podem surgir quando se emprega essa abordagem. Qual é o modo mais efetivo de especificar os componentes reutilizáveis?

8- Quais são as vantagens e as desvantagens de utilizar o mecanismo OLE, da Microsoft, para o desenvolvimento rápido de aplicações?

ANÁLISE E GERÊNCIA DE REQUISITOS

9- Uma entidade beneficente pediu que você fizesse o protótipo de um sistema que controle todas as doações recebidas. Esse sistema precisa manter os nomes e os endereços dos doadores, seus interesses específicos, a quantia doada e quando foi feita a doação. Quando a doação é superior a um certo valor, o doador pode estabelecer condições associadas à doação (por exemplo, ela deve ser gasta em um projeto específico), e o sistema deve manter o controle dessas doações e como elas foram empregadas. Discuta como você faria esse protótipo, tendo em mente que na entidade há trabalhadores pagos e voluntários. Muitos voluntários são aposentados que têm pouca ou nenhuma experiência com computadores.

10- Você desenvolveu um sistema de protótipo descartável para um cliente, que está muito feliz com ele. Contudo, ele sugere que não existe nenhuma necessidade de desenvolver outro sistema, mas que você deve fornecer o protótipo, e oferece um ótimo preço pelo sistema. Você sabe que pode haver futuros problemas com relação à manutenção do sistema. Discuta como você poderia responder a esse cliente.

8. BIBLIOGRAFIA

ABREU, M. C. O Professor Universitário em Sala de Aula. São Paulo, MG Associados, 1990

BERRY, D. M.; LAWRENCE, B. Requirements engineering. 1. ed. USA : IEEE Software, Mar./Apr. 1998. p. 26-29.

BEZERRA, E. Princípios de Análise e Projeto de Sistemas com UML. Rio de Janeiro, Campus, 2002.

FIORINI, S. T. & STAA, A. & BAPTISTA, R. M. Engenharia de Software com CMM. Rio de Janeiro, Brasport, 1998.

GAUSE, D. C.; WEINBERG, G. M. Exploring requirements (quality before design). 1. ed. USA : Dorset House Publishing Co. Inc., 1989. p. 300.

GAUSE, D. C.; WEINBERG, G. M. Are your lights on? 1. ed. USA: Dorset House Publishing Co. Inc., 1990. p. 157.

JACKSON, M. Software requirements and specifications: a lexicon of practice, principles and prejudices. 1. ed. Massachusstes : Addison-Wesley, 5. p. 228.

KOTONYA, G. & SOMMERVILLE, I. Requirements engineering (Processes and techniques). 1. ed. England : J. Wiley & Sons, 1998. p. 282.

MACAULAY, L. A. Requirements engineering. 1. ed. Great Britain : Springer-Verlag London, 1996. p. 202.

MULLER, R. J. Projeto de Banco de Dados. São Paulo, Berkeley, 2002.

REZENDE, D. A. Engenharia de Software e Sistemas de Informação. Rio de Janeiro, Brasport, 2002.

SOMMERVILLE, I. Engenharia de Software. São Paulo, Add