INSTITUTO DE GESTÃO E TECNOLOGIA DA INFORMAÇÃO

Requisitos Arquiteturais e Modelagem Arquitetural

Bootcamp Arquiteto de Software

Prof. Dr. João Paulo Aramuni

2020

Requisitos Arquiteturais e Modelagem Arquitetural

Bootcamp Arquiteto de Software

Prof. Dr. João Paulo Aramuni

Sumário

Capítulo 1. O que é e para que serve a Engenharia de Requisitos 4
Capítulo 2. Processo de Engenharia de Requisitos 6
Capítulo 3. Levantamento de Requisitos 10
Capítulo 4. Análise de Requisitos 14
Capítulo 5. Engenharia de Requisitos Tradicional vs Ágil 16
Capítulo 6. Documentação de Requisitos em Projetos Ágeis 20
Capítulo 7. Gerência de Requisitos no Dev. Ágil de Software 24
Capítulo 8. Gestão de Riscos na Documentação Ágil de Software 28
Capítulo 9. Requisitos Funcionais 31
Capítulo 10. Classificação de Requisitos Funcionais 34
Capítulo 11. Requisitos Não Funcionais 37
Capítulo 12. Classificação de Requisitos Não Funcionais 40
Capítulo 13. Modelo de Classificação FURPS+ 45
Capítulo 14. Metas SMART 48
Capítulo 15. O que é Backlog Grooming 51
Capítulo 16. Reuniões de Backlog Grooming 54
Capítulo 17. O que é UML 57
Capítulo 18. Diagramas de Caso de Uso 60
Capítulo 19. As cinco visões do Modelo 4+1 63
Capítulo 20. Modelo 4+1 – O que usar 67
Referências 70

Capítulo 1. O que é e para que serve a Engenharia de Requisitos

A elicitação de requisitos é considerada por alguns autores como a parte mais crítica no desenvolvimento software, pois a qualidade do produto final depende fortemente da qualidade dos requisitos elicitados (FERGUSON; LAMI, 2006).

Pesquisas apontam que 85% dos problemas de software tem origem na atividade de elicitação de requisitos (FERNANDES; MACHADO; SEIDMAN, 2009).

Os requisitos guiam as atividades do projeto e normalmente são expressos em linguagem natural para que todos possam obter o entendimento.

Os requisitos são as bases para todo projeto, definindo o que as partes interessadas de um novo sistema necessitam e também o que o sistema deve fazer para satisfazer as suas necessidades.

Além disso, também devemos definir os riscos e prover soluções satisfatórias para cada um deles. Dessa forma, os requisitos definem as bases para:

- Planejamento do Projeto;
- Gerenciamento de Riscos;
- Testes de Aceitação;
- Controle de Mudanças.

Portanto, a engenharia de requisitos é o processo pelo qual os requisitos de um produto de software são:

- Coletados;
- Analisados;
- Documentados;
- Gerenciados.

Entendido o que é a engenharia de requisitos, podemos partir para conhecer como funciona o processo de engenharia de requisitos.

Fazendo um paralelo, tem-se que um processo de software envolve diversas atividades que podem ser classificadas em:

Atividades de Desenvolvimento onde temos atividades que contribuem para o desenvolvimento do produto de software como levantamento e análise de requisitos, projeto e implementação;

Atividades de Gerência que envolvem atividades de planejamento e acompanhamento gerencial do projeto; e

Atividades de Controle da Qualidade que estão relacionadas com a avaliação da qualidade do produto.

Podemos concluir que:

- De uma forma geral, os requisitos possuem um papel fundamental para o desenvolvimento de software.
- Os requisitos de software são uma das principais medidas de sucesso de um software.
- Requisitos são a base para estimativas, modelagem, projeto, implementação, testes e até mesmo para a manutenção.
- Requisitos estão presentes ao longo de todo o ciclo de vida de um software.

Capítulo 2. Processo de Engenharia de Requisitos

Ao dar início a um projeto, temos que: I) levantar os requisitos, II) entendê-los e III) documentá-los.

Como os requisitos são extremamente importantes para o sucesso de um projeto, devemos também realizar atividades de controle da qualidade para verificar, validar e garantir a qualidade dos requisitos.

Outra medida fundamental é gerenciarmos a evolução dos requisitos, visto que os negócios são dinâmicos e não temos como garantir que esses requisitos não sofrerão alterações.

Dessa forma, devemos manter a rastreabilidade entre os requisitos e os demais artefatos produzidos no projeto.

Portanto, podemos constatar que os requisitos envolvem atividades de desenvolvimento através do Levantamento e Análise e Documentação de Requisitos, gerência através da Gerência de Requisitos e, por fim, o controle da qualidade através da Verificação, Validação e Garantia da Qualidade de Requisitos.

Todas essas atividades, que são relacionadas a requisitos, é o que podemos chamar de Processo de Engenharia de Requisitos.

Podemos dividir o processo de engenharia de requisitos em sete etapas principais:

- Concepção;
- Elicitação;
- Elaboração;
- Negociação;
- Especificação;

- Validação;
- Gerenciamento.

Concepção:

Nesta etapa, identifica-se os stakeholders e seus diferentes pontos de vista sobre o problema.

Então, desenha-se a visão geral do sistema a ser desenvolvido caracterizado por necessidades/demandas dos stakeholders.

Elicitação:

Nesta etapa. levanta-se os requisitos de usuário do sistema sob duas perspectivas:

- Categoria do requisito (Funcionais vs não Funcionais);
- Natureza do requisito (Subconscientes vs conscientes vs inconscientes).

Elaboração:

Nesta etapa, detalha-se cada requisito descrito em linguagem natural em modelos conceituais, como UML.

O principal objetivo é eliminar ambiguidades, inconsistências, omissões e erros dos requisitos.

Alguns tipos de diagramas são:

- Diagramas de Caso de Uso;
- Diagramas de Bloco;
- Diagramas Paramétricos;
- Diagramas de Requisitos;

- Diagramas de Sequência;
- Diagramas de Máquina de Estados.

Negociação:

Nesta etapa, o principal objetivo é identificar os conflitos entre os requisitos para negociar as soluções com os stakeholders priorizando, eliminando, combinando ou modificando os requisitos.

Especificação:

Nesta etapa, o sistema é especificado em termos técnicos, ou seja, desenvolve-se os requisitos de sistema que devem atender os requisitos de usuário.

Na especificação, passa-se da perspectiva do problema (requisitos de usuário) para a perspectiva da solução (requisitos de sistema).

Validação:

Nesta etapa, é validada a cobertura do sistema, ou seja, o atendimento de todos os requisitos de usuário pelo sistema proposto. E, é homologado o aceite dos stakeholders sobre os requisitos desenvolvidos.

Gerenciamento:

Esta etapa permeia todo o ciclo de vida do produto e consiste em dois aspectos fundamentais:

- Garantia do escopo do produto;
- Gestão de mudanças.

Podemos concluir que:

- O entendimento dos requisitos de um problema, está entre as tarefas mais difíceis enfrentadas pelos profissionais de desenvolvimento de sistemas.

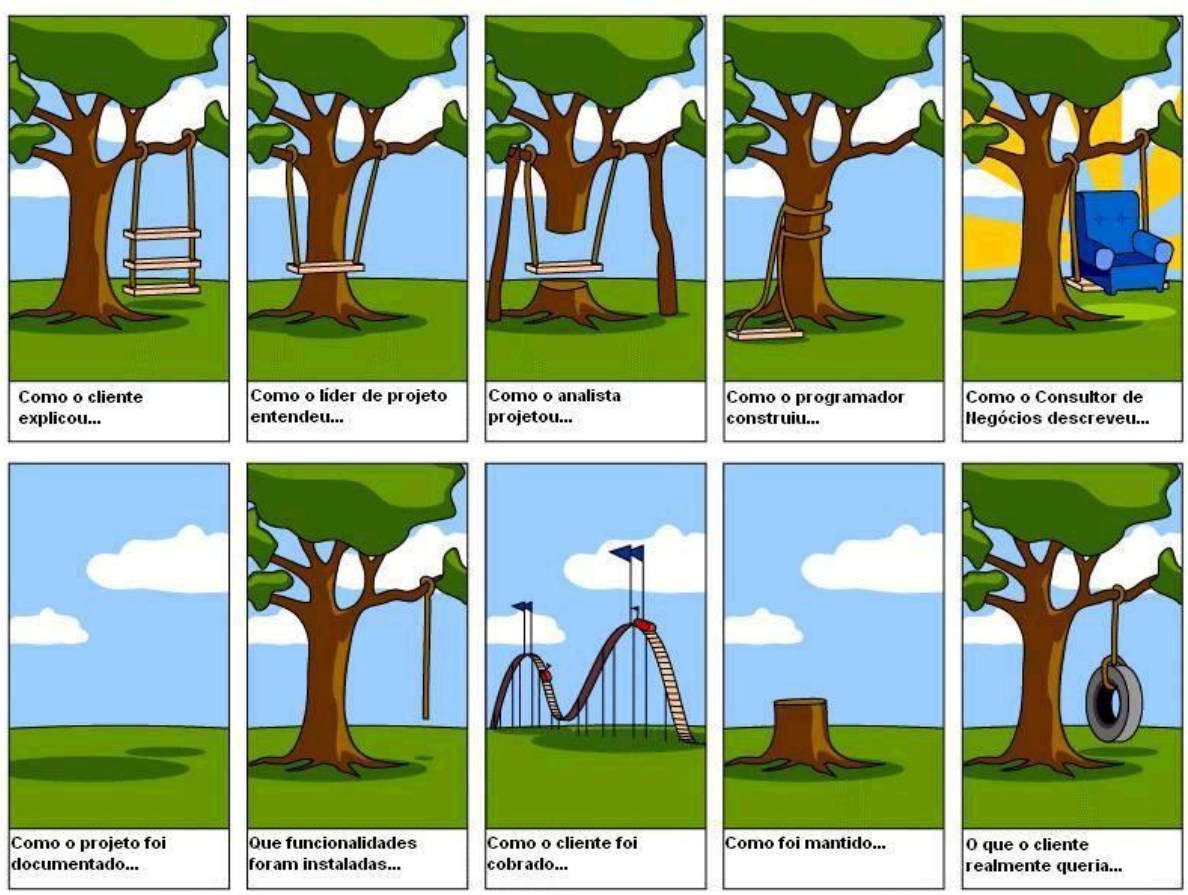
- Isso se deve principalmente pelo fato de o cliente não saber quais são as suas necessidades de forma assertiva.
- Mesmo se os clientes soubessem de tudo isso, provavelmente as necessidades deles mudariam ao longo do projeto.
- A engenharia de requisitos é realizada por analistas de sistemas juntamente com gerentes, clientes, usuários finais e outros que possam ter interesse no software.

Capítulo 3. Levantamento de Requisitos

O Levantamento é uma etapa em que se pergunta ao cliente, usuários e os demais interessados quais são os objetivos de cada um para o sistema, qual será o objetivo do sistema, como o sistema atenderá às necessidades da empresa e como o sistema deverá ser utilizado no dia a dia.

Apesar de parecer simples, esta é uma etapa bastante complicada.

Figura 1 - Levantamento de Requisitos.



Fonte: Project Cartoon.

De acordo com o Chaos Report do Standish Group, o levantamento incorreto ou incompleto de requisitos e a especificação estão entre os maiores fatores de falhas em projetos.

Os autores Christel e Kang identificaram diversos problemas encontrados durante esta etapa, entre eles temos:

Problemas de escopo em que se definem os limites do sistema de forma precária ou clientes e usuários do sistema especificam detalhes técnicos desnecessários que confundem ao invés de esclarecer os objetivos do sistema;

Problemas de Entendimento onde os clientes e usuários não sabem o que precisam, não tem entendimento das capacidades nem das limitações dos seus ambientes computacionais, não sabem transmitir as necessidades aos analistas, omitem informações que consideram óbvias, especificam requisitos que conflitam com as necessidades de outros clientes ou usuários do sistema ou ainda especificam requisitos ambíguos ou impossíveis de serem testados;

Por fim, temos os Problemas de Volatilidade em que os requisitos mudam com o tempo.

Portanto, nesta etapa de levantamento, temos grande parte dos problemas que afetam o software como um todo. Esta etapa deve ser realizada com muita atenção.

Para o levantamento de requisitos, podemos usar diversas técnicas como entrevistas e questionários, workshops de requisitos, cenários, prototipagem, entre outras.

Entrevista: a forma mais utilizada, na qual o analista se reúne com o cliente e coleta os requisitos do sistema por meio de perguntas e observações do cenário apresentado pelo cliente.

Questionário: o analista desenvolve um questionário e envia para o cliente responder. Através das respostas fornecidas, são elaborados os requisitos. É útil quando não é possível realizar uma entrevista pessoalmente com o cliente (embora atualmente isto é resolvido facilmente utilizando Skype, Hangouts, etc.) ou quando existem diferentes usuários em locais distantes, pois pode ser enviado via e-mail.

Pode ser uma boa opção para sistemas simples, porém pode se tornar inviável para sistemas mais complexos com muitos recursos e regras de negócio.

JAD (Joint Application Design): técnica que tem como ponto principal a cooperação de toda a equipe envolvida com a solução a ser criada.

São feitas reuniões com os clientes na qual são definidos os requisitos tendo o ponto de vista de todos os envolvidos, desde o usuário final ou seu representante, quanto analistas, arquitetos, diretores, etc. O ponto principal é que todos os níveis envolvidos com o projeto estejam interagindo com a definição dos requisitos.

Prototipação: a prototipação é mais utilizada como uma técnica de validação de requisitos do que como uma técnica de levantamento de requisitos.

É muito utilizada em cenários onde os requisitos obtidos são muito vagos ou não tão claros. Neste caso, o analista desenvolve um protótipo da solução que ele conseguiu compreender e apresenta ao cliente, e este valida se o protótipo está de acordo com a solução que ele deseja.

Prototipação: A grande vantagem de utilizar protótipos é que o cliente já tem uma visão prévia da solução final, e pode rapidamente validar ou solicitar alguma mudança, permitindo a correção imediata e não durante o desenvolvimento do software.

No levantamento de requisitos, devemos atentar para quatro entendimentos que devemos possuir:

Entendimento do Domínio da Aplicação, onde se entende, de uma maneira geral, a área na qual o sistema será aplicado;

Entendimento do Problema, onde entendemos os detalhes do problema específico a ser resolvido com o auxílio do sistema a ser desenvolvido;

Entendimento do Negócio, onde entendemos como o sistema afetará a organização e como contribuirá para que os objetivos do negócio e os objetivos gerais da organização sejam atingidos;

E, por fim, o Entendimento das Necessidades e das Restrições dos Interessados, onde entende-se as demandas de apoio para a realização do trabalho de cada um dos interessados no sistema, entende-se os processos de trabalho a serem apoiados pelo sistema e o papel de eventuais sistemas existentes na execução e condução dos processos de trabalho.

Trabalhar com levantamento de requisitos é um trabalho investigativo e exigirá de você conhecimento técnico além de habilidades interpessoais, entrevistas e (talvez em menor escala) estudos de documentação.

A principal ferramenta de trabalho de um analista de requisitos é o editor de textos.

O detalhamento e a profundidade com que ele escreve e passa para o papel as necessidades, irá repercutir nas dúvidas na especificação técnica e até mesmo lá na frente no desenvolvimento destes requisitos.

É aqui onde as minúcias devem ser esclarecidas e as perguntas devem ser feitas buscando esgotar todas as possibilidades.

Capítulo 4. Análise de Requisitos

Após a atividade de Levantamento de Requisitos, inicia-se a atividade de Análise de Requisitos, que é onde os requisitos levantados são usados como base para a modelagem do sistema.

Neste processo de análise de requisitos, todas as especificações do cliente são colocadas à mesa, e são estabelecidas as prioridades do projeto.

Isso é, identifica-se os aspectos imprescindíveis do software a ser desenvolvido — nos quais a equipe priorizará seus esforços.

A importância disso se reflete nos benefícios em conhecer mais profundamente as necessidades do cliente.

O maior deles, sem dúvidas, é a compreensão do seu dia a dia, pois isso é o que permitirá à equipe desenvolver uma solução mais condizente com os problemas do cliente, otimizar o uso do tempo e reduzir as possíveis falhas.

Os requisitos são escritos tipicamente em linguagem natural, no entanto, é útil expressarmos requisitos mais detalhados do sistema de maneira mais técnica através de diversos tipos de modelos que podem ser utilizados.

Esses modelos são representações gráficas que descrevem processos de negócio, o problema a ser resolvido e o sistema a ser desenvolvido.

Representações gráficas são muito mais compreensíveis do que descrições detalhadas em linguagem natural, e por isso são utilizadas.

Dessa forma, a análise é uma atividade de modelagem.

Vale ressaltar que essa modelagem é conceitual, pois estamos preocupados com o domínio do problema e não com soluções técnicas.

Portanto, os modelos de análise são elaborados a fim de obtermos uma compreensão maior do sistema a ser desenvolvido e para especificá-lo.

Na análise de requisitos buscam-se principalmente duas perspectivas:

- A estrutural, onde se busca modelar os conceitos, propriedades e relações do domínio que são consideradas relevantes para o sistema em desenvolvimento.
- A comportamental, onde se busca modelar o comportamento geral do sistema, de uma de suas funcionalidades ou de uma entidade.

Os diagramas da UML (Unified Modeling Language) provêm suporte a todos os diagramas necessários nessa fase de análise.

No mercado existem diversos tipos de ferramentas case que auxiliam na construção de diagramas.

Falaremos mais adiante em detalhes sobre a UML e experimentaremos na prática algumas das principais ferramentas de modelagem de requisitos.

Capítulo 5. Engenharia de Requisitos Tradicional vs Ágil

A modelagem, do ponto de vista ágil, é um método eficiente que tem como objetivo tornar mais produtivos os esforços da tarefa de modelar, tão comum nos projetos de software.

Os valores, princípios e práticas da Modelagem Ágil podem auxiliar as equipes na definição de componentes técnicos de alto e baixo nível que farão parte do desenvolvimento de software.

Artefatos sofisticados elaborados por ferramentas de alto custo nem sempre são os melhores para ajudar no desenvolvimento do software. Modelar o software em grupo e com a participação dos usuários, utilizando rascunhos, é vista como uma boa prática para se conseguir isto.

Uma das dificuldades mais comuns nas equipes de desenvolvimento é como conseguir efetivamente capturar as necessidades dos usuários.

Uma das maneiras de se alcançar isso passa pelo desenvolvimento iterativo, onde são entregues software funcionando constantemente com o objetivo de observar se as necessidades foram atendidas.

O processo atual de desenvolvimento de software ainda está em um nível de qualidade muito abaixo do que seria considerado ideal, isto porque os sistemas são entregues aos clientes fora do prazo estipulado no projeto e com um custo maior do que o previsto.

Além disso, esses sistemas frequentemente não alcançam a qualidade desejada pelo cliente e, em muitos casos, não satisfazem as reais necessidades do mesmo, levando a altos índices de manutenção ou passando a ideia de “isso terá de ficar para uma próxima versão”.

A modelagem ágil é vista como um processo de desenvolvimento de software baseado em práticas que visa aumentar a eficiência das equipes dentro dos projetos.

Ao contrário dos processos tradicionais, que requer basicamente os mesmos artefatos para todos os tipos de projetos, a modelagem ágil busca a construção e manutenção eficiente de artefatos, criando-os apenas quando agregarem valor real ao projeto, e focando principalmente os esforços no desenvolvimento do software.

É importante lembrar que a metodologia ágil parte do princípio de que os requisitos do software serão elaborados ao longo do seu desenvolvimento, e não em uma etapa anterior.

Dessa forma, nos primeiros passos do projeto são levantadas apenas as necessidades de alto nível.

Aceitamos como princípio nas metodologias ágeis que os requisitos passarão por evoluções e, inclusive, falharão; são possibilidades inerentes ao desenvolvimento de um produto.

Por isso, a sua elaboração contínua está diretamente integrada ao desenvolvimento como um todo.

Agora, chegamos à questão: como, então, fazer a definição dos requisitos neste cenário tão dinâmico?

- Um consenso na metodologia ágil é que os requisitos não precisam conter toda a informação desde o começo, mas apenas o que for suficiente.

As estórias de usuário vieram originalmente do XP, mas já fazem parte do Scrum e de outros tipos de implementações.

Elas servem para mediar uma comunicação clara entre o cliente/usuário e o time de desenvolvimento — são, acima de tudo, facilitadoras de entendimento.

Por isso, os cartões de User Stories não são alternativas eficientes como documentação de projeto, mas como uma maneira na qual o time desenvolvedor pode compreender melhor o seu público, bem como as suas necessidades, dificuldades e anseios.

Enquanto as User Stories são destinadas a mediar a comunicação entre cliente e desenvolvedor, os Use Cases (ou Casos de Uso) devem ser aplicados durante conversas e reuniões do time.

Eles são maneiras de descrever as interações homem-máquina sistematicamente, permitindo ao desenvolvedor obter uma visão bem estruturada sobre essas interações e a sua atuação em melhorias e correção de erros.

Como serão modelados idealmente em UML e tratam, muitas vezes, com modelos mais complexos, de difícil entendimento — diferentemente das User Stories —, os Use Cases não podem ser feitos pelo usuário.

Boas práticas para definir requisitos:

Tanto os Use Cases quanto as User Stories devem ser trabalhados de forma conjunta, nunca excludente. Afinal, eles geram pontos de vista complementares, e de equivalente importância.

Assim, para fazer o melhor uso deles, é possível seguir um conjunto de melhores práticas para definir os requisitos. Vamos a elas?

- Incentive a participação dos stakeholders

Existe uma correlação forte entre o envolvimento dos stakeholders e do sucesso do projeto. E o uso de ferramentas simples, como post-its e quadros brancos, já ajuda a tornar o processo mais acessível.

- Descreva os requisitos de maneira clara

Os requisitos devem ser descritos de forma clara, sem o uso de termos vagos, como “econômico” ou “agradável”. Não abra margem para dupla interpretação, pois isso abrirá portas para ocorrência de falhas e atrasos na entrega do produto.

- Evite exageros

Para validar cada requisito, deve ser descrito e documentado apenas o necessário, por meio de testes.

Isso porque a natureza de mudança das metodologias ágeis tornaria o ajuste de documentações extensas um verdadeiro pesadelo, além de afetar a clareza das descrições de requisitos que mencionamos acima.

- Foco no cliente

Você não executa os projetos apenas por eles mesmos. Seu objetivo é gerar valor para o cliente — e as suas definições de requisito, assim com o gerenciamento delas, precisa refletir isso.

Ao conhecer melhor o dia a dia do cliente, observando de perto os conflitos e necessidades dos usuários finais, o time de desenvolvimento tem a oportunidade de entregar um sistema muito mais completo e que, de fato, contribui para os seus processos de negócio.

Capítulo 6. Documentação de Requisitos em Projetos Ágeis

Os requisitos e modelos capturados nas etapas de Levantamento de Requisitos e Análise de Requisitos devem ser descritos e apresentados em documentos.

A documentação é uma atividade de registro e oficialização dos resultados da engenharia de requisitos.

Como resultado, um ou mais documentos devem ser produzidos.

Essa documentação, escrita de uma boa forma, apresenta diversos benefícios, como facilidade na comunicação dos requisitos, redução no esforço de desenvolvimento, fornece uma base realista para estimativas, boa base para verificação e validação, entre outros benefícios.

A documentação produzida também possui diversos interessados que usam a documentação para diferentes propósitos.

Os Clientes, Usuários e Especialistas de Domínio atuam na especificação, avaliação e alteração dos requisitos.

Gerentes utilizam a documentação para planejar uma proposta para o sistema e para planejar e acompanhar o processo de desenvolvimento.

Os desenvolvedores utilizam a documentação para compreender o sistema e a relação entre as suas partes.

Os testadores utilizam a documentação para projetar casos de teste.

A documentação deve ser feita ao longo de todo o desenvolvimento do software. Sua construção deve:

- Comunicar a estrutura e o comportamento do sistema;
- Visualizar e controlar a arquitetura do sistema;

- Expor oportunidades de simplificação e reaproveitamento;
- Compor o estudo de gerenciamento de riscos.

Na documentação tradicional, o Documento de Requisitos deve conter uma descrição do propósito do sistema, uma breve descrição do domínio do problema e listas de requisitos funcionais, não funcionais e regras de negócio, tudo descrito em linguagem natural.

Desenvolvedores, clientes, usuários e gerentes utilizam esse documento.

Outro documento que pode ser produzido é o Documento de Especificação de Requisitos que deve conter os requisitos escritos a partir da perspectiva do desenvolvedor, contendo inclusive uma correspondência direta com os requisitos no Documento de Requisitos.

O manifesto ágil cita que “software funcional é mais importante do que documentação extensa”.

A partir desta declaração, foi criado um mito na comunidade de TI de que projeto ágil não precisa de documentação.

A abordagem tradicional gerou um volume excessivo de documentos complexos de difícil entendimento e acesso, resultando na aversão à documentação de projetos.

Em contrapartida, a metodologia de desenvolvimento ágil defende a adaptação de melhores práticas no desenvolvimento de projetos através de documentos e bibliotecas eficientes e funcionais, comunicação e compartilhamento, colaboração e participação.

A aplicação de práticas de documentação ágil permite alcançar artefatos claros e objetivos, acompanhando a transformação e evolução de tecnologias e plataformas tecnológicas.

Uma mudança que a abordagem ágil trouxe em relação à forma de pensar a documentação, foi parar de gerar documentos simplesmente por gerá-los, por estarem previstos em algum processo.

Ao invés de documentar tudo como padrão, passamos a questionar o valor real de cada documento. Se um documento é necessário para cumprir um requisito regulatório importante, isso é um tipo de valor.

Se o produto tem regras de negócio complexas que o cliente acha importante ter por escrito para referência futura ou para divulgar aos usuários, isso é um tipo de valor.

Se o desenvolvimento é feito para um cliente que exige contratualmente a geração de documentos A, B e C (algo muito comum em licitações para o governo), isso também é valor.

Quando esse valor superar o custo de documentar, podemos e devemos fazê-lo.

Mas é bom lembrar que o custo aqui envolve tanto o custo de criar o documento como o de mantê-lo atualizado depois.

E aqui, também, os métodos ágeis trazem uma diferença em relação às abordagens prescritivas tradicionais: o momento em que a documentação é gerada é escolhido de forma a reduzir o custo de manutenção.

Nas abordagens tradicionais, é comum documentar para definir o que será desenvolvido antes de desenvolver, como especificações de requisito, casos de uso detalhados, modelos de projeto.

Esses documentos precisam ser mantidos atualizados até o fim do projeto, quando são então entregues como artefatos do produto.

Notem que, nessa abordagem, estamos documentando as coisas enquanto elas ainda estão sendo definidas.

E esse pode ser o pior momento para documentá-las, porque ainda estão muito instáveis. Isso significa que, mesmo que o custo para criar a documentação seja baixo, o custo para mantê-la atualizada pode ficar muito alto.

Na abordagem ágil, como reduzimos a necessidade de documentar antecipadamente para comunicar o que precisa ser desenvolvido, podemos concentrar a documentação naquilo que tenha real valor e deixar para fazer isso no momento mais oportuno — que pode ser durante ou mesmo após o desenvolvimento daquela funcionalidade, quando as coisas já estão mais estáveis.

Reduzimos, assim, o custo de manter a documentação atualizada.

Na prática, o momento de tratar cada tipo de documentação num projeto ágil pode ser materializado de duas maneiras:

1) **Incremental**: Para a parte da documentação que for importante gerar durante o desenvolvimento, de maneira incremental, isso pode ser feito adicionando um critério na definição de feito do projeto.

Assim, garantimos que nenhum item será considerado aceito se a documentação não tiver sido produzida.

2) **Como item de backlog**: Para a parte da documentação em que for mais interessante gerar de uma única vez para o produto, podemos tratá-la como itens de backlog.

Assim, o Product Owner analisará qual a prioridade desse item em relação aos demais itens de backlog do produto, e decidirá o melhor momento para passar esse item ao time para desenvolvimento.

Seja incremental ou como item de backlog, notem que estamos tratando a documentação como um resultado do desenvolvimento (documentamos o que foi feito), e não como um insumo (documentar para discutir e transmitir o que será feito).

Capítulo 7. Gerência de Requisitos no Dev. Ágil de Software

Mudanças nos requisitos ocorrem durante todo o processo de software, desde o levantamento de requisitos até durante a operação do sistema em produção.

Isso ocorre devido à descoberta de erros, omissões, conflitos, inconsistência nos requisitos, melhor entendimento dos usuários sobre as suas necessidades, problemas técnicos, mudanças de prioridades do cliente, mudanças no negócio, concorrentes, mudanças econômicas, mudanças no ambiente de software, mudanças organizacionais, etc.

Para minimizar os problemas causados por essas mudanças, é necessário gerenciar requisitos.

O processo de Gerencia de Requisitos envolve atividades que ajudam a equipe a identificar, controlar, rastrear requisitos e gerenciar mudanças de requisitos em qualquer momento ao longo do ciclo de vida do software.

Em outras palavras, é o processo que gerencia mudanças nos requisitos de um sistema.

Estas mudanças ocorrem conforme os clientes desenvolvem um melhor entendimento de suas reais necessidades.

As razões para estas constantes mudanças podem ser originadas de vários fatores tais como:

- Nem sempre os requisitos são óbvios e podem vir de várias fontes.
- Nem sempre é fácil expressar os requisitos claramente em palavras.
- Existem diversos tipos de requisitos em diferentes níveis de detalhe.
- O número de requisitos poderá impossibilitar a gerência, se não for controlado.

As razões para estas constantes mudanças podem ser originadas de vários fatores tais como:

- Os requisitos estão relacionados uns com os outros, e também com o produto liberado do processo de engenharia do software.
- Os requisitos têm propriedades exclusivas ou valores de propriedade. Por exemplo, eles não são igualmente importantes nem igualmente fáceis de cumprir.
- Há várias partes interessadas, o que significa que os requisitos precisam ser gerenciados por grupos de pessoas de diferentes funções.
- Os requisitos são alterados.

Pode-se definir que o gerenciamento de requisitos trata-se de um modelo sistemático para:

I) Identificar, organizar e documentar os requisitos do sistema; e

II) Estabelecer e manter acordo entre o cliente e a equipe do projeto nos requisitos variáveis do sistema.

Portanto, os objetivos do processo são:

I) Gerenciar alterações nos requisitos acordados;

II) Gerenciar relacionamentos entre requisitos;

III) Gerenciar dependências entre requisitos e outros documentos produzidos durante o processo de software.

Dessa forma, a gerência de requisitos possui as seguintes atividades:

I) Controle de mudanças;

II) Controle de versão;

III) Acompanhamento do estado dos requisitos; e

IV) Rastreamento de requisitos.

A definição de um processo apropriado para uma organização é muito importante e traz diversos benefícios, pois uma boa descrição de um processo fornece orientações e reduz a probabilidade de erros ou esquecimentos.

O mais importante é saber que não existe um processo ideal, portanto adaptar um processo para as necessidades internas é sempre a melhor escolha ao invés de impor um processo à organização.

Para agravar a situação, os sistemas de modo geral também devem levar em conta que o mundo está em constante mudança – de modo que algumas das hipóteses levantadas nas fases iniciais podem se tornar equivocadas.

Boas práticas de gerenciamento de requisitos, como uma manutenção de dependências entre requisitos, têm benefícios em longo prazo, como maior satisfação do cliente e custos de desenvolvimento mais baixos.

Uma vez que os retornos não são imediatos, o gerenciamento de requisitos pode parecer uma despesa desnecessária. Entretanto, sem a gerência, a economia de curto prazo será devastada pelos custos em longo prazo.

Logo, todo sistema deve ser desenvolvido de modo que as alterações sofridas ao longo do seu desenvolvimento sejam menos impactantes o possível.

O processo de mudança dos requisitos precisa ser controlado de modo a garantir a qualidade do sistema.

O impacto destas mudanças precisa ser avaliado e compreendido de modo que a sua implementação seja feita de maneira eficiente e a baixo custo.

Então, é de fundamental importância que as alterações dos requisitos sejam:

- Identificadas e avaliadas;
- Avaliadas sob o ponto de vista de risco;
- Documentadas;

- Planejadas;
- Comunicas aos grupos e indivíduos envolvidos; e
- Acompanhadas até a finalização.

Todos os artefatos (documentos) produzidos durante o desenvolvimento do software devem tornar a gerência dos requisitos visível e transparente.

Estes documentos devem ser gerados levando-se em conta padrões externos e corporativos, de modo a assegurar consistência e uniformidade das informações.

Políticas bem definidas para a gerência de configuração, controle de mudanças, rastreabilidade e garantia da qualidade precisam ser colocadas em prática de modo a viabilizar um processo dinâmico, ágil e eficaz de gerência de requisitos.

Portanto, para se ter uma gerência de requisitos eficaz é necessário, de antemão, possuir um conjunto de políticas. É necessário definir um conjunto de objetivos para o processo de gerência.

Capítulo 8. Gestão de Riscos na Documentação Ágil de Software

Os métodos ágeis visam aumentar a interação entre clientes e fornecedores de software e antecipar riscos, que futuramente se transformarão em bugs e custos adicionais ao projeto.

Com isso, a ideia de unir as metodologias ágeis às práticas de gestão de risco tradicionais do PMBOK tem como intuito melhorar a análise de riscos em projetos ágeis e com isso mitigar, prever e antecipar problemas e defeitos que ficam mais caros numa escala de 10X a cada fase do processo de desenvolvimento em que o mesmo é corrigido.

A proposta de junção do Scrum ou outra metodologia ágil com a gestão de riscos, segundo o PMBOK, é de criar uma análise formal e a possibilidade de poder classificar, quantificar e qualificar os riscos positivos e negativos de um projeto ágil, a fim de tratá-los, seja apenas entendendo e aceitando os possíveis prejuízos ou mitigando/sanando o mesmo, como cita o PMBOK em seu capítulo sobre gerenciamento de riscos.

Os riscos estão presentes de forma permanente em termos de incerteza (risco que pode ou não ocorrer) e perda (se o risco se tornar uma realidade, perdas indesejáveis ocorreram).

Sendo assim, têm-se definidas as categorias acerca dos riscos:

I) Riscos de projetos, que ameaçam o plano de execução do projeto;

II) Riscos técnicos, que ameaçam a qualidade e o cronograma do projeto;

III) Riscos conhecidos, aqueles descobertos em avaliações;

IV) Riscos previsíveis, conhecidos a partir da experiência; e

V) Riscos imprevisíveis, que podem ocorrer, mas é muito difícil identificar previamente.

Ao diminuir o tempo de liberação do produto ao mercado e buscar ativamente o frequente feedback do usuário, as equipes ágeis reduzem o risco, atingem rapidamente a entrega de valor inicial e aumentam o retorno geral do investimento.

O princípio ágil de "Abraçar a mudança" cria a oportunidade às equipes em estabelecer uma atitude de constante abertura a receber mudanças durante o projeto, minimizando a necessidade de "tentarmos" esgotar todos os detalhes dos requisitos da demanda e naturalmente assumindo riscos (que naturalmente são compartilhados).

De acordo com Alan Moran, autor do livro *Agile Risk Management*, esta atitude cria um ambiente adequado para considerar o risco como parte do processo de execução da solução e, com isto, estabelecer a abertura para "abraçar o risco".

O autor destaca que a integração das técnicas de gerenciamento de riscos em projetos ágeis exige cuidado, considerando o perfil heterogêneo da equipe, a eficiência dos loops de feedback a cada interação e um processo decisório adequado e facilitado, normalmente baseado em Lean. Mas reforça que os princípios ágeis contribuem e alguns artefatos podem ser reusados para abordar as práticas de gerenciamento de riscos.

O modelo proposto sugere que a Identificação de Riscos e sua Avaliação podem ser incorporadas na cerimônia de Planejamento da Sprint, e os participantes, ao identificar as Estórias de Uso e compor o Kanban, podem identificar e classificar as estórias de acordo com o grau de risco avaliado.

Além disso, um segundo Kanban pode ser criado apenas para relatar a identificação e avaliação dos riscos mapeados.

Finalmente, o monitoramento dos riscos é realizado nas reuniões diárias do Scrum, na Revisão da Sprint e na Retrospectiva.

Havendo uma identificação nova, mitigação efetiva ou mudança no mapa de risco, o quadro Kanban de Risco deverá ser atualizado e visitado na próxima sessão de Planejamento da próxima Sprint.

Existe diferença entre riscos e incertezas.

Sobre riscos temos algumas informações, na incerteza muito pouco sabemos. Com este raciocínio, faz-se relevante definir outro termo muito utilizado no mundo dos projetos de TI, a **issue**.

Esta palavra remete as certezas já disparadas por um evento dentro de um projeto e que já se tornou um problema a ser resolvido, devendo ser tratado pela equipe.

Genericamente falando, tanto na ótica tradicional do gerenciamento de riscos como nas novas propostas em função da agilidade, o **apetite**, **tolerância** e **limites dos riscos** são considerados.

Definir a quantidade de incerteza que os stakeholders estão dispostos a assumir reflete o **apetite**.

Indicar o grau de risco que os stakeholders resistirão relaciona-se com sua **tolerância**.

Os **limites dos riscos** referem-se ao nível em que o risco é aceitável para a organização.

Se o risco estiver abaixo do limite, os stakeholders têm maior probabilidade de aceitar. Assim, a comunicação com os stakeholders perante os riscos é fundamental para as duas perspectivas, cascata e ágil.

Hoje encontramos muitos executivos ou gerentes que apenas registram a existência de um risco, cobrando a evolução ou a solução desses riscos de forma mágica, sem aplicar uma gestão eficiente sobre eles.

Capítulo 9. Requisitos Funcionais

Existem dois tipos de classificação de requisitos, são eles: Requisitos Funcionais (RF) e Requisitos Não-Funcionais (RNF).

Esclarecido o que são requisitos é hora de desmembrá-los explicando cada um, começamos pelos requisitos funcionais.

Os requisitos funcionais descrevem a funcionalidade ou os serviços que se espera que o sistema realize em benefício dos usuários (PAULA FILHO, 2000). Eles variam de acordo com o tipo de software em desenvolvimento, com usuários e com o tipo de sistema que está sendo desenvolvido.

Requisitos funcionais podem ser expressos de diversas maneiras e, como já foi dito, em diferentes níveis de detalhamento.

Os requisitos funcionais de usuários definem recursos específicos que devem ser fornecidos pelo sistema (SOMMERVILLE, 2008). São todas as necessidades, características ou funcionalidades esperadas em um processo que podem ser atendidos pelo software.

De forma geral, um requisito funcional expressa uma ação que deve ser realizada através do sistema, ou seja, um requisito funcional é “o que o sistema deve fazer”. Eles referem-se sobre o que o sistema deve fazer, ou seja, suas funções e informações.

Os requisitos não funcionais referem-se aos critérios que qualificam os requisitos funcionais.

- Esses critérios podem ser de qualidade para o software, ou seja, os requisitos de performance, usabilidade, confiabilidade, robustez, etc. Ou então, os critérios podem ser quanto a qualidade para o processo de software, ou seja, requisitos de entrega, implementação, etc.

Exemplos de requisitos funcionais:

- [RF001] O Sistema deve cadastrar clientes (entrada).
- [RF002] O Sistema deve emitir um relatório de clientes (saída).
- [RF003] O Sistema deve passar um cliente da situação "em atendimento" para “atendido" quando o cliente terminar de ser atendido (mudança de estado).
- [RF004] O cliente pode consultar seus dados no sistema.

Outros exemplos de requisitos funcionais:

- Emissão de nota fiscal;
- Consulta ao estoque;
- Geração de pedido;
- Emissão de relatório;
- Lançamento de notas de alunos.

Inicialmente, a especificação de requisitos funcionais deve ser completa – deve definir todos os requisitos de usuário e refletir as decisões de especificação tomadas – e consistente – os requisitos não devem ter definições contraditórias (PAULA FILHO, 2000).

Entretanto, na realidade, em sistemas complexos e grandes, é quase impossível atingir a consistência e a completeza dos requisitos. Isso ocorre, principalmente, em função da complexidade inerente ao sistema e porque as pessoas possuem diferentes pontos de vistas em relação a um problema. Esses problemas somente emergem após uma análise mais aprofundada. À medida que os problemas vão sendo descobertos, deve se ir atualizando o documento de requisitos (SOMMERVILLE, 2008).

Mas afinal, por que eles são chamados de requisitos funcionais?

A categorização dos requisitos citados como requisitos funcionais se deve ao fato de que todos eles são funcionalidades atendidas através de uma ação do software ou comportamento específico do sistema.

Podemos dizer que é considerado um requisito funcional, todo cenário onde o usuário informa um dado, ou um sistema terceiro realiza uma solicitação qualquer durante uma interação com o sistema, que então, responde com determinada ação correspondente.

Os requisitos funcionais são de extrema importância no desenvolvimento de sistemas, pois, sem eles não há funcionalidades nos sistemas.

Seus modelos devem ser construídos em um nível de entendimento claro e objetivo, além de um código fonte totalmente aplicável.

Capítulo 10. Classificação de Requisitos Funcionais

Os Requisitos Funcionais podem ser:

I) Permanentes e transitórios;

II) Evidentes e ocultos;

III) Obrigatórios e desejados.

Permanentes e transitórios:

- O requisito pode ou não mudar? Decisão do analista.
- Decisão relativa a tempo e custo de desenvolvimento:
  - Permanente: mais fácil e rápido implementar, mais caro de manter se o sistema mudar.
  - Transitório: mais caro e complexo desenvolver, mais barato e rápido de manter.
- Requisitos transitórios devem ser suscetíveis a acomodar mudanças.
- Requisitos mais importantes devem ser transitórios.
  - Espera-se que possam mudar no futuro e essa mudança causa maior impacto no sistema.

Evidentes e ocultos:

- Evidentes: funções efetuadas com o conhecimento do usuário.
  - Consultas, entrada de dados, etc.
- Ocultos: cálculos ou atualizações feitas pelo sistema sem a solicitação explícita do usuário.
  - Como consequência de outras funções efetuadas por ele.

- Apenas requisitos evidentes corresponderão aos passos do caso de uso expandido.
  - Requisitos ocultos são associados a eles para serem lembrados no momento de projetar as operações de caso de uso.
- Exemplos:
  - Evidente: Emitir relatório de livros;
  - Oculto: Aplicar uma política de desconto.

Obrigatórios e desejados:

- Indicam uma prioridade no desenvolvimento.
  - Quanto a importância e tempo de desenvolvimento.
- Obrigatórios devem ser feitos de qualquer forma.
- Aplica-se também a requisitos nâo-funcionais.
  - Quais restrições se deseja acomodar e quais são apenas desejáveis, se sobrar tempo e recurso.
- Exemplos:
  - A interface Web deve ser obrigatória.
  - Acesso através de celular deve ser desejável.

Temos ainda os **requisitos de domínio**.

- Tratam-se de requisitos derivados do domínio da aplicação do sistema.
- Ao invés de serem obtidos a partir das necessidades específicas dos usuários do sistema, eles podem se transformar em novos requisitos funcionais, ou serem regras de negócios específicas do domínio do problema.

- Exemplos:
  - Utilização de uma interface padrão utilizando a norma XPTS.
  - Disponibilização de arquivos somente para leitura, devido aos direitos autorais.
  - O cálculo da média final de cada aluno é dado pela fórmula: (Nota 1 * 2 + Nota 2 * 3) / 5.
  - Um aluno pode se matricular em uma disciplina desde que ele tenha sido aprovado nas disciplinas consideradas pré-requisitos.

Requisito Funcional vs Regra de Negócio

- Requisito Funcional:
  - Absorver ar pelas vias aéreas a partir da inalação pelo nariz.
- Regra de Negócio:
  - Viabilizar um engasgo quando houver bloqueio das vias áreas por entupimento.
- Requisito Funcional:
  - Digerir os alimentos inseridos através da boca e transportados pelo tubo digestivo.
- Regra de Negócio:
  - Expelir alimentos pelo tubo digestivo quando houver o preenchimento de 100% do espaço do estômago.

Capítulo 11. Requisitos Não Funcionais

Requisitos não funcionais são aqueles que não estão diretamente relacionados à funcionalidade de um sistema.

O termo requisitos não funcionais é também chamado de atributos de qualidade.

Os requisitos não funcionais têm um papel de suma importância durante o desenvolvimento de um sistema, podendo ser usados como critérios de seleção na escolha de alternativas de projeto, estilo arquitetural e forma de implementação.

Desconsiderar ou não considerar adequadamente tais requisitos é dispendioso, pois torna difícil a correção, uma vez que o sistema tenha sido implementado.

Vejamos um exemplo. Suponha que uma decisão tenha sido feita de modularizar a arquitetura de um sistema de modo a facilitar sua manutenção e adição de novas funcionalidades. Entretanto, modularizar um sistema adicionando uma camada a mais pode comprometer um outro requisito, o de desempenho. Portanto, faz-se necessário definir logo cedo quais requisitos não funcionais serão priorizados na definição de uma arquitetura.

A arquitetura de software deveria oferecer suporte a tais requisitos. Isto resulta da associação existente entre arquitetura de software e requisitos não funcionais. Importante observar que cada estilo arquitetural (isto é, a forma na qual o código do sistema é organizado) suporta requisitos não funcionais específicos.

A estruturação de um sistema é determinante no suporte oferecido a um requisito não funcional. Por exemplo, o uso de camadas permite separar melhor as funcionalidades de um sistema, tornando-o mais modular e facilitando sua manutenção.

Na análise de arquiteturas candidatas para um sistema de software, um arquiteto ou engenheiro de software considera os requisitos não funcionais como um dos principais critérios para sua análise.

Exemplos de RNF:

- Utilização do módulo de Informações Cadastrais em modo off-line.
- O sistema deve ser implementado na linguagem Java.
- O sistema deverá se comunicar com o banco SQL Server.
- Um relatório de supervisão deverá ser fornecido toda sexta-feira.
- O sistema deve ser executável em qualquer plataforma.

Duas perguntas que todo analista deve fazer sobre cada requisito não funcional:

A característica é sobre O QUE (funcionais) o sistema fará, ou COMO (não-funcionais) o sistema fará?

É possível verificar se o requisito não funcional está sendo atendido ou respeitado?

Esse tipo de requisito trata das limitações nos serviços ou funções oferecidas pelo sistema baseando em parâmetros como:

- “número de transações por segundo”,
- “número de usuários operando o sistema ao mesmo tempo”; ou
- limitações impostas por padronizações, por exemplo.

Requisitos de um sistema de software são interligados e um requisito pode gerar limitações, influenciar ou mesmo acabar gerando outros requisitos.

Por isso, é muito importante levantar, analisar e priorizar os requisitos não funcionais e, assim, mapear adequadamente o impacto de um requisito nos demais.

Uma dica importante é que os requisitos não funcionais são geralmente mensuráveis e assim devemos preferencialmente associar uma medida ou referência para cada requisito não funcional.

Capítulo 12. Classificação de Requisitos Não Funcionais

Os requisitos não funcionais ainda são classificados em três tipos, são eles: Requisitos do Produto Final, Requisitos Organizacionais e Requisitos Externos.

Requisitos do Produto Final referem-se a como o produto deve comportar-se, ou seja, a sua velocidade de execução, confiabilidade, etc.

Requisitos Organizacionais referem-se à consequência de políticas e procedimentos organizacionais que devem ser seguidos.

Requisitos Externos referem-se a fatores externos ao sistema e ao processo de desenvolvimento como a legislação.

Exemplo de Requisito do Produto Final: A interface do usuário deve ser implementada como simples HTML.

Exemplo de Requisito Organizacional: Todos os documentos entregues devem seguir o padrão de relatórios XYZ-00.

Exemplo de Requisito Externo: Informações pessoais dos usuários não podem ser vistas pelos operadores do sistema.

Requisitos do produto: **Usabilidade**:

Usabilidade é um dos atributos de qualidade ou requisitos não funcionais de qualquer sistema interativo, ou seja, no qual ocorre interação entre o sistema e seres humanos.

A noção de usabilidade vem do fato que qualquer sistema projetado, para ser utilizado pelas pessoas, deveria ser fácil de aprender e fácil de usar, tornando assim fácil e agradável a realização de qualquer tarefa.

Requisitos de usabilidade especificam tanto o nível de desempenho quanto a satisfação do usuário no uso do sistema.

Dessa forma, a usabilidade pode ser expressa em termos de:

- Facilidade de aprender: Associado ao tempo e esforço mínimo exigido para alcançar um determinado nível de desempenho no uso do sistema.
- Facilidade de uso: Relacionado à velocidade de execução de tarefas e à redução de erros no uso do sistema.

Requisitos do produto: **Manutenibilidade**:

- De um modo geral, a manutenibilidade é um dos requisitos mais relacionados com a arquitetura de um sistema de software.
- A facilidade de fazer alteração no sistema existente, seja adicionando ou modificando alguma funcionalidade, depende muito da arquitetura do mesmo.
- Adição de novos componentes deve ocorrer sem a necessidade de modificar a arquitetura existente e ainda comprometer pouco (ou nada) o desempenho atual do sistema.
- Tal suporte à manutenibilidade (seja para correção ou evolução do sistema) deve ser facilmente acomodada pela arquitetura de software.

Requisitos do produto: **Confiabilidade**:

- Confiabilidade de software é a probabilidade de o software não causar uma falha num sistema durante um determinado período de tempo sob condições especificadas.
- A probabilidade é uma função da existência de defeitos no software. Assim, os estímulos recebidos por um sistema determinam a existência ou não de algum defeito.
- Em outras palavras, a confiabilidade de software, geralmente definida em termos de comportamento estatístico, é a probabilidade de que o software irá operar como desejado num intervalo de tempo conhecido.
- Também, a confiabilidade caracteriza-se um atributo de qualidade de software o qual implica que um sistema executará suas funções como esperado.

Exemplos de métricas utilizadas para avaliar a confiabilidade de software, compreendem:

- Disponibilidade;
- Taxa de ocorrência de falha;
- Probabilidade de falha durante fase operacional;
- Tempo médio até a ocorrência de falha.

Requisitos do produto: **Desempenho**:

- Desempenho é um atributo de qualidade importante para sistemas de software.
- Considere, por exemplo, um sistema de uma administradora de cartões de crédito. Em tal sistema, um projetista ou engenheiro de software poderia considerar os requisitos de desempenho para obter uma resposta de tempo para autorização de compras por cartão.
- Adicionalmente, desempenho é importante porque afeta a usabilidade de um sistema.
- Se um sistema de software é lento, ele certamente reduz a produtividade de seus usuários ao ponto de não atender às suas necessidades.

O requisito de desempenho restringe a velocidade de operação de um sistema de software.

Isto pode ser visto em termos de:

- Requisitos de resposta;
- Requisitos de processamento;
- Requisitos de temporização;
- Requisitos de espaço.

Requisitos do produto: **Portabilidade**:

- Portabilidade pode ser definida como a facilidade na qual o software pode ser transferido de um sistema computacional ou ambiente para outro.
- De um modo geral, a portabilidade refere-se à habilidade de executar um sistema em diferentes plataformas. É importante observar que à medida que aumenta a razão de custos entre software e hardware, a portabilidade torna-se cada vez mais importante.

Requisitos do produto: **Reusabilidade**:

- Uma característica das engenharias é fazer uso de projetos existentes a fim de reutilizar componentes já desenvolvidos, objetivando minimizar o esforço em novos projetos.
- O reuso pode ser visto sob diferentes perspectivas. Ele pode ser orientado a componentes, orientado a processos ou orientado ao conhecimento específico de um domínio.

Exemplos de reuso de componentes:

  - **Aplicação**: Toda a aplicação poderia ser reutilizada.
  - **Subsistemas**: Os principais subsistemas de uma aplicação poderiam ser reutilizados.
  - **Objetos ou módulos**: Componentes de um sistema, englobando um conjunto de funções, podem ser reutilizados.
  - **Funções**: Componentes de software que implementam uma única função (como uma função matemática) podem ser reutilizados.

Requisitos do produto: **Segurança**:

Em um sistema de software, este requisito não funcional caracteriza a segurança de que acessos não autorizados ao sistema e dados associados não serão permitidos.

Portanto, é assegurada a integridade do sistema quanto a ataques intencionais ou acidentes. Dessa forma, a segurança é vista como a probabilidade de que a ameaça de algum tipo será repelida.

Exemplos de requisitos de segurança são:

- Apenas pessoas que tenham sido autenticadas por um componente de controle acesso e autenticação poderão visualizar informações.
- As permissões de acesso ao sistema podem ser alteradas apenas pelo administrador de sistemas.
- Deve ser feito cópias (backup) de todos os dados do sistema a cada 24 horas e estas cópias devem ser guardadas em um local seguro, sendo preferencialmente num local diferente de onde se encontra o sistema.
- Todas as comunicações externas entre o servidor de dados do sistema e clientes devem ser criptografadas.

Capítulo 13. Modelo de Classificação FURPS+

Para ajudar os analistas a identificar o real propósito das informações obtidas junto aos usuários, existe um sistema de classificação de requisitos chamado FURPS+.

FURPS+ são as iniciais dos nomes abaixo, onde cada um tem um propósito:

- **Functionality** (Funcionalidade): É todo o aspecto funcional do sistema sendo desenvolvido.
- **Usability** (Usabilidade): Indica o tempo de treinamento para um usuário se tornar produtivo, Tempo de duração desejado para determinada operação no sistema e Ajuda on-line, documentação do usuário e material de treinamento.
- **Reliability** (Confiabilidade): Refere-se à Disponibilidade, Tempo de correção – tempo permitido para indisponibilidade quando ocorre uma falha, Precisão, Número máximo de defeitos (bugs/KLOC – mil linhas de código), Categorias de bugs – bugs devem ser categorizados por nível de impacto.
- **Performance** (Performace ou Desempenho): Indica o Tempo de resposta para uma transação, Troughput (ex.: transações por segundos), Capacidade (ex.: transações concorrentes), Operação Parcial (Situação do sistema aceitável quando estiver prejudicado de alguma forma), Uso de recursos: memória, espaço em disco, comunicação, etc.
- **Supportability** (Suportabilidade): Indica o Padrão de codificação, Convenção de nomenclatura, Bibliotecas de classes, Utilitários de manutenção.
- **Plus (+)**: Indica outros como Design, implementação, interface, físicos, etc.

O “+” do acrônimo engloba outros requisitos não-funcionais que devem ser lembrados:

- **Requisitos de design** (desenho) – Um requisito de design, frequentemente chamado de uma restrição de design, especifica ou restringe o design de um

sistema. Exemplos podem incluir: linguagens de programação, processo de software, uso de ferramentas de desenvolvimento, biblioteca de classes, etc.

- **Requisitos de implementação** – Um requisito de implementação especifica ou restringe o código ou a construção de um sistema.
- Como exemplos, podemos citar:
  - Padrões obrigatórios;
  - Linguagens de implementação;
  - Políticas de integridade de banco de dados;
  - Limites de recursos;
  - Ambientes operacionais.
- **Requisitos de interface** – Especifica ou restringe as funcionalidades inerentes a interface do sistema com usuário.
- **Requisitos físicos** – Especifica uma limitação física pelo hardware utilizado, por exemplo: material, forma, tamanho ou peso. Podendo representar requisitos de hardware, como as configurações físicas de rede obrigatórias.

FURPS+ **Usabilidade**:

- A leitura do código de barras deve ser automática para diminuir a digitação de dados, agilizando a transação e evitando erros.
- Documentação quanto ao uso e funcionamento deve ser fornecido por meio de ajuda on-line aos agentes arrecadadores.

FURPS+ **Confiabilidade**:

- O Word recupera documento não salvo após o desligamento inesperado do computador.

- Se o processo de importação de contas arrecadadas for interrompido por uma falha, o sistema deve permitir recuperar os dados já importados e continuar a partir do ponto em que ocorreu a falha.

FURPS+ **Desempenho**:

- O sistema deve suportar até 100 usuários simultâneos sem se degradar.
- O tempo de resposta de qualquer tela do sistema não deve exceder 2 segundos.

FURPS+ **Suportabilidade**:

- A instalação do software deve ser realizada de forma semiautomática, ou seja, com o mínimo de intervenção humana.
- A solução deve oferecer suporte aos idiomas: inglês, português e espanhol.

Capítulo 14. Metas SMART

S.M.A.R.T representa uma sigla, em inglês, formada pelas iniciais dos principais pontos que devem ser utilizados na hora de planejar uma meta: Específico, Mensurável, Alcançável, Relevante e Baseado no tempo.

Portanto, é uma ferramenta utilizada para validar qualquer tipo meta.

A criação de metas é uma forma de estimular os envolvidos em prol de um objetivo. Para as empresas, em um cenário em que o meio digital permite a criação de novos negócios a cada dia, a concorrência cresce bastante. É preciso buscar, constantemente, novos propósitos, sempre com foco em melhorias contínuas.

As metas servem como um direcionamento. Você traça um lugar que quer chegar ou algo que deseja conquistar. Então, analisa e considera ações que o levarão até o resultado esperado.

Vários caminhos são possíveis, cada um com as suas dificuldades e as suas oportunidades. Deve-se, então, analisar as melhores práticas e comparar os efeitos.

S - Específico (Specific)

Significa que sua meta deve ser específica naquilo que você se propõe a fazer, ou seja, deve ter um propósito claro.

Não pode haver subjetividade, por isso, deve ser elaborada de forma bem objetiva, precisa e clara, pois isto irá impactar em muito o resultado.

Exemplo: Se você deseja aumentar as suas vendas, não adianta apenas dizer que quer aumentar, você deve especificar em quanto, nem que seja em 1%.

Tipos de perguntas para te ajudar a especificar sua meta:

- Como será alcançada?
- Quem será responsável?

- O que vou alcançar com essa meta?
- Por que tenho que realizar?

M - Mensurável (Measurable)

- Você só terá real noção dos resultados, sejam positivos ou negativos, se suas ações forem mensuradas.
- É de extrema importância que de forma periódica as ações sejam medidas e analisadas para saber se o objetivo foi alcançado ou não.

Exemplo: se você quer aumentar a visibilidade do seu site, é preciso mensurar quantos novos visitantes você está obtendo à medida que você divulga um novo conteúdo.

Tipos de perguntas para te ajudar a mensurar sua meta:

- O que devo mensurar?
- Quando devo medir?
- Qual resultado devo esperar

A - Atingível (Achievable)

- Significa que sua meta deve ser alcançável e realista, ou seja, coerente com a realidade na qual você se encontra. Sua meta deve ser desafiadora, pois se for muito difícil ou praticamente impossível você ou seus subordinados não terão incentivos para tentar alcançá-la e isso se tornará um motivo para desmotivação.
- Caso a meta seja muito fácil, você também pode se sentir frustrado por não demandar muito esforço e não perceberá valor ou sentido em ter que realizar a atividade.

R - Relevante (Relevant)

- Ao traçar uma meta, você deve ter em mente os benefícios que você obterá caso consiga realizá-la. Ou seja, ela deve ser relevante para você ou sua empresa, deve ser algo que de fato te agregue um tipo de valor.
- Para ser relevante, você ou sua equipe deve ter o sentimento de dever cumprido e que seus esforços valeram a pena, de modo despertar o potencial e motivação de quem executa.

T - Temporal (Time-based)

- Ao traçar uma meta é de extrema importância que ela possua um prazo para ser cumprida, ela deve ter tempo definido para ser feita.
- Sem uma data de entregue marcada não há incentivos para fazê-la, pois ela não se torna uma prioridade e qualquer tarefa rotineira pode ser mais atraente para se fazer.
- Exemplo: Aumentar as vendas de um determinado produto em 10% em três meses.

A metodologia de estabelecimento de metas SMART facilita a criação de diretrizes de negócios realistas.

Você deve usar as metas da SMART porque elas ajudam a esclarecer suas prioridades, permitem que você se concentre e fique direcionado para o que é mais relevante.

Em vez de objetivos obscuros, sem nenhuma clareza, as métricas SMART são específicas e permitem que você saiba o que precisa fazer, quando e a importância de atingir essas metas em relação aos objetivos estratégicos do negócio. Assim, isso força você a administrar melhor o seu tempo.

Além disso, desfruta de um sentimento de realização, quando consegue superar as barreiras e chegar no ponto que planejou.

Capítulo 15. O que é Backlog Grooming

O termo Grooming é uma palavra de origem inglesa que significa “preparar”.

No Scrum, “grooming” é utilizado e falado pelos profissionais que trabalham com os Métodos Ágeis, e foi inserido na comunidade por Ken Schwaber, um dos fundadores dessa metodologia ágil há alguns anos.

O termo se refere à preparação de backlog (que são os requisitos ou lista de pendências dentro do Scrum) e é mais adequadamente conhecido como refinamento. É o ato de detalhar, entender mais profundamente, adicionar características, estimar, priorizar e manter o backlog do produto vivo.

Então, **o que é grooming**? Nada mais é que um ou vários momento(s) específico(s) em que se iniciam os preparativos para o próximo sprint ou release.

Só para relembrar: sprint é o intervalo no qual são preparadas diversas entregas do projeto no planejamento ágil, especialmente na metodologia Scrum.

Enquanto o time trabalha na sprint atual, o Product Owner (PO), com mais alguns convidados de seu interesse, começa a refinar e amadurecer as estórias e objetivos da próxima sprint ou release.

Simplificando, Grooming é: entender melhor os itens de backlog das Sprints futuras, olhando de preferência apenas o trabalho a ser realizado de uma a três Sprints para a frente.

Então, é possível dizer que para grooming, definição é: preparação da lista de pendências, também conhecida como refinamento da lista de pendências ou tempo da estória.

Origem:

O termo backlog grooming surgiu em 2005, quando Mike Cohn fazia a lista de discussão sobre o desenvolvimento da metodologia Scrum;

- Já em 2008, Kane Mar deu uma das primeiras descrições formais da preparação das pendências com o nome “Tempo da Estória”. Na ocasião, ele sugeriu uma reunião regular para tratar do assunto;
- Em 2011, a prática foi incluída oficialmente no Guia do Scrum.

Para Roman Pichler, assim como o seu jardim, que vai crescendo até se tornar uma selva se não for devidamente cuidado, o seu backlog também vai crescendo e tornando desajeitado se não for dada a ele a devida atenção.

Organizar o backlog é um processo contínuo que envolve:

- A descoberta de novos itens, assim como alteração e remoção de itens antigos.
- Quebrar estórias muito grandes (épicos).
- A priorização dos itens do backlog (trazendo os mais importantes para o topo).
- Preparar e refinar os itens mais importantes para a próxima reunião de planejamento.
- Estimar e corrigir estimativas dos itens do backlog (em caso de novas descobertas).
- Incluir Critérios de Aceitação.

Embora a manutenção do Backlog seja de responsabilidade do Product Owner, outros membros do time podem colaborar na reunião de Organização do Backlog, Ken Schwaber sugere a participação de **10%** da equipe durante de **5 a 10%** do tempo da sprint, de forma a incentivar a comunicação face-a-face entre as pessoas ao invés de outros meios, como comunicação por e-mail ou documentos.

Trazendo um pouquinho de Scrum.org à conversa, na página 15 do Scrum Guide é dito que a Refinement não pode consumir mais de 10% da capacidade da equipe de desenvolvimento.

Texto original: “Refinement usually consumes no more than 10% of the capacity of the Development Team.”

Considerando 40 horas semanais, não pode levar mais do que 4 horas por semana.

Roman sugere a utilização de cartões de papel como ferramenta para facilitar a colaboração na reunião, cada participante pode pegar um cartão e escrever suas ideias.

Depois, estas ideias podem ser agrupadas (juntando-se esses cartões em pequenos montes e agrupando num quadro na parede) e, caso necessário, pode-se transferir as informações para uma ferramenta eletrônica.

Com base no resultado do Backlog Grooming você vai poder organizar seu backlog, fazer o planejamento de iterações e releases, e organizar os itens de trabalho da sua equipe em categorias e marcá-los com tags para localizar facilmente itens semelhantes ou relacionados (para isso utilizar um software pode ser muito útil).

Capítulo 16. Reuniões de Backlog Grooming

Para que seja realizado um grooming de maneira eficiente, durante a Sprint corrente, o PO e time de desenvolvimento, podendo ser um integrante ou mais, se reúnem para refinar os itens de backlog do futuro.

O Product Owner deve trazer o entendimento das necessidades de negócio já obtidas e apresentá-las ao time. O time, por sua vez, aproveita a oportunidade para tirar dúvidas e fazer perguntas ao PO a respeito dos itens que estão sendo “groomados”.

Com isso, o PO terá a oportunidade de responder as perguntas nas quais já tem as respostas enquanto o time entende melhor e toma decisões técnicas a respeito de como o item será completado.

Para as perguntas que o PO não tiver resposta ele poderá aproveitar para formular novas perguntas a serem feitas ao usuário final que deverão ser trazidas com as devidas respostas e o entendimento necessário para a próxima Sprint Planning.

De acordo com o produto, poderá ser necessário construir especificações mais técnicas, envolver outros times, contatar fornecedores ou parceiros, e o grooming servirá para disparar estas necessidades antes que o momento de trabalhar nos itens chegue e o time seja pego de surpresa sem os entendimentos necessários e o item fique impedido de ser completado.

Definition Of Ready (DoR)

- Apesar de não fazer parte do Scrum Guide (Guia do Scrum), a DoR (Definition Of Ready ou, traduzindo, Definição de Pronto) pode atuar como uma diretriz para a equipe durante o refinamento da backlog.
- O grooming é uma forma de preparar os itens de backlog das Sprints futuras, refinando-os durante a Sprint corrente e preparando um backlog futuro que evite impedimentos, satisfazendo a sua DoR.

O grooming também irá contribuir para que a reunião de planejamento da Sprint futura seja mais objetiva e até mais curta, pois o time, ou parte dele, já conheceu um pouco mais dos itens antes da cerimônia de planejamento e ajudou o PO a antecipar questionamentos que seriam feitos durante a planning.

Apesar de o PO ou o gerente de produtos serem os facilitadores do entendimento da lista de pendências, não é incomum que um gerente de projetos ou até mesmo o Scrum Master assumam esse papel também.

Uma reunião de Backlog Grooming deve ser realizada próximo ao final da iteração (sprint), garantindo assim que o Product Backlog esteja sempre pronto para a próxima.

Além da chance para a equipe fazer perguntas, e gerentes e PO explicarem as estratégias por trás dos itens, a reunião de grooming envolve:

- A priorização dos itens do Product Backlog;
- Realizar a descoberta de novos itens;
- Dividir itens muito grandes (épicos) para facilitar planejamento, execução e entrega;
- Quando ocorre uma quebra nas estórias, muitas vezes, é preciso alterar itens e remover aquelas estórias de usuário desnecessárias ou antigas;
- Preparar e refinar os itens mais importantes com as informações mais necessárias (critérios e regras do negócio, fluxos, protótipos, etc.) para a próxima reunião de planejamento;
- Estimar, corrigir e reestimar estimativas dos itens do Product Backlog (em caso de alterações);
- E tudo mais que envolver organização, ordenação e limpeza.

Algumas dicas importantes:

- No dia anterior à reunião, envie um e-mail à toda equipe de desenvolvimento com as User Stories que farão parte da reunião. Em geral, entre 4 e 7 dependendo do tamanho e complexidade atribuídos a elas.
- As equipes têm tempo para analisar as User Stories e pensar com calma nos pontos que precisam ser discutidos/esclarecidos.
- Com base nesta informação eles escolhem quem participará da **Refinement**. Não é necessário avisar quem estará presente.

Muitos profissionais consideram a reunião de refinamento como a segunda mais importante “cerimônia do Scrum”, embora não seja oficialmente “do Scrum”.

A(s) primeira(s) Refinement(s) tendem a estenderem-se e a não estimar todas as User Stories previstas. Isso é normal e, como todo o processo Scrum, melhora muito ao longo do tempo.

Capítulo 17. O que é UML

No final dos anos 80 e início dos anos 90, tínhamos muitos conflitos de definições e nomenclaturas na área de modelagem. A escolha para utilização de um determinado padrão era definido mais pelo “gosto” pessoal do que por fatores técnicos oferecidos.

Então, os três mais respeitados nomes nesse campo, cada qual com seu conceito e implementação de modelo, Ivar Jacobson (OOSE – Object Oriented Software Engineering), Grady Booch (The Booch Method) and James Rumbaugh (OMT – Object Modeling Technique) decidiram pôr fim aos debates e trabalhar juntos na definição de um modelo único que veio a ser a UML – Unified Modeling Language.

A UML permite que você “desenhe” uma “planta” do seu sistema.

Esta “planta” garante, em todas as fases do projeto, seja na definição, desenvolvimento, homologação, distribuição, utilização e manutenção do mesmo, uma maior clareza e objetividade para execução de cada ação, e, com certeza, quanto maior a solução, maior a necessidade de um projeto definido adequadamente.

Desta forma, a UML é uma linguagem padrão para visualização, especificação, construção e documentação de um aplicativo ou projeto de software, e objetiva aumentar a produtividade, otimizar as etapas que envolvem o desenvolvimento de um sistema, aumentando assim a qualidade do produto a ser implementado.

Ela independe da ferramenta em que o aplicativo será desenvolvido. A ideia é prover uma visão lógica de todo o processo de forma a facilitar a implementação física do mesmo.

A UML disponibiliza, através de conceitos, objetos, símbolos e diagramas, uma forma simples, mas objetiva e funcional, de documentação e entendimento de um sistema.

Você pode utilizar os diagramas e arquivos que compõe um modelo UML para o desenvolvimento, apresentação, treinamento e manutenção durante todo o ciclo de vida da sua aplicação.

Ela é mais completa que outras metodologias empregadas para a modelagem de dados, pois tem em seu conjunto todos os recursos necessários para suprir as necessidades de todas as etapas que compõe um projeto, desde a definição, implementação, criação do modelo de banco de dados, distribuição, enfim, proporcionando sem qualquer outra ferramenta ou metodologia adicional, um total controle do projeto.

A UML implementa uma modelagem com uma visão orientada a objetos. Através dela, podemos definir as classes que compõe a nossa solução, seus atributos, métodos e como elas interagem entre si.

Apesar da UML ter como base a orientação a objetos, não significa que a ferramenta e a linguagem utilizada para a implementação do modelo sejam também orientadas a objetos, embora seja recomendável.

Os diagramas têm como objetivo representar, através de um conjunto de elementos, como o sistema irá funcionar e como cada peça do sistema irá trabalhar e interagir com as outras.

Outra vantagem vem da facilidade de leitura dos diagramas que compõe a UML, além da facilidade de confeccioná-los, pois existem inúmeras ferramentas para modelagem de dados orientados a objetos (ferramentas Case).

Além dos diagramas citados a UML disponibiliza outros diagramas, dentre os quais podemos citar:

- Diagrama de Objetos;
- Diagrama de Sequência;
- Diagrama de Colaboração;

- Diagrama de Estado;
- Diagrama de Atividade; e
- Diagrama de Componentes.

Dado que estão entre os diagramas mais utilizados no mercado, vamos conhecer agora os elementos básicos de um diagrama de caso de uso e de um diagrama de classe.

A UML se mostra como parte essencial no “ciclo de vida” de uma aplicação.

Foi mostrado, utilizando apenas dois diagramas, toda a funcionalidade operacional de um sistema bem como a definição de elementos internos referentes ao desenvolvimento do mesmo.

Esses dois diagramas fazem parte da documentação do sistema, e podem ser utilizados para uma apresentação da solução para o requisitante antes da implementação da mesma, visualização dos processos que o sistema irá disponibilizar, definição de elementos inerentes ao desenvolvimento como estrutura de telas, procedimentos operacionais, referência para criação de objetos de persistência em um banco de dados, etc.

Capítulo 18. Diagramas de Caso de Uso

Esse diagrama documenta o que o sistema faz do ponto de vista do usuário. Em outras palavras, ele descreve as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do mesmo sistema.

Nesse diagrama não nos aprofundamos em detalhes técnicos que dizem como o sistema faz.

Este artefato é comumente derivado da especificação de requisitos, que por sua vez não faz parte da UML.

Pode ser utilizado também para criar o documento de requisitos.

O diagrama de caso de uso não oferece muitos detalhes — não espere, por exemplo, que ele mostre a ordem em que os passos são executados. Em vez disso, um diagrama de caso de uso adequado dá uma visão geral do relacionamento entre casos de uso, atores e sistemas.

Recomenda-se usar o diagrama de caso de uso para complementar um caso de uso descrito em texto.

Diagramas de Casos de Uso são compostos basicamente por quatro partes:

- Cenário: Sequência de eventos que acontecem quando um usuário interage com o sistema.
- Ator: Usuário do sistema, ou melhor, um tipo de usuário.
- Use Case: É uma tarefa ou uma funcionalidade realizada pelo ator (usuário).
- Comunicação: É o que liga um ator com um caso de uso.

Um ator é uma entidade externa ao sistema que de alguma forma participa de um caso de uso.

Um ator pode ser um ser humano, máquinas, dispositivos ou outros sistemas. Atores típicos são cliente, usuário, gerente, computador, impressora, etc.

Os atores representam um papel e iniciam um caso de uso que após executado, retorna um valor para o ator.

Um caso de uso especifica um serviço que será executado ao usuário e é composto por um ou mais cenários. Um cenário é uma narrativa de uma parte do comportamento global do sistema.

Para melhorar nossos diagramas, podemos utilizar relacionamentos.

Include e **extend** são relações entre os casos de uso.

- Include: Seria a relação de um caso de uso que, para ter sua funcionalidade executada, precisa chamar outro caso de uso.
- Extend: Esta relação significa que o caso de uso estendido vai funcionar exatamente como o caso de uso base só que alguns passos novos inseridos no caso de uso estendido.
- Tanto um como o outro, são notados como setas tracejadas com o texto <> ou <>.

Um bom diagrama de caso de uso ajuda sua equipe a representar e discutir:

- Cenários em que o sistema ou aplicativo interage com pessoas, organizações ou sistemas externos.
- Metas que o sistema ou aplicativo ajuda essas entidades (conhecidas como atores) a atingir.
- O escopo do sistema.

Com os diagramas de caso de uso podemos trabalhar em três áreas muito importantes nos projetos:

- **Definição de Requisitos** - Novos casos de usos geralmente geram novos requisitos conforme o sistema vai sendo analisado e modelado;
- **Comunicação com os Clientes** - Pela sua simplicidade, sua compreensão não exige conhecimentos técnicos, portanto o cliente pode entender muito bem esse diagrama, que auxilia o pessoal técnico na comunicação com clientes;
- **Geração de Casos de Teste** - A junção de todos os cenários para um caso de uso pode sugerir uma bateria de testes para cada cenário.

Capítulo 19. As cinco visões do Modelo 4+1

O Modelo de Arquitetura de Software 4 + 1 foi descrito por Philippe Kruchten's em *Architectural Blueprints - O "4 + 1" View Model of Software Architeture* que foi publicado originalmente no IEEE Software (novembro de 1995).

Esta publicação não faz referências específicas à UML.

Em vez de tentar mapear cada uma das visualizações para tipos específicos de diagramas, considere quem é o público-alvo de cada visualização e quais informações são necessárias.

Visão de **caso de uso**:

Descreve a arquitetura do sistema através do uso de Diagramas de casos de uso.

Cada diagrama descreve sequências de interações entre os objetos e processos. São usados para identificar elementos de arquitetura e ilustrar e validar o design de arquitetura.

Visão **lógica**:

Se concentra na funcionalidade que o sistema disponibiliza para o usuário final.

Os diagramas UML usados para representar a visão lógica incluem: Diagrama de classes, Diagrama de comunicação e Diagrama de sequência.

A visão lógica suporta principalmente os requisitos funcionais e é fortemente centrada nos conceitos da orientação à objetos.

A visão lógica é projetada para atender às preocupações do usuário final em garantir que toda a funcionalidade desejada seja capturada pelo sistema.

Em um sistema orientado a objeto, isso geralmente ocorre no nível de classe. Em sistemas complexos, você pode precisar de uma visão de pacote e decompor os pacotes em vários diagramas de classes.

Em outros paradigmas, você pode estar interessado em representar os módulos e as funções que eles fornecem.

O resultado final deve ser um mapeamento da funcionalidade necessária para os componentes que fornecem essa funcionalidade.

Visão de **implementação**:

- Usada para capturar os subsistemas em um modelo de implementação, normalmente representada como um diagrama de componentes.
- Expressa a gestão da configuração.
- Ilustra o sistema do ponto de vista do programador e se preocupa com o gerenciamento de projeto.
- Esta visão também é conhecida como visão de desenvolvimento.
- Usa o Diagrama de componentes ou Diagrama de pacotes.

A visão de implementação é focada na modularização do software, a qual o sistema é divido em pequenos subsistemas organizados hierarquicamente em camadas, servindo como base para linha de produto e dando suporte na divisão de tarefas para a equipe de desenvolvimento.

A visão de desenvolvimento é principalmente para desenvolvedores que estarão construindo os módulos e os subsistemas.

Ele deve mostrar dependências e relacionamentos entre os módulos, como os módulos são organizados, reutilizados e portáveis.

Visão de **implantação**:

Mostra o sistema do ponto de vista do engenheiro.

Se preocupa com a topologia dos componentes de software (no contexto físico) assim como a comunicação entre esses componentes.

Esta visão também é conhecida como visão física. Os diagramas UML usados para descrever esta visão incluem o diagrama de implantação.

A visão de implantação leva em consideração os requisitos não funcionais do sistema e tem um impacto mínimo em todo código-fonte, sendo seu mapeamento feito em todos os nós, por isso precisa ser altamente flexível.

A visão física é principalmente para os criadores de sistemas e administradores que precisam entender os locais físicos do software, as conexões físicas entre os nós, a implantação e a instalação e a escalabilidade.

Visão de **processos**:

Permite visualizar as partes dinâmicas do sistema, explicar os processos e como eles se comunicam, focando no comportamento do sistema.

A visão de processo se encarrega da concorrência, distribuição, integração, performance e escalabilidade.

O Diagrama de atividades é usado nesta visão.

A visão de processos está relacionada com a parte dos requisitos não funcionais.

A visão de processo é projetada para pessoas que projetam todo o sistema e, em seguida, integram os subsistemas ou o sistema em um sistema de sistemas.

Essa visualização mostra tarefas e processos que o sistema possui, interfaces com o mundo externo e/ou entre componentes dentro do sistema, as mensagens enviadas e recebidas e como o desempenho, a disponibilidade, a tolerância a falhas e a integridade estão sendo resolvidos.

Os pontos de vista não apresentam independências, eles estão ligados entre si, ou seja, há uma correspondência entre as visões, que são organizadas em: da

lógica para a visão de processos, da lógica para o desenvolvimento e do processo para a física.

Nem sempre arquiteturas precisam ter “4+1” pontos de vista, porém os cenários são úteis em todos os casos.

Capítulo 20. Modelo 4+1 – O que usar

Nem todos os sistemas precisam de todas as visões. Da mesma forma que alguns sistemas precisam de visões adicionais.

- Sistemas pequenos normalmente ignoram a visão de implementação.
- Se o processo for único, normalmente ignora-se a visão de processos.
- Se as questões de implantação forem simples, ignora-se essa visão. Porém, se forem complexas, mais profissionais podem ser demandados. Como analistas de suporte, de rede, arquitetos de nuvem, analistas de sistemas, etc.

Alguns sistemas podem demandar visões adicionais:

- Visão de dados.
- Visão de segurança.
  - LGPD – Lei Geral de Proteção de Dados Pessoais.
- E quaisquer outros aspectos pertinentes ao contexto do negócio em análise.

Hoje em dia é muito comum necessitarmos da visão de dados:

- Seja para questões de análise de dados/Data Science;
- Seja para questões de BI - Business Intelligence;
- Ou para outras questões relacionadas à análise de big data.

A união entre time de dados, arquitetos de software e analista de negócio/requisitos e testes é essencial para o sucesso dessa visão.

A visão de dados é uma especialização da visão lógica.

Use esta visualização se a persistência for um aspecto significativo do sistema e a conversão do modelo de design para o modelo de dados não for feita automaticamente pelo mecanismo de persistência.

Para cada visualização, existe um conjunto diferente de diagramas UML que podem ser úteis para transmitir as informações que você deseja contar nessa visualização.

O Modelo 4+1 não é um modelo bala de prata que pode ser utilizado para resolver todos os problemas de arquitetura. Porém, pode oferecer uma visão ampla que permita com que a gestão de riscos, entre diversas outras áreas do projeto, seja feita com mais clareza e assertividade.

As visões de arquitetura de software surgiram em resposta à insistência de alguns autores em representar toda a arquitetura do sistema através de uma única representação.

A visão de casos de uso faz a integração entre as outras quatro visões.

Visão de arquitetura é uma representação da informação contida na arquitetura que facilita o entendimento por parte do interessado.

O arquiteto de software pode usar diferentes visões para lidar com a complexidade, sua representação serve como guia para o projeto de sua implementação, teste e implantação do sistema.

Visões permitem reduzir a quantidade de informação que o arquiteto trata em um dado momento.

A arquitetura deve descrever visões complementares. Segundo o próprio Philippe Kruchten:

- O modelo de visualização “4 + 1” é bastante “genérico”:
- Outras notações e ferramentas podem ser usadas, outros métodos de design podem ser usados, especialmente para as decomposições lógicas e de processo, mas indicamos aqueles que usamos com sucesso.

A pessoa arquiteta de software deve se atentar para:

- Escolher as visões relevantes para o seu sistema.

- Documentar a visão.
- Adicionar documentação que se aplica a mais de uma visão.

Referências

CAMARGO, Robson. *Backlog grooming:* as melhores práticas para refinar o seu produto. Robson Camargo Projetos e Negócios, 2019. Disponível em: <https://robsoncamargo.com.br/blog/Conheca-as-melhores-praticas-de-uma-backlog-grooming/>. Acesso em: 30 set. 2020.

CHUNG, L. et al. *Non-Functional Requirements in Software Engineering*. Disponível em: <http://www.utdallas.edu/~chung/BOOK/book.html>. Acesso em: 30 set. 2020.

COELHO, Jailton. *Gerenciamento de Requisitos*. Lean TI, 2014. Disponível em: <https://www.leanti.com.br/artigos/16/gerenciamento-de-requisitos.aspx>. Acesso em: 30 set. 2020.

FERGUSON, Robert; LAMI, Giuseppe. An Empirical Study on The Relationship Between Defective Requirements and Test Failures. In: *Software Engineering Workshop*, v. 0., 2006. p. 7-10

FriendsLab. *Meta SMART:* saiba como fazer uma meta inteligente. 2019. Disponível em: <https://www.friendslab.co/o-que-e-meta-smart/>. Acesso em: 30 set. 2020.

KRUCHTEN, Philippe. Architectural Blueprints — The “4+1” View Model of Software Architecture. In: *IEEE Software*, v. 12. n. 6., 1995. p. 42-50. Disponível em: <https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf>. Acesso em: 30 set. 2020.

LEITE, Julio Cesar. *Livro Vivo*: Engenharia de Requisitos. Blogspot. Disponível em: <http://livrodeengenhariaderequisitos.blogspot.com.br/>. Acesso em: 30 set. 2020.

MACHADO, R.; SEIDMAN, S. A Requirements Engineering and Management Training Course for Software Development Professionals. In: *22th Conference on Software Engineering Education and Training*, 2009.

MORAN, Alan. *Agile Risk Management*. Editora Springer. 2014.

PAULA FILHO, Wilson de Pádua. *Engenharia de Software*: fundamentos, métodos e padrões. São Paulo: LTC Editora, 2000.

PMBOK. *Um guia do conhecimento em gerenciamento de projetos* (Guia PMBOK). 4. ed. Pennsylvania: Project Management Institute, Inc., 2008.

PRESSMAN, R. S. *Software Engineering:* A Practitioner's Approach. 7. ed., McGraw Hill, 2010.

SCHWABER, K. SUTHERLAND, J. *The Scrum Guide - The Definitive Guide to Scrum:* The Rules of the Game. 2013. Disponível em: <http://www.scrumguides.org/>. Acesso em: 30 set. 2020.

SITEWARE. *O que é meta SMART*: Veja as 5 características mais importantes para definir uma meta que funcione de verdade. 2019. Disponível em: <https://www.siteware.com.br/metodologias/o-que-e-meta-smart/>. Acesso em: 30 set. 2020.

SOMMERVILLE, I. *Software Engineering*. 8. ed., Addison-Wesley, 2007.

The Software Architecture Portal. Disponível em: <http://www.softwarearchitectureportal.org/>. Acesso em: 30 set. 2020.

TYLDESLEY, D. A. *Employing usability engineering in development of office products*. In: Computer Journal, v. 31. n. 5., 1998. p. 431-436.