

Apostila da disciplina de Engenharia de Software – I

Prof. Cleber Nardelli¹

Rio do Sul, 24/02/2009

¹ Todos os direitos reservados, cópia deste material sem aviso prévio é considerada uma violação dos direitos autorais.

Conteúdo

Aula 1	5
Parte I - Introdução	5
Aula 2	6
Conceitos e Objetivos.....	6
Problemas comuns no Desenvolvimento de Software	7
Aula 3	8
Ciclo de Vida de Software.....	8
Fase de Definição	9
Aula 4	12
Processos de Software	12
Modelos Prescritos de Processo	13
Tópicos Adicionais.....	16
CMMI – Capability Maturity Model Integration	16
MPS.br – Melhoria do Processo de Software Brasileiro.....	18
Aula 5	21
Qualidade de Software.....	21
Custo da Qualidade.....	24
Modelos e Características de Qualidade.....	24
Característica: Funcionalidade (F).....	25
Sub-Característica: Interoperabilidade (I)	25
Garantia da Qualidade	27
Garantia da Qualidade de Software.....	27
Revisões de Software	28
Garantia estatística de qualidade de software	28
Controle de Qualidade	28
Caracterização de Defeito e Falha	29
Defeito.....	29
Falha	29
Aula 6	30
Parte II – Análise	30
Processo de Análise.....	31
Introdução	31
Aplicação.....	32
Fluxo de Análise	33
Regras práticas de Análise	33

Abordagens de modelagem de análise	33
Requisitos de Software.....	33
Funcionais	34
Não-Funcionais	34
Tipos de Requisitos	34
Engenharia de Requisitos	34
Tarefas da engenharia de requisitos	35
Aula 7	38
Aula 8	43
Técnicas Estruturadas	43
Diagrama de Fluxo de Dados (Conceitos).....	46
Projeto de Software	47
Processo de Projeto	47
Qualidade de Projeto	48
Aula 9	50
Projeto de Software	50
Modelo de Projeto	52
Projeto de Software Baseado em Padrão	52
Projeto Arquitetural	53
Projeto no nível de componentes	54
Projeto de Interface com o usuário	55
Aula 10	56
Atividade sobre Projeto de Software	56
Aula 11	57
Análise Essencial	57
Modelo Ambiental	58
Declaração de objetivos	58
Diagrama de Contexto.....	59
Lista de Eventos	59
Modelo Comportamental.....	65
Modelo de Implementação	65
Prototipação	65
Protótipo Descartável.....	65
Protótipo Evolucionário.....	66
Aula 12	67
Parte III - Metodologias e Técnicas	67

Gerenciamento de Projetos de Software	67
Resumo	71
Plano de Desenvolvimento de Software	72
Aula 13	75
Revisão – Planejamento e Gerenciamento de Projeto de Software	75
Exemplo de Documento de Projeto de Software	75
Aula 14	85
Métricas e Estimativas de Custo	85
Princípios de Medição	86
Tipos e Aplicação de Métricas de Produto	87
Método GQM	87
Estimativa do Projeto de Software	88
Técnicas de Decomposição	88
Estimando Projetos	92
Cronogramação de Projetos (Dicas Práticas).....	93
Aula 15	96
Teste de Software	96
Organização do teste de software	97
Estratégia de teste de software	97
Teste Unitário	99
Depuração	99
Leitura e Criação de Diagramas	100
Aula 16	103
Reengenharia.....	103
Reengenharia de Processo de Negócio	103
Reengenharia de Software	103
Engenharia Reversa.....	105
Reestruturação de Código	105
Reestruturação de Dados	105
Engenharia Avante	105
Referência Bibliográfica	106

Parte I - Introdução

Hoje o software de computador é a tecnologia única mais importante no palco mundial. É também um importante exemplo da lei das consequências não-pretendidas. Ninguém na década de 50 poderia prever que o software fosse se tornar uma tecnologia indispensável, para negócios, ciência e engenharia, que o software fosse permitir a criação de novas tecnologias (por exemplo, engenharia genética).

Software de computador é o produto que os profissionais de software constroem e, depois, mantêm ao longo do tempo. Abrange programas que executam em computadores de qualquer tamanho e arquitetura, conteúdo que é apresentado ao programa a ser executado e documentos tanto em forma impressa quanto virtual que combinam todas as formas de mídia eletrônica. Engenheiros de software os constroem e mantêm, e praticamente todas as pessoas do mundo industrializado usam direta ou indiretamente.

Você constrói software de computadores como constrói qualquer produto bem-sucedido, aplicando um processo ágil e adaptável que leva a um resultado de alta qualidade e que satisfaz às necessidades das pessoas que vão usar o produto. Você aplica uma abordagem de engenharia de software. Do ponto de vista do engenheiro de software, o produto do trabalho são os programas, conteúdo (dados) e documentos que compõem um software de computador. Mas, do ponto de vista do usuário, o produto do trabalho é a informação resultante que, de algum modo, torna melhor o mundo do usuário.

Conceitos e Objetivos

Os objetivos da disciplina são:

- Introduzir e aplicar técnicas para desenvolvimento de software, utilizando métodos, técnicas e ferramentas tanto para gerenciamento quanto para desenvolvimento.

Os Objetivos específicos da disciplina

- Conhecer e discutir os Conceitos e Objetivos da Engenharia de software nos Ciclos de Vida de Software e seus Problemas comuns no Desenvolvimento;

- Aplicar conceitos de Qualidade de Software na Análise e Planejamento de Projeto de Software;

- Introduzir o uso de Métricas e Estimativas de Custos para a Análise de Viabilidade e Preparação para Desenvolvimento;

- Aplicar técnicas de desenvolvimento de software;

- Introduzir planejamento e gerenciamento de software;

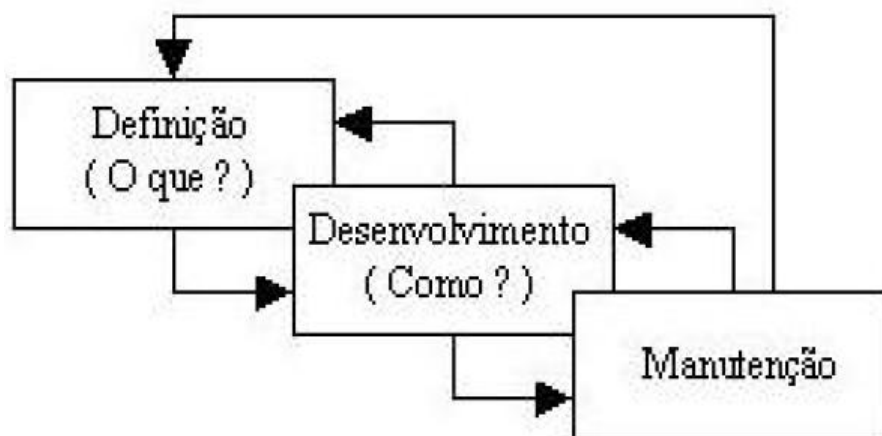
- Explorar a criação de diagramas para análise de software;

- Introduzir a Reengenharia de software;

- Introdução e utilidade da engenharia reversa, aplicação de ferramentas para o processo;

- Fundamentar UML.

Visão genérica de Engenharia de Software



1) Definição:

Função, desempenho, interface, restrições de projeto, critérios de validação.

Disciplinas:

Análise de Sistemas; Planejamento de projeto de software; Análise de requisitos;

2) Desenvolvimento:

Estrutura de dados, Arquitetura de software, detalhes procedimentais, programas, testes;

Disciplinas:

Projeto de Software; Codificação e Testes;

3) Manutenção:

Corretiva: para corrigir defeitos;

Adaptativa: para acomodar mudanças no ambiente externo do software (S.O, periféricos, etc);

Perfectiva: para inclusão de novas funcionalidades;

Problemas comuns no Desenvolvimento de Software

O desenvolvimento de um software sempre começa com um grande problema, que deverá ser quebrado em pequenos pedaços até que o problema seja compreendido e resolvido de forma computacional por completo. Após, os artefatos resultantes devem ser reunidos, formando assim o produto (software) final.

Dentre os principais problemas enfrentados na engenharia de software destacam-se:

1. Os projetos nem sempre são constituídos de processo definido, e esses são problemáticos quando tenta-se aplicar algum método;
2. Os requisitos são mal levantados, obtêm-se muitos requisitos implícitos;
3. Nas entrevistas realizadas com usuários não se tem um objetivo claramente definido, e nem usuário nem analista sabem efetivamente onde vão chegar;
4. O cliente não sabe exatamente o que quer;
5. Os usuários têm opiniões e pontos de vista diferentes de outros usuários sobre um mesmo procedimento operacional da empresa;
6. Quem faz não usa, ou seja, quem constrói o software não o utiliza efetivamente, ocasionando dificuldade na sua operação contínua;
7. Um objetivo/escopo claro nem sempre é definido, ocasionando um resultado desastroso ao final do desenvolvimento do software;
8. Os projetos têm tempo definido para início e término, especificar um tempo coerente para finalizar o projeto é um problema;
9. Os custos dos projetos, em alguns projetos são mal dimensionados, conflitando também com a questão do escopo do projeto, que quando mal especificado pode prejudicar tanto no tempo como no custo;
10. Um método eficiente de desenvolvimento é difícil de ser implantando em uma empresa (de software) que já está em funcionamento, existem culturas para serem quebrados, os custos por vezes são considerados altos, não existe respaldo da alta direção, sem uma ferramenta para controle eficiente um processo pode não ser seguido por todos.

Ciclo de Vida de Software

Todo projeto de software começa por alguma necessidade do negócio, a necessidade de corrigir um defeito em uma aplicação existente. Em 1970, menos de 1% do público poderia ter dito que o "Software de computadores" significa. Hoje, a maioria dos profissionais e muitos membros do público em geral pensam que entendem de software. Mas será verdade?

A definição de software poderia ser: Software são (1) *as instruções (programas de computadores) que quando executadas fornecem as características, função e desempenho desejadas*; (2) *estruturas de dados que permitem aos programas manipular adequadamente a informação*; e (3) *documentos que descrevem a operação e o uso dos programas*. Não há dúvida de que definições mais completas poderiam ser oferecidas, mas nós precisamos mais do que uma definição formal.

Para entender de software precisamos analisar as características do software que o tornam diferente de outras coisas produzidas. O software é um elemento de um sistema lógico e não de um sistema físico. Assim ele possui características que são consideravelmente diferentes daquelas do hardware:

1. O Software é desenvolvido ou passa por um processo de engenharia; não é fabricado no sentido clássico.

Apesar de existirem algumas semelhanças no desenvolvimento de software e hardware, as duas são fundamentalmente diferentes. Em ambas a alta qualidade é conseguida por um bom projeto, mas a fase de fabricação do hardware pode gerar problemas de qualidade, que são inexistentes (ou facilmente corrigidos) para o software. Ambas as atividades são dependentes de pessoas, mas a relação entre as pessoas envolvidas e o trabalho realizado é inteiramente diferente. Os custos do software são concentrados na engenharia, isto significa que não podem ser geridos como se fossem projetos de fabricação.

2. Software não se desgasta

O software não é suscetível aos males ambientais que causam o desgaste do hardware. Teoricamente, entretanto, a curva de taxa de falhas para o software deveria tomar a forma de "curva idealizadora". Defeitos não detectados causarão altas taxas de falhas no início da vida de um programa. Todavia, eles são corrigidos (teoricamente sem a adição de novos erros) e a curva se achata. O Software não se desgasta, mas se deteriora.

Quando um hardware falha pode ser concertado com peças sobressalentes, para um software não existem peças sobressalentes. Toda falha de software indica um erro de projeto, ou no processo pelo qual o projeto foi traduzido para código de máquina executável, assim sua manutenção é consideravelmente mais complexa que a manutenção de hardware.

3. Apesar de a indústria estar se movendo em direção à montagem baseada em componentes, a maior parte dos softwares continua a ser construída sob encomenda

À medida que uma disciplina de engenharia evolui, uma coleção de componentes de projeto padronizados é criada. Os componentes reutilizáveis foram criados para que engenheiros possam se concentrar nos elementos realmente inovadores de um projeto, isto é, as partes do projeto que representam algo novo. No mundo do software isto está apenas começando a ser obtido em ampla escala.

O ciclo de vida de um software descreve as fases pelas quais o software passa desde a sua concepção até ficar sem uso algum. O conceito de ciclo de vida de um software é muitas vezes confundido com o de modelo de processo.

Existem várias propostas e denominações para as fases do ciclo de vida de um software. Cada fase inclui um conjunto de atividades ou disciplinas que devem ser realizadas pelas partes envolvidas. Essas fases são:

- Definição
- Desenvolvimento
- Operação
- Retirada

Fase de Definição

A fase de definição do software ocorre em conjunto com outras atividades como a *modelagem de processos de negócios* e *análise de sistemas*. Nesta atividade, se concentra a busca pelo conhecimento da situação atual e a identificação de problemas para a elaboração de propostas de solução de sistemas computacionais que resolvam tais problemas. Dentre as propostas apresentadas, deve-se fazer um *estudo de viabilidade*, incluindo *análise custo-benefício*, para se decidir qual solução será a escolhida. O resultado desta atividade deve incluir a decisão da aquisição ou desenvolvimento do sistema, indicando informações sobre hardware, software, pessoal, procedimentos, informação e documentação.

Caso seja decidido pelo desenvolvimento do sistema, no escopo da engenharia de software, é necessário elaborar o documento de *proposta de desenvolvimento de software*. Esse documento pode ser a base de um *contrato de desenvolvimento*.

Profissionais de engenharia de software atuam nesta atividade com o objetivo de identificar os *requisitos de software* e *modelos de domínio* que serão utilizados na fase de desenvolvimento. Os requisitos são também fundamentais para que o engenheiro possa elaborar um *plano de desenvolvimento de software*, indicando em detalhes os recursos necessários (humanos e materiais), bem como as estimativas de prazos e custos (cronograma e orçamento).

Não existe um consenso sobre o que caracteriza o final da fase de definição. Isto varia de acordo com o modelo de processo adotado. Em algumas propostas, a fase de definição é considerada concluída com a apresentação da proposta de desenvolvimento. Outros modelos de processo, consideram que o software apenas está completamente definido com a *especificação de requisitos* e com a elaboração do *plano de desenvolvimento de software*. De acordo com o modelo de processo adotado, pode-se iniciar atividades da fase de desenvolvimento mesmo que a fase de definição não esteja completamente concluída.

Fase de Desenvolvimento

A fase de desenvolvimento ou de produção do software inclui todas as atividades que tem por objetivo a construção do produto. Ela inclui principalmente o design, a implementação e a verificação e validação do software.

Design

A atividade de design compreende todo o esforço de concepção e modelagem que têm por objetivo descrever como o software será implementado. O design inclui:

- Design conceitual

- Design da interface de usuário
- Design da arquitetura do software
- Design dos algoritmos e estruturas de dados

O **design conceitual** envolve a elaboração das idéias e conceitos básicos que determinam os elementos fundamentais do software em questão. Por exemplo, um software de correio eletrônico tradicional inclui os conceitos: *mensagem*, *caixa de entrada*, *caixa de saída*, etc. A mensagem, por sua vez, inclui os conceitos de *para*, *cc*, *bcc*, *assunto*, *corpo*, etc. Embora seja um design adotado pela maioria dos softwares, novos modelos conceituais podem vir a ser adotados. O design conceitual exerce influência na interface de usuário e na arquitetura do software.

O **design da interface de usuário** envolve a elaboração da maneira como o usuário pode interagir para realizar suas tarefas, a escolha dos objetos de interfaces (botões, menus, caixas de texto, etc.), o *layout* de janelas e telas, etc. A interface deve garantir a boa *usabilidade* do software e é um fundamental fator de sucesso do software.

O **design de arquitetura de software** deve elaborar uma visão macroscópica do software em termos de componentes que interagem entre si. O conceito de componente em arquitetura varia de acordo com a visão arquitetônica adotada. São exemplos de visões arquitetônicas, a visão conceitual, visão de módulos, visão de código e visão de execução.

- Na *visão conceitual*, os componentes de software são derivados do design conceitual. Estes componentes são abstrações que devem definir outros elementos menos abstratos. Exemplos são arquiteturas cliente-servidor e arquitetura em camadas.
- Na *visão de código*, deve-se determinar como as classes e/ou funções estão organizadas e interagindo entre si. Estas classes implementam os componentes abstratos ou conceituais.
- Na *visão de módulos*, deve-se determinar quais são os módulos que serão utilizados na implementação e como eles organizam as classes e/ou funções.
- Na *visão de execução*, a arquitetura deve descrever os diferentes processos que são ativados durante a execução do software e como eles interagem entre si. Enquanto as anteriores oferecem uma visão estática, esta é uma visão dinâmica do software.

O **design de algoritmos e estrutura de dados**, também conhecido como design detalhado, visa determinar, de maneira independente da linguagem de programação adotada, as soluções algorítmicas e as estruturas de dados associados. Deve-se decidir, por exemplo, como as informações podem ser ordenadas (algoritmo de bolha ou quicksort) e em qual tipo de estrutura de dados (array, lista encadeada) elas vão ser armazenadas.

Implementação

A **implementação** envolve as atividades de codificação, compilação, integração e testes. A codificação visa traduzir o design num programa, utilizando linguagens e ferramentas adequadas. A codificação deve refletir a estrutura e o comportamento descrito no design. Os componentes arquiteturais devem ser codificados de forma independente e depois integrados. Os testes podem ser iniciados durante a fase de implementação. A depuração de erros ocorre durante a programação utilizando algumas técnicas e ferramentas. É fundamental um controle e gerenciamento de versões para que se tenha um controle correto de tudo o que está sendo codificado.

Verificação e validação

Verificação e validação destinam-se a mostrar que o sistema está de acordo com a especificação e que ele atende às expectativas de clientes e usuários. A validação visa assegurar se o programa está fazendo aquilo que foi definido na sua especificação (fazendo a coisa certa). A verificação visa verificar se o programa está correto, isto é,

não possui erros de execução (fazendo certo a coisa). Existem diferentes formas de verificação e validação. Inspeção analítica e revisão de modelos, documentos e código fonte são formas que podem ser usadas antes mesmo que o programa seja completamente codificado. Os testes de correção, desempenho, confiabilidade, robustez, usabilidade, dentre outros, visam avaliar diversos fatores de qualidade a partir da execução do software. Diferentes técnicas de testes podem ser aplicadas para cada um destes fatores

Fase de Operação

A fase de operação envolve diferentes tipos de atividades:

- Distribuição e entrega
- Instalação e configuração
- Utilização
- Manutenção

A **distribuição e entrega** pode ser feita diretamente pelo desenvolvedor (em caso de software personalizado), ou em um pacote a ser vendido em prateleiras de lojas ou para ser baixado pela Internet (em caso de softwares genéricos).

O processo de **instalação e configuração**, normalmente, pode ser feito com a ajuda de software de instalação disponibilizados pelos fabricantes dos ambientes operacionais.

A atividade de **utilização** é o objeto do desenvolvimento do software. A qualidade da utilização é a usabilidade do software.

A **manutenção** normalmente ocorre de duas formas: corretiva e evolutiva. A manutenção corretiva visa a resolução de problemas referentes a qualidade do software (falhas, baixo desempenho, baixa usabilidade, falta de confiabilidade, etc.). A manutenção evolutiva ou adaptativa visa a produção de novas versões do software de forma a atender a novos requisitos dos clientes, ou adaptar-se às novas tecnologias que surgem (hardware, plataformas operacionais, linguagens, etc). Mudanças no domínio de aplicação implicam em novos requisitos e incorporação de novas funcionalidades. Surgimento de novas tecnologias de software e hardware e mudanças para uma plataforma mais avançada também requerem evolução.

Fase de retirada

A fase de retirada é um grande desafio para os tempos atuais. Diversos softwares que estão em funcionamento em empresas possuem excelentes níveis de confiabilidade e de correção. No entanto, eles precisam evoluir para novas plataformas operacionais ou para a incorporação de novos requisitos. A retirada desse *software legado* em uma empresa é sempre uma decisão difícil: como abrir mão daquilo que é confiável e ao qual os funcionários estão acostumados, após anos de treinamento e utilização?

Processos de *reengenharia* podem ser aplicados para viabilizar a transição ou *migração* de um software legado para um novo software de forma a proporcionar uma retirada mais suave.

Processos de Software

Quando se elabora um produto ou sistema é importante percorrer uma série de passos previsíveis – um roteiro que o ajuda a criar a tempo um resultado de alta qualidade. O roteiro que você segue é chamado de processo de software. Este roteiro deverá ser adaptável de acordo com os objetivos de negócio da organização.

O processo a ser adotado depende muito do tipo de software que está se construindo. Um processo poderia ser utilizado para a construção de um software para aviões, porém este mesmo processo poderia não ser muito útil na produção de um site.

Um processo bem definido para construção de software é importante, pois fornece sustentabilidade, estabilidade, controle e organização para uma atividade que pode, se deixada sem controle, tornar-se bastante caótica. No entanto uma abordagem moderna de engenharia de software precisa ser “ágil”. Precisa exigir apenas aquelas atividades, controles e documentação adequados à equipe de projeto e ao produto que deve ser produzido.

A engenharia de software é uma tecnologia em camadas, qualquer abordagem de engenharia (inclusive a de software) deve se apoiar num compromisso organizacional com a qualidade. Gestão de qualidade total, Seis Sigmas (Six Sigma) e filosofias análogas levam à cultura de um processo contínuo de aperfeiçoamento, e é essa cultura que, em última análise, leva ao desenvolvimento de abordagens cada vez mais efetivas para a engenharia de software. A base em que se apóia é o foco na qualidade.

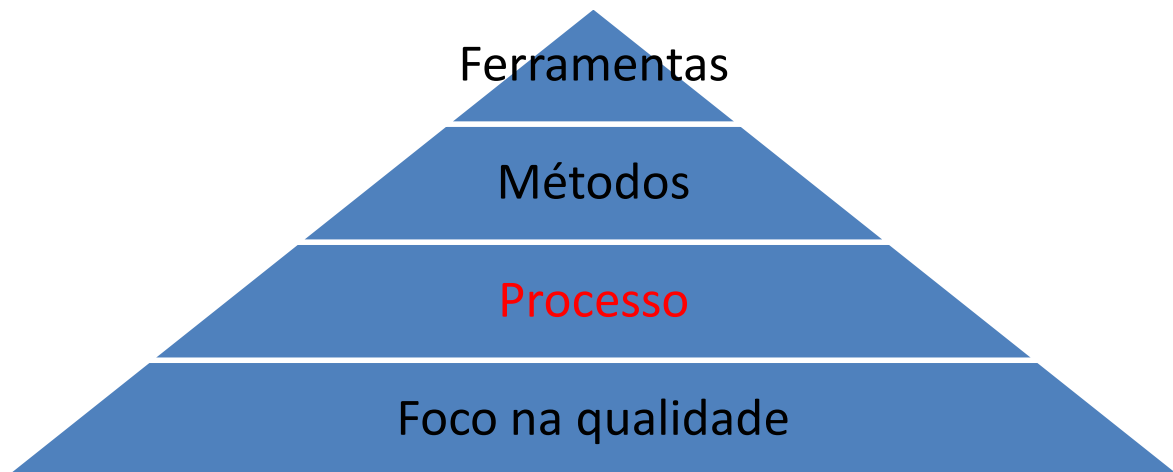


Figura 1 Camadas da Engenharia de Software

O principal ponto da engenharia de software é a camada de processo. A camada de processo é responsável pela junção das camadas de tecnologia e objetiva o desenvolvimento racional e oportuno de softwares de computador.

Os *processos* de software formam a base para o controle gerencial de projetos de software e estabelecem o contexto no qual os métodos técnicos são aplicados, os produtos de trabalho (modelos, documentos, dados, relatórios, formulários, etc.) são produzidos, os marcos são estabelecidos, a qualidade é assegurada e as modificações são adequadamente geridas. Em síntese um processo de software constitui-se de atividades realizadas pelas pessoas, utilizando ferramentas, métodos, etc., para adquirir, desenvolver, manter e melhorar o software e produtos associados.

Os *métodos* de engenharia de software fornecem a técnica de “como fazer” para construir softwares. Eles abrangem um amplo conjunto de tarefas que incluem comunicação, análise de requisitos, modelagem de projeto, construção de programas, testes e manutenção, incluem atividades de modelagem e outras técnicas descritivas.

As *ferramentas* de engenharia de software fornecem apoio automatizado ou semi-automatizado para o processo e para os métodos. Quando ferramentas são integradas de modo que a informação criada por uma ferramenta possa ser usada por outra, um sistema de apoio ao desenvolvimento de software, chamado engenharia de software apoiada por computador, é estabelecido.

Uma visão pragmática de processo poderia ser ilustrada com a seguinte figura:

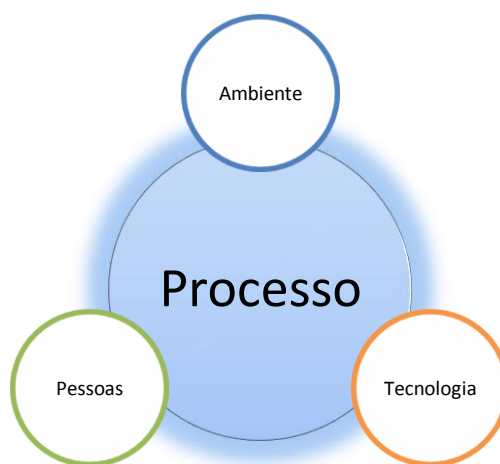


Figura 2 Visão Pragmática de Processo

Uma mudança em qualquer direção requer reavaliação de processo.

Modelos Prescritos de Processo

Cascata/Clássico

Há ocasiões em que os requisitos de um problema são razoavelmente bem compreendidos - quando o trabalho flui da comunicação até a implantação de um modo razoavelmente linear. Essa situação é encontrada quando adaptações ou aperfeiçoamentos de um sistema existente são necessárias (por exemplo, uma adaptação em um software de contabilidade que deve ser feita devido a modificações na regulamentação governamental).

O Modelo em cascata, algumas vezes também chamado de ciclo de vida clássico, sugere uma abordagem sistemática e seqüencial, para o desenvolvimento de softwares e começa com a especificação dos requisitos pelo cliente e progride ao longo do planejamento, modelagem, construção e implantação, culminando na manutenção progressiva do software acabado.



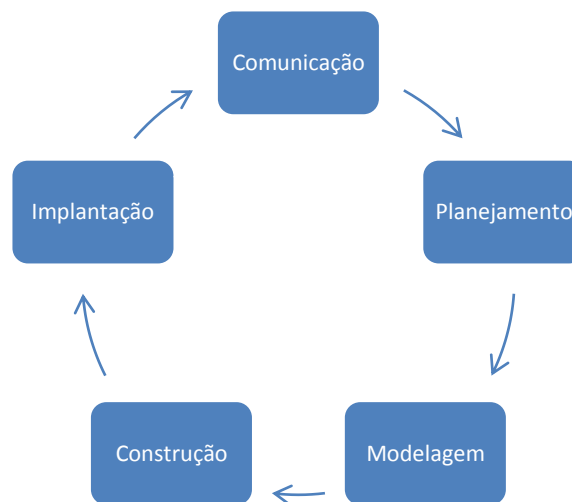
Este é um modelo mais antigo na engenharia de software, porém algumas discussões realizadas principalmente nas últimas décadas têm provocado dúvidas até entre os mais ardentes adeptos deste, são questionamentos sobre sua eficácia. Entre os maiores problemas podemos destacar:

1. Projetos reais raramente seguem fluxo seqüencial que o modelo propõe. Como resultado as modificações podem provocar confusão à medida que a equipe do projeto segue;
2. É difícil para o cliente estabelecer todos os requisitos explicitamente. O modelo exige isso e tem dificuldade de acomodar a incerteza natural que existe no começo do projeto;
3. O cliente precisa ter paciência. Como uma versão executável do(s) programa(s) não fica disponível antes do final do projeto, um erro grosseiro pode ser desastroso para todo o projeto;

Incremental

Assim como no modelo cascata, o modelo incremental exige um número de requisitos razoavelmente definidos, porém como o escopo global do esforço de desenvolvimento elimina um processo puramente linear, o projeto pode ser conduzido de forma incremental. Além disso pode haver uma necessidade de que versões preliminares de certas funcionalidades do projeto, sejam liberadas antes do final, para que depois estas funcionalidades sejam refinadas e expandidas em versões subseqüentes do software.

O Modelo incremental combina elementos do modelo em cascata aplicado de maneira iterativa. O modelo incremental aplica seqüências lineares de uma forma racional à medida que o tempo passa. Cada seqüência linear produz “incrementos” do software passíveis de serem entregues. Quando o modelo incremental é usado, o primeiro incremento é freqüentemente chamado de núcleo do produto, isto é, requisitos básicos são satisfeitos, mas muitas características suplementares (algumas conhecidas outras desconhecidas) deixam de ser elaboradas. Neste ponto, um plano é desenvolvido para o próximo incremento como resultado do uso e/ou avaliação. Um incremento é diferente de prototipagem, pois exige um produto operacional como saída.



A cada ciclo tem-se um incremento como saída. O Modelo incremental também é utilizado por todos os modelos de processos “ágeis” discutidos mais a frente.

Evolucionário

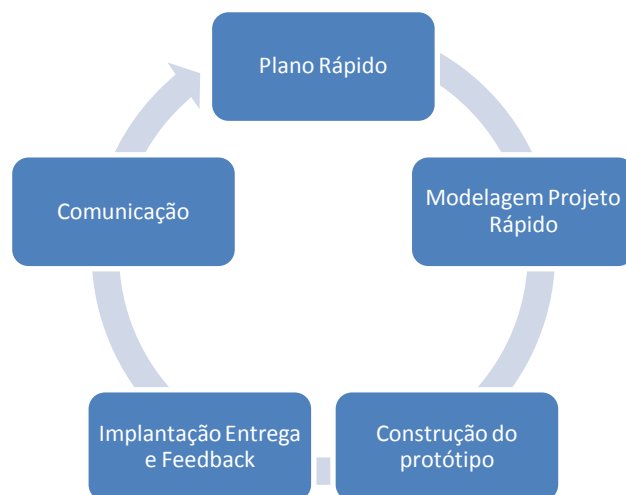
O software, como todo sistema complexo, evolui com o passar do tempo. Os requisitos do negócio e do produto mudam freqüentemente à medida que o desenvolvimento prossegue, dificultando um caminho direto para um produto final, porém uma versão reduzida poderá ser elaborada para fazer face a competitividade ou às pressões do negócio. Um conjunto de requisitos básicos do sistema é entendido, mas os detalhes das extensões do produto ou sistema ainda precisam ser definidos.

Os modelos evolucionários são iterativos, eles são caracterizados de forma a permitir aos engenheiros de software desenvolver versões cada vez mais completas do software. Incluem a **prototipagem** e o **modelo espiral**.

Prototipagem

O Cliente freqüentemente define um conjunto de objetivos gerais para o software, mas não identifica detalhadamente requisitos de entrada, processamento ou saída. Em outros casos o programador pode estar inseguro do funcionamento de um algoritmo ou da forma como homem/máquina devem interagir. Para esses casos o paradigma da prototipagem pode oferecer a melhor abordagem.

A prototipagem começa com a comunicação. O Engenheiro de software e o cliente encontram-se e definem os objetivos gerais do software, identificam as necessidades conhecidas e delineiam áreas que necessitam de mais informações. Essa abordagem ocorre de forma rápida.



A prototipagem pode ser problemática pelas seguintes razões:

1. O cliente vê o que parece ser uma versão executável do software, ignorando que o protótipo apenas consiga funcionar precariamente, sem saber que na pressa de fazê-lo rodar, ninguém considerou a sua qualidade global;
2. O desenvolvedor faz algumas concessões na implementação a fim de conseguir rapidamente um protótipo executável.

Apesar dos problemas que podem ocorrer a prototipagem é um paradigma efetivo para a engenharia de software.

Modelo Espiral

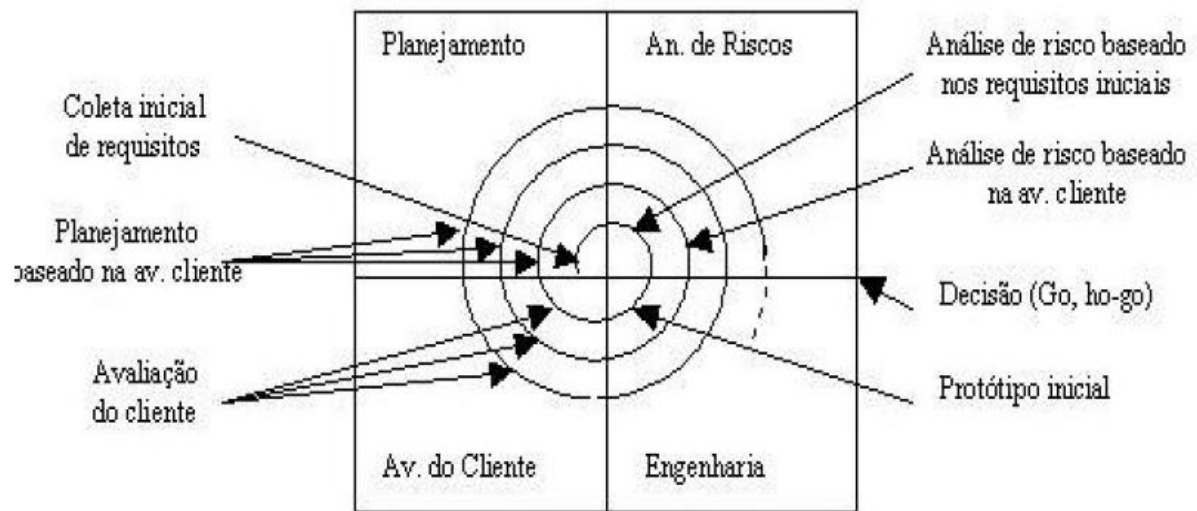
O Modelo espiral, é um modelo evolucionário de processo de software que combina a natureza iterativa da prototipagem com os aspectos controlados e sistemáticos do modelo em cascata. Ele fornece potencial para o desenvolvimento rápido de versões de software cada vez mais completas.

Usando o modelo espiral, o software é desenvolvido numa série de versões evolucionárias. Durante as primeiras iterações, as versões podem ser um modelo de papel ou protótipo. Durante as últimas iterações são produzidas verões cada vez mais completas do sistema submetido à engenharia.

O primeiro circuito em torno da espiral poderia resultar no desenvolvimento da especificação de um produto; passagens subseqüentes em torno da espiral poderiam ser usadas para desenvolver um protótipo e depois,

progressivamente, versões mais sofisticadas do software. Cada passagem pela região de planejamento resulta em ajustes no plano de projeto. O custo e o cronograma são ajustados com base no retorno do cliente após a entrega.

O Modelo espiral é uma visão realista do desenvolvimento de sistemas e softwares de grande porte.



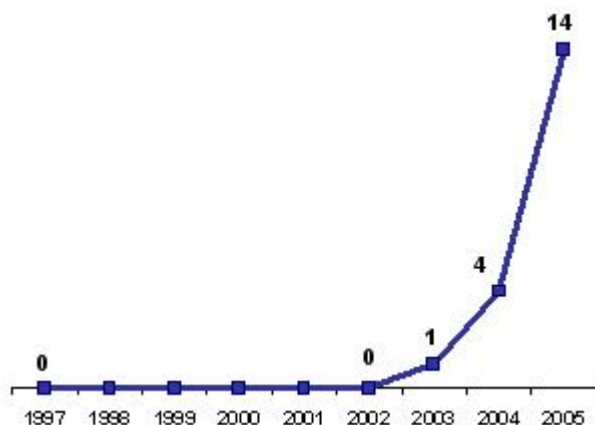
Principal problema do modelo espiral é que por se tratar de um modelo relativamente novo, requer uma certa dose de conhecimento, podendo nunca chegar a um final.

Tópicos Adicionais

CMMI – Capability Maturity Model Integration

O CMMI foi desenvolvido pelo SEI (Software Engineering Institute), é um metamodelo que abrange um conjunto de capacidades de engenharia de software que devem estar presentes à medida que as organizações alcançam diferentes níveis de capacidade de maturidade de processo. Ele é adotado e reconhecido internacionalmente por ajudar as organizações a alcançarem processos maduros, estáveis e consistentes. O CMMI serve também como requisito para que empresas possam exportar software.

Organizações com Qualificação CMMI no Brasil – 1997-2005¹



Desde	Nível Atual				No ano	Até o ano
	2	3	4	5		
2003		1			1	1
2004	2			1	3	5
2005	7	2		1	10	14
Total	9	3		2		14

Fontes: ISD Brasil, Procesix, empresas qualificadas e imprensa especializada, compilado por MCT/SEPIN/DIA.
¹ Situação em novembro/2005.

CMMI significa Capability Maturity Model Integration – Modelo Integrado de Maturidade e Capacidade, é um modelo alinhado com a norma ISO/IEC 15504. Tem por objetivo prover um guia para melhorar a capacidade dos

processos organizacionais e sua eficiência em gerenciar o desenvolvimento, aquisição e manutenção de produtos e serviços, além disso ele organiza as práticas comprovadamente efetivas, em uma estrutura que ajuda a Organização a estabelecer prioridades para melhoria e fornece um guia na implementação destas. O CMMI aplica os princípios de TQM (Gerenciamento de Qualidade Total, que é a aplicação de métodos quantitativos e recursos humanos) ao desenvolvimento de software.

O CMMI é dividido em duas representações básicas, por estágios e contínua.

A representação por estágios organiza as áreas de processo em cinco níveis de maturidade para suportar e guiar a melhoria de processos. Esses níveis de maturidade representam um caminho de melhoria de processos para toda a organização.

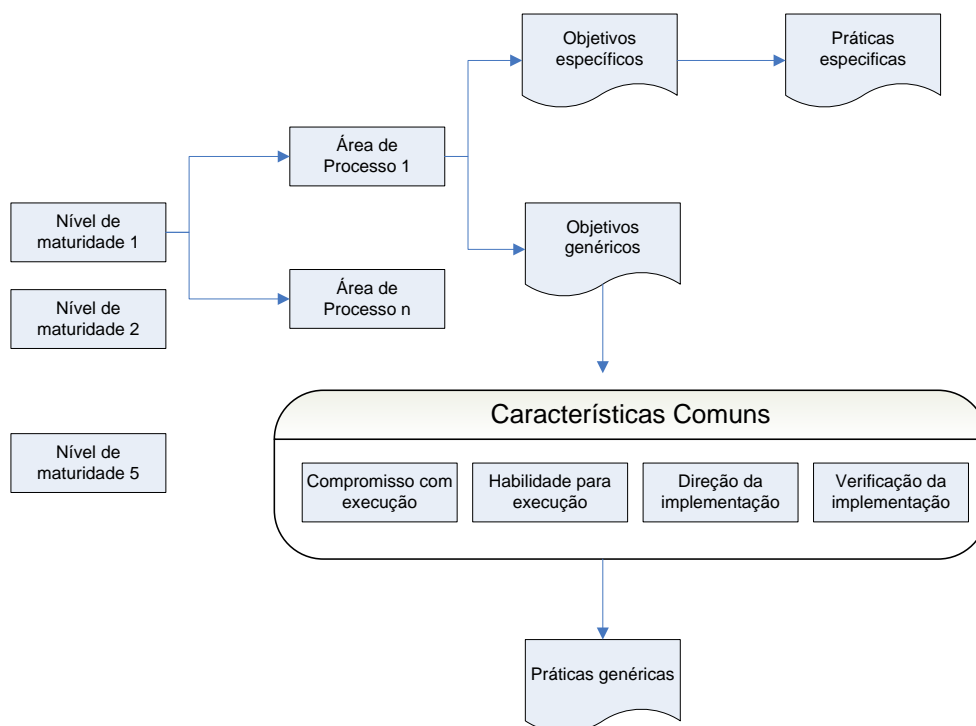


Figura 3 Estrutura do CMMI por estágios (adaptado do relatório técnico do SEI, ref. CMU/SEI-2002-TR-029)

Um nível de maturidade consiste em práticas específicas e genéricas para uma área de processo, que podem levar as melhorias nos processo organizacionais. Na representação do CMMI por estágios, existem cinco níveis de maturidade, designados pelos números de 1 a 5: inicial, gerenciado, definido, gerenciado quantitativamente e otimizado. Os níveis de maturidade consistem em um conjunto predefinido de áreas de processo. Para uma organização passar para um próximo nível deverá sempre obedecer aos níveis imediatamente anteriores.

Nível 1: Inicial

No nível 1, os processos são caóticos. A organização que se encontra neste nível, geralmente não possui um ambiente estável de desenvolvimento, padrões não existem, ou se existem, não são seguidos. Nestes casos geralmente existem problemas no cumprimento de prazos e custos, bem como de requisitos. A organização sobrevive na dependência de talentos individuais.

Este fato não significa que os produtos são ruins. É possível até mesmo nesta situação que bons produtos sejam entregues. Mas para isto ocorrer, a organização dependerá de seus heróis que fazem muitas horas extras para compensar um planejamento mal feito. Pode acontecer também de o produto ser entregue, mas a um custo mais alto ou com prazo excessivo. Organizações neste nível dificilmente conseguem repetir sucessos anteriores.

Nível 2: Gerenciado

Neste nível, os projetos da organização possuem requisitos gerenciados e processos planejados, medidos e controlados. Estas práticas possibilitam que a organização cumpra os planos no desenvolvimento dos projetos.

Os requisitos, processos e serviços são gerenciados. Isto significa que há grande preocupação em seguir os planos. *Isto se deve ao fato de os cronogramas serem periodicamente revistos.*

As datas intermediárias de entregas de partes dos produtos são estabelecidas em comum acordo com os *stakeholders* e revisadas, com os produtos, sempre que necessário. O Acompanhamento pelos gerentes aumenta a probabilidade de que os prazos sejam cumpridos.

Nível 3: Definido

Neste nível os processos são bem caracterizados e entendidos. A padronização de processos possibilita maior consistência nos produtos gerados pela organização.

Na descrição dos processos são usados padrões, procedimentos, ferramentas e métodos bem definidos. Esses fatores diferenciam os níveis 3 e 2. Organizações com nível 2 podem variar padrões, descrições de processo e procedimentos a cada projeto. Outra distinção ao nível anterior são os níveis de detalhe a respeito da descrição de processos.

Nível 4: gerenciado quantitativamente

Os processos são selecionados para contribuir com o desempenho geral dos demais processos. Utiliza-se métodos estatísticos e outras técnicas quantitativas. Os aspectos quantitativos devem ser traduzidos em números o que permite que sejam mais bem compreendidos e comparados.

No nível 4 dados sobre os processos são coletados e analisados estatisticamente, facilitando o acompanhamento do desempenho da empresa.

Nível 5: otimizado

No nível de maturidade otimizado, os processos são continuamente melhorados com base em um entendimento quantitativo das causas comuns de alterações de desempenho. Esta melhoria é conquistada através de inovações e busca por novas tecnologias seus efeitos são medidos e avaliados.

A Melhoria de processos é uma tarefa de todos, não apenas uma ordem específica dos níveis hierárquicos mais altos, desta forma evita-se a volta a níveis anteriores do CMMI.

MPS.br – Melhoria do Processo de Software Brasileiro

O MPS.BR foi criado por pesquisadores brasileiros para a melhoria do processo de desenvolvimento de software em empresas brasileiras. É um modelo recente que está em desenvolvimento desde 2003, pelas instituições SOFTEX, Riosoft, COPPE/UFRJ, CESAR, CenPRA e CELEPAR. O foco são as micro, pequenas e médias empresas de software brasileiras que possuem poucos recursos para melhoria de processos, mas que estão diante da necessidade de fazê-los. A descrição do modelo baseia-se em três guias:

- **Guia geral:** contém a descrição geral do MPS.BR e detalha o modelo de referência (MR-MPS), seus componentes e as definições comuns necessárias para seu entendimento e aplicação.
- **Guia de aquisição:** contém recomendações para a condução de compras de software e serviços correlatos. Foi elaborado para guiar as instituições que irão comprar produtos de software e correlatos.

- **Guia de avaliação:** contém a descrição do processo de avaliação, os requisitos para o avaliador e para a avaliação, o método e os formulários para guiar a avaliação.

O MPS.BR está dividido em três componentes: Modelo de referência (MR-MPS), Método de avaliação (MA-MPS) e Modelo de negócio (MN-MPS).

O MR-MPS, possui as definições dos níveis de maturidade, da capacitação de processos e dos processos em si. É baseado nas normas NBR ISO/IEC 12207 e suas emendas 1 e 2, na ISO/IEC 15504 e além disso adequado ao CMMI-SE/SW.

O MA-MPS contém o processo de avaliação, os requisitos para os avaliadores e os requisitos para averiguação da conformidade ao modelo MR-MPS este foi baseado no documento ISO/IEC 15504.

O MN-MPS contém uma descrição das regras para implementação do MR-MPS pelas empresas de consultoria, de software e de avaliação.

O Modelo de referência MR-MPS define níveis de maturidade que são uma combinação de processos e capacitação de processos. O nível de maturidade em que se encontra uma organização permite prever seu desempenho futuro. Os objetivos de melhoria forma divididos em sete níveis de maturidade, de A (melhor) a G (pior). Segundo os autores do MPS.BR os sete níveis devem permitir uma implantação mais gradual que o CMMI, facilitando a tarefa em empresas de pequeno porte. Para cada nível definiu-se um perfil de processos e um perfil de capacitação de processos.

Nível de maturidade no MPS.br, significa o grau de melhoria de processo para um pré-determinado conjunto de processos no qual todos os objetivos dentro do conjunto são atendidos. Ao todo são 7 níveis:

A. Em Otimização

Analisa causa de problemas e resolução

B. Gerenciado quantitativamente

Gerência de Projetos (evolução)

C. Definido

Análise de Desisção e Resolução

Desenvolvimento para Reutilização

Gerência de Riscos

Gerência de Reutilização (evolução)

D. Largamente definido

Desenvolvimento de Requisitos

Projeto e Construção do Produto

Integração do Produto

Verificação / Validação

E. Parcialmente definido

Avaliação e Melhoria do Processo Organizacional

Definição do Processo Organizacional

Gerência de Reutilização / Gerência de Recursos Humanos

Gerência de Projetos (evolução)

F. Gerenciado

Medição / Gerência de Configuração

Aquisição / Garantia de Qualidade

G. Parcialmente gerenciado

Gerência de Requisitos

Para atividade em sala de aula:

Considere os seguintes requisitos:

Nome	Descrição
RF1	O Software deverá permitir o cadastramento de pessoas, com possibilidade de consulta, manutenção e emissão de relatório em uma única tela, podendo disparar telas separadas.
RF2	Na tela de consulta poderá ser informado um parâmetro de entrada para pesquisar por nome da pessoa ou pelo seu código
RF3	Um cliente poderá efetuar compras na loja, portanto deverá possuir uma tela para efetuar as transações

Qualidade de Software

Histórico

A idéia de qualidade é aparentemente intuitiva; contudo, quando examinado mais longamente, o conceito se revela complexo. Definir um conceito de qualidade para estabelecer objetivos é, assim, uma tarefa menos trivial do que aparenta a princípio.

Embora o controle de qualidade e o uso de padrões como ISO sejam algo que tenha atraído bastante atenção nas últimas décadas, historicamente o assunto é muito antigo. Existem relatos históricos segundo os quais há mais de quatro mil anos os egípcios estabeleceram um padrão de medida de comprimento: o cúbito. Essa medida correspondia ao comprimento do braço do faraó reinante. Curiosamente, a troca de faraó significava que a medida deveria ser atualizada. Todas as construções deviam ser realizadas utilizando o cúbito como unidade de medida. Para isso eram empregados bastões cortados no comprimento certo e periodicamente – a cada lua cheia – o responsável por uma construção devia comparar o padrão que estava sendo utilizado com o padrão real. Se isso não fosse feito e houvesse um erro de medição, o responsável poderia ser punido com a morte [Juran e Gryna, 1998]. O resultado da preocupação com rigor é evidente na construção das pirâmides, em que os egípcios teriam obtido precisões da ordem de 0,05 %.

A história da qualidade prosseguiu com inúmeros exemplos de resultados extraordinários: os grandes templos construídos na Grécia e Roma antigas, os feitos de navegação no século XVI, as catedrais medievais. Em todas essas realizações não se dispunha de instrumentos de precisão ou técnicas sofisticadas. Na França os construtores de catedrais utilizavam simples compassos e cordas com nós a intervalos regulares para desenhar e construir edifícios [Vincent, 2004].

Em geral, espera-se que obter mais precisão exija mais recursos ou mais tecnologia. Assim, a regulagem da carburação de um motor de um veículo moderno não pode ser feita como no passado, quando, com uma lâmpada, um mecânico conseguia “acertar o ponto” do distribuidor. O exemplo dos antigos egípcios nos faz pensar uma questão curiosa: como teriam sido as pirâmides se, na época, os trabalhadores dispusessem de medidores a laser ?

Como veremos neste tópico da disciplina, a qualidade de software depende principalmente do correto emprego de boas metodologias pelos desenvolvedores. Embora eles sejam apoiados por várias ferramentas, ainda restam problemas sérios sem tal suporte. As técnicas para verificação automática, dentre as quais a interpretação abstrata [Consot, 2000] é um excelente exemplo, ainda são incipientes.

Um grande marco na história da qualidade foi, com certeza, a revolução industrial. Esse período também é associado a profundas mudanças econômicas e sociais, como o início da automação e o surgimento do consumo de massa. A criação de diversas indústrias levou rapidamente à concorrência entre elas, o que, por sua vez, desencadeou um processo de melhoria contínua que perdura até hoje. O aumento da eficiência tornou-se uma condição imprescindível para garantir a sobrevivência. Uma ilustração clara disso é a extinção de centenas de fábricas de automóveis nos Estados Unidos: no início do século XX, esse país contava com cerca de 1.800 fabricantes diferentes.

Na década de 1920 surgiu o controle estatístico de produção. Nas fábricas que produziam grande quantidade de itens tornou-se impossível garantir a qualidade individual de cada peça, ao contrário do que se fazia (e ainda se faz) no trabalho artesanal. Dessa forma foi preciso desenvolver mecanismos diferentes e a resposta veio da estatística. Um dos primeiros trabalhos associados ao assunto é o livro publicado por Walter Shewhart em 1931,

Economic Control of Quality Manufactured Product, dos Bell Laboratories, teria introduzido os diagramas de controle (control charts ou Shewhart chart). A figura a seguir apresenta um exemplo desse tipo de diagrama.

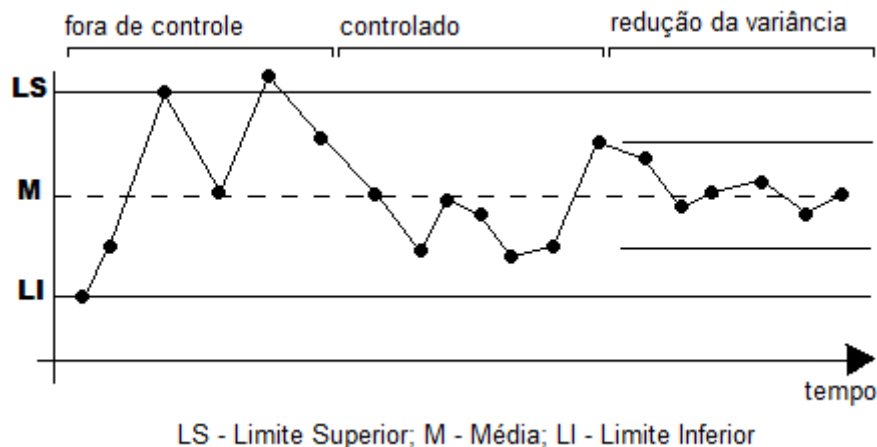


Figura 4 Diagrama de Shewhart

Para entender o diagrama, vamos supor que o problema tratado consista em controlar o diâmetro de parafusos. O diagrama apresenta três linhas verticais que definem o valor médio e os limites superior e inferior máximos tolerados. Cada ponto do diagrama representa o valor de uma amostra, isto é, um parafuso recolhido aleatoriamente na saída da fábrica. O gráfico permite verificar se os processos estão sendo bem controlados e se há tendências de melhora ou piora da qualidade. Por exemplo, se a variância nas medidas diminui, isso indica uma melhora da fabricação, enquanto um aumento pode indicar um problema como desgaste das máquinas. Outro exemplo de análise possível é detectar desvios: se uma série de medidas está acima do valor médio, isso pode indicar necessidade de calibragem. Para um processo seguindo uma distribuição estatística normal, é pouco provável que várias peças estejam todas com dimensões acima da média.

Na década de 1940 surgiram vários organismos ligados à qualidade; por exemplo, a ASQC (American Society for Quality Control), a ABNT (Associação Brasileira de Normas Técnicas) e, ainda a ISO (International Standardization Organization). A Segunda Guerra Mundial também contribuiu com o processo, quando as técnicas de manufatura foram aprimoradas para fabricação de material bélico.

Na década de 1940 o Japão destacou-se como um importante pólo no assunto e contribuiu com diversas novas ferramentas: o método de Taguchi para projeto experimental, a metodologia 5S ou, ainda, os diagramas de causa e efeito de Ishikawa, também conhecido como diagramas espinha de peixe.

No pós-guerra o impulso recebido pelas indústrias se manteve. Os computadores digitais já estavam em uso nessa época, embora estivessem restritos sobretudo a meios militares e acadêmicos. Alguns anos mais tarde, quando as máquinas se tornaram mais acessíveis e um maior número de pessoas as utilizava, a qualidade dos softwares começou a se mostrar um objeto importante.

Nas décadas de 1960 e 1970 ocorreram mudanças tecnológicas importantes que afetaram a construção dos computadores e, em consequência, a de software. O período é conhecido como “crise de software” e as repercussões são sentidas até hoje [Pressman, 2002].

Um dos fatores que exerce influência negativa sobre a qualidade de um projeto é a complexidade, que está associada a uma característica simples: o tamanho das especificações.

Construir um prédio de 10 andares implica tratar um número de problemas muito maior do que os existentes em uma simples residência: a diferença entre as duas construções, é claro, está longe de ser resolvida

com um número de tijolos maior. Em programas de computador, o problema de complexidade e tamanho é ainda mais grave, em razão das interações entre os diversos componentes do sistema.

A mudança tecnológica teve um efeito dramático na produção de software. Num breve período de tempo, os recursos de hardware aumentaram muito e permitiram que produtos mais complexos fossem criados. Traçando um paralelo, seria como se os engenheiros civis, depois de anos construindo apenas casas ou pequenos prédios de 2 ou 3 andares, se vissem repentinamente com a tarefa de construir grandes arranha-céus.

Mas a situação ainda era agravada por outro motivo: os primeiros programadores não possuíam ferramentas como dispomos hoje. A tarefa que enfrentavam poderia ser comparada, em certos aspectos, a erguer prédios empilhando mais e mais tijolos. Embora a noção de ciclo de vida já houvesse sido esboçada [Naur e Randell, 1968], não havia técnicas consagradas de trabalho. Não havia escolas ou sequer profissão de programador; as pessoas aprendiam e exerciam essa atividade de maneira empírica.

Provavelmente a primeira vez em que se utilizou o termo “Engenharia de Software” foi em uma conferência com esse nome, realizada em 1968 na Alemanha. Hoje, mais de trinta anos depois, quais são os problemas enfrentados na construção e utilização de software ? Ao lermos o relatório da conferência de 1968 e outros documentos produzidos na década de 1970, fazemos uma descoberta assustadora: os problemas são os mesmos que encontramos atualmente.

Façamos uma pequena lista:

- Cronogramas não observados;
- Projetos com tantas dificuldades que são abandonados;
- Módulos que não operam corretamente quando combinados;
- Programas que não fazem exatamente o que era esperado;
- Programas que simplesmente param de funcionar.

Os erros parecem estar por toda parte, como uma epidemia que não conseguimos controlar depois de décadas de trabalho e pesquisa. O último exemplo e certamente mais conhecido é o bug do milênio, quando foi predito o apocalipse da sociedade da informação: aviões cairiam, contas bancárias seriam zeradas, equipamentos militares perderiam o controle e bombas seriam disparadas. Embora seja verdade que poucos problemas de fato aconteceram, também é verdade que o erro realmente existia em muitos sistemas.

Qualidade

Não é suficiente dizer que a qualidade de software é importante, você tem que (1) definir explicitamente o que quer dizer “qualidade de software”, (2) criar um conjunto de atividades que ajudarão a garantir que todo o produto de trabalho de engenharia de software exibe alta qualidade, (3) realizar atividades de controle e garantia de qualidade de software em todo o projeto, (4) usar métricas para desenvolver estratégias para aperfeiçoar seu processo de software e, como consequência, a qualidade do produto final. Para tanto todos os envolvidos no processo de engenharia de software são responsáveis pela qualidade.

A qualidade de software é importante, você pode fazer direito ou fazer novamente. Se uma equipe de software enfatiza a qualidade em todas as atividades de engenharia de software, reduz a quantidade de trabalho que tem que refazer. Isso resulta em menores custos e, mais importante, melhor prazo para colocação no mercado.

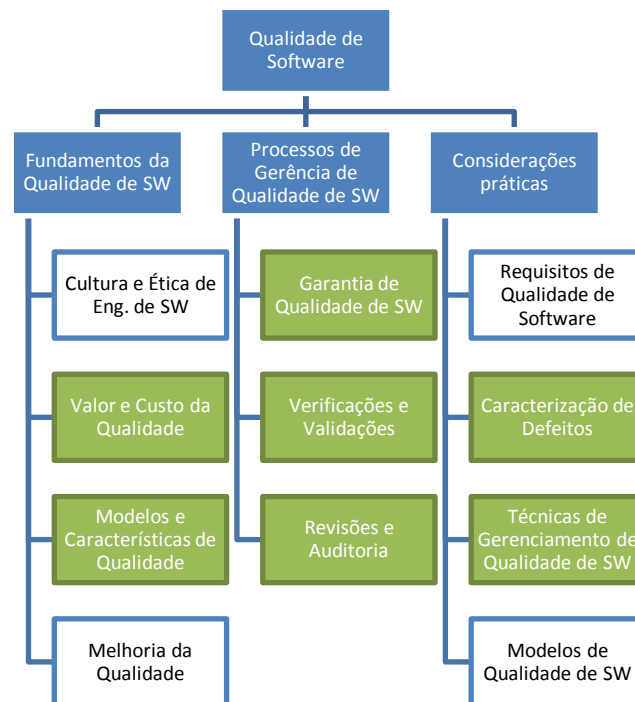


Figura 5 Divisão em tópicos da qualidade - SWEBOK 2004.

Podemos definir qualidade como sendo uma característica ou atributo de alguma coisa. Como atributo de um item, a qualidade se refere a características mensuráveis – coisas que nós podemos comparar com padrões conhecido tais como comprimento, cor, propriedades elétricas e maleabilidade. Todavia, o software, que é essencialmente uma entidade intelectual, é mais difícil de caracterizar do que objetos físicos.

Gestão de qualidade abrange:

1. Um processo de garantia de qualidade de software (*Software Quality Assurance – SQA*);
2. Tarefas específicas de garantia de qualidade e controle de qualidade (incluindo revisões técnicas e estratégia de teste multicamadas);
3. Prática de engenharia de software efetiva (métodos e ferramentas);
4. Controle de todos os produtos de trabalho de software e das modificações feitas neles;
5. Um procedimento para garantir a satisfação de normas de desenvolvimento de software (quando aplicável);
6. Mecanismos de medição e relatório.

Custo da Qualidade

O custo da qualidade inclui todos os custos decorrentes da busca da qualidade ou da execução das atividades relacionadas à qualidade. Os custos da qualidade podem ser divididos em custos associados com a prevenção, com a avaliação e com as falhas. Os *custos de prevenção* incluem o planejamento da qualidade, revisões técnicas formais, equipamento de teste, treinamento. Os *custos de avaliação* incluem atividades para obter entendimento da condição do produto na “primeira execução” de cada projeto. Os *custos de falha* são aqueles que desapareceriam se nenhum defeito aparecesse antes de se entregar um produto ao cliente.

Modelos e Características de Qualidade

Segundo a norma SQuaRE (que é uma evolução das séries de normas ISO/IEC 9126 e ISO/IEC 14598 que tratam da qualidade de produto de software, significa Software product Quality Requirements and Evaluation, ou,

Requisitos de Qualidade e Avaliação de Produtos de Software), que é hierárquico, a qualidade é decomposta em uma série de fatores de influência, conforme imagem abaixo:

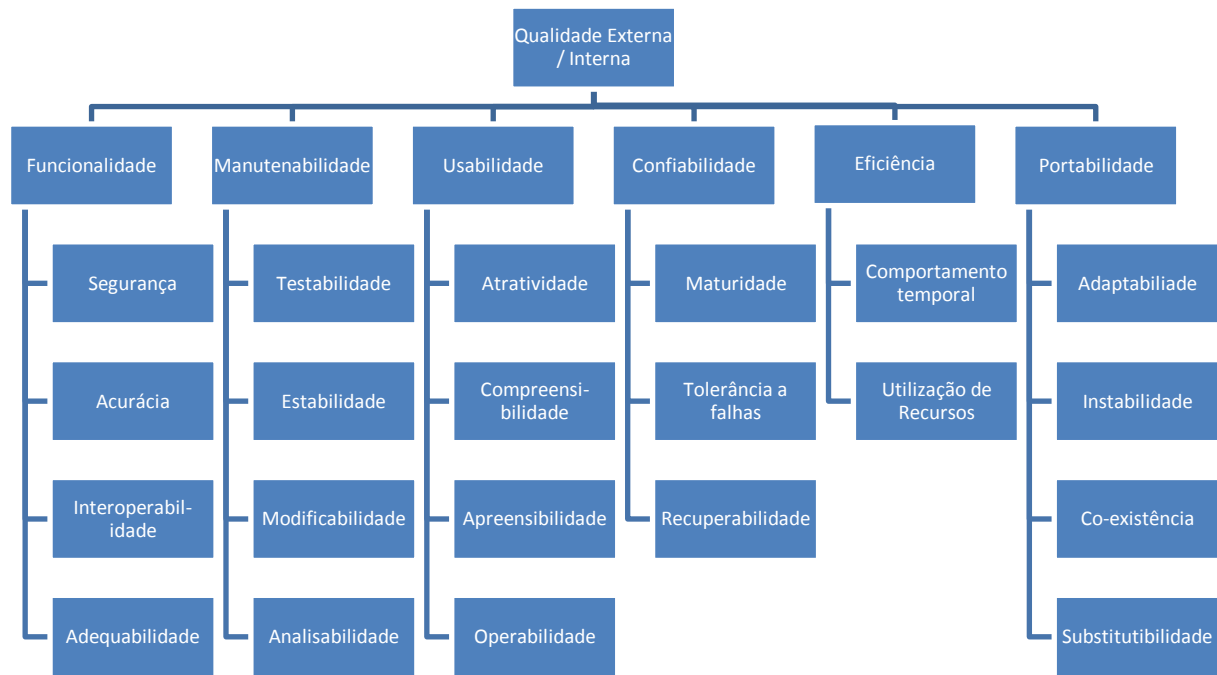


Figura 6 Requisitos de qualidade

Exemplo de Requisitos de Qualidade de Software

Característica: Funcionalidade (F)		
Capacidade para o cumprimento de tarefas.		
<i>Devem ser avaliados os requisitos funcionais do sistema.</i>		
Sub-Característica: Segurança de Acesso (S)		
F01-S	O acesso às funções do programa deverá ser permitida somente a usuários autorizados. Deve ser testada a concessão/retirada de privilégios para o usuário	Essencial
F02-S	Caso existam funcionalidades dentro do programa que necessitem o uso de privilégios especiais, estes devem ser testados.	Depende do Programa
Sub-Característica: Acurácia (C)		
F01-C	Devem ser consistidos os resultados apresentados pelo programa em suas mais diversas saídas: Consulta, Tela de Manutenção (em cálculos de agrupamento), Relatórios, Arquivos de Exportação / Importação.	Depende do tipo de programa
Sub-Característica: Interoperabilidade (I)		
F01-I	As Integrações do programa com outros programas / módulos e/ou rotinas devem ser consistidas. Neste ponto todos os processos que devem ser disparados por este, são de teste obrigatório.	Depende do tipo de programa
F02-I	Quando existirem bibliotecas (DLL) e/ou componentes (OCX, OLE, etc) que são utilizados pelo programa, o seu uso em conjunto com o	Depende do tipo de

	programa deve ser testado.	programa
Sub-Característica: Adequabilidade (Q)		
F01-Q	Caso existam parâmetros que contemplem o funcionamento do programa, estes devem ser testados. Deverá ser verificado para isto o escopo dos parâmetros, situação em que os mesmos podem influenciar outros programas e/ou módulos.	Depende do tipo de programa

Característica: Usabilidade (U)		
Capacidade de mostrar o quão fácil é operar o produto		
Sub-Característica: Operabilidade (O)		
Id	Teste	Escopo
U01-O	O programa deverá apresentar help de contexto para campos.	Essencial
U02-O	Em tela de processamento o botão cancelar deve estar disponível e ao clicá-lo deve solicitar ao usuário a confirmação para interromper ou não a rotina. Deve ser verificado se o processo está sendo interrompido quando isto ocorrer.	Depende do processo
U03-O	Deverá ser consistido o TAB-ORDER de cada campo na tela, ou seja, uma seqüência lógica de objetos deve ser mantida. Deverá utilizar tanto TAB como ENTER para passar de um objeto para outro.	Essencial
Sub-Característica: Apreensibilidade (A)		
U01-A	Ao pressionar CTRL+F12 no programa deve ser mostrada a tela contendo informações sobre o mesmo: Nome do Programa, Nome da Tela, Versão e Data de Compilação, são informações mínimas.	Essencial
U03-A	É necessária uma interface de fácil compreensão, utilizando-se para isto de uma interface intuitiva, que utilize meios de comunicação já conhecidos pelo usuário. Ex: A posição dos botões deve ser a mesma, o texto dos botões deve ser o mesmo.	Essencial
Sub-Característica: Compreensibilidade (C)		
U01-C	O uso de vocabulário adequado na interface com o usuário deve ser considerado. Ex: Item de menu, texto de campos, texto de botões, mensagens de advertência/erro, etc., ou seja, todo tipo de texto que faz interface com o usuário.	Essencial
Sub-Característica: Atratividade (T)		
U01-T	Nos relatórios títulos devem estar com as letras em formato maiúsculo , demais informações (como sub-títulos e títulos de colunas) devem estar com a primeira letra em maiúscula e o restante em minúsculo .	Essencial

Característica: Confiabilidade (C)		
Capacidade de manter um certo nível de desempenho quando operando em um certo contexto de uso		
Sub-Característica: Tolerância (T)		
Id	Teste	Escopo
C01-T	O programa deverá na medida do possível se recuperar em caso de falhas inesperadas. Ex: Quando uma tabela não existir no banco de dados e uma consulta for acessada. Neste caso seria desejável que o programa contornasse o problema, evitando erros ou, no pior caso, encerrando o software de forma controlada.	Essencial
Sub-Característica: Recuperabilidade (R)		
C01-R	Após um erro inesperado ocorrer e o programa se auto-recuperar, o mesmo deve voltar a funcionar corretamente.	Depende da ocorrência de erros inesperados

Garantia da Qualidade

Garantia da qualidade consiste de um conjunto de funções para auditar e relatar que avalia a efetividade e completude das atividades de controle de qualidade. A meta da garantia da qualidade é fornecer à gerência os dados necessários para que fique informada sobre a qualidade do produto, ganhando assim compreensão e confiança de que a qualidade do produto está satisfazendo suas metas.

Garantia da Qualidade de Software

Podemos redefinir a qualidade de software como: Conformidade com requisitos funcionais e de desempenho *explicitamente* declarados, normas de desenvolvimento *explicitamente* documentadas e características implícitas, que são esperadas em todo software desenvolvido profissionalmente. Esta definição serve para enfatizar três pontos importantes:

1. Os requisitos de software são a base pela qual a qualidade é medida. A falta de conformidade com os requisitos é a falta de qualidade;
2. As normas especificadas definem um conjunto de critérios de desenvolvimento que guia o modo pelo qual o software é submetido à engenharia. Caso contrário ocorrerá falta de qualidade;
3. Um conjunto de requisitos implícitos freqüentemente não é mencionado (por exemplo o desejo de facilidade de uso e boa manutenibilidade). Tanto os requisitos explícitos quanto os requisitos implícitos devem ser satisfeitos em um projeto de software, caso contrário a qualidade será suspeita.

A garantia da qualidade é composta de uma variedade de tarefas associadas a dois grupos distintos – os engenheiros de software que fazem o trabalho técnico (relativos à análise e desenvolvimento) e um grupo de SQA, que tem responsabilidade pelo planejamento, supervisão, registro, análise e relato da garantia da qualidade.

Os engenheiros buscam a qualidade (e desenvolvem atividades de garantia de qualidade e de controle de qualidade) aplicando métodos e medidas técnicas sólidas, conduzindo revisões técnicas formais e efetuando teste de software bem planejado.

O grupo de SQA é ajudar a equipe de software a conseguir um produto final de alta qualidade. Um grupo independente de SQA conduz as seguintes atividades:

- *Prepara um plano de SQA para um projeto*: O plano identifica avaliações a serem realizadas, auditorias e revisões a serem realizadas, normas que são aplicáveis ao projeto, procedimentos para relato e acompanhamento de erros, documentos a serem produzidos pelo grupo de SQA e quantidade de realimentação fornecida à equipe de projeto de software;
- *Participa do desenvolvimento da descrição do processo de software do projeto*: Verificar a satisfação da política empresarial, padrões internos de software, padrões externamente impostos (por exemplo, ISO-9001) e outras partes do plano de projeto de software;
- *Revê as atividades da engenharia de software encomendado para verificar a satisfação do que foi definido como parte do processo de software*: Revisão dos produtos selecionados do trabalho, identificação, documentação e acompanhamento de desvios, verificação periódica das correções;
- *Garante que os desvios do trabalho de software e dos produtos do trabalho sejam documentados e manipulados de acordo com um procedimento documentado*: Os desvios podem ser encontrados no plano de projeto, na descrição do processo, nas normas aplicáveis ou nos produtos do trabalho técnico;
- *Registra qualquer eventual não-satisfação e a relata à gerência superior*: Os itens que não atendem ao padrão são acompanhados até que sejam resolvidos.

Revisões de Software

Servem como um “filtro” para o processo de engenharia de software. Isto é, as revisões são aplicadas em vários pontos durante a engenharia de software (análise, projeto e codificação) e servem para descobrir erros e defeitos que podem depois ser removidos.

Uma das atividades da revisão de software são as *revisões técnicas formais*, que é uma atividade de garantia de qualidade de software realizada por engenheiros de software (e outros). Os objetivos das FTR são (1) descobrir erros na função, na lógica ou na implementação, para qualquer representação de software; (2) verificar se o software sob revisão satisfaz seus requisitos; (3) garantir que o software tenha sido representado de acordo com padrões predefinidos; (4) conseguir software que seja desenvolvido de modo uniforme; e (5) tornas os projetos mais administráveis. Além disso, ela serve como uma oportunidade de treinamento, permitindo a jovens engenheiros observar abordagens diferentes para a análise, projeto e implementação de software.

Garantia estatística de qualidade de software

É uma tendência cada vez mais forte, pois reflete uma tendência crescente em toda a indústria, de forner-se quantitativa a respeito da qualidade. Para o software implica nos seguintes passos:

1. Informação sobre defeitos de software é coletada e categorizada;
2. Uma tentativa de rastrear cada defeito até sua causa subjacente (por exemplo, não-conformidade com as especificações, erro de projeto, violação de normas, pouca comunicação com o cliente) é feita;
3. Usando o princípio de Pareto (onde, para 80% dos defeitos, podem ser rastreados 20% de todas as causas possíveis), isole os 20% (os “pouco vitais”);
4. Uma vez identificadas as poucas causas vitais, trate de corrigir os problemas que causaram os defeitos.

Controle de Qualidade

O controle de qualidade envolve a série de inspeções, revisões e testes usada ao longo do processo de software para garantir que cada produto de trabalho satisfaça os requisitos para ele estabelecidos. O controle de qualidade inclui um ciclo de realimentação no processo de trabalho que criou o produto. A combinação de medida e realimentação nos permite ajustar o processo quando os produtos de trabalho criados deixam de satisfazer suas especificações.

Caracterização de Defeito e Falha

Qual a melhor palavra para explicar que um programa “travou” ou não funciona corretamente ? Embora evoquem idéias parecidas, defeito, erro e falha não são sinônimos entre si e são usadas para designar conceitos distintos.

Defeito

Defeito é uma imperfeição de um produto. O defeito faz parte do produto, e em geral, refere-se a algo que está implementado no código de maneira incorreta.

Considere, por exemplo, o código a seguir:

```
a : integer;  
b : integer;  
c : integer;  
readln a;  
readln b;  
c = a /b
```

Esse código pode representar um defeito, pois a divisão poderá resultar em um erro de divisão por zero, que pode, se não controlado, gerar um defeito para o programa executável.

Mas a palavra “defeito” não significa apenas um problema que faz um programa não funcionar. Um programa defeituoso, segundo o dicionário Houaiss, é um programa “que não funciona como deve”.

Falha

Falha é o resultado errado provocado por um defeito ou condição inesperada. No exemplo anterior de código defeituoso – uma divisão por zero – e possível que o programa funcione durante longos anos, sem que jamais ocorra uma falha: tudo dependerá dos valores retornados para A e B.

Falhas também podem ocorrer por fatores externos ao programa, como corrupção de bases de dados ou invasões de memória por outros programas (access violation).

Parte II – Análise

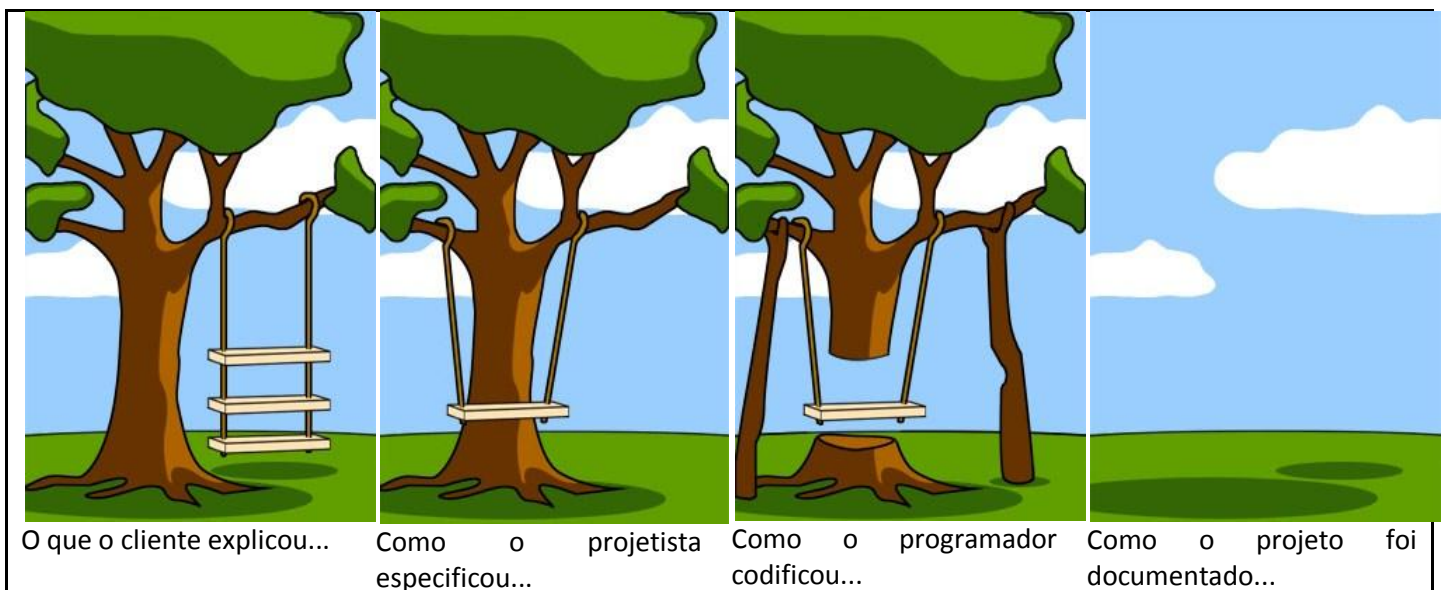
Planejamento é essencial, e o desenvolvimento de software não é exceção. Alcançar o sucesso no desenvolvimento de software requer planejamento, o qual, neste caso, envolve decidir quais tarefas precisam ser feitas, em que ordem fazê-las e que recursos são necessários para cumpri-las.

Uma das primeiras atividades é decompor as grandes tarefas em pequenas tarefas. Isso significa encontrar partes identificáveis das tarefas, que podem ser usados para medir o progresso. A estrutura de decomposição de trabalho (*work breakdown structure – WBS*) deve ser uma estrutura de árvore. O nível mais elevado é usualmente identificado pelo modelo de ciclo de vida (*life cycle model – LCM*) usado na organização. O próximo nível de decomposição pode se identificar como os processos do modelo de processo (*process model – PM*) da organização.

A figura abaixo ilustra uma situação em que um cliente procura uma empresa dizendo precisar de um software. O Analista de sistemas o escuta com toda a atenção e faz uma série de perguntas. Posteriormente, ele escreve durante alguns minutos e faz um esboço do problema. Supondo que o cliente não consiga expressar corretamente suas necessidades em relação ao programa, o analista acrescenta um toque pessoal, de maneira a cobrir certas lacunas. Afinal, por falta de conhecimento na área; o cliente tem dificuldades para explicar o que precisa.

Posteriormente em reunião com a equipe do projeto, o analista expõe suas anotações. O Gerente de projeto decide fazer algumas modificações para reduzir o cronograma inicial e cortar os custos de implementação. A documentação toda passa, em seguida, para o projetista. Prevendo que os cortes realizados pelo gerente do projeto irão impactar na solução final, ele faz algumas modificações com base em sua própria experiência sobre o assunto e projeta a solução a ser implementada.

Finalmente o projeto completo chega ao programador. Ao realizar uma análise mais detalhada ele descobre que está diante de um software que lhe custará muito trabalho. Sendo assim, ele reutiliza componentes e funções já implementadas em outras soluções, poupando assim trabalho e sobrando mais tempo para os testes.



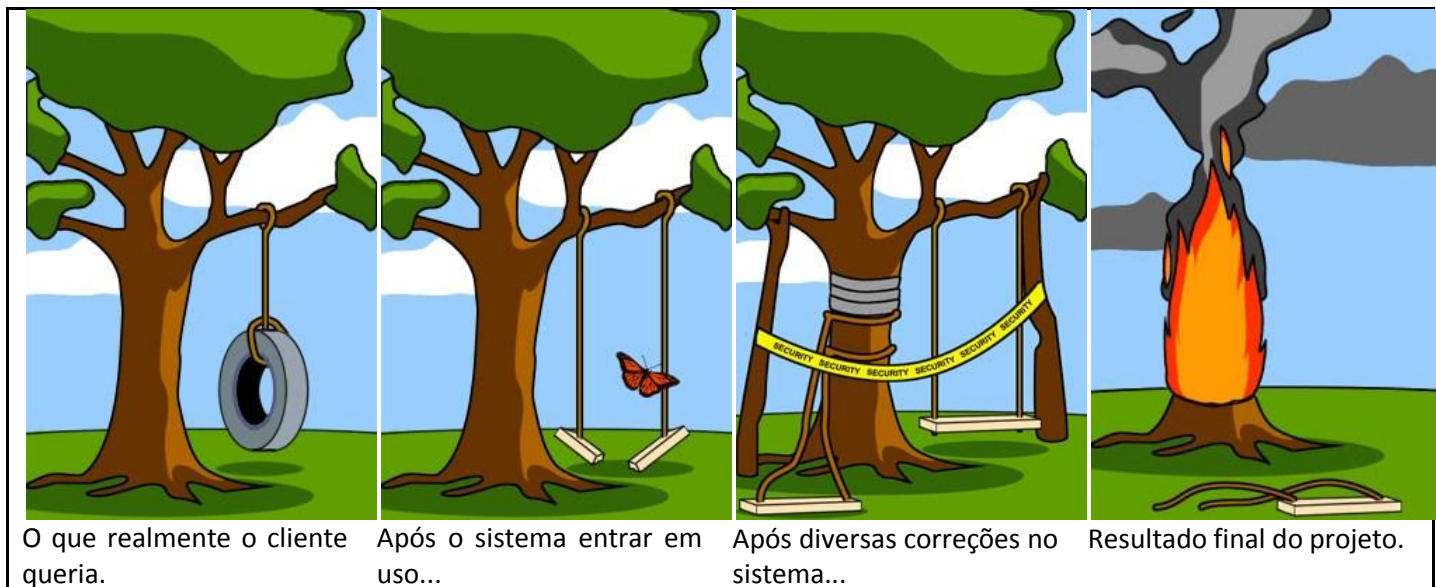


Figura 7 Problemas de Comunicação na produção de software

Considerando a situação em que as características de um produto não estão escritas em nenhum documento e são transmitidas apenas oralmente, temos então como resultado uma documentação muito pobre, havendo muita possibilidade de que as informações sejam compreendidas de diversas maneiras por diferentes indivíduos.

Processo de Análise

Introdução

Classicamente, o processo de desenvolvimento de software divide-se em quatro grandes fases: análise, projeto, implementação e testes. Claro, que essas quatro grandes fases podem ser classificadas e divididas de formas diferentes dependendo de cada organização.

A análise enfatiza a investigação do problema. O objetivo é levar o analista a investigar e a descobrir. Para que essa etapa seja realizada em menos tempo e com maior precisão, deve-se ter um bom método de trabalho.

Essa etapa é importantíssima porque ninguém é capaz de entender com perfeição um problema usual de sistemas de informação na primeira vez que o olha. Assim, esse tempo de análise deve ser bem aproveitado na investigação do problema. Entretanto deve-se ter em mente que erros de concepção sempre vão existir.

Pode-se dizer que o resultado da análise é o enunciado do problema e o projeto a sua resolução. Problemas mal enunciados podem até ser resolvidos, mas a solução não corresponderá às expectativas.

A qualidade do processo de análise é importante porque um erro de concepção resolvido na fase de análise tem um custo; na fase de projeto, um custo maior; na fase de implementação, maior ainda; e na fase de implantação do sistema, um custo relativamente astronômico.

Quando um profissional diz que não tem tempo para fazer uma boa análise porque o patrão quer o projeto “para ontem”, pode-se perceber uma certa inconsistência no processo produtivo da empresa, pois, se nunca há tempo para fazer uma boa análise, sempre há tempo para consertar um programa malfeito (Booch, 1996). E normalmente esse tempo é muito maior.

Projetar e construir software de computador é desafiador, criativo e gostoso. De fato, construir software é tão compulsivo que muitos programadores desejam começar logo, antes de terem um entendimento claro do que é necessário. Eles alegam que as coisas vão ficar mais claras à medida que eles constroem; que os interessados no

projeto vão poder entender melhor as necessidades após examinar as primeiras iterações do software; que as coisas se modificam tão rapidamente que a engenharia de requisitos é perca de tempo. Todos esses argumentos possuem um ponto de verdade, porém, cada um deles é errado e pode levar a um projeto falho de software.

Antigamente, as fases do desenvolvimento de software eram monolíticas. No ciclo de vida waterfall (Boehm, 1976), o analista terminava a análise do sistema, para ai fazer o projeto do sistema e, depois de pronto fazer a programação. Por isso a maioria dos sistemas antigos não tem documentação atualizada, pois somente quando iniciava a fase de implementação é que muitos problemas eram detectados.

Aplicação

Levando em consideração o desenvolvimento de software, a análise é o caminho utilizado para encontrar as soluções para os problemas. Geralmente boa parte dos problemas é de grande complexidade, difíceis de resolver, especialmente quando representam algo novo, nunca antes resolvido. Assim sendo, uma boa técnica é dividir o problema em pequenas partes, formando mais fácil a manipulação e entendimento.

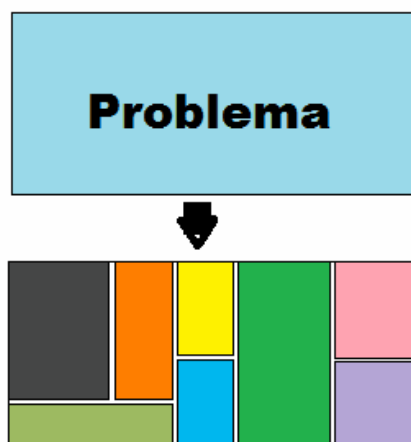


Figura 8 Representação de Problema complexo

Sub-Problemas:

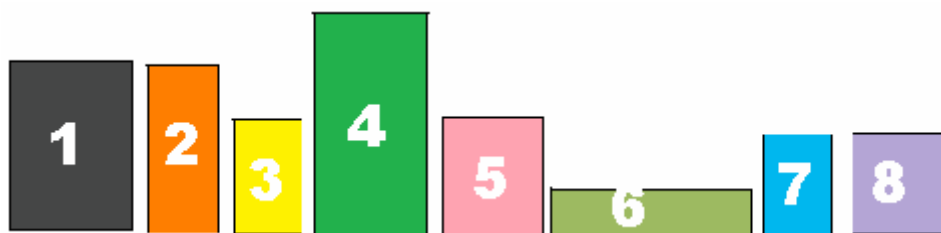


Figura 9 Divisão em Sub-Problemas

Através da análise do problema em pequenas partes, ou subproblemas, obtêm-se soluções individuais que quando compostas através do processo de síntese formam a solução para o problema.

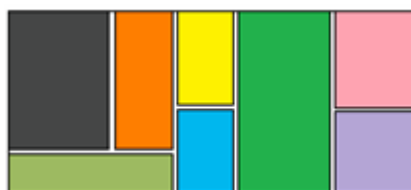


Figura 10 Soluções Reagrupadas

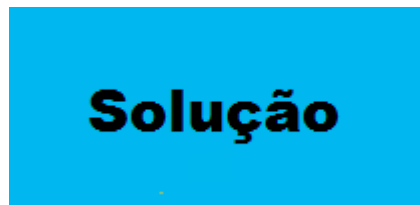


Figura 11 Problema Solucionado

Fluxo de Análise

Para cada subproblema encontrado recomenda-se utilizar o seguinte fluxo de análise:



O processo de análise deve ser conduzido sempre utilizando-se documentação simples e objetiva, para alinhar informações e eliminar conflitos entre as partes.

Regras práticas de Análise

Abaixo é mostrada uma série de “boas práticas” na hora de realizar o processo de análise:

- O Modelo deve focalizar os requisitos que são visíveis no problema ou domínio do negócio. Não se atole em detalhes, que tentam explicar como o sistema funcionará;
- Cada elemento do modelo de análise deve contribuir para um entendimento global dos requisitos do software e fornecer uma visão aprofundada do domínio da informação, função e comportamento do sistema;
- Adie as considerações de modelos de infra-estrutura e outros não funcionais para o projeto;
- Certifique-se de que o modelo de análise tem valor para todos os interessados;
- Mantenha o modelo tão simples quanto puder.

Abordagens de modelagem de análise

Uma visão de análise chamada de análise estruturada (iremos estudar mais a frente), considera os dados e os processos que transformam os dados em entidades separadas. Objetos de dados são modelados para que definam seus atributos e relacionamentos.

Uma segunda abordagem para modelagem de análise, chamada de análise orientada a objetos, focaliza a definição de classes e o modo pelo qual elas colaboram umas com as outras para atender aos requisitos do cliente. UML e Processo Unificado são predominantemente orientados a objetos.

Requisitos de Software

Os requisitos expressam as propriedades (características) e restrições do software do ponto de vista de satisfação das necessidades do usuário/cliente, em geral dependem da tecnologia empregada na construção da solução, sendo a parte mais crítica e propensa a erros no desenvolvimento de software. Tradicionalmente os requisitos são separados em requisitos funcionais e não-funcionais.

Funcionais

São conhecidos como a descrição das diversas funções que usuários querem ou precisam que o software faça. Eles definem a funcionalidade desejada do software. O termo função é usado no sentido genérico de operação que pode ser realizada pelo software, seja através de comandos dos usuários ou seja pela ocorrência de eventos internos ou externos ao software.

A especificação de um requisito funcional deve determinar o que se espera que o software faça, sem a preocupação de como ele faz.

Exemplos:

- O Software deve possibilitar o cálculo dos gastos diários, semanais, mensais e anuais com pessoal;
- O Software deve emitir relatórios de compras a cada quinze dias;
- Os usuários devem obter o número de aprovações, reprovações e trancamentos em todas as disciplinas por um determinado período de tempo.

Não-Funcionais

São compreendidos pelas qualidades globais de um software, como manutenibilidade, usabilidade, desempenho, custos e várias outras. Normalmente estes requisitos são descritos de maneira informal, de maneira controversa (por exemplo, o gerente quer segurança mas os usuários querem facilidade de uso) e são difíceis de validar.

Exemplos:

- A base de dados deve ser protegida para acesso apenas de usuários autorizados;
- O tempo de resposta do software não deve ultrapassar 30 segundos;
- O software deve ser operacionalizado no Linux;
- O tempo de desenvolvimento não deve ultrapassar seis meses;

Tipos de Requisitos

Os requisitos são divididos em 3 grupos distintos:

- **Explícitos:** São aqueles descritos em um documento que arrola os requisitos de um produto, ou seja, um documento de especificação de requisitos;
- **Normativos:** São aqueles que decorrem de leis, regulamentos, padrões e outros tipos de normas a que o software deve obedecer;
- **Implícitos:** São expectativas, são cobrados pelos usuários, embora não documentados.

Engenharia de Requisitos

Entender os requisitos de um problema está entre as tarefas mais difíceis enfrentadas por um engenheiro de software.

A engenharia de requisitos ajuda os engenheiros de software a compreender melhor o problema que eles têm para resolver. Ela inclui o conjunto de tarefas que levam a um entendimento de qual será o impacto do software sobre o negócio, do que o cliente quer e de como os usuários finais vão interagir com o software.

Projetar e construir um software elegante que não resolva o problema de ninguém, essa é a razão pela qual é importante entender o que o cliente deseja antes de começar a projetar e construir um sistema baseado em computador.

A engenharia de requisitos precisa ser adaptada às necessidades do processo, do projeto, do produto e do pessoal que está fazendo o trabalho. Na perspectiva do processo de software a ER é uma ação que começa durante a atividade de comunicação e continua durante a atividade de modelagem.

Tarefas da engenharia de requisitos

A engenharia de requisitos fornece o mecanismo apropriado para entender o que o cliente deseja, analisando as necessidades, avaliando a exequibilidade, negociando uma condição razoável, especificando a solução de modo não ambíguo, validando a especificação e gerindo os requisitos à medida que eles são transformados em um sistema operacional.

O processo de engenharia de requisitos é realizado por meio da execução de sete funções distintas: *concepção, levantamento, elaboração, negociação, especificação, validação e gestão*.

É importante ressaltar que algumas dessas atividades ocorrem em paralelo e que todas são adaptadas ao projeto. Iremos enfatizar o estudo das funções: *Levantamento, Especificação, Validação e Gestão de Requisitos*.

Levantamento

Aparentemente parece muito simples – pergunte ao cliente, aos usuários e aos outros quais são os objetivos do sistema ou do produto, o que precisa ser conseguido, como o sistema ou o produto se encaixa nas necessidades do negócio e, finalmente, como o sistema ou o produto será usado no dia-a-dia. Mas não é simples – é muito difícil.

Vejamos abaixo alguns problemas que mostram por que o levantamento de requisitos é difícil:

- **Problemas de escopo:** O limite do sistema é mal definido ou o usuário/cliente pode especificar detalhes técnicos desnecessários que podem confundir em vez de esclarecer, os objetivos globais do sistema;
- **Problemas de entendimento:** Os clientes/usuários não estão completamente certos do que é necessário, têm pouca compreensão das capacidades e limitações de seu ambiente computacional, não tem pleno entendimento do domínio do problema, omitem informações que acreditam ser “óbvias”;
- **Problemas de Volatilidade:** Os requisitos mudam ao longo do tempo.

Para ajudar a resolver esses problemas deve-se utilizar uma atividade de coleta dos requisitos de um modo organizado.

Especificação

No contexto de software, o termo especificação significa coisas diferentes para pessoas diferentes. Uma especificação pode ser um documento escrito, um modelo gráfico, um modelo matemático formal, uma coleção de cenários de uso, um protótipo ou qualquer combinação desses elementos.

Algumas pessoas sugerem que um gabarito padrão deve ser desenvolvido e usado para especificação de sistemas, argumentando que isso leva a requisitos que são apresentados de um modo consistente e, conseqüentemente, mais inteligível. Esta visão não é totalmente equivocada, porém algumas vezes é necessário o uso do chamado “bom censo” quando uma especificação precisa ser desenvolvida. Para sistemas grandes um documento escrito, combinando descrições em linguagem natural e modelos gráficos pode ser a melhor abordagem. Cenários de uso podem, entretanto, ser tudo o que é necessário para produtos ou sistemas menores que residam em ambientes técnicos bem entendidos.

A especificação é o produto de trabalho final produzido pelo engenheiro de requisitos. Ela serve como fundamento das atividades de engenharia de software subsequentes.

Nome projeto:			Data:	
Nome entrevistado:				
Módulo Projeto:				
Requisitos	Entrada	Processo	Saída	
Objetivo (Escopo):				
Problema:				
Funcionamento:				
Saídas Esperadas:				
Aceite do Entrevistado:			Assinatura:	

Modelo Básico de Ficha de entrevista

Validação de Requisitos

Os produtos de trabalho resultantes da engenharia de requisitos são avaliados quanto à qualidade durante o passo de validação. A Validação de requisitos examina a especificação para garantir que todos os requisitos do software tenham sido declarados de modo não ambíguo; que as inconsistências, omissões e erros tenham sido detectados e corrigidos e que os produtos de trabalho estejam de acordo com as normas estabelecidas para o processo, o projeto e o produto.

O principal mecanismo de validação de requisitos é a revisão técnica formal. A equipe de revisão que valida os requisitos inclui engenheiros de software, clientes, usuários e outros interessados que examinam a especificação procurando por erros do conteúdo ou de interpretação, áreas em que esclarecimentos podem ser necessários, informações omissas, inconsistências (um problema importante quando produtos ou sistemas de grande porte passam por engenharia), requisitos conflitantes ou requisitos irrealísticos (intangíveis).

A medida que cada elemento do modelo de análise é criado, ele é examinado quanto a consistência, omissões e ambigüidade. Os requisitos representados pelo modelo são priorizados pelo cliente e agrupados em pacotes de requisitos que serão implementados como incrementos de software e entregues ao cliente.

Vejamos agora um pequeno checklist para validação de requisitos:

1. Os requisitos foram claramente estabelecidos? Eles podem ser mal interpretados?
2. A fonte (por exemplo, pessoa, regulamento, documento) do requisito foi identificada? A declaração final do requisito foi examinada pela fonte original ou com ela?
3. O requisito está limitado em termos quantitativos?
4. Que outros requisitos se relacionam a este requisito? Eles foram claramente anotados em uma matriz de referência cruzada ou em outro mecanismo?
5. O requisito viola alguma restrição de domínio?
6. O requisito pode ser testado? Em caso positivo, podemos especificar os testes (algumas vezes chamados critérios de validação) para exercitar o requisito?
7. Pode-se relacionar o requisito a qualquer modelo de sistema que tenha sido criado?
8. O requisito está relacionado aos objetivos globais do sistema/produto?
9. A especificação do sistema está estruturada de modo que leve a fácil entendimento, fácil referência e fácil tradução em produtos de trabalho mais técnicos?
10. Foi criado um índice para especificação?
11. Os requisitos associados ao desempenho, ao comportamento e às características operacionais do sistema foram claramente declarados? Que requisitos parecem estar implícitos?

Essas e outras questões devem ser formuladas e respondidas para garantir que o modelo de requisitos reflita adequadamente as necessidades dos clientes e forneça uma base sólida para o projeto.

Gerência de Requisitos

Como sabemos os requisitos para sistemas baseados em computador mudam e que o desejo de mudar os requisitos persiste ao longo da vida do sistema. A gestão de requisitos é um conjunto de atividades que ajudam a equipe de projeto a identificar, controlar e rastrear requisitos e modificações de requisitos em qualquer época, à medida que o projeto prossegue (*a gestão formal de requisitos é iniciada para grandes projetos com centenas de requisitos identificáveis, para projetos menores essa função de engenharia de requisitos é consideravelmente menos formal*). Muitas dessas atividades são idênticas às técnicas de gestão de configuração de software (**GCS**).

A gestão de requisitos começa com a identificação. A cada requisito é atribuído um modo identificador. Uma vez identificados os requisitos, tabelas de rastreamento são desenvolvidas. Mostradas na tabela a seguir, cada tabela de rastreamento relaciona os requisitos identificados a um ou mais aspectos do sistema ou de seu ambiente. Entre muitas tabelas de rastreamento possíveis estão as seguintes:

Tabela de rastreamento de características: Mostra como os requisitos se relacionam a características importantes do sistema/produto observáveis pelo cliente;

Tabela de rastreamento de fontes: Identifica a fonte de cada requisito;

Tabela de rastreamento de dependência: Indica como os requisitos estão relacionados uns aos outros;

Tabela de rastreamento de subsistemas: Caracteriza os requisitos pelo(s) subsistema(s) que eles governam;

Tabela de rastreamento de interface: Mostra como os requisitos se relacionam com as interfaces internas e externas do sistema.

Em muitos casos, essas tabelas de rastreamento são mantidas como parte do banco de dados de requisitos, de modo que elas possam ser pesquisadas rapidamente para entender como a modificação em um requisito afetará diferentes aspectos do sistema a ser construído.

Rastreabilidade

A rastreabilidade é uma técnica utilizada para prover o relacionamento entre os requisitos, projeto e implementação final do software, através dela pode-se visualizar quais os requisitos são claramente ligados às suas fontes e aos artefatos criados durante o ciclo de vida de desenvolvimento do software.

É através da rastreabilidade que pode-se identificar quais os impactos causados por uma mudança nos requisitos de um software. A tabela a seguir apresenta um exemplo de uma matriz de rastreabilidade entre requisitos:

	RF1	RF2	RF3	RF4	RF5
RF1		X			
RF2			X	X	
RF3					
RF4					
RF5	X				

Tabela 1 Matriz de Rastreabilidade

Documentos de Requisitos

Escrever é uma tarefa complexa e propensa a erros. Frequentemente um leitor encontra dificuldades de compreensão, que não são evidentes ao autor em um primeiro momento. Escrever requisitos de forma correta e concisa é fundamental para sua compreensão. Este é um bom motivo para que profissionais da área de computação aperfeiçoem suas habilidades em redigir documentos.

Por vezes, basta uma pequena inconsistência para levar a um sério problema de desenvolvimento. Um caso famoso foi uma falha de comunicação entre projetistas da sonda Mars Polar Orbiter da NASA. Citando um dos relatórios de investigação: “Os dados de impulso... foram transmitidos em libras por segundo em vez da unidade esperada e especificada, em Newtons por segundo”.

O Resultado desse erro de comunicação foi a perda da sonda, a um custo de mais de 100 milhões de dólares.

Existem algumas práticas simples de boa escrita que podem auxiliar a melhorar a qualidade dos textos de requisitos, as quais serão brevemente apresentadas:

- Alocar tempo: a pressa excessiva ao escrever documentos leva quase inevitavelmente a um número maior de erros. O tempo usado na escrita é economizado futuramente, ao serem evitados enganos de interpretação por parte de pessoas diferentes;
- Consistência: requisitos de software são lidos constantemente e por pessoas diferentes. Em uma boa especificação de requisitos todos os leitores devem interpretar as funcionalidades de forma única. Por exemplo, deve-se evitar o uso de sinônimos desnecessários. Escrever “cadastro de compradores” como sinônimo de “inserção de clientes” pode levar a confusões na interface que atrapalhem os usuários, ou até mesmo ocasionar a implementação duplicada de funções.
- Concisão: uma das características de textos técnicos é evitar o uso de adjetivos. Por exemplo, uma especificação poderia declarar que um sistema deve ser rápido. A frase é inútil para um desenvolvedor: ele precisa de dados qualitativos, como o número de requisições por hora a processar.

Não existe um padrão único para o documento de requisito. É recomendável, contudo, que cada organização elabore um formato que seja conhecido e seguido por todas as equipes e pessoas ligadas ao desenvolvimento do software. Independentemente do formato ou do nome dado a esse documento, deve conter:

- Os serviços e funcionalidades que o software deve ter;
- As restrições de operação;
- Propriedades gerais do software;
- Requisitos de hardware;
- Definição de quais outros softwares devem estar integrados.

Diversos padrões internacionais foram propostos para o documento de requisitos, como o IEEE-Std-830-1998, que propõe a estrutura mostrada na tabela abaixo:

Seção	Conteúdo
1. Introdução	
1.1 Propósito do documento	Define o propósito do documento e os leitores a quem se destina.
1.2 Escopo do documento	Identifica o produto e especifica suas funções em alto nível de abstração
1.3 Definições e abreviações	Provê explicações para todas as abreviações e termos utilizados ao longo do documento.
1.4 Referências	Identifica outros documentos, como normas de qualidade.
1.5 Visão geral do restante do documento	Explica a organização do texto.
2. Descrição geral	
2.1 Perspectiva do produto	Explica a relação do software com outros componentes do sistema. Fazem parte desta seção a descrição de interfaces com outros softwares, com redes, com o hardware e com os usuários, além de requisitos de memória e de operação.
2.2 Funções do produto	Descreve as funções a serem desempenhadas.
2.3 Características do usuário	Descreve as exigências em relação aos usuários, tais como conhecimento prévio para operar o software ou necessidades de treinamento.
2.4 Restrições gerais	Quaisquer limitações para os desenvolvedores, como limitações do hardware ou leis específicas.
2.5 Suposições e dependências	Descreve exigências que, se forem modificadas, terão um impacto sobre os requisitos. Por exemplo, supor que um determinado tipo de hardware esteja disponível para executar o produto.
2.6 Requisições adiáveis	Requisitos que podem ser exigidos em uma versão futura do software.
3 Requisitos específicos	Esta seção deve apresentar um detalhamento dos requisitos em um nível que

	permita sua implementação.
Apêndices	...
Índices	...

Resumo

É necessário entender os requisitos antes que o projeto e a construção de um sistema baseado em computador possam começar. Para conseguir isso, um conjunto de tarefas de engenharia de requisitos é conduzido. A engenharia de requisitos ocorre durante as atividades de comunicação com o cliente e modelagem que definimos para o processo genérico de software. Sete funções distintas de engenharia de requisitos - concepção, levantamento, elaboração, negociação, especificação, validação e gestão – são conduzidas pelos membros da equipe de software.

Atividade realizada na aula passada.

Nome projeto:	DNA – Clínica	Data:	01/08/2005
Nome entrevistado:	Pedro Alves		
Módulo Projeto:	Global		
Requisitos	Entrada	Processo	Saída
	Dentistas	Agendamento / Escalonar	Laudo
	Pacientes	Faturamento	Fluxo de Caixa
	Contatos	Fechamento Mensal/Semestral	Ponto de Pedido
	Forma de Pagto	Pagamento	Estatísticas
	Contas Bancárias	Retiradas C/C	Consulta Agenda
	Requisição	Requisição Entrada	
	Mat. Prima		
	Convênios		
Objetivo (Escopo):	Clínica, que faz <u>radiografias</u> , possui <u>dentistas conveniados</u> que enviam os <u>pacientes</u> para serviço. Funcionamento interno, não usa Web.		
Problema:	Atendimento, cumprimento de horários, <u>agenda</u> , agendamentos em horários iguais, <u>escalonar agenda</u> , <u>contatos</u>		
	Finanças, pagamentos, forma de recebimento, que são <u>dinheiro</u> e <u>cheque pré-datado</u> (1 + 2 vezes, sem juro/multa)		
	Fluxo de Caixa, hoje é feito em planilha do Excel		
	<u>Conta bancária</u> em nome da clínica, controlado junto ao fluxo de caixa		
Funcionamento:	O dentista encaminha o paciente com uma requisição com detalhes da radiografia, ao final é gerado um faturamento por cliente para pagamento		
	É realizado um <u>fechamento</u> semanal e outro mensal para avaliar <u>faturamento</u> e quantidade de <u>atendimentos</u>		
	A requisição gera um <u>laudo</u> que é feito retirada pelo cliente com a radiografia neste momento o cliente <u>paga</u> pelo serviço		
	Na conta corrente são realizadas <u>retiradas</u> para pagamentos diversos		
	É realizado serviço de <u>mala direta</u>		
	Existem vários <u>tipos</u> de radiografia, cada tipo leva um <u>tempo estimado</u> de trabalho. Ao agendar deve-se levar em conta os tempos de cada radiografia		
	Uma requisição pode <u>solicitar vários tipos</u> de radiografia		
	Possui <u>estoque</u> , os <u>pedidos</u> são realizados de tempos em tempos. Existe um número mínimo de unidades que cada matéria prima deve ter.		
	Cada tipo de radiografia pode usar <u>chapas</u> de diferentes tamanhos		
	As chapas são a matéria prima para as radiografias, os produtos em estoque.		
Saídas Esperadas:	Estatística de atendimentos p/dentista p/período (relatório)		
	Estatística de atendimentos p/cliente p/período (relatório)		
	Laudo para envio ao dentista		
	Estatística por tipo de radiografia		
	Listagem de Ponto de pedido		
	Consulta de agenda		
Aceite do Entrevistado:			

Técnicas Estruturadas

Análise estruturada é a utilização das seguintes ferramentas:

- Diagramas de Fluxo de Dados
- Dicionário de Dados
- Português Estruturado
- Tabelas de Decisão
- Árvores de Decisão

Utilizadas para a construção de um novo tipo de documento alvo: a Especificação Estruturada. Embora a construção da especificação estruturada seja o aspecto mais importante da análise estruturada, há ainda outros pontos:

- Estimando com heurísticas;
- Métodos que facilitem a transição da análise para o projeto;
- Apoios para geração de teste de aceitação;
- Técnicas de “Walkthrough” (técnica utilizada para detecção de problemas e detecção de erros, através de um trabalho em equipe).

A análise estruturada consiste no estudo de sete componentes, conforme imagem abaixo:

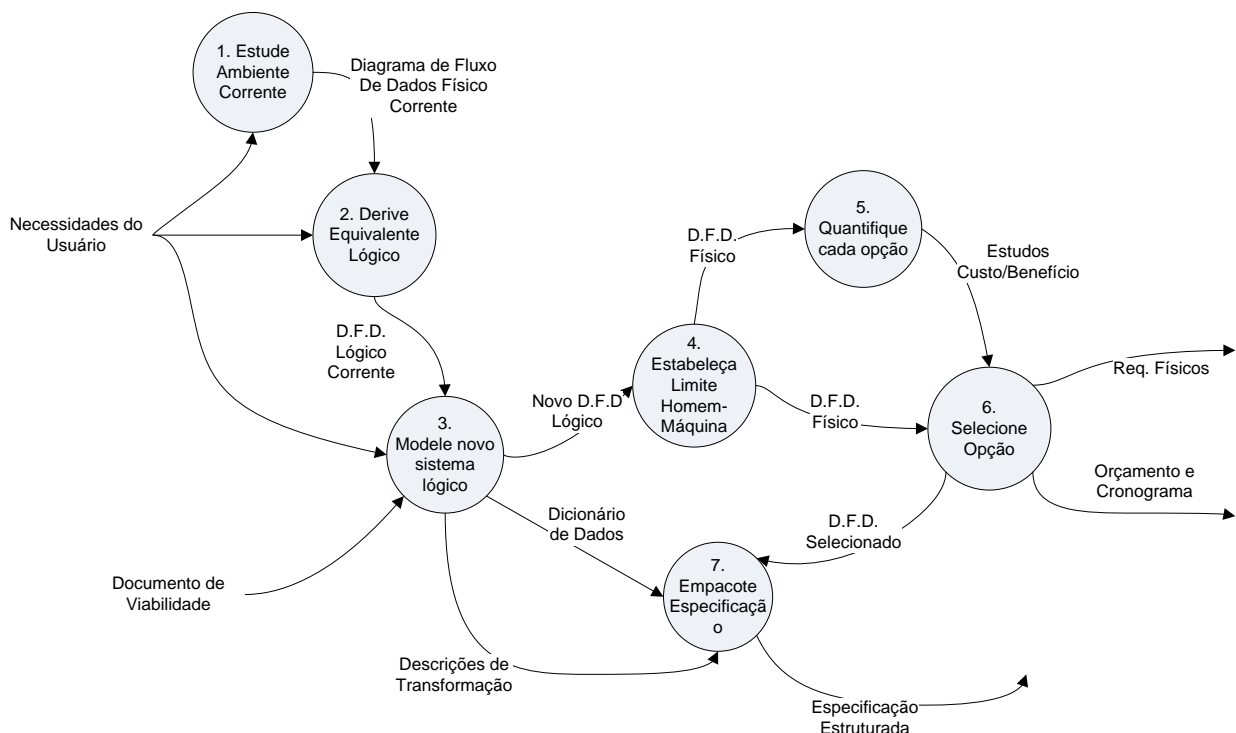


Figura 12 Definição Procedural da Análise Estruturada

- 1. Estudo do ambiente físico corrente, resultando em sua documentação por um Diagrama de Fluxo de Dados Físico Corrente:**

Esta primeira fase é um estudo completo das áreas usuárias afetadas. Antes do seu início, considerar a finalidade do projeto. Isto é necessário para determinar quem são os usuários, e quanto de seu trabalho pode estar sujeito a modificações, se o novo sistema for introduzido.

Trabalhando perto dos usuários aprende-se e documenta-se como as coisas funcionam no momento. Ao invés de fazer isto do ponto de vista de um usuário ou conjunto de usuários, você tenta avaliar operações *sob o ponto de vista dos dados*. Esta abordagem facilitará a construção do Diagrama de Fluxo de Dados, já que o DFD é na realidade, uma imagem de uma situação sob o ponto de vista dos dados.

A documentação resultante desta subfase, deve ser verificável e para isto deve-se fazer o usuário refletir sobre o seu conceito de operações correntes. Então use os termos dele e, em alguns casos, seu particionamento. Isto fará com que o DFD fique cheio de nomenclaturas próprias, sejam elas do usuário ou do seu departamento, números de formulários, etc. Tudo isto servirá para que o usuário possa relacionar o modelo construído com o seu “mundo real”. Já que a maioria destes pontos é de natureza física, isto resultará em um “Diagrama de Fluxo de Dados Físico Corrente”.

Esta fase é considerada completa quando chegamos a um **Diagrama de Fluxo de Dados Físico Corrente Completo**, que descreve a área, e que o usuário o aceitou como uma representação precisa de seu modelo de operação corrente. Vejamos uma check-list de tarefas, que compõe o estudo físico corrente:

- Determinação do contexto a ser estudado;
- Identificação dos usuários afetados (considerar todos os três níveis de usuário, **operacionais, táticos e estratégicos**);
- Entrevistas com o usuário;
- Diagramação do Fluxo de Dados;
- Coleção de tipos de dados de amostra, arquivos e formulários para a complementação dos Diagramas de Fluxo de Dados;
- “Walkthroughs” com o usuário para verificação de correção;
- Publicação da documentação resultante;

2. Derivação do equivalente lógico do ambiente corrente, resultando em um Diagrama de Fluxo de Dados Lógico Corrente;

O próximo passo é “Logicalizar” nosso modelo do ambiente corrente. É basicamente um trabalho de limpeza, no qual consiste em retirar os itens físicos de controle, substituindo cada um pelo seu equivalente lógico. Os objetivos essenciais da operação corrente são separados dos métodos para a realização desses objetivos.

Esta fase se completa quando o Diagrama de Fluxo de Dados Lógico Corrente é desenhado, inspecionado e verificado pelo usuário e publicado.

3. Derivação do novo ambiente lógico, como representado pelo Novo Diagrama de Fluxo de Dados Lógico, mais documentação de apoio;

Até este ponto não foi considerada em nenhum momento a modificação dos métodos de trabalho exigidos no documento de viabilidade. As fases verificadas até agora servem para documentar o ambiente corrente. Somente agora incorporamos a modificação do documento de viabilidade e começamos a descrever o nosso ambiente.

Nosso objetivo ao executar este passo é construir um modelo, no papel, do sistema a ser instalado. O modelo é composto de Diagramas de Fluxo de Dados (para indicar particionamento e interfaces), Dicionário de Dados (para documentar arquivos de fluxo de dados) e Descrições de Transformação (para documentar as partes internas dos processos DFD).

Novamente estamos trabalhando em modo lógico, isto é, estamos tentando descrever o que deve ser feito, e não como será realizado. Nem mesmo distinções de processos automáticos e manuais são feitas.

4. Determinação de certas características físicas do novo ambiente, para produzir um conjunto de Novos Diagramas de Fluxo de Dados Físicos tentativos;

O Processo seguinte envolve a modificação do Novo Diagrama de Fluxo de Dados Lógico para permitir algumas considerações físicas, em particular para responder a seguinte pergunta: Quanto devemos automatizar?, isto envolve demarcar o limite homem-máquina, com isto a parte “Lógico” torna-se o “Físico”, neste ponto apenas definimos a finalidade do sistema automatizado.

Uma boa prática de análise requer a produção de um menu de alternativas funcionais neste ponto,. Além de se fazer isto, produzimos um número de Diagramas de Fluxo de Dados diferenciados, mostrando a variação de níveis quanto à função a ser automatizada.

5. Quantificação de dados relativos a custos e cronograma, associados com cada uma das possibilidades representadas pelo conjunto de Novos Diagramas de Fluxo de Dados Físicos;

Neste processo consideramos cada um dos Novos Diagramas de Fluxo de Dados Físicos tentativas produzidas pelo processo anterior, e tentamos determinar custos e benefícios associados. Observe que até aqui não selecionamos hardware.

É importante que a seleção de hardware espere para ser feita depois que os estágios iniciais do projeto estejam sob controle, porque só então teremos todos os critérios necessários para a seleção estabelecida.

6. Seleção de uma opção, resultando em um novo Diagrama de Fluxo de Dados Físico selecionado;

7. Empacotamento do Novo Diagrama de Fluxo de Dados Físico e documentos de apoio na Especificação Estruturada.

Em cada um dos componentes (ou subfases) acima faz uso do conceito de Diagrama de Fluxo de Dados - DFD. Sobre este assunto especificamente iremos estudar em breve.

O Resultado final, a Especificação Estruturada, consiste em um conjunto integrado de:

- *Diagramas de Fluxo de Dados*, mostrando a maior decomposição de função e todas as interfaces entre as partes;
- *Dicionário de Dados*, documentando cada um dos fluxos de interface e depósitos de dados de qualquer um dos Diagramas de Fluxo de Dados;
- *Descrições de Transformação*, documentando as partes internas dos processos DFD de forma rigorosa (geralmente, através do uso de Português Estruturado, Tabelas de Decisão e Árvores de Decisão).

Diagrama de Fluxo de Dados (Conceitos)

O Diagrama de Fluxo de Dados mostra o fluxo dos dados entre um conjunto de componentes. Os componentes podem ser tarefas, componentes de software ou mesmo abstrações das funcionalidades que serão incluídas no sistema de software. Os atores não são incluídos neste tipo de diagrama. A sequência de ações pode ser inferida a partir da sequência das caixas de atividades.

As regras e interpretações para um diagrama de fluxo de dados correto são:

1. Caixas são processos e devem ser frases com verbos;
2. Linhas representam dados e devem ser nomeados com nomes;
3. Controle não é mostrado. Algumas seqüências podem ser inferidas pela ordem;
4. Um processo pode ser uma unidade única ou pode implicar em um processamento contínuo;
5. Duas linhas saindo de uma caixa podem indicar que ambas as saídas são produzidas ou que uma ou outra é produzida.

Ex: O Cálculo da fórmula matemática $(x + y) * (w + z)$ pode ser mostrada como uma seqüência de operações, como mostrado na imagem abaixo:

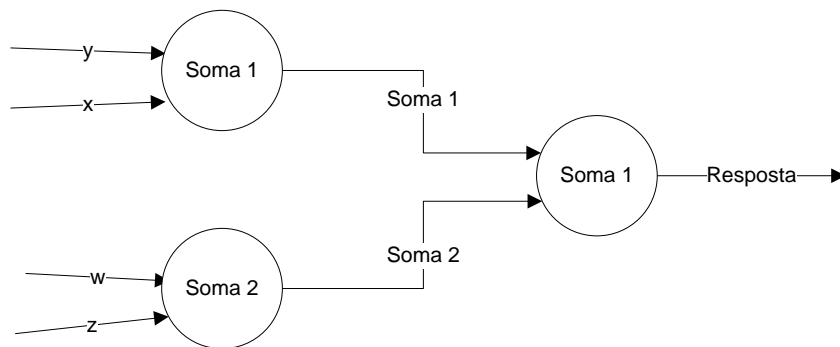


Figura 13 Exemplo de DFD

Os Diagramas de Fluxo de Dados são compostos por somente quatro elementos básicos:

1. Fluxos de Dados, representados por vetores;
2. Processos, representados por círculos ou “bolhas”;
3. Arquivos, representados por linhas retas;
4. Fontes e destinos de dados representados por caixas.

No exemplo baixo é mostrado uma parte de um Diagrama de Fluxo de Dados que inclui cada um dos quatro elementos. Este modelo lê-se como segue:

X's chegam da fonte S e são transformados em Y's pelo processo P1 (que requer acesso ao arquivo A para fazer seu trabalho). Y's são transformados subseqüentemente em Z's pelo processo P2.

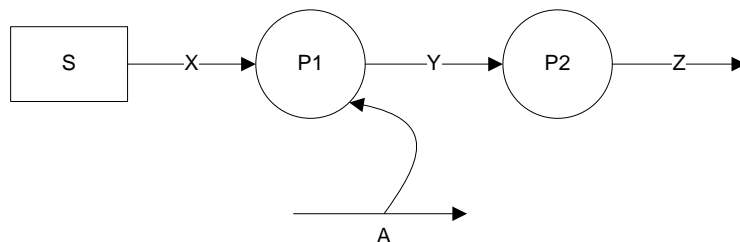


Figura 14 Exemplo 2 de DFD

Projeto de Software

A Engenharia de projeto engloba um conjunto de princípios, conceitos e práticas que levam ao desenvolvimento de um sistema ou produto de alta qualidade. O Objetivo principal da engenharia de projeto é produzir um modelo ou representação que exiba firmeza, comodidade e prazer.

O projeto é o que virtualmente todo engenheiro quer fazer. É o lugar em que a criatividade impera – em que os requisitos do cliente, as necessidades do negócio e as considerações técnicas se juntam na representação do modelo de análise, o modelo de projeto fornece detalhes sobre as estruturas de dados, arquitetura, interfaces e componentes do software necessários para implementar o sistema.

O projeto permite ao engenheiro de software modelar o sistema ou produto que deve ser construído. Esse modelo pode ser avaliado quanto à qualidade e aperfeiçoamento antes que o código seja gerado, testes sejam conduzidos e usuários finais fiquem envolvidos em grande número. Projeto é o lugar em que a qualidade do software é estabelecida.

O projeto mostra o software de vários modos diferentes. Primeiro, a arquitetura do sistema ou produto precisa ser representada. Depois as interfaces que conectam o software aos usuários finais, a outros sistemas e dispositivos e aos seus próprios componentes constitutivos são modelados. Por fim, os componentes de software usados para construir o sistema são projetados. Cada uma dessas visões representa uma diferente ação de projeto, mas todas precisam estar de acordo com um conjunto de conceitos básicos de projeto que norteiam todo o trabalho de projeto de software.

A Engenharia de projeto de software de computador muda continuamente à medida que novos métodos, melhor análise e mais amplo entendimento evoluem. Mesmo atualmente, falta à maioria das metodologias de projeto de software a profundidade, a flexibilidade e a natureza quantitativa, normalmente associadas às disciplinas mais clássicas da engenharia de projeto.

Projeto de software situa-se no núcleo técnico da engenharia de software e é aplicado independentemente do modelo de processo de software usado. Ele inicia-se logo após a engenharia de requisitos e é a última ação do engenheiro de software dentro da atividade de modelagem e prepara a cena para a construção (geração de código e teste).

A importância do projeto de software pode ser definida com uma única palavra – qualidade. Projeto é a etapa na qual a qualidade é incorporada na engenharia de software. O projeto nos fornece representações do software que podem ser avaliadas quanto à qualidade. É o único modo pelo qual podemos traduzir precisamente os requisitos do cliente em um produto ou sistema de software e de suporte de software que se seguem.

Processo de Projeto

Projeto de software é um processo iterativo por meio do qual os requisitos são traduzidos em um “documento” para construção de software. Inicialmente, o documento mostra uma visão holística do software, ou seja, um nível alto de abstração.

Ao longo do processo, a qualidade do projeto em evolução é avaliada com uma série de revisões técnicas formais de projeto, vejamos abaixo uma lista com três características que servem de orientação para a avaliação de um bom projeto:

- *Projeto deve implementar todos os requisitos* explícitos contidos no modelo de análise e acomodar todos os requisitos implícitos desejados pelo cliente;
- *O Projeto deve ser um guia legível*, compreensível para aqueles que geram código e para os que testam e, subseqüentemente, dão suporte ao software;
- *O Projeto deve fornecer um quadro completo do software*, focalizando os domínios de dados funcional e comportamental sob uma perspectiva de implementação.

Fases do processo de projeto:

A seguir, estão as fases do projeto:

- **Projeto dos dados** – Esta fase produz as estruturas dos dados;
- **Projeto arquitetural** – Esta fase produz as unidades estruturais (classes, funções, componentes);
- **Projeto de interface** – Esta fase especifica as interfaces entre unidades;
- **Projeto procedimental** – Esta fase especifica os algoritmos de cada método;

Qualidade de Projeto

Para avaliarmos a qualidade de uma representação de projeto, precisamos estabelecer critérios técnicos para um bom projeto. Abaixo são exibidas algumas das diretrizes:

1. Um projeto deve exibir uma arquitetura que:
 - a. Tenha sido criada por meio de estilos ou padrões arquiteturais reconhecíveis;
 - b. Seja composta de componentes que exibam boas características de projeto;
 - c. Pode ser implementada de modo evolucionário (para projetos menores, o projeto pode ser desenvolvido linearmente), facilitando assim a implementação e o teste.
2. Um projeto deve ser modular, o software deve ser logicamente particionado em elementos ou subsistemas;
3. Um projeto deve conter representações distintas dos dados, arquitetura, interfaces e componentes;
4. Um projeto deve levar a estruturas de dados adequadas às classes a ser implementadas e baseadas em padrões de dados reconhecíveis;
5. Um projeto deve levar a componentes que exibam características funcionais independentes;
6. Um projeto deve levar a interfaces que reduzam a complexidade das conexões entre componentes e com o ambiente externo;
7. Um projeto deve ser derivado por meio de um método passível de repetição que seja conduzido pela informação obtida durante a análise dos requisitos do software;
8. Um projeto deve ser representado usando uma notação que efetivamente comunique seu significado.

Atributos de Qualidade

A Hewlett-Packard desenvolveu um conjunto de atributos de qualidade de software que recebeu a sigla FURPS (em português: Funcionalidade, Usabilidade, Confiabilidade, Desempenho e Suportabilidade). Esses atributos representam uma meta para todo o projeto de software:

- **Funcionalidade:** é avaliada pela observação do conjunto de características e capacidades do programa, generalidade das funções entregues e segurança do sistema global;
- **Usabilidade:** é avaliada considerando fatores humanos, como estética, consistência e documentação globais;

- *Confiabilidade*: é avaliada medindo-se a frequência e a severidade das falhas, a precisão dos resultados de saída, o tempo médio entre falhas, a capacidade de recuperação de falhas e a previsibilidade do programa;
- *Desempenho*: é medido pela velocidade de processamento, tempo de resposta, consumo de recursos, vazão e eficiência;
- *Suportabilidade*: combina a capacidade de estender o programa (extensibilidade), adaptabilidade, reparabilidade – esses três atributos representam um termo mais comum, manutenibilidade – além de testabilidade, compatibilidade, configurabilidade (a capacidade de organizar e controlar elementos de configuração e organização de software).

Nem todo atributo de qualidade de software é igualmente ponderado à medida que o projeto de software é desenvolvido. Uma aplicação pode salientar funcionalidade com ênfase especial em segurança, outra pode exigir desempenho com ênfase particular na velocidade de processamento. Independentemente da ponderação, é importante notar que esses atributos de qualidade devem ser considerados quando o projeto começa, não depois que o projeto estiver completo e a construção, começada.

Independentemente do tamanho do projeto ele deverá seguir um conjunto genérico de tarefas conforme mostrado abaixo:

1. Examinar o modelo de domínio de informação e projetar estruturas de dados adequadas para os objetos de dados e seus atributos;
2. Usando o modelo de análise, selecionar um estilo arquitetural (padrão) que seja adequado para o software;
3. Particionar o modelo de análise em subsistemas de projeto e alocá-los na arquitetura. Certifique-se de que cada subsistema é funcionalmente coeso;
4. Criar um conjunto de classes de projeto ou componentes. Traduzir cada descrição de classe de análise em uma classe de projeto;
5. Projetar qualquer interface exigida por sistemas ou dispositivos externos;
6. Projetar a interface de usuário. Revisar os resultados da análise de tarefas. Criar um modelo comportamental da interface;
7. Conduzir o projeto de componentes. Especificar todos os algoritmos em um nível relativamente baixo de abstração;
8. Desenvolver um modelo de implantação.

Projeto de Software

Um conjunto de conceitos fundamentais de projeto de software tem evoluído durante a história da engenharia de software. Apesar do grau de interesse em cada conceito ter variado ao longo dos anos, cada um resistiu ao teste do tempo.

Abstração

Quando consideramos uma solução modular para qualquer problema, muitos níveis de abstração podem ser colocados. No nível mais alto de abstração, uma solução é enunciada em termos amplos usando a linguagem do ambiente do problema. A medida que nos deslocamos entre diferentes níveis de abstração, trabalhamos para criar abstrações procedimentais e de dados.

Uma *abstração procedural* refere-se a uma sequência de instruções que tem uma função específica e limitada. Um exemplo de abstração procedural seria a palavra *abrir* para uma porta. *Abrir* implica uma longa sequência de passos procedurais lógicos.

Uma *abstração de dados* é uma coleção de dados com uma denominação que descreve um objeto de dados. No contexto da abstração procedural *abrir*, podemos definir uma abstração de dados chamada de **porta**. Como qualquer objeto de dados, a abstração de dados para **porta** abrange um conjunto de atributos que descrevem a porta (**tipo de porta, direção de giro, mecanismo de abertura, peso, dimensões**).

Arquitetura

Arquitetura de software segundo Shaw (95), refere-se à “estrutura global do software e aos modelos pelos quais essa estrutura fornece integridade conceitual para um sistema”. Em termos gerais é a estrutura ou organização dos componentes de programa (módulos), o modo pelo qual esses componentes interagem e as estruturas de dados que são usadas pelos componentes.

Uma meta do projeto de software é derivar um quadro arquitetural do sistema. Um conjunto de padrões de software arquiteturais permite ao engenheiro de software reusar conceitos de projeto.

Padrões

Um padrão de projeto descreve uma estrutura de projeto que resolve um problema particular de projeto em um contexto específico e em meio a “forças” que podem ter impacto na maneira pela qual o padrão é aplicado e usado.

A intenção de cada padrão de projeto é fornecer uma descrição que habilite o projetista a determinar:

1. Se o padrão é aplicável ao trabalho corrente;
2. Se o padrão pode ser reutilizado;
3. Se o padrão pode servir como guia para desenvolvimento de um padrão similar, mas funcionalmente ou estruturalmente diferente dele;

Modularidade

O Software é dividido em componentes nomeados separadamente e endereçáveis, algumas vezes chamados de módulos, que são integrados para satisfazer aos requisitos do problema.

Um software monolítico (um programa grande composto de um único módulo) não pode ser facilmente compreendido por um engenheiro de software. Isso leva à estratégia “dividir e conquistar” – é mais fácil resolver um problema complexo quando você o divide em partes gerenciáveis.

Modularizamos um projeto (e o programa resultante) de modo que o desenvolvimento possa ser mais facilmente planejado; incrementos de software possam ser definidos e liberados; modificações possam ser mais facilmente acomodadas; teste e depuração possam ser conduzidos mais eficientemente; e manutenção a longo prazo possa ser conduzida sem sérios efeitos colaterais.

Ocultamento da informação

O conceito apresentado anteriormente sobre modularidade, leva o projetista de software a uma pergunta: “Como decompomos uma solução de software para obter o melhor conjunto de módulos?”. O princípio de *ocultamento da informação* sugere que os módulos sejam “caracterizados por decisões de projeto que (cada um) esconde de todos os outros”, ou seja, módulos devem ser especificados e projetados para que a informação (algoritmos e dados) contida em um módulo seja inacessível a outros módulos que não necessitam dessa informação.

O uso de ocultamento da informação como critério de projeto para sistemas modulares fornece os maiores benefícios quando são necessárias modificações durante o teste e, posteriormente durante a manutenção do software.

Independência Funcional

O conceito de *independência funcional* é uma decorrência direta da modularidade e dos conceitos de abstração e ocultamento da informação. A independência funcional é obtida pelo desenvolvimento de módulos com função de “finalidade única” e uma “aversão” à interação excessiva com outros módulos.

Módulos independentes são mais fáceis de manter (e testar), porque os efeitos secundários causados por modificações de projeto ou código são limitados, a propagação de erros é reduzida e os módulos reusáveis são possíveis.

Independência é avaliada usando dois critérios qualitativos: coesão e acoplamento. *Coesão* indica a robustez funcional relativa de um módulo, em outras palavras, um módulo coeso deveria fazer apenas uma coisa. *Acoplamento* indica a interdependência relativa entre módulos. Conectividade simples entre módulos resulta em software bem mais fácil de entender e menos propenso a “efeito de propagação”, que acontece quando erros que ocorrem em um lugar se propagam por todo o sistema.

Refinamento

Refinamento é um processo de decomposição passo-a-passo descendente. É um processo de *elaboração*. Começamos com um enunciado da função (ou descrição da informação) que é definida em um alto nível de abstração. O Enunciado descreve a função ou informação conceitualmente. O Refinamento leva o projetista a elaborar o enunciado original, fornecendo mais e mais detalhes à medida que cada refinamento (elaboração) sucessiva acontece.

Refabricação

Uma importante atividade de projeto sugerida por muitos métodos ágeis, *refabricação* é uma técnica de reorganização que simplifica o projeto (ou código) de um componente sem mudar sua função ou comportamento, melhorando a estrutura interna de um sistema sem alterar o seu comportamento externo.

Quando o software é refabricado, o projeto existente é examinado quanto à redundância, elementos de projeto não usados, algoritmos ineficientes ou desnecessários, estruturas de dados impróprias ou pobremente construídas, ou qualquer outra falha de projeto que possa ser corrigida.

Classes de projeto

Durante o processo de análise podemos definir um conjunto completo de classes de análise. Cada uma dessas classes descreve algum elemento do domínio do problema, focalizando aspectos do problema que são visíveis ao usuário ou ao cliente. O nível de abstração de uma classe de análise é relativamente alto.

A medida que o modelo de projeto evolui, a equipe de software deve definir um conjunto de *classes de projeto* que:

1. Refina as classes de análise, fornecendo detalhes de projeto que vão permitir que as classes sejam implementadas;
2. Crie um conjunto de classes de projeto que implemente uma infra-estrutura de software para apoiar a solução do negócio.

As definições de classes de projeto, ficam mais evidentes em projetos de software utilizando conceitos de UML, que veremos em outra oportunidade.

Modelo de Projeto

O modelo e projeto pode ser visto de duas dimensões diferentes, a dimensão *processo* indica a evolução do modelo de projeto à medida que as tarefas são executadas como parte do processo de software. A dimensão *abstração* representa o nível de detalhe à medida que cada elemento do modelo de análise é transformado em um projeto equivalente e, então refinado iterativamente.

Projeto de Software Baseado em Padrão

Durante todo o processo de projeto, um engenheiro de software deve procurar toda a oportunidade de reutilizar padrões de projeto existentes (quando eles satisfazem às necessidades) em vez de criar outros.

Projeto baseado em padrão é uma técnica que reusa elementos de projeto que foram aprovados com sucesso no passado. Cada padrão arquitetural, padrão de projeto ou idioma é catalogado, profundamente documentado e cuidadosamente considerado à medida que ele é avaliado para inclusão em uma aplicação específica.

Descrição de um padrão de projeto

Padrões de projeto podem ser descritos por meio do gabarito mostrado abaixo:

Gabarito de Padrão de Projeto

Nome do Padrão – descreve a assência do padrão em um curo, mas expressivo, nome;

Intenção – descreve o padrão e o que ele faz;

Também-conhecido-como: lista os sinônimos do padrão;

Motivação – fornece um exemplo do problema;

Aplicabilidade – notifica situações específicas de projeto nos quais o padrão é aplicável;

Estrutura – descreve as classes necessárias para implementar o padrão;

Participantes – descreve as responsabilidades das classes que são necessárias para implementar o padrão;

Colaborações – descreve como os participantes colaboram para cumprir suas responsabilidades;

Conseqüências – descreve as “influências do projeto” que afetam o padrão e os potenciais compromissos que devem ser considerados quando o padrão é implementado;

Padrões relacionados – referências cruzadas relacionadas a padrões de projeto.

Uma descrição de padrão de projeto pode também considerar um conjunto de influências de projeto, que em síntese, descrevem requisitos não funcionais (facilidade de manutenção, portabilidade) associados ao software ao qual o padrão deve ser aplicado. Em essência, as influências de projeto descrevem o ambiente e as condições que devem existir para tornar o padrão de projeto aplicável.

Uso de padrões no projeto

Padrões de projeto podem ser usados durante todo o projeto de software. Uma vez desenvolvido o modelo de análise, o projetista pode examinar uma representação detalhada do problema a ser resolvido e as restrições impostas pelo problema.

- **Padrões arquiteturais:** Esses padrões definem uma estrutura global do software, indicam o relacionamento entre subsistemas e componentes do software e definem as regras para especificar relacionamentos entre os elementos (classes, pacotes, componentes, subsistemas) da arquitetura;
- **Padrões de projeto:** Atendem a um elemento específico do projeto tal como uma agregação de componentes para resolver algum problema de projeto.
- **Idiomas ou Padrões de código:** São basicamente padrões e linguagem, geralmente implementam um elemento algorítmico de um componente, um protocolo de interface, ou um mecanismo para comunicação entre componentes.

Frameworks

Em alguns casos pode ser necessário fornecer uma infra-estrutura do esqueleto de implementação específica, chamada de *framework*. O projetista pode selecionar uma espécie de *miniarquitetura reusável* que fornece a estrutura e o comportamento genéricos para uma família de abstrações de software.

Projeto Arquitetural

Projeto arquitetural representa a estrutura dos componentes de dados e programas que são necessários para construir um sistema baseado em computador. Ele considera o estilo arquitetural que o sistema vai adotar, a estrutura e as propriedades dos componentes que constituem o sistema e os inter-relacionamentos que ocorrem entre todos os componentes arquiteturais de um sistema.

Por que é importante? Você não tentaria construir uma casa sem uma planta, tentaria? Você também não começaria a desenhar as plantas esboçando a disposição dos encanamentos da casa. Você precisaria olhar o quadro geral – a casa em si – antes de se preocupar com os detalhes. Isso é o que projeto arquitetural faz – fornece o quadro geral e garante que você faça direito.

Na imagem abaixo existe uma primeira versão de um DFD (contexto) de um sistema de controle financeiro pessoal. As informações neste nível estão no modo mais abstrato possível, de forma conceitualmente demonstradas.

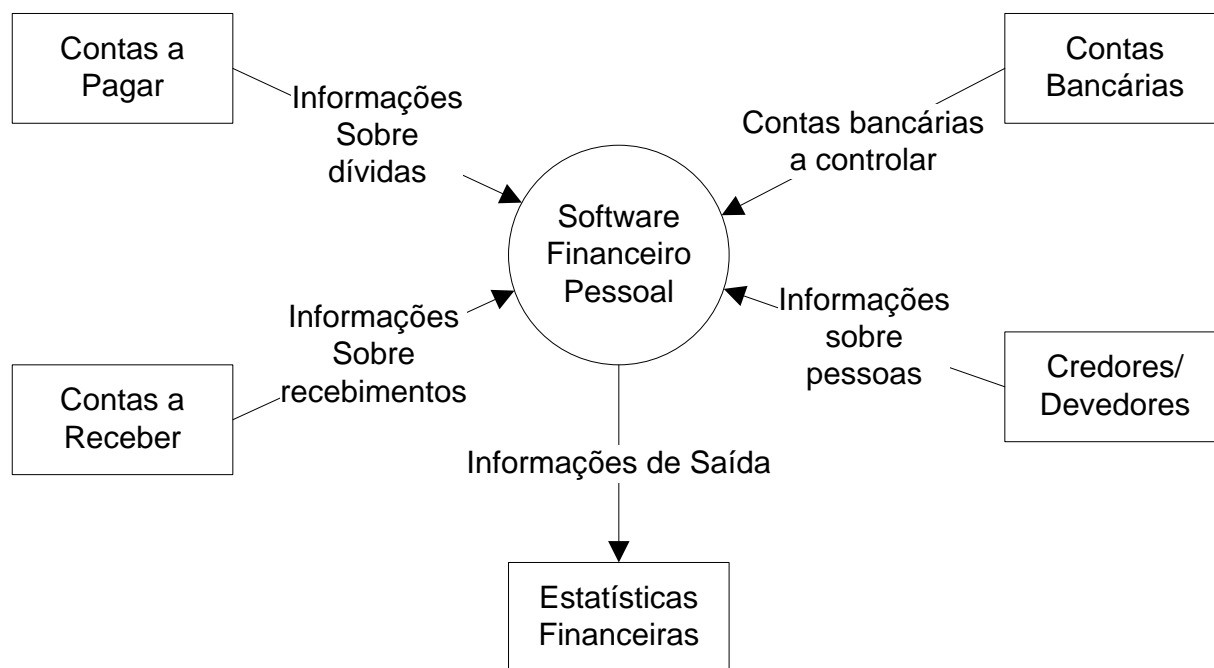


Figura 15 DFD de Contexto

O projeto arquitetural começa com o projeto dos dados e depois prossegue para a derivação de uma ou mais representações da estrutura arquitetural do sistema. Estilos ou padrões arquiteturais alternativos são analisados para derivar a estrutura mais adequada aos requisitos do cliente e aos atributos de qualidade. Uma vez selecionada uma alternativa, a arquitetura é elaborada usando um método de projeto arquitetural.

Projeto no nível de componentes

Um conjunto completo de componentes de software é definido durante o projeto arquitetural. Mas as estruturas de dados internas e detalhes de processamento de cada componente não são representadas em um nível de abstração próximo ao código. Projeto no nível de componente define as estruturas de dados, algoritmos, características de interface e mecanismos de comunicação alocados a cada componente de software.

Por que é importante? Você precisa poder determinar se o software vai funcionar antes de construí-lo. O projeto no nível de componentes representa o software de um modo que lhe permite revisar os detalhes do projeto quanto à correção e consistência com as primeiras representações do projeto (ou seja, os projetos de dados, arquitetural e de interface).

A imagem a seguir mostra o sistema de controle financeiro pessoal em nível de componentes (sinteticamente).

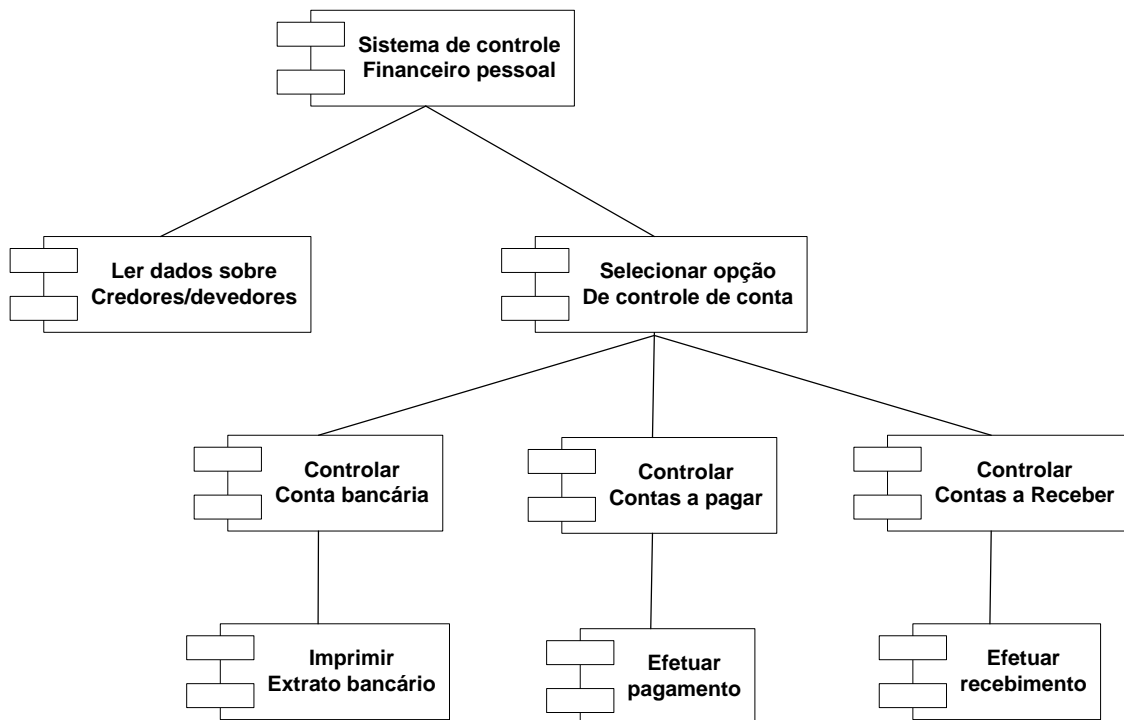


Figura 16 Diagrama de estrutura convencional - componentes

Representações de projeto dos dados, arquitetura e interfaces formam a fundação para o projeto no nível de componentes. A definição de classes ou narrativa de processamento para cada componente é traduzida em um projeto detalhado que faz uso de formas diagramáticas ou baseadas em texto que especificam as estruturas de dados internas, detalhes de interface local e lógica de processamento.

Projeto de Interface com o usuário

O projeto de interface com o usuário cria um meio efetivo de comunicação entre o ser humano e o computador. Seguindo um conjunto de princípios de projeto de interface, o projeto identifica objetos e ações de interface e depois cria um layout de tela que forma a base para um protótipo de interface com o usuário.

Por que é importante? Se o software é difícil de usar, se o leva a cometer erros ou se frustra seus esforços de alcançar suas metas, você não gostará dele, independentemente do poder computacional que exibe ou da funcionalidade que oferece. A interface tem de ser correta, porque ela molda a percepção do software pelo usuário.

O projeto de interface com o usuário começa com a identificação dos requisitos do usuário, das tarefas e do ambiente. Uma vez identificadas as tarefas do usuário, cenários do usuário são criados e analisados para definir um conjunto de objetos e ações de interface.

Os cenários formam a base para a criação do layout de tela que ilustra o projeto gráfico e a colocação de ícones, definição de texto descritivo na tela, especificação e títulos para janelas e especificação de itens de menu principais e secundários. Ferramentas são usadas para prototipar e, por fim, implementar o modelo de projeto, e o resultado é avaliado quanto à qualidade.

O processo de prototipação, que contempla o projeto de interface com o usuário será nosso próximo tema.

Aula 10

Atividade sobre Projeto de Software

Atividade complementar individual. Tempo estimado: 45 min.

Nome: _____ Data: _____

Com base no conteúdo publicado na sala virtual e discutido em sala de aula, sobre engenharia de software, responda as seguintes questões:

1. Com suas próprias palavras faça um breve resumo sobre projeto de software.
2. Como avaliamos a qualidade de um projeto de software?
3. Descreva as fases do processo de projeto de software.
4. Descreva o funcionamento básico do projeto a nível de componentes
5. Por que o projeto arquitetural é importante?

Análise Essencial

Como sabemos para que iniciemos a construção de um sistema de informações, precisamos antes de tudo conhecer o foco da gestão de negócios. Conhecer como o negócio funciona – **os fatos** – para isto podemos utilizar a análise essencial.

A análise essencial se difere da análise estruturada pelos seguintes motivos:

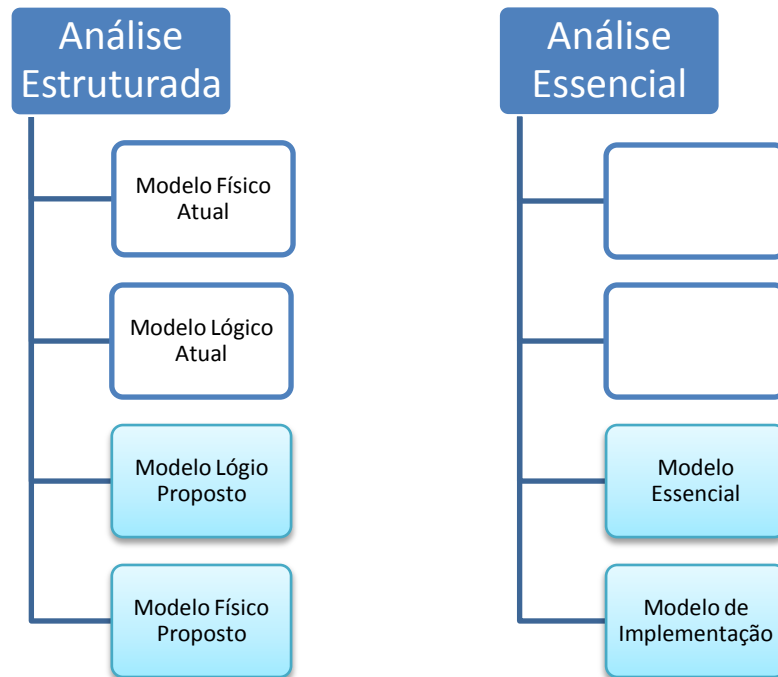


Figura 17 Diferenças entre Análise Estrutural e Essencial

A análise essencial exerce também alguma vantagem sobre a análise estruturada:

- A análise essencial começa pelo modelo essencial, o que equivale, na análise estruturada, começar diretamente pelo modelo lógico proposto;
- A análise estruturada aborda duas perspectivas de sistema – função e dados -, ao passo que a Análise Essencial aborda três perspectivas – função, dados e controle;
- Na análise estruturada o particionamento é feito através da abordagem top-down, enquanto na análise essencial, o particionamento é por eventos

Os eventos são a pedra fundamental dos sistemas, e a especificação de um sistema, segundo McMenamin e Palmer, deve começar pela identificação dos eventos. Concordamos com esta afirmativa, porém não podemos jamais confundir *eventos* com *procedimentos*.

Um fator interessante da análise essencial é a conceituação de modelos de sistema, com a existência de um **modelo essencial** e de um **modelo de implementação**, conforme mostrado na imagem a seguir:

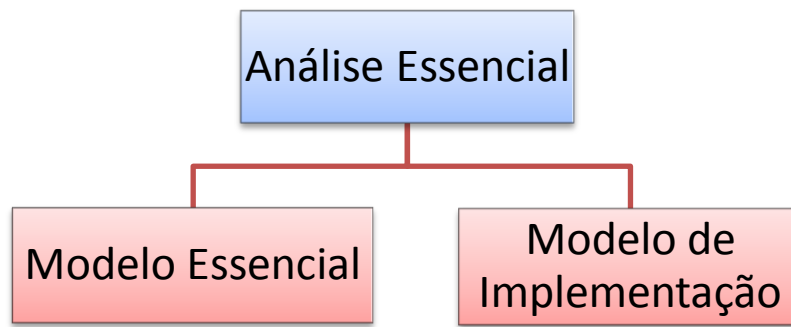


Figura 18 Estrutura do Modelo de análise essencial

O modelo **essencial** apresenta o sistema de informação computacional, em um grau de abstração independente de restrições tecnológicas. É neste nível que devemos trabalhar a análise de eventos, livre de qualquer tipo de restrição tecnológica. O objetivo aqui é sempre retratar o ambiente atual. Este modelo de sistema será o ideal, descrevendo quais os requisitos a que o sistema deve atender, sem se preocupar como isto poderá ou deverá ser implementado.

O modelo de **implementação**, apresenta o sistema em um grau de abstração praticamente dependente na sua íntegra de restrições tecnológicas, sendo derivado do modelo essencial. Correspondente ao modelo físico proposto para ser implementado, considerando as ferramentas de gerenciamento de banco de dados a serem utilizadas e recursos de linguagens de programação.

O **modelo essencial** se divide em dois, sendo um modelo dito **ambiental**, que define os limites do sistema, ou seja, o que pertence ao sistema e que é externo a ele, e o modelo **comportamental** que nada mais é do que a definição funcional das partes do sistema. É o como se faz em cada evento que interage com o sistema.

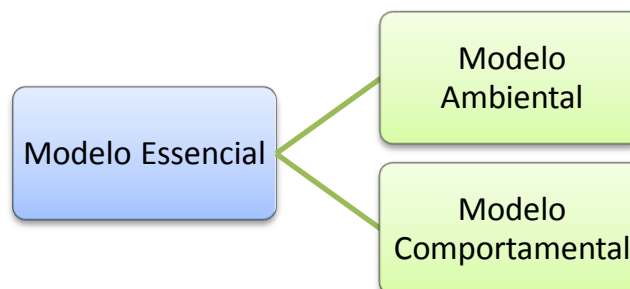


Figura 19 Divisão do modelo Essencial

O modelo essencial envolve dois métodos de análise: a **Modelagem de dados** e a **Modelagem funcional**.

Modelo Ambiental

O modelo ambiental é composto de quatro componentes:

1. Declaração de Objetivos;
2. Diagrama de Contexto;
3. Lista de Eventos;
4. Dicionário de Dados Preliminar (opcional).

Declaração de objetivos

Consiste em uma breve e concisa declaração dos objetivos do sistema. É dirigida para alta gerência, gerência usuária ou outras pessoas não diretamente envolvidas no desenvolvimento do sistema. Pode ter uma, duas ou várias

sentenças mas não deve ultrapassar um parágrafo e também não deve pretender dar uma descrição detalhada do sistema.

Exemplo: “O Objetivo do sistema de processamento de livros ABC é manusear todos os detalhes de pedidos de compra de livros dos clientes, bem como a remessa, faturamento e cobrança de clientes em atraso. Informações sobre pedidos de livros devem ficar disponíveis para outros sistemas tais como: Marketing, Vendas e Contabilidade”.

Diagrama de Contexto

Apresenta uma visão geral das características importantes do sistema:

- As Pessoas, organizações ou sistemas com os quais o sistema se comunica (Entidades externas);
- Os dados que o sistema recebe do mundo externo e que devem ser processados;
- Os dados produzidos pelo sistema e enviados ao mundo externo;
- A Fronteira entre o sistema e o resto do mundo.

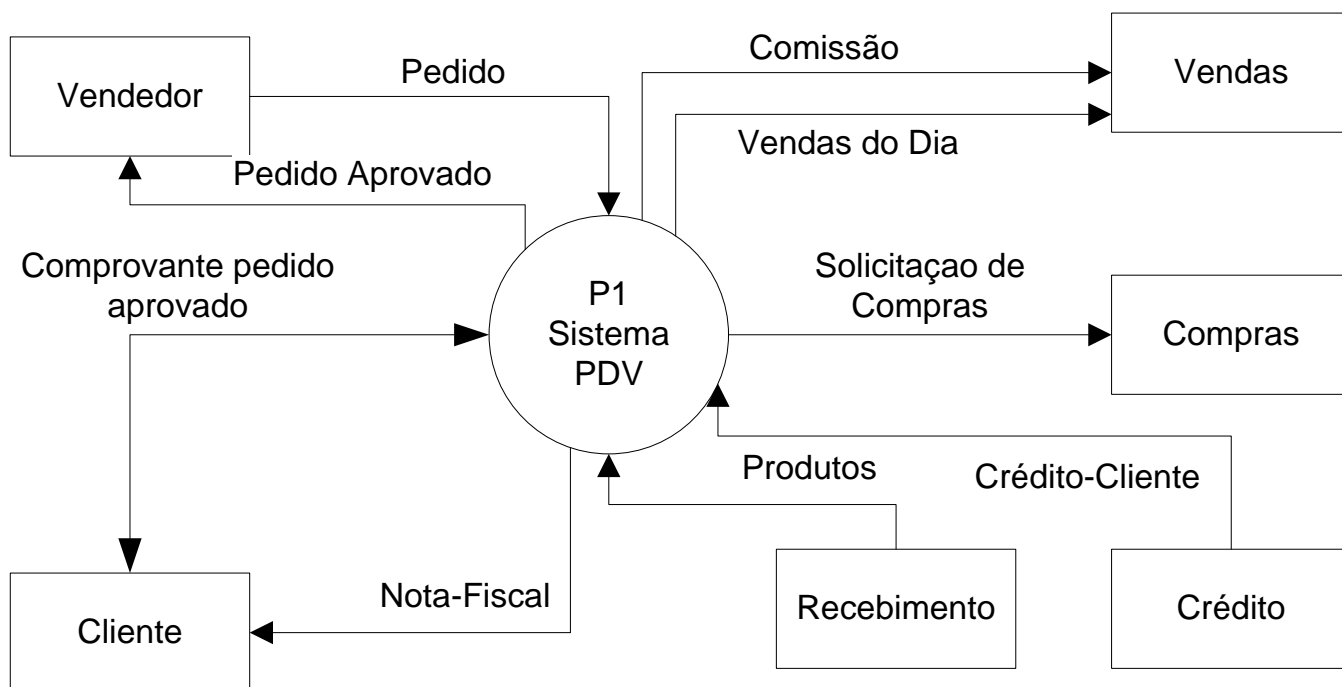


Figura 20 DFD do sistema de PDV

Lista de Eventos

É uma relação dos estímulos que ocorrendo no mundo exterior implicam em alguma resposta do sistema.

Nós analistas de sistemas, construímos sistemas para as mais diversas áreas, que tem como ponto comum a interatividade, ou seja, esses sistemas agem sobre coisas que estão normalmente fora de seu controle e essas mesmas coisas agem sobre o sistema. Senão, vejamos o seguinte exemplo para entender melhor:

Os usuários de ônibus intermunicipal/interestadual que interagem com o sistema de venda de passagens estão fora do controle dos vendedores de passagem. Um usuário pode decidir não utilizar o ônibus num determinado dia, ou utilizar sem comprar sem comprar uma passagem naquele dia. O Vendedor das passagens não tem poder para obrigar o usuário a comprar uma passagem, ou escolher uma das opções anteriores. Mas se um usuário resolve comprar uma passagem, ele terá de

interagir com o vendedor de passagens, fornecendo dinheiro. Como resposta o vendedor age de uma certa forma que afeta o passageiro: ele entrega as passagens e talvez o troco em dinheiro.

Observando esse exemplo da vida real, podemos entender que o usuário do ônibus é parte de um determinado ambiente, age sobre o sistema, digamos de venda de passagens e o sistema reage sobre ele. A figura a seguir representa esse sistema e suas interações:

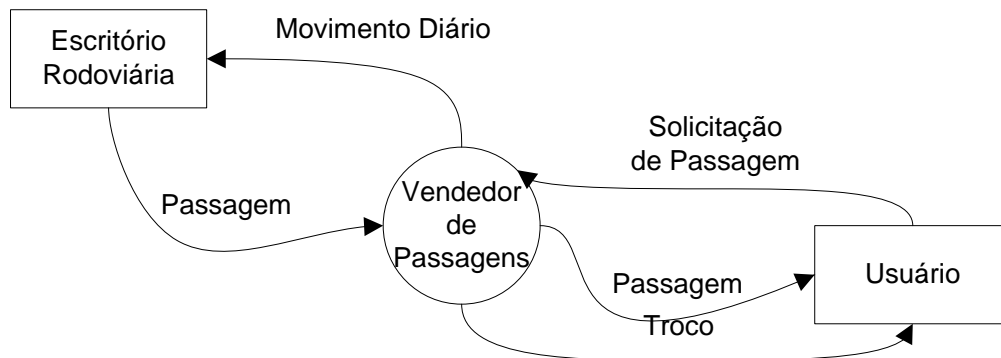


Figura 21 Sistema de venda de passagens de ônibus

Na imagem abaixo, segundo McMenamin, designamos de evento e resposta as interações entre o sistema e o ambiente.

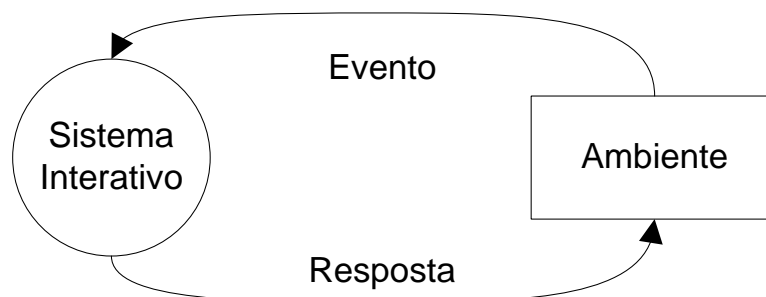


Figura 22 Interações entre o sistema

Vale destacar aqui que evento é alguma mudança que ocorre no ambiente, caracterizada pela disponibilização ou envio de uma informação, e uma resposta é um conjunto de ações executadas e um conjunto de dados apresentados pelo sistema sempre que ocorre um determinado evento. Por exemplo, o vendedor de passagens fornece-as e, possivelmente, o troco como resposta ao evento “passageiro solicita passagem”.

Observe que neste caso temos ações e informações tanto no evento como na resposta. O importante de entendermos o conceito aqui mostrado é que um sistema de informações é interativo, logo ele é composto de eventos e respostas.

Podemos dizer que quando a resposta de um sistema a um evento foi determinada antes da ocorrência do evento, então esta é uma resposta planejada. Um sistema de informação computacional, tem como característica ser um sistema de respostas planejadas e não ad-hoc. Logo na abordagem essencial, o fundamental no levantamento de informações para a construção de um sistema é o descobrimento dos seus eventos e respostas.

Um sistema de respostas planejadas responde a um conjunto de eventos predefinidos, com formato de resposta também predefinido; logo, ao modelarmos os requisitos de um sistema, podemos derivar e analisar os fatos e ações que existem em um ambiente a ser sistêmico.

Vamos utilizar como exemplo a nossa casa. O sistema de estoques de alimentos e materiais de limpeza de nossa casa possui quais eventos ?. Por exemplo: **Verificação de quantidades em estoque;**

O sistema interage com o ambiente que é a família. O sistema é delimitado pelos armários nos quais colocamos produtos de alimentação e os materiais de limpeza, não sendo considerados pelo sistema depósitos intermediários, tais como: baldes, saleiros, açucareiros, etc.

Um evento provoca uma resposta planejada de solicitação de compra, ou melhor, como é comum, uma anotação na lista de compras semanal ou mensal basicamente, conforme demonstrado na imagem abaixo:

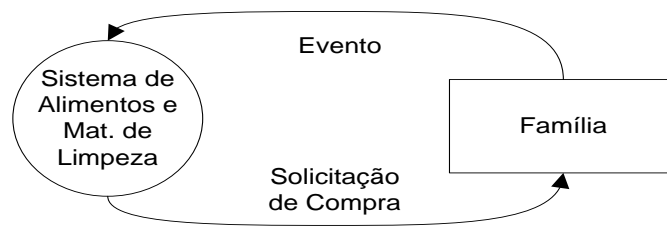


Figura 23 Estrutura de eventos do sistema e compras familiar

Sempre o mais difícil é entender que evento provoca a resposta, pois o usuário apresenta-nos normalmente em levantamentos as respostas que ele necessita e conhece, sendo muito difícil que ele nos apresente o evento de forma destacada.

Continuando com o exemplo, que evento provoca em nossa casa, que se faça uma anotação na lista de compras semanal, ou mensal ? Poderíamos chamar de **verificação de produtos.**

Se pensarmos mais um pouco, podemos verificar que existem alguns itens de compra que são necessários (em alguns momentos) imediatamente, porém a lista de compras do mês não leva em consideração esse fato, ela é feita baseada no consumo, no saldo em estoque e necessidade que a família tem para um mês de consumo. Logo existe mais de um evento nesse sistema, pois dois tipos de respostas são apresentadas:

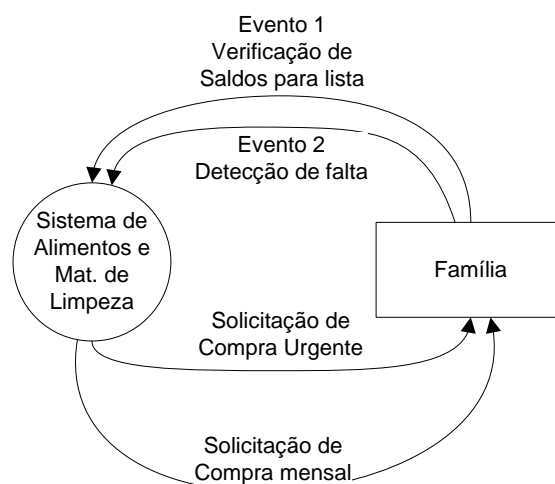


Figura 24 Detecção de vários eventos

Os eventos aos quais um sistema responde são de dois tipos: **eventos externos**, que são iniciados por fatos ou acontecimentos do ambiente, e os **eventos temporais** que são iniciados pela passagem de determinado tempo.

O primeiro evento do exemplo, caracteriza-se por um evento externo ao sistema, e o segundo como temporal, pois ocorre devida à passagem do tempo, normalmente no final do mês.

Ainda observando o exemplo, poderíamos criar um outro evento que seria: caso eu possua determinado item a ser utilizado porém não possuo mais nenhum item na dispensa, posso disparar uma compra para renovar o estoque da dispensa imediatamente.

Que evento seria esse ? É um terceiro evento ? Ou é um evento que produz duas respostas ?

Neste caso existem dois eventos somente, pois somente dois fatos externos ao sistema caracterizam-se como eventos e obtêm resposta.

É comum, ao modelarmos um sistema, apresentarem-se fatos externos ao ambiente do sistema inclusive. Torna-se necessária a conciliação entre o que é do escopo do sistema e o que não é.

Um evento pode ser definido informalmente como um acontecimento do mundo exterior que requer do sistema uma resposta.

Um **estímulo**: É um ativador de uma função. É a forma como o evento age sobre o sistema. É a consequência do fato de ter ocorrido um evento externo. É a chegada de um estímulo que indica que um evento ocorreu e isto faz com que o sistema então ative uma função predeterminada para produzir a resposta esperada.

Uma **resposta**: É o resultado gerado pelo sistema devido à ocorrência de um evento. Uma resposta é sempre o resultado da execução de alguma função interna no sistema como consequência do reconhecimento pelo sistema de que um evento ocorreu. Pode ser:

- Um fluxo de dados saindo do sistema para o ambiente externo;
- Uma mudança de estado de algum dado (o que equivale à inclusão, exclusão ou modificação de algum registro de um arquivo);
- Um fluxo de controle saindo de uma função para ativar outra função.

Respostas de um Evento

Todo evento somente é um evento se ele fornecer respostas ao ambiente externo ao sistema. Observando o primeiro diagrama desenhado no exemplo anterior, observa-se que o estímulo oriundo do ambiente externo ao sistema foi uma solicitação de retirada de um produto, que provocou uma resposta do sistema.

O importante é que não devemos procurar eventos dentro da essência de um sistema e sim no ambiente externo a este. O que estimula um evento é uma informação que entra no sistema, no caso do exemplo, um produto solicitado.

A existência de duas respostas caracteriza bem um evento, pois como ele é composto de vários processos, logo pode fornecer mais de uma resposta, inclusive resultados de decisão.

Evento	Estímulo	Ação	Resposta
Retirada de Produto	Dados do Produto	Verificação de Saldo Atualização de Saldo	Produto Fornecido Pedido de Compra urgente

Atividades Essenciais

Um sistema de respostas planejadas interage com o mundo à sua volta, ou seja, seu ambiente, pela identificação e reconhecimentos de determinados estímulos e pela execução de um conjunto de instruções preparadas em resposta.

Este conjunto de instruções é o que chamamos de atividade essencial, e responde a um único evento que pode ser externo ou um evento temporal.

Logo, o princípio é o evento, e depois as atividades que constituem este evento, ou melhor, que produzem resposta a este evento. Devemos ter em mente sempre que, um evento sem resposta não é um evento.

Dados e Memória Essencial

Nós humanos executamos diversas atividades mentalmente, ou com auxílio de papel e caneta, e utilizamos a nossa memória para fazer cálculos e decidir quantidades a comprar. Entretanto para que um sistema forneça uma resposta adequada, as atividades de um evento, deverá ter a sua disposição várias informações.

No exemplo da dispensa, o nome do produto, a quantidade em estoque, a quantidade de consumo médio no mês, etc.

O ideal no projeto de sistema, é modelarmos em primeiro lugar todos os eventos de um sistema, porém sem detalharmos seus processos ou atividades em nível de detalhe, sempre obteremos a visão de quais são os eventos e principais atividades sempre em visão macro.

Após o levantamento de todos os eventos, podemos então, já com a localização de alguns objetos de dados, partir para o processo de modelagem de dados, então esmiuçando o máximo e com alto nível de abstração para retratarmos uma realidade de um ambiente.

Imaginamos a seguinte situação: Vamos até o mercado e efetuamos as compras necessárias para a nossa dispensa. O supermercado nos fornece uma nota fiscal, ou cupom de caixa em que estão discriminados todos os produtos comprados e suas quantidades. Isto irá criar um evento, que consiste em movimentar os estoques, com a entrada de produtos que adquirimos na compra.

Qual é a ação deste evento, ou quais são as suas atividades ?

Observe o diagrama de eventos e depois para um DFD com resposta:

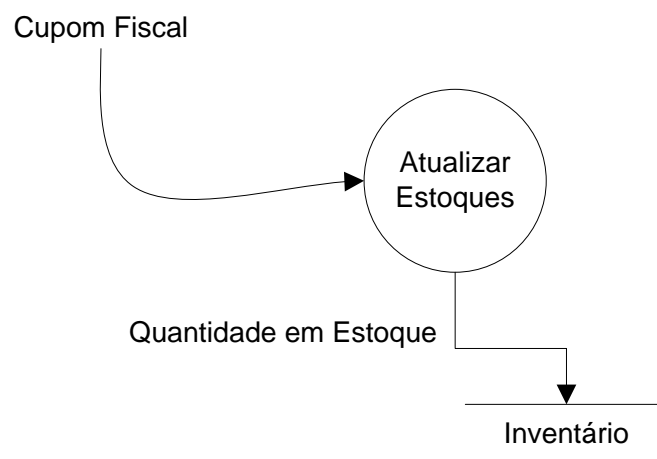


Figura 25 DFD (Incompleto)

Então, algo ainda está faltando neste caso.

Mentalmente quando colocamos os produtos comprados no armário, realizamos uma contagem de cada um, obtendo o número final em estoque. Essa lista mental de quantidades atualizadas em estoque também deve ser a resposta de nosso evento, ficando desta forma caracterizada uma resposta do sistema ao ambiente:

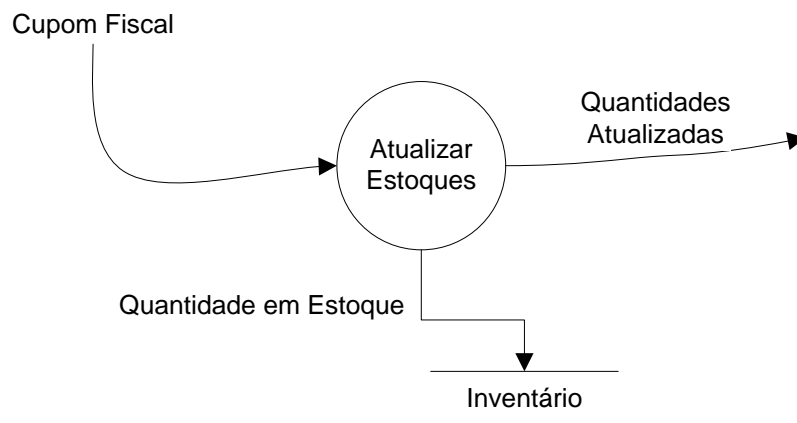


Figura 26 DFD Completo (com resposta)

Abaixo está discriminada uma lista de eventos para que possamos abstrair e visualizar principalmente as diferenças entre o ambiente externo a um sistema e as funções de um sistema, assim como identificar o conceito de resposta a um evento.

Nr. do Evento	Evento	Estímulo	Ação	Resposta
1	Cliente Faz pedido	Pedido	Registrar pedido	Pedido registrado
2	Cliente cancela pedido	Pedido de cancelamento	Cancelar pedido	Pedido cancelado
3	Cliente envia pagamento	Cheque	Emitir recibo de pagamento	Recibo de Pagamento
4	É hora de verificar pedidos em atraso	Solicitação de Relatório de Pedidos em atraso	Verificar e emitir relatório	Relatório de pedidos em atraso
5	Verificação de suprimento	Solicitação de verificação de nível de suprimento	Verificar nível de suprimento Emitir solicitação de compra	Solicitação de compra

Considerações Finais

Não existe uma sequência pré-definida para produção do Diagrama de Contexto ou a Lista de Eventos. Pode-se começar por qualquer um dos dois, bastando no final verificar se estão consistentes um com outro.

Quando for difícil produzir o DFD e a Lista de eventos, pode-se começar por um Diagrama Entidade Relacionamento (DER). A partir deste pode-se buscar eventos candidatos identificando atividades ou operações que causem a criação ou exclusão de instâncias de entidades.

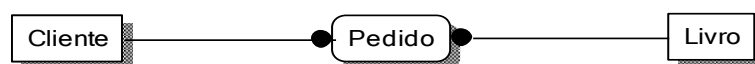


Figura 27 Exemplo de DER

Modelo Comportamental

Define o comportamento interno que o sistema deve ter para se relacionar com o ambiente externo. Descreve de que maneira o sistema enquanto conjunto de elementos, inter-relacionados reage internamente como um todo organizado aos estímulos do exterior.

É expresso por:

- Diagrama de Fluxo de Dados;
- Mini-Especificação;
- Diagrama de Transição de Estado;
- Diagrama Entidade Relacionamento;

Modelo de Implementação

Tem por finalidade produzir um modelo para a implementação do sistema, a partir de suas especificações conceituais e dos requisitos para ele estabelecidos. Envolve questões relativas à utilização do sistema pelo usuário.

As atividades necessárias para a construção do modelo de implementação são as seguintes:

- Construir o modelo lógico de dados;
- Determinar as características de processamento de cada função ou processo;
- Especificar interface homem-máquina.

Prototipação

Como já vimos anteriormente, a prototipação é muito útil quando se está tentando mostrar aos usuários como **poderá** ser o resultado final do projeto de software. Iremos aqui apenas complementar este assunto.

Para colocar essa técnica em uma perspectiva correta, dentro dos processos de desenvolvimento de software, é importante distinguir entre **protótipo descartável** e **protótipo evolucionário**.

Protótipo Descartável

É tipicamente construído durante a engenharia de requisitos, com a única finalidade de demonstrar aos usuários chaves o que o analista captou quanto aos requisitos do produto, ou parte deles. Na construção de um protótipo descartável, o fundamental é a rapidez de construção; um protótipo descartável deve ser produzido em poucas horas, ou no máximo em poucos dias.

O protótipo descartável tem por objetivo explorar aspectos críticos dos requisitos de um produto, implementando de forma bastante rápida um pequeno subconjunto de funcionalidades desse. Ele é indicado para se estudar por exemplo:

- Alternativas de interface de usuário;
- Problemas de comunicação com outros produtos;
- Viabilidade de atendimento aos requisitos de desempenho.

Em projetos maiores, o protótipo descartável pode também ser empregado em problemas de desempenho e implementação, sempre que seja possível decidir uma questão importante através de um pequeno experimento.

Como exemplo de áreas tratáveis por protótipos e os respectivos ambientes de prototipagem citam-se os seguintes:

- **Interface com o usuário:** protótipo visual, escrito seja no ambiente definitivo, seja em um ambiente de programação rápida, ou com ferramentas de desenho;
- **Relatórios textuais:** processador de textos ou ferramentas de geração de relatórios;
- **Relatórios gráficos:** ferramenta de desenho ou ambiente de programação rápida com biblioteca gráfica;
- **Organização e desempenho de banco de dados:** ferramenta de desenvolvimento rápido integrada ao sistema de gerência de bancos de dados que se pretende usar;
- **Cálculos complexos:** planilha ou ferramenta matemática;
- **Partes de tempo de resposta crítico:** em sistemas de tempo real – pequeno programa de teste no ambiente alvo;
- **Tecnologia no limite do estado-da-arte:** Pequeno programa de teste no ambiente que for mais adequado;

O material de código dos protótipos descartáveis não deve ser reaproveitado. Antes que se decida usar protótipos descartáveis, o cliente e os usuários devem ser plenamente esclarecidos quanto aos objetivos e comprometer-se com seu descarte.

Protótipo Evolucionário

Conterá um subconjunto dos requisitos do produto final, mas nenhum dos padrões de engenharia de software é afrouxado em sua construção. O objetivo da partição da construção de um produto em liberações é permitira implementação incremental e iterativa de um aplicativo mais complexo.

Quem quiser usar protótipos evolutivos deve planejá-los como tais desde o começo, pode ser o caso, por exemplo, em relação as interfaces com o usuário.

Protótipos – Considerações finais

Quando se trata de protótipo para definição de interface com o usuário, é importante que sigamos uma lista de pré-requisitos necessários a uma boa construção:

1. Divida a tela em regiões;
2. Agrupe os dados logicamente;
3. Alinhe os dados verticalmente;
4. Explícite o tamanho dos campos de entrada;
5. Evite telas poluídas;
6. Evite exagero no uso de recursos visuais;
7. Exiba mensagens elucidativas e objetivas;
8. Peça confirmação para funções críticas (como exclusão);
9. Proporcione Feedback;
10. Ordene as listas exibidas;
11. Divida strings-longos em sub-strings;
12. Não exija preenchimento de caracteres não significativas;
13. Utilize um vocabulário consistente;
14. Considere a seqüência dos dados do documento associado;
15. Seja consistente, mantenha padrões.

Parte III - Metodologias e Técnicas

Nesta parte da disciplina trataremos de algumas questões importantes:

- Como as pessoas, processos e problemas precisam ser geridos durante um projeto de software ?
- Como as métricas de software podem ser usadas para gerir projeto de software e processo de software ?
- Como vamos estimar esforço, custo e duração de projeto ?
- Que técnicas podem ser usadas para avaliar formalmente os riscos que podem causar impacto no sucesso do projeto ?
- Como um gerente de projeto de software seleciona um conjunto de tarefas de trabalho de engenharia de software ?
- Como é criado e para que serve um cronograma de projeto ?
- O que é gestão da qualidade ?
- Por que revisões técnicas formais são tão importantes ?
- Como é gerida a modificação durante o desenvolvimento de software de computador e depois da entrega ao cliente ?

Gerenciamento de Projetos de Software

A Gestão de projetos envolve atividades de planejamento, monitoração e controle de pessoal, processo e eventos que ocorrem à medida que o software evolui de um conceito preliminar para uma implementação operacional.

Um engenheiro de software gerencia suas atividades do dia-a-dia, planejando, monitorando e controlando tarefas técnicas. Gerentes de projeto planejam, monitoram e controlam o trabalho de uma equipe de engenheiros de software. Gerentes seniores coordenam a comunicação entre o negócio e a equipe técnica.

Por que é importante ? A construção de software de computador é complexa, particularmente quando existem várias pessoas envolvidas, trabalhando durante um período relativamente longo.

Primeiramente precisamos entender os 4 P's:

Pessoal	•Pessoas precisam ser organizadas para realizar o trabalho de software efetivamente.
Produto	•A comunicação com o cliente precisa ocorrer para que o escopo e os requisitos do projeto sejam entendidos.
Processo	•Um processo precisa ser selecionado a fim de se adequar ao pessoal e ao produto.
Projeto	•O projeto precisa ser planejado, estimando o esforço e o tempo para executar as tarefas de trabalho

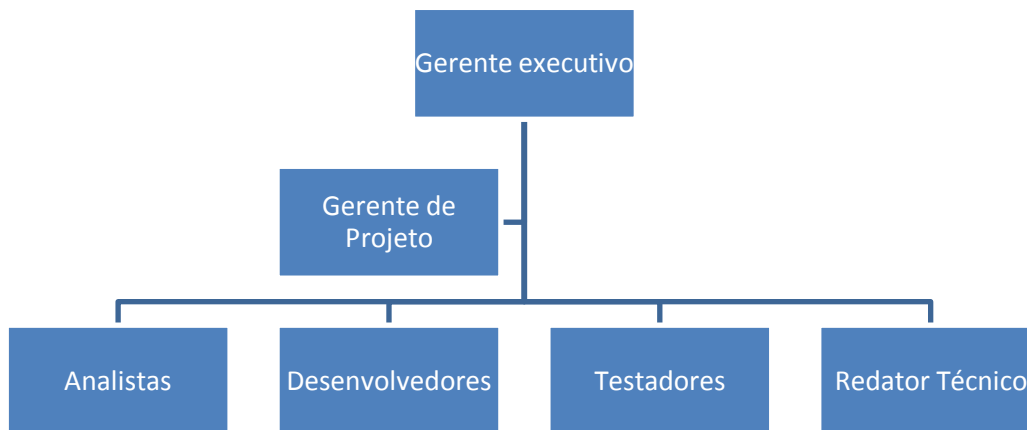
O Pessoal

Pessoal motivado e altamente qualificado para o projeto é algo que tem sido discutido desde os anos 60. De fato, esta atividade é algo tão importante que o SEI (Software Engineering Institute) desenvolveu um *modelo de maturidade de capacidade de gestão de pessoal*.

Melhorar a capacidade de organizações de software desenvolverem aplicações cada vez mais complexas, ajudando-as a atrair, desenvolver, motivar, dispor e reter o talento necessário para melhorar sua capacidade de desenvolvimento de software.

Organizações que atingem uma maior maturidade na área de gestão de pessoal têm grandes chances de implementar práticas efetivas de engenharia de software. O PM-CMM é parte integrante do modelo de maturidade da capacidade de software (CMMI).

O organograma abaixo mostra como estão dispostos os cargos/funções em um projeto de software comum:



O Produto

Antes do planejamento efetivo do projeto de software, questões como os objetivos e o escopo do produto devem ser estabelecidas, soluções alternativas devem ser consideradas e as restrições técnicas e gerenciais devem ser identificadas.

Sem estas informações não seria possível delimitar custo, avaliações efetivas do risco, relações realísticas de tarefas de projeto ou cronogramas gerenciáveis o suficiente para garantir o sucesso do projeto. Neste ponto, o desenvolvedor do software e o cliente, devem se reunir para discutir o escopo e os objetivos do produto.

Como vimos anteriormente, a delimitação do escopo e objetivos do projeto devem ser definidas no início do projeto, durante as fases iniciais onde se faz um processo de análise efetivo incluindo aí a etapa de levantamento de requisitos.

A primeira atividade da gestão de um projeto é a determinação do escopo do software. O escopo é definido pela resposta às seguintes questões:

1. **Contexto:** Como o software a ser construído se encaixa no contexto de um sistema maior (mais genérico), do produto ou do negócio, que restrições são impostas como resultado do contexto ?
2. **Objetivos da informação:** Que objetos de dados visíveis para o cliente são produzidos como saída pelo software ? Que objetos de dados são necessários para entrada ?
3. **Função e desempenho:** Que função o software desempenha para transformar os dados de entrada em saídas ? Existem características especiais de desempenho a serem tratadas ?

Consideremos por exemplo, um projeto que construirá um editor de texto, entre as características particulares deste editor estão: entrada, tanto por voz contínua como pelo teclado, características sofisticadas de edição automática de textos, capacidade de disposição de página, preparação automática de índices e sumário, e outras. Primeiro precisaremos estabelecer uma descrição de escopo que irá delimitar tais características. Por exemplo, a entrada por voz exige que o software seja treinado ? Especificamente qual a capacidade oferecida pela característica edição de texto ?

O Processo

Um processo de software fornece a estrutura necessária a partir da qual pode ser definido um plano completo para o desenvolvimento de software. Algumas atividades são aplicáveis a todos os projetos de software, independente do seu tamanho.

O problema aqui é qual o processo a ser utilizado para desenvolver tal projeto. Neste caso devemos estabelecer qual o melhor processo levando-se em conta:

1. O cliente que solicitou o produto e o pessoal que irá executar o trabalho;
2. As características do produto propriamente dito;
3. O ambiente do projeto no qual a equipe de software irá trabalhar.

Definido o modelo do processo, a equipe então define um plano preliminar de projeto, com base no conjunto de atividades comuns padrões do processo.

O planejamento do projeto tem inicio com a fusão do produto e do processo, cada função a ser trabalhada pela equipe, deve passar pelo conjunto de atividades que estão definidas neste processo. Consideremos nós que o seguinte conjunto de atividades foi adotado para a condução do processo: Comunicações, Planejamento, Modelagem, Construção e Implantação.

Todos os membros da equipe trabalham segundo os padrões do processo adotado, em resumo é criada uma matriz, conforme mostrado abaixo. Cada função principal do produto é listada à esquerda, atividades padrão são listadas na linha superior. As tarefas de engenharia de software são colocadas na linha seguinte. O trabalho do gerente do projeto aqui é estimar a necessidade de recursos para cada célula da matriz.

Atividades comuns da estrutura do processo	Comunicação				Planejamento				Modelagem				Construção				Implantação			
Tarefas de Engenharia de Software																				
Funções de Produto																				
Texto de Entrada																				
Edição e Formatação																				
Edição automática de textos																				
Capacidade de layout de página																				
Preparação índice e sumário																				
Gestão de arquivo																				
Produção de documento																				

Tabela 2 Fusão do problema e do processo

Uma equipe de software deve estar apta a adotar os mais variados tipos de processo. Um projeto relativamente pequeno (como no exemplo), pode ser mais bem realizado usando a abordagem seqüencial linear. Se forem impostas restrições severas de tempo e o problema puder ser dividido em unidades menores, o modelo RAD é provavelmente a opção correta. Se o prazo de entrega é tão apertado que praticamente impossibilite entregar toda a funcionalidade, um processo incremental pode ser melhor.

Quando o modelo de processo for selecionado, deve-se vincular o projeto aos passos definidos para o processo, e adicionar outros passos que possam ser necessários para o projeto em questão, ou seja, devemos sempre que necessário adaptar o processo padrão à realidade de cada projeto.

Por exemplo, um projeto relativamente pequeno poderia ter o seguinte check-list na tarefa comunicação:

1. Desenvolver uma lista de pontos a esclarecer;
2. Reunir-se com o cliente para discutir os pontos a esclarecer;
3. Desenvolver conjuntamente uma declaração de escopo;
4. Rever a declaração de escopo com todos os interessados;
5. Modificar a declaração de escopo na medida do necessário.

Esses eventos podem ocorrer em um espaço curto de tempo (até 48 horas), pois eles representam uma decomposição de processo, que é apropriada para um projeto pequeno e relativamente simples.

Consideremos agora, um projeto maior, mais complexo, com mais impacto sobre o negócio, ele poderia ter o seguinte check-list para a mesma atividade:

1. Rever a solicitação do cliente;
2. Planejar e marque um encontro facilitado e formal com o cliente;
3. Fazer pesquisa para especificar a solução proposta e as abordagens existentes;
4. Preparar um “documento de trabalho” e uma agenda para a reunião formal;
5. Conduzir a reunião;
6. Desenvolver, em conjunto, miniespecificações que reflitam as características do software quanto a dados, função e comportamento. Alternativamente, desenvolva casos de uso que descrevam o software do ponto de vista do usuário;
7. Rever cada miniespecificação ou caso de uso quanto à correção, consistência e ausência de ambigüidade;
8. Montar as miniespecificações em um documento de definição do escopo;
9. Rever o documento de definição de escopo ou coleção de casos de uso com todos os interessados;
10. Modificar o documento de definição do escopo ou casos de uso na medida do necessário.

O Projeto

A única forma de se conseguir gerir a complexidade no desenvolvimento de sistemas é através de projetos bem elaborados e ainda assim, sofremos. Para gerir com sucesso um projeto de software, precisamos ter em mente o que dar errado.

1. Não entende-se claramente as necessidades do cliente;
2. O escopo do produto está mal definido;
3. As modificações são mal gerenciadas;
4. A tecnologia escolhida sofre modificações;
5. As necessidades do negócio modificam-se (ou estão mal definidas);
6. Os prazos são irreais;
7. Os usuários são resistentes;

8. O patrocínio é perdido (ou nunca houve);
9. Não existe pessoal capacitado o suficiente;
10. Gerentes e profissionais evitam as melhores práticas e as lições adquiridas em projetos anteriores.

O que um gerente de projeto pode fazer para agir e evitar os problemas mencionados ?

1. *Comece com o pé direito*: Deve-se trabalhar (e muito) para entender o problema que deve ser resolvido, em seguida estabeleça objetivos e expectativas realistas para todos os que serão envolvidos no projeto;
2. *Mantenha a energia de momento*: Muitos projetos partem de um bom princípio depois lentamente se desintegram. Práticas positivas devem ser adotadas, como minimizar a rotatividade de pessoal ao mínimo absoluto e a equipe devem enfatizar a qualidade em todas as tarefas que executa;
3. *Acompanhe o progresso*: Um projeto de software é acompanhado a medida que produtos do trabalho (modelos, código-fonte, conjuntos de casos de teste) são produzidos e aprovados (usando revisões técnicas formais) como parte da garantia da qualidade;
4. *Tome decisões adequadas*: A essência de uma boa decisão aqui é “manter a coisa simples”. Sempre que possível utilize software comercial de prateleira ou componentes de software existentes, trocar interfaces sob medida por abordagens padrão;
5. *Faça uma análise a posteriori*: Extraia de cada projeto lições aprendidas. Avaliar os cronogramas planejados e cumpridos.

Em um trabalho sobre processo e projetos de software, Barry Boehm, sugere uma abordagem com foco nos objetivos do projeto, marcos e cronogramas, responsabilidades, abordagens gerenciais e técnicas, e recursos necessários. Ele denomina o princípio **W5HH**:

Por que (Why)	• <i>Por que o sistema será desenvolvido ?</i> A razão comercial, justifica o gasto de pessoal, tempo e dinheiro no desenvolvimento do produto ?
O que (What)	• <i>O que vai ser feito ?</i> Define o conjunto de tarefas que será necessário para o projeto
Quando (When)	• <i>Quando vai ser feito ?</i> Ajuda a estabelecer um cronograma de projeto, a partir da identificação do quando as tarefas devem ser realizadas e os marcos alcançados.
Quem (Who)	• <i>Quem é o responsável por uma determinada função ?</i> O papel e a responsabilidade de cada membro da equipe devem ser estabelecidos.
Onde (Where)	• <i>Onde estão localizados na organização ?</i> Nem sempre os membros da equipe desempenham todos os papéis necessários. Por vezes, o cliente, usuários e outros também tem responsabilidades.
Como (How)	• <i>Como o trabalho será conduzido técnica e gerencialmente ?</i> Com o escopo definido, deve-se definir uma estratégia gerencial e técnica para o projeto
Quanto (How Much)	• <i>Quanto é necessário de cada recurso ?</i> Utiliza-se estimativas (que são vistas mais adiante), baseadas nas respostas as questões anteriores

Resumo

Gestão de projeto de software é uma atividade genérica dentro da engenharia de software. Começa antes de qualquer atividade técnica ser iniciada e continua ao longo da definição do desenvolvimento e do apoio do software de computador.

Devemos ter em mente, sempre que um projeto de software precisa ser gerenciado, qualitativa e quantitativamente, alguns dos pontos abordados aqui são vitais na condução de um bom projeto de software. Se faz mais do que necessário sempre manter uma clara comunicação entre todos os envolvidos no projeto, iniciando pelos clientes.

Plano de Desenvolvimento de Software

Um plano de desenvolvimento de software (PDSw) deve fornecer a base para o acompanhamento e controle do projeto, até a colocação de seus resultados em operação. Este padrão define os elementos que normalmente devem estar presentes em um PDSw. Conforme a natureza do projeto, alguns elementos podem ser omitidos, desde que a gerência executiva o aprove.

Normalmente as datas indicadas no PDSw podem ser expressas em marcos do projeto, como os finais das iterações, ou em datas de calendário. O primeiro método facilita a atualização do plano, enquanto o segundo deve ser usado em seções de maior interesse para o cliente.

A página de título do PDSw deve incluir os seguintes elementos:

- Nome do documento;
- Identificação do projeto para o qual a documentação foi produzida;
- Nomes dos autores e das organizações que produziram o documento;
- Número de revisão do documento;
- Data de aprovação;
- Assinaturas de aprovação;
- Lista dos números de revisão e datas de aprovação das revisões anteriores.

Visão Geral do Projeto

Esta subseção é um sumário de informações básicas sobre o projeto, destinada a orientar os responsáveis pela decisão de continuá-lo ou não. Tipicamente esta sub-seção deve incluir os seguintes dados:

- Objetivos deste documento, que caracterizam seu escopo e público:
Descrever o plano de desenvolvimento do produto XSoft, definindo os recursos necessários para o desenvolvimento deste, assim como uma previsão dos prazos, custos e riscos associados a esse projeto.
Público-Alvo: cliente e desenvolvedores do projeto XSoft;
- Objetivos do projeto, que devem ser compatíveis com a missão definida na especificação de requisitos de software:
Apoio informatizado ao controle de vendas, compras e de estoque da papelaria Plus Paper & Cia Ltda.
- Produto a ser entregue e suas partes principais. Por definição, cada projeto entrega um único produto, que pode ser composto de várias partes: componentes cliente e servidor, variantes interativas e em lote, documentos, material de treinamento, etc:
XSoft (Componente único) – código objeto e documentação para usuários;
- Marcos, ou seja, os principais resultados a serem gerados no projeto, e respectivas datas previstas para aprovação. Esses resultados devem ser escolhidos de forma a dar ao cliente uma visão sintética (porém completa) do progresso alcançado pelo projeto:

Número de ordem	Resultado a ser produzido	Data prevista para entrega
-----------------	---------------------------	----------------------------

1	Descrição do desenho do projeto (provisório)	22/04/XXXX
2	Documentação dos testes do software (provisório)	22/04/XXXX
3	Liberação 1	18/05/XXXX
4	Liberação 2	13/06/XXXX
5	Manual do usuário do software (provisório)	23/06/XXXX
6	Descrição do desenho do software (completo)	23/06/XXXX
7	Documentação dos testes do software (provisório)	23/06/XXXX
8	Manual do usuário do software (completo)	08/08/XXXX
9	Descrição dos testes do software (completa)	08/08/XXXX

- Recursos que devem ser fornecidos pelo cliente, em visão sintética. Deve-se indicar o tipo do recurso (pessoal, equipamentos, ferramentas, programas a incluir, etc), respectivas quantidades, quando aplicável e a data limite para utilização:

Número de ordem	Recurso	Data Limite
1	Sistema de Gerência de Banco de Dados com o qual o XSoft irá funcionar, instalado e configurado.	25/03/XXXX
2	Base de dados do XSoft, contendo todos os dados das mercadorias em estoque e dos fornecedores	13/06/XXXX
3	Usuários responsáveis por testes beta	23/06/XXXX
4	Equipamentos para testes beta	23/06/XXXX

- Programação das principais atividades. Devem ser indicadas as atividades a serem executadas e suas respectivas datas para início e conclusão:

Número de ordem	Nome da Atividade	Data Inicial Prevista	Data final Prevista
1	Desenho implementável	25/03/XXXX	22/04/XXXX
2	Liberação 1	22/04/XXXX	18/05/XXXX
3	Liberação 2	18/05/XXXX	13/06/XXXX
4	Testes Alfa	13/06/XXXX	23/06/XXXX
5	Testes Beta	23/06/XXXX	10/07/XXXX
6	Operação Piloto	10/07/XXXX	08/08/XXXX

- Principais itens do orçamento (também de forma sintética), deve indicar uma estimativa de custo para o cliente, normalmente são indicadas aqui atividades ou componentes que possam ser negociados separadamente:

Número de ordem	Descrição	Quantidade (pessoas-mês)
1	Desenvolvimento do Software	10.000
2	Participação dos usuários	5

- Referência a projetos correlados, indicando sigla e nome:

Número de ordem	Sigla do Projeto	Nome do Projeto
-----------------	------------------	-----------------

1	SSIM	Sistema para informatização do negócio (anterior)
---	------	---

- Referência aos documentos oficiais de especificação de requisitos desse projeto, que servem de base para esse plano indicando-se o respectivo nome e revisão:

Número de ordem	Nome do Documento	Revisão do Documento
1	Especificação dos Requisitos do Software XSoft	

Modelo do Processo

Descreve as atividades adotadas do processo para o projeto. Caso não haja especificações entre um projeto e outro. Esse modelo é idêntico para todos os projetos, exceto pelo número de Liberações.

Fase	Iteração	Sigla	Descrição
Concepção (CN)	Ativação (Início)	AT	Levantamento e análise das necessidades dos usuários e conceitos da aplicação, em nível de detalhe suficiente para justificar a especificação de um produto de software.
Elaboração (EL)	Levantamento de Requisitos	LR	Levantamento das funções, interfaces e requisitos não funcionais desejados para o produto.
	Análise dos Requisitos	AR	Modelagem conceitual dos elementos relevantes do domínio do problema e uso desse modelo para validação dos requisitos e planejamento detalhado da fase de construção.
	Desenho Implementável	DI	Definição interna e externa dos componentes de um produto de software, em nível suficiente para decidir as principais questões de arquitetura e tecnologia e para permitir o planejamento detalhado das atividades de implementação.
	Liberação 1	L1	Implementação de um subconjunto de funções do produto que será avaliado pelos usuários.
	Liberação 2	L2	Idem.
	Testes Alfa	TA	Realização dos testes de aceitação, no ambiente dos desenvolvedores, juntamente com elaboração da documentação de usuário e possíveis planos de transição.
Transição (TR)	Testes Beta	TB	Realização dos testes de aceitação, no ambiente dos usuários.
	Operação Piloto	OP	Operação experimental do produto em instalação piloto do cliente, com a resolução de eventuais problemas através de processo de manutenção.

Material necessário para realização de atividade

A tabela abaixo será utilizada para a realização de atividade em sala de aula. Na próxima aula, iremos verificar como ocorre a cronogramação de um projeto de software. Precisamos antes de qualquer coisa, determinar o que será feito, para em seguida determinar quando e depois quem.

PJ000-Sistema	200,35 dias	0%	13/10/2004	22/07/2005
1-Ante Projeto	29,54 dias	0%	13/10/2004	25/11/2004
2-Projeto Lógico	31,29 dias	0%	25/11/2004	07/01/2005
3-Projeto Físico	30,31 dias	0%	07/01/2005	21/02/2005
4-Construção	87,07 dias	0%	21/02/2005	22/06/2005
5-Homologação	12,02 dias	0%	22/06/2005	08/07/2005

6-Implantação	10,12 dias	0%	08/07/2005	22/07/2005
---------------	------------	----	------------	------------

Tabela 3 Atividades de projeto de software

Aula 13

Revisão – Planejamento e Gerenciamento de Projeto de Software

Imagine você desejar sair da cidade onde você reside e ir a outra qualquer (supondo-se obviamente que você não reside lá). Sem um mapa, um plano ou qualquer outra fonte de informação para saber por quais cidades, menor percurso, melhores estradas, dentre outras informações, você não terá certeza de alcançar o seu objetivo.

Agora, trazendo esta meta para o contexto de um projeto de software, você necessitará de um mapa de quais atividades devem ser realizadas, sem o qual você ficará perdido. Aqui, também, um plano torna-se essencial para compreender riscos, compromissos e decisões de projeto.

Precisamos de um ‘mapa’ ou ‘guia’ que ofereça uma base sistemática de como conduzir o projeto e quaisquer modificações necessárias além de servir como eficiente mecanismo para comunicação entre os principais interessados no projeto que inclui cliente, usuário final, gerente projeto, dentre outros. Esse ‘mapa’ existe e é conhecido como **plano de projeto**. Trata-se de um dos documentos produzidos durante a realização de projeto. O plano de projeto é essencial e determinante no sucesso para uma boa condução de qualquer projeto.

A gestão de projetos define quem, o que, quando e o porquê dos projetos. Ela faz uso de processos e ferramentas de gestão os quais servem para ajudar o gerente de projetos e equipe a *organizar, documentar, rastrear e relatar as atividades e progresso de um projeto*. Dentro desse contexto, o plano de projeto compreende:

- Escopo de projeto bem definido;
- Um *roadmap* dos artefatos a serem entregues;
- Documentação de papéis e responsabilidades dos participantes;
- Uma linguagem ‘comum’ para comunicação das atividades do projeto, bem como a rastreabilidade e relatórios dessas atividades;
- Mecanismos de resolução de conflitos e mitigação ou atenuação de riscos.

Exemplo de Documento de Projeto de Software²

Nos exemplos abaixo se procura dar uma dimensão realista sobre um projeto de software. Entre os documentos dividimos em Projeto de Software e Plano de Desenvolvimento de Software.

1. Introdução

Este documento apresenta o planejamento do projeto do sistema Exemplo o qual será utilizado como base às atividades de acompanhamento, revisão, verificação e validação do projeto desde seu início até sua conclusão, a fim de garantir a análise comparativa do desempenho real *versus* planejado. Desta forma, ações corretivas e preventivas poderão ser tomadas, sempre que resultados ou desempenhos reais desviarem significativamente do planejado. Sua elaboração é derivada das informações contidas no Plano de Trabalho e convênio assinado com o cliente.

1.1 Termos e acrônimos

² O modelo apresentado foi extraído da edição 6 da revista “Engenharia de Software Magazine”, em Gerenciamento de Projetos.

Esta seção explica o conceito de um subconjunto de termos importantes que serão mencionados no decorrer deste documento. Estes termos são descritos na Tabela 2, estando apresentados por ordem alfabética.

Termo	Descrição
Artefato	Tudo que é produzido e documentado em qualquer atividade de qualquer fluxo do projeto. Por exemplo: documento de requisitos, diagrama de casos de usos e glossário.
<i>Milestone</i>	Ponto de checagem; marco que indica a conclusão de uma fase ou etapa.
NA	Não Aplicável
Patrocinador	Representante da empresa cliente ou contratada responsável pelo sucesso do projeto em instância superior, garantindo o cumprimento de responsabilidades estabelecidas.
Revisão	Apresentação de produtos de software para os interessados visando comentário e aprovação dos mesmos.
SQA	Software Quality Assurance, profissional ou grupo responsável por garantir a qualidade do produto de software e processo de desenvolvimento.

Tabela 4 Termos e acrônimos do projeto

2. Visão Geral do Projeto

Analisando-se os aspectos técnicos (métodos, processos e ferramentas) e não técnicos (gerenciamento, planejamento e questões econômicas) de produção de aplicativos de software baseados em componentes, este projeto propõe construir a infra-estrutura necessária para o desenvolvimento de componentes e aplicativos, fazendo uso da plataforma tecnológica orientada a serviços (web services).

Nesse sentido, os componentes orientados a serviço a serem desenvolvidos pela Empresa AM Ltda (*desenvolvedora do projeto*) servirão de base para construção de aplicações pelo cliente direcionadas à área de Turismo. Como resultado, isto permitirá elevar a produtividade e competitividade, promovendo a posição do cliente no mercado com uso de soluções tecnologicamente avançadas. Este projeto propõe ainda realizar pesquisa e desenvolvimento da infra-estrutura para o desenvolvimento de componentes e aplicativos orientado a serviços a serem usados pelo cliente.

2.1 Participantes

Esta seção lista o conjunto de participantes e parceiros envolvidos no desenvolvimento do projeto que serão mencionados no decorrer deste documento. Esta lista é apresentada por ordem alfabética.

- Empresa AM Ltda no papel de Executor
- Organização BrasilTur no papel de Cliente / Financiador

2.2 Objetivos Específicos

De acordo com o plano de trabalho assinado com o Cliente, os objetivos desse projeto compreendem:

- Análise de plataformas tecnológicas;
- Definição de um modelo de desenvolvimento de web services;
- Desenvolvimento de web services para monitoração e controle de acesso;
- Desenvolvimento de web services para controle de qualidade de serviço;
- Levantamento e avaliação de requisitos de componentes e aplicações de negócio para a área de Turismo;
- Desenvolvimento de protótipo: aplicação de web services para área de Turismo;

2.3 Critérios de Aceitação do Projeto

A aceitação final do projeto está condicionada a:

- Todos os artefatos e indicadores físicos de execução descritos na seção 9.1 devem ter sido aprovados pela Divisão de Qualidade da empresa;
- Todos os objetivos listados na seção 2.2 devem ter sido atingidos.

2.4 Mecanismos de Evolução do Plano de Projeto

O plano do projeto deve ser mantido atualizado para refletir a situação corrente do projeto. Dessa forma, as seguintes situações representam os gatilhos para atualização deste documento:

- Alterações de rubricas junto ao patrocinador (Cliente);
- Mudanças nos critérios de aceitação do projeto;
- Alterações dos objetivos previstos no Plano de Trabalho aprovado pelo cliente;
- Mudanças na gerência do projeto da empresa ou do cliente.

3. Requisitos do Sistema

Esta seção apresenta os requisitos do sistema que servirão de base ao seu planejamento, bem como do escopo não contemplado (ou escopo negativo). Mudanças nestes requisitos devem ser submetidas ao controle de mudanças estabelecido para o projeto, descrito no plano de gerencia de configuração.

3.1 Requisitos Técnicos

Os requisitos a seguir representam uma visão dos produtos a serem desenvolvidos nesse projeto. Estes requisitos serão descritos em detalhes no Documento de Requisitos do Projeto, que será complementado e refinado no decorrer do ciclo de vida do projeto.

3.1.1 Requisitos Funcionais

Os requisitos funcionais considerados compreendem:

- A infra-estrutura de desenvolvimento de componentes e aplicativos será baseada na plataforma orientada a serviços (web services) e compreenderá:
 - Web services de controle de acesso;
 - Web services de controle de qualidade de serviço;
 - Web services para área de Turismo;
- Desenvolvimento de web services para uma subárea de aplicação de Turismo, bem como disponibilizar seus respectivos componentes de serviços no repositório utilizado. A definição da subárea de Turismo será realizada na etapa inicial deste projeto.

3.1.2 Requisitos Não Funcionais

Os requisitos não funcionais considerados neste projeto compreendem:

- O modelo de desenvolvimento de web services deverá considerar uma infra-estrutura que permita o desenvolvimento rápido de componentes e aplicativos baseado na plataforma orientada a serviços. O conjunto de componentes orientados a serviços resultantes deste projeto servirá de base para a construção de várias aplicações para o setor de turismo.
- O processo de validação de web services se encerrará com a entrega da versão final dos web services.

3.4 Mecanismos de Evolução do Plano de Projeto

O plano do projeto deve ser mantido atualizado para refletir a situação corrente do projeto. Abaixo, é apresentado um conjunto de situações que requer a atualização deste documento:

- Alterações de rubricas junto ao cliente;
- Mudanças nos critérios de aceitação do projeto;
- Alterações dos objetivos previstos no Plano de Trabalho aprovado pelo cliente;
- Mudanças na gerência do projeto da empresa ou do cliente.

3.2 Requisitos Não Técnicos

Os requisitos não técnicos compreendem:

- Todo e qualquer material de divulgação resultante da execução deste projeto deverá conter informação do suporte financeiro do cliente e, especialmente, no caso de:
 - Seminários e eventos científicos e tecnológicos;
 - Publicações técnicas e científicas em revistas especializadas;
 - Relatórios técnicos publicados ou divulgados em qualquer mídia.

3.3 Escopo Não Contemplado

O escopo do projeto não contempla a realização das seguintes atividades:

- Elaborar e/ou realizar treinamento no uso dos web services desenvolvidos no projeto;
- Correção de bugs, identificados em qualquer um dos produtos, após a duração prevista do projeto perante o cliente.

Quaisquer outros artefatos não previstos na seção de produtos deste plano também são considerados como não contemplado no escopo do projeto.

4. Organização do Projeto

Esta seção apresenta o organograma utilizado no projeto juntamente com seus papéis e responsabilidades.

4.1 Organograma

Esta seção apresenta o organograma do projeto, incluindo os papéis exigidos para realização do projeto e a relação entre os mesmos.

4.2 Papéis e Responsabilidades

A tabela a seguir descreve um conjunto de papéis e respectivas responsabilidades.

Papel	Responsabilidades
Diretor Executivo	<ul style="list-style-type: none"> • Divulgar as diretrizes estratégicas; • Tomar decisões estratégicas; • Garantir o cumprimento de responsabilidades estabelecidas entre as partes, possibilitando o sucesso do projeto; • Apoiar as decisões da equipe do projeto.
Gerente de Operações	<ul style="list-style-type: none"> • Prover todos os recursos necessários para a execução do projeto (capital humano, hardware, software, treinamento, etc.); • Realizar acompanhamento técnico, financeiro, de escopo, riscos e cronograma do projeto, conjuntamente com o gerente de projeto; • Assumir a responsabilidade sobre todo o ciclo de vida do serviço; • Posicionar o cliente sobre o andamento dos serviços junto com o gerente do projeto; • Negociar junto ao gerente de qualidade a resolução de questões de qualidade não solucionadas no âmbito do projeto.
Gerente de Negócios	<ul style="list-style-type: none"> • Elaborar propostas comerciais; • Conduzir negociação com o cliente quando houver mudanças no custo do projeto e que impactam o cliente.
Gerente de Projeto	<ul style="list-style-type: none"> • Realizar planejamento do projeto; • Gerenciar a equipe do projeto; • Gerenciar o orçamento do projeto; • Garantir o andamento adequado do projeto com relação ao planejado, gerenciando riscos e tomando ações preventivas e corretivas; • Posicionar o cliente sobre o andamento dos serviços; • Elaborar relatório de acompanhamento e conclusão do projeto; • Coordenar a interação da equipe com o cliente.
Analista de Negócio	<ul style="list-style-type: none"> • Realizar modelagem do negócio, quando apropriado; • Elicitar requisitos e realizar análise e projeto do sistema, elaborando modelos associados; • Elaborar projeto de testes e conduzir testes de sistema; • Elaborar documentação técnica necessária, por exemplo, ajuda (<i>help</i>), guia de usuário, material de treinamento; • Acompanhar atividades dos engenheiros de software, assegurando integridade com requisitos e casos de uso especificados; • Conduzir implantação do sistema.
Arquiteto de Software	<ul style="list-style-type: none"> • Definir a arquitetura do sistema; • Liderar e coordenar as atividades de engenharia de software do projeto; • Suportar o uso de ferramentas no âmbito do projeto; • Acompanhar os engenheiros de software, esclarecendo dúvidas técnicas; • Participar dos testes integrados do sistema; • Integrar os diversos componentes de software produzidos, gerando versão do sistema para implantação.

Papel	Responsabilidades
Desenvolvedor	<ul style="list-style-type: none"> • Implementar componentes do sistema; • Realizar testes unitários dos componentes de software, de acordo com os padrões adotados pelo projeto; • Participar da fase de projeto, quando apropriado; • Participar dos testes integrados do sistema.
Engenheiro de Qualidade	<ul style="list-style-type: none"> • Documentar e configurar o processo de software a ser utilizado; • Auditar o uso do processo; • Participar de revisões quando adequado; • Apoiar uso processo.

Tabela 5 Tabela de papéis x Responsabilidades

5. Equipe e Infra-Estrutura do Projeto

Esta seção define a composição da equipe e lista a relação de ferramentas (**Tabela 6**) necessárias ao ambiente de desenvolvimento do projeto com o objetivo de garantir uma estrutura adequada para a execução das atividades previstas neste plano.

5.1 Planejamento da Alocação de Pessoal

A tabela a seguir apresenta o planejamento de alocação de pessoal do projeto. A quantidade de funções em relação ao cronograma foi reduzida no sentido de simplificação.

Função	Quantidade	% Alocação
Gerente de projeto	1	100
Analista de negócio	1	100
Arquiteto de software	1	50
Engenheiro de software	1	100
Web designer	1	50
Engenheiro de Qualidade	1	20
Engenheiro de Configuração	1	20
Administrador de Banco de Dados	1	10
Administrador de Sistemas	1	10

Tabela 6 Planejamento de Alocação de Pessoal

5.2 Equipe de Projeto

A tabela a seguir lista a relação de participantes do projeto e informações de período de participação e formas de contato. Novamente, o número de linhas nesta tabela foi reduzido para efeitos de simplificação, em comparação com o quantitativo da tabela anterior.

Nome	Papel	Período	E-mail	Telefone
Nome do gerente	Gerente de Projetos	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9990

Nome do analista	Analista de Negócio	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9992
Nome do arquiteto	Arquiteto de Software	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9993
A definir	Engenheiro de configuração	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9997
Engenheiro	Web designer	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9999
Engenheiro	Administrado de dados	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9900
A definir	Administrador de Sistemas	03/2007 a 12/2007	nome@empresa.com.br	(11) 9999 9901

Tabela 7 Equipe da empresa

5.3 Ferramentas do Projeto

Funcionalidade	Ferramenta	Nº Licenças Necessárias
Gerência de projetos	MSPProject	1
Gerência de configuração	CVS	0
Análise e projeto	Rational Rose	6
Implementação	JBuilder	6
Testes	A ser definido	-
SGBD	A ser definido	-

Tabela 8 Ferramentas do projeto

6. Treinamentos do Projeto

Esta seção apresenta treinamentos previstos no projeto e a maneira pela qual eles serão realizados.

O treinamento da equipe técnica envolvida no desenvolvimento do projeto já faz parte do planejamento corporativo de capacitação de pessoal, de acordo com o plano de cargos em vigor na empresa, o qual descreve os perfis necessários para a execução de diferentes papéis. Desta forma, os profissionais selecionados já possuem perfil necessário para o cumprimento dos compromissos firmados. Uma vez que o projeto não incorpora nenhuma tecnologia desconhecida pela equipe técnica, não serão necessários treinamentos adicionais. Entretanto, qualquer treinamento necessário à equipe será relatado no primeiro relatório de acompanhamento de projeto.

7. Acompanhamento do Projeto

Esta seção apresenta as atividades de acompanhamento e verificação do projeto, envolvendo a equipe do projeto, gerente de projeto da empresa, gerente de operações e representante do cliente. Estas atividades incluem a realização de reuniões e geração de relatórios descrevendo informações sobre

o progresso do projeto, questões não resolvidas, dentre outras. A tabela a seguir contempla as atividades de acompanhamento planejadas para o projeto.

Reunião / Relatório	Realização	Participantes / Interessados
Reunião de acompanhamento de atividades	Semanal	<ul style="list-style-type: none"> Gerente de projeto da empresa; Equipe de projeto da empresa.
Relatório de progresso do projeto	Mensal	<ul style="list-style-type: none"> Gerente de Operações; Gerente de projeto da empresa; Representante do cliente, se apropriado.
Revisão formal do projeto	Semestral	<ul style="list-style-type: none"> Gerente de projeto da empresa; Representante do cliente, se apropriado.
Reunião de fechamento do projeto com a equipe	Ao final do projeto	<ul style="list-style-type: none"> Gerente de projeto da empresa; Equipe do projeto da empresa; Engenheiro de qualidade da empresa.
Reunião de fechamento do projeto com o cliente	Ao final do projeto	<ul style="list-style-type: none"> Gerente de projeto da empresa; Representante(s) do cliente.

Tabela 9 Reuniões e Relatórios de Acompanhamento do Projeto

As reuniões e relatórios apresentados possuem o seguinte objetivo:

- **Reunião de acompanhamento de atividades:** tem por objetivo coletar periodicamente informações junto à equipe, cliente e demais áreas envolvidas, além de tomar ações corretivas quando forem identificados desvios do planejado.
- **Relatório de progresso do projeto:** tem por objetivo comunicar mensalmente os principais interessados sobre o andamento do projeto.
- **Revisão formal do projeto:** visa principalmente comunicar a gerência da empresa e representantes do cliente sobre status das atividades do projeto. Adicionalmente, deve-se verificar se o trabalho essencial da etapa anterior foi completado com sucesso (planejado vs. realizado), determinando pré-condições para o sucesso da próxima etapa, resolver questões do projeto, reafirmar compromissos e reavaliar riscos. Estas reuniões deverão ser realizadas a cada semestre e servem como prestação de contas para o cliente.
- **Reunião de fechamento do projeto com a equipe:** visa comunicar o *feedback* do cliente à equipe, além de discutir as lições aprendidas com o projeto e avaliar o *feedback* da equipe.
- **Reunião de fechamento do projeto com o cliente:** tem por objetivo avaliar a realização dos compromissos firmados entre as partes e obter o aceite formal do projeto pelo cliente. Esta formalização deve ser realizada através do formulário de aceitação do produto/serviço.

8. Controle de Mudanças do Escopo de Projeto

O controle de mudanças de escopo do projeto considera solicitações referentes a alterações nas especificações funcionais ou técnicas, adição de novos requisitos, serviços adicionais de consultoria ou apoio técnico, alterações de cronograma e/ou na administração do projeto como um todo. Tais solicitações podem ser propostas pelo cliente ou pela equipe de projeto da empresa, sendo passíveis de um novo dimensionamento do esforço e custo necessários a sua implementação. As solicitações de mudança devem ser registradas na ferramenta de controle de mudanças (e.g. CVS).

O controle de mudanças permite ainda a realização de acordo entre as partes, considerando o impacto sobre o projeto, sobre os acordos e compromissos previamente estabelecidos, e sobre os cronogramas físico e financeiro do projeto. Os custos associados à mudança deverão ser apoiados pelo cliente.

9. Cronograma

O cronograma do projeto contempla as atividades, *milestones*, dependências e recursos humanos alocados. Para obter detalhes sobre o mesmo, o documento do Cronograma do Projeto deve ser consultado.

9.1 Marcos Significativos do Projeto

A tabela a seguir apresenta os marcos significativos do projeto, com datas fictícias, bem como os artefatos importantes que serão entregues ao cliente nestes marcos, quando aplicável. Mudanças acordadas nas datas alvo serão acompanhadas e registradas, através das reuniões de acompanhamento do projeto.

Marco / Meta Física	Artefatos/ Indicadores Físicos de Execução	Resp.	Data Alvo
Estudo e análise de soluções tecnológicas para infra-estrutura orientada a serviços (web services).	Relatório técnico de estudo e análise	Empresa	02/10/2007
Definição de um modelo de desenvolvimento de web services.	Relatório técnico de modelo	Empresa	27/10/2007
Desenvolvimento de web services para controle de acesso	Documento de requisitos de web services de controle de acesso; Documento de análise e projeto; Código fonte implementado; Documento do plano de testes e dos resultados de testes.	Empresa	25/01/2008
Desenvolvimento de web services para a área de Turismo	Documento de requisitos de web services para a área de Turismo; Documento de análise e projeto; Código fonte implementado; Documento do plano de testes e dos resultados de testes.	Empresa	25/03/2008

Tabela 10 Marcos Significativos do Projeto

10. Gerência de Riscos

Os riscos identificados para o projeto serão detalhados nos relatórios de acompanhamento, bem como na planilha de acompanhamento do projeto. Estes documentos contêm a lista de riscos identificados, seus impactos e informações relevantes para definir estratégia de controle e atenuação (ou mitigação) do risco. Todo o acompanhamento dos riscos do projeto (riscos previamente identificados e riscos surgidos no decorrer do andamento do projeto) será registrado nos documentos supracitados.

11. Gerência de Configuração

A gerência de configuração do projeto será detalhada no Plano de Gerência de Configuração do Projeto. A gerência de configuração visa estabelecer e manter a integridade dos produtos de um projeto de software durante o seu ciclo de vida. Suas atividades envolvem identificar a configuração do software, manter sua integridade durante o projeto e controlar sistematicamente as mudanças.

Os artefatos do projeto deverão ser disponibilizados no repositório do projeto. Além disso, os documentos de interesse do cliente e dos gestores sênior ou diretores da empresa serão disponibilizados no site do projeto, cujo acesso será restrito aos principais envolvidos.

12. Testes do Projeto

Os testes são aplicáveis apenas para a implementação dos web services. Os testes de desempenho, carga, estresse e segurança e controle de acesso só se aplicam, quando pertinente, aos web services.

12.1 Estágios de Testes

A tabela a seguir apresenta os estágios de testes previstos para o projeto e seus objetivos. O planejamento e o controle dos testes é apresentado em outro documento (Plano de Testes).

Estágio de Testes	Objetivo
Teste Unitário	Visa validar individualmente os menores componentes (classes básicas e componentes) que serão utilizados na implementação das funcionalidades do sistema.
Teste de Integração	Objetiva validar a integração entre componentes e dos diversos pacotes na implementação das funcionalidades.
Teste de Sistema	Objetiva validar se todos os elementos do sistema foram adequadamente integrados e se realizam corretamente as funções especificadas.
Teste no Ambiente de Aceitação	Visa assegurar que tudo está realmente pronto para ser utilizado pelo usuário. Estes testes são realizados pela equipe de projeto da empresa antes da entrega do sistema ao cliente. Deve ser realizado em um ambiente o mais próximo possível do ambiente de produção.
Teste de Aceitação	Teste realizado pelo cliente objetivando aceitar ou homologar o sistema. Depois de realizado este teste com sucesso, o sistema estará pronto para ser implantado no ambiente de produção.

Tabela 11 Estágios de Testes do Projeto

Métricas e Estimativas de Custo

Métricas de processo e de projeto de software são medidas quantitativas que permitem aos engenheiros de software ter idéia da eficácia do processo de software e dos projetos que são conduzidos usando o processo como estrutura.

Dados básicos de qualidade de produtividade são coletados. Esses dados são então analisados, comparados com médias anteriores e avaliados para determinar se ocorreram melhorias de qualidade e produtividade.

Métricas neste contexto são importantes, pois se as medições não ocorrem o julgamento de determinada situação pode ser baseado somente em avaliação subjetiva. Com avaliações (boas ou ruins) podem ser detectadas tendências, melhores estimativas podem ser feitas e aperfeiçoamentos reais podem ser obtidos ao longo do tempo.

Métricas de processo são coletadas no decorrer de todos os projetos e durante longos períodos. Seu objetivo é fornecer um conjunto de indicadores de processo que leva a aperfeiçoamentos do processo de software no longo prazo.

Métricas de projeto permitem ao gerente de projeto de software:

1. Avaliar o estado de um projeto em andamento;
2. Acompanhar riscos potenciais;
3. Descobrir áreas-problema antes que elas se tornem críticas;
4. Ajustar fluxo de trabalho ou tarefas;
5. Avaliar a capacidade da equipe de projeto de controlar a qualidade dos produtos do trabalho de software.

Métricas de projeto são utilizadas no decorrer de um projeto, para analisar situações reais e atuais, já as métricas de processo, são coletadas e armazenadas no contexto de processo e são utilizadas sempre como norma dos processos durante a realização de projetos de software (novos ou alterações).

Com referência a figura a seguir, o processo situa-se no centro de um gráfico que liga três fatores com profunda influência na qualidade de software e no desempenho da organização. O triângulo do processo encontra-se no centro de um círculo de condições que incluem o ambiente de desenvolvimento (por exemplo, ferramentas CASE), condições de negócio (por exemplo, prazos e regras de negócio) e características do cliente (por exemplo, facilidade de comunicação).

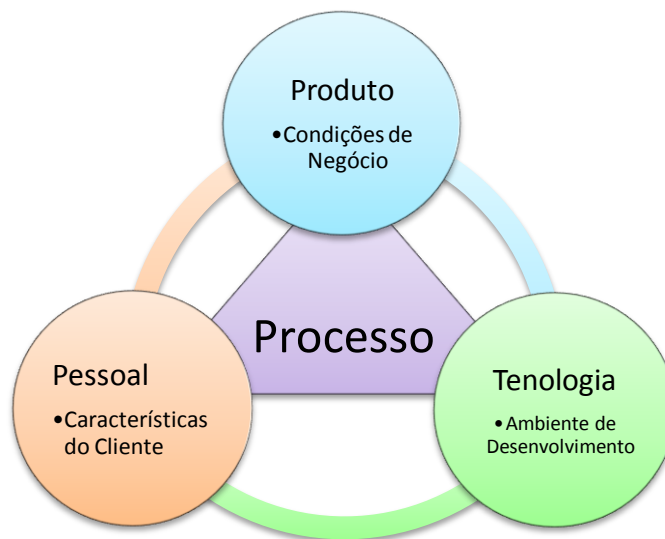


Figura 28 Determinantes da Qualidade de Software e da efetividade organizacional

Existem basicamente dois tipos distintos de métricas para processos: as métricas públicas e as privadas. As métricas privadas devem servir como indicador apenas para o indivíduo. Por exemplo, proporção de defeitos por indivíduo e por componente de softwares e erros encontrados durante o desenvolvimento. Dados privativos de processo podem servir como fator motivacional, à medida que o indivíduo individual trabalha para se aperfeiçoar.

Métricas públicas geralmente assimilam informações que eram originalmente privadas, de indivíduos e equipes. Proporções de defeitos de projeto, esforço, tempo transcorrido e dados relacionados são coletados e avaliados em uma tentativa de descobrir indicadores que possam aperfeiçoar o desempenho do processo organizacional.

Princípios de Medição

Antes de prosseguirmos com métricas, precisamos entender alguns princípios básicos da medição. Podemos caracterizar um processo de medição por cinco atividades básicas:

1. *Formulação*: A derivação de medidas e métricas de software adequadas para a representação do software que está sendo considerado;
2. *Coleta*: Mecanismo usado para acumular os dados necessários para derivar as métricas formuladas;
3. *Análise*: Cálculo de métricas e aplicação das ferramentas matemáticas;
4. *Interpretação*: Avaliação das métricas em um esforço para ganhar profundidade na visão da qualidade de representação;
5. *Realimentação*: Recomendações derivadas da interpretação das métricas de produto transmitidas à equipe de software.

Métricas de software tem valor somente se forem caracterizadas efetivamente e validadas de modo que seu valor seja provado. Os seguintes princípios são representativos de muitos que podem ser propostos para caracterização e validação de métricas:

- A métrica deve ter propriedades matemáticas desejáveis: Isto é, o valor da métrica deve ter um intervalo significativo;
- Quando uma métrica representa uma característica de software que aumenta ou diminui quando eventos (bons ou ruins) ocorrem, o valor da métrica deve aumentar ou diminuir do mesmo modo;

- Cada métrica deve ser validada empiricamente em uma ampla variedade de contextos antes de ser publicada ou usada para tomar decisões.

Tipos e Aplicação de Métricas de Produto

Métricas para o modelo de Análise: Essas métricas tratam de vários aspectos do modelo de análise e incluem:

- *Funcionalidade entregue* – fornece uma medida indireta da funcionalidade que é empacotada com o software;
- *Tamanho do sistema* – mede o tamanho global do sistema definido em termos de informação disponível como parte do modelo de análise;
- *Qualidade de especificação* – fornece uma indicação da especialidade e completeza de uma especificação de requisitos.

Métricas para o modelo de projeto: Quantificam os atributos do projeto de modo que permita um efetivo controle de qualidade. A métrica inclui:

- *Métrica arquitetural* – fornece uma indicação da qualidade do projeto arquitetural;
- *Métrica no nível de componente* – mede a complexidade dos componentes de software e outras características que têm influência na qualidade;
- *Métricas de projeto de interface* – focalizam principalmente a usabilidade;
- *Métricas especializadas em projeto OO* – medem características de classes (comunicação e colaboração);

Métricas para código-fonte: Servem para medir o código-fonte:

- *Métricas de Halstead* – controversas, essas métricas fornecem medidas singulares de um programa de computador;
- *Métricas de complexidade* – medem a complexidade lógica do código-fonte;
- *Métricas de comprimento* – fornecem uma indicação do tamanho do software.

Métricas de teste: Essas métricas ajudam o projeto de casos de teste efetivos e avaliam a eficácia do teste:

- *Métricas de cobertura de comando e desvio* - Levam ao projeto de casos de teste que fornecem cobertura ao programa;
- *Métricas relacionadas a defeito* - enfocam erros encontrados, em vez dos testes em si;
- *Efetividade de teste* - fornecem uma indicação em tempo real da efetividade dos testes que foram conduzidos;
- *Métricas em processo* – métricas relacionadas a processo que podem ser determinadas a medida que o teste é conduzido.

Método GQM

Por existirem muitas medidas diferentes para software, é importante saber escolher quais serão usadas em um projeto. Essa escolha deve levar em conta características como o custo de aplicação de uma medida e, o que é óbvio, considerar os objetivos do projeto.

Uma forma organizada de tratar o planejamento do trabalho de medição é o método GQM (Goal-Question-Metric) [Basili e Weiss, 1984]. Este método é muito utilizado e é base para a ISO/IEC 9126. O Método GQM organiza o planejamento de uma medição de software em etapas. A cada etapa deve-se definir um elemento conforme descrito a seguir:

- **Objetivos** – são estabelecidos de acordo com as necessidades dos stakeholders. Os objetivos de medição devem ser fixados em função dos requisitos do software. Em particular, uma análise de importância de cada requisito, é útil para controlar os custos de avaliação. Aos requisitos mais importantes podem ser alocados mais recursos, tais como tempo e quantidade de usuários contratados para teste.

- **Questões** – são definidas para realizar o trabalho de medição. São as perguntas que se espera responder com o estudo. As respostas obtidas com a medição devem trazer informação útil para melhorar o produto. Por exemplo: “Que aspectos do projeto (design) da interface afetam a facilidade de uso?”. As questões estabelecem uma ponte entre os objetivos planejados e as métricas que devem trazer evidência sobre o sucesso ou não da implementação;
- **Categorias** – particionam o conjunto de dados obtidos. As perguntas criadas no passo anterior podem trazer à tona diferentes tipos de informação. No exemplo citado de avaliação de uma interface, pode-se pensar em vários aspectos que correspondam a diferentes categorias de informação, como quantidade de janelas, distribuição das informações, linguagem utilizada, etc;
- **Formulários** – conduzem o trabalho dos avaliadores. A vantagem de definir os documentos para anotação dos dados é evitar que cada avaliador utilize um formato próprio, o que, além de dificultar a tarefa de analisar as informações, pode induzir a erros como coleta de dados diferentes.

Estimativa do Projeto de Software

O Software é o elemento virtualmente mais caro de todos os sistemas baseados em computador. Para sistemas complexos, feitos sob medida, um erro de estimativa grande pode fazer a diferença entre lucro e prejuízo. Excesso de custo pode ser desastroso para o desenvolvedor.

A estimativa de custo e do esforço de software nunca será uma ciência exata. Um grande número de variáveis – humanas, técnicas, ambientais, políticas – pode afetar o custo final do projeto e o esforço aplicado para desenvolvê-lo.

Existem algumas regras que podem ser úteis para a obtenção de estimativas de custo e esforço confiáveis:

1. Adie a estimativa mais tempo que puder. Quanto mais adiantado estiver o projeto melhor será a avaliação de estimativa. Infelizmente esta opção (apesar de ser muito boa) não é prática. Estimativas de custo precisam ser fornecidas logo;
2. Baseie as estimativas em projetos anteriores, que sejam parecidos. Poderá funcionar muito bem se o presente projeto for muito semelhante a outros. Infelizmente a experiência anterior nem sempre é um bom indicador de resultados futuros;
3. Use técnicas de decomposição simples para gerar estimativas de custo e esforço;
4. Use um ou mais modelos empíricos para estimativas.

Técnicas de Decomposição

Desenvolver uma estimativa de custo e esforço para um projeto de software é muito complexo para ser considerado como um todo. Por essa razão devemos decompor o problema, recaracterizando-o como um conjunto de problemas menores.

No contexto do planejamento de projeto, o tamanho refere-se ao resultado quantificável do projeto de software. Se uma abordagem direta é adotada, o tamanho pode ser medido em linhas de código (LOC). Se uma abordagem indireta é escolhida, o tamanho é representado como pontos por função (FP).

Dados de LOC e FP são usados de dois modos durante a estimativa de projetos de software:

1. Como variável de estimativa para “dimensionar” cada elemento de software;
2. Como métricas referenciais coletadas de projetos anteriores e usadas juntamente com variáveis de estimativa para desenvolver projeções de custo e esforço.

Quando um novo projeto é estimado, deve ser primeiro classificado em um domínio (tamanho de equipe, área de aplicação, complexidade e outros parâmetros relevantes) e depois a média de produtividade do domínio adequado deve ser usada para gerar a estimativa.

Estimativa Baseada em Linhas de Código (LOC)

A medida mais simples para tamanho de um programa é a contagem de linhas de código. Existem algumas variantes para ela, como LOC (Lines Of Code), SLOC (Source Lines Of Code) lógico e SLOC físico.

O método mais simples e também o mais impreciso: utilizando-se LOC não se distingue linhas de código de linhas em branco ou de comentários. Essa métrica é, em princípio, o padrão indicado para uso no PSP (Personal Software Process). Pode ser suficientemente precisa quando se comparam grandes volumes de código: naturalmente, um programa de 4 KLOC (milhares de linhas) deve ser mais complexo e levar mais tempo para ser desenvolvido que outro de 2 KLOC.

Para aumentar a precisão da medida, sugeriu-se o uso do SLOC, onde linhas em branco ou com comentários não são consideradas. As variantes “lógica” e “física” não têm definições amplamente padronizadas.

O exemplo abaixo seria medido como três LOCs, dois SLOCs físicos e três SLOCs lógicos:

```
//uma comparação
IF (a > b) {a++; puts ("ok");
}
```

Estimativa Baseada em Pontos de Função (FP)

Como vimos, a medida LOC é simples e por esse motivo também é imprecisa. Um programador gastaria no máximo alguns minutos para juntar os arquivos-fonte de um projeto e obter do editor a contagem total de linhas. Entretanto o número obtido não revelaria nada a propósito da estrutura de tal sistema.

Para exemplificar essa situação, considere estas duas linhas:

```
s := 'TESTE';
s := IfThen(StringReplace(s,'ES','es',[rfReplaceAll]) = 'TestE','OK','');
```

Embora na segunda linha exista claramente uma estrutura mais complexa da arquitetura do programa, na contagem LOC tem o mesmo peso da primeira linha. Para corrigir isso é necessária uma avaliação em um nível de abstração maior ao analisar o “tamanho” do software. A análise de pontos de função provê solução para isso.

Esta medida pode ser aplicada antes de o software ser construído, baseando-se na descrição arquitetural do projeto. Além disso, FP é independente da tecnologia usada no desenvolvimento. A contagem de FPs pode ser aplicada logo após a definição da arquitetura, permitindo estimar o esforço e o cronograma de implementação de um projeto. Também pode ser aplicada a sistemas já em funcionamento, substituindo LOC para efeito de estimas esforço de manutenção.

Para determinar o número de FPs, deve-se considerar a contagem de dados e de transações. Na contagem de dados, consideram-se arquivos lógicos internos ou arquivos de interface externos. Entre as transações têm-se entradas externas, saídas externas ou consultas externas.

O termo arquivo refere-se a um grupo de dados logicamente relacionados:

- **Arquivos lógicos internos (ALI):** São relativos às funcionalidades do software, como, por exemplo, cadastro de clientes e alteração de produtos. Não são ALI, por exemplo, arquivos temporários e de classificação;

- **Arquivos de interface externos (ALE):** Os dados são mantidos e/ou alimentados por outros programas. As transações representam as funcionalidades providas ao usuário pelo processamento dos dados em um software;
- **Entradas externas (EE):** Processo lógico onde dados são introduzidos no sistema. Os dados podem ser informações de controle ou de negócios. Por exemplo, exclusão, alteração e inclusão de registros;
- **Saídas externas (SE):** Processo lógico em que dados são enviados ao exterior das fronteiras do sistema. Por exemplo, um relatório ou mesmo a apresentação dos dados em tela;
- **Consulta Externa (CE):** Os dados são recuperados exclusivamente de arquivos internos e interfaces externas (nada é computado). Por exemplo, consulta ao cadastro de cliente.

Após terem sido identificados e contados, esses cinco fatores devem ter sua complexidade classificada como baixa, média ou alta. Essa classificação depende de uma análise dos elementos de dados que compõe cada fator. Cada organização deve padronizar seu método de classificação de complexidade.

Dois programas distintos podem ter a mesma contagem de pontos de função. Aspectos como complexidade dos cálculos ou requisitos como tempo real não entram no cálculo, até então. Para resolver isso pode ser utilizada uma técnica conhecida como Fator de Ajuste, que é baseada em 14 características gerais do sistema:

1. Comunicação de Dados;
2. Funções distribuídas;
3. Desempenho;
4. Configuração do equipamento;
5. Volume de transações;
6. Entrada de dados on-line;
7. Interface com o usuário;
8. Atualização on-line;
9. Processamento complexo;
10. Reusabilidade;
11. Facilidade de implantação;
12. Facilidade operacional
13. Múltiplos locais;
14. Flexibilidade de mudanças.

Cada CGS possui um nível de influência de 0 a 5, e é calculado por: $FA = 0.65 + 0.01 \times (n1 + n2 + \dots + n14)$

Onde cada n_i representa um dos 14 níveis de influência.

A partir do valor do FA, pode-se calcular o número de pontos de função ajustados (PFA): $PFA = FA \times PF$

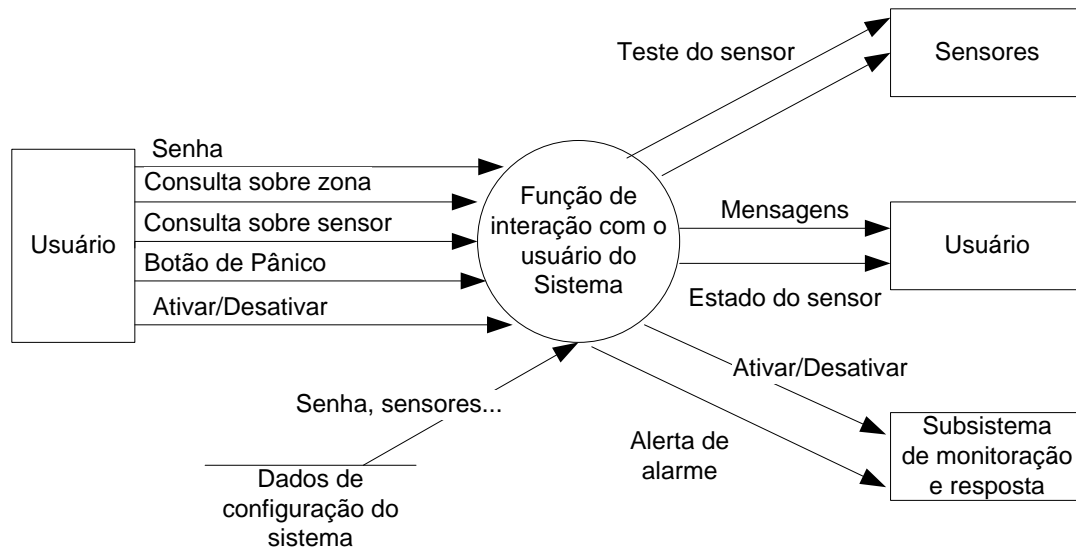
Algumas métricas podem ser derivadas a partir do cálculo de pontos de função, como o número de homens/hora para produzir 1PF (ou 1PFA, Ponto de Função Ajustado) ou, ainda, dada a produtividade média da equipe, o tempo necessário para implementar um projeto.

Por exemplo, se uma equipe produz em média 10 PFs por dia, um projeto cujo tamanho é 300 PFs deve ser implementado em um mês. De maneira semelhante pode-se aproximar valores para o preço de um produto, a partir do custo de desenvolvimento de 1PF dentro da organização.

Exemplo de Contagem de PF

O diagrama de fluxo de dados mostrado a seguir, é avaliado para determinar as medidas-chave exigidas para o cálculo da métrica de pontos por função. Três entradas externas – **SENHA, BOTÃO DE PÂNICO E**

ATIVAR/DESATIVAR – são mostradas na figura junto com duas consultas externas – **CONSULTA DE ZONA** e **CONSULTA DE SENSOR**. Um arquivo (**ARQUIVO DE CONFIGURAÇÃO DO SISTEMA**) é mostrado. Duas saídas externas (**MENSAGENS** e **ESTADO DO SENSOR**) e quatro entradas externas (**TESTE DO SENSOR**, **ESTABELECIMENTO DE ZONA**, **ATIVAR/DESATIVAR** e **ALERTA DO ALARME**) também estão presentes.



Para o diagrama de fluxo de dados apresentado acima, teríamos a seguinte tabela de decomposição para cálculo de PFA:

Valor do Domínio da Informação	Contagem	Fator de Ponderação			Totais
		Simples	Médio	Complexo	
Entradas Externas (Eis)	3 (x)	3	4	6	9
Saídas Externas (EOs)	2 (x)	4	5	7	8
Consultas Externas (EQs)	2 (x)	3	4	6	6
Arquivos Lógicos Internos (ILFs)	1 (x)	7	10	15	7
Arquivos de Interface Externa (EIFs)	4 (x)	5	7	10	20
Contagem Total					50 PFs

Após descoberto o número de PFs, devemos fazer o cálculo de ajuste, para que tenhamos o PFA final. Para o exemplo mostrado acima, deverá ser utilizada a contagem total utilizando a equação (15-1):

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

Onde, contagem total é a soma de todas as entradas de PF obtidas e F_i ($i = 1$ a 14) são valores de ajuste de complexidade (conforme tabela mostrada anteriormente). Para a finalidade desse exemplo, consideramos que o somatório de $\sum (F_i)$ é 46 (um produto moderadamente complexo). Assim,

$$FP = 50 \times [0,65 + (0,01 \times 46)] = 56$$

Controle do Esforço da equipe de projeto

De nada adianta o uso da técnica PFA se não existir um registro histórico adequado do esforço que as equipes de desenvolvimento empregam em seus sistemas. O segredo do sucesso desta técnica é a medição adequada da produtividade da equipe de desenvolvimento. Ou seja, quanto esforço é necessário para fazer um PFA.

Exemplo: PFA's do sistema: 250

Esforço da Equipe:

Analista: 350 h
 Programador: 400 h
 DBA: 25 h
 AD: 25 h
 Suporte Técnico: 25 h
 Gerente de Conta: 50 h
 Total: 875 h
 Produtividade: 875 h/250 FPA
 Produtividade: 3,5 h/FPA

Neste exemplo, concluí-se que esta equipe leva 3 horas e meia para construir um PFA.

Com base em um sistema já feito e cujo esforço para construí-lo devidamente registrado, tem-se um histórico, que com certeza pode ser usado para projetar novos trabalhos.

É importante ressaltar que esta produtividade varia conforme a tecnologia envolvida. Por exemplo: Digamos que esta produtividade de 3,5 h/PFA seja em ambiente Oracle. Se o sistema for desenvolvido em DELPHI, provavelmente a produtividade seria outra.

Estimando Projetos

Uma das tarefas mais difíceis de um projeto é a correta aplicação dos recursos (humanos) durante seu desenvolvimento, ou seja, em que momento precisarei de mais ou menos programadores/analistas.

A técnica PFA, associada a uma outra técnica chamada COCOMO (Constructive Cost Model), introduzido por Barry Boehm, contribuem para facilitar esta tarefa. O método COCOMO nada mais é do que uma pesquisa feita em centenas de empresas norte-americanas que mapeou a distribuição do esforço durante as fases de um projeto. Os principais números são:

Etapa	Fase	Esforço (%)
Anteprojeto	• Estudo preliminar	5
	• Modelagem de Dados	25
Projeto	• Projeto de Implementação	25
	• Construção	35
Implantação	• Homologação	5
	• Implantação	5

Exemplo: PFA's do Projeto = 250

Produtividade: 3,5 h/FPA

Tempo Total: 875 h

Tempo por fase:

Etapa	Fase	Esforço (%)	Tempo (h)
Anteprojeto	• Estude Preliminar	5	44
	• Modelagem de Dados	25	219
Projeto	• Projeto de Implementação	25	219
	• Construção	35	305
Implantação	• Homologação	5	44
	• Implantação	5	44

Assim sendo distribui-se essas horas nos recursos, em um cronograma, conforme estratégia da gerência e também levando em conta os prazos finais estabelecidos pelo usuário.

Métricas de código fonte POO.

Na tabela abaixo são exibidas outras formas de medição, aplicáveis a código fonte (Orientação a Objeto):

Métrica	Aplicada em	Atributo	O que mede
Complexidade Ciclomática – CC (McGabe, 1976)	Classes e métodos	Complexidade	Complexidade e alternativas possíveis no controle de fluxo.
Profundidade da Árvore de Herança (Depth of Inheritance Tree) – DIT (Chidamber e outros, 1994)	Classes e interfaces	Herança	Reuso, compreensão e teste
Número de Classes (Number of Children) – NOC (Chidamber e outros, 1994)	Classes e interfaces	Herança	Reuso
Resposta para uma Classe (Response for Classe) – RFC (Chidamber e outros, 1994)	Classes	Comunicação	Acoplamento, complexidade e pré-requisitos para teste
Acoplamento (Coupling between object classes) – CBO (Chidamber e outros, 1994)	Classes	Comunicação	Coesão e reuso
Falta de Coesão (Lack of Cohesion in Methods – LCOM (Chidamber e outros, 1994)	Classes	Comunicação	Coesão, complexidade, encapsulamento e uso de variáveis
Peso dos métodos por classe (Weighted Methods per Class – WMC (Chidamber e Kemerer, 1994)	Classes	Complexidade	Complexidade, tamanho, esforço para manutenção e reuso
Número de métodos (Number of Methods – NOM (Lorenz e outros, 1994)	Métodos	Tamanho	Corresponde a WMC onde o peso de cada método é 1
Número de Comandos (Number of Statements – NOS (Lorenz e outros, 1994)	Métodos	Tamanho	Obter o número de sentenças em um método
Número e variáveis de instância (Number of Instance Variables – NIV (Lorenz e outros, 1994)	Classes	Tamanho	Obter o número de variáveis de instância
Número de variáveis de classe (Number of Class Variables - NCV (Lorenz e outros, 1994)	Classes	Tamanho	Obter o número de variáveis de classe
Número de métodos herdados (Number of Inherited Methods – NMI (Lorenz e outros, 1994)	Métodos	Herança	Obter o número de métodos herdados e definidos em uma superclasse
Número de métodos sobrescritos (Number of Overriden Methods – NMO (Lorenz e outros, 1994)	Métodos	Herança	Obter o número de métodos definidos em uma superclasse e redefinidos na subclasse

Cronogramação de Projetos (Dicas Práticas)

Você selecionou um modelo de processo adequado, identificou as tarefas de engenharia de software que tem que ser realizadas, estimou a quantidade de trabalho e o número de pessoas, conhece o prazo, até considerou os riscos. Agora é hora de fazer as ligações.

Apesar de haver muitas razões pelas quais o software é entregue atrasado, a maioria pode ser rastreada para uma ou mais das seguintes causas básicas:

- Data de entrega irrealística estabelecida por alguém de fora do grupo de engenharia de software e imposta a gerentes e profissionais do grupo;
- Mudanças nos requisitos do cliente não refletidas em mudanças do cronograma;
- Subestimativa honesta da quantidade de esforço e/ou quantidade de recursos que serão necessários para fazer o serviço;
- Riscos previsíveis e/ou imprevisíveis que não foram considerados quando o projeto teve início;
- Dificuldades técnicas que não puderam ser previstas a tempo;
- Dificuldades humanas que não puderam ser previstas a tempo;
- Falta de comunicação;

- Falta de gerência do projeto.

A cronogramação de projeto de software é uma atividade que distribui o esforço estimado pela duração planejada do projeto, partilhando esse esforço por tarefas específicas de engenharia de software. É importante notar que o cronograma evolui com o tempo.

Durante os primeiros estágios do planejamento do projeto, um cronograma macro é desenvolvido. Esse cronograma identifica todas as principais atividades do processo e as funções do produto a que se aplicam. A medida que o projeto avança, cada entrada nesse cronograma é refinada em um cronograma detalhado.

O exemplo a seguir refere-se a um cronograma de desenvolvimento de um projeto novo. Informações como atividades macro, atividades refinadas e suas respectivas datas são exibidas. Em um cronograma é importante que para cada atividade sejam definidas datas inicial e final.

Id	Nome da tarefa	Duração	% concluída	Início	Término
1	PJ000-Sistema	200,35 dias	0%	Qua 13/10/04	Sex 22/07/05
2	1-Ante Projeto	29,54 dias	0%	Qua 13/10/04	Qui 25/11/04
3	Levantamento Preliminar	2 dias	0%	Qua 13/10/04	Sex 15/10/04
4	Estimativa de tamanho de sistema	0,06 dias	0%	Sex 15/10/04	Sex 15/10/04
5	Distribuição em cronograma	0,06 dias	0%	Sex 15/10/04	Sex 15/10/04
6	Reuniões de análise e levantamento	4,24 dias	0%	Sex 15/10/04	Qui 21/10/04
7	Modelo de processos e funções	5,65 dias	0%	Qui 21/10/04	Sex 29/10/04
8	Modelo preliminar de dados	5,65 dias	0%	Sex 29/10/04	Seg 08/11/04
9	Validação de impacto e integração	2,83 dias	0%	Seg 08/11/04	Qui 11/11/04
10	Protótipo com funções macro	2,83 dias	0%	Qui 11/11/04	Qua 17/11/04
11	Aprovação / Validação do projeto	1,41 dias	0%	Qua 17/11/04	Qui 18/11/04
12	Contagem do tamanho do sistema	2,83 dias	0%	Qui 18/11/04	Ter 23/11/04
13	Distribuição em cronograma completo	1,41 dias	0%	Ter 23/11/04	Qui 25/11/04
14	Controle de qualidade fase 1	0,57 dias	0%	Qui 25/11/04	Qui 25/11/04
15	2-Projeto Lógico	31,29 dias	0%	Qui 25/11/04	Sex 07/01/05
16	Definição detalhada das funções	19,74 dias	0%	Qui 25/11/04	Qui 23/12/04
17	Validação das funções / sistema	3,04 dias	0%	Qui 23/12/04	Ter 28/12/04
18	Modelo definitivo de dados	1,52 dias	0%	Ter 28/12/04	Qua 29/12/04
19	Modelo definitivo de função/protótipos	1,52 dias	0%	Qua 29/12/04	Sex 31/12/04
20	Ajuste na contagem	1,52 dias	0%	Sex 31/12/04	Ter 04/01/05
21	Acerto de cronograma	1,52 dias	0%	Ter 04/01/05	Qua 05/01/05
22	Controle de qualidade fase 2	2,43 dias	0%	Qua 05/01/05	Sex 07/01/05
23	3-Projeto Físico	30,31 dias	0%	Sex 07/01/05	Seg 21/02/05
24	Migração modelo lógico de dados/Refinamento	15,12 dias	0%	Sex 07/01/05	Seg 31/01/05
25	Criação das tabelas no banco de dados	4,56 dias	0%	Seg 31/01/05	Sex 04/02/05
26	Transformação da função em módulo(3 camadas)	9,11 dias	0%	Sex 04/02/05	Qui 17/02/05
27	Controle de qualidade fase 3	1,52 dias	0%	Qui 17/02/05	Seg 21/02/05
28	4-Construção	87,07 dias	0%	Seg 21/02/05	Qua 22/06/05
29	Construção dos módulos	63,79 dias	0%	Seg 21/02/05	Sex 20/05/05
30	Testes isolados	22,78 dias	0%	Sex 20/05/05	Ter 21/06/05
31	Controle de qualidade fase 4	0,5 dias	0%	Ter 21/06/05	Qua 22/06/05
32	5-Homologação	12,02 dias	0%	Qua 22/06/05	Sex 08/07/05
33	Testes integrados pela Célula de Sistema	2,03 dias	0%	Qua 22/06/05	Sex 24/06/05
34	Treinamento da Célula de Qualidade	3,04 dias	0%	Sex 24/06/05	Qua 29/06/05
35	Testes da Célula de Qualidade	3,04 dias	0%	Qua 29/06/05	Seg 04/07/05
36	Testes Realidade Clientes pelos Técnicos	1,88 dias	0%	Seg 04/07/05	Qua 06/07/05
37	Ajustes necessários	1,52 dias	0%	Qua 06/07/05	Qui 07/07/05
38	Controle de Qualidade fase 5	0,51 dias	0%	Qui 07/07/05	Sex 08/07/05
39	6-Implantação	10,12 dias	0%	Sex 08/07/05	Sex 22/07/05
40	Infra-estrutura de implantação	1,01 dias	0%	Sex 08/07/05	Seg 11/07/05
41	Treinamento dos Implantadores, Atendentes e	3,04 dias	0%	Seg 11/07/05	Qui 14/07/05
42	Gerar instalador e migrador	1,01 dias	0%	Qui 14/07/05	Sex 15/07/05
43	Acompanhamento de implantação	5,06 dias	0%	Sex 15/07/05	Sex 22/07/05

Existem diversas ferramentas no mercado que auxiliam no processo de definição de cronograma, bem como atividades paralelas de gestão de projetos.

Teste de Software

Uma estratégia de teste de software deve ser suficientemente flexível para promover uma abordagem de teste sob medida. Ao mesmo tempo, deve ser suficientemente rígida para promover o planejamento razoável e acompanhamento gerencial, à medida que o projeto progride.

O software é testado para que erros que foram feitos inadvertidamente no momento em que ele foi projetado e construído, possam ser descobertos. Entretanto não existe um consenso sobre como os testes devem ser conduzidos. Novamente aplica-se a prática de que cada organização deve conhecer profundamente o mercado em que atua e o tipo de software que produz para que a fase de teste possa ser então norteada.

Contudo, surgem algumas questões básicas:

- Como os testes devem ser conduzidos ?
- Deve ser criado um plano forma para a condução dos testes ?
- Deve-se testar o programa inteiro como um todo ou deve-se testar apenas uma parte ?
- Deve-se reexecutar testes que já foram conduzidos quando novos componentes são adicionados a um sistema grande ?
- Quando envolver o cliente ?
- A organização deve possuir um grupo de pessoas para a condução dos testes ?

Essas e muitas outras perguntas são rotina no dia-a-dia de uma fábrica de softwares. Elas são respondidas, porém, quando se cria uma estratégia de teste de software. Uma estratégia de software deve ser construída pelo gerente do projeto, engenheiros de software e especialistas em teste.

É com frequência que o teste responde por mais esforço de projeto que qualquer outra atividade de engenharia de software. Se é conduzido ao acaso, tempo é desperdiçado, esforço desnecessário é despendido e, ainda pior, erros se infiltram sem serem descobertos.

Os testes normalmente começam “no varejo” e progridem para o “atacado”, ou seja, os primeiros testes focalizam um único componente ou um pequeno grupo de componentes relacionados e são aplicados para descobrir erros nos dados e na lógica de processamento que ficam encapsulados no(s) componente(s). Depois que componentes são testados eles precisam ser integrados até que o sistema completo seja construído.

Neste ponto uma série de testes de alto nível são executados para descobrir erros na satisfação dos requisitos do cliente. À medida que os erros são descobertos, eles precisam ser diagnosticados e corrigidos usando um processo que é chamado de depuração.

O teste de software é um elemento de um aspecto mais amplo, que é frequentemente referido como verificação e validação (V&V). *Verificação* refere-se ao conjunto de atividades que garante que o software implementa corretamente uma função específica. *Validação* se refere a um conjunto de atividades diferente que garante que o software construído corresponde aos requisitos do cliente.

Verificação: Estamos construindo o produto corretamente ?

Validação: Estamos construindo o produto certo ?

Verificação e validação abrangem um amplo conjunto de atividades de garantia de qualidade (GQS – Garantia de Qualidade de Software, ou em inglês SQA), que inclui revisões técnicas formais, auditoria de qualidade e

configuração, monitoramento de desempenho, simulação, estudo de viabilidade, revisão da documentação, revisão da base de dados, análise de algoritmos, teste de desempenho, teste de usabilidade, teste de qualificação e teste de instalação.

O teste oferece efetivamente o último reduto no qual a qualidade pode ser avaliada e, mais pragmaticamente, erros podem ser descobertos. Mas o teste não deve ser encarado como rede de proteção. Como sabiamente se diz: “Você não pode testar a qualidade de um software. Se ela não estiver lá antes de começarem os testes, ela não estará lá quando terminarem os testes”, pois, como sabemos a qualidade é incorporada ao software durante o processo de engenharia.

Em outras palavras podemos afirmar que, não se pode simplesmente testar o objeto executável final (produto), etapas de qualidade devem ser estabelecidas e seguidas antes, durante e após a fase de construção.

Organização do teste de software

Do ponto de vista do construtor (desenvolvedor), o teste pode ser considerado (psicologicamente) destrutivo. Assim o construtor segue suavemente, projetando e executando testes que vão demonstrar que o programa funciona, em vez de descobrir erros. Infelizmente erros estarão presentes, aqueles que o engenheiro de software não achar, o cliente achará !

O desenvolvedor de software é sempre responsável por testar as unidades individuais (componentes) do programa garantindo que cada uma realiza a função ou exibe o comportamento para o qual foi projetada.

O papel do grupo independente de teste (Independent Test Group, ITG) é remover os problemas inerentes associados com deixar o programador testar o software que ele construiu. O teste independente remove o conflito de interesses que pode, de outra forma, estar presente.

Após o ITG começar a interagir com o produto, ele irá trabalhar juntamente com o desenvolvedor durante o projeto de software para garantir que testes rigorosos serão conduzidos. Neste ponto o desenvolvedor deverá estar disponível para a correção de eventuais erros encontrados.

Estratégia de teste de software

O processo de engenharia de software, pode ser visto conforme a imagem abaixo:

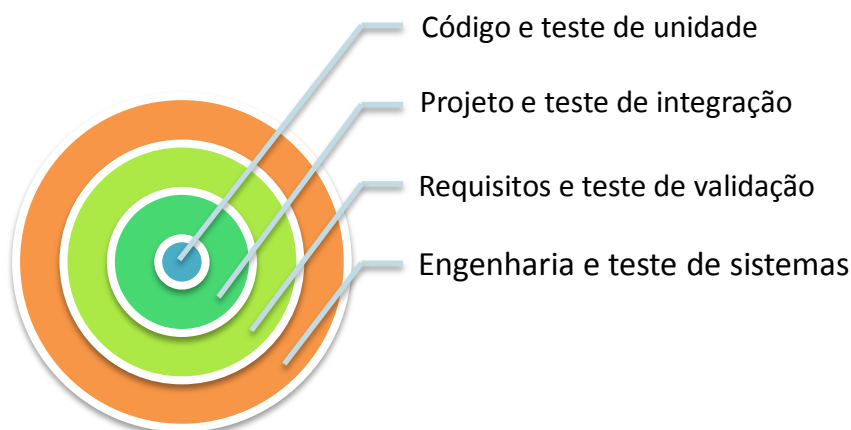


Figura 29 Estratégia de Teste

O processo de desenvolvimento inicia-se pela engenharia de sistemas, passa por análise e engenharia de requisitos, o projeto de software é construído e então inicia-se a codificação. Observe que para cada fase estabelecida existem pontos específicos de teste.

Os testes por sua vez são conduzidos sob um ponto de vista procedural, o teste é na realidade uma série de quatro passos, que são implementados seqüencialmente. Os passos são mostrados na imagem anterior.

Teste de Unidade - Inicialmente o teste focaliza cada componente individualmente, garantindo que ele funciona adequadamente como uma unidade. Este teste faz uso intensivo das técnicas de teste que exercitam caminhos específicos na estrutura de controle de um componente, para garantir completa cobertura e máxima detecção de erros.

Teste de Integração – cuida dos tópicos associados a problemas duais de verificação e construção de programas. Depois que um software tiver sido integrado (construído), um conjunto de testes de alto nível é conduzido.

Teste de Validação – Fornecem garantia final de que o software satisfaz a todos os requisitos funcionais, comportamentais e de desempenho.

Teste de Sistema – O software uma vez validado, deve ser combinado com os outros elementos do sistema (por exemplo, o hardware, o pessoal, os bancos de dados, etc), ele verifica se todos os elementos combinam adequadamente e se a função/desempenho global do sistema é alcançada.

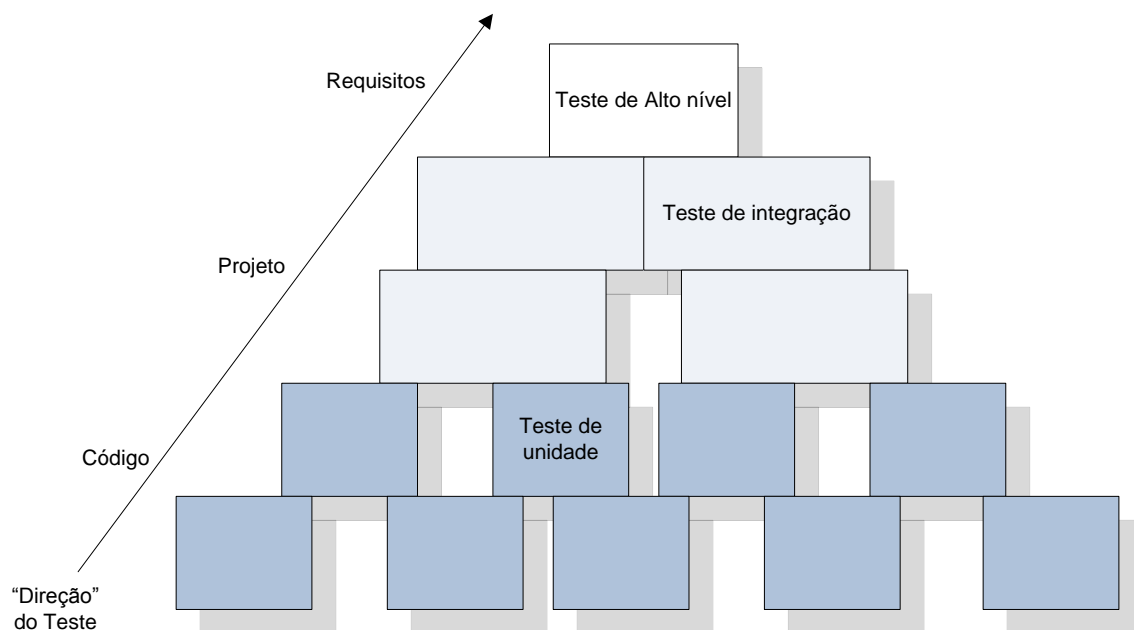


Figura 30 Passos de Teste de Software

Uma questão clássica é levantada toda vez que o teste de software é discutido: “Quando terminarmos o teste como saberemos que testamos o suficiente?”. Infelizmente, não há resposta definitiva para essa questão, mas há algumas respostas pragmáticas e primeiras tentativas de diretrizes experimentais.

Uma resposta é: “A tarefa de teste nunca termina, ela apenas passa de você para o seu cliente”.

Há muitas estratégias que podem ser usadas para teste de software. Em um extremo, uma equipe de software pode esperar até que o sistema esteja totalmente construído e, então conduzir testes no sistema inteiro na esperança de encontrar erros, essa abordagem simplesmente não funciona. Ela vai resultar em um software defeituoso que desaponta o cliente e o usuário final.

No outro extremo, um engenheiro de software pode conduzir testes diariamente, sempre que qualquer parte do sistema seja construída, essa abordagem pode ser muito efetiva.

Teste Unitário

O teste unitário focaliza o esforço de verificação na menor unidade de projeto de software – o componente ou módulo de software. O teste de unidade enfoca a lógica interna de processamento e as estruturas de dados dentro dos limites de um componente. Esse tipo de teste pode ser conduzido em paralelo para diversos componentes.

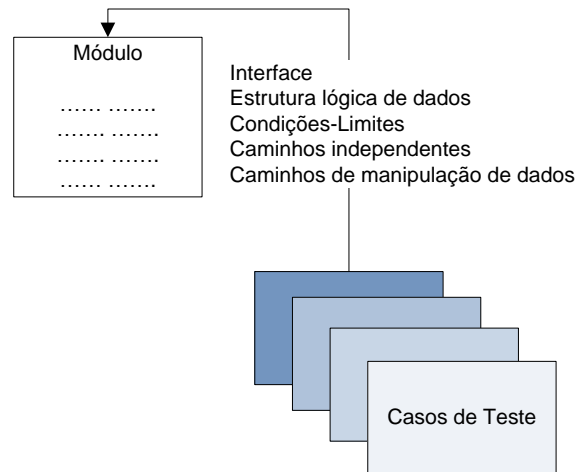


Figura 31 Teste de unidade

O esquema apresentado na imagem acima, apresenta a estrutura básica na condução de testes unitários, que é apresentada (de forma resumida) abaixo.

Interface – A interface do módulo é testada para garantir que a informação flui adequadamente para dentro e para fora da unidade de programa que está sendo testada;

Estrutura lógica de dados – É examinada para garantir que os dados armazenados temporariamente mantenham sua integridade durante todos os passos de execução de um algoritmo;

Condições-limite – São testadas para garantir que o módulo opere adequadamente nos limiares estabelecidos para limitar ou restringir o processamento.

Caminhos independentes e manipulação de dados – Testes de fluxo de dados através da interface de um módulo são necessários, antes que qualquer outro teste seja iniciado. Se os dados não entram e saem adequadamente, todos os outros testes são discutíveis.

O projeto de teste de unidade pode ser iniciado antes que o código seja iniciado (uma abordagem ágil preferida, ver metodologias ágeis) ou depois de o código-fonte ter sido gerado.

Depuração

A depuração ocorre como consequência de teste bem-sucedido. Isto é, quando um caso de teste descobre um erro, a depuração é a ação que resulta na reparação do erro.

A depuração não é teste, mas sempre ocorre como consequência do teste. O processo de depuração começa com a execução de um caso de teste. Os resultados são avaliados e uma falta de correspondência entre a execução esperada e a obtida é encontrada. A depuração tenta relacionar sintoma com causa, levando assim a correção do erro.

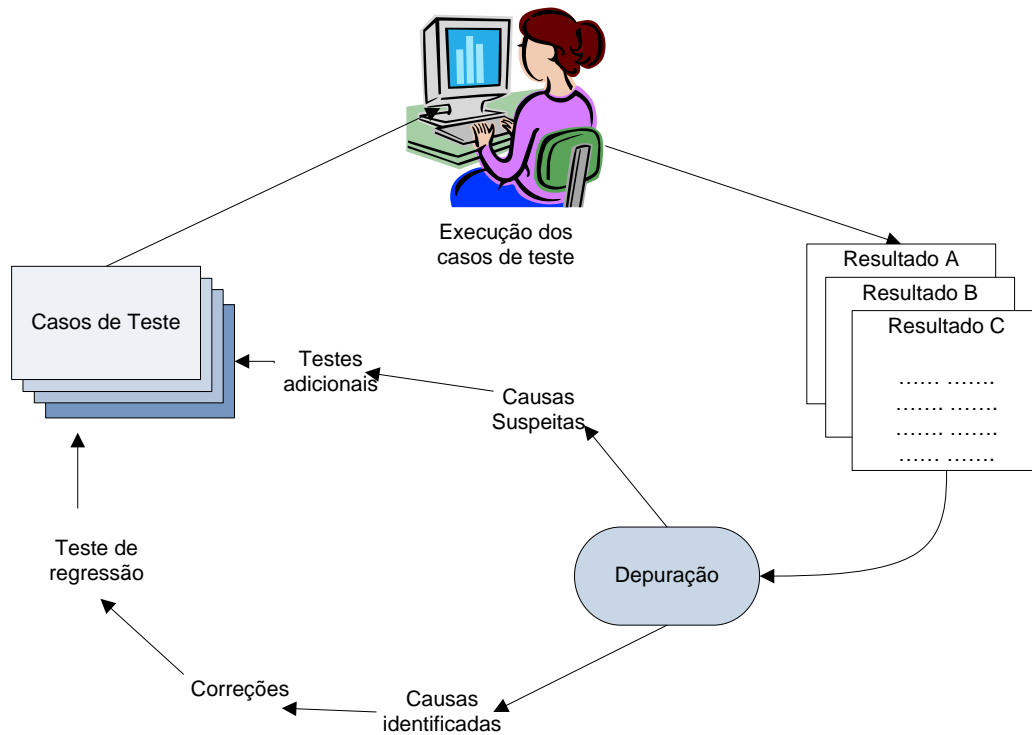


Figura 32 Processo de depuração

O teste de regressão mencionado na figura acima, refere-se a reexecução de testes que já foram conduzidos para garantir que as modificações não propaguem efeitos colaterais indesejáveis, quando alguma alteração ou novo componente for adicionado ao produto.

Leitura e Criação de Diagramas

Na engenharia de software devemos trabalhar com os mais diversos tipos de diagramas e fluxogramas. Entre os mais conhecidos citamos:

- DFD – Diagrama de Fluxo de Dados
- DER – Diagrama Entidade Relacionamento
- UML
 - Diagrama de Casos de Uso
 - Diagrama de Classes
 - Diagrama de Seqüência
 - Diagrama de Colaboração
 - Diagrama de Atividade
 - Diagrama de Objetos
 - Diagrama de Máquina de Estados
 - Diagrama de Componentes
 - Diagrama de Implantação

A base para todos os diagramas utilizados na engenharia de software é o fluxograma. O Fluxograma representa uma seqüência de trabalho qualquer, de forma detalhada (pode ser também sintética), onde as operações ou os responsáveis e os departamentos envolvidos são visualizados nos processos.

Os principais objetivos do fluxograma são:

- Uma padronização na representação dos métodos e os procedimentos administrativos;

- Podem-se descrever com maior rapidez os métodos administrativos;
- Pode facilitar a leitura e o entendimento das rotinas administrativas;
- Podem-se identificar os pontos mais importantes das atividades visualizadas;
- Permite uma maior flexibilização e um melhor grau de análise.

O fluxograma visa o melhor entendimento de determinadas rotinas administrativas, através da demonstração gráfica. Existem estudos que comprovam que o ser humano consegue gravar melhor uma mensagem, quando esta é acompanhada de imagens.

Simbologia

Para a criação de fluxogramas utilizam-se uma série de símbolos (notação) pré-definidos, porém estes não são os únicos símbolos que existem, outros símbolos podem ser criados e agregados aos seus fluxogramas. Abaixo é exibida uma lista dos principais símbolos:



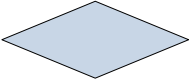




Símbolo	Significado
	Terminal (início ou fim)
	Processo/Operação
	Decisão (Caminho a ser tomado)
	Processo pré-definido
	Documento (indica que como resposta a um processo será criado um documento)
	Referência (para dentro do documento)
	Referência (para outro documento)

Tabela 12 Símbolos do Fluxograma

Obs:

- Estes não são os únicos símbolos existentes;
- Podemos criar nossos próprios símbolos (usa-se muito em peças de divulgação ou apresentações);
- Sugere-se sempre colocar a legenda com o significado dos símbolos usados (nem todos sabem o que significam);
- Pode-se identificar com letras ou números os passos no seu fluxograma. Dessa forma há como relacioná-los a uma explicação textual e detalhada de cada passo.

No exemplo a seguir é exibido um fluxograma básico de um processo de compra em um site de varejo on-line.

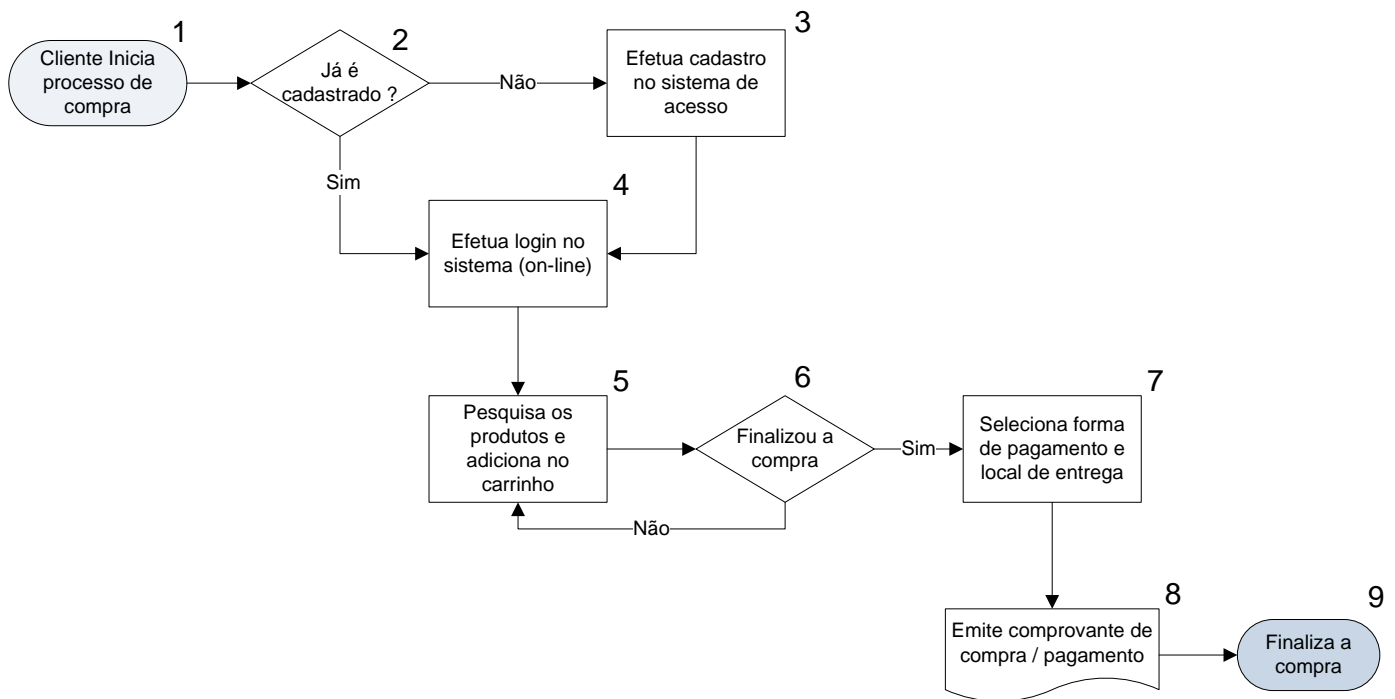


Figura 33 Exemplo de fluxograma

Exercício

Com base nas informações do caso abaixo e utilizando-se apenas os símbolos **terminal**, **processo** e **decisão**, construa um fluxograma básico para o processo de controle de estoque.

A empresa X deseja informatizar o seu controle de estoque. Os pedidos ao estoque ocorrem da seguinte forma: O estoquista ao receber uma solicitação de peça, verifica na listagem do estoque a disponibilidade da mesma. Caso esteja disponível, a peça é entregue ao solicitante e em seguida, é efetuada a baixa no estoque. Caso a peça não esteja disponível, verifica-se junto aos fornecedores de peças*, o tempo de entrega. Informa-se o tempo necessário ao solicitante. Caso este (solicitante) ainda deseje a peça, o pedido ao fornecedor é efetuado imediatamente. Aguarda-se a chegada da peça e a sua entrada no estoque. Em seguida ela é entregue ao solicitante e é efetuada a baixa no estoque.

* Considere que o fornecedor sempre tem a peça solicitada disponível para entrega.

Reengenharia

Durante os anos de 1990, algumas empresas fizeram um esforço legítimo para aplicar a reengenharia e os resultados levaram ao aperfeiçoamento da competitividade. Outras confiaram apenas na redução e na terceirização (em vez de praticar a reengenharia) para melhorar seus resultados.

Considere qualquer produto de tecnologia que tenha servido bem a você. Você o utiliza regularmente, mas ele está ficando velho. Quebra com frequência, leva mais tempo do que gostaria para reparar e deixou de representar a tecnologia mais recente. O que fazer ?

Se estivermos falando de um hardware, provavelmente jogaríamos fora e compraríamos outro mais moderno. Mas quando se trata de software, não é bem assim. Ele precisa ser reconstruído, um produto com funcionalidades adicionais terá que ser criado, com melhor desempenho e confiabilidade e manutenibilidade aperfeiçoada. Isto podemos dizer que é a **reengenharia**.

Vivemos em um mundo que está se modificando constantemente, as demandas nas funções de negócio e na tecnologia de informação que a suportam estão se modificando em um ritmo que coloca enorme pressão competitiva em toda organização comercial. Neste cenário tanto o negócio como o software que da suporte, precisam ser trabalhados pela reengenharia para manter o ritmo.

A reengenharia pode ser dividida em duas atividades básicas, a Reengenharia de Processo de Negócio e a Reengenharia de Software. O objetivo dessas atividades é criar versões dos programas existentes com mais qualidade e melhor manutenibilidade.

Reengenharia de Processo de Negócio

A reengenharia de processo do negócio (Business Process Reengineering, BPR) define as metas do negócio, identifica e avalia os processos de negócio existentes e cria processos de negócio revisados que satisfazem melhor os objetivos atuais.

Reengenharia de Software

O processo de reengenharia de software inclui análise de inventário, reestruturação de documentos, engenharia reversa, reestruturação de programas e dados e engenharia avante.

A ligação entre a reengenharia do negócio e a engenharia de software está em uma visão de sistema. O Software é frequentemente a realização das regras de negócio. À medida que essas regras se modificam, o software também deve ser modificado. A medida que os gestores mudam a forma de conduzir seus negócios, os software que os apóiam também devem ser modificados.

Software não-manutenível não é um problema novo. De fato a ênfase cada vez maior na reengenharia de software tem sido motivada por problemas de manutenção de software que tem crescido em tamanho durante mais de 40 anos.

A manutenção de software existente pode ser responsável por mais de 60 % de todo o esforço despendido por uma organização de desenvolvimento, e a porcentagem continua a crescer à medida que mais software é produzido.

Um problema grave na questão manutenção de software é a rotatividade de pessoal de software. É provável que a equipe (ou pessoa) de software que fez o trabalho original não esteja por perto ou pior, novas gerações de pessoal de software modificaram o sistema e foram embora.

A reengenharia leva tempo, tem um custo significativo em dinheiro e absorve recursos que poderiam, por outro lado, ser usados em necessidades imediatas. Por todas essas razões, a reengenharia não é conseguida em poucos meses ou mesmo em poucos anos.

O paradigma da reengenharia, conforme mostrado na figura abaixo, é um modelo cíclico. Isso significa que cada uma das atividades apresentadas como parte do paradigma pode ser revisitada. Para qualquer ciclo particular, o processo pode terminar depois de qualquer uma dessas atividades.

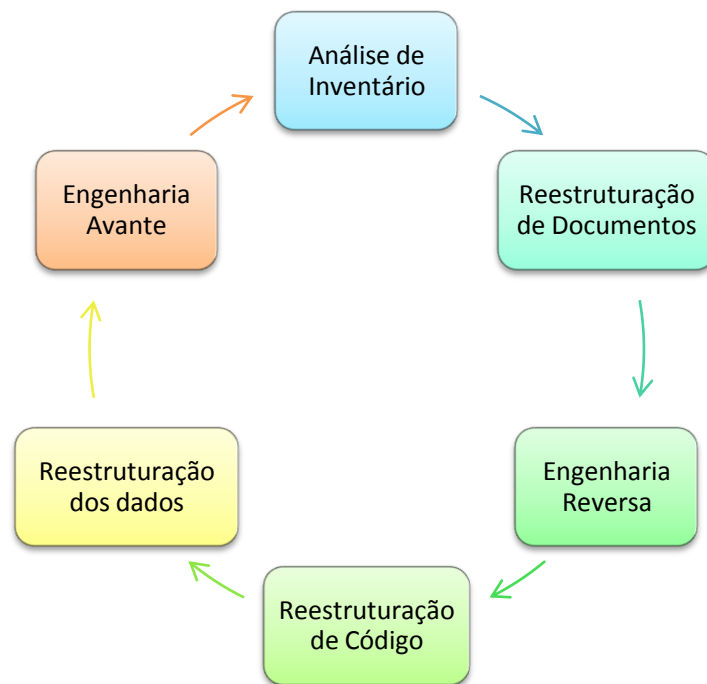


Figura 34 Modelo de Processo de Reengenharia

Análise de Inventário

Toda organização de software deve ter um inventário de todas as aplicações. O inventário pode ser uma planilha que forneça informações como por exemplo, tamanho, idade, criticalidade para o negócio de cada aplicação ativa. Essa informação deve estar ordenada de acordo com a criticalidade para o negócio, longevidade, manutenibilidade corrente e outros critérios localmente importantes.

É necessária uma avaliação regular do inventário, pois o estado de aplicações (por exemplo, criticalidade para o negócio) pode mudar.

Reestruturação de Documentos

Pouca documentação é a marca registrada de muitos sistemas herdados. Mas o que podemos fazer a respeito? Quais são as opções?

- *A criação de documentação consome tempo demais.* Se o sistema funciona, vamos conviver com isso. Em muitos casos essa é a melhor abordagem para esta situação. Não é possível recriar documentação para centenas de programas de computador.

- *A documentação precisa ser atualizada mas temos recursos limitados.* Pode não ser necessário redocumentar totalmente uma aplicação. Em vez disso partes do sistema que estão sendo mantidas são totalmente documentadas.
- *O sistema é crítico para o negócio e precisa ser totalmente redocumentado.* Nesses casos é mais inteligente limitar a documentação ao mínimo essencial.

Engenharia Reversa

O termo engenharia reversa tem sua origem no mundo do hardware. Uma empresa desmonta um produto competidor de hardware em um esforço para entender “os segredos” do projeto de fabricação do concorrente. Para software é bastante semelhante. Na maioria dos casos, no entanto, o programa a passar por engenharia reversa não é o do concorrente, em vez disso é trabalho da própria empresa (frequentemente feito há muitos anos antes).

Os “segredos” a serem entendidos são obscuros, pois nenhuma documentação jamais foi desenvolvida. Assim sendo a engenharia reversa de software é o processo de análise de um programa, em um esforço para representá-lo em uma abstração mais alta do que o código-fonte.

A engenharia reversa é um processo de recuperação de projeto. As ferramentas de engenharia reversa extraem informação do projeto de dados, arquitetural e procedural, para um programa existente.

A engenharia reversa pode extrair informação de projeto do código-fonte, mas o nível de abstração, a completeza da documentação, o grau em que ferramentas e analista trabalham juntos, e a direcionalidade do processo são variáveis.

Um exemplo de engenharia reversa pode ser a geração de um MER – Modelo Entidade Relacionamento, através de um código SQL com as instruções de DDL.

Reestruturação de Código

O tipo mais comum de reengenharia é a reestruturação de código. Alguns programas legados possuem uma arquitetura de programa relativamente sólida, mas módulos individuais foram codificados de um modo que se torna difícil de entendê-los, testá-los e mantê-los, nesses casos o código dos módulos suspeitos pode ser reestruturado.

Reestruturação de Dados

Um programa com arquitetura de dados fraca será difícil de adaptar e aperfeiçoar. De fato, para muitas aplicações, a arquitetura de dados está mais relacionada à viabilidade do programa no longo prazo do que ao código-fonte propriamente dito.

Engenharia Avante

A engenharia avante, também chamada de *renovação* ou *recomposição*, não apenas recupera informação de projeto de software existente, mas usa essa informação para alterar ou reconstituir o sistema existente em um esforço para aperfeiçoar sua qualidade global. Na maioria dos casos, o software trabalhado por reengenharia reimplementa a função do sistema existente e também adiciona novas funções e/ou melhora o desempenho geral.

Referência Bibliográfica

PRESSMANN, Roger S. **Engenharia de Software** São Paulo : Makron Books, 6. Ed. 2006 : McGraw-Hill 2006;

DeMarco, Tom. **Análise Estruturada e Especificação de Sistema**, Rio de Janeiro : Ed. Campos, 1989;

Koscianski, André. **Qualidade de Software** : aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software / André Koscianski, Michel dos Santos Soares. São Paulo : Novatec Editora, 2006;

Oliveira, Jayr Figueiredo de. **Metodologia para Desenvolvimento de Projetos de Sistemas – Guia prático**. 3. Edição revisada em ampliada. São Paulo : Érica, 1999;

Gustafson, David A. **Teoria e problemas de engenharia de software**. Trad. Fernanda Cláudia Alves Campos. Porto Alegre : Bookman, 2003;

Machado, Felipe Nery Rodrigues. **Análise Relacional de Sistemas**. São Paulo: Érica 2001;

Filho, Wilson de Pádua Paula. **Engenharia de Software, Fundamentos, Métodos e Padrões**. Rio de Janeiro: LTC – Livros Técnicos e Científicos Editora S.A. 2003.

ENGENHARIA DE SOFTWARE: revista de engenharia de software. Rio de Janeiro: Grupo Devmedia, 2008.