Módulo 4

Engenharia de Software

Lição 1

Introdução à Engenharia de Software

Versão 1.0 - Jul/2007

Autor
Ma. Rowena C. Solamo

Equipe
Jaqueline Antonio
Naveen Asrani
Doris Chen
Oliver de Guzman
Rommel Feria
John Paul Petines
Sang Shin
Raghavan Srinivas
Matthew Thompson
Daniel Villafuerte

Necessidades para os Exercícios

Sistemas Operacionais Suportados

NetBeans IDE 5.5 para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

Configuração Mínima de Hardware

Nota: IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

Colaboradores que auxiliaram no processo de tradução e revisão

Aécio Júnior
Alexandre Mori
Alexis da Rocha Silva
Allan Souza Nunes
Allan Wojcik da Silva
Anderson Moreira Paiva
Anna Carolina Ferreira da Rocha
Antonio Jose R. Alves Ramos
Aurélio Soares Neto
Bruno da Silva Bonfim
Carlos Fernando Gonçalves
Daniel Noto Paiva
Denis Mitsuo Nakasaki
Fábio Bombonato
Fabrício Ribeiro Brigagão
Francisco das Chagas
Frederico Dubiel
Jacqueline Susann Barbosa
João Vianney Barrozo Costa
Kleberth Bezerra G. dos Santos
Kefreen Ryenz Batista Lacerda
Leonardo Ribas Segala
Lucas Vinícius Bibiano Thomé
Luciana Rocha de Oliveira
Luiz Fernandes de Oliveira Junior
Marco Aurélio Martins Bessa
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauro Cardoso Mortoni
Mauro Regis de Sousa Lima
Paulo Afonso Corrêa
Paulo Oliveira Sampaio Reis
Ronie Dotzlaw
Seire Pareja
Sergio Terzella
Thiago Magela Rodrigues Dias
Vanessa dos Santos Almeida
Wagner Eliezer Rancoletta

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™
Rommel Feria – Criador da Iniciativa JEDI™

1. Objetivos

Quando pessoas pensam sobre computadores, o primeiro detalhe que vêem em suas mentes são as máquinas físicas – monitor, teclado, mouse e CPU. No entanto, o software é o que permite utilizá-los. Um **software de computador** inclui um *conjunto de programas* que executa no interior de um computador, de qualquer tamanho e arquitetura, que estão sendo processados por programas e apresentados aos usuários como cópias *hard* ou *soft*. É construído por engenheiros de software através do emprego de um processo de software que produz produtos de alta qualidade de trabalho que identifica as necessidades das pessoas que irão usar o sistema.

Atualmente, o software é uma importante tecnologia em nossas vidas porque as afeta praticamente em todos os aspectos, incluindo governo, comércio e cultura. Nessa lição, discutiremos sobre engenharia de software como uma disciplina em construção de um software de computador com qualidade. Uma camada de visão será usada para esboçar conceitos necessários à compreensão da engenharia de software. Em seguida, obteremos uma compreensão sobre as pessoas envolvidas no esforço de desenvolvimento de software. Ao final, explicaremos a necessidade de documentação e de como organizar e documentar produtos de trabalho de engenharia de software.

Ao final desta lição, o estudante será capaz de:

- Ter uma visão de camadas sobre a Engenharia de Software
- Obter qualidade dentro do esforço do desenvolvimento
- Conhecer as técnicas e garantias de Qualidade de Software
- Conhecer o processo de software
- Compreender o desenvolvimento de sistemas
- Definir as pessoas envolvidas no esforço do desenvolvimento
- Conhecer as diversas formas e benefícios de documentação

2. Engenharia de Software – Uma visão em camadas

Engenharia de Software é uma disciplina que aplica princípios da engenharia de desenvolvimento na qualidade do software em um determinado tempo e com um custo efetivo. Usando uma abordagem sistemática e metodológica para produzir resultados que possam ser quantificados. Faz uso de medição e métricas para avaliar a qualidade, não somente do software, mas também do processo. Utilizada para avaliar e gerenciar projetos de desenvolvimento de software.

Engenharia de Software é vista de modo diferente pelos diversos profissionais. *Pressman* sugere uma visão da engenharia de software como uma camada tecnológica[1]. Essa visão consiste em quatro camadas: foco na qualidade, processo, método e ferramentas. A Figura 1 ilustra essa visão da engenharia de software.

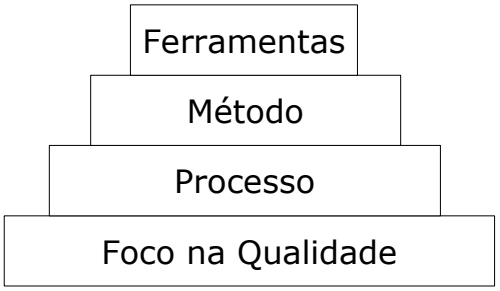


Figura 1: Engenharia de Software – Uma camada de visão

2.1. Foco na Qualidade

Essa camada busca um total **foco na qualidade**. É uma cultura onde o compromisso em melhoria continua no processo de desenvolvimento do software é sustentado. Permite o desenvolvimento de mais abordagens efetivas para engenharia de software.

2.2. Processo

Define uma estrutura, que consiste em áreas de processos chave, que define e permite a entrega racional e a tempo de um *software*. Áreas de processos chave são a base para o gerenciamento de projeto de software. Estabelecem que métodos técnicos sejam aplicados, quais ferramentas são usadas, que produtos de trabalho precisam ser produzidos, e que marcos são definidos. Incluem a garantia que a qualidade será mantida, e que a mudança é devidamente controlada e gerenciada.

2.3. Método

Métodos definem procedimentos sistemáticos e ordenados de construção de software. Eles proporcionam uma estrutura global interna onde as atividades do engenheiro de software são realizadas. Essas atividades incluem um conjunto amplo de tarefas, tais como, análise de requisitos, *design*, construção do programa, teste e manutenção.

Metodologia é a ciência de pensamento sistemático, usando os métodos ou procedimentos para uma disciplina em particular. Existem várias metodologias da engenharia de software que são usadas atualmente. Algumas delas estão enumeradas abaixo:

Metodologias Estruturadas:

- Informações de Engenharia
- Desenvolvimento do Ciclo de Vida do Software/Ciclo de Vida do Projeto
- Metodologia de Desenvolvimento de Aplicação *Rapid*
- Metodologia de Desenvolvimento de Aplicação *Joint*
- Método CASE*

1 Pressman, Roger S., Software Engineering, A Practitioner's Approach, Sixth Edition, (Singapore: McGraw-Hill Internal Edition, 2005), p. 53-54

Metodologias Orientadas a Objeto:

- Método *Booch*
- Método *Coad* e *Yourdon*
- Método *Jacobson*
- Método *Rambaugh*
- Método *Wirfs-Brock*

2.4. Ferramentas

Promovem o suporte aos processos e métodos. Ferramentas CASE (*Computer Aided Software Engineeing*) proporcionam um sistema de suporte ao projeto de desenvolvimento, onde as informações criadas por uma ferramenta podem ser usadas por outras. Podem ser automáticas ou semi-automáticas.

Muitas ferramentas são usadas para desenvolver modelos. Modelos são *patterns* (padrões) de algo que foi criado ou são simplificações. Existem dois modelos que geralmente são desenvolvidos por um engenheiro de software, especialmente, o modelo de sistema e o modelo de software. O **modelo de sistema** é uma representação acessível de um sistema complexo que precisa ser estudado, enquanto o **modelo de software** é chamado de *blueprint* do software que precisa ser construído. Assim como as metodologias, vários modelos de ferramentas são usados para representar sistemas e softwares. Alguns estão descritos abaixo.

Abordagem de Modelos de Ferramentas Estruturada:

- Diagrama de Entidade-Relacionamento
- Diagrama de Fluxo de Dados
- Pseudocódigo
- Fluxograma

Abordagem de Modelo de Ferramenta Orientada a Objeto:

- Linguagem de Modelagem Unificada (UML)

3. Qualidade dentro do Esforço de Desenvolvimento

Conforme mencionado anteriormente, a qualidade é a mente que influencia todo engenheiro de *software*. Focando na qualidade em todas as atividades de engenharia de software, reduz-se custo e melhora-se o tempo de desenvolvimento pela minimização de um novo trabalho de correção. Para proceder dessa forma, um engenheiro de software tem que definir explicitamente que qualidade de software é ter um conjunto de atividades que assegurarão que todo produto de trabalho da engenharia de software exibe alta qualidade, fazer controle de qualidade e atividades garantidas, o uso de métricas para desenvolver estratégias para melhorar o produto de software e o processo.

3.1. O que é qualidade?

Qualidade é a característica total de uma entidade para satisfazer necessidades declaradas e implícitas. Essas características ou atributos têm que ser mensuráveis de modo que possam ser comparados por padrões conhecidos.

3.2. Como definimos qualidade?

Três perspectivas são usadas na compreensão da qualidade, especialmente, olhamos para a qualidade do produto, do processo e no contexto do ambiente de negócios[2].

Qualidade do Produto

Significa coisas diferentes para cada pessoa. É relativo para uma pessoa analisar qualidade. Para os usuários finais, o software tem qualidade se fornecer o que desejam e quando desejam o tempo todo. Também julgam baseados na facilidade de usar e de aprender como usá-lo. Normalmente avaliam e categorizam com base em características externas, tal como, número de falhas por tipo. **Falhas** podem ser categorizadas como: insignificantes, importantes e catastróficas. Para que outros possam desenvolver e manter o software, estes devem ficar de olho nas características internas em vez das externas. Exemplos que incluem erros e falhas encontradas durante as fases de análise de requisitos, design, e codificação são normalmente feitos anteriormente ao carregamento dos produtos para os usuários finais.

Como engenheiros de software, devemos construir modelos baseados em como os requisitos dos usuários externos serão relacionados com os requisitos internos dos desenvolvedores.

Qualidade do Processo

Existem várias tarefas que afetam a qualidade do software. Às vezes, quando uma tarefa falha, a qualidade do software falha. Como engenheiros de softwares, devemos validar a qualidade no processo de desenvolvimento do software. Regras de processo sugerem que pela melhoria do processo de desenvolvimento do software, também há melhora da qualidade do produto resultante. Algumas regras de processo são demonstradas abaixo:

- ***Capability Maturity Model Integration(CMMI)***. Foram formulados pelo *Software Engineering Institute* (SEI). É um processo meta-modelo que é baseado em um conjunto de sistemas e competências da engenharia de software que devem existir dentro de uma organização. Como a mesma atinge diferentes níveis de capacidade e maturidade desses processos de desenvolvimento.
- ***ISO 9000:2000 para Software***. É um padrão genérico, aplicado para qualquer organização que queira melhorar a qualidade global dos produtos, sistemas ou serviços que proporciona.
- ***Software Process Improvement e Capability Determination (SPICE)***. É um padrão que define um conjunto de requisitos para avaliação do processo de software. O objetivo desse padrão é auxiliar organizações a desenvolver uma análise objetiva da eficácia de qualquer processo de software definido.

Qualidade no contexto do ambiente de negócio

2 Pfleeger, Shari Lawrence, Software Engineering Theory and Practice, International Edition, (Singapore: Prentice-Hall, 1999), p. 10-14

Nessa perspectiva, qualidade é visualizada em termos de produtos e serviços sendo proporcionado pelo negócio em que o software é usado. Melhorando a qualidade técnica dos processos de negócio, agrega-se valor ao negócio, por exemplo, valor técnico do software traduz o valor do negócio. Também é importante medir o valor do software em termos de terminologias de negócio, tal como, "quantos pedidos de venda foram processados hoje?", valor do dólar sobre o retorno em cima dos investimentos (ROI), etc. Se o software não agrega valor ao negócio, qual a necessidade de tê-lo em primeiro lugar?

3.3. Como endereçamos os pontos importantes sobre qualidade?

Podemos endereçar os pontos importantes sobre qualidade em:

1. **Uso de padrões de Qualidade**. Padrões de qualidade são um conjunto de princípios, procedimentos, metodologias e regras, para resumir, sobre qualidade no processo, tais como, CMMI, ISO 9000:2000 para Software e SPICE.
2. **Compreender pessoas envolvidas no processo de desenvolvimento incluindo usuários finais e participantes**. Sustenta um ambiente de colaboração e comunicação efetiva.
3. **Compreender as tendências sistemáticas na natureza humana**. Tal como, as pessoas tendem a ser contrárias ao risco quando existe uma perda potencial, são indevidamente otimistas em seus planos e projeções, e preferem usar julgamentos intuitivos ao invés de modelos quantitativos.
4. **Engajamento para a qualidade**. Uma mente focada sobre qualidade é necessária para descobrir erros e defeitos assim que possam ser endereçados imediatamente.
5. **Requisitos de usuários administradores porque mudarão ao longo do tempo**. Requisitos é a base, definindo as características da qualidade de software.

4. Técnicas e Garantias de Qualidade de Software

Garantia de qualidade de Software é um subconjunto da engenharia de software que assegura que todos os produtos de trabalho sejam realizados, e que cumpram com as exigências e padrões estabelecidos pelos usuários. Considera-se como uma das atividades mais importantes que é aplicada durante todo o processo do desenvolvimento do software. O objetivo é detectar defeitos antes do software ser entregue como um produto acabado para o usuário final. Isto abrange uma aproximação eficaz da gerência de qualidade, tecnologia de engenharia de software (métodos e ferramentas), técnicas formais de revisão, várias estratégias de teste, controle de documentação de software e alterações feitas, um procedimento para assegurar a conformidade com os padrões de desenvolvimento de software, e um mecanismo para mensurá-los e documentá-los.

4.1. Qualidade de Software

Um software possui qualidade se ele estiver ajustado para uso, isto é, se estiver trabalhando corretamente. Para que ele trabalhe corretamente, ele deve estar em conformidade com os requisitos funcionais e de performance (características externas dos usuários), padrões explicitamente documentados de desenvolvimento (padrões de qualidade), e características implícitas (características internas aos desenvolvedores) que são esperadas por todo desenvolvimento profissional de software.

Três pontos importantes enfatizados para definir a qualidade do software.

1. Requisitos de Software são a base para a qualidade do software. É necessário explicitar, especificar e priorizar.
2. Padrões definem um de critérios de desenvolvimento que irão mostrar a maneira com a qual o software será desenvolvido.
3. Características implícitas deverão ser identificadas e documentadas; elas influenciam na maneira de como o software será desenvolvido assim como sua manutenibilidade.

4.2. Características para uma Boa Engenharia de Software

Para definir uma boa engenharia de software, dê uma olhada nas características específicas que o software apresenta. Algumas delas estão enumeradas abaixo:

- **Usabilidade**. É a característica do software de apresentar facilidades entre a comunicação dos usuários com o sistema.
- **Portabilidade**. É a capacidade do software ser executado em diferentes plataformas e arquiteturas.
- **Reusabilidade**. É a habilidade do software de se transferir de um sistema para outro.
- **Manutenibilidade**. É a habilidade do software de se envolver e adaptar-se às alterações em um curto espaço de tempo. É caracterizado pela fácil atualização e manutenção.
- **Dependência**. É a característica do software ser confiável e de segurança.
- **Eficiência**. É a capacidade do software utilizar os recursos com maior eficiência.

4.3. Atividades da Garantia de Qualidade de Software

Garantia de Qualidade de Software é composta por uma variedade de atividades com o objetivo de construir software com qualidade. Isto envolve dois grupos de desenvolvedores e a equipe de SQA (*Software Quality Assurance*). A equipe de SQA tem responsabilidade em garantir plenamente à qualidade, supervisionar, manter, analisar e reportar defeitos. As atividades envolvidas são as seguintes:

1. *A equipe de SQA prepara o Plano de SQA*. Isto se dá durante a fase de planejamento de projeto. Identificam-na:

- Avaliação a ser executada;
- Auditorias e revisões a serem executadas;
- Padrões que devem ser aplicados;
- Procedimentos de erros reportados e monitorados;
- Documentos que devem ser produzidos; e
- Conjunto de respostas que se fizer necessário.

2. A equipe de SQA participa na descrição do processo de desenvolvimento de software. O time de desenvolvedores escolhe o processo de desenvolvimento e a equipe de SQA deve verificar se ele se enquadra na política organizacional e nos padrões de qualidade.
3. A equipe de SQA revisa as atividades de engenharia de software empregadas pelo time de desenvolvedores para checar a conformidade com o processo de desenvolvimento de software. Eles monitoram e seguem desvios do processo do desenvolvimento do software. Documentam-no e asseguram-se de que as correções sejam feitas.
4. A equipe de SQA revê o trabalho para verificar se estão conforme o padrão definido. Eles monitoram e marcam defeitos e falhas encontrados em cada trabalho.
5. A equipe de SQA assegura-se que os desvios nas atividades de software e no processo de produção estejam seguramente baseados na definição de procedimentos e padrões de operação.
6. A equipe de SQA envia desvios e desconformidades aos padrões para os gerentes ou a quem for de interesse.

4.4. Técnicas Formais de Revisão

Produtos de trabalho são as saídas esperadas como resultado da execução de tarefas no processo de desenvolvimento de software. Esses resultados contribuem para o desenvolvimento de software com qualidade. Conseqüentemente, devem ser mensurados e verificados novamente se vão ao encontro das exigências e dos padrões. As alterações nos produtos de trabalho são significativas; elas podem ser monitoradas e controladas. A técnica de checar a qualidade dos produtos de trabalho é a técnica formal de revisão. ***Formal Technical Reviews (FTR)*** são executadas em vários pontos do processo do desenvolvimento do software***.*** Ela serve para descobrir erros e defeitos que podem ser eliminados antes do software ser enviado para o usuário final. Especificamente, seus objetivos são:

1. Descobrir erros em funções, na lógica ou na execução para toda a representação do software;
2. Verificar se o software sob a revisão encontra-se de acordo com os requisitos do usuário;
3. Assegurar-se que o software esteja de acordo com os padrões definidos;
4. Conseguir que o software seja desenvolvido de uma maneira uniforme; e
5. Desenvolver projetos mais gerenciáveis.

Um guia geral de condução das técnicas formais de revisão está listado abaixo.

- **Revisar o produto de trabalho e NÃO o desenvolvedor do produto de trabalho**. O objetivo da revisão e descobrir erros e defeitos para melhorar a qualidade do software. O tom da revisão pode ser brando, porém construtivo.
- **Planejar e cumprir a agenda**. Revisões não devem durar mais de duas horas.
- **Minimizar os debates e discussões***.* É inevitável que os problemas sejam levantados e isso não cause efeito nas pessoas. Lembre a todos que não é hora de resolver os problemas que serão apenas documentados, uma outra reunião deve ser agendada para resolvê-los.
- **Indique áreas de problema, mas não às tente resolvê-las***.* Mencione e esclareça áreas de problema. Entretanto, não é hora de resolver problemas, deverão ser resolvidos em uma outra reunião.
- **Tome nota***.* É uma boa prática tomar nota do que foi dito e suas prioridades para que elas

possam ser vistas por outros revisores. Isto ajudará a esclarecer os defeitos e ações a serem tomadas.

- **Mantenha o número dos participantes a um mínimo e insista em preparar-se para a revisão***.* Escrever comentários e observações pelos revisores é uma boa técnica.
- **Forneça uma lista de verificação para o produto de trabalho que é provável ser revista**. A lista de revisão provê uma estrutura que conduza a revisão. Isto também ajuda os revisores a manterem o foco na questão.
- **Programe as revisões como parte do processo de desenvolvimento de software e assegure-se de que os recursos sejam fornecidos para cada revisor***.* Preparação prevê interpretações em uma reunião. Isto também ajuda os revisores a manterem o foco na questão.
- **Sumário da revisão***.* Verifica a eficácia do processo da revisão.

Duas técnicas formais de revisão do produto de trabalho usadas na indústria são *Fagan's Inspection Method* e *Walkthroughs*.

4.4.1. Método de Inspeção de Fagan

Introduzido por *Fagan* em 1976 na IBM. Originalmente foi utilizado para verificar códigos de programas. Entretanto, pode ser estendido para incluir outros produtos de trabalho como técnicas de documentos, modelo de elementos, projetos de códigos e dados etc. Isto é gerenciado por um moderador que é responsável por supervisionar a revisão. Isto requer uma equipe de inspetores designados a verificar se as regras do produto de trabalho vão de encontro à lista de interesse preparada. É mais formal que o *walkthrough*. A seguir estão descritas regras determinadas na qual cada participante deverá aderir:

- As inspeções são realizadas em um número de pontos no processo do planejamento do projeto e do desenvolvimento dos sistemas.
- Todas as classes com defeito são documentadas e os produtos do trabalho são inspecionados não somente a nível lógico, de especificações ou de funções de erros.
- A inspeção é realizada por colegas em todos os níveis exceto o chefe.
- As inspeções são realizadas em uma lista prescrita das atividades.
- As reuniões de inspeção são limitadas a duas horas.
- As inspeções são conduzidas por um moderador treinado.
- Inspetores são designados a especificar regras para aumentar a eficácia. As listas de verificação dos questionários a serem perguntados pelos inspetores são usadas para definir tarefas e estimular a encontrar defeitos. Os materiais são inspecionados minuciosamente para que seja encontrado o máximo número de possíveis erros.
- Estatísticas com os tipos de erros são vitais, são utilizadas para obter análises de uma maneira similar à análise financeira.

Conduzir inspeções requer muitas atividades. Elas estão categorizadas a seguir:

- **Planejamento**. O moderador deve se preparar para a inspeção. Decide quem serão os inspetores e as regras que estes devem obedecer, quem e quando desempenharão seus papéis e distribuir a documentação necessária.
- **Uma rápida apresentação**. 30 minutos de apresentação do projeto dos inspetores é o suficiente. Isto pode ser omitido se todos estiverem bem familiarizados com o projeto.
- **Preparando**. Cada inspetor terá de 1 a 2 horas sozinho para inspecionar o produto de trabalho. Ele irá executar as regras passadas a ele com base na documentação provida pelo moderador. Ele irá tentar descobrir defeitos no produto de trabalho. Ele não deverá reparar defeitos ou criticar o desenvolvedor do produto de trabalho.
- **Realizando a reunião***.* Os participantes das reuniões são inspetores, moderadores e desenvolvedores do produto de trabalho. Os desenvolvedores do produto de trabalho estão presentes para explicar o produto de trabalho, e responder às perguntas que os inspetores fizerem. Nenhuma discussão se o defeito é ou não real é permitida. Uma lista de defeitos deve ser produzida pelo moderador.

- **Refazendo o produto de trabalho***.* A lista de defeitos deve ser atribuída a uma pessoa para repará-la. Normalmente, esta é o desenvolvedor do produto de trabalho.
- **Acompanhando os reajustes***.* O moderador assegura-se que os defeitos nos produtos de trabalho sejam endereçados e solucionados. Mais tarde este deve ser inspecionado por outro inspetor.
- **Realizando uma reunião ocasional de análise***.* Isto é opcional, momento onde é dada a possibilidade aos inspetores de expressarem sua visão pessoal sobre erros e melhorias. A ênfase é dada à maneira que a inspeção foi feita.

4.4.2. Walkthrough

O *walkthrough* é menos formal que a inspeção. Aqui, o produto de trabalho e sua documentação correspondente são entregues para um time de revisores, normalmente em torno de 3 pessoas, onde comentários de sua exatidão são apresentados. Ao contrário da inspeção onde um é o moderador, o desenvolvedor do produto de trabalho coordena o *walkthrough*. Um escrivão também deve estar presente para documentar a lista de ações. Uma lista de ações deve ser feita a fim de melhorar a qualidade do produto final a qual inclui ajustes dos defeitos, resoluções dos problemas etc.

Alguns passos devem ser seguidos para obter sucesso no *walkthrough*. Eles estão listados abaixo:

- Nenhum gerente deve estar presente.
- Enfatizar que o *walkthrough* é para detecção de erros e não para correção.
- Manter o interesse do grupo.
- Nenhuma contagem ou atribuição de nota.
- Criticar o produto; não a pessoa.
- Sempre documentar a lista de ações.

Conduzir o *walkthrough*, é similar à inspeção, requer muitas atividades. Elas estão categorizadas como se segue:

- Antes do *walkthrough*
  - O desenvolvedor do produto de trabalho agenda o *walkthrough*, preferivelmente, com um dia de antecedência ou dois no máximo.
  - Distribuir o material necessário para o produto de trabalho dos revisores.
  - Pede-se especificamente que cada revisor traga pelo menos dois comentários positivos do *walkthrough* e um comentário negativo sobre o produto do trabalho.
- Durante o *walkthrough*
  - O desenvolvedor do produto de trabalho faz uma rápida apresentação do seu produto de trabalho. Este passo pode ser ignorado caso os revisores conheçam bem o produto de trabalho.
  - Solicitar comentários aos revisores. Às vezes, problemas são levantados e apresentados, mas não devem ser solucionados durante o *walkthrough*. Os problemas deverão ser incluídos em uma lista de ações.
  - Uma lista de ações deve ser produzida até o fim do *walkthrough*.
- Após o *walkthrough*
  - O desenvolvedor do produto de trabalho recebe a lista de ações.
  - Pede-se para enviar os estados das ações com o objetivo de resolver erros ou discrepâncias apresentadas na lista de ações.
  - Possivelmente, um outro *walkthrough* deve ser agendado.

5. O Processo de Software

O ***processo de software*** provê uma estratégia que as equipes de desenvolvimento empregam para construir software com qualidade. Ele é escolhido baseado na maturidade do projeto e aplicação, métodos e ferramentas utilizadas e gerência e produtos que são exigidos. *Pressman* fornece uma representação gráfica do processo de software. De acordo com ele, esta representação fornece a estrutura que um plano compreensivo de desenvolvimento de software pode estabelecer. Consiste em **estrutura de atividades**, **conjunto de tarefas** e **atividades guarda-chuva**[3].

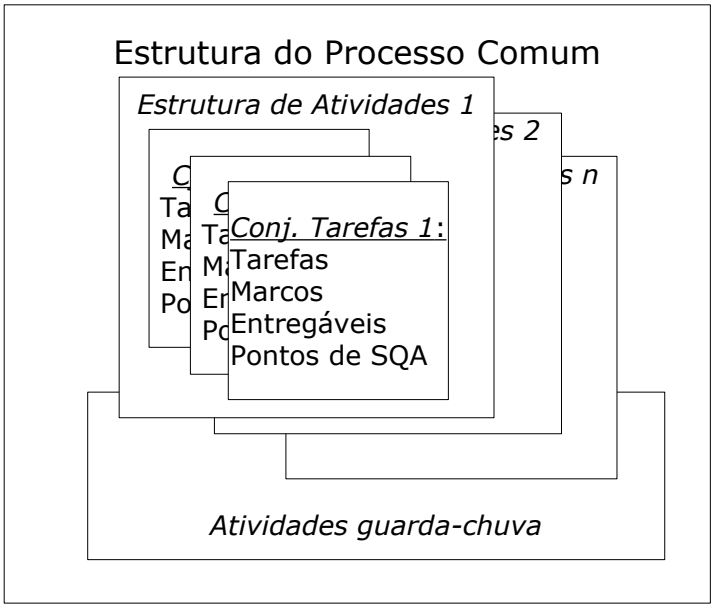


Figura 2: Processo de Software Pressman

Estrutura de Atividades

Estas atividades são executadas por pessoas envolvidas no processo de desenvolvimento aplicado a qualquer projeto de software independente do tamanho, composição da equipe ou complexidade do problema. Também são conhecidos como fases do processo de desenvolvimento de software.

Conjunto de Tarefas

Cada uma das atividades na estrutura do processo define um conjunto de tarefas. Estas tarefas devem ter marcos, produtos que podem ser entregues ou produtos finais e pontos de garantia de qualidade de software (Pontos de SQA). Eles são modificados e ajustados a uma característica específica do projeto de software e dos requisitos de software.

Atividades Guarda-chuva

Estas são atividades que dão suporte às atividades estruturais ao longo do progresso do projeto de software como a gerência de projeto, gerência de configuração, gerência de requisitos, gerência de riscos, revisão técnica formal, etc.

5.1. Tipos de Modelos de Processo de Software

Existem vários tipos de modelos de processo de software. Os mais comuns são discutidos nesta seção.

Modelo Linear Seqüencial

Também conhecido como **modelo cascata** ou **ciclo de vida clássico**. Este é o primeiro modelo formalizado. Outros modelos de processo são baseados em sua abordagem de desenvolvimento

3 Pressman, Software Engineering A Practitioner's Approach, p. 54-55

de software. Ele sugere uma abordagem sistemática e seqüencial para o desenvolvimento de software. Ele inicia pela análise do sistema, progredindo para a análise do software, projeto, codificação, teste e manutenção. Ele prega que uma fase não pode iniciar enquanto a anterior não seja finalizada. A Figura 3 mostra este tipo de modelo de processo de software.

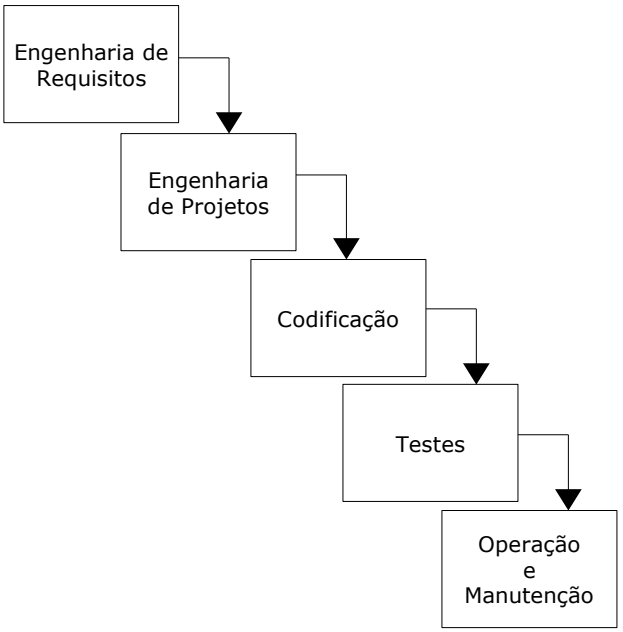


Figura 3: Modelo Linear Seqüencial

As vantagens deste modelo são:

- É o primeiro modelo de software formulado.
- Fornece a base para outros modelos de processo de software.

As desvantagens deste modelo são:

- Projetos de software reais dificilmente seguem restritamente um fluxo seqüencial. De fato, é muito difícil decidir quando uma fase termina e a outra começa.
- O envolvimento do usuário final só ocorre no início (engenharia de requisitos) e no final (operação e manutenção). Ele não mapeia o fato dos requisitos mudarem durante o projeto de desenvolvimento de software.
- Às vezes os usuários finais têm dificuldades de estabelecer todos os requisitos no início. Logo, isto atrasa o desenvolvimento do software.

Modelo de Prototipação

Para auxiliar o entendimento dos requisitos dos usuários, protótipos são construídos. **Protótipos** são softwares desenvolvidos parcialmente para permitir aos usuários e desenvolvedores examinarem características do sistema proposto e decidir se devem ser incluídos no produto final. Esta abordagem é mais bem adaptada nas seguintes situações:

- Os clientes definem um conjunto de objetivos gerais para o software, mas não identificam detalhes de entrada, processamento ou requisitos de saída.
- O desenvolvedor não tem certeza da eficiência do algoritmo, da adaptabilidade da tecnologia ou a forma que a interação homem-computador deve ser.

A Figura 4 mostra este modelo de processo.

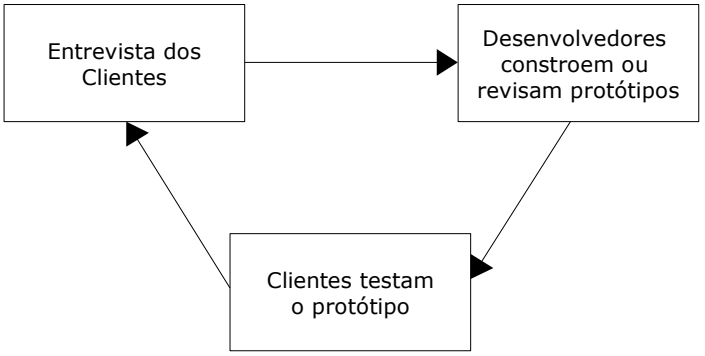


Figura 4: Modelo de Prototipação

A vantagem deste modelo de processo é:

- Os usuários têm participação ativa na definição dos requisitos de interação homem-computador do sistema. Vêem a aparência atual do software.

As desvantagens deste modelo de processo são:

- Os clientes podem erroneamente aceitar o protótipo como uma versão funcional. Isto compromete a qualidade do software porque outros requisitos de software não são alvos de manutenção.
- Os desenvolvedores ficam tentados a implementar um protótipo funcional sem pensar em futuras expansões ou manutenções.

Modelo de Desenvolvimento Rápido de Aplicações (RAD)

Este é um processo de desenvolvimento de software linear seqüencial que enfatiza um ciclo de desenvolvimento extremamente curto. Baseia-se em uma abordagem de construção modular. É melhor utilizado em projetos de software onde os requisitos já estão bem entendidos, o escopo do projeto já está bem definido e uma grande carteira com recursos disponíveis. Todos esperam comprometer-se com a abordagem de desenvolvimento rápido.

Neste modelo de processo, o projeto de software é definido baseado na decomposição funcional do software. Partições funcionais são associadas a diferentes equipes e são desenvolvidas em paralelo. A Figura 5 mostra este modelo de processo.

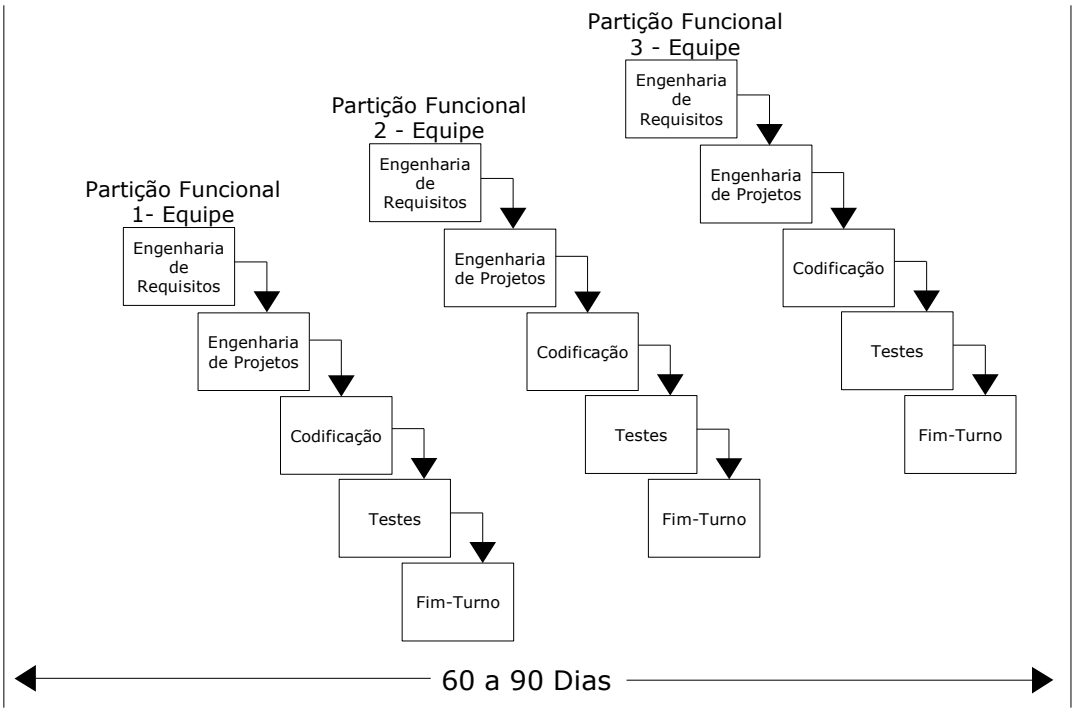


Figura 5: Desenvolvimento Rápido de Aplicações

A vantagem deste modelo é:

- Um sistema totalmente funcional é criado em um curto espaço de tempo.

As desvantagens deste modelo são:

- Para um projeto de larga escala este processo requer um número suficiente de desenvolvedores para formar o número certo de equipes.
- Desenvolvedores e clientes devem se comprometer com as atividades relâmpagos necessárias para desenvolver um software em um curto espaço de tempo.
- Não é um bom modelo de processo para sistemas que não são modularizáveis.
- Não é um bom modelo de processo para sistemas que requerem alta performance.
- Não é um bom modelo de processo para sistemas que fazem uso de novas tecnologias ou têm alto grau de interoperabilidade com programas já existentes como os sistemas legados.

Modelo de Processo Evolucionário

Este modelo de processo reconhece que um software envolve mais de um período de tempo. Ele permite o desenvolvimento de uma versão mais crescentemente complexa de software. A abordagem é naturalmente iterativa. Alguns modelos de processo evolucionário específicos são: **Modelo Incremental**, **Modelo Espiral** e **Modelo Baseado em Montagem de Componentes**.

Modelo Incremental

Este modelo de processo combina os elementos do modelo linear seqüencial com a interatividade do modelo de prototipação. As seqüências lineares são definidas de maneira que cada seqüência produza um ***incremento*** de software. Diferente da prototipação, o incremento é um produto operacional. A Figura 6 mostra este modelo de processo.

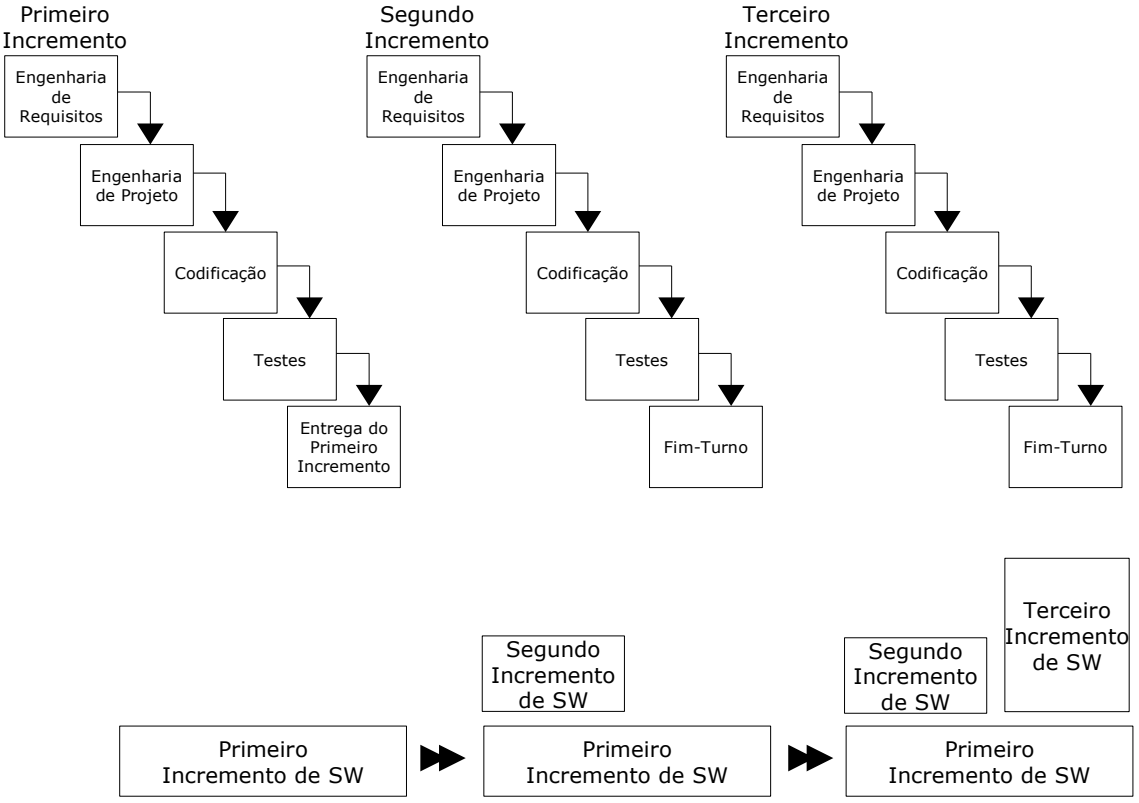
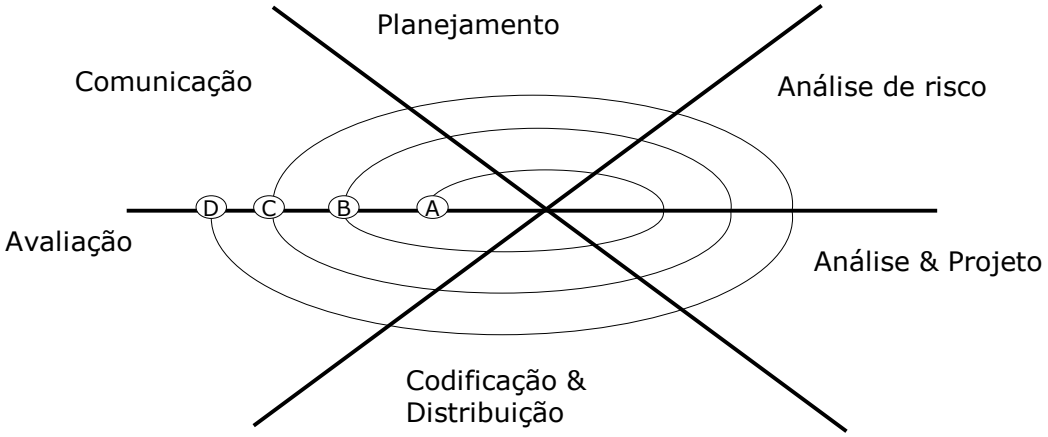


Figura 6: Modelo de Processo Incremental

Modelo Espiral

Foi originalmente proposto por *Boehm*. É um modelo de processo de software evolucionário

acoplado à naturalidade iterativa da prototipação com o controle e os aspectos sistemáticos do modelo linear seqüencial. Ele disponibiliza potencial para desenvolvimento rápido de versões incrementais do software. Uma importante característica deste modelo é que sua análise de risco faz parte da sua estrutura de atividades. Porém, isto requer experiência em estimativa de riscos. A Figura 7 mostra um exemplo do modelo espiral.

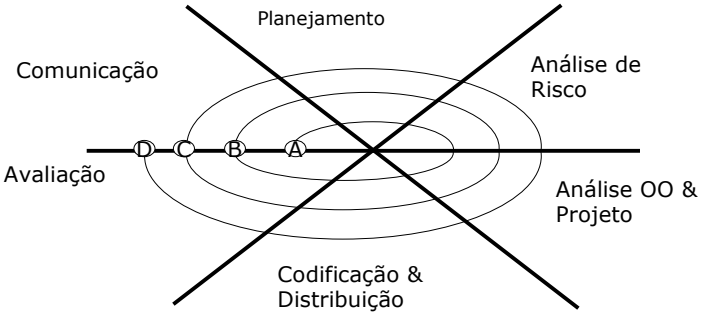


A. Projeto Inicial do Software
B. Manutenção do Novo Software
C. Melhoramento do Software
D. Desenvolvimento de outro sistema inter-relacionado

Figura 7: Modelo Espiral

Modelo de Montagem Baseada em Componentes

É similar ao Modelo de Processo Espiral. No entanto, este usa a tecnologia de objetos onde a ênfase é a criação das classes que encapsulam dados e métodos utilizados para manipular os dados. A reusabilidade é uma das características de qualidade que sempre é checado durante o desenvolvimento do software. A Figura 8 mostra o Modelo de Montagem Baseada em Componentes.



A. Projeto Inicial do Software
B. Manutenção do Novo Software
C. Melhoramento do Software
D. Desenvolvimento de outro sistema inter-relacionado

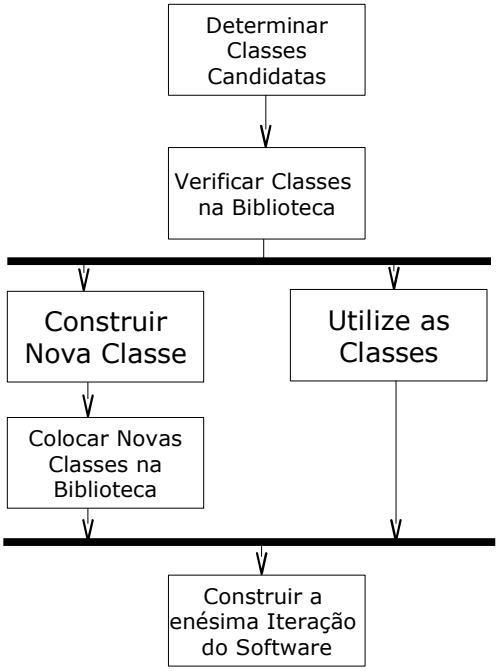


Figura 8: Modelo de Montagem Baseada em Componentes

Modelo de Desenvolvimento Concorrente

O Modelo de Desenvolvimento Concorrente é também conhecido como ***engenharia concorrente***. Ele utiliza os diagramas de estado para representar a relação de concorrência entre tarefas associadas na estrutura de atividades. Ele é representado esquematicamente por uma série de tarefas técnicas maiores e estados associados. A necessidade do usuário, a decisão gerencial e a revisão de resultados dirigem a progressão geral do desenvolvimento. A Figura 9 mostra o modelo de desenvolvimento concorrente.

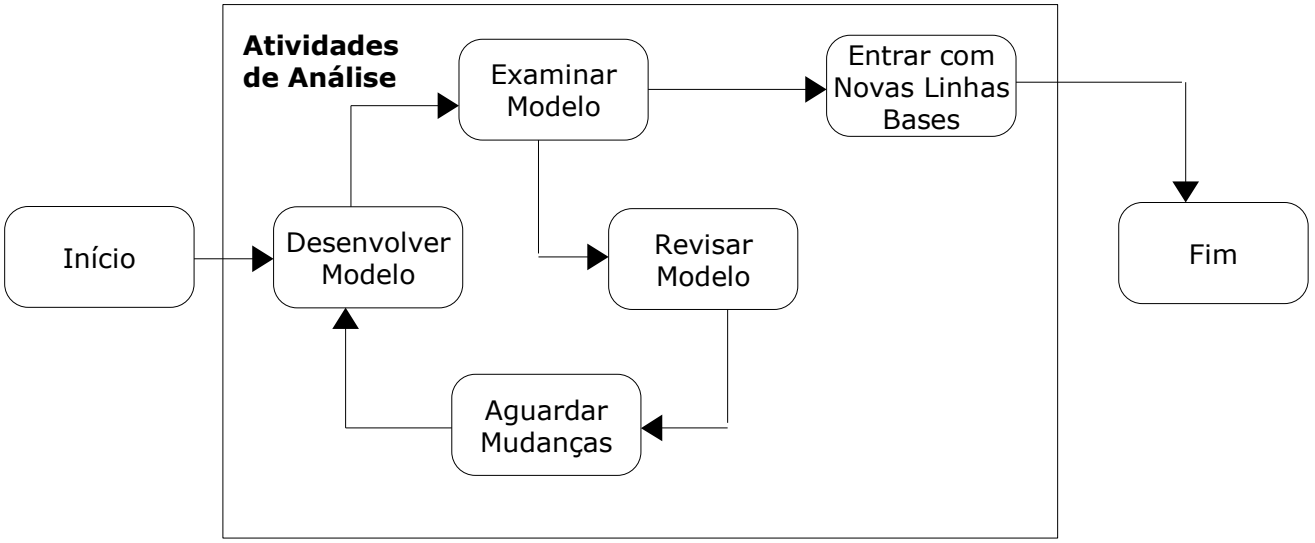


Figura 9: Modelo de Desenvolvimento Concorrente

Métodos Formais

É uma abordagem da engenharia de software que incorpora um conjunto de atividades guiadas por uma especificação matemática do software. Eles fornecem um mecanismo para remover vários problemas que dificultariam o uso de outros paradigmas de engenharia de software. Serve como um meio de verificar, descobrir e corrigir erros que de outra forma não seriam detectados.

5.2. Fatores que Afetam a Escolha do Modelo de Processo

- Tipo do Projeto
- Métodos e Ferramentas a serem utilizadas
- Requisitos e Pessoas-Chaves do Negócio
- Senso Comum e Julgamento

6. Entendendo Sistemas

O projeto de software que necessita ser desenvolvido revolve em torno dos sistemas. Os sistemas consistem em um grupo de entidades ou componentes, Juntos interagem para dar forma aos inter-relacionamentos específicos, organizados por meio da estrutura, e trabalhando para conseguir um objetivo comum. Compreender sistemas fornece um contexto para todo o projeto com a definição dos limites dos projetos. Surge a pergunta, "O que é incluído no projeto? O que não é?" Em definição de limites do sistema, um Engenheiro de Software descobre o seguinte:

- As entidades ou o grupo das entidades que são relacionadas e organizadas de alguma maneira dentro do sistema fornecem a entrada, realizam atividades ou recebem a saída;
- Atividades ou ações que devem ser executadas pelas entidades ou pelo grupo das entidades a fim conseguir a finalidade do sistema;
- Uma lista de entrada; e
- Uma lista de saída.

Como exemplo, a Figura 10 mostra os limites do sistema do estudo de caso. Mostra elementos deste sistema com o uso do diagrama do contexto.

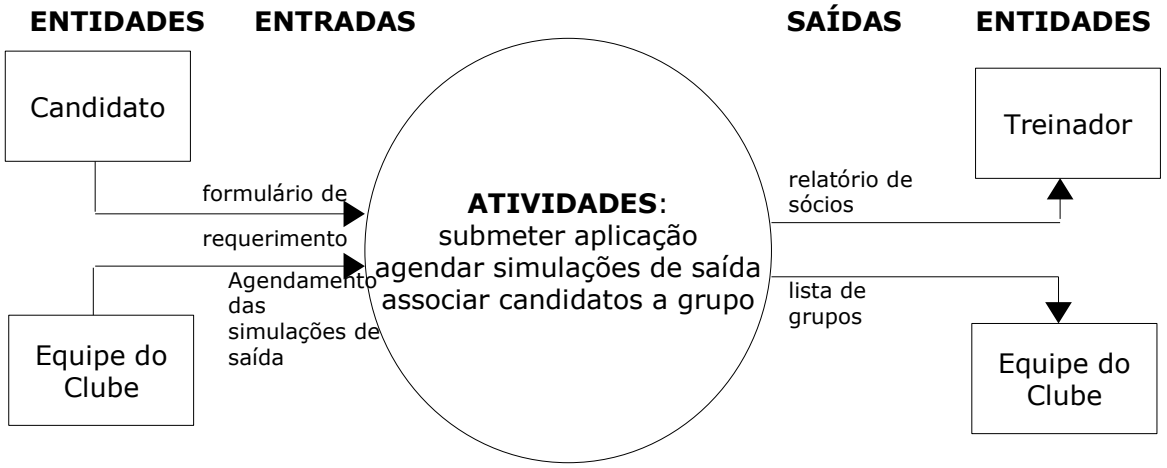


Figura 10: Limites do sistema de aplicação da sociedade do clube

As entidades que são envolvidas neste sistema são os candidatos, a equipe de funcionários do clube e o treinador. São representadas como caixas retangulares. São relacionadas umas com as outras executando determinadas atividades dentro deste sistema. As principais atividades executadas são o envio dos formulários de aplicação, programar as simulações de saídas e a atribuição do pretendente a um grupo. São representados por um círculo no meio que define a funcionalidade da informação mantendo a sociedade do clube. Para executar estas ações, uma lista das entradas é necessária. Especificamente, formulários de aplicação e a programação das simulações de saídas. São representados por uma seta com o nome dos dados que estão sendo passados. A ponta da seta indica o fluxo dos dados. Os resultados mais importantes que se espera deste sistema são os relatórios da sociedade e as listas de grupos. Outra vez, são representados por uma seta com o nome dos dados que estão sendo passados. A ponta da seta indica o fluxo dos dados. O objetivo deste sistema é segurar a aplicação da sociedade do clube.

Princípios Gerais de Sistemas

Alguns princípios gerais dos sistemas são discutidos abaixo. Isto ajudará ao Engenheiro de Software a estudar o sistema onde o projeto do software revolve.

- **Mais especializado um sistema, menos capaz é de adaptar-se a circunstâncias diferentes**. As mudanças teriam um impacto grande no desenvolvimento de tais sistemas. Cada um deve ser cuidadosamente certificado que não há nenhuma mudança dramática no ambiente ou nas exigências quando o software está sendo desenvolvido. As partes interessadas e os colaboradores devem estar cientes dos riscos e dos custos das mudanças

durante o desenvolvimento do software.

- **Maior o sistema é, mais os recursos devem ser devotados à sua manutenção diária**. Como um exemplo, o custo de manter um mainframe é muito caro comparado a manter diversos computadores pessoais.
- **Os sistemas são sempre parte de sistemas maiores, e podem sempre ser divididos em sistemas menores**. Este é o princípio o mais importante que um Engenheiro de Software deve compreender. Porque os sistemas são compostos da versão menor do subsistema e do vice, os sistemas de software podem ser desenvolvidos em uma maneira modular. É importante determinar os limites dos sistemas e de suas interações de modo que o impacto de seu desenvolvimento seja mínimo e possa ser controlado.

Componentes de um Sistema Automatizado

Há dois tipos de sistemas, a saber, sistemas sintéticos e sistemas automatizados. Os sistemas sintéticos são considerados também sistemas manuais. Não são perfeitos. Terão sempre áreas para aumentar a exatidão e melhorias. Estas áreas para a exatidão e as melhorias podem ser direcionadas por sistemas automatizados.

Os sistemas automatizados são exemplos dos sistemas. Consiste em componentes que suporta a operação de um sistema domínio-específico. No geral, consiste no seguinte:

1. ***Hardware***. Esse componente é o dispositivo físico.
2. ***Software***. Esse componente é o programa executado pela máquina.
3. **Pessoal**. Esse componente é responsável pelo uso do hardware e software. Provêem dados como entrada, e interpretam a saída (informação) para as decisões diárias.
4. ***Processos***. Este componente representa a política e os procedimentos que governam a operação do sistema automatizado.
5. ***Dados e Informação***. Este componente fornece a entrada (dados) e a saída (informação).
6. **Conectividade**. Este componente permite a conexão de um sistema computadorizado com um outro sistema. Conhecido também como componente de rede.

A Figura 11 mostra a relação dos cinco primeiros componentes.

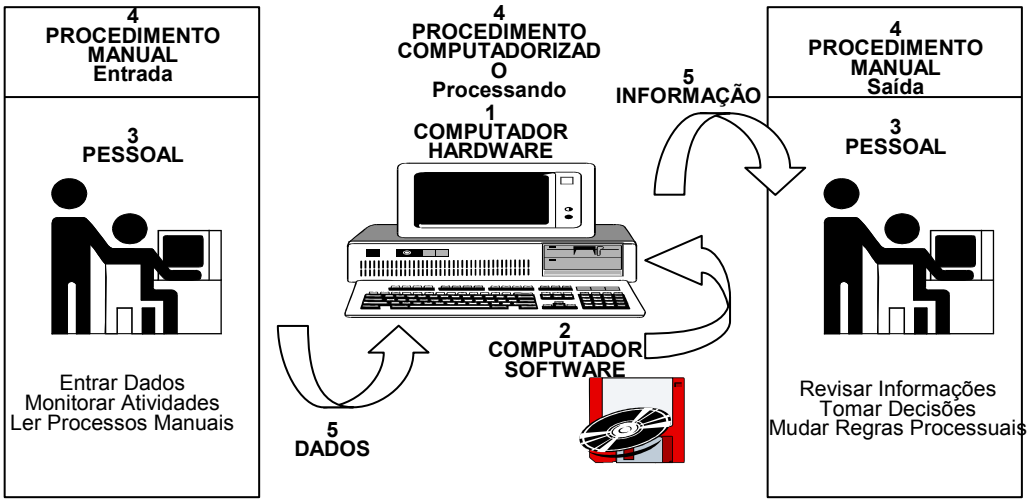


Figura 11: Componentes de um Sistema Automatizado

A Figura 12 representa uma ilustração de um domínio-específico da aplicação de um sistema automatizado. Demonstra o sistema automatizado que processa a aplicação da sociedade do clube.

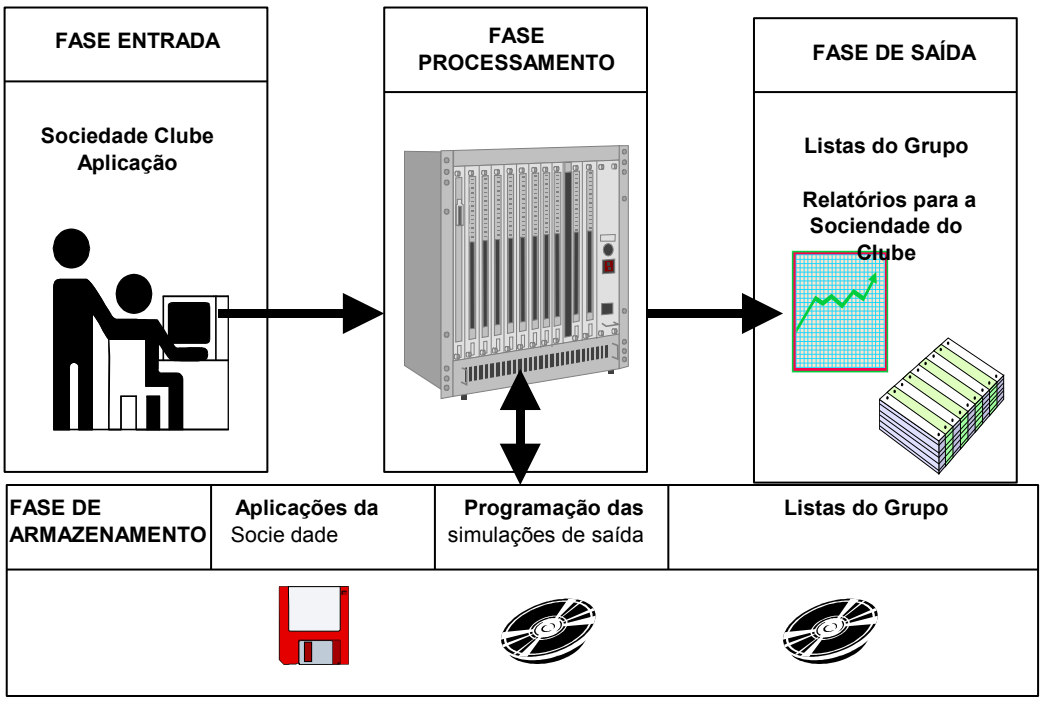


Figura 12: Sistema computadorizado da aplicação da sociedade do clube

A informação da aplicação da sociedade do clube é incorporada ao sistema. O pretendente é programado para a simulação de tentativa de saída. Esta programação é recuperada do armazenamento. Uma vez que um pretendente é atribuído a um grupo, esta informação está incorporada no sistema de modo que o relatório da lista do grupo seja produzido.

7. Entendendo as Pessoas no Esforço de Desenvolvimento

Para ajudar na promoção da idéia comum de qualidade na equipe de desenvolvimento, devemos entender as pessoas envolvidas no processo, particularmente, seus interesses a respeito do sistema e o software que precisa ser desenvolvido. Nesta seção, há dois grupos principais envolvidos no esforço de desenvolvimento de software, especificamente, usuários finais e equipe de desenvolvimento.

7.1. Usuários finais

Usuários finais são pessoas que irão utilizar o produto final. Muitos dos requisitos virão deste grupo. Eles podem ser divididos em dois grupos de acordo com o envolvimento dentro da organização e desenvolvimento, a saber: aqueles envolvidos diretamente e aqueles envolvidos indiretamente.

Aqueles que estão diretamente envolvidos

A Tabela 1 mostra a categorização dos usuários finais de acordo com as funções que eles executam no sistema.

Operacionais	*Supervisores*	*Executivos*
Executam as funções operacionais do sistema.	Executam ações de supervisão sobre as operações diárias do sistema. Eles são em geral avaliados e motivados pela performance de seus orçamentos.	Normalmente possuem iniciativa e servem como financiadores do projeto de desenvolvimento do sistema.
São normalmente mais concentrados com o componente de interface humana do sistema: • Que tipo de teclado irão utilizar? • Que tipo de tela de apresentação o sistema terá? • Haverá muito brilho e haverá caracteres de fácil leitura?	São normalmente mais concentrados na eficiência operacional das funções que precisam ser executadas como a quantidade de saídas em menor tempo.	São menos concentrados nas operações do dia-a-dia. São mais concentrados nas estratégias de lucros e prejuízos a longo prazo.
Têm uma visão local do sistema.	Também tendem a ter a mesma visão local e física do sistema similar aos usuários operacionais, mas com à atenção voltada a performance.	Estão mais interessados na visão global do sistema.
Tendem a pensar no sistema em termos físicos.	São os que têm mais contato com os engenheiros de software.	Geralmente trabalham com modelos abstratos do sistema assim como termos físicos. Possuem maiores interesse em resultados.

Tabela 1: Categorias de Trabalho

Guias Gerais e os Usuários Finais

- O mais alto nível de gerência provavelmente pouco se importa com a tecnologia de computadores. Seria melhor perguntar-lhes sobre o resultado global e performance que os sistemas fornecem. São bons candidatos para entrevistas a respeito de leiautes de relatórios e design de código.
- Os objetivos e prioridades da gerência podem entrar em conflito com os dos supervisores e usuários operacionais. Isto pode ser percebido baseado nos diferentes níveis de preocupações. Como engenheiro de software, tenta-se descobrir áreas comuns. Mais sobre o assunto pode ser visto no capítulo 3 – Engenharia de Requisitos.

- Gerentes podem não fornecer recursos, fundos ou tempo que usuários precisam para construir um sistema efetivo. Restrições financeiras e de recurso irão ocorrer. É importante priorizar os requisitos. Mais sobre isto pode ser visto no capítulo 3 – Engenharia de Requisitos.

Aqueles que estão indiretamente envolvidos

Freqüentemente, este grupo inclui auditores, portadores comuns e grupo de garantia de qualidade. O objetivo geral deste grupo é certificar que o sistema está sendo desenvolvido de acordo com uma variedade de definições como:

- Padrões contábeis desenvolvidas pelas operações de contas da organização ou firma.
- Padrões desenvolvidos por outros departamentos dentro da organização ou pelo cliente ou pelo usuário que irá herdar o sistema
- Vários padrões impostos pelas agências governamentais reguladoras.

Alguns possíveis problemas que podemos encontrar com este grupo. Como engenheiro de software, manter atenção sobre eles e mapeá-los em conformidade.

- Eles não se envolvem no projeto até que se atinja o fim, em particular, o grupo de garantia de qualidade. É importante que eles envolvam-se em todas as atividades que se exija sua experiência e opinião.
- Eles fornecem notação necessária e forma do documento. Eles podem precisar de definição para apresentação e documentação do sistema.
- Eles estão mais interessados em substância do que em formulários.

7.2. Equipe de Desenvolvimento

A equipe de desenvolvimento é responsável pela construção do software que irá dar suporte ao sistema de domínio-específico. Consistem em: analista de sistemas, projetista, programadores e testadores.

Analista de Sistemas

Sua responsabilidade é entender o sistema. Dentro do sistema, ele identifica o que o cliente quer e documenta e prioriza os requisitos. Isto envolve detalhamento do sistema para determinar requisitos específicos que serão a base do projeto de software.

Projetista

Seu trabalho é transformar uma tecnologia livre de arquitetura em uma estrutura interna para que os programadores possam trabalhar. Normalmente, o analista de sistema e o projetista são a mesma pessoa, mas isto é diferenciado pelo foco nas funções e habilidades requeridas.

Programadores

Baseados no projeto do sistema, os programadores escrevem o código do software utilizando uma linguagem de programação.

Testadores

Para cada produto funcional, deve-se realizar uma revisão de modo a encontrar: falhas ou erros. Isto sustenta a cultura de qualidade necessária ao desenvolvimento de software com qualidade. Isto garante que o produto funcional irá de encontro com os requisitos e padrões definidos.

8. Documentação no Esforço de Desenvolvimento

8.1. O que é documentação?

É um conjunto de documentos ou informações do produto que descrevem o sistema. Cada documento é desenhado para executar uma função específica, como:

- REFERÊNCIA, como por exemplo, especificações técnicas ou funcionais.
- INSTRUCIONAL, como por exemplo, tutoriais, demonstrações ou protótipos.
- MOTIVACIONAL, como por exemplo, brochuras, demonstrações ou protótipos.

Há vários tipos de documentação e informações funcionais do produto. Alguns são citados abaixo:

- Características e Funções do Sistema
- Sumário Gerencial e do Usuário
- Manual do Usuário
- Manual de Administração do Sistema
- Vídeo
- Multimídia
- Tutoriais
- Demonstrações
- Guia de Referência
- Guia de Referência Rápida
- Referências Técnicas
- Arquivos de Manutenção do Sistema
- Modelos de Teste do Sistema
- Procedimentos de Conversão
- Manual de Operações/Operador
- *Help On-Line*
- *Wall Charts*
- *Layout* de Teclado ou *Templates*
- Jornais

Bons documentos não geram sistemas complicados. No entanto, eles podem ajudar de outra forma. A tabela seguinte mostra como a documentação ajuda no processo de desenvolvimento de software.

Se o manual do usuário for desenvolvido durante....	***Então, o manual pode...***
A definição do produto	• Clarear procedimentos e regras • Identificar elementos não amigáveis • Aumentar as chances de satisfação do cliente
O projeto e codificação	• Clarear problemas e erros • Identificar causas de inconsistências • Forçar o projetista a tomar decisões mais cedo
A distribuição e uso do sistema	• Ajudar os usuários a se adaptarem ao sistema • Alertar sobre problemas no sistema • Negar responsabilidades

Tabela 2: Significância dos Documentos

Existem dois principais propósitos da documentação. Especificamente, eles:

- Fornecem um argumento racional e permanente para a estrutura do sistema ou comportamento através dos manuais de referência, guia do usuário e documentos de arquitetura do sistema.

- Servem como documentos transitórios que são parte de uma infra-estrutura envolvida em uma execução de um projeto real como: cenários, documentação do projeto interno, relatório de reuniões e problemas.

8.2. Critérios para Medição da Usabilidade de Documentos

Um documento útil aumenta o entendimento do sistema solicitado e o atual comportamento e estrutura. Serve como comunicação entre as versões arquiteturais do sistema. Fornece uma descrição de detalhes que não pode ser diretamente inferida a partir do próprio software ou a partir dos produtos funcionais. Alguns critérios para medição da usabilidade dos documentos estão listados abaixo:

1. ***Disponibilidade***. Usuários devem saber da existência dos documentos. Devem estar presentes no momento e local quando for necessário.
2. ***Adequação***. Devem ser adequados às tarefas e interesses do usuário. Devem ser precisos e completos. Documentos relacionados devem estar localizados em um manual ou livro.
3. ***Acessibilidade***. Devem estar em folhas ofício (8.5in x 11in) para fácil manuseio, armazenamento e recuperação. Deve ser fácil encontrar informações que o usuário deseja. Cada item da documentação deve ter um único nome para referência e referência cruzada, um propósito ou objetivo e um usuário alvo (a quem se destina o documento). Encaminhamentos para outros manuais e livros deverão ser evitados.
4. ***Legibilidade***. Devem ser compreensíveis sem explicações adicionais. Não devem possuir abreviações. Se for necessário, devem vir acompanhados de legenda. Devem ser escritos com fluência e em estilo e formatação de fácil leitura.

8.3. Importância dos Documentos e Manuais

Os documentos e manuais são importantes, pois são utilizados:

- Para diminuir custos. Com bons manuais, necessitam-se menos de treinamento, suporte e manutenção do sistema.
- Como ferramentas de venda e marketing. Bons manuais diferenciam seus produtos – um pobre manual significa um pobre produto – especialmente para produtos de software "fora de prateleira".
- Como entregáveis tangíveis. Gerenciamento e usuários sabem pouco sobre jargões da computação, programas e procedimentos, mas podem encontrar isso no manual do usuário.
- Como obrigações contratuais.
- Como coberturas de segurança. No caso de falta de pessoas, manuais e documentos técnicos servem como backup escrito.
- Como auxílio a teste e implementação. É importante incluir os seguintes itens como parte do manual do usuário – modelos e *scripts* de teste do sistema, procedimentos burocráticos e automatizados, treinamentos práticos para novos usuários e auxílio a projeto.
- Para comparar sistemas antigos e novos.

9. Exercícios

9.1. Especificação de escopo

1. Modele o escopo do Sistema de Informações Técnicas. Utilize a Figura 10 como guia.
2. Modele o escopo de um Sistema de Manutenção de Equipes e Times. Utilize a Figura 10 como guia.

9.2. Praticando o Walkthrough

1. Revise o escopo do modelo do Sistema de Informações Técnicas através de um *walkthrough*. Prepare uma lista de ações.
2. Revise o escopo do modelo do Sistema de Manutenção de Equipes & Times através de um *walkthrough*. Prepare uma lista de ações.

9.3. Trabalho sobre Projeto

O objetivo do trabalho sobre projeto é reforçar o conhecimento e habilidade adquirida nesta lição. Particularmente, são:

1. Definição de Escopo do Sistema
2. Criação de Modelo de Escopo de Sistema
3. Execução de *Walkthrough*

PRODUTOS:

1. Modelo de Escopo do Sistema
2. Lista de Ações

Parceiros que tornaram JEDI™ possível

Instituto CTS
Patrocinador do DFJUG.

Sun Microsystems
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Banco do Brasil
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

Politec
Suporte e apoio financeiro e logístico a todo o processo.

Borland
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

Instituto Gaudium/CNBB
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.