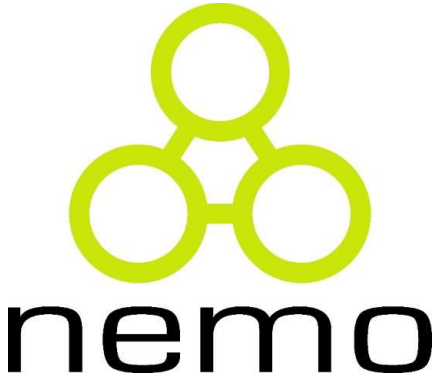


Projeto de Sistemas de Software

Jordana S. Salamon

jssalamon@inf.ufes.br

jordanasalamon@gmail.com

DEPARTAMENTO DE INFORMÁTICA

CENTRO TECNOLÓGICO

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

A Fase de Projeto

- O projeto é o processo criativo de transformar uma especificação de um problema em uma especificação de uma solução.

- O projeto de software utiliza-se da especificação de requisitos e dos modelos conceituais gerados na fase de levantamento e análise de requisitos. A partir dos requisitos, muitas soluções são possíveis e, portanto, muitos projetos diferentes podem ser produzidos.

- Em função das limitações da tecnologia e outras restrições, várias decisões devem ser tomadas de modo a tratar, sobretudo, requisitos não funcionais.

Princípios de Projeto

- Seja o exemplo do projeto de uma casa. Projetar uma casa é prover uma solução para o problema colocado, procurando satisfazer os requisitos do dono da casa (o cliente) e as restrições levantadas.

- **O que o projeto de uma casa deve contemplar?**

Princípios de Projeto

- Seja o exemplo do projeto de uma casa. Projetar uma casa é prover uma solução para o problema colocado, procurando satisfazer os requisitos do dono da casa (o cliente) e as restrições levantadas.

- Assim, de maneira geral, um projeto deve:

- • considerar abordagens alternativas com base nos requisitos do problema, restrições e conceitos de projeto;

- • ser rastreável à sua especificação;

- • reutilizar soluções bem sucedidas previamente adotadas;

Princípios de Projeto

- • exibir uniformidade (estilo) e integração (interfaces bem definidas entre componentes da coisa a ser construída);
- • ser estruturado para acomodar mudanças;
- • ser passível de avaliação da qualidade;
- • ser revisado para minimizar erros.

Princípios de Projeto

- Além disso, em geral, um modelo de projeto deve:
- • prover uma visão da totalidade da coisa a ser construída;
- • decompor o todo em partes e prover diferentes visões da coisa;
- • refinar e descrever com mais detalhes cada parte ou visão da coisa, de modo a prover orientação para a construção de cada detalhe.

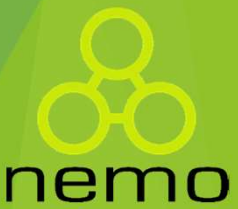
Princípios de Projeto

- As características citadas anteriormente valem tanto para o projeto de uma casa, quanto para o projeto de um sistema de software. O projeto de software deve:

- • considerar abordagens alternativas com base nos requisitos (funcionais e não funcionais) e conceitos de projeto de software;

- • estar relacionado aos modelos de análise e à especificação de requisitos e deve ser a eles rastreado;

- • reutilizar padrões de projeto, componentes, *frameworks* e outras soluções que se mostraram eficazes em outros projetos, sobretudo aqueles similares ao sistema em desenvolvimento;



Princípios de Projeto

- • exibir uniformidade (estilo) e integração (interfaces entre componentes);
- • ser estruturado para acomodar mudanças (alterabilidade);
- • ser passível de avaliação da qualidade;
- • ser revisado para minimizar erros.

Princípios de Projeto

- Além disso, o projeto de software deve:

- • minimizar a distância conceitual e semântica entre o software e o mundo real. Os modelos de projeto devem ser facilmente compreensíveis, tendo em vista que seu propósito é comunicar informações para profissionais responsáveis pela codificação, teste e manutenção;

- • acomodar circunstâncias não usuais. Se necessário abortar o processamento, fazê-lo de modo elegante;

- • apresentar nível de abstração superior ao código fonte, afinal, projeto não é codificação.

Princípios de Projeto

- Por fim, **modelos de projeto também devem ser construídos com o objetivo de prover uma visão geral do sistema a ser construído**, bem como uma variedade de visões mais específicas de seus elementos, de modo a guiar a implementação.

- Um modelo da arquitetura do sistema pode ser usado para prover uma visão geral da organização do sistema. Modelos específicos podem detalhar os diversos elementos da arquitetura.

Princípios de Projeto

- Diferentes diagramas podem ser usados para prover diferentes visões desses elementos, tais como diagramas de classes para uma visão estrutural e diagramas de sequência para uma visão comportamental.

- Por fim, devem-se projetar as classes que compõem cada um dos elementos da arquitetura, definindo detalhes de como implementar atributos, associações e métodos.

Princípios de Projeto

- Além dos princípios gerais de projeto, sete princípios gerais da Engenharia de Software se aplicam também ao projeto de software. São eles:

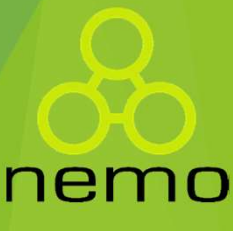
- • Um sistema de software existe para fornecer valor aos clientes e usuários. Todas as decisões, inclusive as de projeto, devem ser tomadas tendo isso em mente.

- • Todo projeto de software deve ser tão simples quanto possível, sem, no entanto, descartar características de qualidade importantes em nome da simplicidade.

Princípios de Projeto

- • O comprometimento com a visão arquitetural do sistema é essencial para o sucesso do projeto de software.

- • Os modelos elaborados na fase de projeto serão usados posteriormente por desenvolvedores responsáveis pela implementação, teste e manutenção do sistema. Assim, esses modelos devem ser claros, não ambíguos e fáceis de entender.



Princípios de Projeto

- • Um sistema com um longo tempo de vida tem mais valor. Contudo, para ter vida longa, um sistema deve ser projetado para acomodar mudanças.

- • A reutilização pode ajudar a poupar tempo e esforço, bem como aumentar a qualidade do sistema em desenvolvimento. Para conseguir um bom nível de reutilização, é necessário planejar o reuso com antecedência. Para a fase de projeto, existem muitos padrões arquitetônicos e padrões de projeto detalhado (*design patterns*) bastante maduros e documentados. Conhecer padrões e comunicar essas e outras oportunidades de reuso para os membros da organização é vital.

- • Raciocinar clara e completamente antes de realizar uma ação quase sempre produz melhores resultados. Aprender com os erros também é importante. Assim, ao raciocinar sobre uma decisão de projeto, soluções anteriores devem ser pesquisadas.

Princípios de Projeto

- Por fim, uma vez que a fase de projeto é essencialmente uma atividade de modelagem, princípios da modelagem ágil também se aplicam.

- Dentre eles, merecem destaque:
- • Seja econômico. Não crie mais modelos do que você precisa. Seja capaz de declarar um objetivo para cada modelo criado.

- • Procure produzir modelos mais simples.

- • Construa modelos de modo que sejam passíveis de mudanças.

- • Obtenha feedback tão logo quanto possível.

Qualidade do Projeto de Software

- Para se obter bons projetos, é necessário considerar alguns aspectos intimamente relacionados com a qualidade de projeto (design), dentre eles:

- • **Níveis de Abstração**: a abstração é um dos modos fundamentais pelos quais os seres humanos enfrentam a complexidade. Assim, um bom projeto deve considerar vários níveis de abstração, começando com em um nível mais alto, próximo da fase de análise.

- À medida que se avança no processo de projeto, o nível de abstração deve ser reduzido. Dito de outra maneira, o projeto deve ser um processo de refinamento, no qual o projeto vai sendo conduzido de níveis mais altos para níveis mais baixos de abstração.

Qualidade do Projeto de Software

- • **Modularidade**: um bom projeto deve estruturar um sistema como módulos ou componentes coesos e fracamente acoplados. A modularidade permite a um projeto de sistema ser intelectualmente gerenciável.

- A estratégia de “dividir para conquistar” é reconhecidamente útil no projeto de software, pois é mais fácil resolver um problema complexo quando o mesmo é dividido em partes menores e, por conseguinte, mais facilmente gerenciáveis.

Qualidade do Projeto de Software

- • **Ocultação de Informações**: o conceito de modularidade leva o projetista a uma questão fundamental: até que nível a decomposição deve ser aplicada? Em outras palavras, quão modular deve ser o software?

- O princípio da ocultação de informações sugere que os módulos / componentes sejam caracterizados pelas decisões de projeto que cada um deles esconde dos demais.

Qualidade do Projeto de Software

- Módulos devem ser projetados e especificados de modo que as informações neles contidas (dados e algoritmos) sejam inacessíveis a outros módulos, sendo necessário conhecer apenas a sua interface.

- Ou seja, a ocultação de informação trabalha encapsulando detalhes que provavelmente serão alterados de forma independente, em módulos distintos. A interface de um módulo revela apenas aqueles aspectos considerados improváveis de mudar.

Qualidade do Projeto de Software

- • **Independência Funcional**: a independência funcional é uma decorrência direta dos conceitos de abstração, modularidade e ocultação de informações. Ela é obtida pelo desenvolvimento de módulos com finalidade única e pequena interação com outros módulos. Em outras palavras, módulos devem cumprir uma função bem estabelecida, minimizando interações com outros módulos.

- Módulos funcionalmente independentes são mais fáceis de entender, codificar, testar e alterar. Efeitos colaterais causados pela modificação de um módulo são limitados e, por conseguinte, a propagação de erros é reduzida. A independência funcional pode ser avaliada usando dois critérios de qualidade: **coesão e acoplamento.**

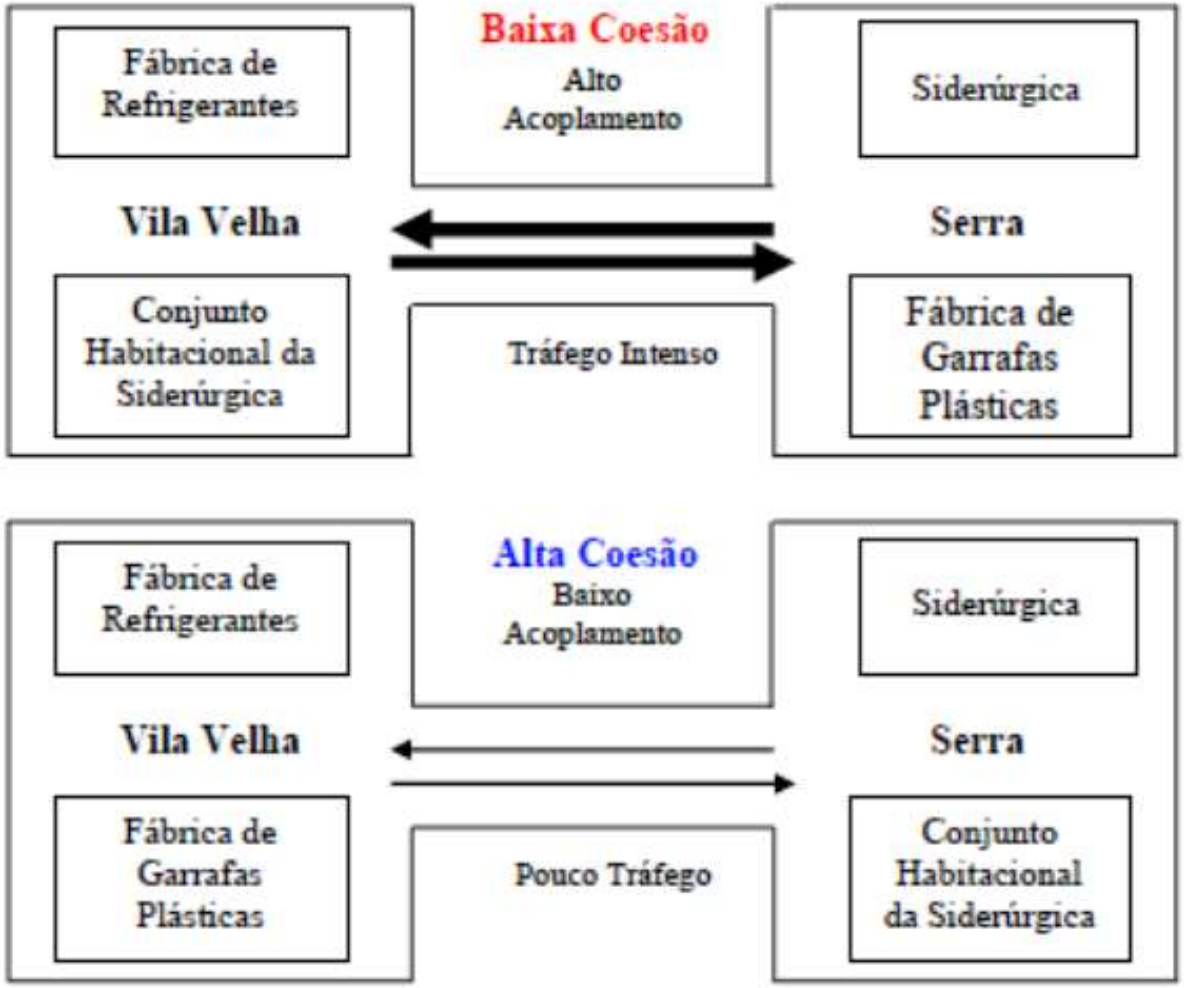
Qualidade do Projeto de Software

- A coesão se refere às responsabilidades atribuídas a um módulo.
- Uma classe, p.ex., é dita coesa quando tem um conjunto pequeno e focado de responsabilidades e aplica seus atributos e métodos especificamente para implementar essas responsabilidades.

- Já o acoplamento diz respeito ao grau de interdependência entre dois módulos. O objetivo é minimizar o acoplamento, isto é, tornar os módulos tão independentes quanto possível. Idealmente, classes de projeto em um subsistema deveriam ter conhecimento limitado de classes de outros subsistemas.

- *Coesão e acoplamento são interdependentes e, portanto, uma boa coesão deve levar a um baixo acoplamento.*

Qualidade do Projeto de Software



nemo

Projeto e Atributos de Qualidade

- Conforme citado anteriormente, a fase de projeto é responsável por incorporar requisitos tecnológicos aos requisitos essenciais. Assim, o projetista deve estar atento aos critérios de qualidade que o sistema deve atender.

- As considerações de negócio determinam as características de qualidade que devem ser acomodadas em um sistema.

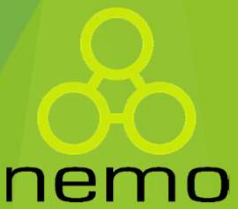
- Essas características de qualidade vão além das funcionalidades, ainda que estejam fortemente relacionadas a elas. De fato, funcionalidades e atributos de qualidade são ortogonais.

Projeto e Atributos de Qualidade

- Muitos sistemas são reconstruídos não porque são funcionalmente deficientes, mas sim porque são difíceis de manter, portar, escalar ou porque são muito lentos ou inseguros. Isso mostra a importância de considerar atentamente os requisitos não funcionais durante a fase de projeto.

- São muitos os atributos de qualidade que potencialmente podem ser importantes para um sistema.

- Por exemplo, o modelo de qualidade de produtos de software definido na norma ISO/IEC 25010, utilizado como referência para a avaliação de produtos de software, define oito características de qualidade para produtos de software, desdobradas em subcaracterísticas:



Projeto e Atributos de Qualidade

- • **Adequação Funcional**: grau em que o produto provê funções que satisfazem às necessidades explícitas e implícitas, quando usado em condições especificadas.

- Inclui subcaracterísticas que evidenciam a existência de um conjunto de funções e suas propriedades específicas, a saber:

- - Completude Funcional: capacidade do produto de software de prover um conjunto apropriado de funções para tarefas e objetivos do usuário especificados. Refere-se às necessidades declaradas. Um produto no qual falte alguma função requerida não apresenta este atributo de qualidade.

Projeto e Atributos de Qualidade

- - Correção Funcional (ou Acurácia): grau em que o produto de software fornece resultados corretos e precisos, conforme acordado. Um produto que apresente dados incorretos, ou com a precisão abaixo dos limites definidos como toleráveis, não apresenta este atributo de qualidade.

- - Aptidão Funcional: capacidade do produto de software de facilitar a realização das tarefas e objetivos do usuário. Refere-se às necessidades implícitas.

Projeto e Atributos de Qualidade

- • **Confiabilidade**: grau em que o produto executa as funções especificadas com um comportamento consistente com o esperado, por um período de tempo. A confiabilidade está relacionada com as falhas que um produto apresenta e como este produto se comporta em situações consideradas fora do normal. Suas subcaracterísticas são:
- - Maturidade: é uma medida da frequência com que o produto de software apresenta falhas ao longo de um período estabelecido de tempo. Refere-se, portanto, à capacidade do produto de software de evitar falhas decorrentes de defeitos no software, mantendo sua operação normal ao longo do tempo.
- - Disponibilidade: capacidade do produto de software de estar operacional e acessível quando seu uso for requerido.

nemo

Projeto e Atributos de Qualidade

- - Tolerância a falhas: capacidade do produto de software de operar em um nível de desempenho especificado em casos de falha no software ou no hardware. Esta subcaracterística tem a ver com a forma como o software reage quando ocorre uma situação externa fora do normal (p.ex., conexão com a Internet interrompida).

- - Recuperabilidade: capacidade do produto de software de se colocar novamente em operação, restabelecendo seu nível de desempenho especificado, e recuperar os dados diretamente afetados no caso de uma falha.

Projeto e Atributos de Qualidade

- • **Usabilidade**: grau em que o produto apresenta atributos que permitem que o mesmo seja entendido, aprendido e usado, e que o tornem atrativo para o usuário. Tem como subcaracterísticas:

- - Reconhecimento da Adequação: grau em que os usuários reconhecem que o produto de software é adequado para suas necessidades.

- - Inteligibilidade: refere-se à facilidade de o usuário entender os conceitos chave do produto e aprender a utilizá-lo, tornando-se competente em seu uso.

- - Operabilidade: refere-se à facilidade do usuário operar e controlar o produto de software.

Projeto e Atributos de Qualidade

- - Proteção contra Erros do Usuário: capacidade do produto de software de evitar que o usuário cometa erros.

- - Estética da Interface com o Usuário: capacidade do produto de software de ser atraente ao usuário, lhe oferecendo uma interface com interação agradável.

- - Acessibilidade: capacidade do produto de software ser utilizado por um amplo espectro de pessoas, incluindo portadores de necessidades especiais e com limitações associadas à idade.

Projeto e Atributos de Qualidade

- • **Eficiência de Desempenho**: capacidade de o produto manter um nível de desempenho apropriado em relação aos recursos utilizados em condições estabelecidas. Inclui subcaracterísticas que evidenciam o relacionamento entre o nível de desempenho do software e a quantidade de recursos utilizados, a saber:

- - Comportamento em Relação ao Tempo: capacidade do produto de software de fornecer tempos de resposta e de processamento apropriados, quando o software executa suas funções, sob condições estabelecidas.

nemo

Projeto e Atributos de Qualidade

- - Utilização de Recursos: capacidade do produto de software de usar tipos e quantidades apropriados de recursos, quando executa suas funções, sob condições estabelecidas.

- - Capacidade: refere-se ao grau em que os limites máximos dos parâmetros do produto de software (p.ex., itens que podem ser armazenados, número de usuários concorrentes etc.) atendem às condições especificadas.

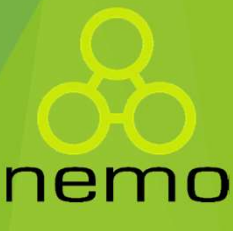
Projeto e Atributos de Qualidade

- • **Segurança**: grau em que informações e dados são protegidos contra o acesso por pessoas ou sistemas não autorizados, bem como grau em que essas informações e dados são disponibilizados para pessoas ou sistemas com acesso autorizado.

- Tem como subcaracterísticas:

- - Confidencialidade: garantir acesso a quem tem autorização para tal.

- - Integridade: impedir ao acesso a quem não tem autorização para tal.

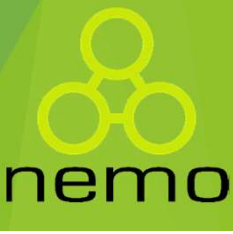


Projeto e Atributos de Qualidade

- - Não Repúdio: garantia de que a ocorrência de ações ou eventos possa ser provada, evitando-se questionamentos futuros.

- - Responsabilização: ações realizadas por uma pessoa ou sistema devem poder ser rastreadas de modo a comprovar por quem foram feitas.

- - Autenticidade: capacidade de o sistema avaliar a identidade de um usuário ou recurso.



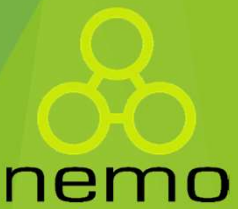
Projeto e Atributos de Qualidade

- • **Compatibilidade**: capacidade do produto de software de trocar informações com outras aplicações e/ou compartilhar o mesmo ambiente de hardware ou software.

- Suas subcaracterísticas são:

- - Coexistência: capacidade do produto de software de coexistir com outros produtos de software independentes, em um ambiente compartilhando recursos comuns.

- - Interoperabilidade: capacidade do produto de software de interagir com outros sistemas especificados, trocando e usando as informações trocadas.



Projeto e Atributos de Qualidade

- • **Manutenibilidade**: capacidade do produto de software de ser modificado. Tem como subcaracterísticas:

- - Modularidade: o produto de software deve possuir componentes coesos e fracamente acoplados, de modo que uma modificação em um componente tenha impacto mínimo em outros componentes.

- - Reusabilidade: capacidade dos componentes do produto de software serem utilizados na construção de outros componentes ou sistemas.

Projeto e Atributos de Qualidade

- - Analisabilidade: capacidade do produto de software de permitir o diagnóstico de deficiências ou de causas de falhas, bem como a identificação das partes a serem modificadas.

- - Modificabilidade: grau em que o produto de software pode ser modificado sem introduzir novos defeitos ou degradar a qualidade do produto, permitindo que uma modificação seja eficazmente implementada.

- - Testabilidade: capacidade do produto de software de permitir que, quando modificado, seja testado para determinar se as alterações foram propriamente realizadas e não introduziram efeitos colaterais.

Projeto e Atributos de Qualidade

- • **Portabilidade**: refere-se à capacidade do software ser transferido de um ambiente de hardware ou software para outro. Tem como subcaracterísticas:

- - Adaptabilidade: capacidade do produto de software de ser adaptado para diferentes ambientes especificados, sem necessidade de aplicação de outras ações ou meios além daqueles fornecidos para essa finalidade pelo próprio software.

- - Capacidade para ser instalado (instalabilidade): avalia a facilidade de se instalar (e desinstalar) o produto de software em um ambiente especificado.

- - Capacidade de substituição (substituibilidade): capacidade do produto de software de ser usado em substituição a outro produto de software especificado, com o mesmo propósito e no mesmo ambiente. Considera, também, a facilidade de atualizar uma nova versão.

nemo

Projeto e Atributos de Qualidade

- Um problema chave para o projeto é definir prioridades para tratar requisitos não funcionais conflitantes.

- Por exemplo, normalmente ao se melhorar o desempenho de uma porção de um sistema diminui-se a sua capacidade de acomodar mudanças. Ou seja, torna-se mais difícil alterar o sistema.

- *Assim, uma importante atividade do projeto de sistemas é avaliar os requisitos não funcionais e resolver requisitos conflitantes*. A base para essa decisão deve ser a importância relativa das várias características levantadas para o sistema em questão.

Projeto e Atributos de Qualidade

- Deve-se observar que, embora os requisitos não funcionais tenham cunho tecnológico, eles, assim como os requisitos funcionais, devem ser levantados junto aos clientes e usuários. Dentre outras, as seguintes informações devem ser levantadas:
- • Qual a localização geográfica dos usuários? Há necessidade de transporte de dados?
- • Quais são problemas operacionais existentes nas atividades dos usuários? Qual será o ambiente de hardware e software de produção? Há restrições técnicas (novo ambiente?) ou ambientais (p.ex., temperatura)?
- • Qual a frequência de disparo das operações do sistema? Qual o tempo de resposta esperado para cada uma delas?
- • Qual o volume de dados esperado (inicial, estimativa de crescimento e política de esvaziamento)?
- • Há restrições de confiabilidade (tempo mínimo entre falhas)?
- • Há restrições de segurança (classes de usuários e acesso)?
- • Quais as características desejadas para a interface com o usuário?

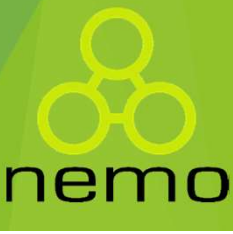
nemo

Especificação de Requisitos Não Funcionais

- Assim como os requisitos funcionais, os requisitos não funcionais (RNFs) precisam ser especificados.

- Essa especificação deve ser tal que o RNF seja passível de avaliação. Não basta dizer coisas como “o sistema deve ser fácil de manter” ou “o sistema deve ter uma interface com o usuário amigável”.

- O que significa “fácil de manter” ou “interface amigável”?



Especificação de Requisitos Não Funcionais

- Os RNFs têm de ser especificados de forma tal que seja possível posteriormente avaliar se os mesmos foram atendidos ou não.

- Uma forma de especificar requisitos não funcionais é prover uma descrição associada a um critério de aceitação. Este último deve ser testável e, para tal, medidas objetivas devem ser providas.

- A ISO/IEC 91262 pode ser uma boa fonte de medidas. As partes 2 (Medidas Externas) (ISO/IEC, 2003a) e 3 (Medidas Internas) (ISO/IEC, 2003b) dessa norma apresentam diversas medidas que podem ser usadas para especificar objetivamente os RNFs. Nessas partes da norma, medidas são sugeridas para as diversas subcaracterísticas descritas na Parte 1, indicando, dentre outros nome e propósito da medida, método de aplicação e fórmula, e como interpretar os valores da medida.

Especificação de Requisitos Não Funcionais

Tabela 2.1 – Medida "Facilidade de aprender a realizar uma tarefa em uso".

Medida	Facilidade de aprender a realizar uma tarefa em uso
Propósito	Quanto tempo o usuário leva para aprender a realizar uma tarefa especificada eficientemente?
Método de Aplicação	Observar o comportamento do usuário desde quando ele começa a aprender até quando ele começa a operar eficientemente a funcionalidade.
Medição	T = soma do tempo de operação do usuário até que ele consiga realizar a tarefa em um tempo especificado.

Tabela 2.2 – Especificação de Requisito Não Funcional.

RNF01 – A funcionalidade "Efetuar Locação de Carro" deve ser fácil de aprender.	
Medida:	Facilidade de aprender a realizar uma tarefa em uso
Critério de Aceitação:	T <= 15 minutos, considerando que o usuário está operando o sistema eficientemente quando a tarefa "Efetuar Locação de Carro" é realizada em um tempo inferior a 2 minutos.

nemo

Especificação de Requisitos Não Funcionais

Tabela 2.3 – Exemplos de Medidas para a Especificação e Avaliação de RNFs

Subcaracterística de Qualidade	Medida	Propósito da Medida	Procedimento de Medição	Fórmula de Medição	Interpretação do valor medido	Fonte de Entrada para Medição
Interoperabilidade	**Consistência de interface (protocolo)**	Quão corretamente os protocolos de interface foram implementados?	Contar o número de protocolos de interface corretamente implementados (NPIC) conforme definido nas especificações e comparar com o número total de protocolos de interface a serem implementados (NPIT) como definido nas especificações.	X=NPIC/NPIT	0 <= X <= 1. Quanto mais próximo de 1, mais consistente.	Especificação de requisitos, Design, Código fonte
Responsabilização	**Auditabilidade de acesso**	Quão auditável é o login de acesso?	Contar o número de tipos de acesso que estão sendo logados corretamente (NALC) conforme definido nas especificações e comparar com o número total de tipos de acesso que são requeridos (NART) como definido nas especificações.	X=NALC/NART	0 <= X <= 1. Quanto mais próximo de 1, mais auditável.	Especificação de requisitos, Design, Código fonte
Maturidade	**Tempo médio entre falhas**	Quanto tempo o produto de software opera sem apresentar falhas?	Calcular o tempo total de funcionamento correto em um período (TTFC) e dividir pelo número de falhas no período (NF), obtendo o tempo médio entre falhas.	X=TTFC / NF	0 <= X. Quanto maior o valor, mais maduro.	Registros de falhas.
Disponibilidade	**Probabilidade de estar disponível**	Qual a probabilidade do produto de software estar disponível?	Calcular o tempo médio entre falhas (TMF) e o tempo médio de reparo (TMR), derivando a probabilidade conforme a fórmula de medição dada.	X= TMF / (TMF+TMR)	0 <= X <= 1. Quanto mais próximo de 1, mais disponível.	Registros de falhas e de controle de alterações.
Reconhecimento da Adequação	**Funções Evidentes**	Qual a proporção de funções do produto de software que estão evidentes para os usuários?	Contar o número de funções evidentes para o usuário (NFE) e comparar com o número de funções total (NFT).	X=NFE/NFT	0 <= X <= 1. Quanto mais próximo de 1, melhor.	Especificação de requisitos, Projeto de IU
Inteligibilidade	**Inteligibilidade de funções**	Qual a proporção de funções do produto de software corretamente entendidas pelos usuários?	Contar o número de funções de interface com o usuário facilmente compreensíveis pelo usuário (NFIC) e comparar com o número de funções de interface com o usuário total (NFIT).	X=NFIC/NFIT	0 <= X <= 1. Quanto mais próximo de 1, melhor.	Projeto de IU
Operabilidade	**Checagem de validade de entrada**	Qual a proporção de itens de entrada que checa se os dados são válidos?	Contar o número de itens de entrada com checagem de validade (NIEC) e comparar com o número de itens de entrada passíveis de serem checados (NIEPC).	X=NIE/NIEPC	0 <= X <= 1. Quanto mais próximo de 1, melhor.	Especificação de requisitos, Projeto de IU
Operabilidade	**Habilidade de desfazer**	Qual a proporção de funções que podem ser desfeitas?	Contar o número funções que podem ser desfeitas pelo usuário após completadas (NFU) e comparar com o número total de funções (NTF).	X=NFU/NTF	0 <= X <= 1. Quanto mais próximo de 1, melhor.	Especificação de requisitos, Projeto de IU

nemo

Táticas para Tratar Atributos de Qualidade

- Entender os atributos de qualidade e definir os níveis em que eles devem ser atingidos é muito importante.
- Contudo, é necessário definir como atingir esses níveis. Assim, deve-se definir que estratégias serão aplicadas para atingir essas metas.
- Existe um conjunto de táticas descritas na literatura que podem ser aplicadas para tratar certos atributos de qualidade.

Táticas para Tratar Atributos de Qualidade

Confiabilidade

- Um defeito (fault) é uma imperfeição do sistema e, em geral, refere-se a algo que está implementado de maneira incorreta.
- Uma falha (failure) é um resultado incorreto, visível, provocado por um defeito.
- Uma falha ocorre quando o sistema não entrega mais um serviço consistente com sua especificação e essa falha é perceptível ao usuário.
- As táticas de confiabilidade são divididas em três grupos: táticas de detecção de falhas, táticas de recuperação de falhas e táticas de prevenção de falhas.

Táticas para Tratar Atributos de Qualidade

Detecção de Falha	
Silvo/Eco (*Ping/echo*)	Um componente envia um silvo (som agudo prolongado) e espera receber um eco de volta, dentro de um tempo predefinido, do componente sendo examinado. Ex.: clientes enviam um silvo para verificar se o servidor e o caminho de comunicação até ele estão operando dentro do limites de desempenho esperados.
Batida de Coração (*Heartbeat*)	Um componente emite mensagens periódicas (batida de coração) e outro componente as escuta. Se a batida de coração falhar, assume-se que o componente de origem falhou e um componente de recuperação de falha é notificado.
Exceções (*Exceptions*)	Uma exceção é gerada quando uma falha é detectada. Um manipulador de exceção é então acionado para tratá-la. O manipulador de exceção opera dentro do mesmo processo que gerou a exceção e geralmente realiza uma transformação da falha em uma forma que possa ser processada.
Votação (*Voting*)	Processos rodando em processadores redundantes recebem entradas equivalentes e computam a saída que é enviada para um votante (*voter*). Se o votante detectar um comportamento desviante de um dado processador, ele gera uma falha no correspondente componente.

nemo

Táticas para Tratar Atributos de Qualidade

Recuperação de Falha	
Redundância Ativa (*Active Redundancy*)	Componentes redundantes respondem a eventos em paralelo. Apenas a resposta de um deles é utilizada (geralmente o primeiro a responder) e o restante é descartado. Em sistemas distribuídos que precisam de alta disponibilidade, a redundância pode ocorrer nos caminhos de comunicação.
Redundância Passiva (*Passive Redundancy*)	Um componente (o principal) responde aos eventos e informa os demais (os componentes em espera – *standbys*) sobre as atualizações de estado que ele fez. Quando uma falha ocorre, o sistema precisa primeiro garantir que o estado de backup é suficientemente recente para prosseguir.
Sobressalente (*Spare*)	Uma plataforma de computação sobressalente em espera é configurada para substituir componentes com falha. Essa plataforma tem de ser iniciada com a configuração apropriada e o estado recuperado. Para tal, periodicamente o estado do sistema deve ser armazenado, bem como se devem registrar as alterações feitas no dispositivo de persistência.
Operação em Sombra (*Shadow Operation*)	Um componente que falhou anteriormente, para voltar a executar normalmente, precisa primeiro rodar em "modo sombra" por um pequeno período de tempo para garantir que ele tem o mesmo comportamento do componente que está funcionando correntemente.
Ressincronização de Estado (*State Resynchronization*)	As táticas de redundância ativa e passiva requerem que um componente sendo restaurado tenha seu estado atualizado antes de retornar ao serviço.
Ponto de Verificação / Reversão (*Checkpoint / Rollback*)	Um ponto de verificação, criado periodicamente ou em resposta a eventos específicos, registra um estado consistente. O sistema, após a ocorrência de uma falha, deve ser restaurado usando um ponto de verificação de um estado consistente e um registro (*log*) das transações que ocorreram após o ponto de verificação.

Táticas para Tratar Atributos de Qualidade

Prevenção de Falha	
Retirada de Serviço (*Removal from service*)	Um componente é removido do sistema em operação para ser submetido a atividades que previnam falhas antecipadamente. Ex.: reiniciar um componente para evitar, por exemplo, estouro de pilha.
Transações (*Transactions*)	Uma transação é a agregação de diversos passos sequenciais de modo que o pacote como um todo possa ser desfeito de uma só vez. Transações são usadas para prevenir que dados sejam afetados caso um dos passos do processo falhe, bem como para prevenir colisões entre *threads* simultâneas acessando os mesmos dados.
Monitor de processo (*Process monitor*)	Uma vez que uma falha é detectada em um processo, um processo de monitoramento pode excluir o processo que não está executando e criar uma nova instância do mesmo.

nemo

Táticas para Tratar Atributos de Qualidade

Desempenho

- Desempenho está dentre os mais importantes requisitos não funcionais.
- Ele depende fortemente da interação (e do tipo da interação) existente entre os componentes de um sistema e, portanto, está muito associado à arquitetura do mesmo.
- De maneira geral, requisitos de desempenho impõem restrições relativas à velocidade de operação de um sistema.
- • Requisitos de tempo de resposta: especificam o tempo de resposta aceitável para certas funcionalidades do sistema;

Táticas para Tratar Atributos de Qualidade

Desempenho

- • Requisitos de processamento (*throughput*): especificam restrições relativas à quantidade de processamento (p.ex., número de transações) realizada em um determinado período de tempo;

- • Requisitos de temporização: especificam quão rapidamente o sistema deve coletar dados de entrada de sensores, antes que novas leituras sejam feitas, sobrescrevendo os dados anteriores;

- • Requisitos de recursos: estabelecem condições (memórias principal e secundária, velocidade do processador etc.) para o bom funcionamento do sistema.

Táticas para Tratar Atributos de Qualidade

Desempenho

- As táticas de desempenho propostas são agrupadas em:
- • Táticas relativas à demanda de recursos: visam tentar diminuir a demanda por recursos. Como os fluxos de eventos são a fonte da demanda por recursos, essas táticas se preocupam em diminuir a frequência da ocorrência de eventos ou a quantidade de recursos consumidos por cada requisição.
- • Táticas relativas à gerência de recursos: visam gerenciar os recursos, normalmente tornando mais recursos disponíveis, de modo a melhorar o desempenho.
- • Táticas relativas à arbitragem de recursos: sempre que houver disputa por recursos, é necessário escaloná-los. Processadores, *buffers* e redes são escalonados. Assim, essas táticas referem-se à escolha da política de escalonamento compatível com cada recurso, visando melhorar o desempenho.

nemo

Táticas para Tratar Atributos de Qualidade

Demanda de Recursos	
Aumentar a eficiência computacional	Uma forma de diminuir o tempo de resposta consiste em aperfeiçoar os algoritmos usados em partes criticas do software.
Reduzir *overhead* computacional	O uso de intermediários, tão importantes para a manutenibilidade, aumenta o consumo de recursos no processamento de um evento e, portanto, removê-los pode ajudar a diminuir o tempo de resposta. Considerações análogas valem para camadas, sobretudo quando utilizada uma arquitetura em camadas fechada.
Reduzir o número de eventos processados	Se for possivel reduzir a frequência de amostragem na qual variáveis do ambiente são monitoradas, então a demanda pode ser reduzida. Quando não se tem controle sobre a chegada de eventos gerados externamente, as solicitações podem ser tratadas com uma frequência menor, com perda de algumas requisições.

Táticas para Tratar Atributos de Qualidade

Gerência de Recursos	
Introduzir concorrência	Concorrência pode ser introduzida, processando diferentes conjuntos de eventos em diferentes linhas de controle (*threads*) ou criando linhas de controle adicionais para diferentes conjuntos de atividades. Uma vez introduzida concorrência, balancear a carga é importante para explorá-la de maneira ótima. Contudo, essa tática só pode ser aplicada se as requisições puderem ser processadas em paralelo.
Manter múltiplas réplicas de dados e serviços	O propósito de uma réplica é reduzir a contenção que ocorreria se todas as computações ocorressem em um computador central. O uso de *cache* e fragmentação de bases de dados são exemplos dessa tática. *Caching* é uma tática na qual dados são replicados para reduzir contenção. Uma vez que, neste contexto, os dados colocados em *cache* são cópias de dados existentes, manter as cópias consistentes e sincronizadas torna-se uma responsabilidade do sistema.
Aumentar a disponibilidade de recursos.	Processadores mais rápidos, processadores adicionais, memória adicional e redes mais rápidas são táticas para aumentar a disponibilidade de recursos.

nemo

Táticas para Tratar Atributos de Qualidade

Arbitragem de Recursos	
Escolher política de escalonamento	Todas as políticas de escalonamento atribuem prioridades. Além disso, para que um evento de mais alta prioridade seja disparado, é necessário interromper o processo usuário corrente do recurso (preempção). Algumas políticas comuns de escalonamento são: FIFO (*first-in first out*), prioridades fixas e prioridades dinâmicas.

Táticas para Tratar Atributos de Qualidade

Segurança

- A segurança tem como objetivo não permitir que pessoas ou sistemas não autorizados tenham acesso a eventos ou dados do sistema.

- Para controlar o acesso (táticas para resistir a ataques), é necessário identificar, autenticar e autorizar usuários. A identificação se dá através de uma forma unívoca, ou seja, que identifique apenas um usuário.

- A autenticação consiste em comprovar que o usuário é mesmo quem ele diz ser na identificação, sendo feita, por exemplo, por meio de uma senha.
- Finalmente, a autorização é dada ao usuário, ou a uma classe de usuários, dando acesso a determinadas funcionalidades do sistema.

nemo

Resistir a Ataques	
Autenticar usuários	Autenticação diz respeito a garantir que um usuário ou computador remoto é realmente quem diz ser. Senhas, certificados digitais e identificação biométrica são alguns meios de se prover autenticação.
Autorizar usuários	Autorização diz respeito a garantir que um usuário autenticado tenha o direito de acessar e modificar tanto dados quanto serviços. Isso é feito normalmente provendo alguns padrões de controle de acesso dentro do sistema. O controle de acesso pode ser por usuário ou classe de usuário. Classes de usuários podem ser definidas por grupos de usuários, por papéis de usuário ou por listas de indivíduos.
Manter confidencialidade dos dados	Dados devem ser protegidos contra acesso não autorizado. Confidencialidade é normalmente atingida aplicando alguma forma de criptografia a dados e links de comunicação. Criptografia provê uma proteção extra para dados mantidos em bases de dados, além da autorização. Já links de comunicação tipicamente não têm controles de autorização e, portanto, a criptografia é a única proteção neste caso.
Manter integridade dos dados	A integridade dos dados também deve ser garantida. Para verificar a integridade, informação redundante, tal como soma de verificação, pode ser codificada junto aos dados.
Limitar exposição	A alocação de serviços a servidores pode ser feita de modo que apenas serviços limitados estejam disponíveis em cada servidor.
Limitar acesso	Firewalls podem ser usados para restringir o acesso com base em fonte de mensagens ou porta de destinação. Mensagens de fontes desconhecidas podem ser uma forma de ataque. Contudo, nem sempre é possível limitar o acesso a fontes conhecidas. Um *site* web público, por exemplo, pode esperar receber solicitações de fontes desconhecidas.

Detectar Ataques	
Sistema de detecção de intrusão	Sistemas de detecção de intrusão funcionam comparando padrões de tráfego de rede. No caso de detecção de mau uso, o padrão de tráfego é comparado com padrões históricos de ataques conhecidos. Detectores de intrusão têm de ter, dentre outros, algum tipo de sensor para detectar ataques, bases de dados para registrar eventos para posterior análise, ferramentas para análise e um console de controle que permita ao analista modificar ações de detecção de intrusão.
Recuperar-se de Ataques	
Restaurar estado	As técnicas de recuperação de falhas sugeridas nas táticas de confiabilidade podem ser aplicadas, já que elas se propõem a restaurar um estado consistente a partir de um estado inconsistente. Atenção especial deve ser dada à manutenção de cópias redundantes de dados administrativos do sistema, tais como senhas, listas de controle de acesso, serviços de nomes de domínio e dados de perfis de usuários.
Manter uma trilha de auditoria	Uma trilha de auditoria é um registro das transações aplicadas aos dados do sistema junto com informações de identificação. Essas informações podem ser usadas para trilhar as ações do agressor, apoiar reconhecimento (provendo evidência de que uma particular requisição foi feita) e apoiar a recuperação do sistema. Trilhas de auditoria são frequentemente alvo de ataques e, portanto, devem ser mantidas de modo confiável.

Táticas para Tratar Atributos de Qualidade

Usabilidade

- A usabilidade está preocupada com o quão fácil é para o usuário realizar uma tarefa desejada e o tipo de apoio provido pelo sistema ao usuário.

- Envolve, dentre outros, aspectos relativos à facilidade de entender, aprender e operar o sistema.

- As táticas para apoiar a usabilidade são organizadas em dois grandes grupos: táticas de tempo de execução e táticas de tempo de projeto.

Táticas para Tratar Atributos de Qualidade

Usabilidade

- As táticas de tempo de execução se referem ao apoio ao usuário durante a execução do sistema.

- Uma vez que o sistema estiver executando, a usabilidade é reforçada dando feedback ao usuário sobre o quê o sistema está fazendo e provendo ao usuário a capacidade de entrar com comandos que permitam operar o sistema de modo mais eficiente ou corrigir erros, tais como cancelar e desfazer.

- As táticas de usabilidade em tempo de execução são agrupadas em táticas para apoiar iniciativas do usuário e táticas para apoiar iniciativas do sistema.

Táticas para Tratar Atributos de Qualidade

Usabilidade

- Quando um usuário toma uma iniciativa, o projetista projeta a resposta tomando por base a funcionalidade do sistema.

- Quando o sistema toma a iniciativa, o projetista precisa apoiar-se em alguma informação sobre o usuário, a tarefa sendo executada pelo usuário ou sobre o estado do sistema.

Táticas para Tratar Atributos de Qualidade

Usabilidade

- As táticas de apoio a iniciativas do sistema são:
- • Manter um modelo da tarefa: o modelo da tarefa é usado para determinar o contexto, de modo que o sistema pode inferir o que o usuário está tentando fazer e prover assistência. Ex.: sabendo que frases começam com letras maiúsculas, um editor de texto pode corrigir um texto sendo digitado.
- • Manter um modelo do usuário: esse modelo mantém informações sobre o conhecimento que o usuário tem do sistema, sobre o comportamento do usuário etc. Esse modelo é usado para prover assistência.
- • Manter um modelo do sistema: esse modelo mantém informações sobre o comportamento esperado do sistema, de modo a dar um feedback para o usuário. Ex.: prever o tempo necessário para completar uma atividade.

nemo

Táticas para Tratar Atributos de Qualidade

Usabilidade

- As táticas de usabilidade de tempo de projeto estão fortemente relacionadas às táticas de manutenibilidade. São táticas dessa natureza:

- • Separar a interface do restante da aplicação. Diversos padrões arquitetônicos foram desenvolvidos para implementar essa tática, dentre eles o padrão Modelo-Visão-Controlador (MVC).

- • Adotar padrões de interface (*design patterns* de interface com o usuário) amplamente utilizados. Para tal, é necessário considerar a(s) plataforma(s) de computação na(s) qual(is) o sistema vai rodar. Ex.: Padrões de navegação, busca, tabelas e listas para aplicativos Android e iOS.

Táticas para Tratar Atributos de Qualidade

Usabilidade

- • Prover ao usuário a capacidade de operar o sistema de modo mais eficiente. De fato, há várias táticas que se enquadram nesta categoria, tais como:
- (i) Permitir, sempre que possível, a entrada por meio de seleção ao invés da digitação de campos;
- (ii) Prover acesso mais fácil a funcionalidades importantes para o usuário;
- (iii) Não disponibilizar funcionalidades ou dados que não possam ser usados pelo usuário;
- (iv) Prover teclas de atalho (ou outros mecanismos de interação rápida) quando pertinente;
- (v) Apresentar mensagens de erro e avisos significativas, dentre outras.

nemo

Táticas para Tratar Atributos de Qualidade

Manutenibilidade

- Para se obter sistemas fáceis de manter, deve-se, dentre outros, facilitar a localização das alterações (analisabilidade), a realização das alterações (modificabilidade) e os testes (testabilidade).

- Existem diversas táticas para apoiar a manutenibilidade, organizadas de acordo c om suas metas, dentre elas:

- • Localização de Alterações: tem como objetivo reduzir o número de módulos que são diretamente afetados por uma alteração.

- Uma tática desse grupo consiste em manter a coerência semântica, procurando garantir que um módulo trabalha em conjunto com outros, mas sem depender excessivamente deles (independência funcional, alta coesão, baixo acoplamento).

Táticas para Tratar Atributos de Qualidade

Manutenibilidade

- • Prevenção de efeito cascata: tem como objetivo limitar a abrangência de uma modificação somente aos módulos localizados, evitando afetar outros módulos não diretamente envolvidos.

- São exemplos de táticas desse grupo:

- (i) ocultação de informação;
- (ii) garantia das interfaces existentes; e
- (iii) uso de intermediários, tais como repositórios para dados e padrões de projeto (*design patterns*) para intermediar serviços (p.ex., padrão fachada, mediador etc.).

Táticas para Tratar Atributos de Qualidade

Manutenibilidade

- • Táticas de Testabilidade: visam facilitar os testes de uma porção de software.

- Dentro desse grupo, merecem destaque táticas relacionadas às entradas e saídas, tais como:

- (i) separar interface da implementação, o que permite a substituição da implementação para vários propósitos de teste, tal como o uso de *stubs*;

- (ii) registro e reprodução, que diz respeito a capturar a informação cruzando uma interface (entradas e saídas) durante a operação normal em algum repositório e utilizá-la para testes.

nemo

Táticas para Tratar Atributos de Qualidade

Portabilidade

- Para se obter sistemas fáceis de portar, deve-se, dentre outros, facilitar a instalação, a substituição de partes do sistema e a adaptação a diferentes plataformas.

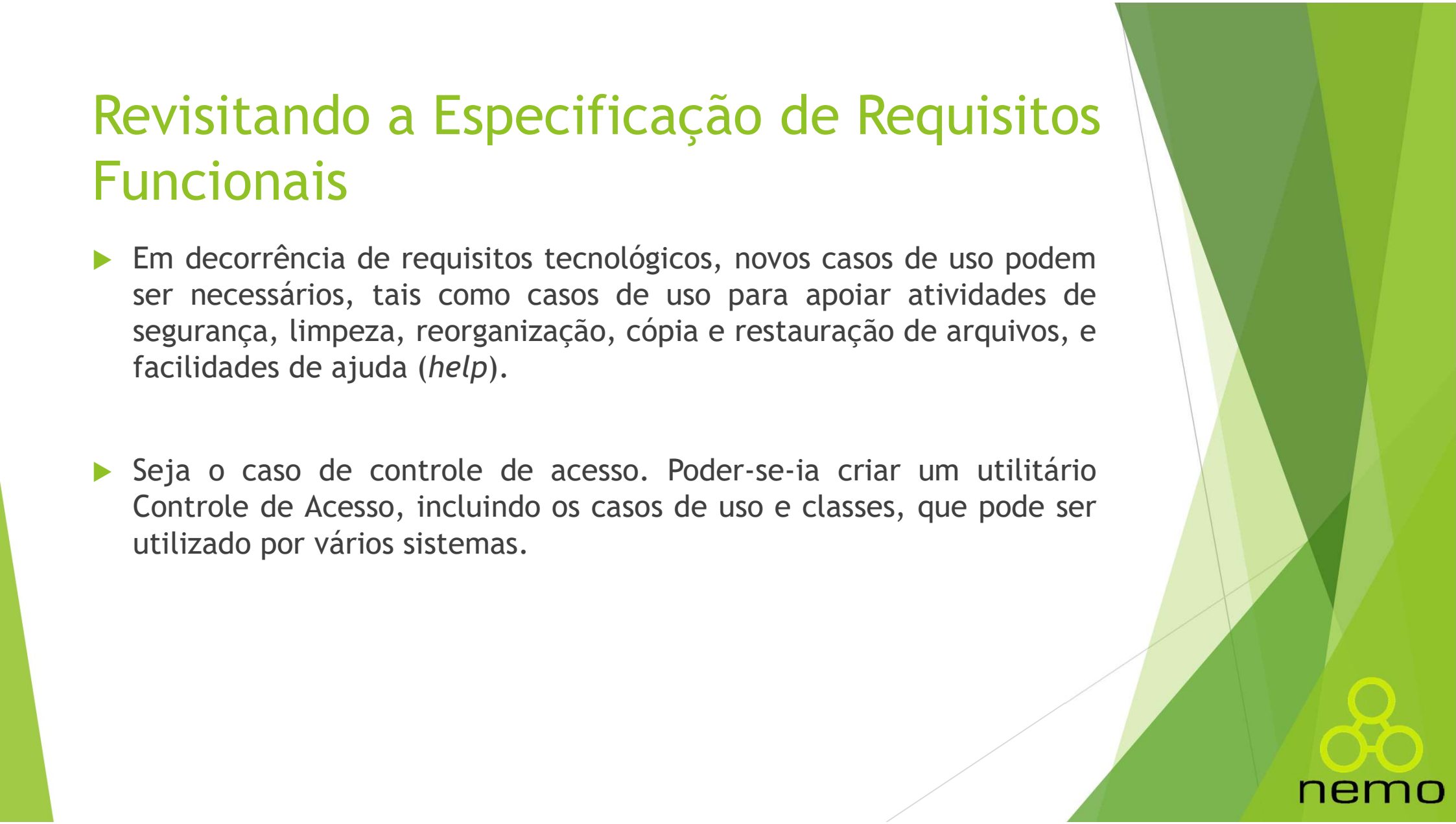
- As táticas de portabilidade estão bastante relacionadas às táticas de manutenibilidade. De fato, algumas delas são as mesmas, tal como garantir interfaces existentes, usar intermediários e separar a interface da aplicação.

- Além disso, o uso de linguagens, bibliotecas e mecanismos de persistência capazes de rodar em diferentes plataformas operacionais é uma importante tática para tratar a portabilidade.

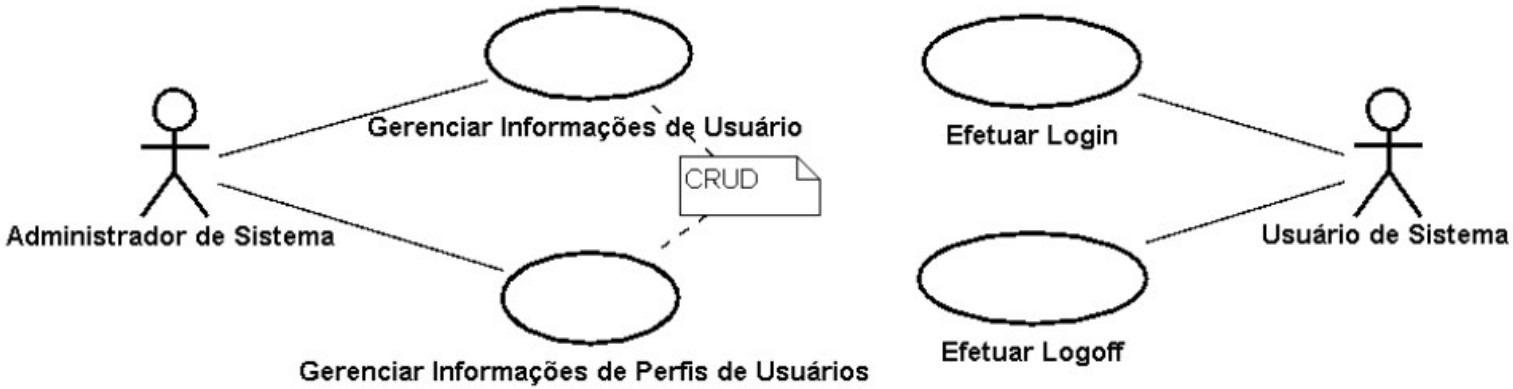
Revisitando a Especificação de Requisitos Funcionais

- Em decorrência de requisitos tecnológicos, novos casos de uso podem ser necessários, tais como casos de uso para apoiar atividades de segurança, limpeza, reorganização, cópia e restauração de arquivos, e facilidades de ajuda (*help*).

- Seja o caso de controle de acesso. Poder-se-ia criar um utilitário Controle de Acesso, incluindo os casos de uso e classes, que pode ser utilizado por vários sistemas.

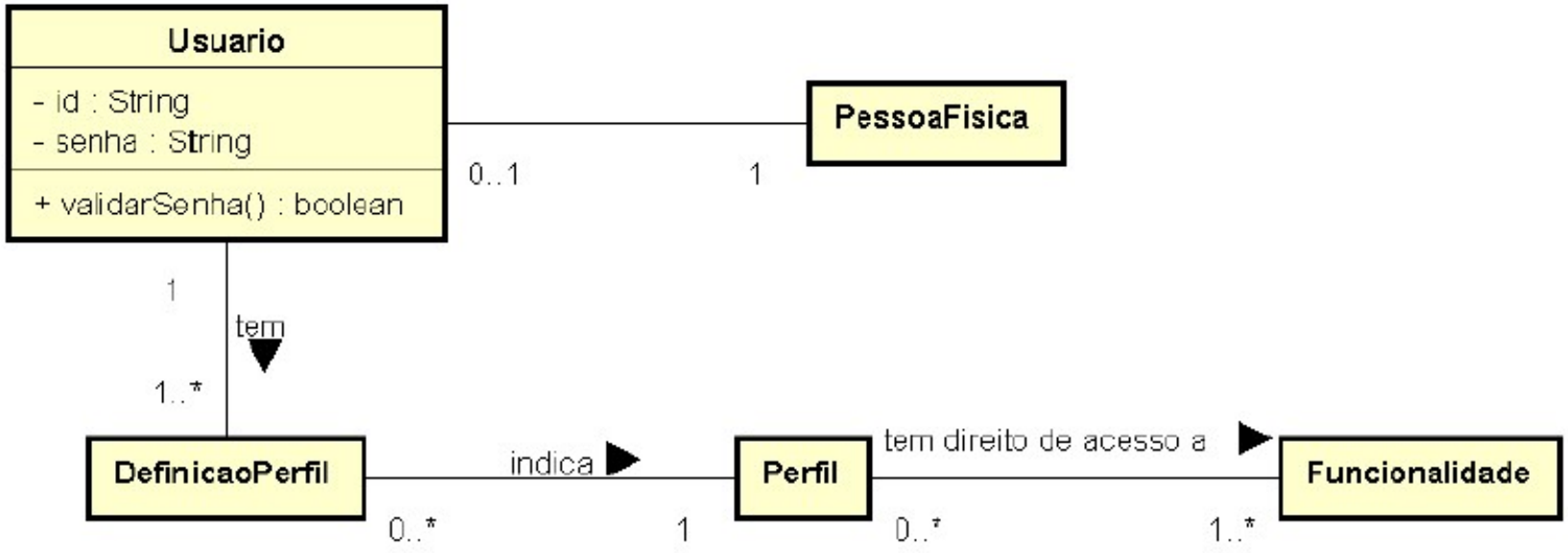


Revisitando a Especificação de Requisitos Funcionais



nemo

Revisitando a Especificação de Requisitos Funcionais



nemo

Revisitando a Especificação de Requisitos Funcionais

- Para capturar aspectos específicos do sistema em desenvolvimento, p.ex., no que tange à definição de perfis, uma matriz *Caso de Uso x Perfil de Usuário* poderia ser utilizada para documentar o nível de autorização adotado no projeto.

	Perfis de Usuários		
Casos de Uso	Cliente	Gerente	Funcionário
Efetuar pedido	X	X	X
Cancelar pedido		X	X
Alterar pedido	X	X	X
Solicitar relatório de pedidos		X	

nemo

Revisitando a Especificação de Requisitos Funcionais

- É bom lembrar que todo caso de uso tecnológico projetado é decorrente de um requisito tecnológico acrescido ao sistema, mas nem todo requisito tecnológico necessariamente implica na criação de um caso de uso.

- Alguns atributos de qualidade, tais como desempenho e usabilidade, são tipicamente incorporados a casos de uso do sistema já existentes.

- Por fim, a escolha de táticas para tratar RNFs também ajuda a melhorar a especificação desses requisitos. Assim, é importante adicionar à especificação de RNFs informações sobre como cada RNF será tratado.

Revisitando a Especificação de Requisitos Funcionais

RNF01 – O sistema deve disponibilizar a venda de livros pela Web, a partir dos principais navegadores disponíveis no mercado, considerando, inclusive, dispositivos móveis (tablets e smartphones).	
Categoria:	Portabilidade
Tratamento:	Os navegadores a serem considerados são: Google Chrome, Microsoft Edge / Internet Explorer, Mozilla Firefox e Safari. Deverão ser considerados os correlatos ao Chrome e Safari nas plataformas Android e iOS para tablets e smartphones, respectivamente. O desempenho do usuário na realização de tarefas deve ser equivalente. Para manter a independência dos demais componentes, a interface com o usuário deverá ser separada do restante da aplicação.
Medida:	Percentual de tarefas não completadas corretamente por um usuário quando usando um novo navegador.
Critério de Aceitação:	O sistema deverá funcionar essencialmente da mesma maneira nos navegadores Google Chrome, Microsoft Edge e Mozilla Firefox. Nesses três navegadores não poderá haver variações superiores a 1%. Serão admitidas variações em até 5% no Safari e no Internet Explorer. Para dispositivos móveis, serão admitidas variações de até 10%.

Projeto de Software e Padrões (Patterns)

- Todo projeto de desenvolvimento é, de alguma maneira, novo, na medida em que se quer desenvolver um novo sistema, seja porque ainda não existe um sistema para resolver o problema que está sendo tratado, seja porque há aspectos indesejáveis nos sistemas existentes.

- Isso não quer dizer que o projeto tenha que ser desenvolvido a partir do zero. Muito pelo contrário.

- A reutilização é um aspecto fundamental no desenvolvimento de software e, em especial, na fase de projeto.

Projeto de Software e Padrões (Patterns)

- Muitos sistemas previamente desenvolvidos são similares ao sistema em desenvolvimento e há muito conhecimento que pode ser reaplicado para solucionar questões recorrentes no projeto de software.

- Os padrões (*patterns*) visam capturar esse conhecimento, procurando torna-lo mais geral e amplamente aplicável, desvinculando-o das especificidades de um determinado projeto ou sistema.

- Um padrão é uma solução testada e aprovada para um problema geral.

Projeto de Software e Padrões (Patterns)

- Diferentes padrões se destinam a diferentes fases do ciclo de vida: análise, projeto da arquitetura, projeto detalhado e implementação.
- Um padrão vem com diretrizes sobre quando usá-lo, bem como vantagens e desvantagens de seu uso.
- Um padrão já foi cuidadosamente considerado por outras pessoas e aplicado diversas vezes na solução de problemas anteriores de natureza similar.
- Assim, tende a ser uma solução de qualidade, com maiores chances de estar correto e estável do que uma solução nova, específica, ainda não testada.

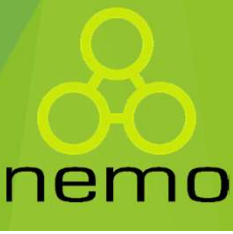
Projeto de Software e Padrões (Patterns)

- Um padrão normalmente tem o formato de um par nomeado problema/solução, que pode ser utilizado em novos contextos, com orientações sobre como utilizá-lo nessas novas situações.

- O objetivo de um padrão de projeto é registrar uma experiência no projeto de software que possa ser efetivamente utilizada por projetistas.

- Cada padrão sistematicamente nomeia, explica e avalia uma importante situação de projeto que ocorre repetidamente em sistemas.

- Um projetista familiarizado com padrões pode aplicá-los diretamente a problemas sem ter que redescobrir as abstrações e os objetos que as capturam. Uma vez que um padrão é aplicado, muitas decisões de projeto decorrem automaticamente.



Projeto de Software e Padrões (Patterns)

- Em geral, um padrão tem os seguintes elementos:
- • **Nome**: identificação de uma ou duas palavras, utilizada para nomear o padrão.
- • **Contexto**: uma situação que dá origem a um problema.
- • **Problema**: explica o problema que surge repetidamente no dado contexto.
- • **Solução**: descreve uma solução comprovada para o problema, incluindo os elementos que compõem o projeto, seus relacionamentos, responsabilidades e colaborações.
- É importante observar que um padrão não descreve um particular projeto concreto ou implementação. Um padrão provê uma descrição abstrata de um problema de projeto e como uma organização geral de elementos resolve esse problema.
- • **Consequências**: são os resultados e os comprometimentos feitos ao se aplicar o padrão.

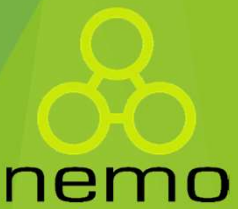
Projeto de Software e Padrões (Patterns)

- No que concerne aos padrões relacionados à fase de projeto, há três grandes categorias a serem consideradas:

- • **Padrões Arquitetônicos**: definem uma estrutura global do sistema.

- Um padrão arquitetônico indica um conjunto pré-definido de elementos, especifica as suas responsabilidades e inclui regras e diretrizes para estabelecer relacionamentos entre eles.

- São aplicados na atividade de projeto da arquitetura de software e de seus elementos, e podem ser vistos como modelos (templates) para arquiteturas de software concretas.



Projeto de Software e Padrões (Patterns)

- • **Padrões de Projeto** (*Design Patterns*): atendem a uma situação específica de projeto, mostrando classes e relacionamentos, seus papéis e suas colaborações e também a distribuição de responsabilidades.

- Um padrão de projeto descreve uma estrutura recorrente de classes que se comunicam, a qual resolve um problema de projeto geral dentro de um particular contexto.

- • **Idiomas**: representam o nível mais baixo de padrões, endereçando aspectos tanto de projeto quanto de implementação.
- Um idioma é um padrão de baixo nível, específico de uma linguagem de programação, descrevendo como codificar aspectos particulares de componentes ou os relacionamentos entre eles, usando as características de uma dada linguagem de programação.

nemo

Documentação de Projeto

- Uma vez que o projeto de software encontra-se no núcleo técnico do processo de desenvolvimento, sua documentação tem grande importância para o sucesso de um projeto e para a manutenção futura do sistema.

- Diferentes interessados vão requerer diferentes informações e a documentação de projeto é crucial para a comunicação.

- Analistas, projetistas e clientes vão precisar negociar prioridades entre requisitos conflitantes; programadores e testadores vão utilizar a documentação para implementar/testar o sistema; gerentes de projeto vão usar informações da decomposição do sistema para definir e alocar equipes de trabalho; mantenedores vão recorrer a essa documentação na hora de avaliar e realizar uma alteração, dentre outros usos.

Documentação de Projeto

- Uma vez que o projeto é um processo de refinamento, a sua documentação também deve prover representações em diferentes níveis de abstração.

- Além disso, o projeto de um sistema é uma entidade complexa que não pode ser descrita em uma única perspectiva. Ao contrário, múltiplas visões são essenciais e a documentação deve abranger aquelas visões consideradas relevantes.

- De fato, como muitas visões são possíveis, a documentação é uma atividade que envolve a escolha das visões relevantes e a documentação das visões selecionadas.

Documentação de Projeto

- A escolha das visões é dependente de vários fatores, dentre eles, do tipo de sistema sendo desenvolvido, dos atributos de qualidade considerados e do público alvo da documentação de projeto.

- Diferentes visões realçam diferentes elementos de um sistema. De maneira geral, o documento de projeto deve conter:

- • Informações gerenciais, tais como versão, responsáveis, histórico de alterações;
- • Uma descrição geral do sistema;
- • Uma lista das visões consideradas e informações acerca do mapeamento entre elas;

Documentação de Projeto

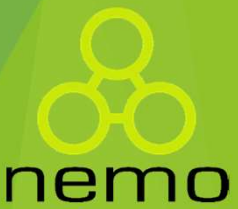
- • Para cada visão, deve-se ter:
  - Uma representação básica da visão, que pode ser gráfica, tabular ou textual, sendo a primeira a mais usual, sobretudo na forma de um diagrama UML.
    - Se for usada uma representação gráfica não padronizada, deve-se ter uma legenda explicando a notação ou simbologia usada.
  - Uma descrição sucinta dos elementos presentes na visão.

Documentação de Projeto

- Três grupos de visões que tipicamente devem ser levadas em consideração, a saber:

- • **Visão de Módulos**: os elementos considerados nessa visão são módulos. Um módulo é tipicamente atribuída uma responsabilidade funcional.
- Essa visão inclui, dentre outras, estruturas de decomposição (módulos decompostos em submódulos) e uso (um módulo usa outro módulo);

- • **Visão de Componente e Conector** (C&C): os elementos considerados nessa visão são componentes de tempo de execução (unidades de computação) e conectores (veículos de comunicação entre componentes).



Documentação de Projeto

- • **Visão de Alocação:** mostra o relacionamento entre elementos de software e elementos do ambiente externo no qual o software está sendo criado ou executado.

- Estruturas de alocação incluem aspectos relacionados com implantação (mostrando como componentes de tempo de execução são alocados a unidades de hardware), implementação (mostrando como módulos são mapeados para estruturas de arquivos) e designação de trabalho (mostrando as equipes responsáveis por implementar e integrar módulos).

Documentação de Projeto

- Além das informações anteriormente relacionadas, uma especificação de projeto deve:

- • contemplar os requisitos contidos na especificação de requisitos, sendo que, muitas vezes, podem ser levantados novos requisitos, sobretudo requisitos não funcionais, durante a fase de projeto;

- • ser um guia legível e compreensível para aqueles que vão codificar, testar e manter o software;

- • prover um quadro completo do software, segundo uma perspectiva de implementação.

That’s all Folks!