Capítulo 1

Introdução ao desenvolvimento de software

- Conceitos iniciais
- Gerações de computadores
- Gerações de linguagem de programação
- Processo de desenvolvimento
- Modelo de ciclo de vida
- Riscos
- Prototipagem
- Levantamento ou especificação de requisitos

A automação, há algum tempo, tornou-se um diferencial competitivo para as organizações. Tal modernização pode ser entendida como o esforço para transformar as tarefas manuais repetitivas em processos independentes, realizados por máquinas. Com isso, a gestão passou por uma verdadeira revolução: erros que antes eram cometidos por falhas de cálculos agora são quase nulos. As empresas não precisam se voltar tanto para atividades contínuas e podem se dedicar mais ao foco do seu negócio. Os benefícios disso se estendem por toda a cadeia de valor, desde compra de materiais, produção e vendas até entrega de produtos para os clientes e pós-venda.

O desenvolvimento de software é uma atividade muito complexa. Sim, porque não existe uma solução pronta para cada cenário. O trabalho é sempre realizado por pessoas, o que torna o sucesso do projeto diretamente relacionado à competência da equipe e à maneira como se trabalha. Outra dificuldade considerável é o fato de, muitas vezes, não ser adotado um processo definido para apoiar essa tarefa.

A tecnologia da informação passou por uma grande evolução nos últimos anos. Com isso, há exigências contínuas e renovadas de aumento na qualificação dos profissionais da área, o que, consequentemente, favorece o seu desenvolvimento. Afinal, esse segmento é composto pela tríade pessoas, gestão e tecnologia.

O emprego desse tripé levou a uma significativa melhoria no desenvolvimento das companhias chamadas inteligentes, aquelas que necessitam de sistemas tolerantes a falhas e capazes de gerar informações críticas para o processo de decisão. Na década de 1960 as empresas trabalhavam com o conceito de processamento centralizado de dados e muitos dos seus recursos eram direcionados ao CPD (Centro de Processamento de Dados). Os sistemas daquela época rodavam de forma mecanizada e em batch (processamento em lote).

Com o passar do tempo, porém, as corporações perceberam a necessidade e a importância de se basear em informações concretas para tomar suas decisões e, assim, aprimorar a gestão dos negócios. Então, abandonaram o padrão do tradicional processamento de dados e passaram a trabalhar com centro de informações. Nesse modelo, já havia integração dos sistemas, mesmo que ainda existissem algumas redundâncias, ou seja, dados duplicados que levavam ao retrabalho.

Atualmente, as organizações vivem a era da Tecnologia da Informação (TI). Agora, os sistemas são todos integrados, possibilitando a otimização dos processos e a diminuição da redundância de dados. Com isso, é possível melhorar muito o dempenho da empresa, pois os processos se tornam mais estruturados, fato que minimiza o retrabalho (**REZENDE**, 2002).

Denis Alcides Rezende é autor de vários livros sobre Tecnologia da Informação, Sistemas de Informação e Engenharia de Software. Atua como consultor em planejamento de Tecnologia da Informação.

1.1. Conceitos iniciais

Com a crescente necessidade das empresas de contar com informações cada vez mais rápidas e confiáveis, foi necessário desenvolver não apenas sistemas, como também novos hardwares. Era preciso ter máquinas que atendessem às especificações de softwares de alto desempenho e de grande disponibilidade.

1.2. Gerações de computadores

Vamos conhecer agora as cinco gerações de equipamentos, a tecnologia empregada em cada uma delas, suas vantagens e desvantagens.

As cinco gerações de máquinas

HULTON ARCHIVE/GETTY IMAGES

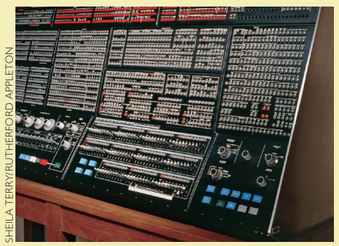
1ª • 1940 - 1958

Univac I: dez vezes mais rápido e com um décimo do tamanho do Eniac, o modelo anterior

2ª • 1958 - 1964

IBM 7090: as válvulas deram lugar aos transistores e a vida útil dos equipamentos aumentou

CERN/SCIENCE PHOTO LIBRARY/LATINSTOCK



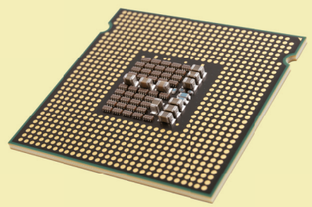
SHEILA TERRY/RUTHERFORD APPLETON LABORATORY/SCIENCE PHOTO LIBRARY/LATINSTOCK

3ª • 1964 - 1971

Com a série 360, da IBM, a velocidade dos equipamentos passou para bilionésimos de segundo

4ª • 1971 - 1987

Chega a era do chip, a lâmina de silício que permite a integração de uma infinidade de circuitos



OLEKSIY MAKSYMENKO/ALAMY/OTHER IMAGES

DIVULGAÇÃO

5ª • 1987

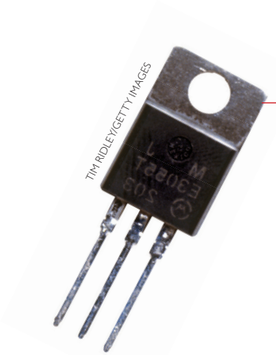
Novas tecnologias revolucionam os conceitos de processamento paralelo e armazenamento de dados

• Primeira (1940-1958)

O primeiro computador digital eletrônico de grande escala foi o Eniac (Electrical Numerical Integrator and Calculator, ou integradora e calculadora numérica elétrica). Começou a ser desenvolvido em 1943, durante a Segunda Guerra Mundial, para auxiliar o Exército dos Estados Unidos nos cálculos de balística. Foi lançado em fevereiro de 1946 pelos cientistas norte-americanos John Presper Eckert e John W. Mauchly, da Electronic Control Company. O Eniac realizava 5 mil operações por segundo, velocidade mil vezes superior à das máquinas da época. No entanto, se comparado com os computadores atuais, o seu poder de processamento seria menor do que o de uma simples calculadora de bolso.

Mas a primeira geração de máquinas teve como marco histórico o lançamento do primeiro computador comercial, o Univac I (Universal Automatic Computer ou computador automático universal), em 1951. Ele possuía cem vezes a capacidade do Eniac, era dez vezes mais rápido e tinha um décimo de seu tamanho. O Univac I tinha como componentes entre 10 mil e 20 mil válvulas eletrônicas, com duração média de oitocentos a mil horas.

O primeiro modelo do Univac foi construído pela empresa Eckert-Mauchly Computer Corporation, adquirida pela Remington Rand pouco depois. Hoje, os direitos sobre o nome Univac pertencem à Unisys, que aponta a Força Aérea Americana, o Exército e a Comissão de Energia Atômica norte-americanos como seus primeiros clientes. Inicialmente, o Univac era usado para executar funções no Escritório de Censo dos Estados Unidos. Além de órgãos governamentais, eram usuárias do computador empresas como a General Electric, a Metropolitan Life e a Du Pont. Naquela época, cada uma das 46 unidades fabricadas do Univac custava US$ 1 milhão. Em 1953, surgiu o IBM 701 e, em 1954, o IBM 650. Ambos tiveram muito sucesso de vendas para a época, chegando a 2 mil unidades em cinco anos.


TIM RIDLEY/GETTY IMAGES

• Segunda (1958-1964)

A fabricação dos computadores foi alterada e as válvulas deram lugar aos **transistores**, cuja vida útil era bem maior (90 mil horas). Com isso, as máquinas ficaram mais rápidas e menores. Pertence a essa segunda geração o IBM 7090, que possuía 40 mil transistores e 1,2 milhão de núcleos magnéticos. Ocupava uma área de 40 m^2, contra a de 180 m^2 do Eniac.

• Terceira (1964-1971)


TIM RIDLEY/GETTY IMAGES

Em 1964, a IBM anunciou o lançamento da série 360 com **circuito integrado**. Os circuitos impressos são milimétricos e podem conter transistores, resistências e condensadores. Para se ter uma ideia, 50 mil desses conjuntos cabem em um dedal. A novidade permitiu um grande aumento na velocidade de computação de dados, passando de milionésimos de segundo para bilionésimos de segundo. Assim, um programa que era executado em uma hora, no modelo de computador de 1951, levava entre três e quatro segundos para rodar nos equipamentos da terceira geração.

• Quarta (1971-1987)

A integração de circuitos teve grandes incrementos, com crescimento de mil vezes a cada dez anos. Foram introduzidas várias escalas de incorporação, definidas pelo número de conjuntos que podem ser colocados em uma única lâmina miniaturizada feita de silício, o famoso chip. Assim, foram surgindo tecnologias consecutivas: em 1970, LSI (Large Scale Integration – integração em larga escala); em 1975, VLSI (Very Large Scale Integration – integração em muito larga escala); e, em 1980, ULSI (Ultra Large Scale Integration – integração em ultralarga escala).

• Quinta (1987- primeira década do século 21)

A geração de equipamentos em uso na primeira década do século 21 surgiu em 1987, com o uso de novas tecnologias, principalmente relacionadas a dispositivos ópticos e a telecomunicações. Houve aumento de processamento paralelo, diversidade de formato, incremento da capacidade computacional e de armazenamento, assim como difusão do processamento distribuído. Além disso, delineou-se a tendência de convergência de computadores e aparelhos de comunicação, o que facilitou a interoperabilidade e a universalização da operação dos sistemas, assim como a sua normatização.

Vantagens e desvantagens das 5 gerações de computadores

Geração	Componente eletrônico	Vantagens	Desvantagens
1ª geração 1940-1958	válvulas eletrônicas	• únicos componentes eletrônicos disponíveis	• grande dimensão • produzem muito calor • necessitam de ar condicionado
2ª geração 1958-1964	transistores	• menor dimensão • produzem menos calor • mais rápidos	• ainda necessitam de constante manutenção • necessitam de ar condicionado
3ª geração 1964-1971	circuitos integrados	• ainda menor dimensão • menor produção de calor • menor consumo de energia • ainda mais rápidos	• inicialmente com muitos problemas de fábrica
4ª geração 1971-1987	circuitos integrados larga escala	• não é necessário ar condicionado • conservação mínima • alta densidade de componentes	• existem ainda computadores com menos potência em relação a computadores de outras gerações
5ª geração 1987-atual	transdutores e circuitos em paralelo	• maior densidade de componentes • reduzido tamanho • auto-regeneração • grande fiabilidade e velocidade • multiprocessamento	• maior complexidade • ainda muito caros

1.3. Gerações de linguagem de programação

É importante conhecer as gerações de linguagens de programação, para entender bem o contexto atual (**SWEBOK**, 2004).

O Swebok (Software Engineering Body of Knowledge, traduzido por Áreas do Conhecimento da Engenharia de Software) é uma iniciativa da Sociedade da Computação do Instituto de Engenharia Elétrica e Eletrônica (IEEE Computer Society), com o propósito de criar um consenso sobre as áreas de conhecimento da engenharia de software. Publicado em 2004, contou com a participação da indústria internacional, de sociedades de profissionais, da academia e de diversos autores consagrados.

Primeira (1GL)

A primeira geração de programação utiliza apenas linguagem de máquina, ou seja, o sistema binário de 0 (zero) e 1 (um) para o desenvolvimento dos softwares. Sua desvantagem é ser pouco intuitiva, pois não utiliza linguagens mais sofisticadas que permitem a portabilidade do programa, isto é, o código utilizado acaba restrito a um único tipo de hardware e à arquitetura utilizada.

Segunda (2GL)

A linguagem de programação chamada Assembly representa a segunda geração. Mais próxima do ser humano do que da máquina (como acontecia na 1GL), cada Assembly ainda é bastante associada à arquitetura do computador, fazendo com que a 2GL também seja pouco portável entre ambientes.

Terceira (3GL)

A terceira geração das linguagens de programação está muito próxima do ser humano, pois é facilmente entendida por uma pessoa com pouco – ou nenhum – conhecimento de informática. Isso porque é similar à comunicação do dia a dia. Essa geração é representada pelas linguagens Cobol, Fortran, Algol, Basic, C, C++, entre outras. Para mais informações sobre a 3GL veja quadro abaixo.

Quarta (4GL)

A linguagem SQL (Structured Query Language, ou Linguagem de Consulta Estruturada) tornou-se tão popular que passou a representar toda a quarta geração. Ainda hoje, é bastante utilizada como linguagem de manipulação e consulta de banco de dados, como o SQL-Server da Microsoft, Oracle Database da Oracle e MySQL da Sun.

A principal característica das linguagens de quarta geração é descrever o que deve ser feito, permitindo ao programador visar um resultado imediato. São consideradas capazes de, por si sós, gerar códigos, ou seja, os RADs (Rapid Application Development, ou Desenvolvimento Rápido de Aplicação), com os quais podem ser criadas aplicações, mesmo sem se especializar na linguagem. Também nesse grupo estão as linguagens orientadas a objetos.

Quinta (5GL)

A quinta geração ainda está pouco desenvolvida e engloba as linguagens para inteligência artificial. Sua maior representante é a LISP, nome originado da expressão LISt Processing (Processamento em Lista), já que a lista é a sua estrutura de dados fundamental.

Para desenvolver um software pode-se utilizar uma geração de linguagem ou um conjunto delas. Entretanto, o melhor resultado no projeto final é obtido quando se vence o desafio de adequar a programação aos sistemas de informação de diversas áreas do conhecimento.

Saiba mais sobre as 3GL

Cobol, sigla para COmmon Business Oriented Language (Linguagem Orientada aos Negócios): usada em sistemas comerciais, financeiros e administrativos para empresas e governos. Foi criada em 1959, durante o CODASYL (Conference on Data Systems Language, a Conferência de Linguagem de Sistemas de Dados), um dos três comitês propostos em uma reunião no Pentágono, organizado por Charles Phillips, do Departamento de Defesa dos Estados Unidos. As fontes de inspiração são as linguagens FLOW-MATIC, inventada por Grace Hopper, e COMTRAN, da IBM, inventada por Bob Bemer.

Fortran, acrônimo para a expressão *IBM Mathematical FORmula TRANslation System* (Sistema de Tradução de Fórmula Matemática da IBM): família desenvolvida a partir dos anos 1950. Usada, principalmente, em Ciência da Computação e Análise Numérica, foi a primeira linguagem de programação imperativa, criada para o IBM 704, entre 1954 e 1957, por uma equipe chefiada por John W. Backus.

Basic, sigla para *Beginners All-purpose Symbolic Instruction Code* (Código de Instrução Simbólico para Todos os Propósitos de Iniciantes): criada com fins didáticos, pelos professores John George Kemeny e Thomas Eugene Kurtz, em 1964, no Dartmouth College. Também é o nome genérico de uma extensa família de linguagens de programação derivadas do Basic original.

C: compilada, estruturada, imperativa, processual, de alto nível e padronizada. Foi criada em 1972, por Dennis Ritchie, no AT&T Bell Labs, como base para o desenvolvimento do sistema operacional UNIX (escrito em Assembly originalmente).

C++: de alto nível, com facilidades para o uso em baixo nível, multiparadigma e de uso geral. Desde os anos 1990, é uma das linguagens comerciais mais populares, mas disseminada também na academia por seu grande desempenho e base de utilizadores. Foi desenvolvida por Bjarne Stroustrup (primeiramente, com o nome C with Classes, que significa C com classes, em português), em 1983, no Bell Labs, como um adicional à linguagem C.

Roger S. Pressman indica sete áreas ou categorias potenciais de aplicação de softwares em seu livro Engenharia de Software (PRESSMAN, 2006). Para saber mais sobre essas sete áreas, consulte o quadro abaixo.

Roger S. Pressman é reconhecido internacionalmente como uma das maiores autoridades em engenharia de softwares. Trabalha como desenvolvedor, professor, escritor e consultor.

1.4. Processo de desenvolvimento

Na área de tecnologia da informação, desenvolvimento de sistemas significa o ato de elaborar e implementar um programa, ou seja, entender as necessidades dos usuários e transformá-las em um produto denominado software, de acordo com a definição de **Ana Regina Cavalcanti da Rocha**, no seu livro *Qualidade de Software: Teoria e Prática* (DA ROCHA, 2004).

Ana Regina Cavalcanti da Rocha é mestra e doutora em Informática; atua na área de Ciência da Computação, com ênfase em Engenharia de Software.

Os processos de desenvolvimento são essenciais para o bom andamento do projeto de um software, assim como a compreensão do sistema como um todo. Quanto mais aumenta a complexidade dos sistemas, mais difícil se torna a sua visibilidade e compreensão, portanto, sem um processo bem definido o projeto tem grande chance de insucesso.

O processo envolve atividades necessárias para definir, desenvolver, testar e manter um software. Exige inúmeras tentativas de lidar com a complexidade e de minimizar os problemas envolvidos no seu desenvolvimento. E devem ser levados em consideração fatores críticos: entrega do que o cliente deseja, com a qualidade, o prazo e o custo acordados (**PMI**, 2004).

A terceira edição do PMBOK® - Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos saiu em 2004. É uma publicação do PMI (Project Management Institute, o Instituto de Gerenciamento de Projetos, em português), para identificar e descrever as boas práticas de projetos que agreguem valor e sejam fáceis de aplicar.

Para assegurar qualidade e padrão aos projetos, devem ser seguidos modelos de desenvolvimento de software: em cascata, iterativo ou incremental e prototipagem. Também é importante levar em conta o ciclo de vida de um programa, desde a sua concepção até a sua extinção. Logo, é necessário saber quanto tempo o software ficará em funcionamento e quais os benefícios gerados durante esse período, sempre tendo em mente o retorno do investimento (**FOWLER**, 2009).

Chad Fowler, autor de vários livros, é um dos mais competentes desenvolvedores de software do mundo. Trabalhou para várias das maiores corporações do planeta. Vive na Índia, onde mantém um centro internacional de desenvolvimento.

É crucial entender perfeitamente os entraves envolvidos no desenvolvimento de um software. Embora não exista uma conduta padrão, é sempre bom partir do princípio de que se está em busca de soluções para os problemas do cliente.

Com a definição de objetivos, atividades e participantes, consegue-se a excelência no desenvolvimento de software, pois há gestão, controle e padronização do processo. Tudo isso diminui os riscos de problemas com cronograma, treinamento, qualidade e aceitação do sistema pelos usuários. Vejamos quais são os objetivos, as atividades e os participantes.

1.4.1. Objetivos

Definição: alinhar todas as atividades a executar no decorrer de todo o projeto e, assim, desenvolver tudo o que foi previsto no seu escopo.

Planejamento: definir, em um cronograma, quando, como (quais os recursos de hardware e de software necessário) e quem irá realizar determinada tarefa.

Controle: ter sempre meios de saber se o cronograma está sendo cumprido e criar planos de contingência caso haja problemas no fluxo.

Padronização: seguir as mesmas metodologias por todos os envolvidos no projeto para não haver dificuldades de entendimento.

As sete áreas, por Pressman

- **Software básico ou de sistema:** conjunto de programas desenvolvidos para servir a outros sistemas, como os operacionais e os compiladores.
- **Sistemas de tempo real:** programas que monitoram, analisam e controlam eventos reais, no momento em que ocorrem. Recebem informações do mundo físico e, como resultado de seu processamento, enviam de volta informações de controle. Devem ter um tempo determinado de resposta para evitar efeitos desastrosos.
- **Sistemas de informação:** softwares da maior área de aplicação de sistemas, que engloba os programas de gerenciamento e acesso a bases de dados de informações de negócio.
- **Software de engenharia ou científico:** sistemas utilizados em áreas como astronomia, análise de resistência de estruturas, biologia molecular etc.
- **Sistemas embarcados ou software residente:** programas alojados nas ROM (Read-Only Memory, ou Memória Apenas de Leitura) e que controlam sistemas de baixo nível.
- **Software de quarta geração:** programas como processadores de texto, planilhas eletrônicas, jogos, aplicações financeiras e acesso a bases de dados.
- **Software de inteligência artificial (IA):** sistemas que utilizam algoritmos não numéricos para resolver problemas complexos, também conhecidos como sistemas baseados em conhecimento.

1.4.2. Atividades

Levantamento de requisitos: entender a necessidade do cliente e as regras do seu negócio (é a fase mais importante do desenvolvimento).

Análise de requisitos: definir o que fazer sob o ponto de vista de análise de sistemas.

Projeto: desenvolver o sistema já com cronograma, necessidades e riscos pré-estabelecidos.

Implementação: começar a usar um novo processo.

Testes: analisar se todas as funcionalidades solicitadas pelo cliente no levantamento de requisitos estão funcionando corretamente.

Implantação: disponibilizar os processos para utilização pelo usuário final.

1.4.3. Participantes

Gerentes de projeto: entram em contato com o cliente para levantar suas necessidades, assumindo a responsabilidade pelo cumprimento das fases de desenvolvimento e do cronograma.

Analistas de sistema: elaboram o projeto do sistema, utilizando UML (Unified Modeling Language, ou Linguagem de Modelagem Unificada), ferramentas de modelagem de processos, técnicas de análises de sistemas e técnicas de projetos de sistemas.

Ichak Adizes é o criador da metodologia que leva seu nome e visa ao diagnóstico e à terapêutica para mudanças organizacionais e culturais. O método está sendo aplicado – com muito sucesso – em organizações de 30 mil a 90 mil empregados de diversos países.

Arquitetos de software: definem a arquitetura em que o sistema funcionará (web, Windows, Linux etc.), com conhecimento dos pontos fortes e fracos de cada ambiente de acordo com as necessidades do cliente.

Programadores: codificam, em linguagem de programação, os requisitos do sistema, elaborados pelo analista de sistemas nos modelos UML.

Clientes: solicitam o desenvolvimento em entrevistas durante as quais serão definidos os requisitos do sistema.

Avaliadores de qualidade: realizam os testes do sistema, utilizando – ou não – ferramentas como o JTest.

1.5. Modelo de ciclo de vida

O ciclo de vida, em sistemas informatizados, tem as mesmas etapas do ciclo de vida de um ser humano (**ADIZES**, 1999) (figura 1). O namoro é considerado o primeiro estágio do desenvolvimento, quando o sistema ainda nem nasceu e existe apenas como uma ideia. Ainda não existe fisicamente, é apenas uma possibilidade. Trata-se, portanto, de um período em que se fala muito e se age pouco. O analista de sistemas tem, nessa etapa, uma função muito importante: entender o que o cliente deseja e, a partir daí, criar um compromisso com ele.

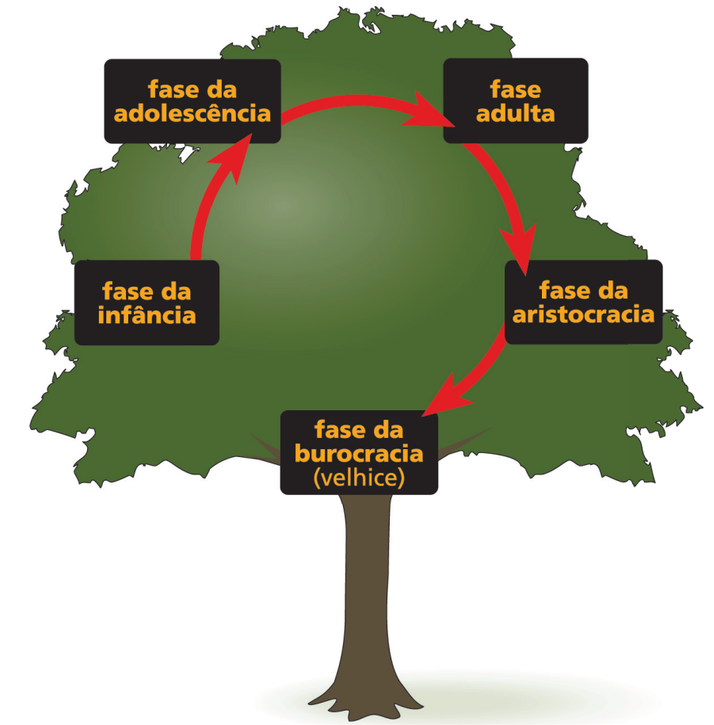


Figura 1
A imagem ao lado mostra que o ciclo de vida dos sistemas informatizados pode ser comparado ao dos seres humanos.

Já a fase da infância, também chamada de sistema-criança, conta com poucos controles formalizados e muitas falhas a serem transformadas em virtudes. Normalmente, o sistema é precário, faltam registros e informações, há resistência das pessoas em fazer reuniões de aprimoramento. Alguns chegam a acreditar que o sistema não vai funcionar.

A adolescência é o estágio do renascimento. Nessa etapa, ainda conforme Adizes, é eliminada grande parte dos erros encontrados na fase anterior. Essa transição é caracterizada por conflitos e inconsistências, muitas vezes causados pelos próprios usuários, os quais ainda não se comprometem a realizar as interações pertinentes. Mesmo percebendo a necessidade de delegar autoridade, mudar metas e liderança, os responsáveis enfrentam dificuldades, pois muitos usuários ainda acreditam que o antigo sistema era melhor.

A estabilidade, ou fase adulta, é o início do estágio de envelhecimento do sistema, ou seja, quando começa a se tornar obsoleto e surgem outros melhores. Um sintoma bem visível é a perda de flexibilidade. Todos começam a achar que ele não funciona tão bem, atribuindo-lhe falhas. É um estágio marcado pelo fim do crescimento e início do declínio.

A aristocracia – também batizada de meia-idade ou melhor idade – é a fase da vaidade, do vestir-se bem e de falar bem. Maria Aparecida Maluche explica que a ênfase está em como as coisas são feitas e não no porquê (MALUCHE, 2000).

É aí que se inicia a etapa de declínio total do sistema, na qual o nível de inovação é baixo e tudo deixa a desejar.

Na fase da burocracia, ou velhice, o sistema perde a funcionalidade e a elasticidade. Com isso, ninguém mais tem confiança nele. Muitos percebem a situação, mas ninguém faz nada, culpando o sistema por todos os erros e falhas na organização. O declínio se intensifica e, mesmo que permaneça em uso por alguns anos, a decadência prossegue, até a morte do sistema (ADIZES, 1999).

As agendas ficam superlotadas, começa-se a perder o controle de prazos e de qualidade. Ao mesmo tempo que se assume muitos compromissos, comete-se falhas. Por isso, deve-se ficar atento às reclamações dos clientes e tentar, de qualquer maneira, atender às necessidades percebidas.

A sequência de etapas

1. **Definição dos objetivos:** finalidade do projeto e sua inscrição dentro de uma estratégia global.
2. **Análise das necessidades e viabilidade:** identificação, estudo e formalização do que o cliente precisa e do conjunto de entraves.
3. **Concepção geral:** elaboração das especificações da arquitetura geral do software.
4. **Concepção detalhada:** definição precisa de cada subconjunto do software.
5. **Codificação, aplicação ou programação:** tradução em linguagem de programação das funcionalidades definidas nas fases de concepção.
6. **Testes unitários:** verificação individual de cada subconjunto do software, aplicados em conformidade com as especificações.
7. **Integração:** certificação e documentação da intercomunicação dos diferentes elementos (módulos) do software.
8. **Qualificação ou receita:** checagem da conformidade do software às especificações iniciais.
9. **Documentação:** registro das informações necessárias para a utilização do software e para desenvolvimentos anteriores.
10. **Produção:** colocação do sistema em operação para o cliente.
11. **Manutenção:** ações corretivas (manutenção corretiva) e evolutivas (manutenção evolutiva) no software.

O ciclo de vida de um software (*software lifecycle*) designa todas as etapas do seu desenvolvimento, da concepção à extinção. Essa segmentação tem por objetivo definir pontos intermediários que permitam checar a conformidade do sistema com as necessidades expressas no escopo do projeto e verificar o processo de desenvolvimento.

Segundo Ana Regina Cavalcanti da Rocha, em geral o ciclo de vida do software compreende, no mínimo, onze atividades (DA ROCHA, 2004). A sequência e a presença de cada uma delas no ciclo de vida dependem da escolha do modelo a ser adotado pelo cliente e pela equipe de desenvolvimento.

Os modelos de ciclo de vida descrevem as etapas do processo de desenvolvimento de software, assim como as atividades a serem realizadas em cada uma delas. A definição desses estágios permite estabelecer pontos de controle para a avaliação da qualidade e da gestão do projeto. Veja o quadro *A sequência das etapas*.

1.5.1. Modelo em cascata ou *waterfall*

Cascata ou *waterfall* (em inglês) é um dos mais simples e conhecidos modelos para o desenvolvimento de software. Criado em 1966 e formalizado em 1970, tem como principais características (FOWLER, 2009):

- Projetos reais raramente seguem um fluxo sequencial.
- Assume que é possível declarar detalhadamente todos os requisitos antes do início das demais fases do desenvolvimento (podendo ter a propagação de erros durante as fases do processo).
- Uma versão de produção do sistema não estará pronta até que o ciclo de desenvolvimento termine.

As fases são executadas sistematicamente, de maneira sequencial, conforme sugere a figura 2.

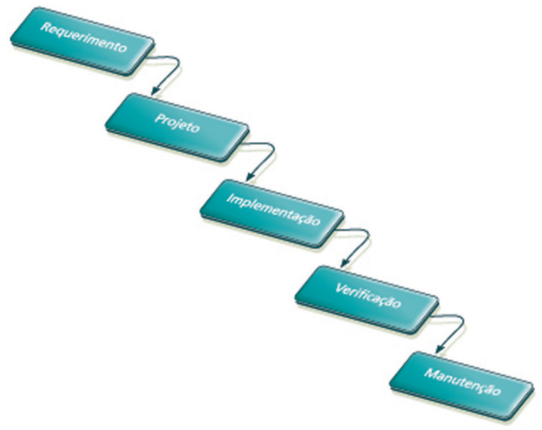


Figura 2
As fases sequenciais do Modelo cascata.

Requerimento (Análise e Definição de Requisitos): as funções, as restrições e os objetivos do sistema precisam ser definidos, via consulta aos usuários, para que sejam elaborados os detalhes específicos.

Projeto (Sistemas e Software): o processo deve agrupar os requisitos de hardware e software, assim como identificar e descrever as abstrações fundamentais do sistema e as suas relações.

Implementação (Testes de Unidade): o projeto é posto à prova como um conjunto de programas – ou de suas partes –, com o teste de cada item, para verificar se aquela unidade atende à sua especificação.

Verificação (Integração e Teste de Sistemas): antes da entrega do software ao cliente, as unidades ou programas individuais são integrados e testados como um sistema completo, de maneira a assegurar que todos os requisitos estejam contemplados.

Manutenção (Operação): o sistema é instalado e colocado em operação, sendo detectados e corrigidos erros não descobertos antes, o que melhora a implementação e também permite descobrir novos requisitos.

O sistema só é entregue ao usuário depois que os desenvolvedores observarem um resultado satisfatório na fase de verificação, com certo grau de confiança. Mesmo assim, o software ainda poderá ter alterações, principalmente para correção de falhas que só os usuários conseguem detectar no dia a dia e também para mais bem adaptá-lo ao ambiente ou ainda por problemas de desempenho. Se houver transformações no negócio da empresa, também será preciso realizar mudanças. A manutenção do software (última etapa proposta pelo modelo em cascata) pode requerer, portanto, voltar a atividades das fases anteriores do ciclo. É desse jeito que se garante a melhoria contínua do produto.

1.5.2. Modelo em cascata evolucionário ou *waterfall* evolucionário

As metodologias baseadas no modelo *waterfall* presumem que uma atividade deve ser concluída antes do início da próxima. Devido às limitações, houve reflexões no sentido de utilizar o modelo de maneira mais flexível. Assim, em uma situação extrema, aconteceriam simultaneamente todas as atividades do ciclo de vida em cascata, enquanto em outra circunstância também limite haveria o uso da abordagem sequencial, ou seja, concluir inteiramente uma fase antes de partir para a seguinte (FOWLER, 2009). A proposta é evoluir com base em protótipo inicial (figura 3). É fundamental que os requisitos sejam muito bem compreendidos. Mesmo com uma completa visão das especificações iniciais, é preciso trabalhar com o cliente durante todo o desenvolvimento. Todos os conceitos e ideias vão sendo materializados em requisitos, durante o processo, até que se chegue ao produto idealizado.

Conceitos e ideias vão sendo materializados, à medida que o trabalho evolui, até chegar ao produto desejado. O objetivo é trabalhar com o cliente durante toda a fase de desenvolvimento, com os requisitos muito bem compreendidos. Entre as vantagens dessa abordagem estão a possibilidade de antecipar o produto final para avaliação do cliente, de manter uma comunicação entre os desenvolvedores

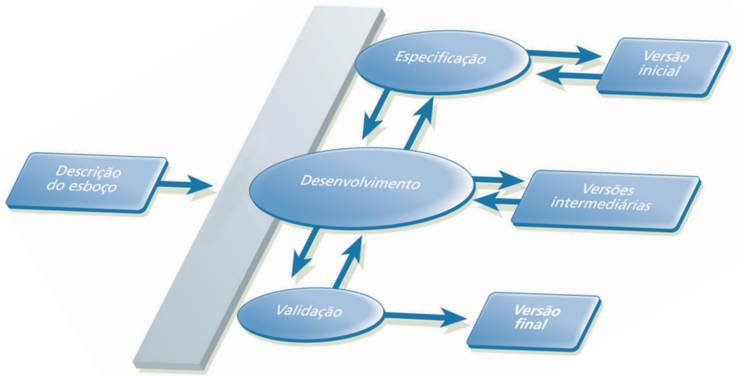


Figura 3
O modelo evolucionário.

e usuários para solucionar problemas técnicos e de começar precocemente o treinamento dos usuários. Isso antes de concluir as suas diferentes versões (inicial, intermediária e final), com espaço para readequações devidamente avalizadas pelo usuário. O modelo se baseia, então, no conceito da implementação inicial, cujo resultado é analisado – e comentado – pelo usuário. A partir desse *feedback*, o sistema é aprimorado por meio de muitas versões, até o seu desenvolvimento completo. A especificação, o desenvolvimento e a validação são executados concomitantemente para possibilitar um retorno rápido.

A abordagem mais radical do ciclo de vida em cascata é aquela em que todas as atividades se desenvolvem paralelamente, desde o início do projeto. Ao contrário, a mais conservadora é a que preconiza completar toda atividade N antes de iniciar a próxima N + 1. Mas há um número infinito de opções entre esses extremos. Segundo Edward **Yourdon**, a escolha entre as diversas possibilidades disponíveis depende de uma série de variáveis como é possível ver no quadro abaixo, que incluem do nível de estabilidade do usuário às exigências de produção (YOURDON, 1995).

Edward Yourdon é desenvolvedor de metodologias de engenharia de software, além de autor de 25 livros sobre computadores, incluindo best-sellers.

Variáveis de Yourdon

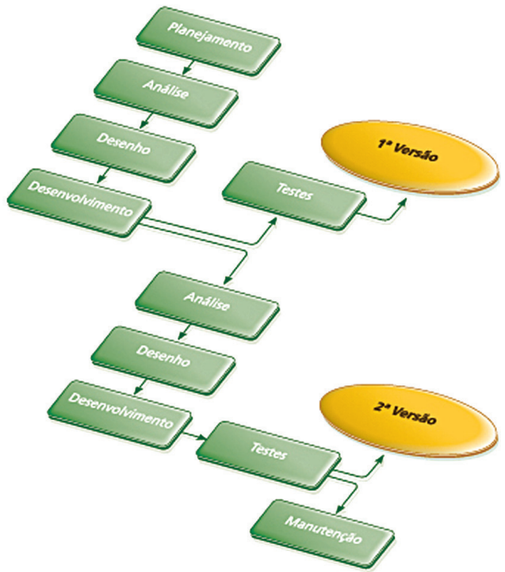
- **Nível de estabilidade do usuário:** se ele é instável ou inexperiente e não sabe definir suas reais necessidades, é preciso usar uma abordagem mais radical.
- **Nível de urgência na produção de resultados tangíveis e imediatos:** se o prazo é curto, por questões políticas ou outras pressões externas, justifica-se assumir uma abordagem radical. Nesse caso, é preferível ter 90% do sistema completo na data especificada do que 100% das etapas de análise e projeto.
- **Exigências para a produção de cronogramas, orçamentos, estimativas etc.:** a maior parte dos grandes projetos requer estimativas de pessoal, recursos de computação e outros detalhes, e tudo isso depende de levantamento e análise. Portanto, quanto mais detalhadas e precisas forem as estimativas, é provável que seja necessária uma abordagem conservadora.

1.5.3. Modelo incremental

O modelo incremental resulta da combinação do linear (em cascata) com o de protótipos (evolutivo). O seu desenvolvimento é dividido em etapas, denominadas incrementos, os quais conduzem aos aprimoramentos necessários, até que se chegue à versão final (FOWLER, 2009). Em cada incremento ocorre todo o ciclo de desenvolvimento de software, do planejamento aos testes. Por essa razão, ao final de uma etapa, já é possível obter um sistema totalmente funcional, embora não contemple todos os requisitos do produto final (figura 4).

Entre as vantagens dessa metodologia está justamente a possibilidade de definir melhor os requisitos, que muitas vezes não ficam muito claros no início do processo. O primeiro passo é trabalhar o núcleo do sistema, ou seja, são implementados os requisitos básicos, sem os detalhes. Em seguida o produto é avaliado pelo usuário, para a detecção de problemas iniciais, os quais já podem ser corrigidos, caso contrário poderiam assumir grandes proporções ao surgirem apenas na versão final. Outro benefício é o fato de o cliente esclarecer os requisitos e as prioridades para os próximos incrementos, além de poder contar com as facilidades da versão já produzida. Assim, com a construção de um sistema menor (sempre muito menos arriscada do que o desenvolvimento de um sistema grande), se um grave erro é cometido, ou se existe uma falha no levantamento de requisitos, somente o último incremento é descartado. Ganha-se em performance, pois são mínimas as chances de mudanças bruscas nos requisitos, e em tempo de desenvolvimento.

Figura 4
O modelo incremental.



1.5.4. Modelo iterativo

O método iterativo foi criado também com base nas vantagens e limitações dos modelos em cascata e evolutivo. Mantém a ideia principal do desenvolvimento incremental, com o acréscimo de novas capacidades funcionais, a cada etapa, para a obtenção do produto final. No entanto, segundo Fowler, modificações podem ser introduzidas a cada passo realizado. Mais informações sobre o modelo interativo no quadro abaixo.

Uma iteração de desenvolvimento abrange as atividades que conduzem até uma versão estável e executável do software. Por esse motivo, pode-se concluir que ela passa pelo levantamento de requisitos, desenvolvimento, implementação e testes. Ou seja, cada iteração é como um miniprojeto em cascata no qual os critérios de avaliação são estabelecidos quando a etapa é planejada, testada e validada pelo usuário.

Um dos principais ganhos com a adoção desse modelo é a realização de testes em cada nível de desenvolvimento (bem mais fácil do que testar o sistema apenas na sua versão final). Isso pode fornecer ao cliente informações importantes que servirão de subsídio para a melhor definição de futuros requisitos do software.

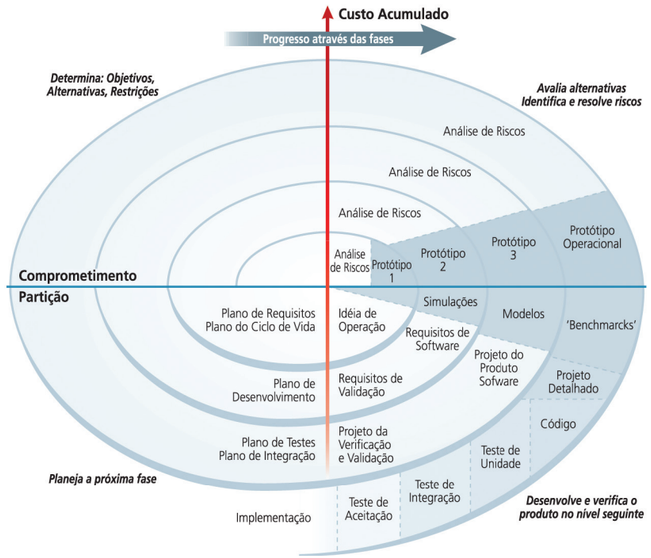
Passos do modelo iterativo

1. implementação inicial
- desenvolver um subconjunto da solução do problema global
- contemplar os principais aspectos facilmente identificáveis em relação ao problema a ser resolvido

2. lista de controle de projeto
- descrever todas as atividades para a obtenção do sistema final, com definição de tarefas a realizar a cada iteração
- gerenciar o desenvolvimento inteiro para medir, em determinado nível, o quão distante se está da última iteração

3. iterações do modelo
- retirar cada atividade da lista de controle de projeto com a realização de três etapas: projeto, implementação e análise
- esvaziar a lista completamente

Figura 5
O modelo espiral.



1.5.5. Modelo espiral

Criado por Barry **Boehm**, em 1988, o modelo espiral (figura 5) permite que as ideias e a inovação sejam verificadas e avaliadas constantemente. Isso porque, a cada interação, existe a volta da espiral que pode ser baseada em um modelo diferente e pode ter diferentes atividades. Esse padrão caracteriza-se, portanto, pelo desenvolvimento em sequência, aumentando a complexidade do processo conforme se chega mais próximo do produto final.

Barry W. Boehm é considerado um guru da Universidade do Sul da Califórnia, da qual é professor emérito, e tem lugar garantido entre os estudiosos que mais influenciaram o pensamento econômico a respeito de software nos últimos 40 anos. Ele se graduou em Harvard em 1957, é mestre e Ph.D. em Matemática e foi diretor do Software and Computer Technology Office do Departamento de Defesa dos Estados Unidos. Desde 1964 Boehm escreveu seis livros, entre eles *Software Risk Manager*, publicado pela IEEE Computer Society Press em 1989.

Em cada ciclo da espiral, uma série de atividades é realizada em uma determinada ordem, como comunicação com o cliente, planejamento, análise de riscos e avaliação dos resultados (FOWLER, 2009).

O modelo espiral é cíclico. Considerado um metamodelo, abrange diferentes padrões, desde o cascata até vários tipos de protótipos. Com isso, é possível prever os riscos e analisá-los em várias etapas do desenvolvimento de software.

1.6. Riscos

O risco do projeto é um evento ou uma condição incerta que, se ocorrer, terá um efeito positivo ou negativo sobre, pelo menos, um objetivo do projeto, como tempo, custo, escopo ou qualidade. E o modelo espiral foi a primeira proposta de inclusão de **Gerência de Risco** no ciclo de vida de desenvolvimento de software. A interatividade e os riscos, portanto, são as suas principais características (PMBOK, 2004). Aliás, como bem lembra Tom Gilb: "Se você não atacar os riscos [do projeto] ativamente, então estes irão ativamente atacar você".

O PMBOK define Gerência de Risco como todos os processos necessários para identificar, analisar, responder, monitorar e controlar os riscos de um projeto. E Dalton Valeriano, no seu livro Moderno Gerenciamento de Projetos, reforça o conceito ao afirmar que essa gestão deve ser constante durante todo o projeto, tendo como objetivos maximizar a probabilidade de riscos favoráveis e minimizar os negativos (VALERIANO, 2005).

1.6.1. Atividades do gerenciamento de riscos

Segundo o PMBOK, 2004, o processo de gerenciamento de riscos para desenvolvimento de software é composto por quatro atividades (figura 6):

1. **Identificação dos riscos:** reconhecimento do projeto, do produto e dos riscos de negócio envolvidos no desenvolvimento do software.

2. **Análise dos riscos:** estudo das possibilidades e das consequências da ocorrência dos riscos, com determinação de valores qualitativos para as possibilidades (muito baixo, baixo, moderado, alto e muito alto) e para as consequências (insignificante, tolerável, sério e catastrófico).

3. **Planejamento do risco:** medição de cada risco e desenvolvimento de uma metodologia de gerenciamento dentre algumas opções: refutar o risco quando a probabilidade é reduzida; minimizar o risco quando há baixo impacto no projeto de desenvolvimento de software ou no produto final; ou planos de contingência para eliminar o risco quando eles se tornarem realidade.

4. **Monitoramento do risco:** identificação e avaliação regular de cada risco para tomada de decisão quanto à diminuição, estabilidade ou aumento da possibilidade de ocorrência do evento; assim como discussão em cada reunião de avaliação do progresso do projeto sobre a permanência efetiva dos esforços para manter cada risco sob controle.

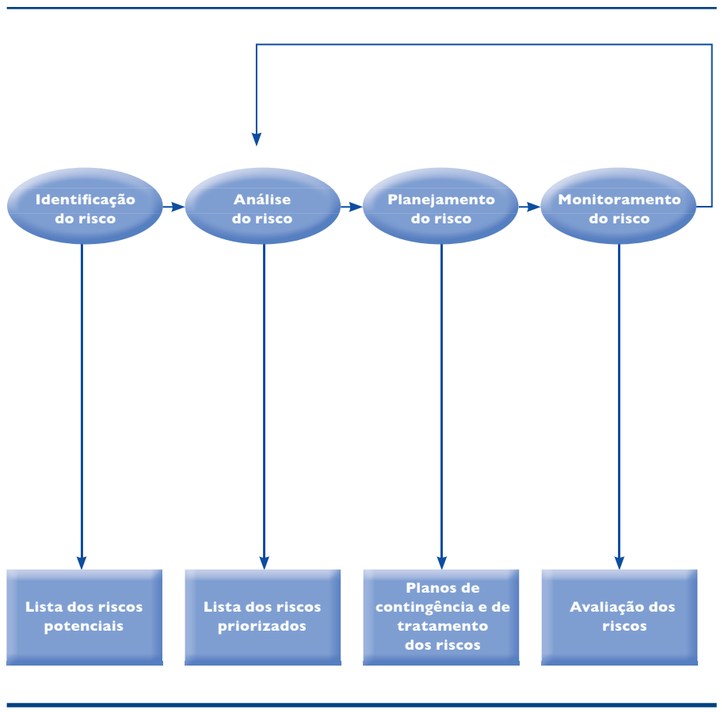


Figura 6
Atividades do processo de gerenciamento de riscos.

Para facilitar o desenvolvimento de um software, podem ser utilizadas ferramentas que auxiliem na construção de modelos, na integração do trabalho de cada membro da equipe, no gerenciamento do andamento do desenvolvimento etc. Os sistemas utilizados para dar esse suporte são normalmente chamados de CASE, sigla em inglês para Computer-Aided Software Engineering (Engenharia de Software de Suporte Computacional). Além dessas ferramentas, outros instrumentos importantes são aqueles que fornecem suporte ao gerenciamento, como desenvolvimento de cronogramas de tarefas, definição de alocações de verbas, monitoramento do progresso e dos gastos, geração de relatórios de gerenciamento etc.

1.7. Prototipagem

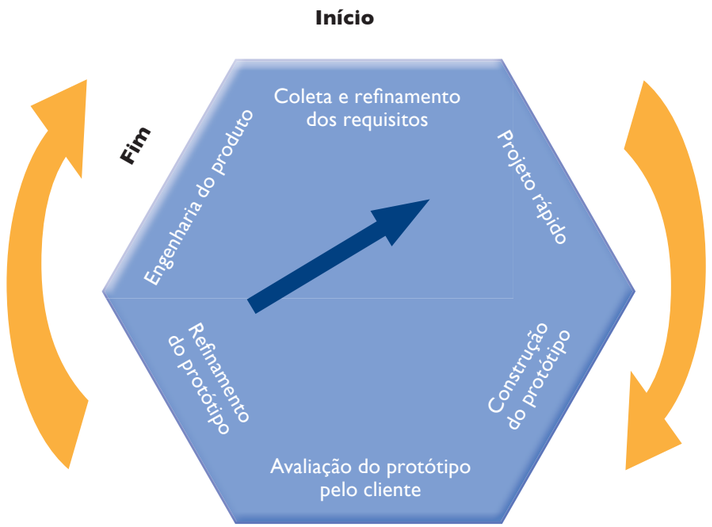
Fazer um protótipo de software garante a possibilidade de examinar antecipadamente os requisitos do programa (figura 7). Ou seja, **desenvolve-se** um exemplar simplificado, de forma rápida, para que as questões relativas a requisitos sejam entendidas ou esclarecidas. Com ele é possível melhorar muito a comunicação com o cliente, pois, primeiramente, ele é ouvido e, então, é feito um desenho e a construção do modelo. E é só depois de cumpridas essas três tarefas que se dá a validação, o que aumenta, em muito, as chances de sucesso do projeto (FOWLER, 2009).

Portanto, ao criar um protótipo, previnem-se possíveis deficiências no projeto de desenvolvimento de software. Em um número grande de casos, o usuário fica insatisfeito com o produto acabado. Isso ocorre por não ter havido troca de informação suficiente entre cliente e desenvolvedor. A comunicação no levantamento de requisitos é falha. Um protótipo deve ser utilizado como instrumento de análise, com a finalidade de superar as dificuldades de comunicação existentes entre o analista de sistemas e o usuário do sistema.

A prototipagem é uma técnica que pode ser empregada em pequenos ou grandes sistemas, em que os requisitos não estão claramente definidos. Seu uso é aconselhavel em projetos para os quais o usuário não saiba exatamente o que deseja (FOWLER, 2009).

Alguns fatores devem ser levados em conta. Um protótipo é um esboço de alguma parte do sistema e a prototipagem é uma técnica complementar à análise de requisitos, com o objetivo de assegurar que as demandas foram bem entendidas. Já a construção desses protótipos utiliza ambientes com facilidades para a construção de interface gráfica, então alguns SGBDs (Sistemas Gerenciadores de Banco de Dados) também fornecem ferramentas para a concepção de telas de entrada e saída de dados.

Figura 7
As fase da prototipagem.



Essa técnica é frequentemente aplicada quando existem dificuldades no entendimento dos requisitos do sistema e/ou quando existem requisitos que precisam ser mais bem entendidos.

Após o Levantamento de Requisitos (LR) – tema do próximo tópico deste livro –, um protótipo já pode ser construído e usado na validação. Os usuários fazem suas críticas e são feitas correções. Esse processo de revisão e refinamento continua até que o sistema seja aceito pelos usuários. A partir daí, é descartado ou utilizado apenas como uma versão inicial do sistema.

Importante: a prototipagem não substitui a construção de modelos do sistema. Trata-se de uma técnica complementar. E os erros detectados na sua validação devem ser utilizados para modificar e refinar o programa.

1.8. Levantamento ou especificação de requisitos

Obter qualidade nos processos e produtos de engenharia de software não é uma tarefa fácil, pois são vários os fatores que dificultam o alcance desse objetivo. No entanto, nada é mais decepcionante do que desenvolver um sistema que não satisfaça as necessidades dos clientes. Grandes volumes de recursos são gastos e, em muitos casos, os clientes sentem-se frustrados com a versão final do software encomendado. Muitos desses problemas são derivados da falta de atenção na hora de definir e acompanhar a evolução dos requisitos durante o processo de construção do sistema (DA ROCHA, 2004).

Por esse motivo, é necessário realizar o levantamento dos requisitos do sistema. As funcionalidades e os recursos devem ser definidos pelo usuário do sistema, isto é, pela pessoa da empresa que necessita de um programa para solucionar um determinado problema. Então, fica claro que o usuário está no centro de levantamento de requisitos do software. É quem conhece as necessidades a serem supridas pelo sistema e, por isso, precisa ser informado de sua importância no processo.

A falta de cuidado com o levantamento dos requisitos de um sistema pode gerar problemas muito sérios, como: resolução de um problema errado ou de forma errada; funcionamento diferente do esperado; complexidade na utilização e entendimento por parte dos usuários e alto custo (SWEBOK, 2004 – *Guide to the Software Engineering Body of Knowledge*). Tudo isso mostra a importância de empregar bem o tempo para entender o problema e seu contexto e obter, assim, os requisitos certos na primeira vez.

1.8.1. O que é um requisito

Requisito é a condição ou a capacidade que um sistema deve atender. Segundo uma das normas padrões do IEEE (1220, de 1994), um requisito é uma sentença identificando uma capacidade, uma característica física ou um fator de qualidade que limita um produto ou um processo. Isso significa uma condição

ou uma capacitação que um produto (no caso, um software) ou um serviço precisa atender ou ter para satisfazer um contrato, um padrão, uma especificação ou outro documento formalmente estabelecido entre as partes.

Os requisitos estão associados aos principais problemas do desenvolvimento de software, pois, quando não refletem as reais necessidades dos usuários, estão incompletos e/ou inconsistentes. Existem mudanças em requisitos já acordados, e a dificuldade para conseguir um entendimento entre usuários e executores é um dos principais problemas relatados, capaz de provocar retrabalho e, consequentemente, atrasos no cronograma, custos acima do orçamento e insatisfação do cliente (SWEBOK, 2004). Veja as categorias de requisitos no quadro abaixo.

Características

Os requisitos também são agrupados por suas características, em uma gama que inclui desde os necessários aos concisos e completos, entre outros.

Necessário: indica que, se retirado, haverá uma deficiência no sistema. Isto é, o programa não atenderá plenamente às expectativas do usuário. Não deve haver requisitos que seriam apenas interessantes, caso existissem. Ou o requisito é necessário ou é dispensável. Deve ser levado em consideração que cada requisito aumenta a complexidade e o custo do projeto, logo, não podem ser introduzidos de forma espúria.

Não ambíguo: tem apenas uma interpretação. É muito importante prestar atenção na linguagem utilizada para não causar duplo entendimento.

Verificável: não é vago nem genérico e deve ser quantificado para permitir a verificação de uma das seguintes formas: inspeção, análise, demonstração ou teste.

Conciso: para não gerar confusão, cada requisito define, de maneira clara e simples, apenas uma única exigência a ser desenvolvida. Para ser conciso, o requisito não inclui explicações, motivações, definições ou descrições do seu uso. Tais detalhes podem estar em outros documentos.

Alcançável: é realizável a um custo definido.

Completo: não precisa ser explicado ou aumentado, garantindo capacidade suficiente do sistema.

Consistente: não contradiz nem mesmo duplica outro requisito e utiliza os termos na mesma forma que a adotada nos outros requisitos.

Ordenável: tem uma ordem por estabilidade, importância ou ambos.

Aceito: acolhido pelos usuários e desenvolvedores.

DICAS PARA A ESCRITA

Algumas técnicas para escrever requisitos de software:

- **Preferir sentenças curtas.**
- **Utilizar verbos no futuro.**
- **Para cada requisito, avaliar se a definição determina se esse requisito está pronto ou não.**
- **Garantir que todos os requisitos podem ser verificáveis e o modo de fazer essa verificação.**
- **Verificar os requisitos agregados ou inter-relacionados.**
- **Estabelecer sempre o mesmo nível de detalhamento em todos os requisitos.**

Categorias de requisitos

- **Funcionais:** descrevem as funcionalidades e os serviços do sistema e dependem do tipo de software a ser desenvolvido, do número de usuários esperado e do padrão do sistema no qual o programa será utilizado. Exemplos: o sistema deve oferecer diversas maneiras de visualizar os dados, de acordo com o perfil do usuário; os relatórios devem estar disponíveis para impressão de acordo com o nível hierárquico de cada usuário etc.

- **Não funcionais:** definem as propriedades e as restrições do sistema, podendo especificar, por exemplo, quais linguagens de programação, bancos de dados e métodos de desenvolvimento serão utilizados. São mais críticos do que os requisitos funcionais, pois sua definição correta influencia diretamente a performance do sistema a ser desenvolvido.

- **De domínio:** originam-se do domínio da aplicação do sistema e refletem as suas características desse domínio. Podem ser requisitos funcionais ou não funcionais; requisitos funcionais novos; restrições sobre requisitos existentes ou sobre computações específicas.

Tipos

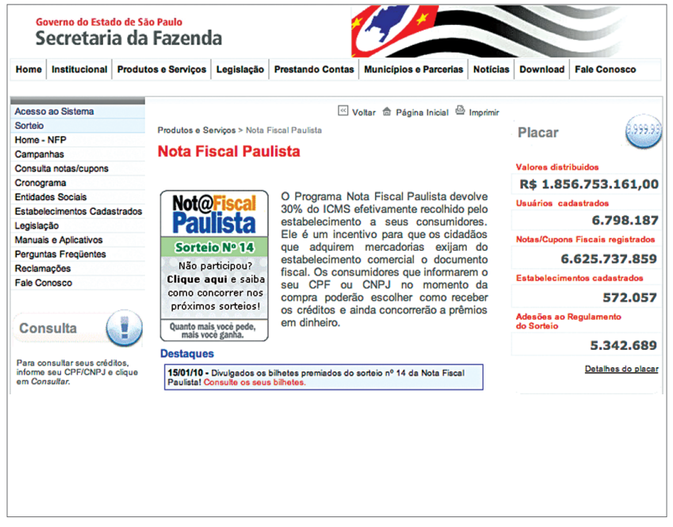
Seja qual for o sistema, há várias formas de atendê-lo. Dependendo das necessidades que se apresentam existem vários tipos de requisito:

- **Do usuário:** é a característica ou o comportamento que o usuário deseja para o software ou para o sistema como um todo. São descritos pelo próprio usuário ou por um analista de sistemas, a partir de um levantamento de dados com o cliente.

- **Do sistema:** comportamento ou característica que se exige do sistema como um todo, incluindo hardware e software. São normalmente levantados por engenheiros de software ou analistas de sistemas, que devem refinar os requisitos dos usuários, transformando-os em termos de engenharia de software.

- **Do software:** comportamento ou característica exigido do software. São normalmente levantados por analistas de sistemas com o objetivo de comparar todos os requisitos com aqueles que possuem real relevância.

Figura 8
Imprimir a Nota Fiscal.



EXEMPLOS DE REQUISITOS

1. O programa deve imprimir nota fiscal e fazer a integração com o sistema da Nota Fiscal Paulista e de vendas registradas (figura 8).

2. O software deve cadastrar novos produtos, gerar relatórios de vendas etc.

3. O sistema deve permitir a realização de vários orçamentos e análise, levando em consideração a condição de pagamento e o prazo de entrega.

Quando um requisito for funcional, deverá ser também:
Independente da implementação: define o que deve ser feito, mas não como deve ser feito.

1.8.2. Como devemos escrever requisitos

Normalmente as especificações de requisitos são escritas em linguagem natural (inglês ou português, por exemplo). Devem ser utilizadas técnicas de padronização para formato, estilo de linguagem e organização das informações. Tudo isso facilita a manipulação desse conjunto de requisitos.

1.8.3. Dependência de requisitos

Os requisitos não são independentes uns dos outros. E a maioria deles só pode ser implementada se outros forem implantados antes (eis o conceito de **requisitos predecessores**). Uma das atividades mais importantes da gerência de requisitos é manter esse relacionamento de dependência, que influenciará todo o processo de desenvolvimento do sistema. Para isso existem algumas soluções possíveis, como manter uma tabela de dependência de requisitos ou manter um banco de dados de requisitos que inclua relações de dependência. Existem alguns produtos no mercado especializados na gerência de requisitos que administram essas dependências.

Por exemplo, é impossível fazer um relatório de vendas sem registrá-las previamente no sistema: o requisito predecessor ao relatório de vendas é o registro das vendas.

É fundamental ter uma atuação constante e muito próxima aos usuários para que qualquer divergência em relação a requisitos ou a adaptações seja facilmente detectada e corrigida. Mas para tanto os usuários precisam estar comprometidos com o desenvolvimento do sistema e se sentirem donos dele. Se isso não ocorrer, há grandes probabilidades de o sistema não ser um sucesso.

1.8.4. Documentação de requisitos

A documentação de requisitos é uma atividade crucial para a engenharia de software, pois o documento gerado nessa fase é uma descrição oficial dos requisitos do sistema para cliente, usuários e desenvolvedores. Isso significa que todos os envolvidos no processo se basearão nesse documento para o desenvolvimento do sistema. Tal documento deve trazer muitas designações: especificação funcional, definição, especificação e responsáveis pelo requisito (FOWLER, 2009).

Ao documentar um requisito, deve-se ter em mente algumas questões que podem influenciar, em muito, todo o processo de desenvolvimento. É necessário se preocupar com a boa qualidade do documento, e alguns cuidados precisam ser observados no modo de escrever.

1. Quem escreve, para quem se escreve e qual o meio utilizado para escrever são três fatores diretamente relacionados à qualidade da documentação dos requisitos.

2. A adoção de linguagem natural – não técnica – no texto, complementada por diagramas, tabelas, fotos e imagens, facilita o entendimento daquilo que se deseja documentar.

3. O grau de detalhamento depende de quem escreve, da organização das informações, do propósito da documentação, entre outros quesitos.

4. A documentação serve para comunicar o que se pretende fazer em um determinado sistema e se ele se dirige a clientes, usuários, gestores e desenvolvedores do sistema.

5. A especificação tem de trazer os serviços que o sistema deve prestar, assim como suas propriedades e restrições impostas à operação e ao desenvolvimento.

6. O documento pode ser tanto sucinto e genérico quanto extenso e detalhado.

1.8.5. Métodos de identificação e coleta

Existem duas técnicas normalmente utilizadas para a identificação e coleta de requisitos: entrevista e reuniões de **JAD**, aquelas das quais participam usuários e analistas, para trabalhar na arquitetura de uma determinada aplicação. Ambas são consideradas as melhores práticas.

JAD (Joint Application Development) é um sistema de desenvolvimento de aplicações em grupo criado pela IBM. O objetivo é reduzir custos e aumentar a produtividade nesses processos.

1.8.5.1. Entrevista

Entre as técnicas mais importantes de identificação de requisitos está a **entrevista**, também presente em outras metodologias, pois, quase sempre, a coleta de informações exige a comunicação direta com os usuários e interessados. Tem como finalidade captar o pensamento do cliente para que se entenda o que é necessário desenvolver. Por essa razão, o entrevistador tem grande responsabilidade. Além de fazer as entrevistas, cabe a ele integrar diferentes interpretações,

Existem formas distintas de entrevista: por questionário, aberta e estruturada.

objetivos, metas, estilos de comunicação e terminologias adequadas aos diversos níveis culturais dos entrevistados.

Por questionário

O questionário é muito usado como técnica de entrevista, principalmente em pesquisas de mercado e de opinião. Sua preparação exige bastante cuidado, e alguns aspectos particulares do processo merecem destaque: emprego de vocabulário adequado para o público entrevistado; inclusão de todos os conteúdos relevantes e de todas as possibilidades de resposta; atenção aos itens redundantes ou ambíguos, contendo mais de uma ideia ou não relacionados com o propósito da pesquisa; redação clara e execução de testes de validade e de confiabilidade da pesquisa.

Uma das principais dificuldades que envolvem esse trabalho é o fato de as palavras possuírem significados distintos para pessoas diversas em contextos diferentes (culturais e educacionais, por exemplo). Em conversações corriqueiras, tais dificuldades de interpretação são esclarecidas naturalmente, mas, em entrevistas com questionários, o treinamento e o método utilizados proíbem essa interação. Por outro lado, tudo isso garante menos ambiguidade, uma das principais vantagens da entrevista por questionário.

Entrevista aberta

A entrevista aberta supre muitas das lacunas dos questionários, porém gera outros problemas. O entrevistador formula uma pergunta e permite ao entrevistado responder como quiser. Mesmo que o primeiro tenha a liberdade de pedir mais detalhes, não há como determinar exatamente o escopo da entrevista. Por isso essa técnica pode gerar dúvidas: as perguntas podem ser de fato respondidas ou a resposta faz parte do repertório normal do discurso do entrevistado? Existem muitas coisas que as pessoas sabem fazer, mas têm dificuldade para descrever, assim como há o conhecimento tácito, de difícil elucidação.

Os benefícios das entrevistas abertas são a ausência de restrições, a possibilidade de trabalhar uma visão ampla e geral de áreas específicas e a expressão livre do entrevistado. Há desvantagens também. A tarefa é difícil e desgastante, mas entrevistador e entrevistado precisam reconhecer a necessidade de mútua colaboração para que o resultado seja eficaz.

Também deve-se levar em conta a falta de procedimentos padronizados para estruturar as informações recebidas durante as entrevistas, uma vez que a análise dos dados obtidos não é trivial. É difícil ouvir e registrar simultaneamente: alguns fatos só tomam determinada importância depois de outros serem conhecidos e, aí, o primeiro pode não ter sido registrado. Por isso vale gravar toda a conversa e transcrevê-la, o que facilita a tarefa de selecionar e registrar o que é relevante para validar com o entrevistado posteriormente.

O relacionamento entre os participantes de uma entrevista deve ser baseado no respeito mútuo e em algumas outras premissas, tais como deferência ao co-


TAXI/GETTY IMAGES

Figura 9
O sucesso da entrevista depende da escolha da pessoa certa para oferecer as respostas precisas.

nhecimento e habilidade do especialista; percepção de expressões não verbais; sensibilidade às diferenças culturais; cordialidade e cooperação.

Entrevista estruturada

A entrevista estruturada (figura 9) tem por finalidade coletar e organizar as informações sobre questões específicas, necessárias para o projeto. É fundamental realizar a entrevista com a pessoa certa, ou seja, quem realmente entende e conhece o negócio e os seus processos. E preparar muito bem o roteiro do que se quer obter, aprofundando o assunto.

A principal vantagem dessa técnica é a obtenção de respostas diretas com informações detalhadas. Todavia, como desvantagem, muitas vezes aspectos relevantes ao projeto de desenvolvimento de software precisam ser identificados antes da realização da entrevista. De maneira geral, requer muito mais trabalho prévio e, quanto melhor for feito, maiores serão as chances de realizar esse trabalho com eficácia.

Preparação

Para tudo que se faz na vida é preciso se preparar. Em relação às entrevistas não é diferente. Deve-se elaborar um roteiro coerente ao alcance dos objetivos do projeto e informar ao entrevistado tanto o propósito da conversa quanto sua importância no processo. Selecionar o entrevistado é outro ponto de atenção, pois somente ao final das entrevistas será possível ter uma visão clara e completa do problema a ser solucionado e das diversas formas de analisar e resolver.

A linguagem precisa se adequar ao entrevistado, levando em consideração seu perfil cultural e tomando muito cuidado com a utilização de termos técnicos,

muitas vezes corriqueiros para o entrevistador, mas eventualmente desconhecidos pelo respondente. Às vezes, por vergonha, essa pessoa não admite que não sabe do que se trata e responde de maneira equivocada. Daí a importância da coerência das perguntas e de observar, com atenção, o real entendimento dos entrevistados. Mais um ponto a cuidar: a programação e o cumprimento dos horários estabelecidos.

Realização

O objetivo central de uma entrevista é conseguir informações relevantes do entrevistado. Para isso é fundamental fazer com que ele não se limite a falar, mas que também reflita bastante sobre cada uma das suas respostas. Por isso o entrevistador deve deixar o respondente totalmente à vontade, sem pressa para acabar o trabalho. Veja o quadro *Questionário bem feito*.

É importante evitar interrupções ocasionadas por fatores externos (telefone, entra-e-sai de pessoas da sala etc.). Tudo isso pode levar o entrevistado a perder o foco.

Comportamento do entrevistador

O entrevistador deve demonstrar claramente as suas intenções para o entrevistado. Primeiramente, precisa estar vestido conforme os costumes da empresa para a qual está trabalhando, de maneira a não causar desconforto aos entrevistados. Até há pouco tempo, consultores e desenvolvedores de software usavam terno, normalmente escuro, intimidando entrevistados não habituados a esse tipo de vestuário.

Requisitos para a documentação

- Visão geral do sistema e dos benefícios decorrentes do seu desenvolvimento.
- Glossário explicativo dos termos técnicos utilizados.
- Definição dos serviços ou dos requisitos funcionais do sistema.
- Definição das propriedades do sistema (requisitos não funcionais), como viabilidade, segurança etc.
- Restrições à operação do sistema e ao processo de desenvolvimento.
- Definição do ambiente em que o sistema operará e as mudanças previstas nesse ambiente.
- Especificações detalhadas, incluindo modelos e outras ferramentas.

Certos movimentos também podem causar desconforto no interlocutor. Balançar os pés ou bater a caneta na mesa são gestos involuntários que podem tirar a atenção do entrevistado e do entrevistador. E, no caso de entrevistas longas, é necessário fixar um breve intervalo.

Documentação

Toda entrevista deve ser documentada logo após sua realização. Desse modo, têm-se os dados arquivados e facilmente recuperáveis para eventuais necessidades. Esse material deve conter informações mínimas, podendo ser ampliado, dependendo da necessidade de cada projeto de desenvolvimento:

- data, hora e local da entrevista;
- nome do entrevistador;
- cargo do entrevistador;
- nome do entrevistado;
- função do entrevistado e descrição do cargo;
- organograma do entrevistado (superior imediato, colegas do mesmo nível, subordinados);
- objetivo da entrevista;
- nomes e títulos de todos os outros presentes na entrevista;
- descrição completa dos fatos tratados e opiniões do entrevistado;
- transcrição da entrevista (opcional);
- conclusões tiradas com base nos fatos e nas opiniões apresentados;
- problemas do negócio levantados durante a entrevista;
- exemplos de relatórios, diagramas, documentos etc.;

DICA

É importante que o desenvolvedor tenha conhecimentos do negócio do cliente e resista a priorizar a tecnologia em relação às suas necessidades.

Questionário bem-feito

- Evitar palavras ambíguas ou vagas que tenham significados diferentes para pessoas distintas.
- Redigir itens específicos, claros e concisos e descartar palavras supérfluas.
- Incluir apenas uma ideia por item.
- Evitar itens com categorias de respostas desbalanceadas.
- Evitar itens com dupla negação.
- Evitar palavras especializadas, jargões, abreviaturas e anacronismos.
- Redigir itens relevantes para a sua pesquisa.
- Evitar itens demográficos que identifiquem os entrevistados.

- desenhos e diagramas elaborados a partir da entrevista (ou durante a conversa);
- comentários relevantes feitos pelo entrevistado;
- todos os números importantes: quantidades, volume de dados etc.

Perguntas para conclusão

Para finalizar a conversa é preciso obter a avaliação do entrevistado sobre a atividade realizada. Essa tarefa exige que ele responda a um formulário contendo as seguintes perguntas:

- A entrevista cobriu todo o escopo necessário?
- Foram feitas perguntas adequadas?
- O entrevistado era realmente a pessoa mais indicada para dar as respostas solicitadas?

1.8.5.2. Metodologia JAD (Joint Application Design)

Outra técnica importante de identificação e coleta de requisitos é o JAD (iniciais de Joint Application Design ou, literalmente, desenho de aplicação associada).

Caso o levantamento de requisitos não seja feito de maneira eficiente – sem a identificação das verdadeiras necessidades do usuário –, ao entregar o software o grupo de informática corre o risco de receber um *feedback* negativo do cliente. Isso porque a percepção desse cliente será de que não recebeu aquilo que solicitou.

Tal problema de comunicação pode ter diversas causas. A adoção de linguagem técnica por ambas as partes é uma das principais razões de erros no levantamento de requisitos, pois tanto o usuário quanto o analista de sistemas podem utilizar termos pertinentes exclusivamente à sua área. Por exemplo, profissionais do departamento financeiro possuem um conjunto de vocábulos técnicos (jargões) desconhecidos dos analistas de sistemas ou que podem levar a um entendimento dúbio. O desconhecimento dos desenvolvedores sobre a área de atuação é outro motivo de equívocos.

Figura 10
A metodologia JAD substitui as entrevistas individuais por reuniões entre usuários e desenvolvedores.

Recomenda-se, portanto, que esses profissionais tenham conhecimentos sobre o negócio do cliente, para estarem mais integrados ao sistema a ser desenvolvido. Também não se deve priorizar a tecnologia em detrimento das necessidades dos usuários e, consequentemente, da solução do problema do cliente. Na fase inicial do projeto, a dificuldade de comunicação entre clientes e equipe de desenvolvimento é a principal causa de defeitos encontrados no produto final (AUGUST, 1993).

Na tentativa de resolver os problemas de comunicação entre desenvolvedores e clientes (usuários), foram criados, no final da década de 1970, diversos métodos baseados na dinâmica de grupo.

O processo da entrevista

O processo de entrevista não se reduz ao ato em si. Por isso, é necessário pensar em todas as atividades que antecedem e sucedem a tarefa:

1. Determinação da necessidade da entrevista
2. Especificação do objetivo da entrevista
3. Seleção do entrevistado
4. Marcação da entrevista
5. Preparação das questões ou do roteiro
6. Entrevista propriamente dita
7. Documentação da entrevista, incluindo as informações obtidas
8. Validação com o entrevistado da transcrição da entrevista
9. Correção da transcrição
10. Aceitação por parte do entrevistado
11. Arquivamento

Roteiro para realização de entrevista

Para a realização de uma entrevista bem estruturada é preciso seguir um roteiro ou um script. Pode-se utilizar um modelo (template), retirando-se ou acrescentando-se itens sempre que necessário. E uma boa opção é estabelecer uma conversa informal antes da entrevista, para deixar o entrevistado mais à vontade. Depois é só seguir o roteiro:

1. Apresentar-se ao entrevistado, identificando-se e informando de qual projeto participa e qual a sua função (gerente de projetos, analista de sistemas etc.).
2. Informar ao entrevistado qual o motivo da entrevista e as razões pelas quais ele foi selecionado: é a melhor pessoa para auxiliar no levantamento de requisitos, por conhecer os procedimentos.
3. Deixar claro que o conhecimento e as opiniões do entrevistado são de extrema importância e serão muito úteis no processo de análise de requisitos.
4. Dizer ao entrevistado o que será feito com as informações levantadas.
5. Informar o entrevistado de que lhe será fornecida uma transcrição da entrevista, para que tenha oportunidade de ler e corrigir algum detalhe eventualmente equivocado e lhe assegurar que nenhuma informação será fornecida a outras pessoas até essa validação.
6. Determinar assuntos confidenciais ou restritos a serem tratados na entrevista.
7. Deixar claro que não haverá consequências negativas em função do resultado da entrevista.
8. Solicitar sempre a permissão do entrevistado para gravar a entrevista.
9. Se o entrevistado autorizar, deve-se iniciar a gravação com um texto de apresentação, para identificar a entrevista, informando o assunto e a data.
10. Avisar ao entrevistado quando o tempo estiver se esgotando e perguntar se ele gostaria de adicionar alguma informação.
11. No final, solicitar ao entrevistado que responda às perguntas de conclusão.
12. Concluir a entrevista de forma positiva, relatando brevemente o bom resultado alcançado.
13. Se necessário, marcar outra entrevista.
14. Despedir-se educadamente, agradecendo a atenção e o tempo dispensado.

Pela forma tradicional de desenvolvimento de software, os analistas de sistemas entrevistam usuários um após outro, área a área. Anotam tudo o que é dito e, somente em um segundo momento, as informações são consolidadas e validadas com os entrevistados. Após essa verificação, os dados são compilados em um documento de análise. Já ao se utilizar o JAD, as entrevistas individuais são substituídas por reuniões em grupo nas quais os envolvidos com o processo (usuários) participam ativamente ao lado da equipe de desenvolvimento.

Quando o método JAD é aplicado de forma correta, os resultados são excelentes. Além de atingir o objetivo de obter informações dos usuários, cria-se um ambiente de integração da equipe onde todos se sentem envolvidos e responsáveis por encontrar soluções. Esse comprometimento dos usuários faz com que eles se sintam "proprietários" do sistema e "colaboradores" no seu desenvolvimento, gerando, assim, maior aceitação do software quando este for implementado (AUGUST, 1993).