

UNIVERSIDADE ABERTA DO BRASIL
UNIVERSIDADE FEDERAL DE ALAGOAS
Instituto de Computação
Curso de Bacharelado em Sistemas de
Informação

Engenharia de Software

Unidade 1: Conceitos

Prof. Arturo Hernández Domínguez

Conteúdo:

- 1. Introdução*
- 2. Engenheiro de Software e a Resolução de Problemas*
- 3. Conceitos no Contexto de Engenharia de Software*
- 4. Responsabilidade Profissional e Ética*

Exercícios



Este trabalho está licenciado com uma Licença [Creative Commons - Atribuição-NãoComercial-SemDerivações 4.0 Internacional](https://creativecommons.org/licenses/by-nc-nd/4.0/).

Agosto – 2010

1. Introdução

Este texto representa uma introdução ao estudo da disciplina de Engenharia de Software (PRESSMAN, 2010; SOMMERVILLE, 2007; PFLEEGER, 1998; MAGELA, 2006). Neste contexto, são apresentados conceitos básicos (SOMMERVILLE, 2007) e considerações éticas (SOMMERVILLE, 2007).

Também é apresentado o código de ética da ACM¹, a IEEE² que objetiva fornecer ao aluno de Engenharia de Software um conjunto de informações relacionadas sobre o comportamento do profissional de Engenharia de Software no desenvolvimento da sua prática profissional.

Na seção 3, serão apresentadas questões e conceitos no contexto de Engenharia de Software, na seção 3, no contexto do engenheiro de software e a resolução de problemas é apresentado o processo de solução de problemas, na seção 4, considerações do ponto de vista responsabilidade profissional e ética são apresentadas, e na seção 5, são apresentadas as considerações finais.

2. Engenheiro de Software e a Resolução de Problemas

O engenheiro de software usa conhecimento de computação para ajudar na resolução de problemas (PFLEEGER, 1998), para isto é essencial o entendimento do problema.

Solucionando problemas

Solucionando problemas é representada através de um processo de análise e síntese (PFLEEGER, 1998).

Análise representa “quebrar” o problema maior em partes ou subproblemas (Figura 1) de tal forma que possamos lidar e entender mais facilmente o problema maior a partir do entendimento de cada subproblema e as interações entre esses subproblemas.

Uma vez realizada a análise, se deve construir a solução do problema a partir da integração da solução específica de cada subproblema.

1 ACM: Association for Computing Machinery

2 IEEE: Institute of Electrical and Electronic Engineers

Síntese representa a elaboração de uma estrutura maior a partir da integração das partes (Figura 2).

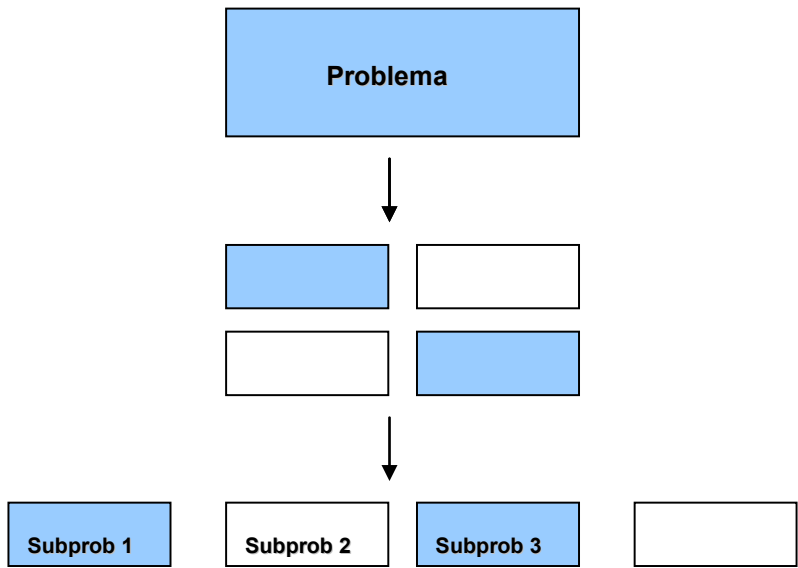


Figura 1: Análise no contexto da resolução de problemas.

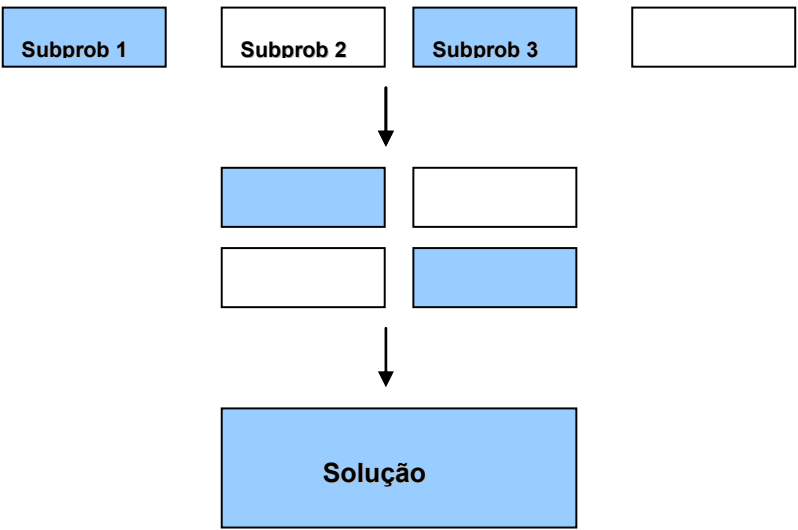


Figura 2: Síntese no contexto da resolução de problemas.

Exemplos no contexto de resolução de problemas:

- a construção de uma casa;
- a construção de um carro;
- a construção de um relógio digital;
- a construção de um computador.

No contexto do processo de construção de uma casa (PFLEEGER, 1998):

- Definição e análise dos requisitos da casa. Como queremos a casa ?
- Elaborar e documentar o projeto da casa.

Planta de uma casa, projeto de fundação, projeto estrutural, projeto hidro-sanitário (hidráulico e esgoto).

- Produção detalhada das especificações da casa. Projeto hidro-sanitário (hidráulico e esgoto). Projeto elétrico e telefônico.
- Identificação e projeto dos espaços e componentes (elementos) da casa.
- Execução da obra: Construção dos espaços e componentes (elementos) da casa. Também considerar Água, Esgoto e Energia elétrica.
- Teste de cada espaço e componente (elemento) da casa. Também testar as instalações de Água, Esgoto e Energia elétrica.
- Integração de todos os espaços, componentes (elementos), água, esgoto e energia elétrica da casa.
- Teste dos serviços da casa.
- Entrega da casa e validação do uso funcional da casa por parte dos moradores.
- Manutenção da casa pelos moradores.

3. Conceitos no Contexto de Engenharia de Software

Nesta seção, objetivando introduzir a disciplina, questões e conceitos no contexto de Engenharia de software são apresentadas.

Porque da engenharia de software?

O contexto no ano de 1968 em relação ao desenvolvimento de sistemas era:

Atraso em projetos importantes, custo superava as previsões, difícil de manter e desempenho insatisfatório. O desenvolvimento de software estava em crise. Objetivando discutir a crise de software, o conceito de Engenharia de Software foi proposto em 1968 (SOMMERVILLE, 2007).

O que é software?

Segundo Sommerville (2007), software é o programa, mas também a documentação e configuração associadas e necessárias para que o programa opere corretamente. Um sistema de software (SOMMERVILLE, 2007) consiste de um conjunto de programas separados; arquivos de configuração; documentação do sistema, que descreve a estrutura do sistema; a documentação do usuário, que explica como usar o sistema.

Características de software

O software é um elemento de um sistema lógico e não de um sistema físico (PRESSMAN, 2010). O software possui características diferentes daquelas do hardware (PRESSMAN, 2010):

1. O software é desenvolvido ou passa por um processo de engenharia; não é fabricado no sentido clássico.
2. O software não se desgasta. Com o passar do tempo o hardware começa a se desgastar. Também o software não é suscetível aos males ambientais que causam desgaste no hardware.

Atributos essenciais de um bom software.

Segundo Sommerville (2007), os atributos essenciais de um sistema de software bem projetado são:

<i>Caraterística do produto</i>	<i>Descrição</i>
--	-------------------------

<i>Facilidade de manutenção</i>	<i>O software deve ser escrito de modo que possa evoluir para atender às necessidades de mudança dos clientes. A mudança de software é uma consequência inevitável de um ambiente de negócios em constante mutação.</i>
---------------------------------	---

<i>Confiança</i>	<i>Um software confiável não deve causar danos físicos ou econômicos no caso de falha no sistema.</i>
------------------	---

<i>Eficiência</i>	<i>O software não deve desperdiçar os recursos do sistema, como memória e ciclos de processador. Portanto, a eficiência inclui tempo de resposta, tempo de processamento, utilização de memória, etc.</i>
-------------------	---

<i>Usabilidade</i>	<i>O software deve ser usável, sem esforço excessivo, pelo tipo de usuário para o qual ele foi projetado. Deve apresentar uma interface com o usuário adequada.</i>
--------------------	---

Engenharia de software

Segundo Sommerville (2007), a engenharia de software é uma disciplina de engenharia relacionada com todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até sua manutenção, depois que o sistema entrou em operação.

É importante destacar que a engenharia de software além de considerar os processos técnicos de desenvolvimento de software, também está relacionada com o gerenciamento de projeto de software e o desenvolvimento de ferramentas, e métodos que apoiem a produção de software (SOMMERVILLE, 2007).

Processo de software

Um processo de software é um conjunto de atividades e resultados associados que produz um produto de software. As atividades fundamentais e comuns a todos os processos de software são (SOMMERVILLE, 2007):

1. Especificação de software: clientes e engenheiros definem o software a ser produzido e as restrições para a sua operação.
2. Desenvolvimento de software: o software é projetado e programado.
3. Validação de software: na qual o software é verificado para garantir que é o que o cliente deseja.
4. Evolução do software: o software é modificado para se adaptar às mudanças dos requisitos do cliente, e do mercado.

Modelo de processo de software

Um modelo de processo de software é uma descrição simplificada de um processo de software (SOMMERVILLE, 2007).

Métodos de engenharia de software

Um método de engenharia de software segundo Sommerville (2007), é uma abordagem estruturada para desenvolvimento de software, objetivando facilitar a produção de software de alta qualidade dentro de custos adequados.

Todos os métodos estão baseados em modelos de desenvolvimento de um sistema, que pode ser representado graficamente, os modelos representam a especificação e projeto de um sistema.

Os componentes de um método são (SOMMERVILLE, 2007):

- Modelos e notação usada para a definição desses modelos.

Exemplos de modelos: Modelo de objetos (Diagrama de Objetos), Modelo de máquina de estados (Diagrama de Estados), Diagrama de Classes, Diagrama de Componentes, Diagrama de Atividades, Diagrama de Implantação.

Notação: UML (Unified Modeling Language) – Linguagem Unificada de Modelagem de sistemas orientados a objetos (BOOCH et al., 2005).

- Regras ou restrições que são sempre aplicáveis aos modelos do sistema.

Exemplo de regra: Cada entidade em um modelo deve ter um único nome

- Guia de processo. Descrições das atividades que devem ser seguidas para desenvolver os modelos de sistema e a organização dessas atividades.
- Recomendações. Heurísticas que caracterizam uma boa pratica de projeto nesse método.

CASE (Computer-Aided Software Engineering)

CASE (Computer-Aided Software Engineering) corresponde a Engenharia de Software Auxiliada por Computador, isto é um conjunto de programas usados para dar apoio às atividades de processo de software, tais como análise de requisitos, modelagem do sistema, programação e teste.

Exemplos de Tecnologia CASE:

- Ferramentas de edição (editor de texto, editor de diagramas);

- Ferramentas de apoio a métodos (editores dos modelos de análise e projeto, dicionário de dados e gerador de código);
- Ferramentas de programação (um ambiente integrado para o desenvolvimento de software – IDE);
- Ferramentas de teste (geradores de dados de teste).

4. Responsabilidade Profissional e Ética

O trabalho, no contexto de engenharia de software, implica responsabilidades mais amplas do que a aplicação de habilidades técnicas. Se deve comportar de forma responsável ética e moralmente (SOMMERVILLE, 2007). Se deve defender comportamentos padrões normais de honestidade e integridade. Em algumas áreas um comportamento aceitável está associado a uma noção de responsabilidade profissional, exemplos (SOMMERVILLE, 2007):

<i>Área</i>	<i>Descrição</i>
<i>Confidencialidade</i>	<i>Se deve respeitar a confidencialidade dos funcionários ou clientes. Independentemente de ter ou não assinado um acordo formal.</i>
<i>Competência</i>	<i>Não se deve conscientemente aceitar um trabalho que esteja fora da sua competência.</i>
<i>Direitos sobre propriedade intelectual</i>	<i>Se deve ter cuidado para assegurar que a propriedade intelectual de funcionários e clientes seja protegida.</i>
<i>Mau uso de computadores</i>	<i>Não se deve usar as habilidades técnicas para fazer mau uso dos computadores de outras pessoas.</i>

4.1 Código de Ética e Prática Profissional da Engenharia de Software

Um código de conduta profissional ou código de ética foi publicado pelas organizações internacionais (SOMMERVILLE, 2007): ACM, a IEEE e British Computer Society.

O fundamento do código de ética da ACM/IEEE é:

Os computadores desempenham um papel central e crescente no comércio, na indústria, no governo, na medicina, na educação, no entretenimento e na sociedade em geral. Os engenheiros de software são aqueles que contribuem, por participação direta ou pelo ensino, com a análise, especificação, projeto, desenvolvimento, certificação, manutenção e teste de sistemas de software. Em razão de seu papel no desenvolvimento de sistemas de software, eles têm

oportunidades significativas de praticar o bem ou de causar o mal, tornarem outros capazes de praticar o bem ou causar o mal ou de influenciar outros indivíduos a praticar o bem ou causar o mal. Para garantir, tanto quanto possível, que seus esforços sejam utilizados para o bem, os engenheiros de software devem se comprometer a fazer da engenharia de software uma profissão benéfica e respeitada (SOMMERVILLE, 2007).

Os oito princípios do código de ética da ACM/IEEE são (SOMMERVILLE, 2007):

<i>Princípio do Código de Ética</i>	<i>Descrição</i>
<i>1. Público</i>	<i>Os engenheiros de software devem agir consistentemente com o interesse público.</i>
<i>2. Cliente e Empregador</i>	<i>Os engenheiros de software devem agir dentro dos melhores interesses de seu cliente e empregador, de forma consistente com o interesse público.</i>
<i>3. Produto</i>	<i>Os engenheiros de software devem assegurar que seus produtos e as modificações a eles relacionadas atendam aos mais altos padrões profissionais possíveis.</i>
<i>4. Julgamento</i>	<i>Os engenheiros de software devem manter a integridade e a independência em seu julgamento profissional.</i>
<i>5. Gerenciamento</i>	<i>Os gerentes e líderes de engenharia de software devem aceitar e promover uma abordagem ética no gerenciamento de desenvolvimento e manutenção de software.</i>
<i>6. Profissão</i>	<i>Os engenheiros de software devem promover a integridade e a reputação da profissão de forma consistente com o interesse público</i>
<i>7. Colegas</i>	<i>Os engenheiros devem ser honestos e colaborativos com seus colegas.</i>
<i>8. Indivíduo</i>	<i>Os engenheiros de software devem participar, ao longo da vida, aprendendo, respeitando e promovendo uma abordagem ética na prática da profissão.</i>

Conforme Massiero e Bigonha (2008) “Falhas éticas graves, quando ocorrem, podem causar severos prejuízos às organizações e às pessoas envolvidas”.

5. Considerações Finais

Conforme citado inicialmente este texto representa uma introdução ao estudo da Engenharia de Software. Objetiva fornecer um conjunto de conceitos básicos e também contextualiza a atividade do engenheiro de software do ponto de vista ético e responsabilidade profissional.

No contexto de Engenharia de Software foram apresentados os seguintes conceitos: software, características de software, atributos essenciais de um bom software, engenharia de software, processo de software, método, ferramenta CASE.

No contexto do engenheiro de software e a resolução de problemas foi apresentado o processo de solução de problemas.

Também foram apresentadas algumas considerações de natureza ética e o código de ética (oito princípios) da ACM/IEEE foi apresentado.

Referências Bibliográficas

Booch G., Rumbaugh J., Jacobson I. UML: Guia do usuário, 2ª edição. Editora Campus. 2005.

Magela, R. Engenharia de Software Aplicada: Princípios (volume 1). Alta Books. 2006.

Massiero P. C., Bigonha R. S. Ética e Computação, Volume 1 - Número 1 - Dezembro, SBC HORIZONTES. 2008.

Pfleeger S. L. Software Engineering: theory and practice. Prentice-Hall. 1998.

Pressman R. Engenharia de Software, 6a edição, AMGH Editora. 2010.

Software Engineering Institute, Disponível em: <<http://www.sei.cmu.edu>>. Acesso em 1 ago. 2010.

Sommerville, I. Engenharia de Software. 8a Edição. Addison Wesley. 2007.

Unified Modeling Language, Disponível em: <<http://www.uml.org>>. Acesso em 1 ago.2010.

Exercícios propostos

1. No contexto do processo de resolução de problemas, mostrar a aplicação desse processo passo a passo no contexto de uma casa, a ideia é a construção de uma escada e um andar com quartos para a família.
2. No contexto do processo de resolução de problemas, propor um exemplo do cotidiano na sua casa, município, cidade ou trabalho, ilustrando passo a passo a aplicação desse processo.
3. No contexto do processo de resolução de problemas, mostrar a aplicação desse processo passo a passo no contexto da realização de uma viagem de férias com sua família sem o uso de uma agencia de viagens, o transporte a ser utilizado é um carro a ser dirigido por alguém da sua família.
4. Refletir e descrever como seria um processo para a construção de um produto, tal como um carro ou uma calculadora.
5. Discutir com os seus colegas sobre as características de software e suas diferenças em relação a produtos, tais como, um carro ou um prédio.
6. Pesquisar na internet sobre a crise de software (1968) e comparar com a realidade das organizações, atualmente (2010), do ponto de vista desenvolvimento de software.
7. Discutir com os seus colegas sobre comportamento não ético nas organizações no contexto de desenvolvimento de software.

UNIVERSIDADE ABERTA DO BRASIL
UNIVERSIDADE FEDERAL DE ALAGOAS
Instituto de Computação
Curso de Bacharelado em Sistemas de
Informação

Engenharia de Software

Unidade 2: Processo de Software

Prof. Arturo Hernández Domínguez

Conteúdo:

- 1. Introdução*
 - 2. O ciclo de vida clássico*
 - 3. Modelos de processos*
 - 4. Iteração de processo*
 - 5. Atividades de processo*
 - 6. Processo unificado*
 - 7. Desenvolvimento ágil de software*
- Atividades usando o Ambiente Virtual de Aprendizagem*
- Exercícios*



Este trabalho está licenciado com uma Licença [Creative Commons - Atribuição-NãoComercial-SemDerivações 4.0 Internacional](https://creativecommons.org/licenses/by-nc-nd/4.0/).

Setembro – 2010

1. Introdução

Um processo é um conjunto de passos envolvendo atividades e recursos que produz um resultado (PFLEEGER, 1998). Um processo, geralmente, envolve um conjunto de técnicas e ferramentas.

A seguir as características de um processo (PFLEEGER, 1998):

O processo prescreve as principais atividades do processo;

O processo usa recursos e produz produtos intermediários e finais;

O processo pode ser composto de subprocessos;

As atividades são organizadas em sequência;

Cada processo tem associado orientações que explicam os objetivos de cada atividade.

No contexto de desenvolvimento de software, a engenharia de software é uma disciplina que integra processo, métodos e ferramentas para o desenvolvimento de softwares de computador (PRESMAN, 2010).

“Um processo define quem está fazendo o quê, quando e como para alcançar um certo objetivo.” I. Jacobson, G. Booch, e J. Rumbaugh

Podemos pensar no processo de software como o roteiro ou serie de passos previsíveis que ajuda a criar a tempo um resultado (sistema de software) de alta qualidade (PRESSMAN, 2010).

Na seção 2, o ciclo de vida clássico de software é apresentado. Na seção 3, se refere a modelos de processo. Na seção 4, considerações sobre a iteração de processo são apresentadas. Na seção 5, se refere as atividades básicas de processo. Na seção 6, as fases do processo unificado são descritas e na seção 7, o desenvolvimento ágil de software é apresentado.

2. O Ciclo de Vida

Quando o processo envolve a construção de algum produto, as vezes se refere a esse processo como ciclo de vida. Desta forma o processo de desenvolvimento de software as vezes é chamado de ciclo de vida de software (PFLEEGER, 1998), porque ele

descreve a vida de um produto de software desde a concepção a implementação, liberação, uso e manutenção.

No contexto de engenharia de software, o modelo de ciclo de vida tradicional é o chamado modelo em cascata (ver seção 3.1).

Segundo AUDY (2005) no contexto de um sistema de informação, o ciclo de vida de um sistema de informação baseado em computador abrange as fases de análise, projeto, construção, instalação, produção e manutenção.

3. Modelos de Processos

Existem na literatura diferentes processos de software propostos (SOMMERVILE, 2007). Nesta seção, serão apresentados três modelos de processo de software: o modelo em cascata, desenvolvimento evolucionário e engenharia de software baseada em componentes.

3.1 Modelo em cascata

O modelo cascata¹ sugere uma abordagem sistemática e sequencial (Figura 1) para o desenvolvimento de software (Requisitos e Análise, Projeto, Implementação, Integração, Operação e Manutenção) (PRESSMAN, 2010). Quando os requisitos são bem definidos e estáveis, este modelo é adequado, já que o trabalho flui até a implantação de um modo linear.

As atividades associadas ao modelo cascata são (SOMMERVILE, 2007):

1. Análise e definição de requisitos. As funções, restrições e objetivos do sistema são estabelecidos por meio da consulta aos usuários do sistema.
2. Projeto de sistemas de software. Estabelece uma arquitetura do sistema. O projeto de software envolve a identificação e especificação de subsistemas e suas interações.
3. Implementação e teste de unidades. Neste estágio, o projeto de software é compreendido como um conjunto de unidades. O teste de unidades envolve verificar que cada unidade atenda a sua especificação.

¹ : O modelo cascata é o paradigma mais antigo da engenharia de software. Algumas vezes chamado de ciclo de vida clássico (Pressman, 2010).

4. Integração e teste de sistemas. As unidades de programas são integradas e testadas como um sistema completo a fim de garantir que os requisitos de software foram atendidos. O sistema é entregue ao cliente depois dos testes.

5. Operação e manutenção. O sistema é instalado e colocado em operação. A manutenção consiste em corrigir erros e acrescentar novas funções à medida que novos requisitos são identificados.

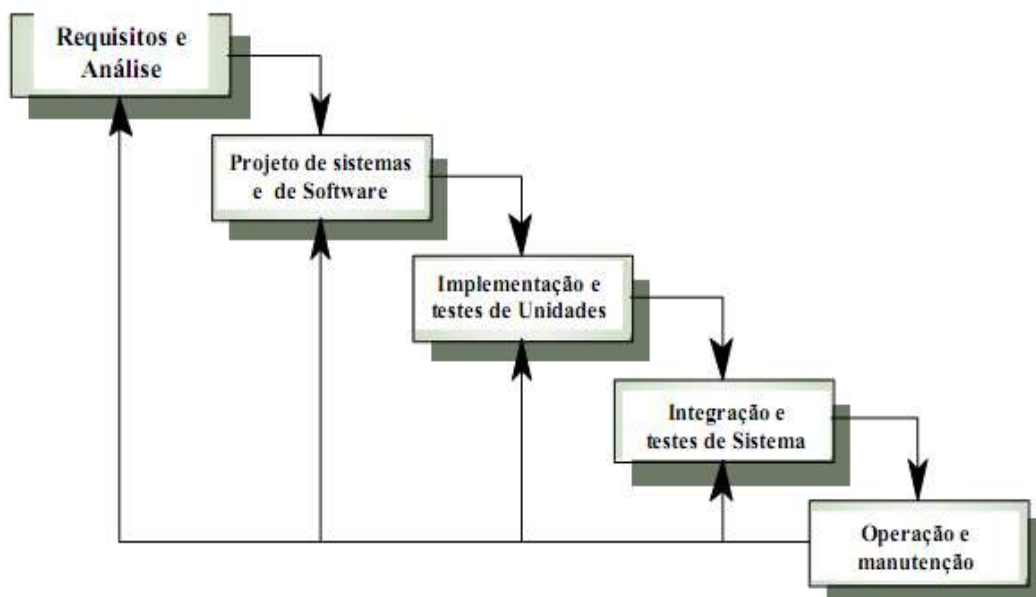


Figura 1: Modelo em cascata.

Os principais problemas encontrados quando o modelo em cascata é aplicado são (PRESSMAN, 2010):

1. Projetos reais dificilmente seguem o fluxo sequencial proposto pelo modelo em cascata.
2. Para o cliente é difícil estabelecer todos os requisitos explicitamente.
3. Uma versão executável do sistema só ficará disponível depois da conclusão da modelagem (análise e projeto), e isto poderá ser demorado.

Atualmente, o desenvolvimento de software é em ritmo rápido e sujeito a muitas mudanças (de funções e conteúdo da informação). Para este tipo de desenvolvimento o modelo em cascata não é adequado.

3.2 Desenvolvimento evolucionário

O desenvolvimento evolucionário (Figura 2) é baseado na ideia de desenvolver uma implementação inicial, obter um retorno por parte dos usuários e fazer seu aprimoramento através de varias versões (SOMMERVILLE, 2007). As atividades de especificação, desenvolvimento e validação são realizadas concorrentemente.

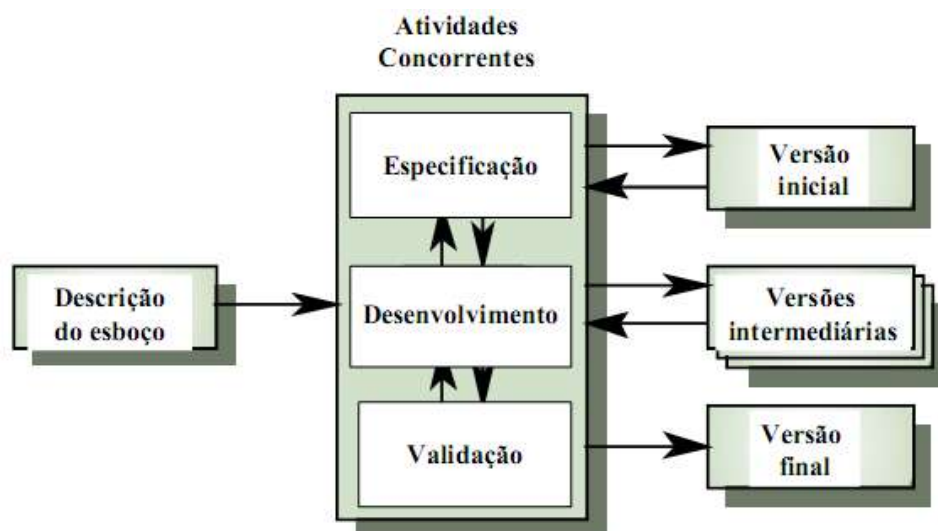


Figura 2: Desenvolvimento evolucionário.

No contexto de produzir uma versão imediata a abordagem evolucionaria é mais eficaz do que a abordagem em cascata. As desvantagens são: o processo não é visível, não existe a elaboração de documentos sobre a especificação do sistema.

3.3 Engenharia de software baseada em componentes

A abordagem de engenharia de software com base em componentes conta com uma base de componentes de software reutilizáveis e um framework de integração (SOMMERVILLE, 2007). Um componente de software é uma peça de software reutilizável que é desenvolvida de forma independente, pode ser utilizada junto com outros componentes para construir (compor) unidades de software maiores. Os componentes de software podem ser reutilizados para fazer parte de um novo sistema (Figura 3). As vantagens desta abordagem é a redução da quantidade de software a ser

desenvolvida, a redução de custos e riscos. A desvantagem é devido a adequações nos requisitos, devido aos componentes disponíveis, isso pode levar a um sistema que não atenda as reais necessidades dos usuários.

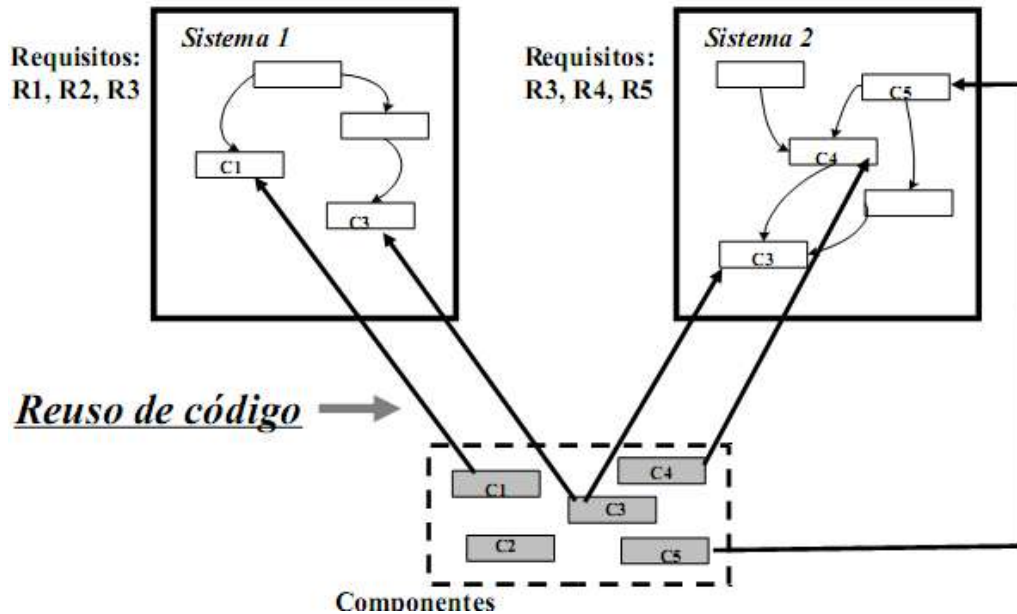


Figura 3: Desenvolvimento de novos sistemas de software reutilizando componentes de software.

4. Iteração de Processo

A ideia básica dos processos iterativos é que partes do processo são repetidas, à medida que os requisitos do sistema evoluem. Neste contexto, a essência de um processo iterativo é que a especificação é desenvolvida em conjunto com o software (SOMMERVILLE, 2007). Para a maioria dos grandes sistemas é necessário utilizar diferentes abordagens para diferentes partes do sistema, assim um modelo híbrido é necessário. Dois modelos projetados para apoiarem a iteração de processo são: desenvolvimento incremental e desenvolvimento em espiral.

4.1 Desenvolvimento incremental

Em um processo de desenvolvimento incremental (Figura 4), a partir da identificação de funções e correspondente classificação (quais são as mais importantes e quais as menos importantes), é definida uma série de incrementos, com cada incremento fornecendo um subconjunto das funcionalidades do sistema (SOMMERVILLE, 2007)(LARMAN, 2000). As funções são alocadas em função da prioridade. As funções

mais importantes são entregues, primeiro, ao cliente, posteriormente serão entregues as funções menos importantes. Não é necessário utilizar o mesmo processo para o desenvolvimento de cada incremento. Para um incremento pode ser utilizado o modelo em cascata se a especificação das funções são bem definidas. Quando a especificação não for bem definida, poderá ser utilizado um modelo de desenvolvimento evolucionário.

As vantagens do desenvolvimento incremental são (SOMMERVILLE, 2007):

1. Os clientes não precisam aguardar até que todo o sistema seja entregue, para tirar proveito dele.
2. Os clientes podem utilizar os primeiros incrementos como um protótipo e obter uma experiência que forneça os requisitos ajustados para incrementos posteriores do sistema.
3. Existe um risco menor de fracasso completo do sistema.
4. As funções de sistema mais importantes são as mais testadas.

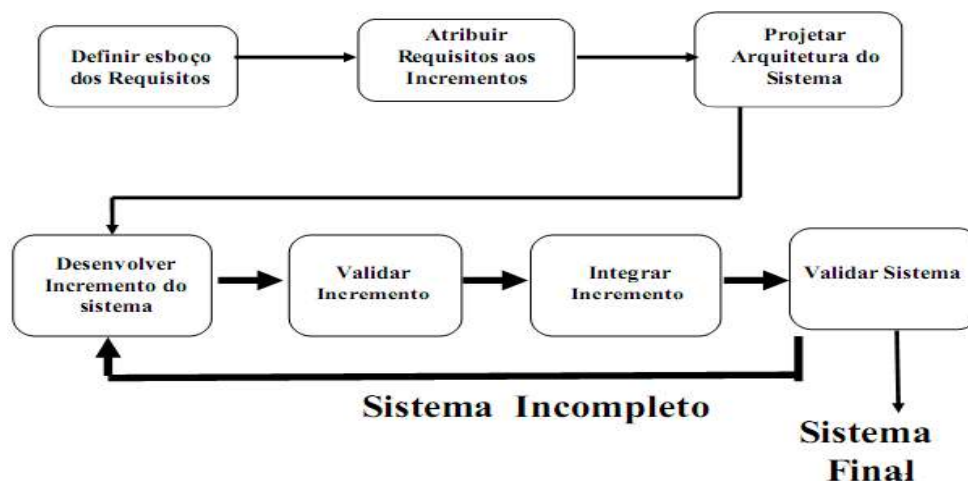


Figura 4: Desenvolvimento incremental.

4.2 Desenvolvimento em espiral

O processo é representado como uma espiral (Figura 5). Cada loop na espiral representa uma fase do processo de software (SOMMERVILLE, 2007). Desta forma o loop mais

interno pode corresponder a viabilidade do sistema, o loop seguinte, à definição de requisitos, o próximo loop, ao projeto do sistema, ...

Quatro setores são associados para cada loop da espiral (SOMMERVILLE, 2007) (Figura 5):

1. Definição de objetivos. São definidos os objetivos específicos para esta fase, são identificadas restrições e é preparado um plano de gerenciamento. Também são identificados os riscos.
2. Avaliação e redução de riscos. Para cada risco, é realizada uma análise e providências são tomadas para reduzir esse risco.
3. Desenvolvimento e validação. Depois da avaliação de riscos, é definido um modelo de desenvolvimento.
4. Planejamento. O projeto é revisto e é tomada uma decisão sobre continuar ou não com o próximo loop da espiral.

A distinção do modelo em espiral em relação a outros modelos é a explícita consideração dos riscos². O modelo da espiral abrange outros modelos de processo, um desenvolvimento evolucionário poderá ser utilizado num loop e no seguinte loop, um desenvolvimento em cascata poderá ser realizado.

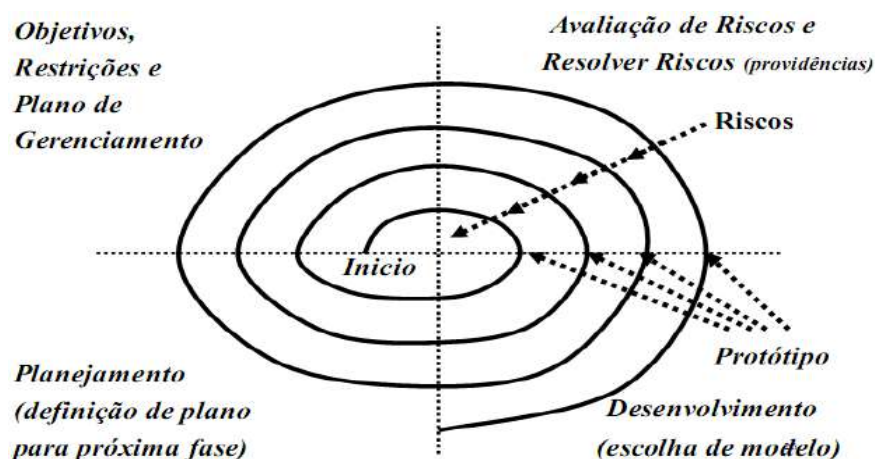


Figura 5: Modelo de processo em espiral.

2 Um risco é algo que pode acontecer de errado (SOMMERVILLE, 2007)

5. Atividades de Processo

Todos os processos de software incluem as quatro atividades básicas do processo (SOMMERVILLE, 2007): especificação de software, desenvolvimento (projeto e implementação), validação de software e evolução de software.

A engenharia de requisitos (Figura 6) é o processo de desenvolvimento de uma especificação de software, na qual a funcionalidade do software deve ser definida. Ela se refere a desenvolver uma especificação que possa ser compreendida pelos usuários do sistema e uma especificação mais detalhada para os desenvolvedores do sistema.

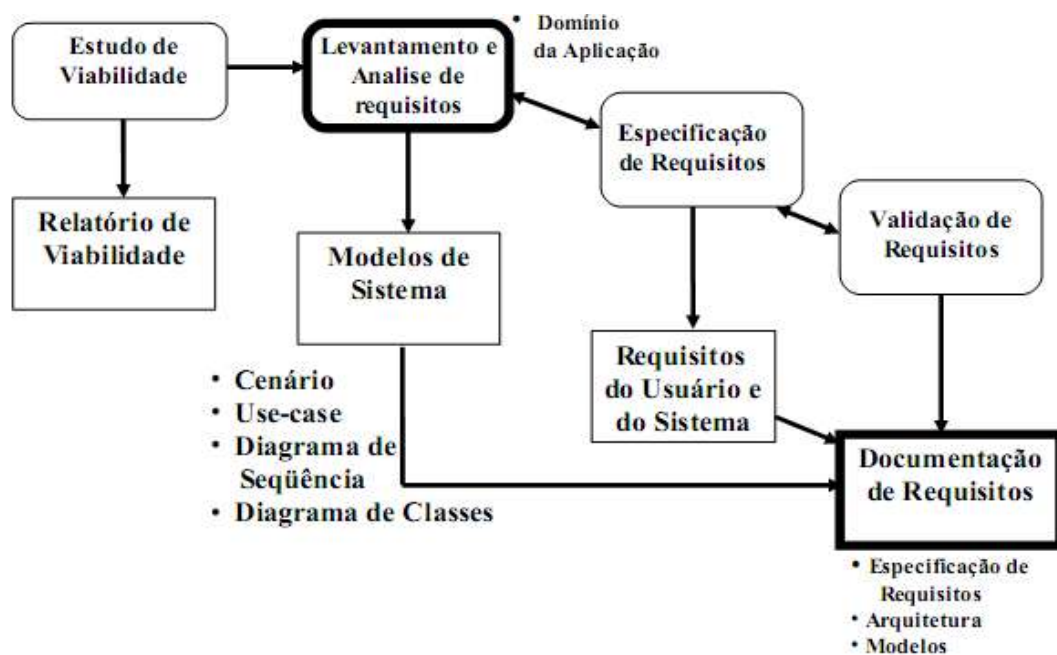


Figura 6: Engenharia de requisitos.

O projeto (Figura 7) e a implementação se ocupam da transformação de uma especificação de requisitos em um sistema executável. O software é projetado e programado segundo a especificação realizada.

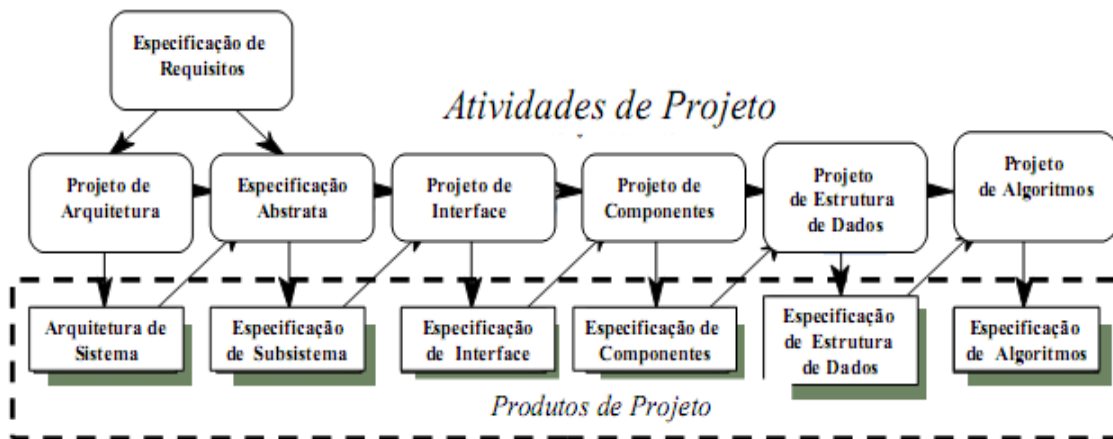


Figura 7: Projeto de software.

A validação de software verifica se o sistema está em conformidade com sua especificação e se ele atende às reais necessidades dos usuários do sistema.

A evolução do software se ocupa de modificar os sistemas de software existentes, considerando às necessidades mutáveis do cliente.

6. Processo Unificado

O processo unificado (Figura 8) (PRESSMAN, 2010) é um processo de software “orientado por casos de uso, centrado na arquitetura, iterativo e incremental”, projetado para métodos e ferramentas UML (Linguagem Unificada de Modelagem) (BOOCH et al, 2005). O processo unificado é um modelo incremental representado por cinco fases (concepção, elaboração, construção, transição e produção) (PRESSMAN, 2010).

6.3. Fase de construção

Nesta fase, se realiza a conversão do modelo de projeto para componentes de software implementados. Usando o modelo arquitetural como entrada, na fase de construção os componentes de software são desenvolvidos ou adquiridos para tornar cada caso de uso operacional. As funções requeridas do incremento (versão) serão implementadas. Desta forma, os componentes são implementados, testes unitários são projetados e executados para cada um deles. As atividades de integração (montagem de componentes e testes de integração) são realizadas (PRESSMAN, 2010).

6.4. Fase de transição

Nesta fase, se transfere o software do desenvolvedor para o usuário final para a realização de testes beta e aceitação. A equipe de software elabora informações de apoio necessárias (exemplos: manuais de usuário e procedimentos de instalação). Na conclusão da fase de transição, o incremento de software se torna uma versão utilizável do software (PRESSMAN, 2010).

6.5. Fase de produção

Na qual um monitoramento contínuo e suporte são realizados. Relatórios de defeitos e solicitações de modificações são elaborados e avaliados (PRESSMAN, 2010).

7. Desenvolvimento Ágil de software

Os processos prescritivos são atrativos para a gerência, mas não para a maioria dos desenvolvedores (AMBLER, 2004), isto é, os processos prescritivos são baseados no paradigma comando e controle.

7.1 Os valores do manifesto ágil

Segundo AMBLER (2004) os valores do Manifesto Ágil são:

1. Indivíduos e interações valem mais que processos e ferramentas, equipes constroem sistemas de software, elas precisam trabalhar junto com programadores, testadores, gerentes de projeto, projetistas e clientes;
2. Um software funcionando vale mais que a documentação extensa;

3. A colaboração do cliente vale mais que a negociação de contrato, os desenvolvedores de sucesso trabalham próximos aos clientes, descobrindo o que os clientes necessitam;
4. Responder a mudanças vale mais que seguir um plano, a mudança é uma realidade no desenvolvimento de software que o processo de software deve refletir.

7.2 Os princípios do desenvolvimento ágil do software

A Aliança Ágil definiu um manifesto contendo doze princípios aos quais as metodologias ágeis de desenvolvimento de software devem se adequar. Os doze princípios do manifesto são (AMBLER, 2004):

1. Nossa maior prioridade é satisfazer ao cliente mediante entregas de software de valor em tempo hábil e continuamente.
2. Receber bem mudanças de requisitos, mesmo em uma fase mais avançada no desenvolvimento.
3. Entregar software em funcionamento com frequência de algumas semanas a alguns meses, de preferência na menor escala de tempo.
4. As equipes de negócios e de desenvolvimento devem trabalhar juntas diariamente durante o tempo todo.
5. Construa projetos ao redor de indivíduos motivados. Dê-lhes o ambiente e o apoio de que eles precisam e confie neles para realizar o trabalho.
6. O método mais eficiente de levar informações para uma equipe de desenvolvimento e fazê-las circular é a conversa cara a cara.
7. Ter o software funcionando é a principal medida de progresso.
8. Processos ágeis promovem o desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários deveriam ser capazes de manter um ritmo constante indefinidamente.
9. Atenção contínua a excelência técnica e a um bom projeto aumentam a agilidade.
10. Simplicidade é essencial.
11. As melhores arquiteturas, requisitos e projetos provêm de equipes organizadas.

12. Em intervalos regulares, a equipe deve refletir sobre como se tornar mais eficaz e então se ajustar e adaptar seu comportamento.

7.3 Modelagem ágil

A metodologia modelagem ágil é definida como um conjunto de práticas guiado por princípios e valores para profissionais de software aplicarem em seu dia a dia. A modelagem ágil não é um processo prescritivo, não define procedimentos detalhados sobre como criar um determinado tipo de modelo. Segundo AMBLER (2004) a modelagem ágil não significa menos modelagem.

A modelagem ágil não é um processo completo de software. Seu foco é a modelagem e a documentação eficazes. Devido ao foco da modelagem ágil ser uma parte do processo de software, precisa-se usá-la com outros processos, como XP (eXtreme Programming – programação extrema) ou Processo Unificado (Figura 9) (AMBLER, 2004). As práticas da modelagem ágil poderão ser adotadas em projetos não ágeis e delas se beneficiar.

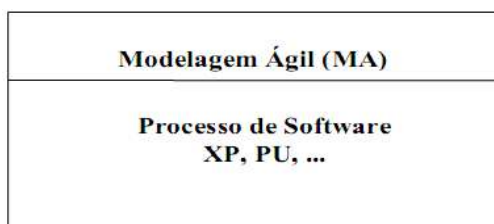


Figura 9: A modelagem ágil aperfeiçoa outros processos de software
Fonte: (AMBLER, 2004)

7.3.1 Escopo da modelagem ágil

Conforme AMBLER (2004), a seguir os pontos principais que objetivam sintetizar o escopo da modelagem ágil:

1. A modelagem ágil é uma atitude e não um processo descritivo. A modelagem ágil não é uma receita de bolo para o desenvolvimento.
2. A modelagem ágil é um suplemento dos métodos preexistentes; não uma metodologia completa. As técnicas da modelagem ágil devem ser utilizadas para melhorar o trabalho de modelagem de equipes de projetos que seguem metodologias ágeis, como XP, SCRUM e outras.
3. A modelagem ágil é algo que funciona na prática.

4. A modelagem ágil não é um ataque à documentação. A documentação ágil é tão simples quanto possível, tão mínima quanto possível.

8. Considerações Finais

Este texto apresentou o que é um processo de software, o ciclo de vida clássico de software, três modelos de processo de software.

Os modelos de processo apresentados foram o modelo em cascata, desenvolvimento evolucionário e engenharia de software baseada em componentes. No contexto da iteração de processo, dois modelos projetados para apoiarem a iteração de processo, o desenvolvimento incremental e o desenvolvimento em espiral, foram apresentados.

As fases do processo unificado (concepção, elaboração, construção, transição e produção) foram apresentadas.

Os valores do manifesto ágil e os princípios do desenvolvimento ágil foram apresentados, assim como o que se entende por modelagem ágil.

Referências Bibliográficas

Ambler S. W., Modelagem Ágil. Editora Bookman. Porto Alegre, 2004.

Audy J. L. N., Andrade G. K., Cidral A., Fundamentos de Sistemas de Informação, Editora Bookman. Porto Alegre, 2005.

Booch G., Rumbaugh J., Jacobson I. UML: Guia do usuário, 2ª edição. Editora Campus. 2005.

Larman C. Utilizando UML e Padrões: uma introdução à análise e ao projeto orientado a objetos. Editora Bookman. Porto Alegre, 2000.

Pfleeger S. L. Software Engineering: theory and practice. Prentice-Hall. 1998.

Pressman R. Engenharia de Software, 6a edição, AMGH Editora. 2010.

Sommerville, I. Engenharia de Software. 8a Edição. Addison Wesley. 2007.

Anexo

1. O modelo da espiral detalhado é mostrado na Figura 10 (SOMMERVILLE, 2007).

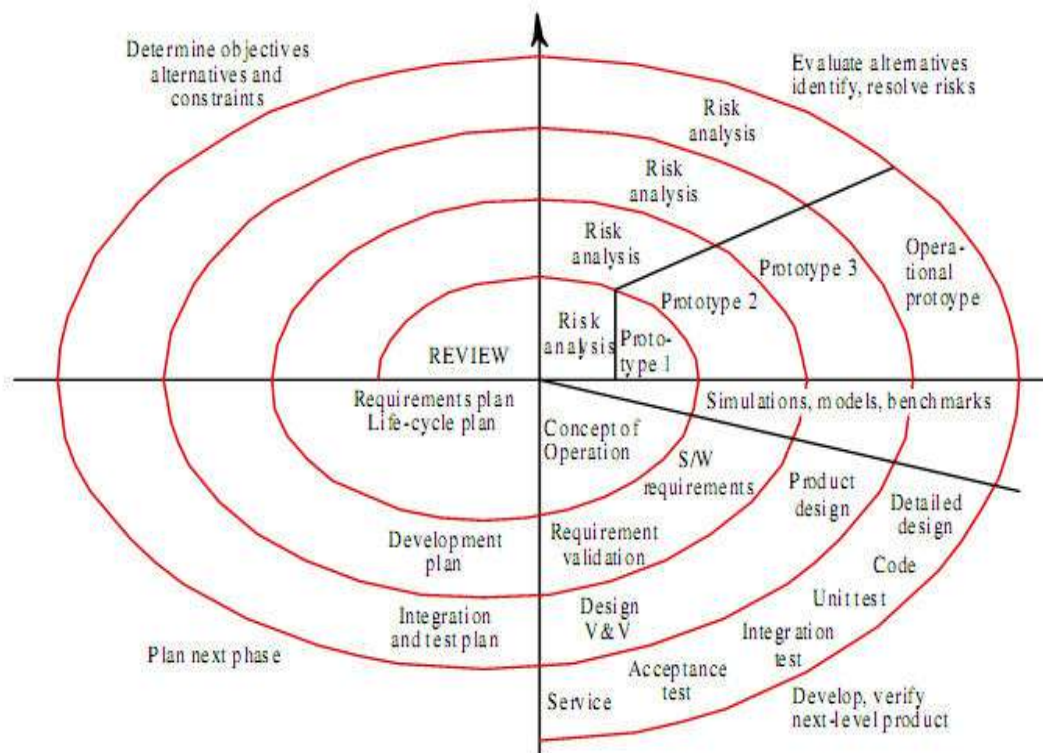


Figura 10: Modelo da Espiral detalhado.
Fonte: SOMMERVILLE (2007)

Exercícios propostos

1. No contexto de uma festa, definir passo a passo como acontece o processo de organização e realização de uma festa de aniversário. Descrever, passo a passo de forma detalhada, as atividades envolvidas na realização da festa, com a participação da sua família, convidados, música, comida e bebida.
2. No contexto de uma casa, definir passo a passo como acontece o processo de construção de uma casa (projeto, construção, uso e manutenção de uma casa). Fazer uma comparação com o processo de desenvolvimento de software.
3. Pesquisar, analisar e descrever as diferenças entre os modelos Cascata e Espiral.
4. Pesquisar, analisar e refletir sobre qual seria o modelo de processo mais adequado para o desenvolvimento de um software de tipo:
 - livraria virtual (venda de livros pela internet);
 - sistema a ser utilizado por uma agência de viagens para o gerenciamento, pela internet, da reserva e venda de passagens aéreas. Também considerar os pacotes turísticos, envolvendo a passagem e a hospedagem;
 - ambiente virtual de aprendizagem (AVA).
5. Pesquisar e refletir sobre as condições necessárias, no contexto de uma organização, para ter um desenvolvimento ágil de software bem sucedido. Para que tipo de projetos é adequado o desenvolvimento ágil de software.

UNIVERSIDADE ABERTA DO BRASIL
UNIVERSIDADE FEDERAL DE ALAGOAS
Instituto de Computação
Curso de Bacharelado em Sistemas de
Informação

Engenharia de Software

Unidade 3: Engenharia de Requisitos

Prof. Arturo Hernández Domínguez

Conteúdo:

- 1. Introdução***
- 2. Requisitos***
- 3. Estudo de viabilidade***
- 4. Elicitação e análise de requisitos***
- 5. Especificação de requisitos***
- 6. Validação de requisitos***
- 7. Documento de requisitos***

Atividades usando o Ambiente Virtual de Aprendizagem

Exercícios



Este trabalho está licenciado com uma Licença [Creative Commons - Atribuição-NãoComercial-SemDerivações 4.0 Internacional](https://creativecommons.org/licenses/by-nc-nd/4.0/)

Setembro – 2010

1. Introdução

O desenvolvimento de um sistema de software que não atenda as reais necessidades de um cliente não será útil. Para evitar isto, deve se comunicar com o cliente e compreender as reais necessidades desse cliente para que se possibilite a construção de um sistema de software adequado que resolva e satisfaça as reais necessidades do cliente.

“A parte individual mais difícil da construção de um sistema de software é decidir o que construir. Nenhuma parte do trabalho danifica tanto o sistema resultante se for feita errado. Nenhuma outra parte é mais difícil de consertar depois.” (Fred Brooks apud PRESSMAN, 2010)

Neste contexto, a engenharia de requisitos (Figura 1), objetiva uma melhor compreensão do problema, a ser resolvido, por parte dos analistas ou engenheiros de software (PRESSMAN, 2010). A engenharia de requisitos (SOMMERVILLE, 2007) é o processo de desenvolvimento de uma especificação de software, na qual a funcionalidade do software deve ser definida. Ela se refere a desenvolver uma especificação que possa ser compreendida pelos usuários do sistema e uma especificação mais detalhada para os desenvolvedores do sistema.

A importância da engenharia de requisitos é possibilitar a criação de uma base sólida para o projeto e a implementação correspondente (PRESSMAN, 2010). Sem essa base, o software construído, provavelmente, não atenderá as necessidades do cliente.

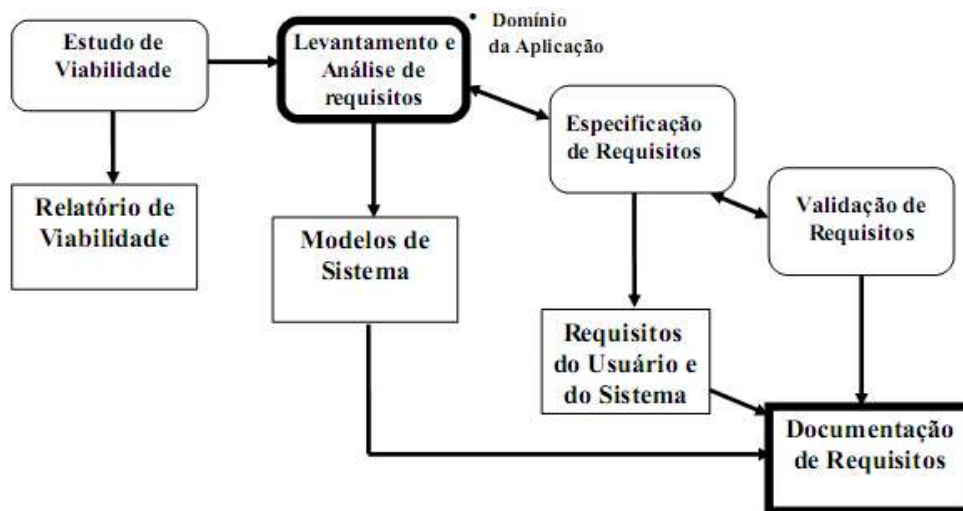


Figura 1: Engenharia de requisitos.

Na seção 2, requisitos de software são apresentados. Na seção 3, se refere ao estudo de viabilidade. Na seção 4, se aborda a elicitação e análise de requisitos. Na seção 5, a especificação de requisitos é apresentada, na seção 6, considerações sobre a validação de requisitos são realizadas, e finalmente na seção 7, o documento de requisitos é apresentado.

2. Requisitos

Segundo SOMMERVILLE (2007), os requisitos estão diretamente associados às necessidades dos clientes de um sistema. Os requisitos de um sistema definem o que o sistema deve fazer, descrições das funções, e as suas restrições operacionais.

Um requisito é uma característica do sistema ou uma descrição de algo que o sistema é capaz de fazer em relação a atender os propósitos do sistema (PFLEEGER, 1998).

Frequentemente os requisitos de sistemas de software são classificados em requisitos funcionais, requisitos não funcionais ou requisitos de domínio (SOMMERVILLE, 2007).

2.1 Requisitos funcionais

Os requisitos funcionais correspondem a declarações de serviços que o sistema deve fornecer (SOMMERVILLE, 2007).

Exemplos de requisitos funcionais: apresentar catálogo de produtos, registrar venda, e autorizar pagamento a crédito.

2.2. Requisitos não funcionais

São restrições sobre os serviços ou as funções oferecidos pelo sistema (SOMMERVILLE, 2007), incluem restrições sobre o tempo de resposta, sobre o processo de desenvolvimento e padrões.

Exemplos de requisito funcional: tempo de resposta apropriado nas interações com o sistema.

2.3 Requisitos de domínio

São derivados do domínio de aplicação do sistema, refletem os fundamentos do domínio da aplicação (SOMMERVILLE, 2007), se esses requisitos não forem satisfeitos, o sistema não funcionará satisfatoriamente.

Exemplo: Deve existir uma interface com o usuário-padrão para todos os bancos de dados e que deverá ser baseada no padrão Z39.50

2.4 Requisitos de usuário e de sistema

Também são considerados os requisitos de usuário e requisitos de sistema (SOMMERVILLE, 2007).

Os requisitos de usuário correspondem a descrição dos requisitos funcionais e não funcionais, de forma que sejam fácil de entender por parte dos usuários do sistema (não especialistas).

Os requisitos de sistema (SOMMERVILLE, 2007), correspondem a versões expandidas dos requisitos de usuário, adicionam detalhes e explicam como os requisitos de usuário devem ser fornecidos pelo sistema. São usados pelos engenheiros de software como ponto de partida para o projeto do sistema. Correspondem a especificação do sistema, isto é, o que. Eles não correspondem a como o sistema pode ser projetado ou implementado.

3. Estudo de viabilidade

O estudo de viabilidade representa o início do processo de engenharia de requisitos. O estudo de viabilidade tem como entrada os requisitos de negócio, um esboço da descrição do sistema e como o sistema pretende apoiar os processos de negócios (SOMMERVILLE, 2007).

O estudo de viabilidade (SOMMERVILLE, 2007) é um estudo que responde, principalmente, a seguinte questão: o sistema contribui para os objetivos gerais da organização?, outras questões que devem ser respondidas são: o sistema pode ser implementado com tecnologia atual e dentro das restrições definidas de custo e prazo?, e o sistema pode ser integrado a outros sistemas já implantados?

Se um sistema não contribui para os objetivos da organização, esse sistema não é relevante para a organização e para esse sistema deve ser realizada uma recomendação de não prosseguir no relatório de estudo de viabilidade (SOMMERVILLE, 2007).

4. Elicitação e análise de requisitos

O domínio da aplicação, serviços a serem fornecidos pelo sistema e restrições são compreendidos pelos engenheiros de software através das interações com os clientes e usuários (SOMMERVILLE, 2007). As atividades do processo de elicitação e análise de requisitos são (SOMMERVILLE, 2007):

1. Obtenção de requisitos: representa o processo de interação com os stakeholders¹ para coletar seus requisitos.
2. Classificação e organização de requisitos: Se refere a agrupar os requisitos relacionados em grupos coerentes.
3. Priorização e negociação de requisitos: atividade focada a priorização dos requisitos e à resolução de conflitos (devido a participação de vários stakeholders) via negociação.
4. Documentação de requisitos: implica a produção de documentos de requisitos formais ou informais.

Segundo SOMMERVILLE (2007), a elicitação e análise de requisitos são um processo iterativo (Figura 2). A compreensão dos requisitos por parte do analista aumenta a cada volta do ciclo do processo iterativo. Para cada volta do processo são realizadas as atividades: obtenção de requisitos, classificação e organização de requisitos, priorização e negociação de requisitos e documentação de requisitos.

¹ Stakeholder: termo usado para se referir a qualquer pessoa ou grupo afetado pelo sistema (exemplos: usuários finais, engenheiros, gerentes de negócios e especialistas do domínio).



Figura 2: Processo de elicitação e análise de requisitos.
Fonte: (SOMMERVILLE, 2007)

4.1 Obtenção de requisitos

Os requisitos de usuário e de sistema são obtidos através do processo que reúne informações sobre o sistema proposto (SOMMERVILLE, 2007). Várias fontes de informação são consideradas, nesta fase: documentos, usuários, analistas e especificações de sistemas similares.

Entrevistas são utilizadas nas interações com os stakeholders, também observações são úteis. O uso de cenários e protótipos auxilia na obtenção de requisitos (SOMMERVILLE, 2007).

4.1.1 Entrevista

A realização de entrevistas (formais ou informais) faz parte do processo de engenharia de requisitos. Os requisitos são obtidos a partir das respostas dos usuários ou clientes às questões colocadas pelo engenheiro de requisitos sobre o sistema existente ou novo sistema a ser desenvolvido (SOMMERVILLE, 2007).

A utilidade das entrevistas reside na compreensão geral de o que os usuários fazem, como interagem com o sistema, e dificuldades e necessidades atuais.

Em relação ao domínio da aplicação, é necessária a realização de entrevistas com um especialista do domínio, preferencialmente que faz parte da organização, ou a consulta de documentos ou informações pertinentes ao domínio.

As entrevistas possibilitam a obtenção de informações que complementam informações obtidas a partir de observações, documentos consultados, etc.

4.1.2 Cenários

Um cenário representa um exemplo de uma sessão de interação do usuário com o sistema de software. Cada cenário implica várias interações possíveis.

Na elaboração de um cenário se deve, principalmente, incluir (SOMMERVILLE, 2007):

1. Uma descrição do que os usuários esperam do sistema no início do cenário.
2. Uma descrição do fluxo normal (interações sem quaisquer entradas fora do normal ou condições de erros) de eventos do cenário.
3. Uma descrição do que pode dar errado (erros do usuário e falhas) como isso é tratado.
4. Informações sobre outras atividades que ocorrem simultaneamente.

A seguir um exemplo de cenário de fluxo normal (Figura 3) e um exemplo de cenário com exceções (Figura 4) no contexto de um caixa eletrônico de um banco (ATM)(RUMBAUGH ET AL., 1994) com acesso a vários bancos.

A ATM (caixa eletrônico de um banco 24 horas com acesso a vários bancos) solicita que o usuário introduza um cartão; o usuário introduz um cartão magnético. A ATM aceita o cartão e lê seu número de série. A ATM solicita a senha; o usuário introduz "1234". A ATM verifica o número de série e a senha com o consórcio; o consórcio faz a confirmação com o banco "39" e notifica a ATM sobre a aceitação. A ATM solicita que o usuário escolha o tipo de transação (retirada, depósito, transferência, consulta); o usuário especifica retirada. A ATM solicita a quantia em dinheiro; o usuário introduz 100. A ATM verifica que o valor está dentro dos limites pré-fixados e solicita que o consórcio processe a transação; o consórcio passa a solicitação para o banco; que normalmente confirma o sucesso e informa o novo saldo da conta. A ATM entrega o dinheiro e solicita que o usuário o recolha; o usuário recolhe o dinheiro.
--

Figura 3: Exemplo de cenário normal de um caixa eletrônico (ATM).

A ATM (caixa eletrônico de um banco 24 horas com acesso a vários bancos) solicita que o usuário introduza um cartão; o usuário introduz um cartão magnético.
A ATM aceita o cartão e lê seu número de série.
A ATM solicita a senha; o usuário introduz “9999”.
A ATM verifica o número de série e a senha com o consórcio, que os rejeita após consultar o banco apropriado.
A ATM indica senha inválida e solicita que o usuário a reintroduza; o usuário introduz “1234” que a ATM verifica com sucesso com o consórcio.
A ATM solicita que o usuário escolha o tipo de transação; o usuário escolhe retirada
A ATM solicita a quantia de dinheiro; o usuário muda de ideia e digita “cancelar”
A ATM ejeta o cartão e solicita que o usuário o recolha; o usuário assim faz.
A ATM solicita que o usuário introduza um cartão

Figura 4: Exemplo de cenário de um caixa eletrônico (ATM) com exceções.

4.1.3 Casos de uso

Os casos de uso representam uma técnica baseada em cenário para elicitación de requisitos (SOMMERVILLE, 2007).

Um caso de uso identifica as interações com o sistema. Um caso de uso abrange vários cenários. Existirá um cenário para representar a interação normal e outros para cada possível exceção (SOMMERVILLE, 2007).

A UML (Linguagem Unificada de Modelagem) (BOOCH ET. AL, 2005) possibilita a representação de casos de uso através do diagrama de casos de uso (BOOCH ET. AL, 2005) (Figura 5). No diagrama de casos de uso da Figura 6, foram colocados três casos de usos (Comprar Itens, Log in e Devolução de itens) (LARMAN, 2000) e dois atores (Caixa e Cliente). Um caso de uso descreve o que um sistema (ou subsistema) faz, define funcionalidades ou comportamentos fornecidos pelo sistema. Um caso de uso representa a descrição de um conjunto de sequências de ações (BOOCH ET. AL, 2005) para produzir um resultado. Um ator, no diagrama de casos de uso, representa um papel que um ser humano, um dispositivo de hardware ou até outro sistema desempenha quando interage com o sistema.

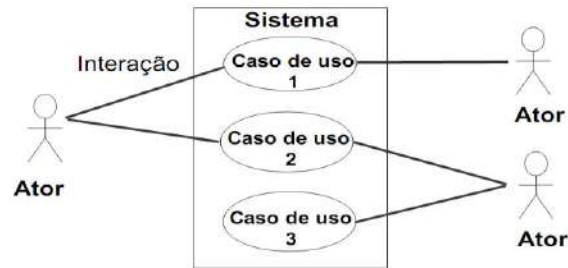


Figura 5: Diagrama de casos de uso segundo a notação da UML.

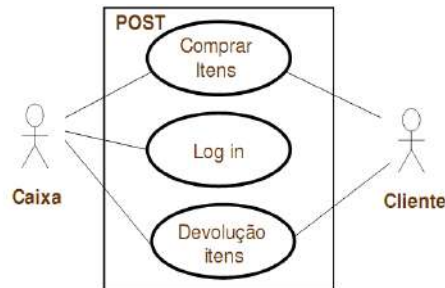


Figura 6: Exemplo de diagrama de casos de uso para um sistema de controle de venda de produtos no contexto de um supermercado.

Fonte: (LARMAN, 2000)

Um caso de uso pode ser descrito textualmente de forma geral ou alto nível (Figura 7) ou de forma detalhada ou expandida (Figura 8) (LARMAN, 2000).

Exemplo: Caso de uso de alto nível

Caso de Uso:	Comprar itens
Atores:	Cliente, Caixa
Descrição:	Um Cliente chega a um ponto de pagamento com itens que deseja comprar. O Caixa registra os itens da compra e recebe o pagamento. Quando concluída a compra, o cliente pode ir com os itens.

Figura 7: Descrição geral ou alto nível do caso de uso Comprar itens.

- **Caso de uso expandido (mais detalhes)**

Use case:	Comprar itens com dinheiro
Atores:	Cliente, Caixa
Finalidade:	Capturar uma venda e o pagamento correspondente.
Descrição:	Um cliente chega ao ponto de pagamento com itens que deseja comprar. O caixa registra os itens da compra e recebe um pagamento, o qual pode necessitar autorização. Quando concluída a compra o Cliente pode ir com os itens.

- **Sequência de eventos**

<u>Ação-Ator</u>	<u>Resposta - Sistema</u>
1. Começa quando um Cliente chega a um ponto de pagamento com vários itens	
2. O Caixa registra o identificador de cada item, também a quantidade	3. Determina o preço dos itens e acrescenta informação sobre o item à transação de vendas em andamento.
4. Quando concluída a entrada de itens O Caixa indica ao Post que a entrada de itens foi concluída	5. Calcula e mostra o Total da Venda

6. O Caixa informa ao Cliente o total
7. O Cliente faz pagamento em dinheiro
 - a - se pagamento em dinheiro - ver seção
Pagar com Dinheiro
 - b - se pagamento com cartão - ver seção
Pagar com Cartão de Crédito
 - c - se pagamento com cheque - ver seção
Pagar com Cheque
8. O Caixa registra a venda completada
9. Atualiza estoque
10. Gera um recibo
11. O Caixa dá ao Cliente o recibo da compra.
12. O Cliente sai com os itens comprados.

● Sequência de eventos - Seção pagar com Dinheiro

<u>Ação-Ator</u>	<u>Resposta - Sistema</u>
1. O Caixa faz um pagamento com dinheiro "Valor fornecido" possivelmente maior que o total da venda.	
2. O Caixa registra a quantia fornecida	
	3. Apresenta o valor do troco
4. O Caixa deposita o dinheiro recebido e retira o troco. O caixa dá o troco ao Cliente.	

Figura 8: Descrição detalhada ou expandida do caso de uso Comprar itens.

4.1.4 Negociação de requisitos

Acontece na realidade que clientes e usuários as vezes solicitam mais, em relação aos recursos disponíveis, do que é possível realizar. Também requisitos conflitantes podem ser propostos por diferentes usuários. Neste contexto é necessário um processo de negociação. Os envolvidos, clientes, usuários e outros interessados são convidados a participar de reuniões e discutir os conflitos sobre os requisitos e estabelecer prioridades nos requisitos (PRESSMAN, 2010).

5. Especificação de requisitos

Frequentemente a linguagem natural é usada para redigir especificações de requisitos de sistema como de usuário.

É necessário escrever os requisitos de usuário em uma linguagem que os não especialistas consigam compreender. Neste contexto, notações para a especificação de requisitos têm sido propostas (SOMMERVILLE, 2007), como: linguagem natural estruturada (todos os requisitos são definidos de forma padronizada), modelos gráficos dos requisitos como os casos de uso (seção 4.1.3), até especificações matemáticas.

No contexto da linguagem natural estruturada consiste na definição de formulários padrão para expressar a especificação de requisitos.

Um exemplo de formulário padrão para a especificação de requisitos é apresentado na Figura 9 e um exemplo de especificação de requisito, pagamento no contexto de um sistema de gerenciamento de vendas, usando esse formulário padrão é ilustrado na Figura 10.

Identificação do requisito:
Função:
Descrição: descrição da função
Entradas: descrição de entradas
Origem: descrição da origem da entrada
Saídas: descrição de saídas
Destino: para onde as saídas prosseguirão
Ação: descrição da ação a ser tomada
Requer: quais outras entidades são usadas
Precondição: o que deve ser verdadeiro antes da chamada da função
Pós-condição: o que é verdadeiro após a chamada da função

Figura 9: Um formulário padrão para a especificação de requisitos.

Identificação do requisito: R2.1 Função: Pagamento de uma compra de produtos a vista Descrição: Registrar o pagamento e imprimir o recibo Entradas: os itens (produtos) de uma venda Origem: Obtenção do preço dos produtos a partir da leitura do código de barras dos produtos Saídas: comprovante do pagamento Destino: Cliente Ação: Realizar o pagamento de produtos num supermercado, para isto realizar o registro correspondente no sistema, calcular o troco e imprimir o recibo. Requer: Precondição: Total da venda já foi calculado, verificar a disponibilidade do troco. Pós-condição: Pagamento foi realizado, atualização, no sistema, dos produtos vendidos e registro da venda realizada.

Figura 10: Exemplo de especificação do requisito pagamento usando o formulário padrão.

6. Validação de requisitos

A validação de requisitos objetiva mostrar que os requisitos realmente definem o sistema que o usuário deseja. Essa validação procura problemas com os requisitos, se encontrar erros em um documento de requisitos, devem ser corrigidos já que esses erros podem levar a custos excessivos de retrabalho, uma vez descobertos quando o sistema está em operação (SOMMERVILLE, 2007). Técnicas de requisitos podem ser usadas individualmente ou em conjunto: revisões de requisitos, protótipo e geração de casos de teste.

7. Documento de requisitos

O que deverá ser implementado pelos desenvolvedores é definido no documento de requisitos de software (SOMMERVILLE, 2007). Deve conter os requisitos de usuário e uma especificação detalhada dos requisitos de sistema.

Os usuários, possíveis, de um documento de requisitos são: clientes, gerentes, analistas e desenvolvedores participando na construção do sistema, testes e manutenção.

A estrutura proposta para um documento de requisitos, segundo o padrão IEEE/ANSI 830-1998, é apresentada na Figura 11.

- 1. Introdução**
 - 1.1 Propósito do documento de requisitos
 - 1.2 Escopo do produto
 - 1.3 Definições, acrônimos e abreviaturas
 - 1.4 Referências
 - 1.5 Visão geral do restante do documento
- 2. Descrição geral**
 - 2.1 Perspectiva do produto
 - 2.2 Funções do produto
 - 2.3 Características dos usuários
 - 2.4 Restrições gerais
 - 2.5 Suposições e dependências
- 3. Requisitos específicos** que abrangem requisitos funcionais, não funcionais e de interface.
Os requisitos podem estar relacionados a interface, funcionalidades, desempenho do sistema, restrições, e características de qualidade.
- 4. Apêndices**

Figura 11: A estrutura para um documento de requisitos, padrão IEEE/ANSI 830-1998.

8. Resumo

Este texto abordou a engenharia de requisitos.

Foi apresentado o que é um requisito, tipos de requisitos, a elicitação e a análise de requisitos.

Três técnicas de obtenção de requisitos foram apresentadas (entrevista, cenários e casos de uso).

Considerações sobre a validação de requisitos foram apresentadas.

Finalmente, foi apresentada a estrutura de um documento de requisitos segundo o padrão IEEE/ANSI 830-1998.

Referências Bibliográficas

Booch G., Rumbaugh J., Jacobson I. UML: Guia do usuário, 2ª edição. Editora Campus. 2005.

Larman C. Utilizando UML e Padrões: uma introdução à análise e ao projeto orientado a objetos. Editora Bookman. Porto Alegre, 2000.

Pfleeger S. L. Software Engineering: theory and practice. Prentice-Hall. 1998.

Pressman R. Engenharia de Software, 6a edição, AMGH Editora. 2010.

Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W. Modelage e Projetos Baseados em Objetos. Editora Campus. 1994.

Sommerville, I. Engenharia de Software. 8a Edição. Addison Wesley. 2007.

Exercícios propostos

1. No contexto de um sistema de tipo livreria virtual (venda de livros pela internet), por exemplo: o site da Eufal (<http://www.edufal.ufal.br>) ou um sistema utilizado pelos caixas nos supermercados (exemplo: Bom preço) para o gerenciamento da venda de produtos.

Analisar e refletir sobre o funcionamento de um site de tipo livreria virtual ou sobre o funcionamento de um sistema de supermercado para o gerenciamento da venda de produtos.

Identificar, só para um dos sistemas citados, os sete requisitos funcionais mais relevantes desse tipo de sistema.

2. Elaborar um diagrama, numa folha de papel, de casos de uso para os sete requisitos mais relevantes identificados no exercício 1.

3. Utilizar o editor de diagramas de casos de uso de uma ferramenta CASE para UML (Linguagem Unificada de Modelagem) e elaborar o diagrama de casos de uso do exercício 2.

4. Descrever de forma detalhada cada caso de uso contido no diagrama de casos de uso do exercício 3.

5. Refletir e elaborar um documento de requisitos para o sistema escolhido.

UNIVERSIDADE ABERTA DO BRASIL
UNIVERSIDADE FEDERAL DE ALAGOAS
Instituto de Computação
Curso de Bacharelado em Sistemas de
Informação

Engenharia de Software

Unidade 4: Análise de Sistemas e a
Linguagem Unificada de Modelagem
de Sistemas Orientados a Objetos

Prof. Arturo Hernández Domínguez

Conteúdo:

- 1. Introdução***
- 2. Análise de sistemas orientados a objetos***
- 3. A linguagem unificada de modelagem de sistemas orientados a objetos (UML)***

Exercícios



Este trabalho está licenciado com uma Licença [Creative Commons - Atribuição-NãoComercial-SemDerivações 4.0 Internacional](https://creativecommons.org/licenses/by-nc-nd/4.0/).

Setembro – 2010

1. Introdução

As principais fases no desenvolvimento orientado a objetos de um sistema, são (SOMMERVILLE, 2007) (PRESSMAN, 2010):

- Análise orientada a objetos;
- Projeto orientado a objetos;
- E programação orientada a objetos.

A análise orientada a objetos (SOMMERVILLE, 2007) se concentra a desenvolver um modelo orientado a objeto do domínio da aplicação. Os objetos identificados refletem entidades e operações associadas ao problema a ser resolvido.

O projeto orientado a objetos (SOMMERVILLE, 2007) se concentra a desenvolver um modelo orientado a objeto de um sistema de software para implementar os requisitos identificados. Os objetos em um projeto orientado a objetos estão relacionados à solução do problema.

A programação orientada a objetos (SOMMERVILLE, 2007) possibilita a implementação de um projeto de software usando uma linguagem de programação orientada a objetos. Uma linguagem de programação orientada a objetos permite a criação de objetos a partir de classes. Exemplos de linguagens de programação orientada a objetos: Java e C++.

Este documento aborda a análise de um sistema segundo a orientação a objetos e os modelos utilizados no contexto da notação padrão para sistemas orientados a objetos.

Na seção 2, se refere a análise de sistemas orientados a objetos. Na seção 3, a linguagem unificada de modelagem de sistemas orientados a objetos UML (Unified Modeling Language) (BOOCH ET. AL, 2005) e seus diagramas são apresentados. Na seção 4, é mostrado como representar uma arquitetura em camadas no contexto da UML.

2. Análise de sistemas orientados a objetos

A Análise Orientada a Objetos (AOO) consiste na criação de um modelo orientado a classes segundo os conceitos de Orientação a Objetos.

O objetivo da análise orientação a objetos é identificar todas as classes relevantes ao problema a ser resolvido e representar essas classes no modelo, desta forma serão representadas as operações e os atributos associados a elas, as associações entre elas e o comportamento associado.

Neste contexto, segundo PRESSMAN (2010), as tarefas seguintes devem ser realizadas:

1. Os requisitos do usuário precisam ser discutidos entre o cliente e o engenheiro de software.
2. As classes precisam ser identificadas (ou seja atributos e métodos são definidos).
3. Uma hierarquia de classes precisa ser especificada.
4. As conexões entre objetos devem ser representadas.
5. O comportamento do objeto precisa ser modelado.
6. As tarefas 1 a 5 são reaplicadas iterativamente até que o modelo seja completado.

3. A linguagem unificada de modelagem de sistemas orientados a objetos (UML)

Na Figura 1, é apresentada a evolução da linguagem unificada de modelagem de sistemas orientados a objetos UML (Unified Modeling Language) (BOOCH ET. AL, 2005). A UML resolveu o problema de falta de padronização da notação de modelagem de um sistema orientado a objetos, já que antes da padronização, foram propostos vários métodos orientados a objetos no período de fragmentação (Figura 1), com notações próprias e isto dificultava a comunicação entre analistas ou projetistas de organizações diferentes que usavam métodos diferentes com notações de modelagem de sistemas orientados a objetos diferentes.

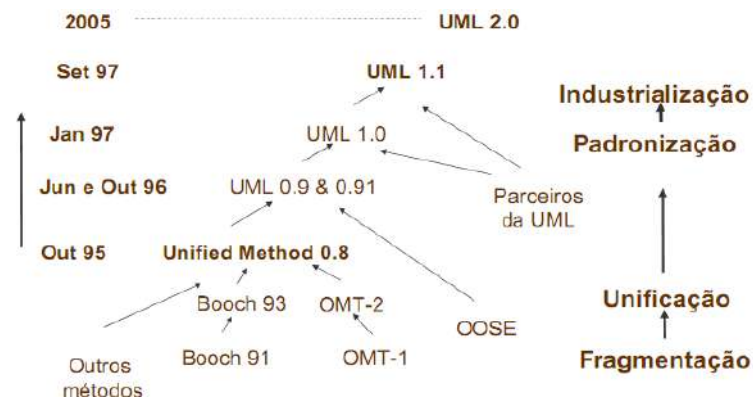


Figura 1: Evolução da UML.

Na especificação da UML se objetivou (BOOCH ET. AL, 2005):

- representar conceitual e fisicamente um sistema;
- linguagem de modelagem visual;
- mecanismos de extensibilidade;
- independente (de linguagens de programação e processos);
- encorajar o crescimento de ferramentas orientadas a objetos;
- integrar as melhores praticas.

A UML possibilita a visualização (através de modelos representados como diagramas que facilitam a comunicação), a especificação (através da elaboração dos modelos - diagramas) e a construção (mapeando os modelos da UML em linguagens de programação orientados a objetos).

A modelagem de um sistema (análise e projeto) pode ser documentada através dos modelos da UML.

3.1 Diagramas

Nesta seção, a maioria dos diagramas da UML (BOOCH ET. AL, 2005) é apresentada.

3.1.1 Diagrama de casos de uso

Um caso de uso (BOOCH ET. AL, 2005) descreve o que um sistema (ou subsistema) faz. Casos de uso definem funcionalidades ou comportamentos fornecidos pelo sistema. Os casos de uso representam os requisitos funcionais. Um caso de uso é uma descrição de um conjunto de ações para produzir um resultado. Um diagrama de caso de uso descreve o comportamento de um sistema percebido por atores externos (usuário, dispositivo ou outro sistema). Um ator representa um papel que um ser humano, um dispositivo de hardware ou até outro sistema desempenha quando interage com o sistema.

A Figura 2, apresenta um exemplo de diagrama de casos de uso no contexto de um sistema de um ponto de vendas, são representados três casos de uso (comprar itens, log in e devolução de itens) e dois atores (caixa e cliente).

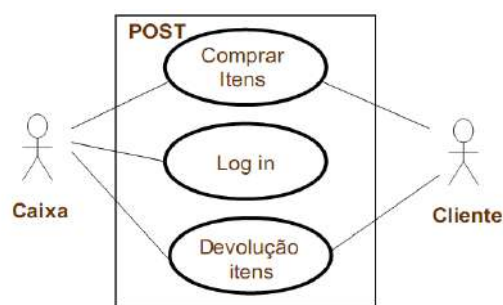


Figura 2: Diagrama de casos de uso no contexto de um sistema de ponto de vendas.
Fonte: (LARMAN, 2000)

3.1.2 Diagrama de sequência

Um diagrama de sequência (BOOCH ET. AL, 2005) é um diagrama de interação que dá ênfase à ordenação temporal das mensagens. Apresenta a sequência de mensagens enviadas entre objetos. As dimensões consideradas no diagrama de sequência são: na horizontal representa objetos diferentes e na vertical corresponde ao tempo. Na Figura 3, se apresenta um diagrama de sequência, no contexto do caso de uso empréstimo de um livro para um sistema de gerenciamento de uma biblioteca. Nesse diagrama de sequência, são representados o Usuário e os objetos Terminal, Cliente, Livro e Empréstimo e as mensagens entre esses objetos.

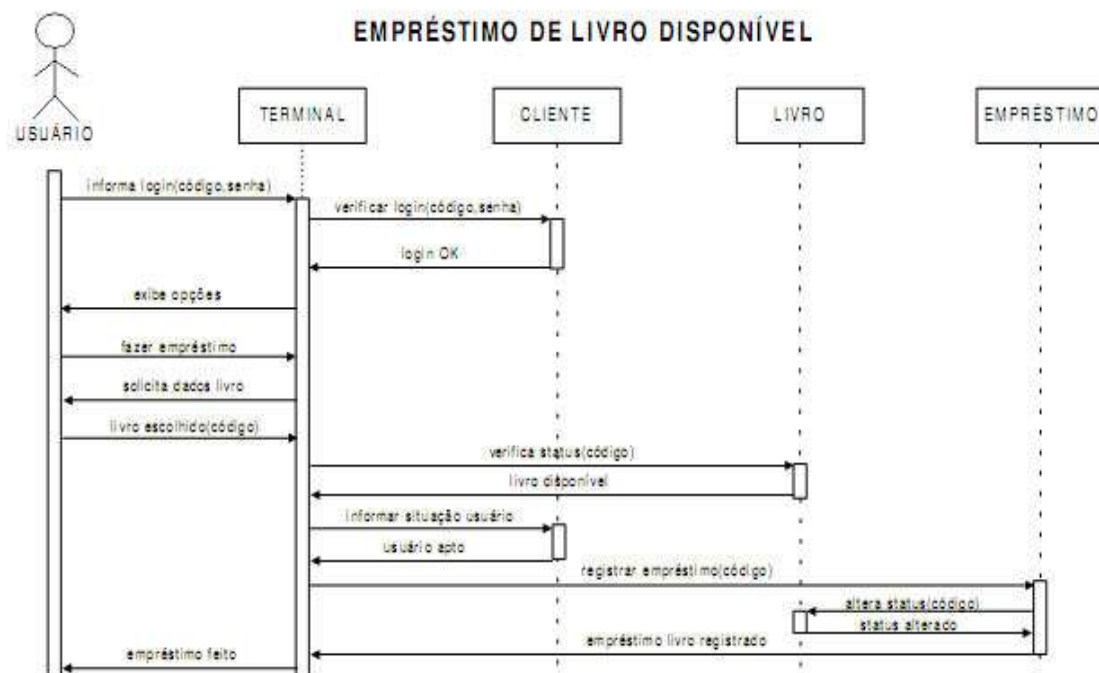


Figura 3: Diagrama de sequência no contexto do caso de uso empréstimo de um livro para um sistema de gerenciamento de uma biblioteca.
Fonte: (FIGUEIRA, 2001)

3.1.3 Diagrama de comunicação

Um diagrama de comunicação (BOOCH ET. AL, 2005) é um diagrama de interação que dá ênfase à organização dos objetos que participam de uma interação. Na Figura 4, se apresenta um diagrama de comunicação no contexto de um sistema de ponto de vendas.

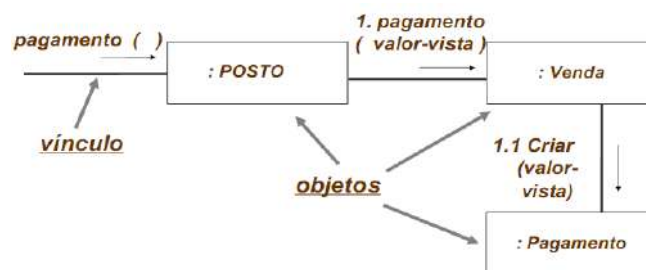


Figura 4: Diagrama de comunicação no contexto de um ponto de vendas.

3.1.4 Diagrama classes

Um diagrama de classes (BOOCH ET. AL, 2005) denota a estrutura estática de um sistema. Uma classe (Figura 5) é uma abstração e um objeto (Figura 5) é uma manifestação concreta ou instância dessa abstração.

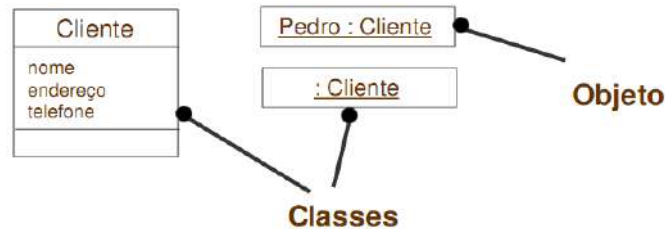


Figura 5: Representação de classes e objetos na UML.
Fonte: (BOOCH ET. AL, 2005)

Uma associação é um relacionamento estrutural que descreve um conjunto de conexões entre objetos (Figura 6). A agregação é um tipo particular de associação, representando um relacionamento estrutural entre o todo e suas partes (Figura 6).

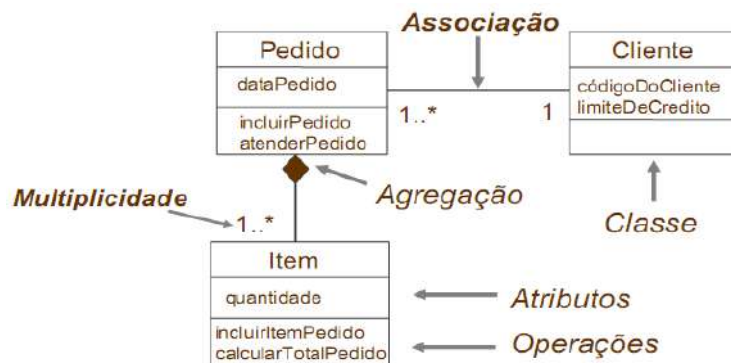


Figura 6: Diagrama de classes e uso da composição (agregação de composição) e associação.

A herança permite compartilhar atributos e métodos de uma classe (herança simples) ou de várias classes (herança múltipla) (Figura 7).

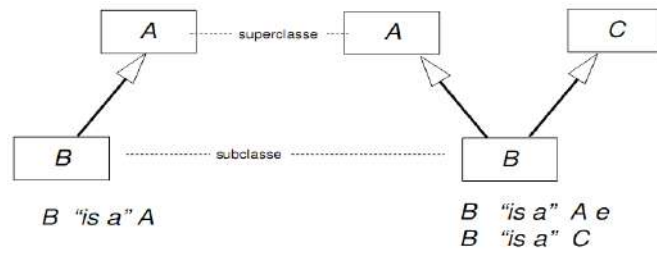


Figura 7: Herança simples e múltipla.

A Figura 8 apresenta um diagrama de classes com duas classes (Produto-perecível e Produto-não perecível) que especializam a classe Produto. A classe Produto tem uma associação com a classe Item.

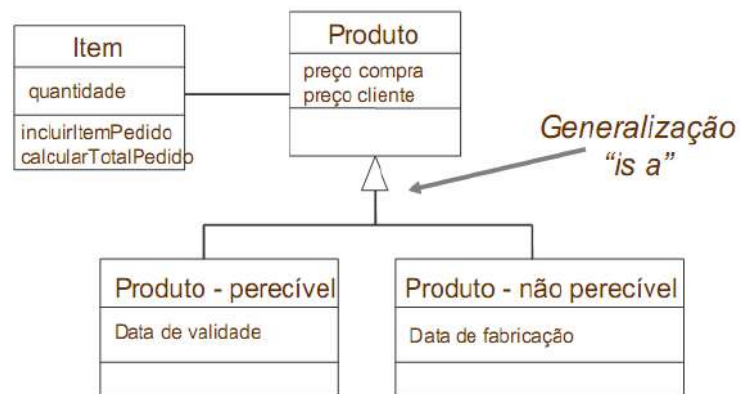


Figura 8: Diagrama de classes e uso da herança e associação.

A Figura 9 mostra um diagrama de classes no contexto de um sistema de gerenciamento de uma biblioteca (Figueira, 2001).

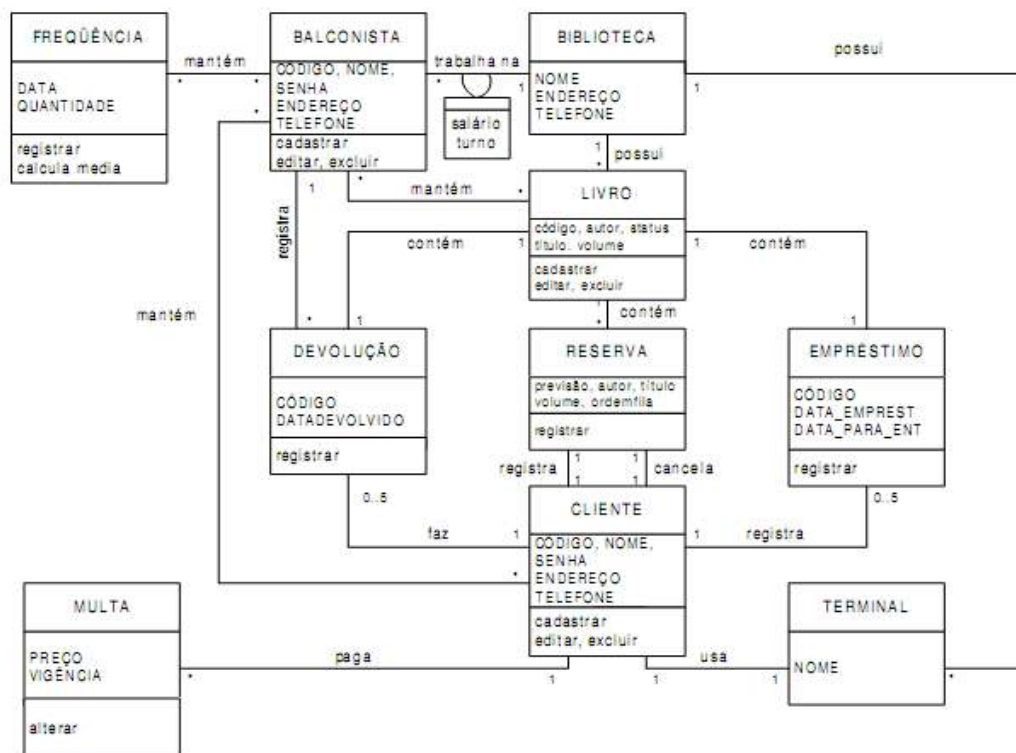


Figura 9: Exemplo de diagrama de classes no contexto de um sistema de gerenciamento de uma biblioteca. Fonte: (FIGUEIRA, 2001)

3.1.5 Diagrama de objetos

Os diagramas de objetos (BOOCH ET. AL, 2005) permitem a modelagem de instâncias de itens contidos em diagramas de classes. Um diagrama de objetos corresponde a uma instância de um diagrama de classes. Um diagrama de objetos é um retrato do sistema em determinado momento, nele é representado um conjunto de objetos e vínculos. Na Figura 11 é representado um diagrama de objetos associado ao diagrama de classes da Figura 10.

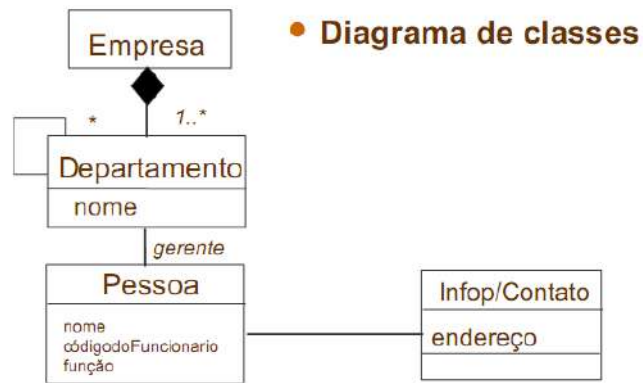


Figura 10: Diagrama de classes no contexto de uma empresa.
 Fonte: (BOOCH ET. AL, 2005)

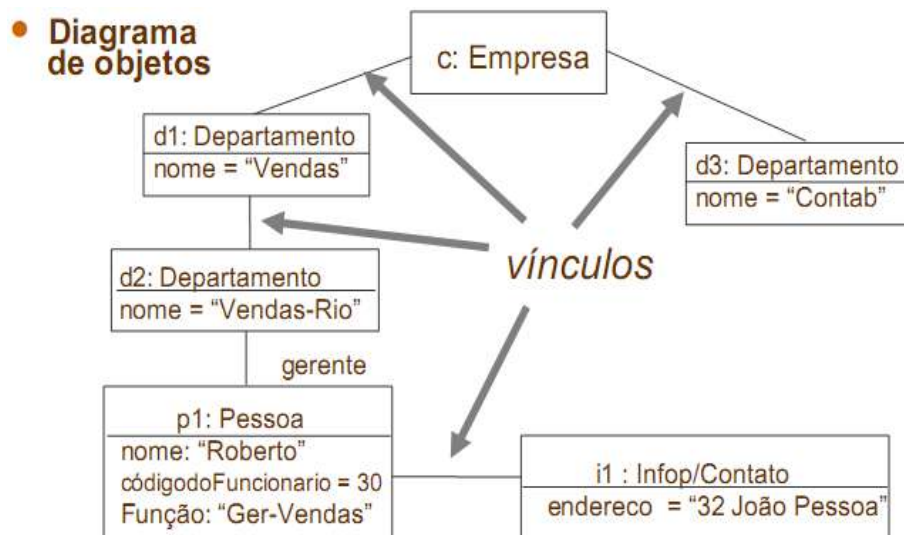


Figura 11: Exemplo de diagrama de objetos.
 Fonte: (BOOCH ET. AL, 2005)

3.1.6 Diagrama de estados

As máquinas de estados (BOOCH ET. AL, 2005) são empregadas para a modelagem dos aspectos dinâmicos de um sistema. Um diagrama de estados mostra as sequências de estados pelos quais um objeto passa durante seu tempo de vida em resposta a eventos. Um diagrama de estados é o complemento de uma classe. Um estado é uma condição ou situação na vida de um objeto durante a qual satisfaz alguma condição, realiza alguma atividade ou aguarda um evento (Figura 12). Um evento é a especificação de uma ocorrência. Um evento é a ocorrência de um estímulo capaz de

ativar uma transição de estado. Uma transição é um relacionamento entre dois estados, indicando que um objeto no primeiro estado realizará certas ações e entrará no segundo estado quando um evento especificado ocorrer. Uma atividade é uma execução em andamento em uma máquina de estados.

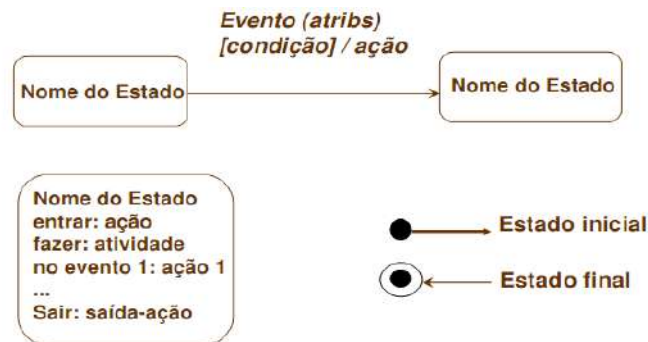


Figura 12: Notação utilizada em um diagrama de estados.

A Figura 13 apresenta um diagrama de estados para a classe switch ou interruptor. Os estados identificados são Off (desligado) e On (ligado).

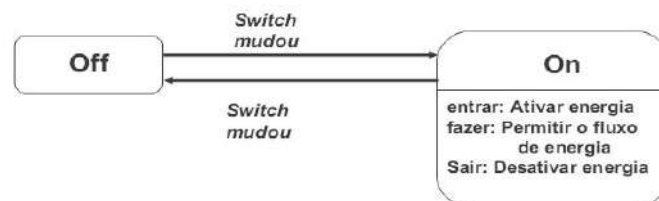


Figura 13: Diagrama de estados associado ao switch (interruptor).

Fonte: (RUMBAUGH ET. AL, 2005)

A Figura 14 apresenta um diagrama de estados para a classe menu instantâneo. Os estados identificados são Inativo e Menu visível.

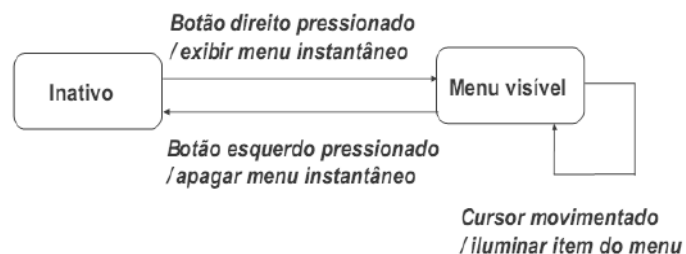


Figura 14: Diagrama de estados associado a um menu instantâneo.

Fonte: (RUMBAUGH ET. AL, 2005)

A Figura 15 apresenta um diagrama de estados para a classe livro no contexto de um sistema de gerenciamento de uma biblioteca.

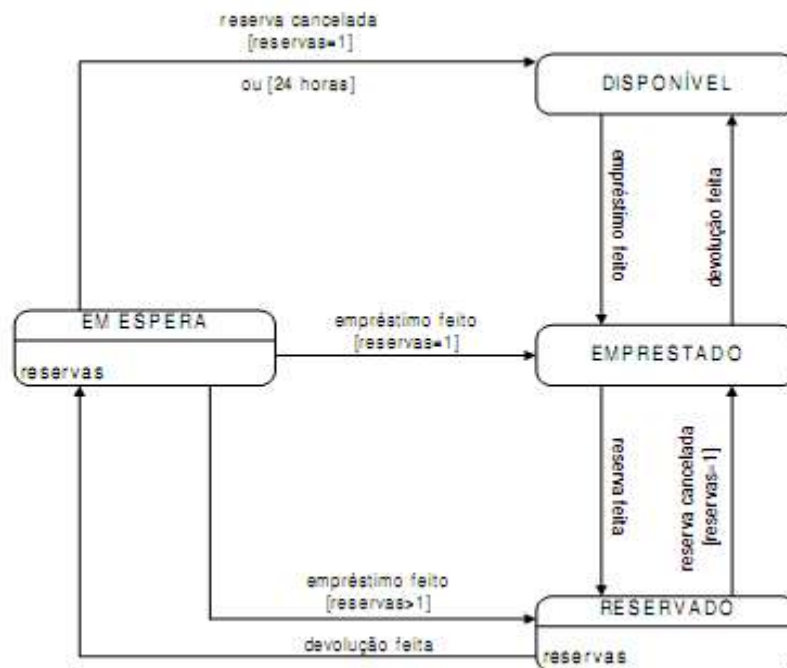


Figura 15: Diagrama de estados associado a classe livro no contexto de um sistema de biblioteca.
Fonte: (FIGUEIRA, 2001)

3.1.7 Diagrama de componentes

Ilustram a organização e dependências entre componentes de software. Um componente pode ser considerado como um bloco de construção para reuso. Um componente pode ser considerado como uma peça (unidade) de software reutilizável, que é desenvolvida de forma independente, pode ser utilizada junto com outros componentes para construir (compor) unidades de software maiores. Na Figura 16 é apresentado um diagrama de componentes (FURLAN ET. AL, 1998) com os seguintes componentes: Cobrança e sua interface Sistema de Cobrança, Pessoa e usa interface Usuário e o componente Registro.

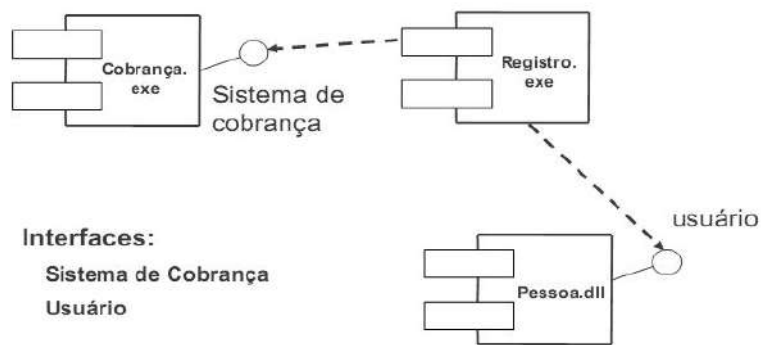


Figura 16: Exemplo de diagrama de componentes.

3.1.8 Diagrama de implantação

Um diagrama de implantação (BOOCH ET. AL, 2005) é um diagrama que mostra a configuração de nós de processamento em tempo de execução e os componentes que nele existem. Mostra os elementos de processamento e os componentes de software, processos e objetos associados. Inclui o uso físico do sistema considerando computadores, dispositivos e suas interconexões. A Figura 17 apresenta uma aplicação cliente servidor com dois nós de processamento e componentes associados.

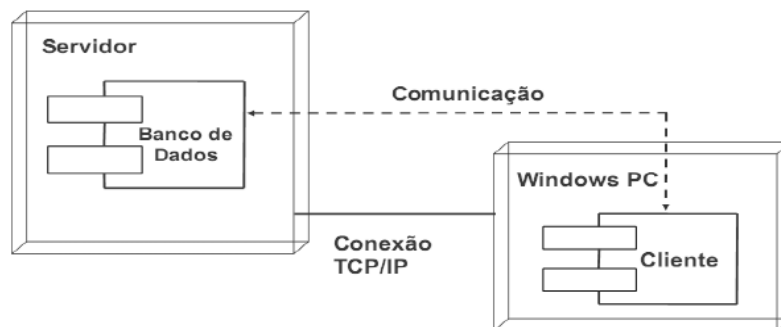


Figura 17: Exemplo de diagrama de implantação para uma aplicação cliente servidor.

A Figura 18 mostra um diagrama de implantação para um sistema de biblioteca internacional com cinco nós de processamento.

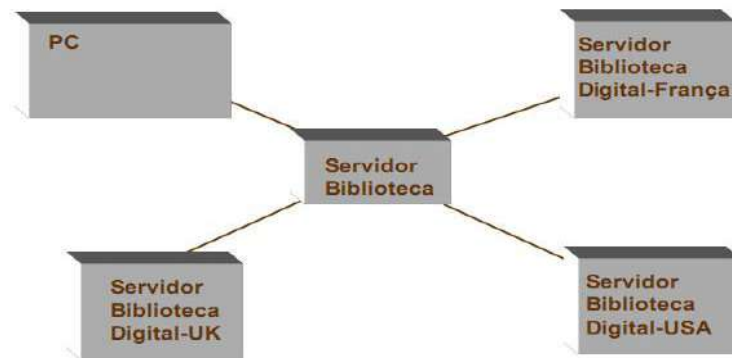


Figura 18: Diagrama de implantação no contexto de uma biblioteca internacional.

3.2 Representação de uma arquitetura em camadas no contexto da UML

Um pacote (BOOCH ET. AL, 2005) é um mecanismo para a organização de elementos (de modelagem) em agrupamentos. Um pacote pode conter outros elementos (classes, interfaces, componentes, casos de uso e até outros pacotes). Na Figura 19 os pacotes Pedido-Interface Usuário e Pedido-Aplicação representam agrupamentos de elementos.

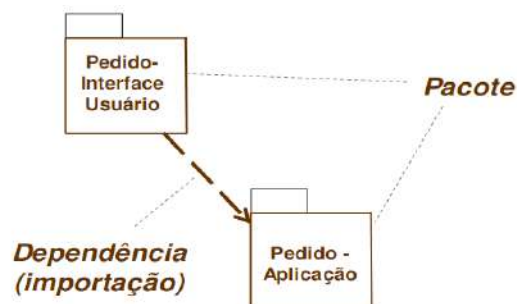


Figura 19: Uso de pacotes como agrupamentos de elementos.

Na Figura 20, se apresenta uma arquitetura em camadas, cada camada ou subsistema é representada através de um pacote.

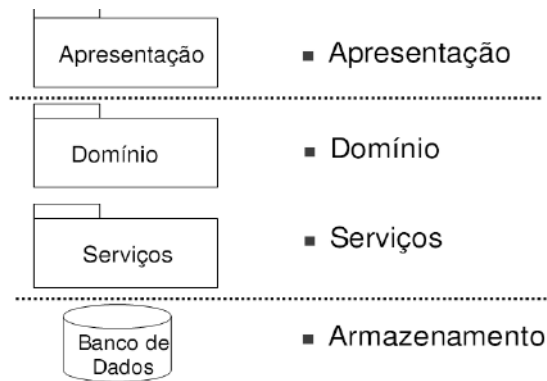


Figura 20: Arquitetura em camadas e uso do pacote para representar os subsistemas de um sistema.

Na Figura 21, ilustra um exemplo de arquitetura para um sistema de gerenciamento de um comercio. Observar que um subsistema pode conter outros elementos representados através de pacotes, por exemplo: O subsistema Domínio contém os pacotes Vendas, Pagamentos e Impostos.

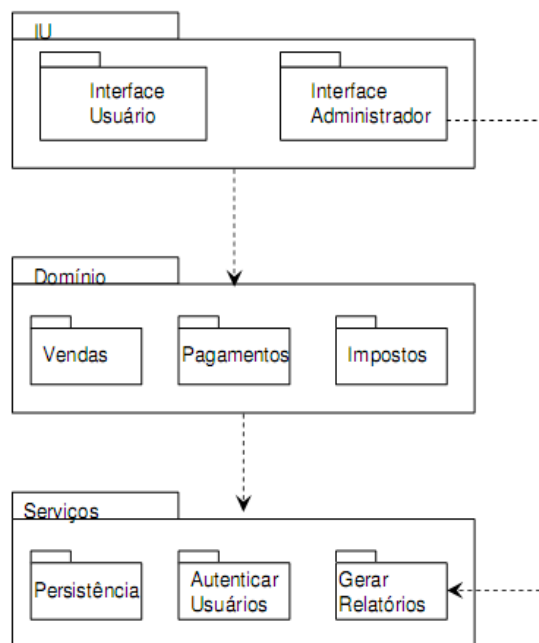


Figura 21: Exemplo de arquitetura em camadas no contexto de um sistema de comercio.

A Figura 22 ilustra um exemplo de arquitetura para um sistema de gerenciamento de uma biblioteca.

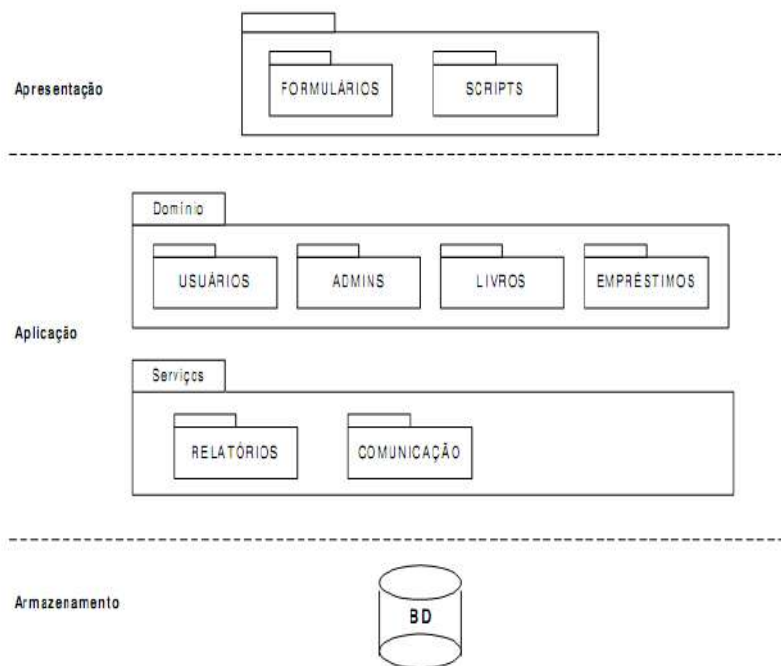


Figura 22: Exemplo de arquitetura para um sistema de gerenciamento de uma biblioteca.
Fonte: (FIGUEIRA, 2001)

5. Resumo

No contexto da modelagem de um sistema de software. Este texto abordou a análise de sistemas orientados a objetos e a linguagem unificada de modelagem de sistemas orientados a objetos (UML).

Foram apresentados os diagramas da notação UML: diagrama de casos de uso, diagrama de sequência, diagrama de comunicação, diagrama de classes, diagrama de objetos, diagrama de estados, diagrama de componentes e diagrama de implantação.

Também, foi apresentado, como representar uma arquitetura em camadas no contexto da UML.

Referências Bibliográficas

Booch G., Rumbaugh J., Jacobson I. UML: Guia do usuário, 2ª edição. Editora Campus. 2005.

Figueira R. A. R. B. Desenvolvimento de Um Sistema de Informação Acadêmico: Módulos GED e Biblioteca, Trabalho de Conclusão de Curso. Departamento de Tecnologia da Informação da Universidade Federal de Alagoas. 2001.

Furlan J. D. Modelagem de objetos através da UML. Makron Books do Brasil Editora, 1998.

Larman C. Utilizando UML e Padrões: uma introdução à análise e ao projeto orientado a objetos. Editora Bookman. Porto Alegre, 2000.

Pressman R. Engenharia de Software, 6a edição, AMGH Editora. 2010.

Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W. Modelagem e Projetos Baseados em Objetos. Editora Campus. 1994.

Sommerville, I. Engenharia de Software. 8a Edição. Addison Wesley. 2007.

Exercícios propostos

1. Analisar uma calculadora eletrônica básica (não científica), no contexto da construção de um sistema de software de tipo calculadora também básica, identificar os objetos envolvidos.
2. Refletir sobre o funcionamento de uma máquina para preparar café (cafeteira), no contexto da construção de um simulador (software) de cafeteira, identificar os objetos envolvidos.
3. Refletir sobre o funcionamento de um relógio, no contexto da construção de um sistema de software de tipo relógio, identificar os objetos envolvidos.
4. No contexto de um sistema de tipo livraria virtual (venda de livros pela internet), por exemplo: o site da Edufal (<http://www.edufal.ufal.br>) ou um sistema utilizado pelos caixas nos supermercados (exemplo: Bom preço) para o gerenciamento da venda de produtos.

Analisar e refletir sobre o funcionamento de um site de tipo livraria virtual ou sobre o funcionamento de um sistema de supermercado para o gerenciamento da venda de produtos.

Identificar os objetos para cada um dos sistemas citados.

5. Elaborar um diagrama de classes, para cada um dos sistemas do exercício anterior, utilizando um editor de diagramas de classes.

UNIVERSIDADE ABERTA DO BRASIL
UNIVERSIDADE FEDERAL DE ALAGOAS
Instituto de Computação
Curso de Bacharelado em Sistemas de
Informação

Engenharia de Software

Unidade 5: Projeto de Sistemas

Prof. Arturo Hernández Domínguez

Conteúdo:

- 1. Introdução***
- 2. Projeto de sistemas***
 - 2.1 Arquitetura***
 - 2.2 Controle e Subsistemas***
 - 2.3 Modelagem de um subsistema***

Exercícios



Este trabalho está licenciado com uma Licença [Creative Commons - Atribuição-NãoComercial-SemDerivações 4.0 Internacional](https://creativecommons.org/licenses/by-nc-nd/4.0/).

Setembro – 2010

1. Introdução

Este documento aborda, o projeto de um sistema segundo a orientação a objetos e os modelos utilizados no contexto da notação padrão, UML – Linguagem Unificada de Modelagem (BOOCH ET. AL, 2005), para sistemas orientados a objetos.

Na seção 2, se refere a projeto orientado a objetos. Essa seção é organizada em três subseções: arquitetura, controle e subsistemas, e modelagem de um subsistema.

2. Projeto

O foco do projeto orientado a objetos (SOMMERVILLE, 2007) é desenvolver modelo(s) orientado(s) a objeto(s) de um sistema de software para implementar os requisitos identificados. Os objetos em um projeto orientado a objetos estão relacionados à solução do problema.



Figura 1: A arquitetura e o projeto
Fonte: (SOMMERVILLE, 2007)

2.1 Arquitetura

Uma arquitetura se refere à estrutura de alto nível de subsistemas, componentes, interfaces e interações. Uma arquitetura representa a organização de um sistema de software. Um sistema é composto de vários subsistemas (Figura 2) que interagem para realizar as funcionalidades do sistema. Também, cada subsistema é composto de vários elementos. A construção de um sistema implica a construção de cada subsistema da arquitetura e posteriormente realizar a integração e interações desses subsistemas para obter o sistema funcionando.

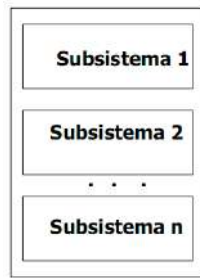


Figura 2: Sistema e subsistemas

2.1.1 Arquitetura cliente servidor

Conjunto de clientes que solicita os serviços oferecidos pelos servidores. Um cliente (Figura 3) faz um pedido a um servidor e espera até receber uma resposta (SOMMERVILLE, 2007).

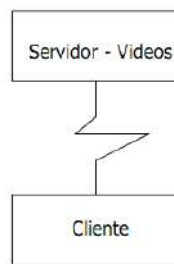


Figura 3: Exemplo de arquitetura cliente servidor

2.1.2 Arquitetura em camadas

Organiza um sistema em camadas (Figura 4), cada uma das quais fornecendo um conjunto de serviços (LARMAN, 2007). Permite o desenvolvimento incremental de subsistemas em diferentes camadas. Uma camada poderá ser substituída por outra equivalente desde que a interface permaneça inalterada.

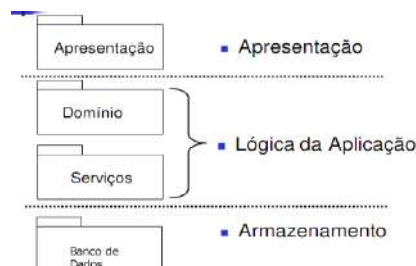


Figura 4: Arquitetura em camadas
Fonte: (LARMAN, 2007)

Um exemplo de arquitetura de arquitetura em camadas é representada na Figura 5, na qual foram representadas as camadas serviços, domínio, e apresentação (interface com o usuário).

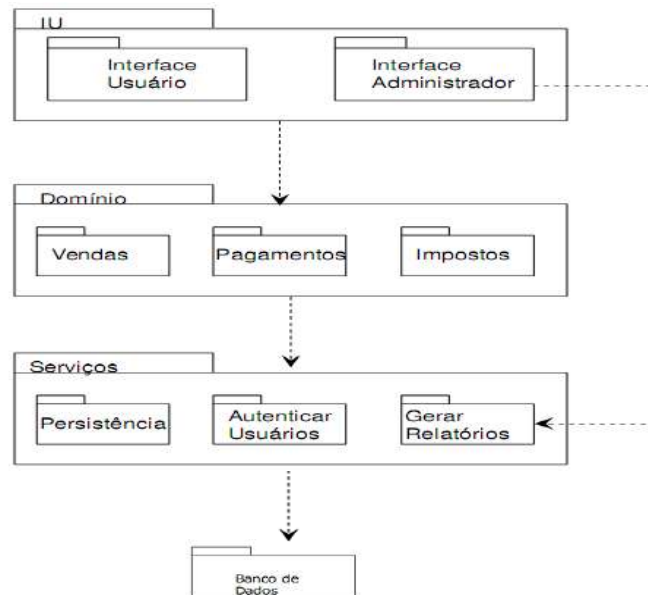


Figura 5: Exemplo de arquitetura em camadas

2.2. Controle e Subsistemas

Para funcionar como um sistema precisa-se de um controle e subsistemas (SOMMERVILLE, 2007). O controle lida com o fluxo de controle entre subsistemas (Figura 6).

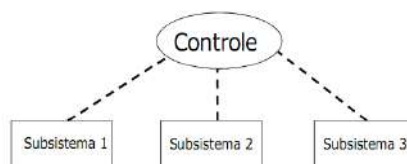


Figura 6: Representação do controle e subsistemas
Fonte: (SOMMERVILLE, 2007)

No contexto de um controle centralizado (SOMMERVILLE, 2007), um subsistema tem a responsabilidade geral pelo controle. Ele pode passar o controle a outro subsistema, mas esperará que o controle seja devolvido a ele (Figura 7).

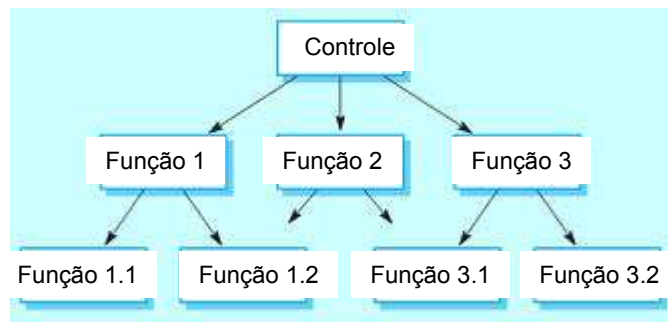


Figura 7: Controle centralizado

2.3 Modelagem de um subsistema

Uma vez definida a arquitetura de um sistema. Todos os subsistemas e interações devem ser especificados para posteriormente implementar cada subsistema. A construção de um sistema requer a modelagem e implementação de cada subsistema e interações identificados na arquitetura. Cada subsistema deve ser testado e integrado para posteriormente realizar testes de integração e validar o sistema integrado.

No contexto da orientação a objetos, será apresentada a modelagem para um sistema de criação de mapas meteorológicos, para cada subsistema da arquitetura (Figura 8) representado através do pacote, notação UML (BOOCH ET. AL, 2005), um pacote pode conter outros pacotes (Figura 9), se deve realizar o diagrama de casos de uso (Figura 11) e a especificação de cada caso de uso (conforme a unidade engenharia de requisitos), se devem identificar os objetos (Figura 12) que fazem parte desse subsistema e elaborar um diagrama de classes. As interações dos objetos também podem ser modeladas através de um diagrama de sequencia. Para cada classe relevante em um sistema orientado a objetos é desejável a elaboração de um diagrama de estados. Um subsistema tem uma interface (Figura 13) e as interações inter-subsistemas acontecem através das interfaces.

No contexto da arquitetura de um sistema de criação de mapas meteorológicos, quatro subsistemas foram identificados (Figura 8): coleta dados, processamento de dados, arquivamento de dados e apresentação.

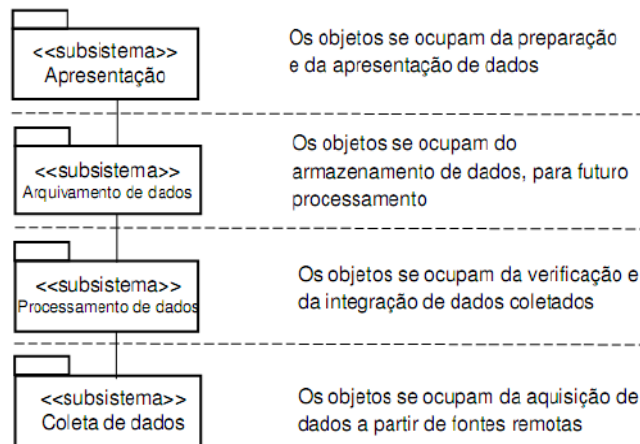


Figura 8: Arquitetura em camadas no contexto de um sistema de criação de mapas meteorológicos
Fonte: (SOMMERVILLE, 2007)

O subsistema coleta de dados (Figura 9) contém cinco pacotes (observador, satélite, estação meteorológica, comunicações e balão).

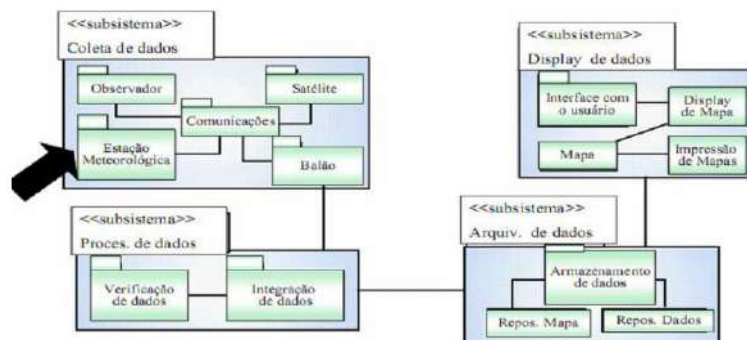


Figura 9: Subsistemas (pacotes) do sistema de criação de mapas meteorológicos
Fonte: (SOMMERVILLE, 2007)

O pacote estação meteorológica (Figura 10) contém três subsistemas (instrumentos, coleta de dados, e interface).

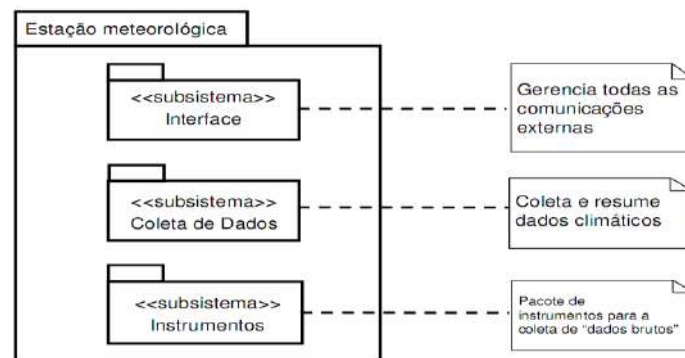


Figura 10: Os subsistemas da estação meteorológica
Fonte: (SOMMERVILLE, 2007)

No diagrama de casos de uso (Figura 11) do subsistema estação meteorológica, foram especificados os seguintes casos de uso: iniciar, desativar, relatar, calibrar e testar.

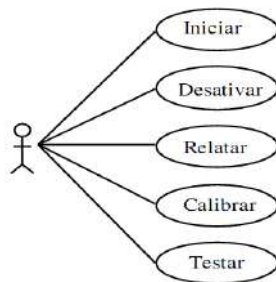


Figura 11: Diagrama de casos de uso do subsistema estação meteorológica
Fonte: (SOMMERVILLE, 2007)

As classes identificadas para os subsistemas interface (Estação Meteorológica), coleta de dados (Dados Meteorológicos) e instrumentos (Termômetro do solo, Anemômetro, Barômetro) são rerepresentadas na Figura 12.

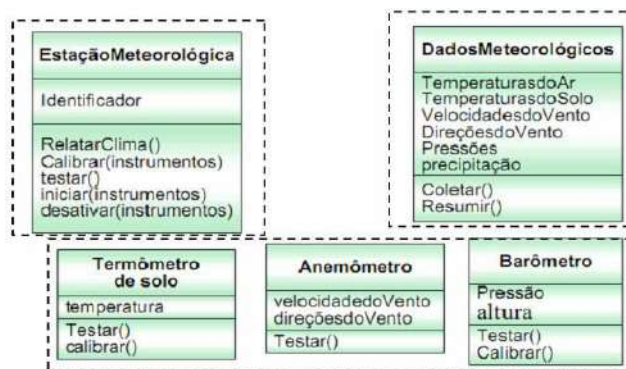


Figura 12: Classes no contexto do subsistema estação meteorológica
Fonte: (SOMMERVILLE, 2007)

A interface associada ao subsistema estação meteorológica é representada na Figura 13.

```

interface Estação Meteorológica {

    public void EstaçãoMeteorológica (); // contrutor

    public void Iniciar (); //iniciar estação
    public void Iniciar (Instrumento i);

    public void desativar (); //desativar estação
    public void desativar(Instrumento i);

    public void relatarClima ( );

    public void testar (); //testar estação
    public void testar ( Instrumento i );

    public void calibrar ( Instrumento i );

    public int obterID ();

} // EstaçãoMeteorológica
  
```

Figura 13: A interface do subsistema estação meteorológica
Fonte: (SOMMERVILLE, 2007)

3. Resumo

No contexto do projeto de um sistema orientado a objetos. Foram apresentados os seguintes tópicos: arquitetura, controle e subsistemas, e a modelagem de um subsistema. Um exemplo de modelagem foi apresentado no contexto da modelagem do subsistema de estação meteorológica para um sistema de criação de mapas meteorológicos.

Referências Bibliográficas

Booch G., Rumbaugh J., Jacobson I. UML: Guia do usuário, 2ª edição. Editora Campus. 2005.

Larman C. Utilizando UML e Padrões: uma introdução à análise e ao projeto orientado a objetos. Editora Bookman. Porto Alegre, 2000.

Pressman R. Engenharia de Software, 6a edição, AMGH Editora. 2010.

Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W. Modelagem e Projetos Baseados em Objetos. Editora Campus. 1994.

Sommerville, I. Engenharia de Software. 8a Edição. Addison Wesley. 2007.

Exercícios propostos

1. No contexto de um sistema de tipo livraria virtual (venda de livros pela internet), por exemplo: o site da Edufal (<http://www.edufal.ufal.br>) ou um sistema utilizado pelos caixas nos supermercados (exemplo: Bom preço) para o gerenciamento da venda de produtos.

Elaborar uma arquitetura lógica. Identificar e colocar, usando a notação de pacote, os subsistemas da arquitetura.

2. Elaborar uma arquitetura (lógica) em camadas para um sistema que deverá ser proposto por você. Identificar e colocar, usando a notação de pacote, os subsistemas da arquitetura.

UNIVERSIDADE ABERTA DO BRASIL
UNIVERSIDADE FEDERAL DE ALAGOAS
Instituto de Computação
Curso de Bacharelado em Sistemas de
Informação

Engenharia de Software

Unidade 6: Gerenciamento de Projetos
de Sistemas de Software

Prof. Arturo Hernández Domínguez

Conteúdo:

- 1. Introdução***
- 2. Gerenciamento de Projetos de Sistemas de Software***
 - 2.1 Atividades de gerenciamento***
 - 2.2 Planejamento de projeto***
 - 2.3 Cronograma do projeto***

Exercícios



Este trabalho está licenciado com uma Licença [Creative Commons - Atribuição-NãoComercial-SemDerivações 4.0 Internacional](https://creativecommons.org/licenses/by-nc-nd/4.0/).

Outubro – 2010

1. Introdução

O gerenciamento de projetos de software é muito importante no contexto da engenharia de software. Ser o gerente de software em uma organização é um dos possíveis cargos que um engenheiro de software poderá assumir profissionalmente. Um gerente que atua com pessoas, processos, recursos diversos, etc (REZENDE, 2002).

Os gerentes de software realizam um trabalho similar a aquele que outros gerentes de outras áreas da engenharia fazem. No entanto, lembrar que a engenharia de software é diferente de outros tipos de engenharia, principalmente devido a que o software é intangível e não pode ser visto ou tocado, e isto dificulta o gerenciamento de software (SOMMERVILLE, 2007). No contexto da construção de um projeto de engenharia civil, o gerente do projeto pode ver o andamento da construção e no contexto do gerente de software precisa interagir com os projetistas para obter informações (relatórios e documentos que descrevem o estado do software que é desenvolvido) necessárias para examinar o progresso do projeto.

Na seção 2, se refere a gerenciamento de projetos de sistemas de software. Essa seção é organizada em três subseções: atividades de gerenciamento, planejamento de projeto e cronograma do projeto.

2. Gerenciamento de Projetos de Sistemas de Software

No contexto do desenvolvimento de um sistema de software, um mau gerenciamento, geralmente, provocará em falhas no projeto, isto é acontecerá atraso na entrega do sistema, o custo aumentará, e haverá falhas no atendimento dos requisitos (SOMMERVILLE, 2007). A gestão efetiva de projetos de software foca: pessoal, produto de software, processo (modelo de processo mais apropriado) e projeto (plano de projeto) (PRESSMAN, 2010).

A responsabilidade do desenvolvimento de planos e cronogramas do projeto é do gerente de software. Também os gerentes devem supervisionar o desenvolvimento do software para verificar o uso de padrões exigidos e se o trabalho está dentro do prazo e orçamento previstos.

Uma equipe de desenvolvimento de software típica consiste em (BEZERRA, 2007): um gerente, analistas (devem ter conhecimento do domínio do negócio), projetistas

(avaliam alternativas de solução do problema e especificam uma solução computacional detalhada), arquiteto de software (elabora a arquitetura do sistema e identifica quais são os subsistemas que compõem o sistema assim como as interfaces entre esses subsistemas), programadores (responsáveis pela implementação do sistema), cliente usuário (especialista do domínio), cliente contratante e avaliadores de qualidade (asseguram o uso de padrões de qualidade estabelecidos pela organização).

2.1 Atividades de gerenciamento

A seguir uma lista de atividades que são realizadas, algumas ou todas, pela maioria dos gerentes de projetos de software:

Elaboração de proposta

A proposta descreve os objetivos do projeto e como ele será realizado. Geralmente inclui a estimativa de custos, o cronograma e justificativa. A elaboração de propostas é muito importante já que a elaboração de propostas, para muitas organizações de software, implica a possibilidade de realizar um contrato, isto é fundamental para a existência dessas organizações.

Planejamento e desenvolvimento do cronograma do projeto

O planejamento de projeto implica identificar as atividades, marcos (ver a seção 2.2.1 deste documento) e produtos gerados por um projeto (SOMMERVILLE, 2007). É necessária a elaboração de um plano de projeto (ver a seção 2.2.1 deste documento) para guiar o desenvolvimento em direção aos objetivos do projeto.

Custo de projeto

A estimativa de custos é uma atividade relacionada à estimativa dos recursos necessários para realizar o plano do projeto.

Monitoramento e revisões de projetos

O gerente deve realizar o acompanhamento do progresso do projeto e comparar com o planejamento previsto, também deve ser monitorado os custos reais utilizados e os planejados. A maioria das organizações definem mecanismos formais de monitoração. Também, o gerente pode obter informações de forma informal, por exemplo através de discussões informais (por exemplo no canto do cafezinho ou no corredor) com alguns

membros da equipe, isto pode ser importante já que um problema, por exemplo defeito de software, poderá ser identificado, na conversa informal com algum membro da equipe, antes de ser relatado e agir rapidamente, objetivando resolver esse problema, alocando um especialista para resolver, e evitar um atraso no cronograma.

As revisões de gerenciamento do projeto objetivam a revisão geral do progresso e o desenvolvimento técnico do projeto e a verificação se o projeto e as metas da organização que está financiando o software estão alinhadas (SOMMERVILLE, 2007).

Seleção e avaliação de pessoal

O gerente de software precisa selecionar pessoas para trabalhar nos projetos. O ideal é que exista a melhor equipe possível, isto é, pessoas com experiência adequada para trabalhar no projeto (SOMMERVILLE, 2007). Na prática, isto não necessariamente acontecerá, então será preciso trabalhar com uma equipe disponível naquele momento, isto poderá acontecer devido a restrições no orçamento, ou a equipe ou pessoal com experiência não estará disponível para o período do projeto ou a organização quer desenvolver as habilidades de uma equipe inexperiente. Em relação a organização, o desenvolvimento de habilidades de uma equipe experiente, isto é tornar uma equipe inexperiente em uma equipe experiente é muito importante, já que isto vai possibilitar que depois de um período, a equipe terá condições de atender tecnicamente, de forma satisfatória, as necessidades da organização em termos de desenvolvimento de sistemas de software.

Elaboração de relatórios e apresentações

Os gerentes de projeto devem se comunicar eficientemente, de forma verbal e escrita (SOMMERVILLE, 2007). Eles devem se comunicar com clientes, pessoas da alta gerencia, e com a equipe de desenvolvimento, para cada interação, o gerente de projeto deve comunicar considerando a pessoa com quem ele interage. Por exemplo, se o gerente fala com a equipe de desenvolvimento, então ele deve falar tecnicamente de forma a se entender com o pessoal de desenvolvimento, agora se o gerente fala com a alta gerencia, ele deve falar de tal forma (objetiva e clara - destacando as informações relevantes) que possa ser compreendido sem precisar aprofundar nos detalhes técnicos, já que pode ser difícil estabelecer um dialogo técnico com as pessoas da alta gerencia que geralmente não terão conhecimento técnico aprofundado. Também é importante

lembrar que a alta gerencia não dispõe de muito tempo, então o tempo disponível nas reuniões com as pessoas da alta gerencia deve ser utilizado da melhor forma possível. Os gerentes devem preparar relatórios e apresentações sobre o projeto para o cliente e direção da organização dele ou contratante. Eles devem redigir documentos concisos e coerentes que resumam as informações contidas em relatórios detalhados do projeto.

2.2 Planejamento de projeto

O gerenciamento eficiente de um projeto de software depende de um planejamento detalhado do progresso do projeto (SOMMERVILLE, 2007).

O plano elaborado inicialmente deve ser utilizado como guia que deve evoluir em função do avanço do projeto e mais informações se tornarão disponíveis. À medida que as informações se tornam disponíveis, durante a realização do projeto, o plano deve ser revisado.

O planejamento de projeto para desenvolvimento de software é um processo iterativo representado na Figura 1 (SOMMERVILLE, 2007), apenas concluído quando o próprio projeto é concluído.

- *Estabeleça as restrições (data de entrega, pessoal disponível, orçamento, ...) do projeto*
- *Faça a avaliação inicial dos parâmetros (estrutura, tamanho, ...) do projeto*
- *Defina os marcos do projeto e os produtos a serem entregues*
- *while projeto não foi concluído ou cancelado loop*
 - Elabore um cronograma do projeto*
 - Inicie as atividades de acordo com o cronograma*
 - Aguarde (por um período)*
 - Examine o progresso do projeto*
 - Revise as estimativas de parâmetros do projeto*
 - Atualize o cronograma do projeto*
 - Renegocie as restrições do projeto e os produtos a serem entregues*
 - if surgirem problemas then*
 - Inicie revisão técnica*
 - end if*

Fonte: (Sommerville, 2007)

Figura 1: Planejamento do projeto – processo iterativo

2.2.1 O plano de projeto

O plano de projeto estabelece os recursos disponíveis para o projeto, a estrutura analítica do projeto e um cronograma para a realização do trabalho.

A estrutura do plano para o processo de desenvolvimento, na maioria das vezes deve incluir as seguintes seções (SOMMERVILLE, 2007):

Introdução

Descrição dos objetivos do projeto e estabelece as restrições, tais como orçamento, prazo, etc que afetam o gerenciamento do projeto.

Organização do projeto

Descreve o modo como a equipe de desenvolvimento está organizada, pessoas envolvidas e seus papéis na equipe.

Análise de riscos

Descrição dos possíveis riscos¹ do projeto e as propostas de estratégias de redução de riscos (SOMMERVILLE, 2007).

Requisitos de recursos de hardware e de software

Descrição de hardware e software necessários para realizar o desenvolvimento. Se for necessário realizar a aquisição de recursos, então devem ser realizadas as estimativas de preços e prazos de entrega.

Estrutura analítica

Estabelece a estrutura analítica do projeto em atividades e identifica os marcos e os produtos a serem entregues com cada atividade.

Um marco é um ponto final reconhecível de uma atividade do processo de software. A cada marco, deve existir uma saída formal, como um relatório, que possa ser apresentado à gerência. Por exemplo, no contexto do processo de requisitos, podemos identificar como marcos: relatório de viabilidade, requisitos de sistema. Os relatórios

1 Risco: é um evento não desejado que provoca consequências negativas (PFLEEGER, 1998)

dos marcos, também, não precisam ser documentos extensos, podem ser simples e breves sobre o que foi concluído.

Um produto é um resultado de projeto entregue ao cliente. É geralmente disponibilizado no fim de alguma fase importante do projeto, como a especificação de requisitos, análise ou projeto.

Os marcos podem ser resultados internos do projeto que não serão entregues ao cliente. Neste caso os marcos são usados pelo gerente do projeto para verificar o progresso.

Cronograma do projeto

Apresenta as atividades, prazo estimado para realizar cada atividade, dependências entre atividades e a alocação de pessoas nas atividades.

Mecanismos de monitoração e elaboração de relatórios

Consiste na definição de relatórios de gerenciamento que devem ser produzidos, quando devem ser produzidos e os mecanismos de monitoração de projeto usados.

2.3 Cronograma do projeto

O cronograma representa a divisão do trabalho total de um projeto em atividades separadas e a estimativa do tempo necessário para completar essas atividades. Deve ser considerado no cronograma os recursos necessários (pessoas envolvidas e outros) para realizar cada atividade. Os cronogramas são geralmente representados através de diagramas que mostram a duração e dependências de atividades, e alocação de pessoal.

Um diagrama (rede) de atividades mostra as dependências entre as diferentes atividades que constituem um projeto. No exemplo do diagrama da Figura 2, são mostradas as atividades ou tarefas (representadas através de um retângulo), a duração (em dias) associada e dependências (por exemplo a tarefa T3 depende de T1 – M1).

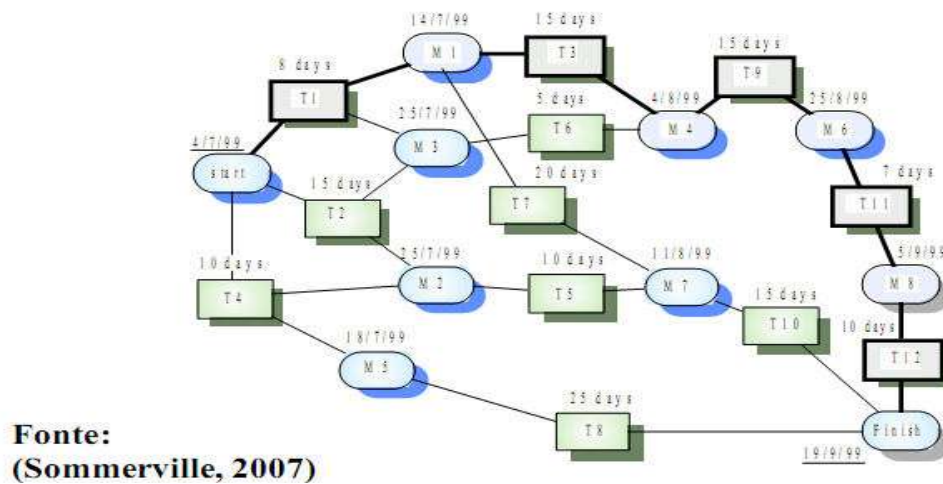


Figura 2: Exemplo de diagrama (rede) de atividades.

Um diagrama de barras² mostra quando uma atividade começa e quando termina. Também no diagrama de barras pode ser mostrada a alocação de pessoal às atividades do projeto de software.

A seguir são apresentados dos exemplos de cronograma.

Exemplo 1 de cronograma

Na Figura 3, é apresentado um exemplo de cronograma, no contexto de um projeto de software, mostrando as atividades e o período de tempo necessário para a realização das mesmas. A duração prevista para esse exemplo de projeto de software é de um ano e as atividades consideradas são: engenharia de requisitos, análise, projeto, implementação, teste de unidade e subsistemas, testes de integração, validação do sistema, treinamento de usuários e liberação do sistema. Para cada atividade, no cronograma da Figura 3, deve ser colocado o nome do responsável pela atividade. Em algumas atividades também pode ser colocado os nomes dos envolvidos na realização da atividade, por exemplo no contexto da engenharia de requisitos (colocar o cliente e/ou usuários envolvidos), validação do sistema (colocar o cliente e/ou usuários envolvidos), treinamento (colocar os usuários ou setor da organização envolvido) e liberação do sistema (colocar o cliente e/ou usuários envolvidos).

² Diagrama de barras: também chamado de diagrama de Gantt. Mostra quando uma atividade começa e termina. Deve ser lido de esquerda a direita.

Período	1º Trimestre			2º Trimestre			3º Trimestre			4º Trimestre		
Atividades	1	2	3	4	5	6	7	8	9	10	11	12
Engenharia de Requisitos <i>Nome do responsável: A</i>												
Análise <i>Nome do responsável: B</i>												
Projeto <i>Nome do responsável: C</i>												
Implementação <i>Nome do responsável: D</i>												
Testes de unidade e subsistemas <i>Nome do responsável: E</i>												
Testes de integração <i>Nome do responsável: F</i>												
Validação de sistema <i>Nome do responsável: G</i>												
Treinamento de usuários e disponibilização de documentação <i>Nome do responsável: H</i>												
Liberação do sistema <i>Nome do responsável: I</i>												

Figura 3: Exemplo de cronograma para o desenvolvimento de um sistema de software

Exemplo 2 de cronograma

No formato de cronograma da Figura 4 (ainda sem a estimativa do tempo para cada atividade), é considerado um desenvolvimento incremental³ de software, para isto nesse exemplo são considerados três incrementos no contexto da construção do sistema de software. Após a realização da especificação de requisitos, para cada incremento, as tarefas a serem realizadas são: análise, projeto, implementação e testes de unidade, incremento e integração. Posteriormente a construção dos três incrementos, serão realizadas as seguintes atividades: validação do sistema, treinamento de usuários e liberação de usuários.

³ : Desenvolvimento incremental de software: consultar o texto da Unidade 2 – Processo de Software.

Período	1º Trimestre			2º Trimestre			3º Trimestre			4º Trimestre		
Atividades	1	2	3	4	5	6	7	8	9	10	11	12
Engenharia de Requisitos <i>Nome do responsável:</i>												
Incremento 1: Análise <i>Nome do responsável:</i>												
Incremento 1: Projeto <i>Nome do responsável:</i>												
Incremento 1: Implementação <i>Nome do responsável:</i>												
Incremento 1: Testes da unidade e incremento <i>Nome do responsável:</i>												
Incremento 2: Análise <i>Nome do responsável:</i>												
Incremento 2: Projeto <i>Nome do responsável:</i>												
Incremento 2: Implementação <i>Nome do responsável:</i>												
Incremento 2: Testes da unidade, incremento e integração <i>Nome do responsável:</i>												
Incremento 3: Análise <i>Nome do responsável:</i>												
Incremento 3: Projeto <i>Nome do responsável:</i>												
Incremento 3: Implementação <i>Nome do responsável:</i>												
Incremento 3: Testes da unidade, incremento e integração <i>Nome do responsável:</i>												
Validação do sistema (considerando todos os incrementos) <i>Nome do responsável:</i>												
Treinamento de usuários e disponibilização da documentação <i>Nome do responsável:</i>												
Liberação do sistema <i>Nome do responsável:</i>												

Figura 4 : Exemplo de formato de cronograma para o desenvolvimento incremental no contexto de um sistema de software

Cada uma das atividades dos exemplos de cronograma das Figuras 3 e 4 podem ser colocadas, também, de forma mais detalhada em outro cronograma. Por exemplo, a atividade Projeto, da Figura 3, poderia ser detalhada da seguinte forma:

Atividade: Projeto com 3 meses de duração.

Sub-atividade 1: Projeto de arquitetura (análise de alternativas e definição da arquitetura mais apropriada)

Duração: duas semanas de duração

Sub-atividade 2: Especificação de cada subsistema da arquitetura, interfaces e interações entre subsistemas da arquitetura

Duração: um mês de duração

Sub-atividade 3: Especificação de todos os componentes envolvidos na arquitetura

Duração: duas semanas de duração

Sub-atividade 4: Especificação de estruturas de dados e algoritmos

Duração: um mês de duração

3. Resumo

No contexto do gerenciamento de projetos de sistemas de software, foram apresentados, neste documento, os seguintes tópicos: atividades de gerenciamento, planejamento de projeto e cronograma do projeto.

Referências Bibliográficas

Bezerra E. Princípios de Análise e Projetos de Sistemas com UML, 2a edição. Elsevier-Editora Campus. 2007.

Pfleeger S. L. Software Engineering: theory and practice. Prentice-Hall. 1998.

Pressman R. Engenharia de Software, 6a edição, AMGH Editora. 2010.

Rezende D. A. Engenharia de software e sistemas de informação, 2a edição. Brasport, 2002.

Sommerville, I. Engenharia de Software. 8a Edição. Addison Wesley. 2007.

Exercícios propostos

1. Elaborar o enunciado de um sistema vídeo-locadora proposto por você, também elaborar um cronograma, mostrando atividades, estimativa de tempo, e quem é responsável por cada atividade para esse sistema de vídeo-locadora.
2. No contexto de um sistema de tipo livraria virtual (venda de livros pela internet), por exemplo: o site da Edufal (<http://www.edufal.ufal.br>) ou um sistema utilizado pelos caixas nos supermercados (exemplo: Bom preço) para o gerenciamento da venda de produtos. Elaborar um cronograma, mostrando atividades, estimativa de tempo, e quem é responsável por cada atividade para o projeto de software do sistema escolhido.
3. Elaborar o enunciado de um sistema de software proposto por você e também elaborar um cronograma, mostrando atividades, estimativa de tempo, e quem é responsável por cada atividade para esse sistema de software proposto.

UNIVERSIDADE ABERTA DO BRASIL
UNIVERSIDADE FEDERAL DE ALAGOAS
Instituto de Computação
Curso de Bacharelado em Sistemas de
Informação

Engenharia de Software

Unidade 7: Gerenciamento de Qualidade
de Software

Prof. Arturo Hernández Domínguez

Conteúdo:

- 1. Introdução***
- 2. Qualidade de Software***
- 3. Qualidade de Processo e de Produto***
- 4. Garantia de Qualidade e Padrões***
- 5. Planejamento de Qualidade***
- 6. Controle de Qualidade***
- 7. Medições e Métricas de Software***

Atividades usando o ambiente virtual de aprendizagem

Exercícios



Este trabalho está licenciado com uma Licença [Creative Commons - Atribuição-NãoComercial-SemDerivações 4.0 Internacional](https://creativecommons.org/licenses/by-nc-nd/4.0/)

Outubro – 2010

1. Introdução

Atualmente, a preocupação com a qualidade dos produtos em geral que são adquiridos pelos consumidores no comércio é algo comum. No contexto de software, para o sucesso de uma organização e sobretudo se manter ao longo do tempo, é necessário que os produtos de software criados pela organização tenham qualidade. Um produto, fornecido por uma organização de software, sem qualidade terá um impacto negativo na imagem daquela organização e o cliente na próxima aquisição de um software procurará uma outra organização.

2. Qualidade de Software

A qualidade refere a características mensuráveis (PRESSMAN, 2010), no contexto de software é mais difícil de caracterizar do que em objetos físicos.

Uma definição de qualidade, como ponto de partida (PRESSMAN, 2010), “qualidade de software é a satisfação de requisitos funcionais e de desempenho explicitamente declarados, normas de desenvolvimento explicitamente documentadas e características implícitas que são esperadas em todo o software desenvolvido profissionalmente”.

A definição anterior que poderia ser debatida, modificada e estendida, serve para destacar 3 pontos importantes (PRESSMAN, 2010):

1. A falta de conformidade com os requisitos é falta de qualidade.
2. Se as normas de desenvolvimento não são seguidas, quase sempre resulta em falta de qualidade.
3. Se o software satisfaz seus requisitos explícitos, mais deixa de satisfazer os requisitos implícitos (por exemplo, facilidade de uso), a qualidade do software é suspeita.

A qualidade de software é importante, mas se o usuário/cliente não está satisfeito, devido a não entrega dentro do prazo ou gastos maiores ao previsto, a situação é problemática. Neste contexto é considerada a satisfação do cliente/usuário da seguinte forma:

$$\text{Satisfação do usuário} = \text{produto adequado} + \text{máxima qualidade} + \text{entrega dentro do orçamento e do cronograma}$$

A norma ISO9126 identifica seis atributos-chave de qualidade (PRESSMAN, 2010):

Atributo de qualidade	Descrição
<i>1. Funcionalidade</i>	<i>Grau em que o software satisfaz as necessidades declaradas.</i>
<i>2. Confiabilidade</i>	<i>Período de tempo em que o software está disponível para uso.</i>
<i>3. Usabilidade</i>	<i>Grau em que o software é fácil de usar.</i>
<i>4. Eficiência</i>	<i>Grau em que o software faz uso otimizado dos recursos do sistema.</i>
<i>5. Manutenibilidade</i>	<i>Facilidade com a qual podem ser feitos reparos no software.</i>
<i>6. Portabilidade</i>	<i>Facilidade com a qual o software pode ser transposto de um ambiente para outro.</i>

Para cada atributo de qualidade são definidos subatributos que devem ser considerados segundo a norma ISO9126.

A produção de software de qualidade é uma meta básica da engenharia de software (ROCHA ET AL., 2001), que disponibiliza métodos, técnicas e ferramentas para esse fim.

Segundo SOMMERVILLE (2007), bons gerentes de qualidade objetivam o desenvolvimento e implantação de uma `cultura de qualidade' na organização, na qual todos os responsáveis pelo desenvolvimento do produto estão envolvidos e comprometidos em atingir um alto nível de qualidade de produto.

O gerenciamento de qualidade de software para sistemas de grande porte pode ser estruturado em três atividade principais (SOMMERVILLE, 2007):

1. Garantia de qualidade. Consiste na definição de procedimentos organizacionais e padrões que levam a um software de alta qualidade.
2. Planejamento de qualidade. Seleção de procedimentos e padrões apropriados, adaptados para um projeto de software específico.
3. Controle de qualidade. Definição e aprovação de processos que assegurem que a equipe de desenvolvimento de software tenha seguido os procedimentos e os padrões de qualidade do projeto.

A equipe de garantia de qualidade deve ser uma equipe independente da equipe de desenvolvimento, isto é, a equipe de garantia de qualidade deve trabalhar (sem interferências) e ter uma visão objetiva do software, relatando problemas e dificuldades para a gerência da organização. A equipe de gerenciamento de qualidade é responsável pelo gerenciamento de qualidade de software na organização e essa equipe assegura que os objetivos de qualidade de software não sejam comprometidos por orçamentos de curto prazo e considerações de cronograma.

3. Qualidade de Processo e de Produto

A qualidade de processo influencia de forma significativa na qualidade de software (SOMMERVILLE, 2007). O gerenciamento da qualidade de processo pode ter como efeito um software entregue com poucos defeitos.

O gerenciamento de qualidade de processo envolve (SOMMERVILLE, 2007):

1. Definição de padrões de processo, 'como' e 'quando' as revisões devem ser conduzidas.
2. Monitoração do processo de desenvolvimento para assegurar que os padrões estão sendo seguidos.
3. Relato do processo de software para a gerência de projeto e para o comprador de software.

4. Garantia de Qualidade e Padrões

O processo de garantia de qualidade (norma ISO/IEC 12207) possibilita garantir que os processos e produtos de software, no ciclo de vida do projeto, estejam em conformidade com os requisitos especificados e referentes aos planos estabelecidos (ROCHA ET AL., 2001). Garantia de qualidade é o processo de definição de como a qualidade de software pode ser atingida.

Os dois tipos de padrões que podem ser estabelecidos como parte do processo de garantia de qualidade são (SOMMERVILLE, 2007):

1. Padrões de produto correspondem ao produto de software em desenvolvimento. Incluem padrões de documentos como a estrutura do documento de requisitos e documentos de modelagem, padrões de documentação de código como documentação

de classes e métodos, padrões de codificação no contexto de uso de uma linguagem de programação como java e c++.

2. Padrões de processo, definem os processos que devem ser seguidos durante o desenvolvimento de software. Incluem as atividades e documentos associados a especificação, projeto e implementação, e validação de software.

Os gerentes de qualidade precisam seguir os seguintes passos (SOMMERVILLE, 2007):

1. Envolver os engenheiros de software na seleção de padrões de produto e justificar as decisões tomadas sobre a padronização.

2. Revisar e atualizar os padrões segundo as mudanças de tecnologia.

3. Prover ferramentas de software para apoiar o uso de padrões.

O gerente de projeto e o gerente de qualidade devem evitar os problemas de padrões inapropriados através de um planejamento de qualidade no início do projeto.

5. Planejamento de Qualidade

Planejamento de qualidade é o processo de desenvolvimento de um plano de qualidade para um projeto. O plano de qualidade estabelece as qualidades de software desejadas e descrever como elas devem ser avaliadas. Nesse plano, são estabelecidos os padrões para determinado produto e o processo de desenvolvimento. A estrutura geral para um plano de qualidade é:

1. Apresentação do produto. Descrição do produto, o mercado pretendido e as expectativas de qualidade para o produto.

2. Planos de produto. As datas críticas de liberação e as responsabilidades para o produto junto com planos para serviços de distribuição e de produto.

3. Descrições de processo. Os processos de desenvolvimento e serviços que devem ser usados para o desenvolvimento e gerenciamento de produto.

4. Metas de qualidade. As metas de qualidade para o produto incluindo identificação e justificativa de atributos críticos de qualidade de produto.

5. Riscos e gerenciamento de riscos. Os riscos principais que poderiam afetar a qualidade de produto e as ações para tratar esses riscos.

Planos de qualidade muito longos as pessoas não vão ter disposição de ler, então escrever planos de qualidade tão breves quanto possível (SOMMERVILLE, 2007).

6. Controle de Qualidade

Controle de qualidade envolve a monitoração do processo de desenvolvimento de software para assegurar que os procedimentos e os padrões de garantia de qualidade estão sendo seguidos (SOMMERVILLE, 2007). Os produtos do processo são verificados contra os padrões de projeto definidos no processo de controle de qualidade.

A abordagem de revisões de qualidade, realizadas por um grupo de pessoas, de um processo ou produto são amplamente usadas. A revisão é responsável por verificar que os padrões de projeto foram seguidos e que o software e os documentos estão em conformidade com esses padrões.

7. Medições e Métricas de Software

A medição de software (SOMMERVILLE, 2007) se dedica a derivar um valor numérico para algum atributo de um produto de software ou de um processo de software. A qualidade de software ou dos processos de software pode ser analisada comparando os valores das medições uns com os outros e aos padrões que se aplicam na organização.

Uma métrica de software (SOMMERVILLE, 2007) é qualquer tipo de medição referente a um sistema de software, processo ou documentação relacionada.

Exemplos de métricas são (SOMMERVILLE, 2007) (ROCHA ET AL., 2001):

medida de tamanho de um produto em linhas de código;

o número de defeitos relatados em um produto de software entregue;

tempo total do projeto, isto é, tempo nas fases de análise, projeto, codificação, testes, tempo em reuniões de revisão, tempo em retrabalho;

precisão da estimativa do cronograma de todo o projeto e das fases de análise, projeto, codificação e testes;

número de modificações na especificação de requisitos, projeto ou código após a sua aprovação;

rotatividade do pessoal, isto é, percentual de pessoas que saíram, entraram ou mudaram de função durante o desenvolvimento do projeto;

a experiência da equipe, isto é, tempo de experiência na linguagem de programação, no domínio da aplicação, nas ferramentas, no método e no processo de desenvolvimento.

As medições de software podem ser usadas (SOMMERVILLE, 2007):

1. Para fazer previsões gerais sobre um sistema. Ao medir as características dos componentes de sistema, é possível derivar uma estimativa geral de algum atributo de sistema, como o número de defeitos.
2. Para identificar componentes anômalos, isto é, as medições podem identificar componentes individuais cujas características mostram um desvio de alguma regra. Esses componentes podem ter problemas de qualidade e precisam ser analisados mais detalhadamente.

É importante considerar que não existem métricas de software padronizadas e universalmente aplicáveis (SOMMERVILLE, 2007), as organizações devem realizar uma seleção das suas métricas e analisar as medições baseadas no conhecimento e nas circunstâncias locais.

8. Resumo

No contexto de qualidade de software e gerenciamento de qualidade de software, foram apresentados, neste documento, os seguintes tópicos: qualidade de software, qualidade de processo e de produto, garantia de qualidade e padrões, planejamento de qualidade, controle de qualidade, medições e métricas de software.

Referências Bibliográficas

Pfleeger S. L. Software Engineering: theory and practice. Prentice-Hall. 1998.

Pressman R. Engenharia de Software, 6a edição, AMGH Editora. 2010.

Rezende D. A. Engenharia de software e sistemas de informação, 2a edição. Brasport, 2002.

Rocha A. R. C., Maldonado J. C., Weber K. C. Qualidade de Software: Teoria e Pratica, Prentice Hall, 2001.

Sommerville, I. Engenharia de Software. 8a Edição. Addison Wesley. 2007.

Atividades usando o Ambiente Virtual de Aprendizagem

Atividade 13 - Discussão no Fórum:

O que é o gerenciamento de qualidade de software?

No contexto da garantia de qualidade de software: O que é um padrão de produto? e

O que é um padrão de processo?

Atividade 14 (a realização desta atividade é em grupo ou individual) -

Pesquisar na Internet e outros documentos. Construir um texto, de forma detalhada, sobre duas (2) organizações, no Brasil, bem sucedidas do ponto de vista qualidade de software.

Exercícios propostos

1. No contexto de um sistema a ser desenvolvido em uma organização, dizer quais são os elementos que caracterizam a satisfação do cliente/usuário.
2. No contexto de um sistema de tipo livraria virtual (venda de livros pela internet), refletir, e propor três elementos que representam a falta de qualidade nesse sistema.
3. No contexto de uma organização produtora de software proposta por você, refletir sobre o que deve ser realizado por essa organização para garantir a qualidade dos produtos de software construídos e fornecidos.