INSTITUTO FEDERAL DE MATO GROSSO DO SUL
CAMPUS NAVIRAÍ

NOTAS DE AULA DE

ANÁLISE E PROJETO ORIENTADO A OBJETO I

Prof. MSc. Luiz F. Picolo

NAVIRAÍ - MS

Atualizado em 15 de março de 2020

1 Introdução

Não é o mais forte que sobrevive, nem o mais inteligente.
Quem sobrevive é o mais disposto à mudança.
Charles Darwin

Praticamente todos os países, hoje em dia, dependem de complexos sistemas com base em computadores. Cada vez mais os produtos incorporam, de algum modo, computadores e software de controle. Nesses sistemas, o software representa uma grande e crescente proporção do custo total do sistema. Por isso, produzir software de um modo que apresente uma boa relação custo-benefício é essencial para o funcionamento das economias nacionais e inter- nacionais [Sommerville 2003].

1.1 Engenharia de Software

Ao se citar o nome Engenharia, muitas áreas associadas a ela despontam em nossa mente, tais como engenharia civil, engenharia elétrica, engenharia mecânica, entre tantas outras. Dentre estes vários ramos da engenharia se encontra a **Engenharia de software**. A engenharia de software é uma disciplina da engenharia, cuja meta é o desenvolvimento de sistemas de software com boa relação custo-benefício [Sommerville 2003].

1.1.0.1 Objetivos

Podemos destacar, dentro vários objetivos, quatro. Nestes, a engenharia de software tem como objeto:

- Ser uma abordagem sistemática e organizada;
- Organização de um processo;
- Buscar a redução do custo com manutenção (80%);
- Produzir software de alta qualidade.

1.1.1 O que é software?

Muita gente associa o termo software aos programas de computador. Na verdade, essa é uma visão muito restritiva. Software não é apenas o programa mas também toda a documentação associada e os dados de configuração necessários para fazer com que esses pro- gramas operem corretamente [Sommerville 2003]. Em geral, os engenheiros de

software adotam uma abordagem sistemática e organizada em seu trabalho, uma vez que essa é, com frequência, a maneira mais eficaz de produzir software de alta qualidade.

Os engenheiros de software se ocupam de desenvolver produtos de software, isto é, software que possa ser vendido a um cliente. Há dois tipos de produtos de software:

1. **Produtos genéricos:** São sistemas do tipo *stand-alone*, produzidos por uma organização de desenvolvimento e vendidos no mercado a qualquer cliente capaz de adquiri-los.

2. **Produtos sob encomenda:** São os sistemas encomendados por um cliente em particular. O software é desenvolvido especialmente para aquele cliente por uma empresa de software.

1.1.2 Ciclo de Vida de Software

Todo projeto de software começa por alguma necessidade do negócio, a necessidade de corrigir um defeito em uma aplicação existente. O ciclo de vida de um software descreve as fases pelas quais o software passa desde a sua concepção até ficar sem uso algum. O conceito de ciclo de vida de um software é muitas vezes confundido com o de modelo de processo.

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

- **Definição**: Nesta atividade, se concentra a busca pelo conhecimento da situação atual e a identificação de problemas para a elaboração de propostas de solução de sistemas computacionais que resolvam tais problemas.

- **Desenvolvimento**: A fase de desenvolvimento ou de produção do software inclui todas as atividades que tem por objetivo a construção do produto.

- **Operação**: Nesta fase o software é entregue e todos os processos inerentes a fase se inicial como a instalação, configuração, manutenção, atualização, entre outros.

- **Retirada**: O software com o tempo se torna legado. Ele deve evoluir para novos sistemas operacionais ou para incorporar novos requisitos.

1.1.3 Processo de software

Segundo Sommerville 2003 um processo de software é um conjunto de atividades e resultados associados que geram um produto de software. Quando se elabora um produto

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

1. **Levantamento de requisitos:** A atividade de levantamento de requisitos (também conhecida como elicitação de requisitos) corresponde à etapa de compreensão do problema aplicada ao desenvolvimento de software.

2. **Análise:** As fases de levantamento de requisitos e de análise de requisitos recebem o nome de engenharia de requisitos. Formalmente, o termo análise corresponde a "quebrar" um sistema em seus componentes e estudar como eles interagem entre si com o objetivo de entender como esse sistema funciona.

3. **Projeto:** O foco principal da análise são os aspectos lógicos e independentes de implementação de um sistema (i.e., os requisitos desse sistema). Na fase de projeto, determina-se "como" o sistema funcionará para atender aos requisitos, de acordo com os recursos tecnológicos existentes.

4. **Implementação:** o sistema é codificado.

5. **Testes:** Diversas atividades de teste são realizadas para verificação do sistema construído, levando-se em conta a especificação feita nas fases de análise e de projeto.

6. **Implantação:** Na fase de implantação, o sistema é empacotado, distribuído e instalado no ambiente do usuário.

1.1.4 Modelo de processo de software

Para Sommerville 2003 um modelo de processo de software é uma descrição simplificada de um processo de software, que é apresentada a partir de uma perspectiva específica. Ou seja, é uma abstração do processo real que está sendo descrito. Existe uma série de diferentes modelos gerais, ou paradigmas, de desenvolvimento de software:

1.1.4.1 Cascata

Este ciclo de vida também é chamado de clássico ou linear e se caracteriza por possuir uma tendência na progressão sequencial entre uma fase e a seguinte. Eventualmente, pode haver uma retroalimentação de uma fase para a anterior, mas, de um ponto de vista macro, as fases seguem sequencialmente [Bezerra 2015]

Esse modelo considera as atividades de especificação, desenvolvimento, validação e evolução, que são fundamentais ao processo, e as representa como fases separadas do

processo, como a especificação de requisitos, o projeto de software, a implementação, os testes e assim por diante.

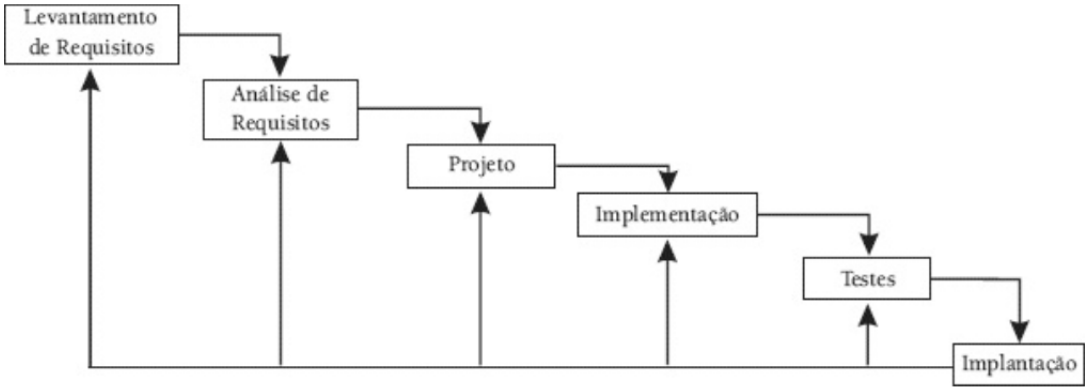

Figura 1 – Modelo cascata [Bezerra 2015]

1.1.4.2 O modelo de processo de software iterativo e incremental

Segundo Bezerra 2015 , o modelo de processo de software incremental e iterativo foi proposto como uma resposta aos problemas encontrados no modelo em cascata. Um processo de desenvolvimento utilizando essa abordagem divide o desenvolvimento de um produto de software em ciclos. Em cada ciclo dessa etapa podem ser identificadas as fases de análise, projeto, implementação e testes. Essa característica contrasta com a abordagem clássica, na qual as fases de análise, projeto, implementação e testes são realizadas uma única vez.

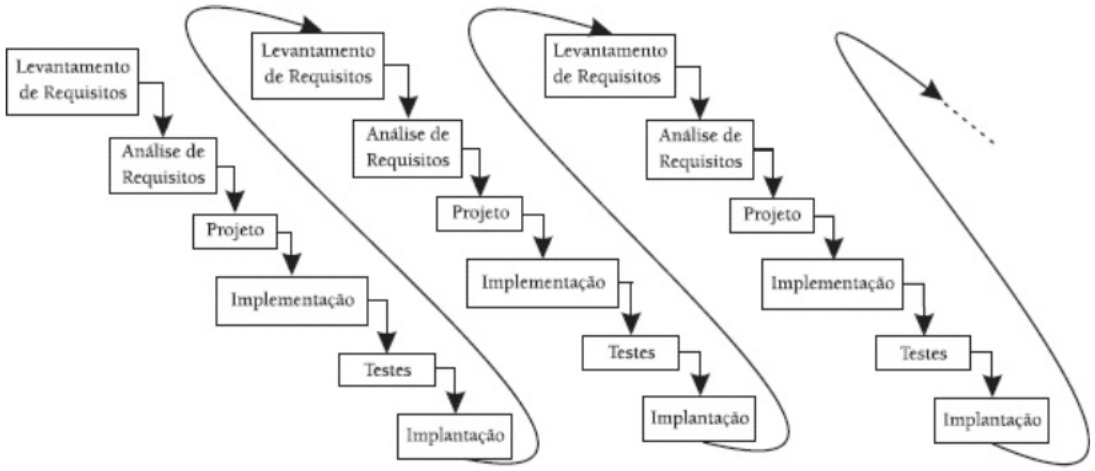

Figura 2 – No processo incremental e iterativo, cada iteração é uma minicascata. [Bezerra 2015]

1.1.4.3 Modelos de processo evolucionário

Software, assim como todos sistemas complexos, evolui ao longo do tempo. Conforme o de- senvolvimento do projeto avança, as necessidades de negócio e de produto mudam

frequente- mente, tornando inadequado seguir um planejamento em linha reta de um produto final [Pressman e Maxim 2016]. Assim, faz-se necessário um modelo de processo que tenha sido projetado especificamente para desenvolver um produto que evolua ao longo do tempo.

Modelos evolucionários são iterativos. Apresentam características que possibilitam desen- volver versões cada vez mais completas do software. Nos parágrafos seguintes, são apresenta- dos dois modelos comuns em processos evolucionários [Pressman e Maxim 2016].

1.1.4.3.1 Prototipação

Na sua forma ideal, o protótipo atua como um mecanismo para identificar os requisitos do software. Caso seja necessário desenvolver um protótipo operacional, pode-se utilizar partes de programas existentes ou aplicar ferramentas (por exemplo, geradores de relatórios e gerencia- dores de janelas) que possibilitem gerar rapidamente tais programas operacionais.

O protótipo pode servir como "o primeiro sistema". Aquele que Brooks recomenda que se jogue fora. Porém, essa pode ser uma visão idealizada. Embora alguns protótipos sejam cons- truídos como "descartáveis", outros são evolucionários, no sentido de que evoluem lentamente até se transformar no sistema real [Pressman e Maxim 2016].

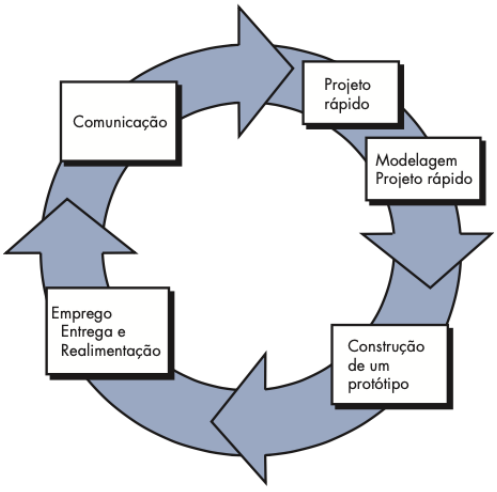

Figura 3 – Modelo Prototipação [Pressman e Maxim 2016]

1.1.4.3.2 Modelo Espiral

É um modelo de processo de software evolucionário que acopla a natureza iterativa da prototi- pação com os aspectos sistemáticos e controlados do modelo cascata. Fornece potencial para o rápido desenvolvimento de versões cada vez mais completas do software [Pressman e Maxim 2016].

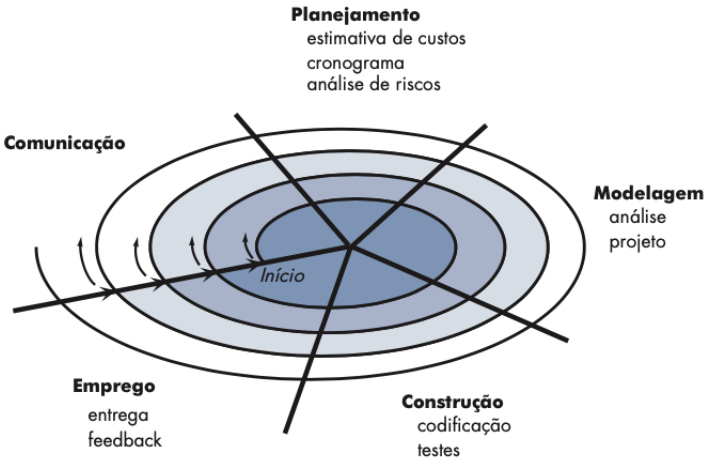

Figura 4 – Modelo em espiral [Pressman e Maxim 2016]

O modelo espiral é uma abordagem realista para o desenvolvimento de sistemas e de software em larga escala. Pelo fato de o software evoluir à medida que o processo avança, o desenvolvedor e o cliente compreendem e reagem melhor aos riscos[1] em cada nível evolucionário.

[1] Risco pode ser entendido como a avaliação da probabilidade de um perigo ocorrer e no cálculo de seu possível impacto e prejuízo para a corporação.

2 Engenharia de requisitos

Um ladrão rouba um tesouro, mas não furta a inteligência.
Uma crise destrói um herança, mas não uma profissão.
Não importa se você não tem dinheiro, você é uma pessoa rica,
pois possui o maior de todos os capitais: a sua inteligência.
Invista nela. ***Estude!****.*
Augusto Cury

Os problemas que os engenheiros de software têm para solucionar são, muitas vezes, imensamente complexos. Compreender a natureza dos problemas pode ser muito difícil, especialmente se o sistema for novo sendo que, um dos pontos que mais dificultam é o conflito entre a visão ampla e o foco, ou seja, muitas vezes o cliente não consegue enxergar as minucias do problema mas somente ele de forma geral. Assim, os engenheiros de software necessitam de meios que os auxiliem na descoberta destas minucias que, juntas, constroem a identidade do software. Portanto, para auxiliar os engenheiros nesta difícil tarefa, foi criada a Engenharia de requisitos.

2.1 Engenharia de Requisitos

Para que possamos compreender o que é Engenharia de requisitos, devemos entender primeiramente o que é um requisito.

Um requisito de software, de forma geral, são as descrições das funções e das restrições de um sistema. Entende-se por função algo que o sistema deve fazer, e por restrição o contrário, o que o sistema não deve executar. Já o processo de descobrir, analisar, documentar e verificar essas funções e restrições é chamado de **engenharia de requisitos** [Sommerville 2003].

Contudo, para [Sommerville 2003], o termo requisito é usado de forma errada pela industria. Assim, o autor separa a ideia de requisitos em dois grandes grupos, os quais são:

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

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

Diferentes níveis de especificação de sistema são úteis porque comunicam informações sobre o sistema para diferentes tipos de leitores. A Figura 5 ilustra a distinção entre os requisitos de usuários e os de sistemas. Ela mostra como um requisito de usuário pode ser expandido em diversos requisitos de sistema.

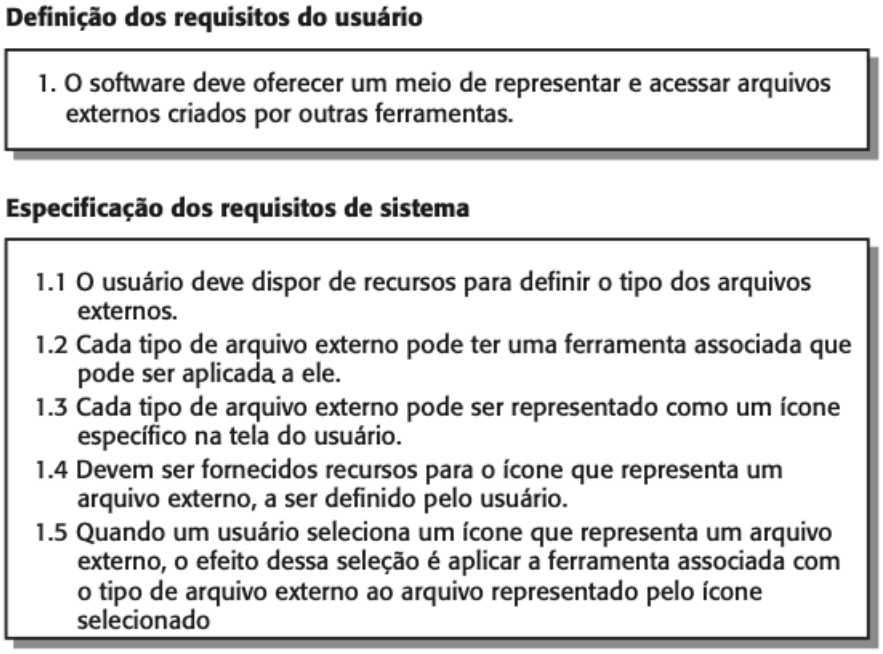

Figura 5 – Requisitos do usuário e do sistema. [Sommerville 2003]

2.2 Requisitos funcionais e não funcionais

Os requisitos de sistema de software são, frequentemente, classificados como funcionais ou não funcionais:

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

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

Porém, Sommerville 2003 esclarece que, na realidade, a distinção entre esses diferentes tipos de requisitos não é tão clara como sugerem essas definições simples. Um requisito de usuário relacionado à proteção, digamos, parece ser um requisito não funcional. Contudo, quando desenvolvido com mais detalhes, pode levar a outros requisitos que são

claramente funcionais, como a necessidade de incluir recursos de autorização de usuários no sistema. Portanto, embora seja útil classificar os requisitos dessa maneira quando os discutimos, devemos lembrar que essa é, na verdade, uma distinção artificial.

2.2.1 Requisitos funcionais

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

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

[RF01] - O software deve permitir que o atendente efetue o cadastro de clientes.

[RF02] - O software deve permitir que o caixa efetue o registro de itens vendidos.

[RF03] - O software deve permitir que o administrador gere um relatório de vendas por mês.

Dica: Uma boa prática para definição de requisitos funcionais: O software deve permitir que alguém ou alguma coisa realize algo no sistema.

Portanto, baseando-se nas ideias de Sommerville 2016, a princípio, a especificação dos requisitos funcionais de um sistema deve ser **completa** e **consistente**. Completude significa que todos os serviços requeridos pelo usuário devem ser definidos. Consistência significa que os requisitos não devem ter definições contraditórias. Lembrando também que o **documento de requisito não é estático**. Os problemas somente emergem depois de uma análise mais profunda. À medida que os problemas são descobertos durante as revisões ou em fases posteriores do ciclo de vida, os problemas no documento de requisitos devem ser corrigidos [Sommerville 2003].

2.2.2 Requisitos não funcionais

Os requisitos não funcionais, como o nome sugere, são requisitos que não estão diretamente relacionados com os serviços específicos oferecidos pelo sistema a seus usuários. Eles podem estar relacionados às propriedades emergentes do sistema, como confiabilidade, tempo de resposta e ocupação de área [Sommerville 2016].

Da mesma maneira que os requisitos funcionais podem ser classificados, os requisitos não funcionais também podem de acordo com a sua especificidade:

1. **Requisitos de produto.** Esses requisitos especificam ou restringem o comporta-

mento do software.

2. **Requisitos organizacionais.** Esses são os requisitos gerais de sistemas derivados das políticas e procedimentos da organização do cliente e do desenvolvedor.

3. **Requisitos externos.** Esse tipo abrange todos os requisitos que derivam de fatores externos ao sistema e seu pro- cesso de desenvolvimento.

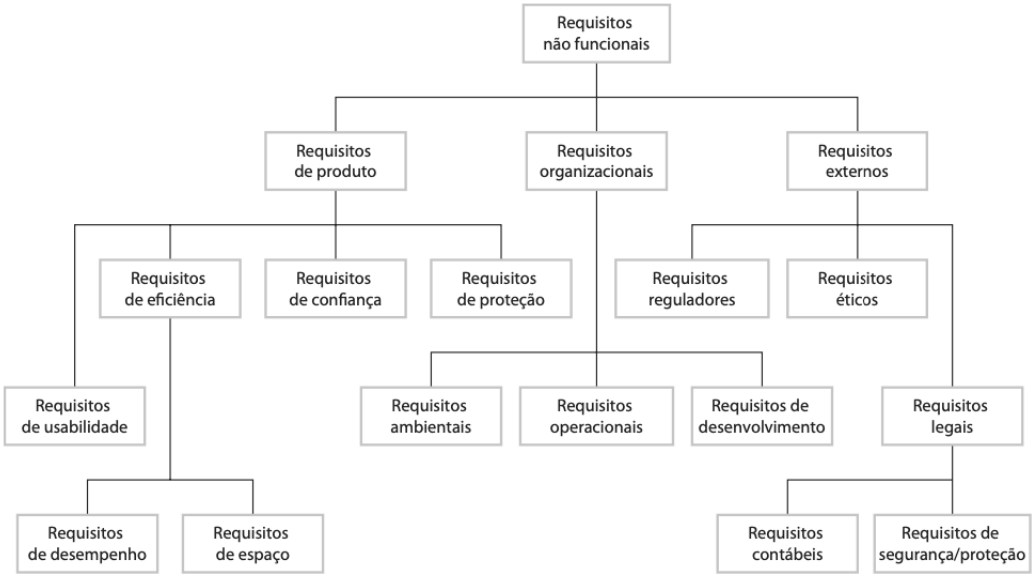

Figura 6 – Tipos de requisitos não funcionais [Sommerville 2016]

Referências

BEZERRA, E. *Princípios de Análise e Projeto de Sistemas com UML*. [S.l.]: Elsevier Rio de Janeiro, 2015. v. 3. Citado 2 vezes nas páginas 4 e 5.

NARDELLI, C. Apostila da disciplina de engenharia de software–i. Citado na página 10.

PRESSMAN, R.; MAXIM, B. *Engenharia de Software-8ª Edição*. [S.l.]: McGraw Hill Brasil, 2016. Citado 2 vezes nas páginas 6 e 7.

SOMMERVILLE, I. *Engenharia de Software. 6ª*. [S.l.: s.n.], 2003. Citado 6 vezes nas páginas 2, 3, 4, 8, 9 e 10.

SOMMERVILLE, I. *Software Engineering GE*. [S.l.]: Pearson Australia Pty Limited, 2016. Citado 2 vezes nas páginas 10 e 11.