Certified Professional for Requirements Engineering

Handbook

RE@Agile

Practitioner | Specialist

Peter Hruschka, Kim Lauenroth,
Markus Meuten, Gareth Rogers,
Stefan Gärtner, Hans-Jörg Steffe

2.1.0 | 18 de fevereiro de 2025

Termos de uso

Agradecimentos

Este handbook foi inicialmente criado em 2018 por Bernd Aschauer, Peter Hruschka, Kim Lauenroth, Markus Meuten, Gareth Rogers.

Nossos agradecimentos a Rainer Grau por suas revisões intensivas do syllabus RE@Agile, a todos os revisores deste documento, bem como a Stefan Sturm, Sibylle Becker e Ruth Rossi por seu encorajamento e apoio. Os comentários de revisão foram fornecidos por Sacha Reis e Sven van der Zee. Revisão por Michiel van der Voort. Traduzido do Inglês por Ana Moreira, Carlos André e Silva, George Fialkovitz, Guilherme Simões. Anselmo Peretto e Martin Tornquist.

Aprovado para release em 18 de fevereiro de 2022, pelo Conselho do IREB, por recomendação de Xavier Franch.

Agradecemos a todos pelo seu envolvimento.

Sumário

Prefácio

Este Handbook complementa o syllabus do módulo RE@Agile do CPRE.

Este Handbook é destinado a provedores de treinamento que desejam oferecer seminários ou treinamento sobre RE@Agile Practitioner e/ou Specialist de acordo com o padrão IREB. Também se destina ao treinamento de participantes e interessados que desejam obter uma visão detalhada sobre o conteúdo deste módulo. Ele também pode ser usado ao aplicar métodos de Engenharia de Requisitos em um ambiente Ágil, de acordo com a norma IREB.

Este Handbook não substitui o treinamento sobre o tema. O Handbook representa uma ligação entre o Syllabus (que lista e explica os objetivos de aprendizagem do módulo) e a ampla gama de literatura que foi publicada sobre o tema.

O conteúdo deste Handbook, juntamente com referências a literatura mais detalhada, apoia os prestadores de treinamento na preparação dos participantes do treinamento para o exame de certificação. Este Handbook oferece aos participantes do treinamento e aos interessados uma oportunidade de aprofundar seus conhecimentos de Engenharia de Requisitos em um ambiente Ágil e de complementar o conteúdo detalhado com base nas recomendações da literatura. Além disso, este Handbook pode ser usado para atualizar o conhecimento existente sobre os vários tópicos do RE@Agile, por exemplo, após ter recebido o RE@Agile Practitioner ou o certificado RE@Agile Specialist.

Sugestões de melhorias e correções são sempre bem-vindas!

Contato por e-mail: info@ireb.org

Esperamos que você goste de estudar por este Handbook e que seja aprovado no exame de certificação para o IREB Certified Professional for Requirements Engineering RE@Agile Practitioner ou Specialist.

Mais informações sobre o módulo RE@Agile do IREB Certified Professional for Requirements Engineering Module podem ser encontradas em: http://www.ireb.org.

Histórico das Versões

Versão	Data	Comentário
2.1.0	18 de fevereiro de 2025	Versão Inicial baseada no Syllabus original do IREB em inglês

1 O que é RE@Agile

Uma boa engenharia de requisitos é um fator de sucesso reconhecido para o desenvolvimento de produtos ou sistemas, independentemente da metodologia de desenvolvimento aplicada.

Neste capítulo você terá uma compreensão dos antecedentes e história da Engenharia de Requisitos e dos antecedentes e história das abordagens Ágeis (capítulo 1.1). Você aprenderá porque às vezes estas duas disciplinas são consideradas incompatíveis – o que é um conceito popular errado. Você aprenderá que – apesar de sua história – as técnicas e métodos da disciplina de Engenharia de Requisitos estão sendo utilizadas (sem uma referência clara a sua origem) em abordagens específicas de desenvolvimento (como Cascata e Scrum). Você também aprenderá que abordagens Ágeis (como Scrum, Lean Development e Kanban) precisam de boas práticas de requisitos para fornecer produtos e sistemas de sucesso.

No capítulo 1.2 discutiremos os pontos fortes e fracos dos métodos de Engenharia de Requisitos e de abordagens Ágeis. Enquanto a Engenharia de Requisitos enfatiza a importância de elucidar, compreender e documentar os requisitos dos principais stakeholders a fim de não construir o produto ou sistema errado, a maioria das abordagens Ágeis enfatiza a importância de uma cooperação confiável entre stakeholders. No Ágil, loops de feedback frequentes com base em resultados visíveis são usados para evitar suposições erradas ou períodos de mal-entendidos que duram muito tempo.

O IREB desenvolveu o módulo avançado RE@Agile para combinar os pontos fortes de ambas as disciplinas. Como você pode adivinhar: os objetivos da Engenharia de Requisitos E abordagens Ágeis não estão em conflito, mas eles se complementam – se aplicados corretamente!

O capítulo final 1.3 apresenta a definição da RE@Agile do IREB. Em poucas palavras, você aprenderá como seus projetos de desenvolvimento podem se beneficiar desta abordagem integrada.

1.1 História da Engenharia de Requisitos e o Ágil

Com base em suas respectivas histórias, a Engenharia de Requisitos e abordagens Ágeis são frequentemente consideradas separadamente e não em conjunto. Consideremos alguns marcos-chave nestas histórias para entender melhor como surgiu esta situação. Estes marcos são capturados de forma geral em Figura 1. (note que eles foram escolhidos pelos autores deste Handbook para enfatizar fontes importantes para esto módulo. Não afirmamos ter capturado o histórico completo dos métodos de desenvolvimento).

No final dos anos 70, o termo "crise de software" ressoou em toda a comunidade de TI. A queixa mais importante: O desenvolvimento de produtos é um processo complexo e muitas vezes os produtos não satisfazem os usuários. A resposta dos cientistas e metodologistas foi o modelo cascata (sugerido originalmente por Winston Royce, mas tornado popular por Barry Boehm). Uma de suas soluções para a crise de software foi introduzir uma "Fase de

Requisitos" antes de projetar, construir e testar sistemas. Seu objetivo era chegar a um acordo entre os stakeholders importantes sobre o produto pretendido fazer antes de construí-lo.

As especificações dos requisitos naquela época eram, na sua maioria, documentos em linguagem natural. Por volta da mesma época (meados dos anos 70 até seu final) muitas sugestões foram feitas para usar modelos gráficos, além de texto, com o objetivo de melhorar a precisão dos requisitos evitando inconsistências. Em 1975, Peter Chen sugeriu modelos de Entidade-Relacionamento para capturar dados relevantes para os negócios. Em 1978/1979 Douglas Ross e Tom DeMarco introduziram a Análise Estruturada e Técnica de Projeto (SADT) para capturar a funcionalidade de negócio.

Em meados dos anos 80, Barry Boehm formulou o "Modelo Espiral". A Engenharia de Requisitos tornou-se uma técnica iterativa, ao mesmo tempo em que introduz ciclos de gerenciamento de risco e de feedback mais frequentes.

De um ponto de vista de método e notação, 1992 foi um marco importante para a Engenharia de Requisitos: Ivar Jacobson propôs uma "Abordagem Orientada por Casos de Uso". A ideia de enfatizar "atores" (ou usuários) no contexto do sistema, e de pensar de ponta a ponta na totalidade do produto, não eram novas.

McMenamin/Palmer (em 1984) e Hammer/Champy, em sua Metodologia "Business Process Reengineering", também enfatizaram este tipo de pensamento de processo. Mas a notação de Ivar Jacobson – figuras simples de traços e elipses, apoiadas por descrições de linguagem natural destes casos de uso – tornou-se muito popular.

Outro marco importante para a Engenharia de Requisitos foi o Processo Unificado de Ivar Jacobson – tornado popular como o "Rational Unified Process" (RUP). O RUP reconhece a Engenharia de Requisitos como uma "disciplina" ao invés de uma "fase". Esta disciplina abrange todas as fases (com diferentes graus de ênfase).

Todos os modelos modernos de processo adotaram esta distinção entre disciplinas (como análise de negócio, requisitos, projeto, implementação, teste) e fases (como início, elaboração, construção, transição – na terminologia do RUP). Os últimos permitem marcos controláveis, enquanto os primeiros garantem que sejam estabelecidas técnicas e práticas apropriadas para o trabalho em andamento.

A padronização internacional da UML (Unified Modeling Language) em 1997 pelo OMG (Object Management Group) ajudou a tornar mais popular as especificações de requisitos usando modelos de casos de uso, diagramas de atividades, gráficos de estados e assim por diante, especialmente porque muitas ferramentas suportavam essas notações.

Pensar em termos de processos de negócios de ponta a ponta foi ainda mais aprimorado pela padronização do BPMN (Business Process Model and Notation). Enquanto os casos de uso de Ivar Jacobson foram muitas vezes mal interpretados para serem "apenas a parte de TI dos processos de negócio", os modelos BPMN estão mais próximos do "negócio". Isto tratou de uma importante questão da RE: o alinhamento dos negócios e da TI.

Outro aspecto importante da Engenharia de Requisitos foi discutido já em 1986 com a introdução do FURPS pela HP: a importância dos requisitos de qualidade. FURPS

(Functionality, Usability, Reliability, Performance and Supportability) foi uma das primeiras abordagens para enfatizar os aspectos de qualidade além da funcionalidade.

Isto foi detalhado pela norma ISO/IEC 9126, que estabeleceu muitas categorias adicionais de qualidades a serem alcançadas pelos sistemas. A última revisão desta norma é a ISO/IEC 25010 (também conhecida como SQuaRE – Systems and Software Quality Requirements and Evaluation), na qual é enfatizada a importância da segurança nos sistemas modernos.

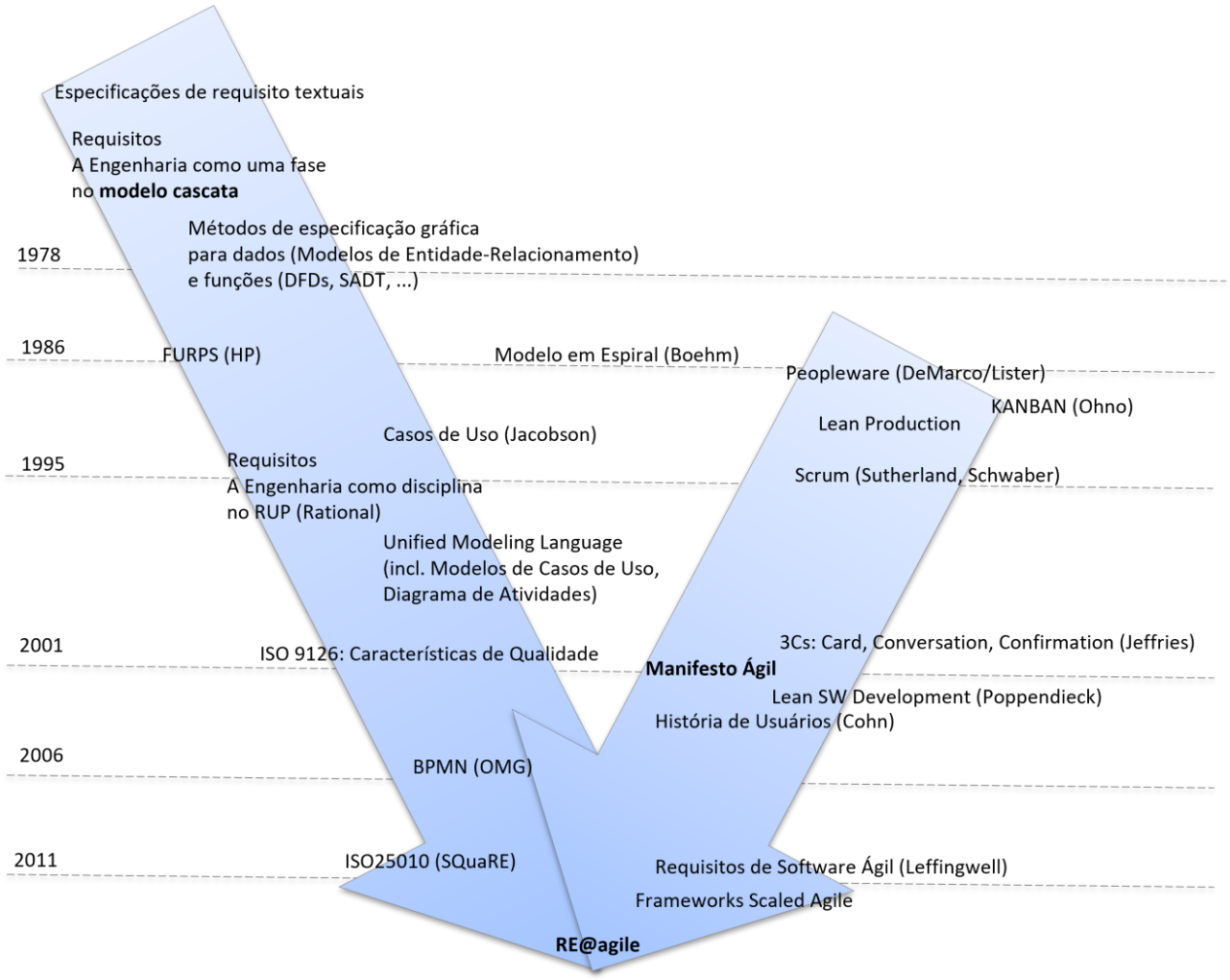


Figura 1: Marcos selecionados em RE e Ágil

Algumas ideias-chave de abordagens Ágeis foram publicadas muito antes do aparecimento do Manifesto Ágil, em 2001.

Tom DeMarco e Tim Lister cunharam o termo "Peopleware" em 1987 para enfatizar a importância da cooperação humana e das equipes.

A Toyota publicou histórias de sucesso envolvendo Kanban e Lean Manufacturing (ou Lean Production) no final dos anos 80. Ambos os conceitos (Kanban e Lean) são ideias centrais nos métodos Ágeis de hoje.

Scrum, uma "estrutura para desenvolver e sustentar produtos complexos", foi publicada pela primeira vez por Jeff Sutherland e Ken Schwaber em 1995. Introduz o papel de "Product

Owner", responsável pelo sucesso do produto dentro de uma organização. O Product Owner[1] estabelece as prioridades dos requisitos (muitas vezes chamados de épicos ou histórias). Em parte, devido a sua simplicidade (3 funções, 4 artefatos, 5 reuniões), Scrum tornou-se muito popular em todo o mundo.

Em 2001, um grupo de 17 indivíduos representando abordagens populares como Extreme Programming, Scrum, DSDM, Desenvolvimento de Software Adaptativo, Crystal, Orientado por Funcionalidades e Programação Pragmática se reuniu em Utah e concordou em um "manifesto". O Manifesto Ágil, como ficou conhecido, transferiu a ênfase do desenvolvimento do sistema de contratos, documentos, e planejamento a longo prazo e processos para a cooperação, disponibilidade para mudanças e feedback baseado em lançamentos frequentes.

No mesmo ano, Ron Jeffries – um dos signatários do Manifesto Ágil – publicou o modelo 3C (Card, Conversation, Confirmation), para distinguir histórias de usuários "sociais" de práticas "documentais", tais como casos de uso.

Alguns anos depois, Mike Cohn sugeriu um formato para estes cartões: histórias de usuários. As histórias de usuários enfatizam três questões importantes: quem quer, o que quer, e o porquê quer. (Como um <tipo de usuário> eu quero <algum objetivo> para que <algum motivo>).

Como o Scrum foi desenvolvido principalmente para equipes menores (até dez pessoas), mais e mais estruturas de escalonamento (por exemplo SAFe, LeSS, DAD...) foram publicadas de 2010 em diante, sugerindo formas de cooperação em equipes maiores ou distribuídas.

Dean Leffingwell [Leffingwell2010] cunhou o termo "Ágil Software Requirements" ao publicar um livro em 2011 com este título, que leva ao termo "Engenharia de Requisitos Ágil". Embora este termo tenha se tornado popular para executar tarefas de Engenharia de Requisitos de acordo com os princípios do Manifesto Ágil, existe o perigo de que possa levar ao mal-entendido de que existem duas formas de Engenharia de Requisitos: a clássica Engenharia de Requisitos e a Engenharia de Requisitos Ágil. A visão do IREB é que só existe uma boa ou má Engenharia de Requisitos – em um mundo não Ágil ou Ágil. Portanto, chamamos a abordagem IREB de RE@Agile.

1.2 Aprendendo uns com os outros

O Ágil e a Engenharia de Requisitos são duas disciplinas com origens diferentes e objetivos distintos que, no entanto, podem aprender muito uma com a outra.

Comecemos com algumas ideias-chave da Engenharia de Requisitos e vejamos como elas podem se beneficiar com a mentalidade Ágil. Depois disso, analisaremos alguns princípios

[1] Por uma questão de brevidade, usaremos o nome "Product Owner" neste handbook, sempre que nos referirmos a uma pessoa responsável pela gestão dos requisitos. Para uma definição, consulte o glossário.

Ágeis básicos e discutiremos como as técnicas de Engenharia de Requisitos podem melhorá-los ainda mais.

O IREB define a Engenharia de Requisitos como uma abordagem sistemática e disciplinada para a especificação do sistema com os seguintes objetivos:

1. Conhecer os requisitos relevantes, obter consenso entre os stakeholders sobre esses requisitos, documentá-los de acordo com determinadas normas e gerenciá-los sistematicamente;
2. Compreender e documentar os desejos e necessidades dos stakeholders;
3. Especificar e gerenciar os requisitos, para minimizar o risco de entregar um sistema que não atenda aos desejos e necessidades dos stakeholders.

O primeiro ponto – conhecer os requisitos relevantes antes de se aprofundar em uma solução – é indiscutível. Mas o Ágil é muito explícito sobre como "relevante" deve ser interpretado: just in time! (na hora certa!) Nem todos os requisitos são relevantes no início de um empreendimento. Uma declaração de visão ou um conjunto de objetivos são suficientes para começar. Antes que partes da solução sejam desenvolvidas, é necessário um entendimento profundo deste subconjunto de requisitos. Outros – que não são tão urgentes para o negócio – podem ser deixados mais vagos e podem ser refinados mais tarde.

Uma meta da Engenharia de Requisitos é alcançar consenso entre os stakeholders. Quem poderia desafiar este objetivo? O Ágil sugere alcançar o consenso através de uma colaboração intensiva e confiável: discussões intensivas sobre os requisitos devem ocorrer até que todos os stakeholders compartilhem o sentimento de que eles são bem compreendidos. Outro mecanismo Ágil para alcançar consenso entre os stakeholders, é o feedback rápido através de incrementos demonstráveis do produto. Em muitos ambientes, ver um incremento (parcial) do produto e poder usá-lo é bem mais propício a encontrar problemas em aberto do que criar grandes volumes de documentos precisos que muitas vezes não serão lidos.

A Engenharia de Requisitos se esforça para documentar os desejos e necessidades dos stakeholders. O Ágil nos adverte que não devemos produzir documentos por causa da produção de documentos. A documentação de requisitos (de forma adequada para os stakeholders) deve (1) apoiar o processo de obtenção de consenso ou (2) satisfazer restrições impostas externamente (p. ex., restrições legais ou requisitos de rastreabilidade) ou (3) facilitar a vida para definir requisitos para o próximo lançamento sem ser forçado a começar do zero.

O último objetivo na definição da Engenharia de Requisitos é gerenciar os requisitos, para minimizar o risco de entregar um sistema que não atenda aos desejos e necessidades dos stakeholders. Para conseguir isso, o Ágil sugere a verificação constante da prioridade e das estimativas dos itens em atraso (os requisitos).

Os Princípios Ágeis ajudam a reorientar a Engenharia de Requisitos em termos de sua eficiência, flexibilidade e natureza colaborativa. Por outro lado, há muitos insights da Engenharia de Requisitos dos quais as abordagens Ágeis também podem se beneficiar.

O Ágil encoraja fortemente a colaboração confiável e a comunicação entre todos os stakeholders relevantes. Em muitos métodos Ágeis isto geralmente significa comunicação verbal frequente e aberta entre clientes e usuários de um lado (aqueles que têm necessidades ou requisitos) e desenvolvedores do outro (aqueles que podem fornecer soluções para as necessidades e requisitos).

Embora a comunicação confiável seja uma excelente maneira de alcançar uma compreensão conjunta dos requisitos, esta não é de longe a única maneira de obter os requisitos. A Engenharia de Requisitos desenvolveu um extenso corpo de conhecimentos sobre técnicas de elicitação (p. ex. [RoRo2013]) adequado para uso em diferentes ambientes e sob restrições particulares.

> Por exemplo: técnicas de criatividade, tais como brainstorming, ajudam a criar rapidamente itens de backlog de produtos em projetos inovadores; a arqueologia de produtos pode criar resultados mais rápidos ao trabalhar em novas versões de produtos existentes; questionários podem ajudar a obter rapidamente feedback de um grande número de stakeholders amplamente dispersos que você nunca traria para uma sala de reunião.

Os Product Owners Ágeis podem se beneficiar muito tendo à sua disposição uma gama de tais técnicas de elicitação e escolhendo um subconjunto adequado que ajude a preencher o backlog de produtos mais rapidamente do que "apenas conversar".

Ao se concentrar em abordagens Ágeis de comunicação confiável, muitas vezes desvaloriza-se a importância de uma documentação precisa. A história do usuário, em particular, enfatiza que os cartões para denotar histórias são principalmente um lembrete da discussão, e não um substituto para os requisitos exatos (ver também o capítulo 3.3). Concordamos que uma linguagem natural simples (em contraste com notações mais formais) é muitas vezes uma forma adequada de entendimento. Entretanto, a linguagem natural às vezes não é suficientemente precisa para evitar interpretações errôneas. Muitas outras notações de requisitos foram desenvolvidas nas últimas décadas – incluindo muitas notações gráficas – que permitem aos stakeholders superar a falta de precisão da linguagem natural. Alguns processos de negócio podem ser mais facilmente discutidos usando diagramas de atividade, diagramas de fluxo de dados ou Business Process Model and Notation (BPMN) do que escrevendo cartões para as etapas do processo. Alguns objetos a serem tratados são às vezes mais facilmente esboçados usando modelos de informação, e alguns sistemas orientados pelo estado poderiam se beneficiar de modelos de estado para esclarecer quais atividades devem ser realizadas em qual situação. Mais uma vez, os Product Owners e desenvolvedores devem conhecer tais notações – não para aplicar um formalismo, mas sim para encurtar as discussões.

Outro princípio Ágil é fornecer software funcional com frequência, ou seja, trabalhar iterativamente e criar uma série de incrementos de produto. Não faz sentido, entretanto, começar com um desenvolvimento iterativo se a equipe não estiver alinhada com uma visão ou um conjunto de objetivos. Para um único Product Owner Scrum em pleno comando de um produto, pode ser fácil ter uma visão ou um conjunto de objetivos. Se, entretanto, o Product Owner tiver que coordenar com uma série de stakeholders "importantes", então a

análise dos stakeholders, o alinhamento das metas e a definição do escopo devem preceder qualquer trabalho detalhado de requisitos. Estas atividades estão incluídas na ideia de um "projeto inicial limpo", introduzida no capítulo 2.

Resumindo os pensamentos deste capítulo: o Ágil nos ajuda a criar uma cultura para o desenvolvimento de produtos bem-sucedidos e ajuda a Engenharia de Requisitos a se tornar mais flexível, eficiente e colaborativa. Do ponto de vista dos requisitos, os princípios do Ágil são a cooperação confiável de todos os stakeholders e a busca de resultados incrementais a curto prazo. Enquanto técnicas como histórias de usuários esboçadas em cartões de histórias funcionam bem, há muitas outras técnicas de elicitação e validação, desenvolvidas ao longo de décadas de pesquisa em Engenharia de Requisitos, que podem tornar os Product Owners e desenvolvedores ainda mais produtivos – se, é claro, aplicadas corretamente e sem exageros formalistas.

No RE@Agile Primer [Primer2017] concluímos: "O valor mais importante compartilhado pela Engenharia de Requisitos e o Ágil, é fazer o usuário final do produto feliz porque a solução se adapta às suas necessidades ou cura suas maiores dores".

Neste módulo, entraremos em detalhes para mostrar como as ideias de ambos os mundos podem ser usadas em conjunto para atingir este objetivo. Na seguinte definição de RE@Agile, primeiro achamos útil estabelecer nossos próprios princípios orientadores para o resto deste Handbook.

1.3 RE@Agile - Uma Definição

RE@Agile é uma abordagem cooperativa, iterativa e incremental com quatro objetivos:

1. Conhecer os requisitos relevantes em um nível de detalhe apropriado (a qualquer momento durante o desenvolvimento do sistema);
2. Conseguir acordo suficiente sobre os requisitos entre os stakeholders relevantes;
3. Capturar (e documentar) os requisitos de acordo com as restrições da organização;
4. Realizar todas as atividades relacionadas aos requisitos de acordo com os princípios do Manifesto Ágil.

Como mencionado acima, usaremos a terminologia Scrum de um Product Owner como o papel que é responsável pela abordagem cooperativa e, portanto, como o papel responsável pela boa Engenharia de Requisitos.

Vamos explorar em detalhes as ideias-chave desta definição:

1. **RE@Agile é uma abordagem cooperativa:**
   "Cooperativa" enfatiza a ideia Ágil de interação intensiva dos participantes, inspecionando frequentemente o produto, fornecendo feedback sobre ele e adaptando e esclarecendo os requisitos, já que trabalhando em conjunto todos podem aprender e alcançar mais.
2. **O RE@Agile é um processo iterativo:**
   Isto sugere a ideia de requisitos "just in time". Os requisitos não precisam estar completos antes de iniciar o desenho técnico e a implementação.

Os stakeholders podem definir (e refinar) iterativamente aqueles requisitos que devem ser implementados em breve com o nível de detalhe apropriado.

3. **O RE@Agile é um processo incremental:**
   A implementação daqueles requisitos que são considerados como proporcionando o maior valor de negócio ou reduzindo os maiores riscos formam o primeiro incremento. Os incrementos iniciais se esforçam para criar um produto mínimo viável (MVP) ou um produto mínimo comercializável (MMP). A partir daí, os próximos incrementos podem ser adicionados a esse produto, escolhendo constantemente os que prometem o maior valor de negócio, aumentando assim constantemente o valor de negócio do produto resultante.

O primeiro objetivo ("requisitos relevantes conhecidos no nível de detalhe apropriado") refere-se novamente à abordagem iterativa: "relevantes" são os requisitos que devem ser implementados logo. E estas devem ser entendidas com muita precisão (incluindo seus critérios de aceite) – especialmente pelos desenvolvedores.

Eles têm que estar de acordo com a "definição de preparado" (DoR). Outros requisitos – que ainda não são prioridade máxima – podem ser mantidos em um nível de abstração mais alto – apenas para serem mais detalhados assim que se tornarem mais importantes.

O pré-requisito para o segundo objetivo ("acordo suficiente entre stakeholders relevantes") é conhecer todos os stakeholders e sua relevância para o sistema que está sendo desenvolvido.

O responsável pelos requisitos (geralmente o Product Owner) têm que negociar os requisitos com os stakeholders relevantes para determinar a ordem de sua implementação.

As abordagens ágeis valorizam a comunicação intensiva e contínua sobre os requisitos em relação à comunicação sobre documentação. Entretanto, o terceiro objetivo enfatiza a importância da documentação em um nível de detalhe apropriado (que difere de situação para situação). As organizações podem ter que manter documentação sobre os requisitos (p. ex., para fins legais, rastreabilidade ou manutenção). Nesses casos, as abordagens Ágeis têm que garantir que a documentação apropriada tenha sido produzida. No entanto, não precisa ser criada antecipadamente. Pode economizar tempo e esforço criar a documentação em paralelo à implementação, ou mesmo após a implementação. Também pode ser útil criar alguns artefatos como modelos de dados, modelos de atividade ou modelos de estado como documentação temporária para apoiar a discussão sobre os requisitos.

O gerenciamento de requisitos resume todas as atividades a serem realizadas uma vez que você tenha requisitos existentes e artefatos relacionados aos requisitos. Na maioria dos casos, as atividades de gerenciamento de requisitos são incluídas no processo de refinamento constante dos itens em atraso. Mas o gerenciamento clássico de requisitos também inclui a atribuição de requisitos, gerenciamento de versões, gerenciamento de configuração, bem como a rastreabilidade entre os requisitos e a outros artefatos de desenvolvimento.

A RE@Agile sugere minimizar esses esforços ou equilibrar esforços e benefícios:

- O gerenciamento de versões extensivas pode ser substituído por iterações rápidas levando a incrementos de produtos (p. ex., a história de mudanças de requisitos desde que eles foram levantados pela primeira vez é menos importante do que seu estado atual válido);
- O gerenciamento da configuração (baseline) está incluído na determinação iterativa do backlog da sprint, ou seja, agrupando os requisitos que atualmente prometem o maior valor de negócio.

Portanto, algumas das atividades de gerenciamento de requisitos que consomem tempo (e papel) de abordagens não Ágeis são substituídas por atividades baseadas em princípios Ágeis. E algumas outras são bem apoiadas por ferramentas que ajudam a acompanhar automaticamente as relações entre os requisitos e a história sem esforço humano adicional.

Os próximos capítulos deste Handbook discutirão vários aspectos do RE@Agile com mais detalhes.

Capítulo 2 discutirá os pré-requisitos para o desenvolvimento bem-sucedido do sistema: equilíbrio entre visão e/ou objetivos, stakeholders e escopo do sistema.

Capítulos 3 e 4 discutirá o manuseio de requisitos funcionais, requisitos de qualidade e restrições em diferentes níveis de granularidade.

Capítulo 5 discutirá o processo de estimativa, ordenação e priorização dos requisitos para determinar a sequência de incrementos.

Os capítulos 2 até 5 enfatizará principalmente o manuseio de requisitos por um grupo de desenvolvedores (de 3 a 9 pessoas).

Capítulo 6 discutirá o dimensionamento da Engenharia de Requisitos para equipes maiores e potencialmente distribuídas, incluindo planejamento geral de produtos e mapeamento roteiros.

2 Início de um Projeto Limpo

Preparar um workshop antes de iniciar um grande projeto é uma tradição estabelecida em muitos casos. Isso inclui a preparação e coleta do material necessário, limpeza e classificação das ferramentas, remoção de resíduos do projeto anterior e acordo sobre as bases do próximo projeto. Devido à natureza imaterial do software, tal comportamento pode parecer inadequado ou antiquado. Na verdade, o oposto é verdadeiro.

A maior parte do trabalho no desenvolvimento de software é trabalho mental ou pensamento simples. Isto significa que a maior parte do trabalho não é visível externamente em comparação com trabalhos manuais. Em uma oficina ou canteiro de obras, os erros são muitas vezes visíveis para os outros e podem, portanto, ser corrigidos imediatamente. Um erro de pensamento só pode ser notado se a saída do pensamento for visível de alguma forma. A saída pode então ser reconhecida por uma pessoa ou sistema como errada, levando à compreensão ou identificação do erro no pensamento.

As abordagens Ágeis muitas vezes não estão cientes deste problema. As pessoas pensam que a comunicação direta e ciclos rápidos de feedback são suficientes. Embora sejam realmente úteis e valiosas, elas não são suficientes.

> **Por exemplo**: se um grande quadro compartilhado e outros artefatos visíveis estiverem faltando quando o desenvolvimento começar, então a comunicação direta e os ciclos rápidos de feedback não podem impedir múltiplos retrabalhos.

A ideia de um projeto inicial limpo apresentada neste capítulo descreve pré-requisitos importantes que permitem um desenvolvimento iterativo e incremental do sistema.

Você aprenderá que um Projeto Inicial Limpo deve consistir em três atividades que produzem três resultados tangíveis que podem ser usados para conduzir o trabalho iterativo:

- Definição da visão e/ou objetivos do sistema
- Identificação do escopo atualmente conhecido do sistema e do limite do sistema
- Identificação dos stakeholders relevantes e outras fontes de requisitos importantes

Você aprenderá detalhes para cada atividade e suas técnicas correspondentes nos próximos capítulos. No final deste capítulo, apresentaremos o estudo de caso iLearnRE incluindo exercícios para praticar um projeto inicial limpo. O estudo de caso será utilizado como um exemplo contínuo para exercícios adicionais nos capítulos seguintes.

2.1 Visões e objetivos

2.1.1 Fundamentos

A visão do produto e/ou os objetivos do produto são da maior importância para toda atividade de desenvolvimento. Eles estabelecem a direção geral para o desenvolvimento e orientam todas as outras atividades. Visão e/ou objetivos são desencadeados por problemas ou circunstâncias insatisfatórias encontradas no ambiente atual, ou por mudanças no ambiente que nos obrigam a reagir (p. ex., a introdução de uma nova legislação), ou por ideias inovadoras que prometem mais ou melhores negócios.

Utilizamos os dois termos –visão e objetivos– de forma intercambiável. Os métodos Ágeis muitas vezes preferem falar sobre visão enquanto as abordagens de Engenharia de Requisitos geralmente usam a palavra "objetivos". Ambos podem ser considerados como a formulação mais abstrata do que deve ser alcançado pelo sistema. Todos os membros da equipe e todos os stakeholders relevantes devem estar cientes da visão e dos objetivos definidos para entender o que a equipe está lutando.

O Product Owner é responsável pela formulação da visão e/ou objetivos. O Product Owner também é responsável por explicar os detalhes aos membros da equipe. Ser responsável não significa que o Product Owner deva definir sozinho a visão ou as metas. Tipicamente, o Product Owner discute a visão e/ou objetivos com os stakeholders relevantes e coleta suas contribuições e feedback.

Uma armadilha comum ao definir uma visão e/ou objetivos é escolher a perspectiva errada, ou seja, formular uma visão e/ou objetivos que dizem algo sobre o sistema em desenvolvimento ou parte dele. Um exemplo é a seguinte declaração:

> "Criar um website para compra e leitura de livros eletrônicos e livros em áudio."

Esta visão descreve um sistema (um website) para a compra e leitura de livros eletrônicos. Dependendo das circunstâncias, desenvolver tal sistema pode ser uma boa ideia ou não. Entretanto, esta declaração é muito restritiva para se tornar uma boa declaração de visão porque caracteriza o sistema em vez de afirmar o que deve ser alcançado através do desenvolvimento do sistema. A seguinte declaração escolhe uma perspectiva diferente:

> "Vender livros eletrônicos/áudio a pessoas em todos os lugares do mundo (com conexão à Internet) e permitir que elas leiam/ouçam o livro adquirido instantaneamente."

Esta afirmação é melhor comparada com a anterior por várias razões:

1. A declaração define o que o sistema deve alcançar em vez de definir a função do sistema.
2. A declaração enfoca os benefícios do sistema para as pessoas (e para os usuários).

3. Comprar livros eletrônicos/áudio onde quer que eles estejam e ler/ouvir o livro imediatamente.
4. A declaração não predefine o tipo de sistema.
5. Um website pode não ser a solução adequada para a leitura de livros eletrônicos/escutas de livros em áudio.

O maior inconveniente da formulação de visões e objetivos que se concentram no próprio sistema e não no impacto do sistema, é que tais formulações restringem o espaço de solução para a equipe logo desde o início do projeto. Como regra geral, evite qualquer referência ao sistema em desenvolvimento (e à própria palavra "sistema") em uma visão ou declaração de objetivo.

Visões e objetivos são normalmente associados a um horizonte de tempo. Este horizonte de tempo define o período no qual uma visão ou objetivo deve ser alcançado. Portanto, recomendamos que a definição de visões e objetivos tenha sempre um período (ou mesmo uma data específica) anexado a ele. Não é necessário incluir o período na formulação da visão ou objetivo em si, mas o período deve ser claro para todos os membros da equipe e stakeholders.

O Ágil recomenda a definição de visões e/ou objetivos para cada iteração. Portanto, pode haver diferentes declarações para diferentes períodos de tempo. Um sistema ou desenvolvimento de produto poderia ter metas de longo prazo (ou visões estratégicas), por exemplo, para os próximos 3 anos, que por sua vez podem ser divididas em metas a serem alcançadas em anos específicos. E, é claro, no desenvolvimento iterativo também se deve ter objetivos a serem alcançados na próxima iteração.

O benefício de definir visões ou objetivos com uma perspectiva de longo prazo é que os membros da equipe e todos os stakeholders tenham uma compreensão clara do quadro geral e do prazo em que ele será alcançado. Este benefício pode ser ilustrado com o exemplo da livraria apresentado anteriormente. A visão declarada poderia ser subdividida da seguinte forma:

Visão geral "Vender livros eletrônicos e áudio a pessoas em todos os lugares do mundo (com uma conexão à Internet) e permitir que elas leiam/ouçam o livro adquirido instantaneamente."

- Fim do mês 6: Vender livros eletrônicos às pessoas em todos os lugares do mundo e permitir que elas leiam o livro eletrônico imediatamente.
- Fim do ano 1: Vender livros áudio a pessoas em todos os lugares do mundo e permitir que elas ouçam o livro eletrônico imediatamente.
- Fim do ano 2: Vender livros eletrônicos e áudio combinados a pessoas em todos os lugares do mundo e permitir que elas leiam e escutem o livro eletrônico ao mesmo tempo imediatamente.

A visão subdividida apresenta um cronograma claro para o projeto e inclui as informações importantes de que haverá um pacote de livros eletrônicos e de áudio onde o leitor poderá ouvir e ler o texto ao mesmo tempo. Esta informação é muito importante para a equipe, pois eles devem projetar o sistema de leitura de livros eletrônicos de tal forma que seja possível

incluir o livro de áudio mais tarde no processo. Além disso, a equipe agora está apta a dar feedback para responder à pergunta: É realista atingir os três objetivos dentro do prazo definido?

2.1.2 Técnicas para Especificação da Visão/Objetivo

No capítulo anterior, você já viu fundamentos relacionados à definição da visão e/ou objetivos. Neste capítulo, você aprenderá técnicas específicas que podem apoiá-lo no desenvolvimento e definição da visão e/ou objetivos. Qualquer que seja a forma escolhida: todos os stakeholders têm o direito de saber o que a equipe está buscando. Portanto, a definição da visão e dos objetivos iniciais deve ocorrer no início de um esforço de desenvolvimento.

2.1.2.1 SMART.

SMART é um acrônimo e refere-se a um estilo simplificado de escrita de metas e objetivos, proposto em 1981 por George T. Doran [Doran1981].

De acordo com Doran, o acrônimo significa:

- Específico (Specific) – visar uma área específica para melhorias;
- Mensurável (Measureable) – quantificar ou pelo menos sugerir um indicador de progresso;
- Atribuível (Assignable) – especificar quem o fará;
- Realístico (Realistic) – declarar quais resultados podem ser alcançados de forma realista, dados os recursos disponíveis;
- Relacionado ao tempo (Time-related) – especificar quando o(s) resultado(s) pode(m) ser alcançado(s).

Esta definição original foi adaptada pela comunidade Ágil de várias maneiras. Do ponto de vista da Engenharia de Requisitos, a seguinte definição é apropriada:

- Específico (Specific) – visar uma área específica para melhorias;
- Mensurável (Measureable) – quantificar ou pelo menos sugerir um indicador de progresso;
- Atingível (em vez de atribuível) – declarar uma meta que é atingível para a equipe;
- Relevante (ao invés de realístico) – declarar uma meta que seja relevante para os stakeholders;
- Tempo limite – especificar quando o(s) resultado(s) pode(m) ser alcançado(s).

Esta modificação leva em conta duas ideias por trás de um desenvolvimento Ágil:

1. As metas devem se concentrar na capacidade de realização da equipe sem se concentrar em recursos. Os recursos não são planejados; eles são atribuídos por priorização.
2. A relevância da meta, ou seja, o valor que está ligado à meta, é mais importante do que a questão da exequibilidade com relação aos recursos disponíveis.

Para ilustrar a aplicação do SMART, usamos o exemplo dado acima:

> "Vender livros eletrônicos e áudio a pessoas em todos os lugares do mundo (com uma conexão à Internet) e permitir que elas leiam/ouçam o livro adquirido instantaneamente".

Os critérios SMART são satisfeitos por esta declaração da seguinte forma:

Critério	Exemplo
Específico	A experiência de compra e consumo de livros eletrônicos e áudio será melhorada
Mensurável	O resultado mensurável é a compra de livros eletrônicos e de áudio em todos os lugares do mundo (com conexão à Internet) e a leitura/escuta instantânea dos mesmos
Realizável	A Internet e a tecnologia móvel podem fornecer o resultado desejado
Relevante	Os livros eletrônicos e áudio são um meio popular para muitas pessoas
Time-bound	O cronograma é detalhado (ver acima para detalhes)

Os critérios SMART podem ser aplicados como um modelo ou como uma lista de verificação para uma formulação de metas. Na abordagem do modelo, você descreve explicitamente cada elemento dos critérios SMART. A tabela acima é um exemplo desta abordagem. A desvantagem da abordagem do modelo é que ela tipicamente cria redundância na formulação.

Na abordagem da lista de verificação, você usa os critérios SMART para verificar se sua declaração de objetivos cobre todos os aspectos.

Uma boa combinação de ambas as abordagens é a seguinte: Decida-se usando o modelo SMART e depois use o resultado para definir um objetivo preciso que possa ser comunicado facilmente.

2.1.2.2 PAM

O PAM é um conjunto alternativo de critérios para a formulação de metas proposto por [Robertson2003]. Os critérios são definidos da seguinte forma:

- Qual é a finalidade (**P** – purpose)?
- Qual é a vantagem comercial (**A** – advantage)?
- Como podemos medir (**M** – measure) essa vantagem?

Os critérios PAM enfocam o valor de negócio por trás de um objetivo e excluem a especificação de tempo dos critérios SMART. Um benefício de utilizar esta abordagem em um estágio inicial é que ela se concentra nos diferentes valores, em vez de forçar uma especificação de tempo para a definição do objetivo.

Referindo-se novamente ao nosso exemplo acima, os critérios do PAM não estão completamente satisfeitos, como mostra a tabela a seguir:

Critério	Exemplo
Objetivo	A experiência de compra e consumo de livros eletrônicos e áudio será melhorada
Vantagem de negócio	Não indicado explicitamente
Medida	O resultado mensurável é a compra de livros eletrônicos e de áudio em todos os lugares do mundo (com conexão à Internet) e a leitura/escuta instantânea dos mesmos

A vantagem de negócio não é clara em nosso exemplo. Uma vantagem de negócio poderia ser:

- As pessoas compram mais livros eletrônicos ou em áudio quando estão disponíveis instantaneamente;
- As pessoas compram mais livros eletrônicos ou em áudio quando estão viajando, já que os livros estão disponíveis em todo o mundo.

Como os critérios SMART, os critérios PAM podem ser usados como um modelo ou como uma lista de verificação para formulações de metas.

2.1.2.3 Uma visão do produto

A ideia por trás do SMART e do PAM é a definição de critérios explícitos que apoiam a formulação de objetivos. Estes critérios são úteis quando você reúne muitas informações e deseja estruturar estas informações em objetivos adequados e/ou uma visão adequada.

Outra forma de abordar a definição de visões e objetivos é a Product Vision Box introduzida por [Highsmith2001]. A ideia por trás da Product Vision Box é que você crie uma embalagem física para seu produto que mostre os principais benefícios e ideias do produto para clientes potenciais em uma loja.

Um formato comum da Product Vision Box é um workshop de meio dia. Convide os principais stakeholders, se possível de todo o espectro dos envolvidos com seu produto (p. ex., usuários finais, marketing, pessoal técnico).

Você fornece caixas de papelão desmontadas, vários tipos de material (p. ex., papel, lápis, giz de cera, marcadores de cartão, papel alumínio, fios) e material de mídia (p. ex., jornais, revistas, fotos) aos participantes do workshop.

A agenda do workshop deve consistir na alternância das fases de construção e apresentação. Durante a fase de construção, uma equipe de participantes do workshop (3–4 pessoas) cria uma ou mais caixas (pacotes).

Durante a fase de apresentação, as caixas são apresentadas sem qualquer explicação para os participantes. Cada participante pode se decidir sobre cada caixa. Em seguida, os criadores apresentam as ideias por trás da(s) caixa(s) e ocorre uma discussão.

Como opção, as pessoas que não fizeram parte do workshop podem ser convidadas a participar da última fase de apresentação. Desta forma, o feedback externo é fornecido ao grupo, reduzindo o efeito do raciocínio de grupo.

A principal vantagem da Product Vision Box é que as pessoas pensam sobre a ideia do produto a partir do resultado final. Um pacote de produtos normalmente fornece informações sobre as principais funcionalidades ou benefícios mais importantes de um produto. Tal abordagem apoia implicitamente um foco nos objetivos e na visão geral. É uma grande diversão para os participantes, pois cria um resultado tangível que pode ser usado mais tarde como uma espécie de ponto de referência para posterior discussão.

Uma objeção comum contra a Product Vision Box é que existem tipos de produtos que não podem ser vendidos em embalagens simples, mas que podem ser desenvolvidos de forma Ágil. Um exemplo do campo de projetos não-TI: Os projetos de Mudanças Organizacionais têm que lidar com vários domínios problemáticos e necessidades e, portanto, soluções multidimensionais têm que ser criadas, e que não caberão em uma única caixa.

2.1.2.4 Notícias do futuro

Outra técnica para abordar a formulação da visão e dos objetivos é escrever um artigo de jornal sobre seu produto que vem do futuro (ver [HeHe2010]). Esta técnica deriva de técnicas de desenvolvimento da personalidade que motivam as pessoas a pensar em sua vida desde o final, por exemplo, escrevendo seu próprio obituário.

As notícias do futuro podem cobrir vários tópicos e manchetes. Bons pontos de partida podem ser:

- Apresentação de produto bem-sucedida – escreva um artigo a partir da perspectiva de um jornalista que participou de sua apresentação de produto bem-sucedida. Mencione as funcionalidades, impressões ou ideias que este jornalista achou ótimo sobre seu produto;
- Feliz 10º aniversário – imagine que seu produto celebre seu 10º aniversário e que um jornalista escreva sobre este evento em um artigo de jornal. Mencione altos e baixos na história de seu produto e como ele teve um impacto na vida das pessoas ou no negócio em que você se encontra;
- Relatório de falha do produto – imagine que seu produto falhe e que um jornalista informe sobre sua falha. Mencione as razões que levaram ao fracasso e pense em lacunas no seu conhecimento do cliente, funcionalidades ausentes ou outros problemas de qualidade.

O artigo resultante pode ser analisado para identificar a visão e os objetivos potenciais. É também um bom ponto de partida para outras atividades.

> Por exemplo, os critérios SMART ou PAM podem ser usados posteriormente para criar declarações precisas de visão e/ou objetivos.

A técnica da notícia-do-futuro pode ser realizada por indivíduos ou pode ser feita como um exercício em grupo durante um workshop. No exercício de grupo, os participantes devem escrever artigos bastante curtos que possam ser lidos e discutidos durante o workshop.

2.1.2.5 Quadros de Visão

O termo "Vision Board" refere-se a uma classe de técnicas que fornecem representação gráfica estruturada da visão e/ou dos objetivos em um quadro físico. A ideia geral é que:

1. O conselho fornece uma estrutura de conteúdo ou orientada ao tempo para visualizar todo o conjunto de visões e/ou objetivos para os stakeholders;
2. O quadro de visão é considerado como uma entidade viva que é modificada constantemente para representar o entendimento atual de todos os stakeholders;
3. O quadro de visão é o único ponto da verdade para todos os stakeholders quando se trata da visão e/ou objetivos.

Um exemplo muito simples de um Vision Board é composto de três colunas:

- Visão de curto prazo e metas de curto prazo relacionadas: o que queremos alcançar no futuro próximo (p. ex., 4 semanas)?
- Visão a médio prazo e metas a médio prazo relacionadas: o que queremos alcançar a médio prazo (p. ex., 6 meses)?
- Visão de longo prazo e objetivos de longo prazo relacionados: o que queremos alcançar a longo prazo (p. ex., 3 anos)?

Um segundo exemplo, orientado para a estrutura, é o "Product Vision Board", definido por [Pichler2016]. Ela consiste nos seguintes elementos:

- **Visão**: Qual é a sua motivação para criar o produto? Que mudança positiva deve provocar?
- **Grupo alvo**: A que mercado ou segmento de mercado o produto se destina? Quem são os clientes e usuários-alvo?
- **Necessidades**: Qual é o problema que o produto resolve? Que benefício ele proporciona?
- **Produto**: Qual é o produto? O que faz ele sobressair? É viável desenvolver o produto?
- **Objetivos de negócio**: Como a empresa se beneficiará com este produto? Quais são as metas de negócio?

2.1.2.6 Técnicas de Canvas

O termo "Técnica de Canvas" refere-se a um conjunto de técnicas que visam fornecer uma visão geral estruturada de vários aspectos de um produto. As Técnicas de Canvas são próximas às técnicas do Vision Board, mas normalmente têm um escopo mais amplo e não se concentram apenas na visão e/ou objetivos do produto. No entanto, a visão e/ou objetivos são sempre parte de telas e são desenvolvidos em conjunto com os outros aspectos cobertos pela tela.

Devido a este escopo mais amplo, há mais aberturas quando se usam as Técnicas de Canvas que quando se usa uma Vision Board. Portanto, as Técnicas de Canvas requerem mais espaço para documentar todos os aspectos. Portanto, o termo canvas é usado, porque uma tela pode ser muito maior que uma quadro. No entanto, a ideia geral por trás das Técnicas de Canvas é semelhante às Vision Boards.

Um exemplo popular de Técnica de Canvas é a "Business Model Canvas" de [OsPi2010]. A ideia por trás disso é descrever a proposta de valor de uma empresa ou produto, sua infraestrutura, clientes e finanças.

Outro exemplo é a tela Opportunity Canvas de [Patton2014]. Esta tela pressupõe um produto já existente que precisa ser melhorado.

2.1.3 Mudando a visão e/ou as metas

As metas podem mudar durante um esforço de desenvolvimento devido a novos stakeholders ou devido a uma mudança na compreensão do sistema ou do contexto. Portanto, a documentação da visão e/ou objetivos deve ser atualizada regularmente. Técnicas como o Vision Board proporcionam uma representação física da visão e/ou objetivos e permitem uma fácil comunicação dos objetivos alterados.

As mudanças na visão ou nos objetivos devem ser documentadas explicitamente, incluindo as razões para mudá-las. A documentação formal dessas mudanças não é necessária. As formas mais leves de documentar as mudanças são:

- Um diário ou jornal (analógico ou em uma ferramenta) para a visão e/ou objetivos, onde cada mudança é documentada com uma data e a justificativa.
- Uma foto do Vision Board (ou outra representação), incluindo notas que refletem a mudança.

Esta documentação deve ser considerada como a memória comum da visão e/ou objetivos e é útil para refletir sobre as mudanças e reconhecer a frequência das mudanças. Esta frequência é uma métrica importante: mudanças muito frequentes, especialmente em estágios posteriores de desenvolvimento do produto, devem ser consideradas um indicador de que o desenvolvimento geral do produto está em perigo, já que não há uma direção geral clara para o produto.

2.1.4 Especificando o Limite Fundamental do Sistema

O conceito "Limite do Sistema" consiste em um conjunto de termos que permite pensar e documentar com precisão o escopo e o contexto do sistema. Uma compreensão adequada do termo "Limite do Sistema" requer uma compreensão dos termos "Escopo" e "Contexto".

As seguintes definições estão incluídas no glossário IREB:

- **Limite do Sistema**: A fronteira entre um sistema e seu contexto circundante (sistema).
- **Contexto**: A parte do ambiente de um sistema sendo relevante para a compreensão do sistema e de seus requisitos.
- **Escopo**: A gama de coisas que podem ser modificadas e desenhadas ao desenvolver um sistema.

Às vezes o contexto de um sistema deve permanecer inalterado e o Limite do Sistema não é negociável.

Exemplos típicos são:

- Substituição de um componente técnico dentro de um sistema existente maior.
  Por exemplo, um componente de software de uma unidade de controle incorporado em uma linha de produção de automóveis existente deve ser substituído devido a requisitos legais alterados. Os carros já estão em uso e o componente deve caber dentro das interfaces e hardware existentes. A mudança destes aspectos não é possível.
- Desenvolvimento de um sistema dentro de um ecossistema existente.
  Por exemplo, uma companhia de seguros tem um portal web para que os clientes possam gerenciar seus contratos de seguro. A empresa decidiu desenvolver um aplicativo para smartphone como um segundo canal para os clientes. O aplicativo terá a mesma funcionalidade que o portal web. O projeto de desenvolvimento do aplicativo não pode alterar o portal ou as interfaces com outros sistemas.

Em muitas iniciativas de desenvolvimento, entretanto, o escopo e o limite do sistema podem ser negociados. Ou seja, elementos do contexto podem de fato ser modificados durante o esforço de desenvolvimento. Esta afirmação pode parecer abstrata e teórica, mas tem um impacto significativo em todos os esforços de desenvolvimento. Deve ficar claro desde o início quais elementos do contexto do sistema podem ser modificados e quais elementos devem permanecer inalterados.

Uma situação típica é a melhoria de um processo comercial com um novo sistema. Por exemplo, um banco quer substituir uma aplicação baseada em papel para novas contas por uma solução de portal web[2]. No processo existente, o cliente potencial envia o formulário de solicitação em papel para o banco. Um funcionário bancário aprova a solicitação em papel e

[2] *Comentário*: A descrição deste exemplo é incompleta intencionalmente. Descobriremos mais aspectos faltantes nas páginas seguintes para ilustrar o benefício de várias técnicas. Caso você acredite que já tenha detectado algumas coisas faltantes, anote-as e veja se compartilhamos seu ponto de vista.

cria a conta bancária introduzindo os dados no sistema bancário. O novo sistema fornece uma aplicação baseada na web para clientes potenciais: o cliente insere seus dados no formulário e envia os dados ao banco. Imediatamente após o envio dos dados, o cliente recebe a confirmação do pedido via e-mail. Esta é a modificação pretendida no contexto do sistema (clientes potenciais não usam mais um formulário em papel, eles agora usam a aplicação baseada na web).

A parte mais interessante deste exemplo é o processo no back office. Aqui, três cenários poderiam ser possíveis:

1. Os dados da aplicação são enviados por e-mail para o funcionário do banco. O funcionário do banco executa o processo de aprovação existente e insere os dados manualmente no sistema bancário.
2. O processo de aprovação é realizado dentro do novo portal web pelo funcionário do banco. O funcionário do banco verifica os dados da aplicação dentro do portal web. Se o funcionário aceitar a solicitação, então o portal web usa uma nova interface para o sistema bancário para configurar a conta bancária automaticamente.
3. O processo de aprovação é realizado automaticamente pelo novo portal web. O portal web está equipado com um mecanismo de aprovação baseado em regras que permite a aprovação automática de solicitações padrão. Caso o pedido seja aprovado, o portal web estabelece a conta bancária automaticamente. Caso não seja aprovado, o pedido deve ser verificado por um funcionário de um banco.

Este é, naturalmente, um exemplo simplista e incompleto, mas mostra o impacto da decisão do escopo. No primeiro cenário, o escopo é limitado ao portal web, ao novo processo de inscrição e à transferência de dados da inscrição por e-mail para o atendente. No segundo cenário, o escopo também inclui a forma como o funcionário do banco trabalha no back office e a transferência de dados para o sistema bancário, mas a decisão sobre a aplicação permanece com o funcionário. No terceiro cenário, até mesmo o processo de decisão se tornou parte do escopo do projeto.

Qual dos três cenários é apropriado para o banco em questão não está claro a partir de nosso exemplo e depende de vários fatores que devem ser identificados e decididos durante o esforço de desenvolvimento.

Entretanto, o exemplo do banco mostra que um entendimento compartilhado e comum do escopo e do contexto do sistema é um pré-requisito para um esforço de desenvolvimento eficaz e eficiente. Mal-entendidos relacionados aos limites do sistema ou ao escopo podem levar a:

- Desenvolvimento de funcionalidades ou componentes que não estavam sob a responsabilidade do esforço de desenvolvimento. Por exemplo, nosso projeto bancário começou a desenvolver o mecanismo de aprovação baseado em regras (cenário 3), mas os stakeholders nunca concordaram em tal mecanismo de aprovação. Se os stakeholders decidirem que este motor de aprovação não é necessário, então o esforço de desenvolvimento para este motor é perdido.

- A suposição errada de que as funcionalidades ou componentes que são de fato parte do sistema deveriam ter sido desenvolvidos fora do sistema (o escopo suposto era muito pequeno). Por exemplo, nosso projeto bancário de exemplo implementou a transferência por e-mail dos dados da aplicação para o funcionário (cenário 1), mas o stakeholder esperava que a aprovação fosse realizada dentro do portal web (cenário 2).

O sistema e o limite do contexto podem ser definidos através de discussões:

1. **Quais características ou funcionalidades devem ser fornecidas pelo sistema e quais devem ser fornecidas pelo contexto?** Esta pergunta visa os limites do sistema, falando sobre as capacidades concretas do sistema. Por exemplo, em nosso projeto bancário, o sistema pode aprovar uma aplicação automaticamente ou não (cenário 2 ou 3)? Outra discussão poderia ser a criação de uma nova conta bancária: o novo sistema deve ou não realizar esta tarefa (cenário 1 ou 2)?
2. **Quais interfaces técnicas ou de usuário devem ser fornecidas pelo sistema para o contexto?** Esta pergunta visa os limites do sistema e está intimamente relacionada com a questão de funcionalidade acima. Muitas funcionalidades exigem interfaces com os usuários ou outros sistemas. Por exemplo, em nosso projeto bancário, a configuração automática de novas contas bancárias (cenário 2) requer uma interface com o sistema bancário. Além disso, a aprovação pelo funcionário do banco dentro do portal web (cenário 2) requer uma interface de usuário para exibir e aprovar os dados da aplicação.
3. **Quais aspectos do contexto são relevantes / irrelevantes para o sistema?** Esta questão visa o limite do contexto, abordando explicitamente aspectos do contexto que devem ser examinados durante o desenvolvimento do sistema. Por exemplo, em nosso projeto bancário, o formulário de solicitação e o processo de envio dos dados para o banco é definitivamente parte do contexto. Considerando que a configuração da conta bancária pode fazer parte do contexto (cenário 2 e 3) ou pode estar fora do contexto (cenário 1).
4. **Quais aspectos do contexto do sistema podem ser modificados durante o desenvolvimento do sistema?** Esta questão visa o escopo do sistema definindo quais aspectos do contexto podem ser modificados ou não. É importante reconhecer que um elemento do escopo é, por definição, parte do contexto do sistema. Por exemplo, em nosso projeto bancário, poderia ser o caso da decisão de aprovação ter que permanecer com o funcionário do banco (tornando impossível o cenário 3).

Todas as quatro questões estão, naturalmente, intimamente relacionadas e devem ser discutidas em conjunto. Tenha em mente que o Limite do Contexto é sempre incompleto, pois ele só pode ser definido pelas coisas que se exclui explicitamente do Contexto do Sistema.

Da mesma forma, o escopo nunca é definitivo e pode mudar durante um esforço de desenvolvimento. A mensagem importante de uma perspectiva de Engenharia de Requisitos é que as mudanças no Escopo, Limite do Sistema e Limite de Contexto devem ser explicitadas para todos os stakeholders relevantes.

2.1.5 Documentação do Limite do Sistema

O escopo e o limite do sistema podem ser documentados e esclarecidos com várias técnicas. Neste capítulo, apresentaremos quatro destes: diagramas de contexto, linguagem natural, diagramas de caso de uso e mapas de história.

2.1.5.1 Diagrama de contexto

O diagrama de contexto é um elemento da análise essencial do sistema [McPa1984] e utiliza diagramas para representar o contexto. Ele documenta o sistema, aspectos do contexto e seu relacionamento. A notação de um diagrama de contexto consiste em três elementos de modelagem:

- O sistema em consideração (círculo)
- Aspectos do contexto (caixas)
- Setas para representar as conexões entre os elementos. A direção da seta representa o fluxo de informações

A figura a seguir mostra os diagramas de contexto para todos os três cenários do portal de aplicação de contas bancárias.

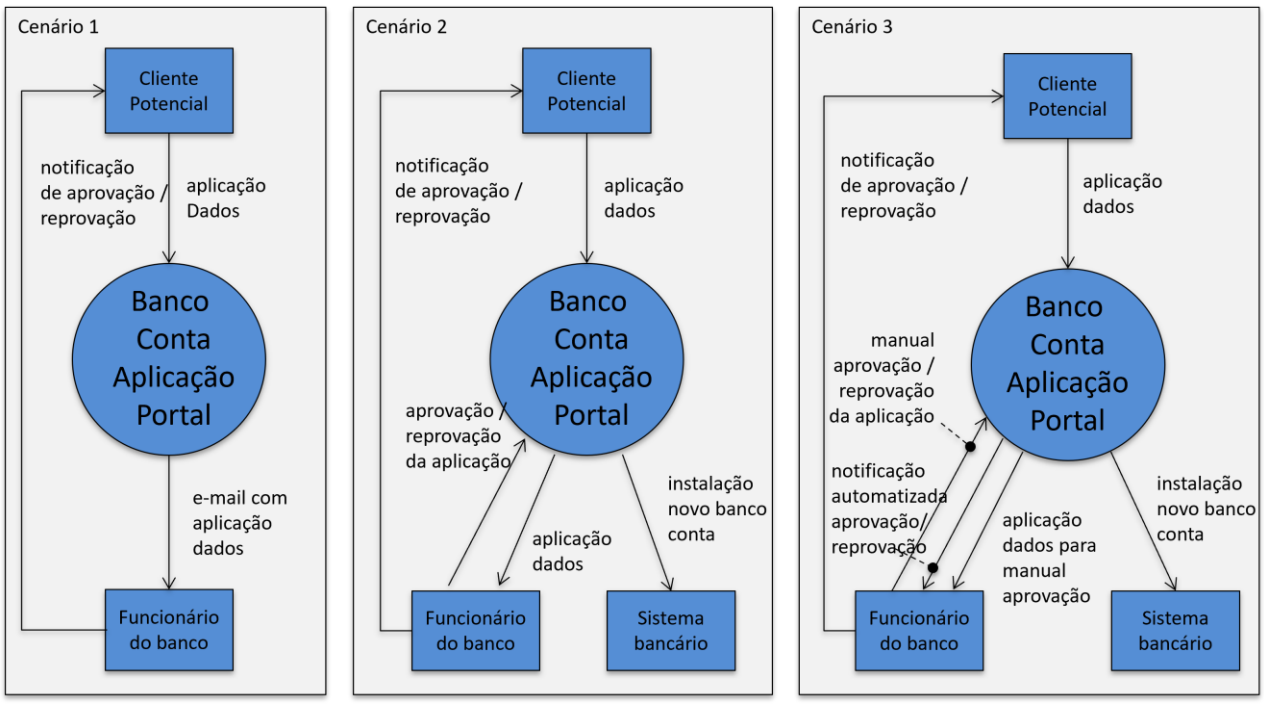


Figura 2: Três diagramas de contexto para o exemplo do portal de aplicação de contas bancárias

Todos os três cenários incluem a relação entre o cliente potencial e o portal (o cliente envia dados da aplicação para o portal) e a relação entre o funcionário do banco e o cliente potencial (o funcionário do banco envia uma notificação para aprovação/recusa da aplicação para o cliente potencial). A documentação da segunda relação (entre o cliente potencial e o funcionário do banco) não faz parte do diagrama de contexto original. Entretanto, é útil documentar esta relação na prática, uma vez que denota claramente que o sistema não é responsável pelo envio desta notificação.

Os diagramas de contexto dos cenários 2 e 3 compartilham a relação com o sistema bancário para configurar a nova conta bancária em caso de aprovação. Esta relação não faz parte do cenário 1, uma vez que o funcionário do banco estabelece a conta manualmente.

Poder-se-ia argumentar que a relação entre o funcionário do banco e o sistema bancário também poderia ser documentada no diagrama de contexto do cenário 1. Há argumentos a favor e contra isso:

- **Para**: A criação da conta bancária é parte do processo comercial geral (criar uma conta bancária) e deve ser documentada para entender o contexto geral.
- **Contra**: A criação da conta bancária foi definida como fora de escopo para o cenário 1. Portanto, ela não deve ser documentada.
- Ambos os argumentos são compreensíveis e válidos. A decisão a favor ou contra a documentação de tais relações depende do contexto geral do projeto.

A principal diferença entre os três cenários é a relação entre o funcionário do banco e o portal. No cenário 2, o funcionário do banco recebe todos os dados de aplicação e deve aprová-los ou recusá-los. No cenário 3, o funcionário do banco só recebe as solicitações que não podem ser decididas automaticamente. Além disso, o diagrama de contexto para o cenário 3 revela um aspecto novo e anteriormente ausente: o funcionário do banco recebe uma notificação para pedidos aprovados/recusados automaticamente. Esta informação é necessária para que o funcionário do banco envie uma notificação ao cliente potencial.

Embora o exemplo do portal seja um exemplo simplista demais, os diagramas de contexto para os três cenários são substancialmente diferentes e fornecem uma visão geral fácil do sistema e do contexto.

2.1.5.2 Documentação em linguagem natural do escopo e do limite do sistema

A Linguagem Natural é a técnica mais flexível e fácil de usar para documentar o Escopo e o Contexto do Sistema. Basta fornecer uma lista de funcionalidades do sistema e uma lista de outros aspectos para documentar o Contexto do Sistema (lembre-se de documentar também aspectos que são considerados fora do sistema). Use uma lista adicional para documentar o escopo do sistema.

A documentação do escopo e do limite do sistema do cenário 1 do projeto bancário poderia ser representada pela seguinte lista.

Escopo e limite do sistema do Portal de aplicação da conta bancária (Cenário 1)	
Funcionalidades do sistema:	▪ Formulário baseado na web para solicitar uma conta bancária ▪ Enviar e-mail para o cliente para confirmar ter recebido o formulário de inscrição ▪ Enviar dados de inscrição via e-mail para o back office do banco
Aspectos dentro do contexto:	▪ Cliente que deseja solicitar uma conta bancária
Aspectos dentro do escopo:	▪ Processo de preenchimento do formulário de solicitação (realizado pelo cliente) ▪ Processo de envio de dados de aplicação para o funcionário do banco
Aspectos fora do contexto:	▪ O funcionário do banco que aprova (ou não) o pedido ▪ Configuração da conta bancária (se o pedido for aprovado) ▪ Enviar aprovação do pedido ao cliente (se o pedido for aprovado) ▪ Enviar recusa do pedido ao cliente (se o pedido não for aprovado)

A comparação desta lista com a descrição do cenário 1 no capítulo 2.2.1 revela um novo aspecto que não foi mencionado anteriormente: A descrição não menciona informações de aprovação ou recusa. Não está claro como o cliente é notificado sobre o pedido aprovado ou recusado.

A lista acima mostra que esta questão não faz parte do contexto, portanto, o esforço de desenvolvimento não precisa se preocupar com este tópico. Sem esta declaração explícita, é muito provável que diferentes stakeholders possam ter expectativas diferentes sobre como a aprovação ou recusa seria tratada com o novo sistema.

2.1.5.3 Diagrama de caso de uso

O Diagrama de Caso de Uso faz parte da UML. É um tipo de diagrama que mapeia os atores e os casos de uso de um sistema. Um caso de uso especifica o comportamento de um sistema da perspectiva de um usuário (ou de outro ator externo ou, por exemplo, de outro sistema): cada caso de uso descreve alguma funcionalidade que o sistema deve fornecer aos atores envolvidos no caso de uso.

O foco dos Diagramas de Caso de Uso nos atores e suas funções relacionadas em um nível detalhado é muito útil para esclarecer o Escopo e o Contexto do Sistema.

Os seguintes elementos de notação de Diagramas de Caso de Uso são úteis para modelar o Contexto do Sistema:

- Limite do sistema (caixa com o nome do sistema)
- Ator (figura humana com nome abaixo ou caixa com nome)
- Caso de uso (oval com o nome do estojo de uso)
- Relação entre Caso de Uso e Ator (linha)

Os diagramas de caso de uso também fornecem elementos de notação para modelar as relações entre casos de uso (p. ex., estende e inclui relações). Os elementos de notação são usados para documentar relações mais detalhadas entre os casos de uso. Este nível de detalhe normalmente não é útil para um esclarecimento inicial do contexto do sistema. A figura a seguir mostra o uso de diagramas de caso para todos os três cenários do portal de aplicação de contas bancárias.

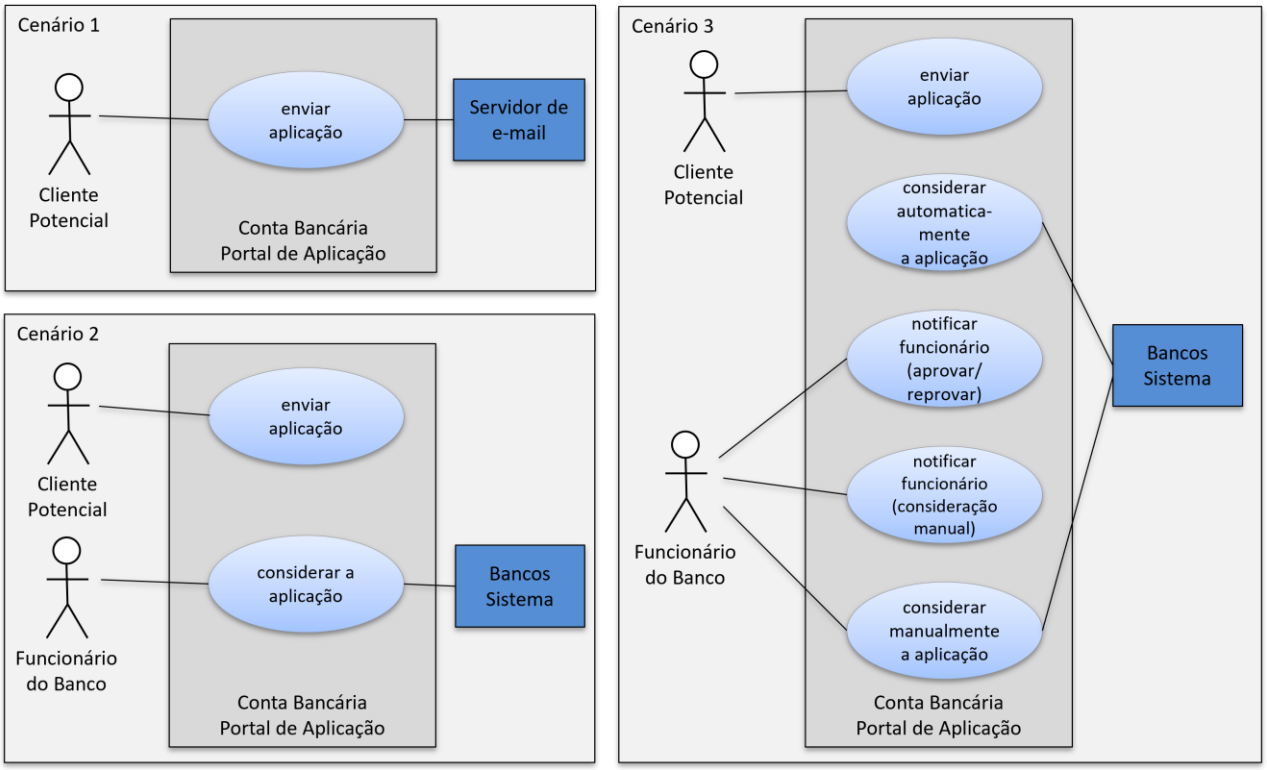


Figura 3: Três diagramas de caso de uso para o exemplo do portal de aplicação de contas bancárias

À primeira vista, os diagramas de caso de uso dão uma visão geral do aumento da complexidade funcional dos três cenários. O cenário 1 é muito simples (um caso de uso), enquanto o cenário 3 é o mais complexo (cinco casos de uso).

As informações centrais do diagrama de uso são levadas pelos nomes dos casos de uso. Portanto, é importante definir cuidadosamente os nomes adequados para os casos de uso. Comparando os diagramas de casos de uso para os cenários 2 e 3, pode-se ver que os casos de uso para "considerar uma aplicação" no cenário 3 são detalhados com os adjetivos "automaticamente" e "manualmente", para esclarecer quem está realizando essa tarefa.

Este esclarecimento não é necessário para o cenário 2, já que o funcionário do banco é responsável por considerar todas as aplicações.

As principais diferenças em termos de limites do sistema entre os três cenários são claramente visíveis:

- No cenário 1, o funcionário do banco não faz parte do contexto do sistema, já que o funcionário não é um ator do portal; o funcionário recebe a aplicação via e-mail.
- Nos cenários 2 e 3, o funcionário do banco faz parte do contexto do sistema, uma vez que o funcionário interage de várias maneiras com o sistema.
- Nos cenários 2 e 3, o sistema bancário é um ator, já que o portal tem que interagir com o sistema bancário para a criação de contas bancárias.

Um aspecto do processo não é mencionado nos diagramas: a notificação do cliente em caso de aprovação ou recusa. Se esta notificação fizer parte do sistema bancário, então os diagramas estão corretos e a notificação está fora do escopo. Mas, se esta for uma notificação que faz parte do portal de aplicação, os diagramas devem ser ampliados para incluir a notificação.

Comparando os diagramas de caso de uso e o diagrama de contexto (ver Figura 2), as principais diferenças entre as duas notações podem ser vistas:

- No diagrama de contexto do cenário 1, o funcionário do banco é documentado, uma vez que o funcionário é um elemento do contexto do sistema que recebe informações (via e-mail) do portal. No diagrama de utilização do cenário 1, o funcionário do banco não está presente, pois o funcionário não é um ator em relação ao portal.
- O diagrama de caso de uso não permite documentar as relações entre atores que estão fora do sistema, mas dentro do contexto do sistema. O diagrama de contexto permite a documentação do fluxo de informações entre os elementos do contexto (a notificação de aprovação/recusa do funcionário para o cliente potencial).
- O diagrama de casos de uso fornece uma decomposição funcional inicial do sistema (os casos de uso). Esta decomposição funcional não é visível no diagrama de contexto.

Essas diferenças têm origem nos elementos de notação de ambos os diagramas e não devem ser entendidas como vantagens ou desvantagens de um diagrama sobre o outro. Se possível, deve-se criar um diagrama de contexto e um diagrama de caso de uso em paralelo para se beneficiar dos pontos fortes de ambos os diagramas. Se for necessário escolher entre o contexto e o diagrama de caso, a seguinte regra geral é útil: se o sistema em consideração estiver inserido em um contexto complexo com várias interações importantes fora do sistema, então seria preferível um diagrama de contexto. Se o sistema em consideração tiver um conjunto complicado de funcionalidades e interações com vários usuários e/ou sistemas relacionados, então seria preferível um diagrama de caso de uso.

2.1.5.4 Mapa de Histórias

Um Mapa de Histórias [Patton2014] é uma técnica para documentar e gerenciar o desenvolvimento de produtos durante todo o processo de seu desenvolvimento. Sua estrutura principal é um arranjo bidimensional de histórias de usuários. A dimensão horizontal concentra-se na espinha dorsal, significando o fluxo narrativo do sistema (ou o processo geral fornecido pelo sistema). A dimensão vertical fornece detalhes para cada parte do fluxo narrativo, assim como uma separação dos itens de acordo com a sequência de desenvolvimento do software.

Assim, o Mapa de Histórias fornece um modelo útil para entender a funcionalidade do sistema e descrever o contexto/escopo em um nível detalhado. Mais detalhes sobre os mapas de histórias serão apresentados no capítulo 3.4.

2.1.6 A Inevitabilidade de uma Mudança de Escopo

A definição de um escopo inicial (incluindo o contexto do sistema) deve ocorrer no início do esforço de desenvolvimento. Sem um escopo claro, a equipe não tem um quadro para seu esforço de desenvolvimento e sem uma compreensão do contexto, a equipe não tem compreensão de onde o sistema estará situado e não tem compreensão de onde procurar informações sobre o que desenvolver.

No entanto, o escopo e o contexto do sistema nunca são finais e estáveis. Na verdade, o único evento que tornaria estável o escopo e o contexto do sistema seria tirar o sistema de operação! Há muitas razões que exigem um ajuste do escopo e/ou do contexto. O cliente pode exigir mudanças e requerer novas funcionalidades; mudanças podem ser necessárias como resultado de legislação nova ou modificada.

A razão mais comum, entretanto, para mudar o escopo e/ou o contexto do sistema, é o entendimento evolutivo dos desenvolvedores e/ou dos stakeholders. Em geral, todo esforço de desenvolvimento constitui uma mudança significativa no contexto do sistema e estas mudanças não podem ser totalmente previstas. O aprendizado de coisas novas é natural em tais situações, e o novo aprendizado muitas vezes tem um impacto no escopo e/ou no contexto do sistema.

Esta situação não é uma desculpa para não ter uma definição adequada do escopo e do contexto do sistema. De uma perspectiva da Engenharia de Requisitos, de fato, é a principal razão para definir sistematicamente o escopo e o contexto. Sem um entendimento inicial adequado do escopo e do contexto atual do sistema, é apenas uma questão de sorte se a necessidade de ajustá-lo mais tarde, será mesmo reconhecida. As técnicas apresentadas neste capítulo são simples e fáceis de usar. A utilização adequada das técnicas requer apenas um pequeno esforço e proporciona enormes benefícios a cada esforço de desenvolvimento.

2.2 Identificação e Gestão de Stakeholders

2.2.1 Fundamentos

De acordo com o glossário do IREB, um Stakeholder é uma pessoa ou organização que tem uma influência (direta ou indireta) sobre os requisitos de um sistema. Além disso, a influência indireta também inclui situações em que uma pessoa ou organização é impactada pelo sistema.

Esta definição enfatiza a importância dos stakeholders, sua a identificação adequada e a sua Gestão durante o esforço de desenvolvimento. A declaração "responder à mudança seguindo um plano" do Manifesto Ágil é frequentemente mal interpretada e usada como desculpa para pular a correta identificação dos stakeholders no início de um esforço de desenvolvimento. A identificação de um novo stakeholder é uma mudança inevitável à qual a equipe deve reagir.

A incapacidade de identificar e incluir um stakeholder importante em um esforço de desenvolvimento pode ter um grande impacto: requisitos importantes podem ser descobertos (demasiado) tarde, ou até mesmo perdidos completamente. Isto pode levar a mudanças caras tardiamente no processo ou mesmo a um sistema inútil. A identificação e gestão dos stakeholders é um investimento importante para minimizar o risco de faltar stakeholders importantes e seus requisitos.

2.2.2 Identificação de Stakeholders

Neste capítulo apresentamos o modelo de cebola como uma técnica simples para a identificação e classificação dos stakeholders. Além disso, discute-se a importância dos usuários como stakeholders centrais, bem como a importância dos stakeholders indiretos.

2.2.2.1 Modelo de cebola para identificação e classificação dos stakeholders

O modelo de cebola de Ian Alexander [Alexander2005] é uma ferramenta simples para a identificação e classificação de stakeholders. O modelo consiste em três tipos de stakeholders (camadas de uma cebola) que podem ser pesquisadas sistematicamente:

- **Stakeholders do sistema**: estes stakeholders são diretamente afetados pelo sistema novo ou modificado. Exemplos típicos desta classe são usuários, pessoal de manutenção e administradores de sistemas.
- **Stakeholders do contexto ao redor**: estes stakeholders são indiretamente afetados pelo sistema novo ou modificado. Exemplos típicos desta classe são os gerentes de usuários, proprietários de projetos, patrocinadores ou proprietários de sistemas conectados (p. ex., sistemas que têm uma interface com o sistema em desenvolvimento, ver capítulo 2.2.4).

- **Stakeholders do contexto mais externo**: estes stakeholders têm uma relação indireta com o sistema novo ou modificado ou com o projeto de desenvolvimento. Exemplos típicos desta classe são legisladores, órgãos de normalização, concorrentes, organizações não governamentais (ONG's), sindicatos, agências de proteção ambiental etc.

Os stakeholders do sistema também são chamados de stakeholders diretos. Os stakeholders do contexto mais externo e mais amplo também são chamados de stakeholders indiretos.

O modelo de cebola pode ser aplicado em vários ambientes para a Identificação de stakeholders:

- Ferramenta de reflexão – use o modelo de cebola para pensar sistematicamente no sistema em desenvolvimento e para anotar todos os stakeholders possíveis que vierem à mente para cada camada.
- Diretriz de entrevista – use o modelo de cebola como diretriz para entrevistas curtas com os stakeholders. Durante a entrevista, pode-se questionar os participantes potenciais dentro de cada camada da cebola.
- Diretriz de workshop – use o modelo de cebola para estruturar um workshop para identificação dos stakeholders. O modelo pode ser usado como uma ferramenta de visualização (p. ex., em um quadro ou flip chart). Cada camada da cebola pode ser analisada com os participantes do workshop. Por exemplo, cada stakeholder escreve os nomes dos stakeholders em um cartão. Alternativamente, cada camada pode ser elaborada durante uma sessão de brainstorming.

Como regra geral, a identificação dos stakeholders deve basear-se em uma ampla gama de fontes. Uma única entrevista com uma pessoa normalmente não é suficiente para identificar os stakeholders mais importantes. Em vez disso, o Product Owner deve planejar várias entrevistas e/ou workshops para identificação dos stakeholders. Se certos nomes são mencionados várias vezes (p. ex., Maria é referida como uma pessoa muito versada em alguns tópicos de negócios), então esta redundância deve ser interpretada como um sinal de importância e não como tempo desperdiçado.

2.2.2.2 Importância do usuário como stakeholder direta

Se um sistema tem usuários humanos, esses usuários estão entre os mais importantes stakeholders diretos. O sucesso de um sistema depende do aceite do sistema por seus usuários. Mesmo que as funcionalidades de um sistema sejam perfeitamente implementadas, então o sistema não vale nada se os usuários não quiserem usar o sistema.

Uma classificação simples com respeito aos stakeholders é a separação entre ambientes abertos e fechados:

- Em um ambiente aberto, os usuários têm alternativas para selecionar. Por exemplo, uma empresa quer desenvolver um novo software de escritório (p. ex., para processamento de texto e apresentações). Existem várias alternativas no mercado para este tipo de produto.

A análise dos stakeholders deve se concentrar em informações que ajudem a convencer os usuários a mudar de seu sistema existente para um novo sistema.

- Em um ambiente fechado, os usuários são "forçados" a usar um novo sistema. Por exemplo, uma empresa desenvolve um novo sistema de administração de negócios para administrar seus negócios e cada funcionário dessa empresa deve usar esse novo sistema porque ele está conectado a cada parte da empresa. Em um ambiente tão fechado, a identificação (e gestão) dos stakeholders pode não receber atenção suficiente, porque os usuários não têm outra escolha senão usar o sistema. Tal comportamento subestima o poder do sistema imunológico corporativo: se os usuários não aceitarem o novo sistema, então o sistema imunológico daquela organização encontrará uma maneira de evitar sua introdução.

Os usuários de um sistema (tanto em ambientes abertos quanto fechados) normalmente cobrem um amplo número de pessoas com expectativas, atitudes e pré-requisitos diferentes. A compreensão deste número de usuários de um sistema é um primeiro passo importante.

Se o número de usuários for pequeno, é aconselhável conhecê-los (ou a seus representantes) através de entrevistas pessoais. Em tais situações, os usuários podem ser questionados diretamente sobre os requisitos.

Se o universo de usuários for grande ou mesmo desconhecido (normalmente em ambientes abertos), o número deve ser capturado por outros meios. Uma ferramenta adequada para tal situação é a Persona Technique [Cooper2004]. Uma Persona representa um exemplo de usuário com características distintas. Tal Persona é normalmente descrita de forma detalhada, incluindo um nome real (por exemplo Jim), uma ou mais fotos, um breve CV e uma lista de hobbies.

O objetivo da descrição é ilustrar a persona da forma mais realista possível e fazer perguntas relacionadas aos requisitos dessa pessoa (p, ex.: Que tipo de função de busca o Jim preferiria?). Uma única pessoa normalmente não é suficiente para um esforço de desenvolvimento. Como regra geral, um projeto deve desenvolver de 3 a 5 pessoas com vários antecedentes. É especialmente aconselhável desenvolver pessoas com posições distintas (p. ex., um novato e um profissional especializado em negócios). Se um novo software for desenvolvido para estes perfis distintos de usuário selecionados, representando os cenários extremos de uso do produto, então a maioria dos usuários principais (p. ex., o usuário médio ou experiente) também aceitará o sistema.

Persona é uma técnica que está embutida no processo de projetar novos softwares. Uma abordagem alternativa, mais orientada para medidas, é a aplicação de análise de dados, análise do Google e Big Data: O comportamento dos usuários online pode muitas vezes ser analisado diretamente pela incorporação de tais tecnologias em incrementos de produtos implantados. O principal benefício de tais técnicas é que elas fornecem dados concretos sobre o comportamento do usuário. A principal desvantagem de tais técnicas é que elas devem ser planejadas em detalhes e implementadas no incremento do software. Portanto, os objetivos de medição de tais técnicas têm que ser claros, uma vez que a coleta dos dados relacionados é cara.

2.2.2.3 A Importâncias dos Stakeholders Indiretos

Os stakeholders indiretos podem ser encontrados no contexto ao redor do sistema e podem ser tão importantes para um esforço de desenvolvimento quanto os próprios usuários. O termo "stakeholder indireto" é por intenção muito amplo, uma vez que os stakeholders indiretos diferem significativamente para diferentes tipos de sistemas. A ideia geral por trás dos stakeholders indiretos é buscar os stakeholders que possam ter impacto no sucesso de um sistema, seja positivo (apoio) ou negativo.

O apoio pode ser fornecido de várias formas. Uma stakeholder pode fornecer informações importantes relacionados ao domínio (p. ex., regras de negócio importantes ou necessidades do usuário) ou sobre desenvolvimentos futuros no domínio (p. ex., um novo tipo de produto, uma nova lei que pode impactar o negócio). Um stakeholder também pode fornecer apoio político durante o desenvolvimento e introdução do sistema (p. ex., um gerente importante de um departamento relacionado).

Um impacto negativo sobre o sucesso de um sistema também pode acontecer de várias maneiras. Um aspecto importante pode ser, por exemplo, a admissão formal de um sistema em ambientes regulamentados (p. ex., sistemas médicos): se os stakeholders relevantes para a admissão de um sistema não estiverem envolvidos no início do processo de desenvolvimento, então o novo sistema pode falhar no cumprimento de critérios importantes de admissão. A dimensão política de um esforço de desenvolvimento é outro aspecto (p. ex., um gerente de um departamento com um produto concorrente pode dificultar o desenvolvimento). O impacto negativo não se limita ao esforço de desenvolvimento. Os tipos subestimados de stakeholders indiretos são ONGs ou pessoas que estão apenas vagamente relacionadas a um sistema. Por exemplo, uma ONG que é ativa no campo da proteção de dados pessoais de saúde pode ter uma forte visão sobre o armazenamento de certos tipos de dados pessoais relacionados à saúde. Se você desenvolver um sistema nesta área, então tal ONG poderá iniciar uma campanha contra seu sistema.

O investimento de tempo na identificação de stakeholders indiretos deve ser considerado como um meio de reunir informações adicionais para o processo de desenvolvimento, a fim de reduzir o risco de fracasso. Como regra geral, um Product Owner responsável pela Engenharia de Requisitos deve desenvolver uma visão ampla sobre os stakeholders indiretos. Falar com os stakeholders indiretos é frequentemente benéfico, mesmo que um deles não forneça novos conhecimentos; a confirmação de informações já conhecidas também é frequentemente benéfica.

2.2.3 Administração de stakeholders

A identificação sistemática dos principais stakeholders deve ocorrer no início de um esforço de desenvolvimento como uma atividade de preparação. A gestão dos stakeholders identificados durante todo o esforço de desenvolvimento é uma atividade contínua. Embora isto pareça muito caro, uma lista simples incluindo detalhes de contato e atributos relevantes (p. ex., áreas de competência ou disponibilidade) será suficiente na maioria dos

contextos. Se o projeto utiliza um wiki para gerenciar a documentação, então a lista de participantes pode ser criada e mantida facilmente no wiki.

A lista pode mudar a qualquer momento, seja porque um stakeholder foi inicialmente negligenciado ou devido a mudanças no contexto, tais como a criação de uma nova ONG. Uma vez que um novo stakeholder tenha sido identificado, ele deve ser abordado para obter os requisitos para o novo sistema e para reunir outras informações valiosas.

Devido à ampla gama de possíveis stakeholders, cada participante em um esforço de desenvolvimento (p. ex., os desenvolvedores e o Product Owner) deve participar da identificação dos stakeholders faltantes. O primeiro passo é criar consciência entre os desenvolvedores sobre a importância dos stakeholders e procurar sinais de novos stakeholders ou ausentes.

2.2.4 Fontes para requisitos além dos stakeholders

Dependendo do sistema e do domínio, a documentação existente, sistemas vizinhos com interfaces com o sistema desenvolvido, sistemas legados ou mesmo sistemas concorrentes também podem ser fontes importantes de requisitos. A lista a seguir fornece alguns exemplos:

- Se o sistema em desenvolvimento tiver um sistema predecessor, a documentação (se houver alguma) e o código fonte deste sistema antigo podem fornecer requisitos importantes (p. ex., requisitos detalhados sobre estruturas de dados);
- Se o sistema em desenvolvimento tiver interfaces com outros sistemas existentes (p. ex., em um contexto empresarial de grande porte), a documentação das interfaces fornece requisitos importantes para a interação entre o sistema em desenvolvimento e esses sistemas. Os usuários, desenvolvedores etc. destes sistemas existentes são, naturalmente, stakeholders importantes;
- Quase todos os sistemas têm um ou mais sistemas similares, ou seja, sistemas que executam tarefas similares em outros contextos. Tais sistemas similares são frequentemente subestimados como fonte de requisitos e ideias. Se você desenvolver, por exemplo, um sistema de compras para um produto altamente especializado, então você deve dar uma olhada nas lojas on-line existentes e suas funcionalidades para ver se elas também podem ser úteis para seus sistemas;
- Se desenvolver um sistema altamente inovador, a pesquisa recente nesta área também poderia ser uma fonte importante para os requisitos. Existem vários bancos de dados na Internet que podem ser pesquisados para material de pesquisa (p. ex., o Google scholar).

Se seu esforço de desenvolvimento pode se beneficiar de fontes adicionais de requisitos, estes devem ser sistematicamente identificados e gerenciados de forma similar à gestão dos stakeholders. Informações detalhadas sobre o gerenciamento de outras fontes de requisitos são fornecidas no módulo Elicitação de IREB.

2.3 Sumário

A definição de Visão e Metas, Stakeholders, Escopo do Sistema e Contexto são interdependentes:

- Os stakeholders relevantes formulam a visão e os objetivos. Portanto, a identificação de um novo stakeholder pode ter um impacto sobre a visão ou os objetivos.
- A visão e os objetivos podem ser usados para orientar a identificação de novos stakeholders, perguntando: Qual pode estar interessado em alcançar a visão e/ou os objetivos ou é afetado por alcançar a visão e/ou os objetivos?
- Visão e objetivos podem ser usados para definir um escopo inicial, perguntando: quais elementos são necessários para alcançar a visão e/ou os objetivos?
- A mudança dos limites do sistema (e, portanto, do escopo) pode ter um impacto sobre a visão e/ou os objetivos. Se aspectos forem removidos do escopo, então o sistema pode não ter mais meios suficientes para alcançar a visão e/ou os objetivos. Por outro lado, se o escopo for ampliado, isto pode fornecer novos meios para cumprir a visão e/ou os objetivos.
- Os stakeholders sugerem o escopo do sistema. Portanto, a identificação de um novo stakeholder relevante pode ter um impacto sobre o escopo. Por exemplo, um importante gerente pode decidir que o escopo do projeto pode ser ampliado.
- Uma mudança do escopo (p. ex., para cumprir um novo objetivo ou modificar um existente) requer o acordo dos stakeholders relevantes.

Estas fortes interdependências significam que é importante equilibrar os três elementos e examinar o impacto da mudança de um dos três elementos sobre os outros. Estar consciente dessas interdependências é o primeiro passo para trabalhar em conjunto na visão e nos objetivos, nos stakeholders e no escopo. Devido a essas interdependências apertadas, recomendamos o manuseio desses elementos em conjunto.

Antes de começar com o desenvolvimento iterativo, recomendamos a criação de uma especificação inicial coerente que inclua:

- visão e/ou objetivos
- escopo e contexto do sistema
- lista inicial de stakeholders (e outras fontes potenciais)

Os métodos e ferramentas apresentados neste capítulo podem ser utilizados de forma leve para criar tal especificação. Um bom e simples ponto de partida é um workshop de meio dia com os três elementos na agenda. Cada participante deve se preparar para o workshop respondendo as seguintes perguntas:

- Qual é a sua visão para o sistema? Quais são os objetivos mais importantes para você?
- Qual é a sua compreensão do contexto e do escopo do sistema?
- Quais stakeholders e outras fontes (documentos, sistemas) têm que ser consideradas para o projeto?

Se os participantes do workshop não estiverem familiarizados com a terminologia, forneça as definições como informação de base para eles. O resultado deste workshop é um ponto de partida para a criação de uma especificação inicial usando os métodos e/ou técnicas descritas neste capítulo.

A especificação inicial deve ser considerada como um documento vivo que deve ser verificado e atualizado regularmente. Os rituais e técnicas de desenvolvimento Ágil proporcionam diversas formas de manutenção leve desta documentação. Uma abordagem pragmática é incluir uma verificação cruzada contra a documentação de contexto/âmbito na definição de pronto. Por exemplo, se o escopo fosse descrito por meio de um diagrama de caso de uso, então cada história de usuário estaria ligada a um caso de uso e a um ator.

2.4 Estudo de Caso e Exercícios

Ao longo deste Handbook, utilizaremos um estudo de caso. Imagine que você quer criar um sistema que permita aos estudantes usar uma plataforma de treinamento via Internet para aprender sobre Engenharia de Requisitos. Pequenas aulas em vídeo devem ser oferecidas juntamente com perguntas para avaliar se um estudante dominou os vários tópicos. A plataforma deve ser utilizável em qualquer dispositivo que permita aos estudantes conectar-se à Internet, ou seja, telefones inteligentes, tablets, laptops, ... Para o gerente de um grupo maior de estudantes, a plataforma deve oferecer informações sobre o progresso de cada estudante. Sugerimos chamar a plataforma de "iLearnRE".

Sugestões para Exercícios:

Se você deseja praticar o Projeto Inicial Limpo, convidamos você a utilizar o estudo de caso iLearnRE. Como uma lista inicial para a visão e/ou objetivos, definimos as seguintes declarações:
- Portal de Treinamento em Vídeo Online para aprender sobre Engenharia de Requisitos e se preparar para o exame IREB
- Disponível em diferentes plataformas, mesmo com conexões de Internet de baixa largura de banda
- Inclui uma sala de bate-papo/fórum de discussão para discutir assuntos com outros estudantes
- Um painel de administração para controlar o progresso dos estudantes em sua equipe

Definimos ainda a seguinte lista de usuários:
- Estudantes
- Administrador do Portal
- Líderes de Equipe (de estudantes)
- Autores de Perguntas

Com estas informações, você pode trabalhar no seguinte exercício:

1. Usar as técnicas do parágrafo 2.1.2 para reformular as declarações de visão/objetivo
2. Criar um diagrama de contexto para o iLearnRE
3. Crie um diagrama de caso de uso para o iLearnRE
4. Pense em outros stakeholders para o iLearnRE

3 Lidando com requisitos funcionais

No capítulo 2 você aprendeu sobre o Projeto Inicial Limpo, por exemplo, sobre pré-requisitos importantes que você deve reunir antes de iniciar o desenvolvimento iterativo e incremental.

Este capítulo trata da elicitação, discussão e captura dos Requisitos Funcionais. As outras duas categorias de requisitos (requisitos de qualidade e restrições) serão discutidas no capítulo 4. Muitas das ideias deste capítulo também são relevantes para estas duas outras categorias.

Neste capítulo, você aprenderá que é bastante normal que os stakeholders falem em diferentes níveis de granularidade o tempo todo. Eles às vezes pedem coisas muito abstratas, onde você como Product Owner terá que trabalhar bastante para descobrir todos os detalhes relevantes. E às vezes eles vão pedir coisas muito pequenas e precisas que já estão próximas do que os desenvolvedores podem entender e implementar. Seu trabalho como Product Owner é lidar com todos esses níveis de granularidade. O alto nível não é ruim se estas funcionalidades não forem necessárias muito em breve. Mas para aqueles que devem ser implementados em uma das próximas iterações, é necessária maior precisão.

No mundo Ágil, os requisitos de granularidade grosseira são frequentemente chamados de épicos, temas ou funcionalidades. Este capítulo mostrará como transformá-los em histórias de usuários do INVEST, ou seja, torná-los suficientemente precisos para que possam ser tratados pelos desenvolvedores.

Assim que você aceita a ideia de que os requisitos existem em diferentes níveis de granularidade, algumas questões surgem naturalmente:

- Como lidamos com múltiplos níveis de granularidade?
- Quais critérios podem e devem ser aplicados para dividir tópicos grandes e abstratos em partes menores?
- É às vezes necessário agrupar pequenos requisitos em partes maiores para que tenhamos um "quadro maior" de orientação?
- Quão precisos temos que ser antes que os desenvolvedores possam começar com a implementação?
- É necessário ou aconselhável manter vários níveis de requisitos, ou podemos jogar fora declarações abstratas assim que tivermos requisitos mais concretos?
- Temos que capturar tudo isso por escrito ou podemos simplesmente falar sobre isso?

Neste capítulo vamos tratar de todas essas questões. Como mencionado anteriormente, vamos nos concentrar nos requisitos funcionais. No capítulo 4 discutiremos os requisitos e restrições de qualidade. No capítulo 5 serão discutidas a Estimativa, a Ordenação e a Priorização de Requisitos. Este capítulo 3 trata exclusivamente de gerenciar requisitos funcionais complexos e refiná-los a um nível tal que possam ser assumidos pelos desenvolvedores.

3.1 Diferentes Níveis de Granularidade de Requisitos

Tomemos alguns exemplos de nosso estudo de caso "iLearnRE". Podemos formular um de nossos objetivos como: "Como estudante eu quero aprender sobre Engenharia de Requisitos em um curso de vídeo online, para não ter que ir a uma sala de aula".

Vamos supor que um de seus stakeholders agora pede a seguinte funcionalidade:

> "Como chefe de departamento, quero poder verificar o progresso do aprendizado de todos os meus funcionários"

Esta não é uma afirmação muito precisa, pois não entendemos necessariamente o que significa "progresso". Além disso, não sabemos como deve ser o resultado desta verificação. Mas é um pedido relevante. Caracterizaríamos isto como um requisito de granularidade grosseira.

Assumir que um dos estudantes apresenta o pedido:

> "Durante a reprodução de um videoclipe quero poder ver o resto de seu tempo de execução em segundos"

Este é um requisito mais preciso que ainda necessita de mais alguns detalhes para a implementação (localização, tamanho, cor da tela de tempo de execução) para a implementação. Estes detalhes podem ser acrescentados pelo Product Owner, o que levará a uma solução do Product Owner, que não é necessariamente a melhor solução. Ou o Product Owner pede à equipe, durante a reunião de refinamento, opções sobre os detalhes e decide com base nas informações disponíveis.

Os stakeholders falam constantemente conosco em todos os níveis de granularidade. Como Product Owner você não pode, e não deve, forçá-lo a ser mais estruturado. Trabalhar com estes diferentes níveis de requisitos e estruturá-los é o seu trabalho como Product Owner, com o apoio daqueles que o ajudam durante o processo de Engenharia de Requisitos.

Como mostra a Figura 4, cada sistema terá requisitos em diferentes níveis de granularidade abaixo da visão e/ou objetivos de nível superior. Como Product Owner, você está se esforçando por dois objetivos:

1. Ter uma visão geral de todos os requisitos funcionais atualmente conhecidos. Isto permite selecionar os mais valiosos para uma implementação precoce, para manter o quadro geral em mente etc.;
2. Compreender os requisitos com detalhes suficientes para que possam ser assumidos pelos desenvolvedores para implementação.

Alguns métodos dão nomes específicos para os níveis de requisitos. SAFe, por exemplo, chama as grandes partes de "épicos", os requisitos de tamanho médio de "funcionalidades" [3] e os requisitos de nível inferior de "histórias de usuários". Outros nomes populares para requisitos mais abstratos são "tópicos" ou "temas".

Não há consenso na comunidade Ágil sobre a terminologia para requisitos mais abstratos. Discutiremos estes termos nos capítulos 3.2 e 3.6.

Durante o processo de levantamento de requisitos e documentação, esta hierarquia de granularidade pode ser estabelecida de diferentes maneiras. Como mencionado anteriormente, os stakeholders normalmente lhe dizem seus desejos em vários níveis. Assim, você pode tentar trabalhar de cima para baixo (de visões e/ou objetivos a requisitos de nível inferior), de baixo para cima (agrupando requisitos de nível inferior em partes maiores), ou de meio para fora (começando com requisitos no meio, dividindo alguns em mais detalhes enquanto outros são agrupados).

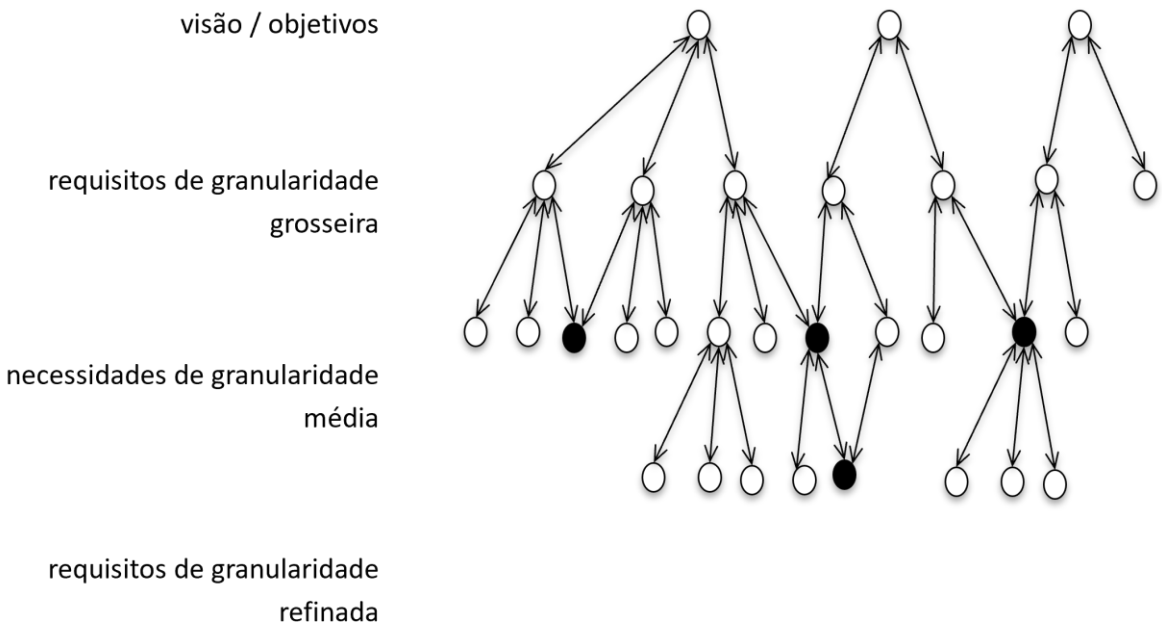


Figura 4: Requisitos de diferentes níveis de granularidade

Como Product Owner, você deve manter as relações (traços ou ligações) entre todos os requisitos. Isto não só lhe dará uma visão geral melhor, mas também lhe permitirá descartar requisitos que não são orientados a objetivos. Assim, é possível evitar o aumento não planejado de requisitos e concentrar-se naqueles que realmente devem ser alcançados.

Observe que alguns requisitos detalhados podem fazer parte de requisitos múltiplos e de nível superior, como indicado pelos pontos negros na Figura 4, por exemplo, uma atividade detalhada pode ser realizada como parte de dois ou mais processos de negócio.

[3] SAFe tem também o nível opcional "Capacidades" entre Épicos e Funcionalidades.

Figura 5 mostra alguns exemplos de requisitos do estudo de caso iLearnRE, incluindo seus links.

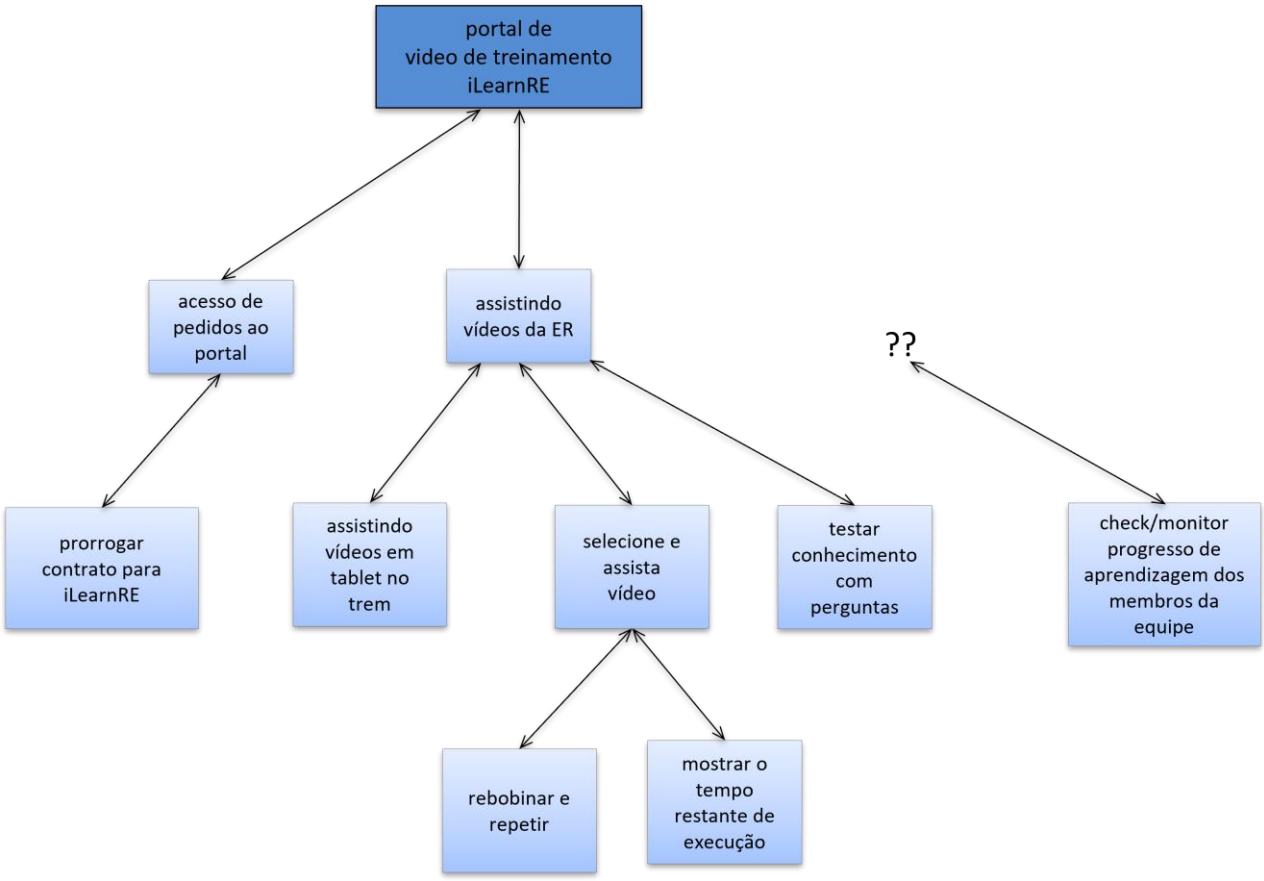


Figura 5: Exemplo de requisitos do estudo de caso

Tal hierarquia estruturada de requisitos permitirá ao Product Owner (e a todas as outros stakeholders) evitar o risco de ser perdido em um projeto maior. Os níveis nesta hierarquia podem ser usados para elaborar estimativas e podem ser usados para priorizar os requisitos. Isto será discutido com mais detalhes no capítulo 5.

Os critérios para agrupar ou dividir os requisitos, notações úteis para capturá-las e ferramentas e técnicas de apoio à visão geral serão discutidos nos próximos capítulos.

3.2 Comunicação e documentação em diferentes níveis

A visão e/ou os objetivos devem ser mais precisos a fim de se chegar a requisitos funcionais que possam ser comunicados e implementados pelos desenvolvedores.

Com base no princípio de "dividir e conquistar", precisamos decompor um sistema ou produto grande em partes menores. Figura 6 ilustra esta abordagem. Discutiremos estratégias e táticas para atingir este objetivo.

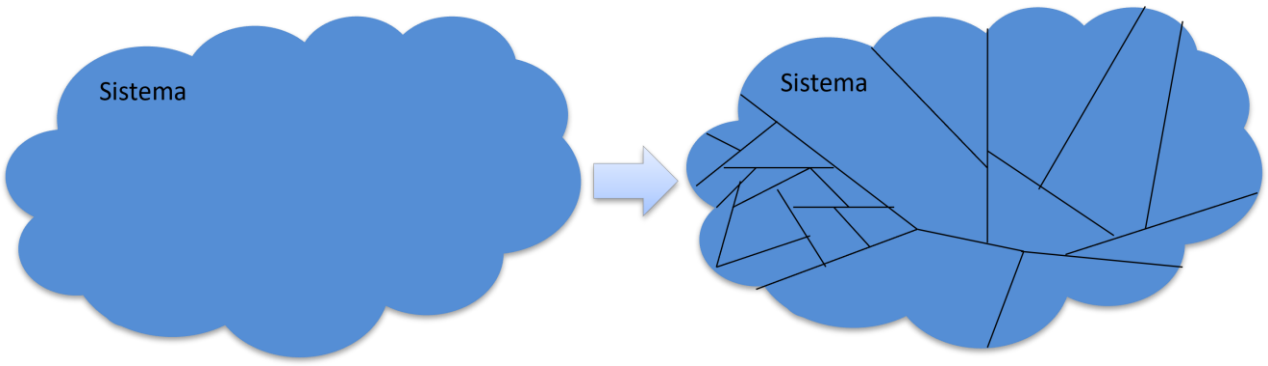


Figura 6: Requisitos funcionais de decomposição

Aqui estão algumas abordagens para a decomposição de um grande sistema (incluindo exemplos do estudo de caso do iLearnRE):

1. **Dividir em funções lógicas (também chamadas de funcionalidades, épicos ou temas):**
   Por exemplo: Estabelecer um contrato para e-learning, assistir a vídeos, testar seus conhecimentos com perguntas ou verificar o progresso do aprendizado
2. **Usar o histórico, por exemplo, a estrutura de um produto existente, como um tema de partição:**
   Como não temos um projeto predecessor de nosso estudo de caso, esta estratégia não funciona aqui.
3. **Dividido por aspectos organizacionais (ou seja, partes que atendem a diferentes departamentos ou grupos de usuários):**
   Por exemplo: Software para estudantes, software de apoio ao líder da equipe, software para os administradores do produto iLearnRE
4. **Dividido de acordo com o hardware:**
   Para exemplos: iLearnRE desktop com design responsivo, aplicativo nativo para iPhone, aplicativo nativo para Android
5. **Dividido por distribuição geográfica:**
   Por exemplo: iLearnRE para um país com o maior número de usuários potenciais, extensão para outros países com legislações diferentes.
6. **Dividido por dados (objetos de negócio):**
   Por exemplo: funções que lidam com vídeos, funções em torno de perguntas, funções em torno de contratos e funções em torno de faturas
7. **Divididos em processos de criação de valor acionados externamente.**

Todas essas abordagens resultarão em pedaços menores que podem então ser analisados separadamente.

As seis primeiras abordagens consideram as estruturas internas do sistema: suas funções, sua estrutura histórica, sua divisão organizacional, seu hardware ou distribuição geográfica ou seus objetos comerciais.

Somente a última abordagem (processos de criação de valor) começa no contexto, fora do escopo de nosso sistema. Ele analisa os gatilhos externos aos quais nosso sistema deve reagir.

Estes gatilhos podem ter diferentes fontes: usuários humanos precisando de algo do sistema, outros sistemas de software enviando entrada e solicitando alguma ação do sistema, dispositivos de hardware (como sensores) acionando uma ação dentro de nosso sistema.

O diagrama de contexto é uma fonte valiosa ao identificar tais gatilhos externos, uma vez que mostra todos os sistemas adjacentes que podem solicitar alguma ação do sistema em consideração.

Esta decomposição orientada a valores tem sido sugerida por muitos autores nas últimas décadas: chamou-a de "decomposição orientada a eventos" [McPa1984], chamou-a de "decomposição de casos de uso" [Jacobson1992], chamou-a de "processos de negócio" [HaCh1993], e finalmente chamou-a de "histórias de usuários" [Cohn2004]. Todas elas sugerem notações diferentes para capturar os resultados desta decomposição. Figura 7 mostra tal decomposição em duas destas notações: casos de uso e histórias de usuários.

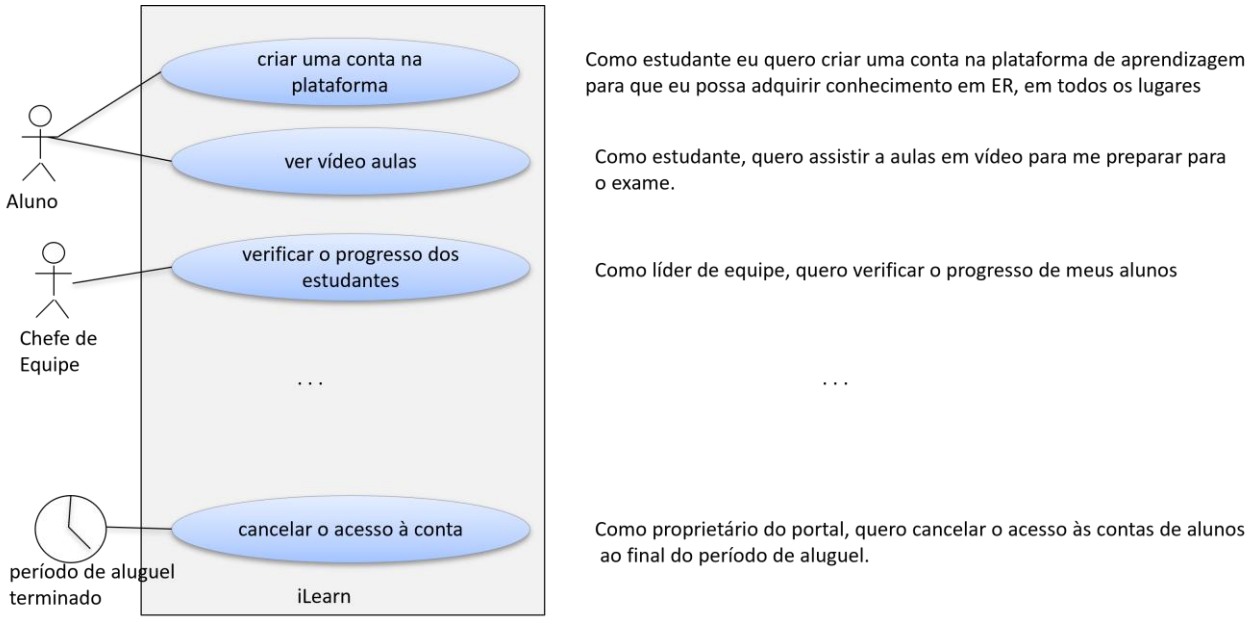


Figura 7: Uma decomposição do sistema orientado a valores em processos em diferentes notações

Ignoremos por um momento as anotações e estudemos as características destas decomposições. Os especialistas ágeis reconhecerão estes critérios como os três primeiros dos critérios do INVEST de Bill Wakes [Wake2003]).

Todos os processos resultantes são:

I: independentes [4] uns dos outros, o que significa que são autocontidos e minimizam as dependências mútuas. Eles não devem se sobrepor em conceito, e gostaríamos de poder programá-los e implementá-los em qualquer ordem.

N: negociáveis, ou seja, ainda não representam um contrato fixo, mas deixam espaço para discussões sobre os detalhes.

V: valiosos: trazem valor real para o requerente, ou seja, para uma pessoa ou outro sistema no contexto.

Os outros critérios do INVEST serão discutidos no próximo capítulo sobre histórias de usuários.

As abordagens para a decomposição, como mencionado anteriormente, também podem ser utilizadas. Especialmente ao escrever requisitos para um sistema existente, sua estrutura atual de componentes ou subsistemas é muitas vezes um bom ponto de partida para elicitar novos requisitos. Existe, entretanto, o perigo de lacunas nas especificações ou sobreposição entre essas partes (ver Figura 8). Uma vez que todas as entradas do backlog serão discutidas e negociadas, você provavelmente pegaria tais lacunas e sobreposições. Mas pensar em termos de processos de criação de valor (com qualquer notação) evita estes perigos logo no início.

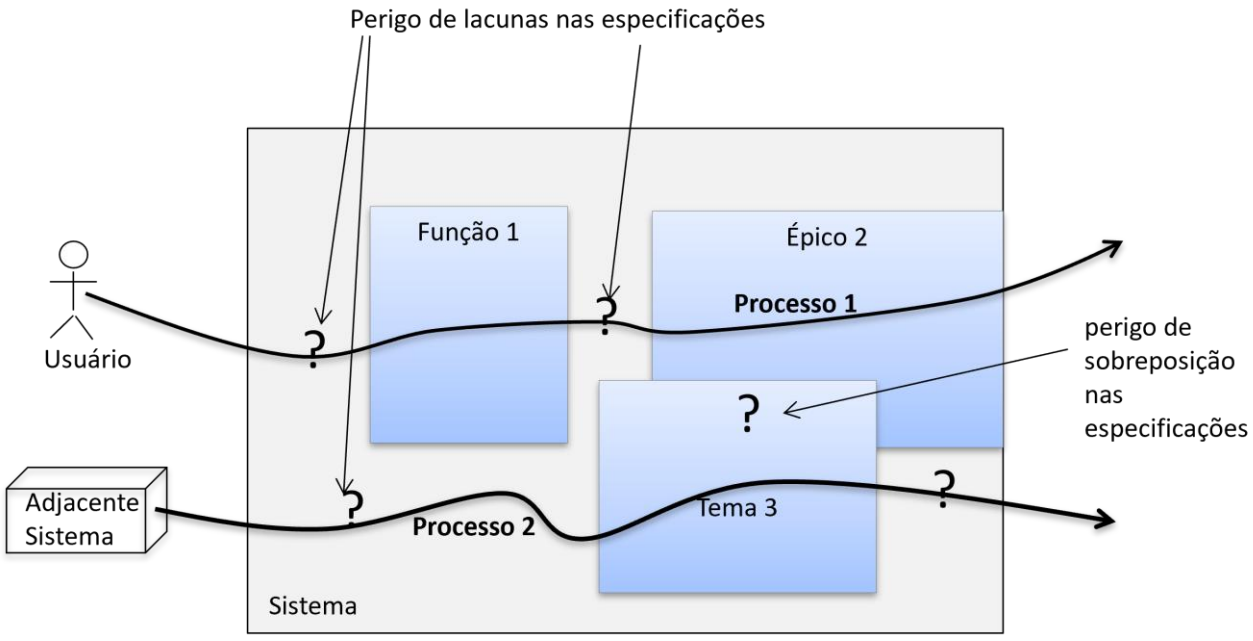


Figura 8: Lacunas e sobreposições de especificações

[4] Outra interpretação da letra "I" é "Imediatamente acionável" [S@S Guide]

Uma sugestão sobre como chegar a uma boa decomposição de processos orientada a valores não é pensar em termos de usuários ou atores do sistema, mas identificar eventos que acontecem no contexto e aos quais o sistema tem que reagir. [McPa1984] identificou dois tipos básicos de eventos:

- **Eventos externos**: Acionados por usuários ou sistemas adjacentes;
- **Eventos temporais**: Acionados pelo tempo ou pela observação dos recursos internos do sistema.

Como Product Owner você pode não identificar a segunda categoria, já que eles não têm nenhum ator ou usuário explícito. O sistema executa um processo pré-definido sem acionamento externo, apenas acionado pelo tempo ou pela observação dos recursos internos.

Exemplos para ambos os tipos a partir de nosso estudo de caso:
– **Evento externo**: "Como estudante, quero avaliar meus conhecimentos com perguntas de teste."
– Evento temporal: "Duas semanas antes do final do período de inscrição, é hora de lembrar aos estudantes sobre uma possível prorrogação."

Vimos agora várias abordagens para encontrar requisitos funcionais para cumprir nossas visões e objetivos. A sugestão é aplicar uma estratégia de decomposição orientada a processos, uma vez que ajuda a identificar partes independentes, negociáveis e valiosas de funcionalidade. Qualquer outra estratégia de decomposição que resulte em tais partes de INV também é boa.

Como Product Owner, você quer obter uma visão geral da funcionalidade do sistema. Naturalmente, seu backlog está sempre aberto para aceitar mais funcionalidade; no entanto, para decisões sobre o roteiro do projeto, para estimativas aproximadas, ou para discutir produtos mínimos viáveis ou mínimos comercializáveis, a visão geral o ajudará. É uma boa base para decidir onde procurar mais detalhes desde o início.

Tendo discutido formas de se chegar a uma decomposição aproximada, concentremo-nos agora na comunicação e documentação destes requisitos funcionais.

A escolha básica é entre desenho e escrita. Você pode visualizar uma decomposição de nível 1 de seus objetivos ou visões, desenhando um diagrama de caso de uso, ou você pode escrever histórias de usuários maiores e colocá-las em cartões separados. Figura 7 mostrou trechos de nosso estudo de caso em ambos os estilos, lado a lado. O capítulo seguinte discutirá as histórias dos usuários com mais detalhes.

Observe que, em princípio, ambas as notações contêm a mesma quantidade de informações e são igualmente detalhadas ou abstratas. É mais ou menos uma questão de gosto pessoal, quer você prefira imagens da visão geral ou histórias escritas.

3.3 Trabalhando com Histórias de Usuários

Para um Product Owner, as histórias de usuários são uma excelente maneira de comunicar os requisitos a todos os stakeholders e também aos desenvolvedores. As histórias de usuários são geralmente capturadas em cartões de histórias, embora uma infinidade de ferramentas esteja disponível para capturá-las eletronicamente. Neste capítulo, vamos nos concentrar na ideia de histórias de usuários.

3.3.1 O modelo 3 C

Como mencionado anteriormente, as histórias são frequentemente escritas em cartões ou notas adesivas e dispostas em paredes ou mesas para facilitar o planejamento e a discussão. Isto muda fortemente o foco da escrita sobre as funcionalidades para a discussão das mesmas. Na verdade, estas discussões podem ser mais importantes do que o texto real escrito no cartão ou na nota.

Ron Jeffries [Jeffries2001] resumiu este aspecto em seu modelo 3C (Cartão, Conversação, Confirmação) para distinguir o caráter mais social das histórias do caráter mais documental de outras notações de requisitos. Suas ideias são explicadas nos capítulos seguintes:

O **"cartão"** (um fichário ou uma nota adesiva) é uma ficha física, dando forma tangível e durável ao que de outra forma seria apenas uma abstração. O cartão não contém todas as informações que compõem o requisito. Em vez disso, o cartão tem apenas texto suficiente para identificar o requisito e para lembrar a todos o que é a história. O cartão é uma ficha que representa o requisito. É utilizado no planejamento. Nele são escritas notas, refletindo, por exemplo, prioridade e custo. É frequentemente entregue aos programadores quando a história está para ser implementada e devolvida ao cliente quando a história estiver completa.

A **"conversa"** ocorre em diferentes momentos e lugares durante um projeto, particularmente quando a história é estimada (geralmente durante o planejamento de lançamento) e novamente na reunião de planejamento de iteração quando a história é agendada para implementação. Envolve várias pessoas preocupadas com uma determinada funcionalidade de um produto de software: clientes, usuários, desenvolvedores, testadores – e é em grande parte uma troca verbal de pensamentos, opiniões e sentimentos.

A conversa pode ser complementada por outros requisitos, artefatos e documentação. Bons suplementos são exemplos; os melhores exemplos são casos de teste executáveis.

A **"confirmação"**: Não importa quanta discussão ou quanta documentação produzimos, não podemos ter tanta certeza quanto precisamos ter sobre o que deve ser feito. O terceiro C nos aspectos-chave da história do usuário fornece a confirmação que temos que ter: os testes de aceite.

A confirmação fornecida pelo teste de aceite nos permite utilizar a abordagem simples do cartão e da conversa. Quando a conversa sobre um cartão se resume aos detalhes do teste de aceite, o Product Owner e os desenvolvedores estabelecem os detalhes finais do que precisa ser feito. Quando a iteração termina e a equipe de implementação demonstra os testes de aceite em execução, o Product Owner aprende que a equipe pode, e irá, entregar o que é necessário.

3.3.2 Um modelo para histórias de usuários:

Mike Cohn define as histórias dos usuários da seguinte forma: (https://www.mountaingoatsoftware.com/agile/user-stories):

As histórias de usuários são descrições curtas e simples de uma funcionalidade contada da perspectiva da pessoa que deseja a nova função, geralmente um usuário ou cliente do sistema. Eles normalmente seguem um modelo simples:

Como um <tipo de usuário> eu quero <algum objetivo> para que <algum motivo>."

Observe os três componentes desta fórmula. Eles garantem isso:

1. temos **alguém** que quer essa funcionalidade ("Como usuário ..."),
2. nós sabemos **o que** o usuário quer ("... eu quero ...") e
3. entendemos **o porquê**, ou seja, a razão ou motivação ("... para que ....").

A fórmula nos ajuda a pensar em quem quer o quê e por quê. Não é tanto o formalismo que faz com que as histórias dos usuários tenham sucesso, é perguntar e responder a estas três perguntas.

Você viu alguns exemplos de histórias de nosso estudo de caso iLearn em Figura 7. Aqui estão alguns exemplos adicionais:

- Como estudante, quero colocar perguntas em um fórum para que outros possam dar respostas ou opiniões.
- Como autor de perguntas, quero acrescentar novas perguntas e respostas ao pool para que os estudantes possam testar seus conhecimentos.
- Como gerente do portal, quero carregar novas versões das perguntas oficiais do IREB para que nosso portal esteja sempre atualizado com o IREB.

Em sua definição, Mike Cohn explicou que as histórias de usuários são contadas a partir da perspectiva da pessoa que deseja a nova função. Note que às vezes a palavra "usuário" é um pouco enganosa, já que a pessoa que deseja uma característica não é necessariamente aquela que trabalha com o sistema como usuário.

Por exemplo, no último exemplo: o administrador que tem que carregar novas versões de perguntas oficiais do IREB não é necessariamente aquele que quer que isso seja feito. É o proprietário do negócio que quer que isto seja feito.

Isto é especialmente verdadeiro para processos que são acionados pelo tempo, o que significa que o processo é executado automaticamente pelo sistema em um determinado momento ou quando alguma condição é preenchida. Um "usuário" não é necessário, mas

tem que haver alguém que se beneficie do processo – caso contrário, executar o processo não faz sentido. Como Product Owner ou Engenheiro de Requisitos você deve sempre procurar por este beneficiário. Pergunte-se: Quem realmente quer esta característica e vê valor em ter a funcionalidade?

Do ponto de vista do negócio, muitas vezes faz sentido falar menos sobre "histórias de usuários", mas simplesmente chamá-las de "histórias" – evitando assim a referência explícita a um usuário. Mas você sempre tem que descobrir quem realmente quer uma história específica. No texto a seguir, usaremos o pequeno formulário "histórias" como um pseudônimo para "histórias de usuários" sempre que apropriado.

Se você quiser evitar esta discussão, basta referir-se a tudo como um item de backlog, de acordo com o guia scrum [S@S Guide].

Aqui está um exemplo de nosso estudo de caso para um processo desencadeado por um evento temporal:

- Duas semanas antes do final do período de inscrição, é hora de lembrar aos estudantes sobre uma possível prorrogação.

Se você quiser escrever esta funcionalidade como uma história, de acordo com o modelo de Mike Cohn, você tem que identificar o proprietário da plataforma como o beneficiário.

- Como proprietário da plataforma, quero que um lembrete automático seja enviado a um estudante duas semanas antes do final do período de inscrição para dar ao estudante a chance de prorrogar o acesso à conta.

3.3.3 INVEST: Critérios para as "boas" histórias

Em 2003 Bill Wake publicou um artigo [Wake2003] aconselhando o INVEST em boas histórias. Já discutimos as três primeiras letras dessa sigla no capítulo 3.1: As histórias devem ser independentes umas das outras, são **n**egociáveis e devem ser **v**aliosas para alguém.

A fim de serem suficientemente bons para serem implementados por um desenvolvedor, elas também têm que cumprir os outros três critérios: **E**stimado, **P**equeno (Small) o suficiente para caber na próxima iteração e **T**estável.

As técnicas de estimativa serão discutidas no capítulo 5.

Se a estimativa mostra que a história ainda é grande demais para ser implementada em uma iteração, ela tem que ser dividida em várias histórias. As técnicas de divisão de histórias são discutidas no capítulo 3.4.

E, finalmente, como mencionado acima no capítulo sobre confirmação, as histórias têm de incluir detalhes suficientes sobre casos de teste ou critérios de aceite (geralmente capturados no verso do cartão). Isto representa um acordo sobre as coisas que os desenvolvedores têm que demonstrar ao Product Owner no final de uma iteração. Ver capítulo 3.5.

3.3.4 Complementando histórias com outros artefatos de requisitos

Como mencionado anteriormente, a história no cartão não contém todas as informações que compõem o requisito. É apenas um sinal físico para fomentar a comunicação entre todos os stakeholders e membros da equipe. Às vezes, é muito útil usar outras notações e artefatos para complementar a história no cartão.

Sinta-se livre para usar diagramas de atividade, BPMN, fluxogramas ou diagramas de fluxo de dados – em resumo: tudo o que você já usou para visualizar um processo de negócio ou etapas de um fluxo.

Exemplo:

Para entender melhor a história "Como estudante eu quero criar uma conta para a plataforma de aprendizagem para que eu possa adquirir conhecimentos de Engenharia de Requisitos em qualquer lugar", você poderia adicionar o seguinte diagrama de atividades:

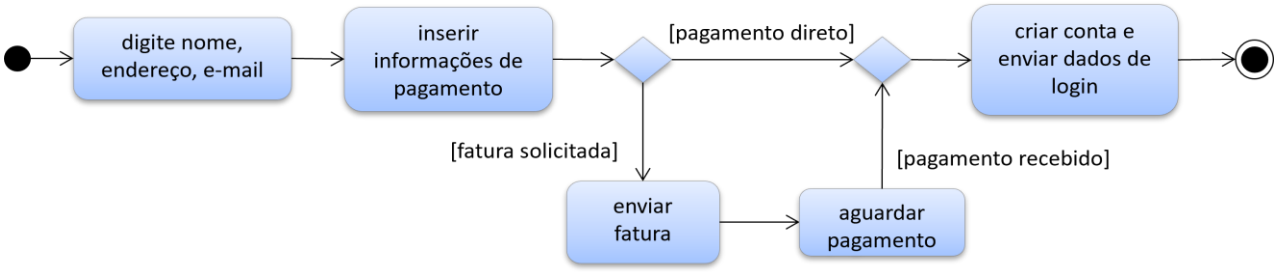


Figura 9: Diagrama de atividades para explicar detalhes de uma história

3.4 Técnicas de Divisão e Agrupamento

A fim de gerar histórias de usuários que sejam pequenas o suficiente para caber em uma única iteração, histórias maiores podem ser divididas em histórias de granularidade refinada. Vários autores sugeriram padrões que podem ser aplicados para este fim, desde a redução da lista de funcionalidades até a redução das variações de negócio ou canais de entrada [Leffingwell2010].

Uma das sugestões mais extensas vem de Lawrence [Lawrence1] e é apresentada na forma de folha de dicas de fácil aprendizagem. Aconselha você a se fazer as seguintes perguntas a fim de conseguir histórias menores:

1. **FLUXO DE TRABALHO**: A história descreve um fluxo de trabalho? Em caso afirmativo, você pode dividir a história de tal forma que você faça o início e o fim do fluxo de trabalho primeiro e que depois melhore com histórias a partir do meio do fluxo de trabalho? Ou, você pode fazer uma pequena parte do fluxo de trabalho primeiro e melhorá-lo com mais histórias depois?
2. **OPERAÇÕES MÚLTIPLAS**: A história inclui operações múltiplas? (p. ex., trata-se de "gerenciar" ou "configurar" alguma coisa? Você pode dividir as operações em histórias separadas?)

3. **VARIAÇÕES DAS REGRAS DE NEGÓCIOS**: A história tem uma variedade de regras de negócios? (p. ex., existe um termo de domínio na história como "datas flexíveis" que sugere várias variações?) Você pode dividir a história de tal forma que você possa fazer um subconjunto das regras primeiro e melhorar com regras adicionais depois?
4. **VARIAÇÃO NOS DADOS**: A história faz a mesma coisa com diferentes tipos de dados? Você pode dividir a história para processar um tipo de dados primeiro e melhorar com os outros tipos de dados depois?
5. **VARIAÇÕES DE INTERFACE**: A história tem uma interface complexa? Existe uma versão simples que você poderia fazer primeiro? A história obtém o mesmo tipo de dados através de múltiplas interfaces? Você pode dividir a história para lidar com dados de uma interface primeiro e melhorá-la com as outras depois?
6. **MAIOR ESFORÇO**: Quando você aplica a divisão óbvia, qual é a história que você faz primeiro, é a mais difícil? Você poderia agrupar as histórias posteriores e adiar a decisão sobre qual história vem primeiro?
7. **SIMPLES/COMPLEXA**: A história tem um núcleo simples que fornece a maior parte do valor de negócio e/ou aprendizado? Você poderia dividir a história para fazer primeiro esse núcleo mais simples e melhorar posteriormente com novas histórias?
8. **DEFERÊNCIA DE PERFORMANCE**: Será que a história tem muito de sua complexidade por satisfazer requisitos de qualidade como performance? Você poderia dividir a história para apenas fazê-la funcionar primeiro e depois melhorá-la para satisfazer os requisitos de qualidade?
9. Último recurso: **QUEBRANDO UMA SPIKE**: Você ainda está perplexo sobre como dividir a história? Você consegue encontrar uma pequena peça que você entenda bem o suficiente para começar? Em caso afirmativo: Escreva essa história primeiro, construa-a e comece de novo no topo das sugestões. Se não, você pode definir a questão de uma a três questões que mais o prendem? Escreva uma spike com essas perguntas, faça o mínimo para respondê-las e comece de novo na parte superior das sugestões.

Note que até mesmo as histórias de usuários com formação específica devem ser definidas de tal forma que tragam algum valor para pelo menos um stakeholder. Portanto, cortar um fluxo de trabalho em suas etapas individuais é muitas vezes contraproducente, já que a implementação de uma ou outra etapa pode não trazer nenhum valor. Portanto, [Hruschka2017] sugere a decomposição de um caso de uso (ou um grande processo) em fatias que vão de ponta a ponta. Isto é baseado na ideia de Ivar Jacobsons sobre fatias de casos de uso [Jacobson2011]. Figura 10 mostra esta ideia em um formato gráfico.

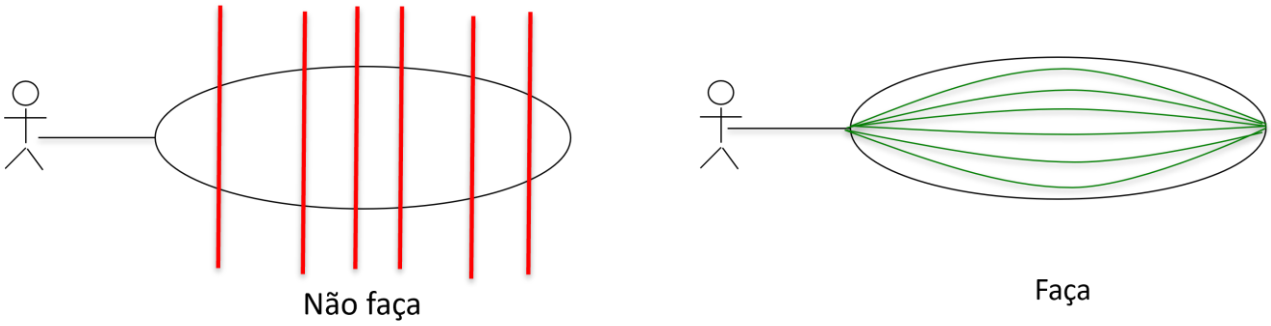


Figura 10: Use fatias de caso em vez de etapas de processo

O fatiamento pode ser feito por diferentes objetos comerciais ou por tecnologia. Então você pode escolher uma das fatias para implementação antecipada e adicionar outras mais tarde. Além disso, você pode encolher uma fatia por:

1. deixar de fora alternativas (p. ex., primeiramente, ir para o fluxo normal, acrescentando casos excepcionais mais tarde),
2. deixar de fora opções (p. ex., deixar de fora coisas que não são absolutamente necessárias para serem implementadas em uma liberação antecipada) ou
3. deixar de fora etapas que ainda podem ser feitas manualmente em lançamentos antecipados.

Se você originalmente inventou histórias que são muito pequenas para criar valor de negócio (especialmente se não forem independentes e sem valor – assim violando partes do princípio INVEST) você deve combinar algumas delas ou reformulá-las para obter boas, mesmo que grandes, histórias iniciais.

Dê uma olhada nas seguintes histórias de nosso estudo de caso:

- Como estudante, quero inserir meu nome e endereço para que eu possa criar uma conta.
- Como estudante, quero adicionar meu endereço de e-mail à minha conta para receber um link para o curso.

Este é um nível muito baixo para histórias valiosas, já que a regra de negócio exige todos estes dados para criar uma conta. É melhor reformular:

- Como estudante, eu quero criar uma conta para ter acesso à plataforma de aprendizagem em vídeo.

A decomposição e o agrupamento de histórias resultarão em hierarquias de requisitos, conforme discutido no capítulo 3.1. Esta hierarquia pode ser visualizada como um mapa de história bidimensional [Patton2014], ver Figura 11. Acima da linha de separação, grupos maiores (como grandes histórias, épicos e funcionalidades) são alinhados de forma a contar a história completa do produto. Isto ajuda a manter uma visão geral dos requisitos; abaixo da linha de separação pode-se anexar todos os detalhes de nível inferior para os grupos maiores e ordená-los para atribuição a sprints e lançamentos como em um backlog linear. Em outras palavras, o mapa da história mostra o backlog por característica ou épico, mantendo intacta a estrutura de nível superior dos requisitos.

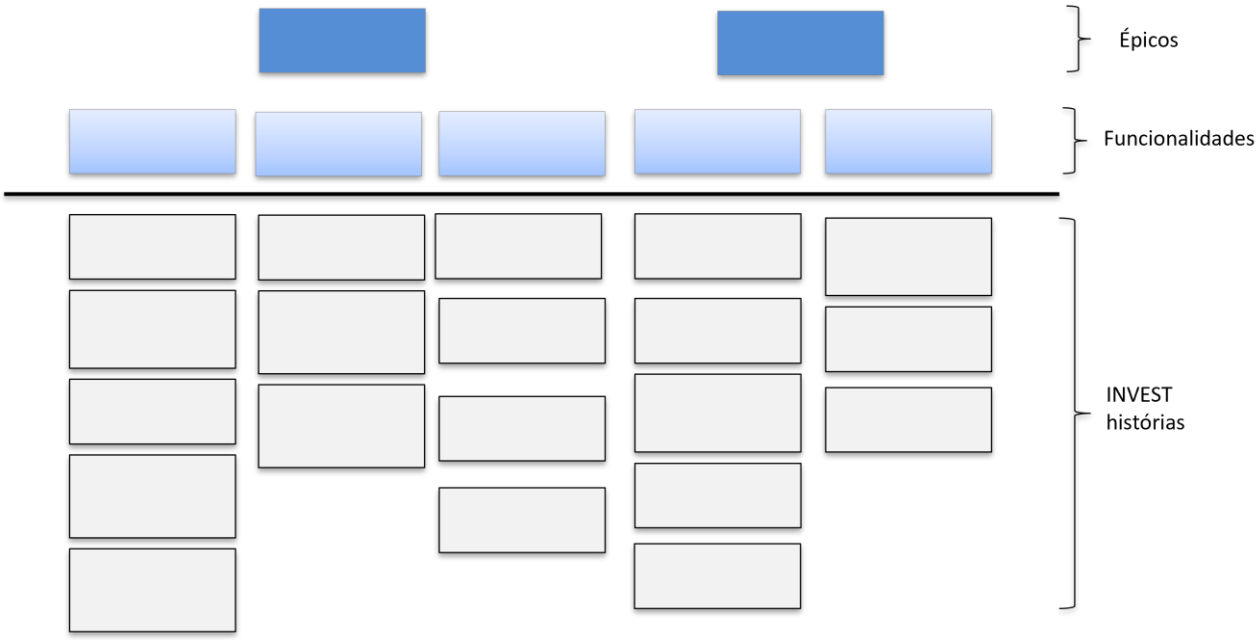


Figura 11: A estrutura de um mapa da história

3.5 Saber quando parar

O Product Owner é responsável por continuar as discussões com os desenvolvedores até que ambos os lados tenham um entendimento comum dos requisitos. [Meyer2014]. O Princípio de Pareto pode ser usado para avaliar quando esse ponto foi atingido: os requisitos não devem ser definidos de forma 100% perfeita, mas suficientemente bem para abordar as principais questões da equipe e suficientemente claros para permitir que o esforço de implementação seja estimado. Iniciar a implementação com muitas questões em aberto pode reduzir consideravelmente a velocidade de desenvolvimento e causar atrasos em relação às previsões.

Para esse nível de entendimento conjunto, o Ágil definiu a definição de preparado (DoR). [AgileAlliance].

Uma história está preparada quando cumpre os critérios do INVEST [Wake2003], especialmente as três últimas letras:

- Os desenvolvedores foram capazes de estimar a história.
- A estimativa é pequena o suficiente para permitir que a história se encaixe em uma iteração.
- Lawrence sugere que a história não deve caber apenas em uma iteração, mas deve ser tão pequena que 6 a 10 histórias possam ser atribuídas à próxima iteração [Lawrence1]. Para conseguir isso, o Product Owner tem que estar ciente da velocidade da equipe. (Para mais detalhes sobre a velocidade consulte o capítulo 5.) Se, por exemplo, a equipe pode lidar com 28 pontos de história por sprint, então as histórias de usuários devem ser tão pequenas que a soma de 6 a 10 histórias não exceda esse valor. O backlog da sprint deve ser composto, por exemplo, de 8 histórias com 1, 1, 2, 3, 5, 8 e 8 pontos de história e um objetivo claro de sprint deve ser formulado caso a equipe não consiga terminar todas as histórias.

- O Product Owner forneceu critérios de aceite para a história. Com base no princípio 3C, todos concordam que houve conversas suficientes e que os critérios para confirmação do sucesso em termos de testes de aceite foram definidos. Se alguém usa cartões para capturar as histórias, os testes de aceite são normalmente escritos no verso do cartão.

Os Product Owners têm uma escolha no caso de uma história que já é pequena o suficiente para caber em um sprint: eles podem manter essa história e adicionar mais testes de aceite ao cartão. Ou eles podem optar por dividir a história em várias histórias, geralmente tendo menos e mais testes de aceite simples para cada uma delas.

Há diferentes estilos disponíveis [Beck2002] ao formular critérios de aceite. Podem ser sentenças informais em linguagem natural a serem verificadas após a implementação.

Os critérios de aceite poderiam ser um pouco mais formais utilizando a sintaxe Gherkin [WyHT2017]. Gherkin é uma linguagem específica de domínio e legível do ponto de vista de negócio, criada especialmente para descrever um comportamento. Ele permite que você remova detalhes lógicos dos testes de comportamento.

Gherkin sugere a seguinte estrutura para escrever cenários de teste:

- **Cenário** <<nome descritivo curto>
- **Dada** <<algumas pré-condições>>
- **E** <<algumas outras condições prévias>>
- **Quando** <<algumas ações do usuário>>
- **E** <<algumas outras ações>>
- **Então** <<algum resultado testável é alcançado>>
- **E** <<somente podemos verificar que algo mais acontece também>>>

Alguns métodos defendem inclusive o uso de TDD (Test Driven Development). Em vez de usar uma Linguagem Específica de Domínio (DSL) como Gherkin, você pode codificar formalmente os casos de teste para que eles possam ser executados automaticamente após a implementação [Meyer2014]. Essa abordagem formal, embora muito precisa, pode ser de difícil entendimento para Product Owners e stakeholders voltados para os negócios.

Para o Product Owner, a DoR é equivalente à definição de pronto (DoD) dos desenvolvedores. O DoD define critérios para determinar se uma história foi implementada com sucesso, enquanto a DoR define que os desenvolvedores têm informações suficientes sobre uma história de usuário para que ela possa ser "concluída" pelos desenvolvedores em uma iteração.

Discutir os requisitos com os desenvolvedores requer tempo e é melhor fazê-lo antes do planejamento da iteração. O planejamento pode então se concentrar na seleção das histórias corretas dos usuários e atribuí-las aos desenvolvedores responsáveis. O ideal é que os desenvolvedores tenham visto a evolução dos requisitos e ajudado o Product Owner fazendo perguntas e estimativas.

São possíveis diferentes formas de refinamentos. As reuniões de refinamento podem, por exemplo, ser uma forma mais eficiente de realizar refinamentos do que perturbar repetidamente os desenvolvedores de forma individual.

O refinamento do backlog do produto e todas as atividades ao redor consomem tempo a partir da capacidade geral da iteração.

O guia Scrum [S@S Guide] recomenda um máximo de 10% de capacidade dos desenvolvedores para refinamento: se for necessário mais tempo do que esse, isso é um sinal de alerta para a má qualidade dos requisitos. O Product Owner deve entender a relação entre a duração da iteração, o risco e a sobrecarga da iteração, e saber que há ciclos de feedback mais curtos do que a própria iteração.

3.6 Documentação do Projeto e do Produto de Requisitos

Projetos ágeis e desenvolvimentos de produtos, especialmente os Scrum, utilizam um backlog de produto, que é uma lista prioritária das funcionalidades a serem desenvolvidas em um produto ou serviço. Embora o backlog de produto possa ser o que a equipe desejar, épicos, funcionalidades e histórias de usuários têm surgido como as formas mais populares de backlog de produtos.

Enquanto um backlog de produto pode ser pensado como um substituto para o documento de requisitos de um projeto tradicional, é importante lembrar que a parte escrita de uma história de usuário Ágil ("Como usuário, eu quero ...") está incompleta até que as discussões sobre essa história tenham acontecido.

Muitas vezes é melhor pensar na parte escrita como um ponteiro para uma representação mais precisa deste requisito. As histórias de usuários poderiam apontar para um diagrama representando um fluxo de trabalho, uma planilha de cálculo ou qualquer outro artefato que o Product Owner ou a equipe deseje.

No RE@Agile Primer [Primer2017] identificamos quatro propósitos diferentes para a documentação dos requisitos.

Consideremos os dois primeiros propósitos:

1. **Documentação para fins de comunicação**: A comunicação eficaz e eficiente é uma ferramenta importante nos métodos ágeis devido à sua interatividade e aos ciclos curtos de feedback. Na prática, existem várias situações que podem dificultar a comunicação verbal direta: equipes distribuídas, barreiras linguísticas ou restrições de tempo das pessoas envolvidas. Além disso, a informação é por vezes tão complexa que a comunicação direta pode ser ineficiente ou enganadora. Um protótipo em papel ou um diagrama de um algoritmo complicado pode, por exemplo, ser relido mais tarde. Às vezes os stakeholders simplesmente preferem a comunicação escrita à leitura do código fonte ou à revisão do software. Nesses casos, a documentação facilita o processo de comunicação entre todas as partes envolvidas e os resultados do processo são armazenados.

O princípio da criação de documentação para fins de comunicação é: um documento é criado como um meio de comunicação adicional se os stakeholders ou os desenvolvedores virem valor na existência do documento. O documento deve ser arquivado quando a comunicação tiver sido bem sucedida.

2. **Documentação para fins de reflexão**: Um aspecto frequentemente esquecido da redação de um documento é que ela é sempre um meio de aprimorar e apoiar os processos de pensamento do redator. Mesmo que o documento seja jogado fora mais tarde no processo, o benefício de melhorar e apoiar o pensamento é duradouro. Por exemplo, escrever um caso de uso força o escritor a pensar em interações concretas entre o sistema e os atores, incluindo, por exemplo, exceções e cenários alternativos.

   Escrever um caso de uso pode, portanto, ser entendido como uma ferramenta para testar o seu próprio conhecimento e compreensão de um sistema.

O princípio para a criação de documentação para fins de pensamento é: o pensador decide sobre a forma de documento que melhor apoia seu pensamento. O pensador não precisa justificar esta decisão. O documento pode ser descartado quando o processo de reflexão estiver terminado.

Para os dois primeiros propósitos, um backlog de produto com épicos e histórias (em qualquer forma (cartões na parede ou histórias capturadas em ferramentas) e talvez aumentadas com esboços, diagramas e protótipos) é suficiente como documentação para apoiar o progresso do desenvolvimento do produto.

Para as duas outras finalidades, deve ser considerada mais documentação de requisitos formais.

3. **Documentação para fins legais**: Determinados domínios ou contextos de projeto (p. ex., software no setor de saúde ou aviônicos) exigem a documentação de determinadas informações (p. ex., requisitos e casos de teste de um sistema) para obter aprovação legal.

O princípio da criação de documentação para fins legais é: as leis e normas aplicáveis descrevem qual documentação legalmente necessária tem que ser criada. Esta documentação é uma parte inseparável do produto.

4. **Documentação para fins de preservação**: Certas informações sobre um sistema têm um valor duradouro para além do esforço inicial de desenvolvimento. Os exemplos incluem as metas do sistema, os casos de uso centrais que ele suporta ou as decisões que foram tomadas durante seu desenvolvimento, por exemplo, para excluir determinadas funcionalidades. A documentação para fins de preservação pode se tornar o arquivo compartilhado da equipe, de um produto ou de uma organização. Ela pode reduzir a dependência da capacidade de memória dos membros individuais da equipe e pode ajudar nas discussões sobre decisões anteriores (p. ex., "Por que decidimos não implementar isso?").

O princípio da criação de documentação para fins de preservação é: a equipe decide sobre o que documentar para fins de preservação.

Para estes dois fins, o backlog de produto – que é uma ferramenta para a interação de um Product Owner com os desenvolvedores – não é suficiente.

A boa notícia é que a documentação para fins legais ou para a preservação do know-how exigido pelo produto não precisa ser criada com antecedência.

Ela pode ser atualizado e mantido sempre que uma nova versão do produto é lançada, por exemplo, após a implementação bem-sucedida das funcionalidades. Assim, ele contém apenas documentação da funcionalidade e qualidades que realmente o transformaram no produto – evitando atividades de gerenciamento de versões e configurações demoradas em documentos enquanto os stakeholders ainda estão negociando e talvez mudando suas opiniões.

A definição de um grau adequado de documentação depende de muitos fatores, como o tamanho dos projetos, o número de participantes envolvidos, as restrições legais e/ou os aspectos críticos de segurança do produto. Com base nestes fatores, as equipes em um ambiente Ágil tentam evitar o excesso de documentação e encontrar um conjunto mínimo útil.

Apesar de trabalhar com um backlog de produto "vivo" ser uma forma eficiente de lidar com a documentação, nem sempre ela é suficiente. Uma documentação estruturada e atualizada de todos os requisitos implementados em um produto pode não só ser uma restrição legal em alguns projetos, mas também um ponto de partida perfeito para uma identificação mais rápida dos pedidos de mudança com base na documentação existente.

3.7 Sumário

O que quer que seus stakeholders digam sobre a funcionalidade necessária é o ponto de partida certo para o trabalho de requisitos. Mas é apenas o ponto de partida. Seu trabalho como Product Owner é trazer estrutura para estes requisitos funcionais.

Épicos, temas, funcionalidades ou grandes histórias (representando processos de negócio complexos) são uma boa maneira de manter um quadro ou uma visão geral de todas as coisas que seus stakeholders querem de um sistema ou produto. Mas você aprendeu que – por definição – eles podem não ser precisos o suficiente para parar nesse nível.

Seu objetivo para um bom trabalho de requisitos é criar histórias de usuários, que satisfaçam a definição de preparado, ou os critérios do INVEST: devem ser independentes e valiosas, pequenas o suficiente para caber em uma iteração, estimáveis e equipadas com critérios de ajuste verificáveis. O modelo de Mike Cohn "Como <usuário> quero <algumas funcionalidades> para atingir <algumas metas>" é um bom ponto de partida, mas você não deve insistir em usar esta fórmula em todos os casos.

Se um requisito ainda é muito grande para caber em uma iteração, você aprendeu várias táticas para dividi-las, enquanto você ainda tenta preservar a independência e valorizar o máximo possível.

4 Lidando com requisitos e restrições de qualidade

O capítulo 3 enfocou o manuseio dos Requisitos Funcionais. Lidar com os requisitos funcionais, ou seja, descobrir qual funcionalidade os diversos stakeholders precisam, será a atividade mais demorada no desenvolvimento do sistema e dominará a maioria das discussões entre o Product Owner, os stakeholders e os desenvolvedores.

Qualidades (das funções) do sistema, como desempenho, facilidade de uso, robustez e extensibilidade são muitas vezes tidas como garantidas. Os usuários e/ou outros stakeholders frequentemente assumem que eles não precisam ser declarados explicitamente, uma vez que os desenvolvedores já sabem sobre eles.

O mesmo é válido para as restrições organizacionais e técnicas. Não sabem todos que temos um modelo de processo padrão, que exige que certos artefatos sejam produzidos? Não estão todos conscientes de que sempre usamos a empresa X para comprar nossos sistemas de banco de dados e, naturalmente, codificamos no idioma Y?

Especialistas em engenharia de requisitos têm afirmado a importância desses requisitos "não-funcionais" por décadas. Embora o termo "requisitos não funcionais" ainda seja usado com frequência na prática, como um termo abrangente para requisitos e restrições de qualidade, o IREB usa as categorias mais concretas e precisas "Requisitos de Qualidade" e "Restrições", de acordo com [Glinz2014].

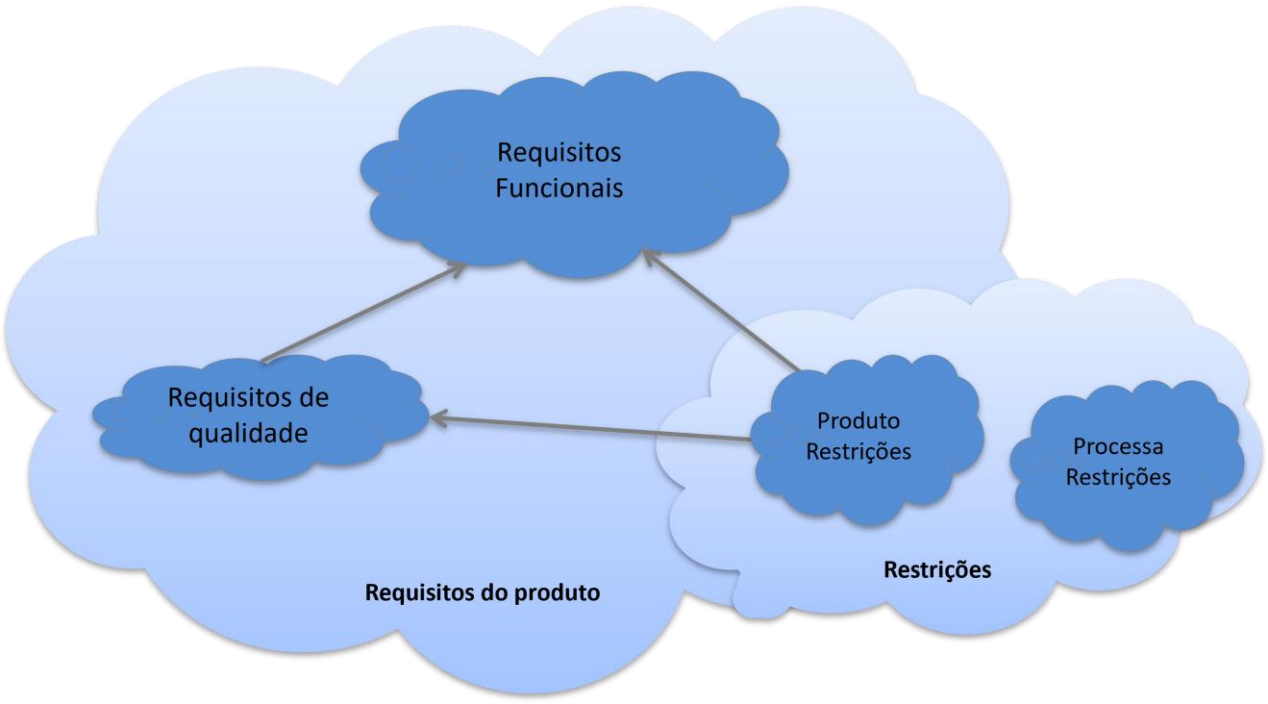


Figura 12: Categorização de requisitos

Figura 12 mostra as três categorias de requisitos e algumas de suas importantes relações. Um requisito de qualidade nunca será autônomo, o que significa que sempre se referirá a um ou mais – ou mesmo a todos – requisitos funcionais.

As Restrições são restrições do produto, limitam o projeto de uma função ou qualidade, ou restrições de processo, limitando o trabalho dos desenvolvedores de uma forma que não está diretamente ligada ao produto em si, por exemplo, certas etapas do processo devem ser executadas ou certos artefatos devem ser criados.

Inicialmente, os requisitos e as restrições de qualidade costumam ser deliberadamente vagos. Nos próximos capítulos, descreveremos como capturar tais qualidades e restrições vagas. Você também verá como transformar requisitos e restrições vagos de qualidade em requisitos mais precisos (até o nível de especificação de critérios de aceite precisos) e como lidar com elas em conjunto com requisitos funcionais.

4.1 Entendendo a importância dos Requisitos de Qualidade e Restrições

[Meyer2014] expressa a preocupação de que "muitos métodos ágeis se concentram apenas nos requisitos funcionais e não dão ênfase suficiente às qualidades e restrições". Bertrand Meyer continua dizendo: "As principais restrições e algumas categorias de qualidades previstas para o sistema devem ser explicitadas no início do ciclo de vida de um produto, pois determinam as principais escolhas estruturais (infraestrutura, arquitetura de software e design de software). Ignorá-los ou aprender muito tarde no projeto pode colocar em perigo todo o esforço de desenvolvimento. Outras qualidades podem ser capturadas iterativamente, na hora certa, como acontece com os requisitos funcionais."

Embora haja muitas categorias de requisitos de qualidade a serem consideradas, a tarefa é facilitada para os Product Owners por uma série de esquemas de categorização publicados – ou listas de verificação – tais como os mostrados nos dois exemplos a seguir. Como Product Owner você deve simplesmente usar um destes "lembretes" para fazer perguntas explícitas sobre estas qualidades. Melhor ainda: com base nas listas de verificação disponíveis, você pode criar sua própria para enfatizar as qualidades que são mais importantes em seu domínio.

Em 2011 a ISO publicou uma nova família de padrões de qualidade, substituindo o conhecido modelo de qualidade ISO/IEC 9126 de 2001. A norma mais importante para a Engenharia de Requisitos é a [ISO25010], que define os requisitos de qualidade. Sua última atualização é a partir de 2017. Figura 13 mostra as oito características de qualidade de nível superior dos sistemas e sua decomposição em sub características. Note que a norma não fala de requisitos, mas das qualidades do sistema. Adicionar a palavra "requisitos" a cada categoria permite discutir suas necessidades nesta área, por exemplo, "capacidade" se torna "requisitos de capacidade".

Figura 13: Categorias de qualidades de acordo com ISO25010

As definições detalhadas de todas essas categorias podem ser encontradas na norma. Além do modelo genérico de qualidade, a norma ISO/IEC 25012 contém um modelo complementar para a qualidade dos dados.

Um esquema similar de categorização para requisitos de qualidade pode ser encontrado no modelo VOLERE [RoRo2017]. Os capítulos 10 – 17 deste modelo descrevem as categorias de requisitos de qualidade. A categorização é baseada em décadas de experiência na especificação de sistemas. O modelo original acrescenta a palavra "requisitos" a cada categoria, ou seja, "longevidade" lê-se "requisitos de longevidade". Em Figura 14 pulamos esta adição para manter as categorias mais legíveis.

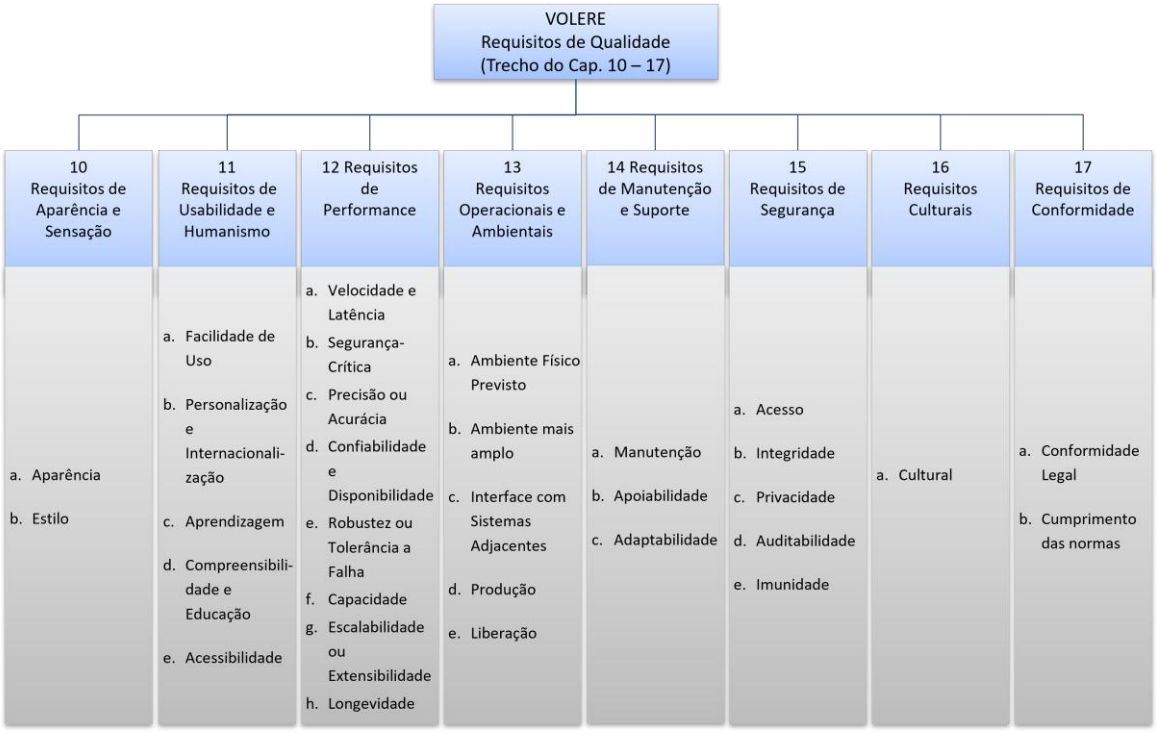


Figura 14: Categorias de qualidade de VOLERE

Em [RoRo2013] você encontrará não apenas as definições de todas estas categorias, mas também a razão pela qual são importantes. Você também encontrará exemplos de como formulá-los, incluindo critérios de aceite.

O exemplo a seguir é retirado de http://volere.co.uk/template.htm [RoRo2017]. Note que os critérios de aceite são chamados critérios de adequação nesta publicação.

11c. Requisitos de Aprendizagem

Conteúdo

Requisitos especificando como deve ser fácil aprender a usar o produto. Esta curva de aprendizagem varia de tempo zero para produtos destinados à colocação no domínio público (p. ex., um parquímetro ou um website) até um tempo considerável para produtos complexos e altamente técnicos.

Motivação

Quantificar o tempo que seu cliente considera aceitável para que um usuário possa usar o produto com sucesso. Este requisito orienta os projetistas na compreensão de como os usuários aprenderão o produto. Por exemplo, os projetistas podem construir instalações elaboradas de ajuda interativa no produto ou o produto pode ser embalado com um tutorial. Alternativamente, o produto pode ter que ser construído de modo que toda sua funcionalidade seja aparente ao encontrá-lo pela primeira vez.

Exemplos:
O produto deve ser fácil de aprender para um engenheiro.
Um funcionário deve ser capaz de ser produtivo em um curto espaço de tempo.
O produto poderá ser utilizado por membros do público que não receberão nenhum treinamento antes de usá-lo.
O produto será utilizado por engenheiros que participarão de cinco semanas de treinamento antes de utilizar o produto.

Critério de Adequação

Um engenheiro deverá produzir um [resultado especificado] dentro de [tempo especificado] ao começar a usar o produto, sem ter que usar o manual.

Após receber [número de horas] de treinamento, um funcionário deverá ser capaz de produzir [quantidade de saídas especificadas] por [unidade de tempo].

[Percentual acordado] de um painel de teste deverá concluir com sucesso [tarefa especificada] dentro de [tempo limite especificado].

Os engenheiros deverão alcançar [porcentagem acordada] a taxa de aprovação no exame final do treinamento.

Sugestões para o exercício:
Discuta para algumas das categorias mostradas em Figura 13 ou Figura 14 se os desenvolvedores devem saber sobre esses requisitos logo no início ou se eles podem ser considerados mais tarde no processo de desenvolvimento.

4.2 Acrescentando Precisão nos Requisitos de Qualidade

Os requisitos de qualidade devem ser comunicados aos desenvolvedores de uma forma que seja ao mesmo tempo inequívoca e verificável. Como mencionado anteriormente, os requisitos de qualidade são muitas vezes vagos no início.

Por exemplo: A nova geração de telefones celulares deve ser atraente para as crianças adolescentes.

Este requisito de qualidade não é inequívoco nem testável (na forma como é expresso), mas pode, no entanto, ser o ponto de partida para discussões sobre qualidades mais detalhadas requeridas para a próxima geração de telefones celulares.

Sua precisão (ou melhor, falta dela) pode ser comparada a um épico funcional como "Como usuário de telefone celular, eu quero capacidade de discagem inteligente". No capítulo 3 discutimos como levar tal épico ao nível de precisão, permitindo que os desenvolvedores o implementem.

Neste capítulo faremos o mesmo em relação aos requisitos de qualidade. Explicaremos primeiro como tornar os requisitos de qualidade mais concretos, até o nível de ter critérios de aceite. Depois – no capítulo 4.3– descreveremos como e onde registrá-los (fisicamente) ou armazená-los.

Há duas maneiras de acrescentar precisão e clareza aos requisitos de qualidade vagos. Você pode detalhá-los ou decompô-los, ou pode derivar requisitos (funcionais) mais precisos a partir do requisito original. Figura 15 mostra graficamente estas alternativas.

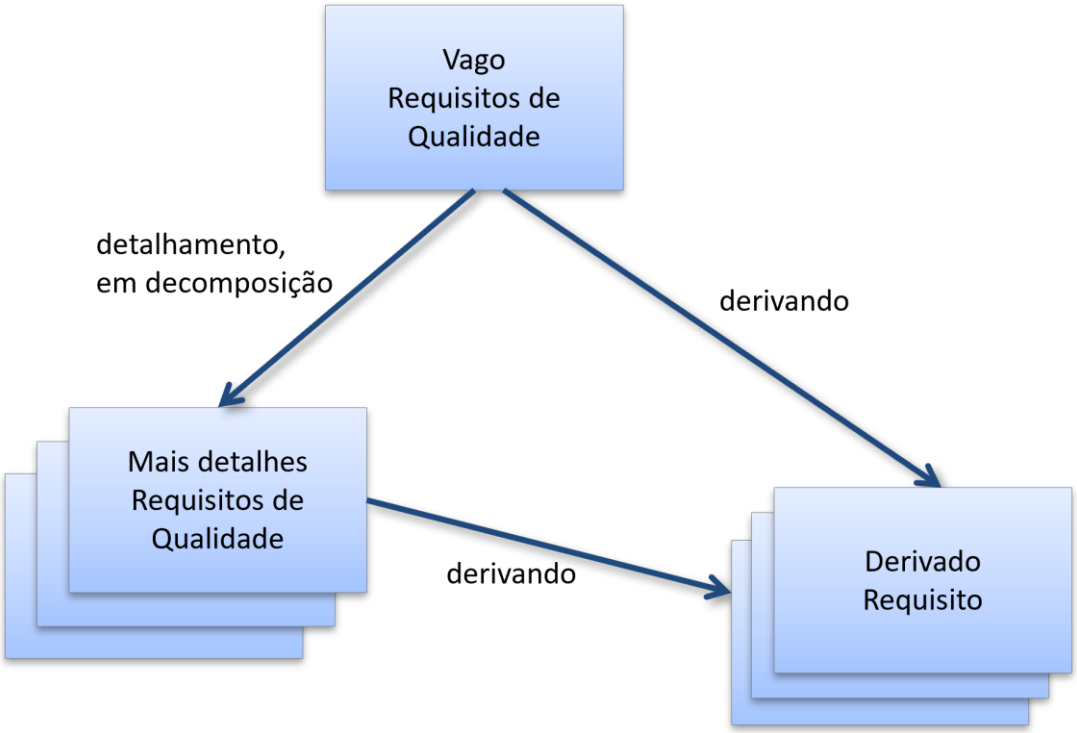


Figura 15: Detalhamento e decomposição dos requisitos de qualidade

O detalhamento ou decomposição toma o requisito de qualidade original vago e o substitui por dois ou mais requisitos de qualidade detalhados.

Exemplo: Analisando o esquema de categorização em Figura 14, você poderia detalhar o requisito de usabilidade (VOLERE categoria 11) "O sistema deve ser de fácil utilização" com os dois requisitos seguintes:

- Como usuário, quero que o sistema seja fácil de aprender (VOLERE categoria 11c), e
- Como usuário, quero que o sistema seja fácil de manusear (VOLERE categoria 11a).

Estes dois ainda são vagos, mas já são mais precisos do que o original.

A segunda alternativa "derivada" significa transformar o requisito de qualidade original em um ou mais requisitos (funcionais).

Veja, por exemplo, o requisito original: "Como oficial de segurança, quero o acesso às seguintes funções restritas a pessoal autorizado."

Derivar requisitos mais precisos significa, por exemplo, decidir que um mecanismo de login com nome de usuário e senha será usado para restringir o acesso.

Observe que a intenção original do requisito de qualidade era apenas garantir o acesso a certas funções. É uma decisão de projeto para conseguir isso, introduzindo papéis e senhas. Você poderia ter outras ideias, como trancar o computador em uma sala à qual somente pessoas autorizadas têm acesso. Alternativamente, você poderia decidir usar as impressões digitais para identificar os usuários autorizados.

Se você derivar novos requisitos funcionais dos requisitos de qualidade originais, você pode querer manter o requisito original, por exemplo, para lembrar sua origem, caso em futuras versões do produto você descubra maneiras mais inteligentes de alcançar a qualidade original. Derivar novos requisitos funcionais das qualidades exigidas aproxima-o de uma solução ou do cumprimento de tal requisito.

Sugestões para o exercício:
Escolha um de seus produtos e aperfeiçoe alguns exemplos de requisitos de qualidade.

Árvores de qualidade [[Clements et al.2001] também são uma forma comprovada de estruturar requisitos de qualidade. Uma árvore de qualidade combina as duas técnicas mencionadas acima. Figura 16 mostra a forma genérica de uma árvore de qualidade. Começa com uma raiz etiquetada como "qualidade específica". Os próximos ramos da árvore são categorias de qualidades, seguidas de subcategorias. As folhas da árvore mostram cenários concretos para uma categoria ou subcategoria, por exemplo, requisitos funcionais ou declarações de qualidade testável.

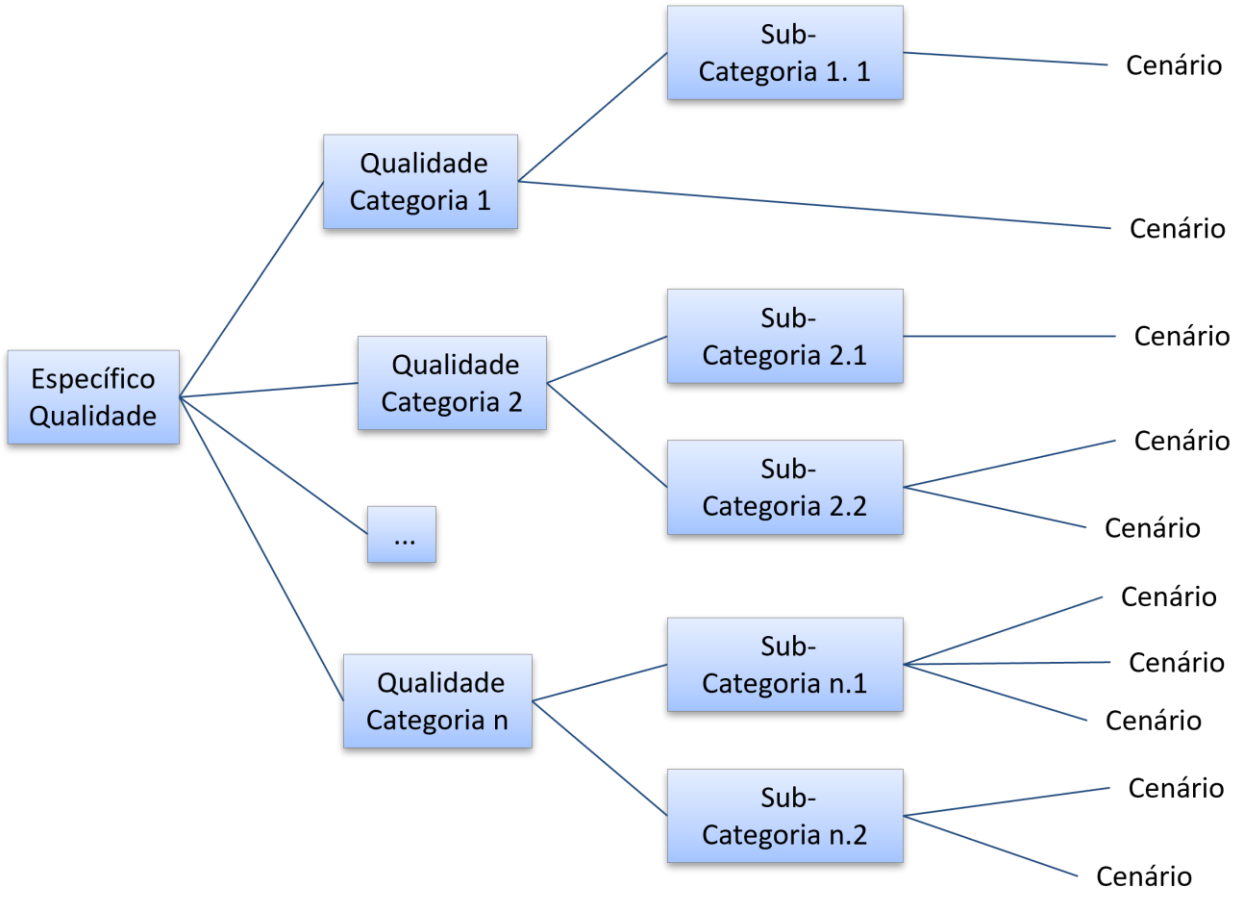


Figura 16: Um esquema genérico para uma árvore de qualidade

Para nosso estudo de caso iLearn Figura 17 mostra trechos de uma árvore de qualidade.

Observe os seguintes pontos:

- As folhas podem ainda não ser suficientemente precisas para serem testadas, por exemplo: "utilizáveis sem treinamento de estudantes". É por isso que os requisitos de qualidade precisam de critérios de aceite para informar os desenvolvedores sobre as expectativas do Product Owner.
- Há uma decisão de negócio muito clara no requisito de "outros idiomas". O Product Owner, juntamente com todos os stakeholders, decidiu que as legendas são suficientes para a comercialização do produto em outros países, em vez de, por exemplo, dublar os vídeos.
- Há até mesmo uma sugestão de projeto no requisito de "adaptabilidade": em vez de apenas solicitar que o sistema funcione em vários tipos ou dispositivos com resoluções diferentes, o Product Owner solicita o uso da tecnologia padrão da empresa: projeto responsivo.

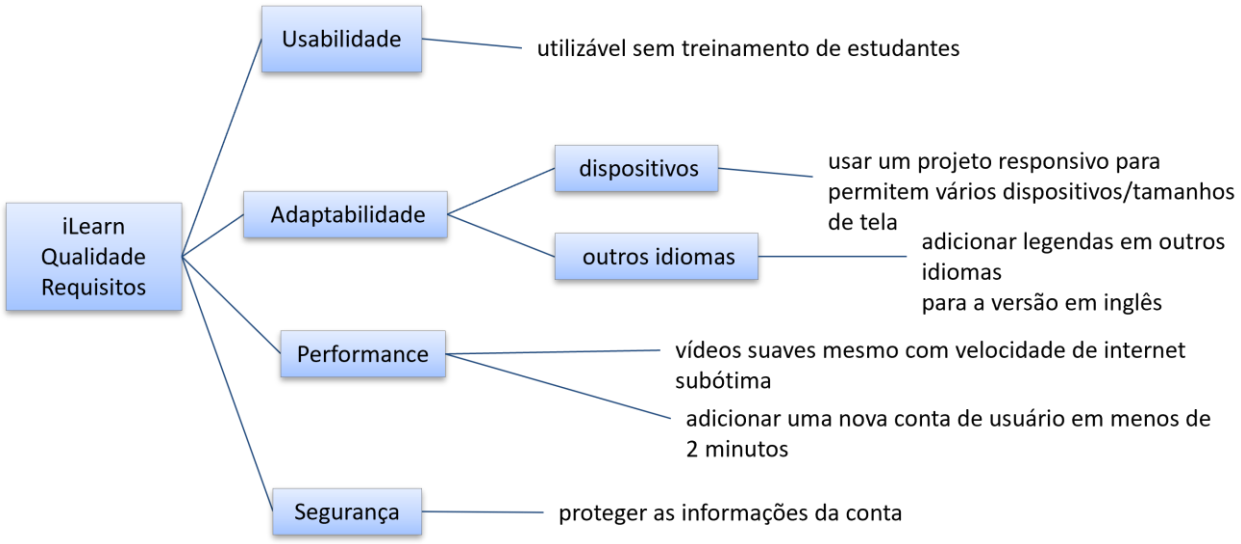


Figura 17: Partes de uma árvore de qualidade para iLearn

Sugestões para o exercício:
Tente pensar em uma árvore de qualidade parcial para um de seus produtos. Certifique-se de que você tem cenários muito concretos como folhas!

Como mencionado anteriormente, os requisitos de qualidade também precisam de critérios de aceite para acrescentar mais precisão. O tipo de critério de aceite utilizado dependerá da categoria da qualidade. A tabela a seguir mostra conselhos sistemáticos sobre como formular critérios de aceite para diferentes categorias de qualidades VOLERE.

Req. Tipo	Escala sugerida
10 Aparência & Sensação	Conformidade com as normas – especificar quem/como isto é testado
11 Usabilidade	Quantidade de tempo de aprendizagem Quantidade de treinamento O painel de teste pode executar funções em tempo alvo
12 Performance	Tempo para completar a ação
13 Operacional	Quantificação do tempo/ facilidade de uso no ambiente
14 Capacidade de manutenção	Quantificação do esforço de portabilidade Especificação do tempo permitido para fazer mudanças
15 Segurança	Especificação de quem pode usar o produto, e quando
16 Cultura e Política	Quem aceita, quantificação de costumes especiais
17 Jurídico	Opinião do advogado / processo judicial

Os capítulos seguintes fornecem exemplos de critérios de aceite dos requisitos de qualidade. Mais informações podem ser encontradas em [RoRo2013].

Requisitos de usabilidade: O produto deve ser utilizável por um membro do público, que pode não falar inglês.

Critério de Aceite: 45 dos 50 não falantes de inglês selecionados aleatoriamente devem ser capazes de usar o produto dentro dos critérios de performance mais 25%.

Requisito de performance: O produto deve ser aceitavelmente rápido.

Critério de Aceite: Cada transação na máquina de venda automática não deve levar mais do que 15 segundos.

Requisito operacional: Como trabalhador, tenho que usar o produto também quando estiver no exterior em condições de frio e chuva.

Critério de aceite: 90% dos trabalhadores no primeiro mês de uso devem utilizar o produto com sucesso dentro dos limites de tempo previstos.

Requisitos de segurança: Somente os gerentes diretos podem ver os registros pessoais de sua equipe. Os registros pessoais da equipe não podem ser vistos por mais ninguém.

Critério de Aceite: Registro dos acessos e testes para ver se alguém que não é gerente tem acesso. Alternativamente, pode-se dizer que o produto deve ser certificado como estando em conformidade com a norma xyz-security.

Requisito legal: As informações pessoais do cliente devem ser utilizadas de acordo com a Lei de Proteção de Dados.

Critério de Aceite: O departamento jurídico deve concordar que o produto esteja em conformidade com o registro de proteção de dados da organização.

Sugestões para o exercício:
Escolha dois exemplos de requisitos de qualidade e acrescente critérios de aceite a eles.

4.3 Requisitos de Qualidade e Backlog

Discutimos como descobrir e elucidar requisitos de qualidade e como torná-los mais precisos. Agora discutiremos como documentá-los em um ambiente Ágil em conjunto com um backlog de produto contendo principalmente requisitos funcionais. Dependendo do tipo de requisitos de qualidade, uma ou outra das seguintes abordagens irá funcionar para você.

A maneira mais fácil de registrar um requisito de qualidade é anexá-lo diretamente a um item de backlog. Esta abordagem só funciona se a qualidade for única a essa característica ou história de usuário.

Uma segunda abordagem é registrar os requisitos de qualidade fora do backlog, também:

- Em cartões separados;
- Como uma árvore de qualidade.

Em ambos os casos, é necessário vinculá-los a todos os requisitos funcionais relevantes. Dependendo das ferramentas utilizadas, isto pode ser feito usando hyperlinks, ou você tem que enumerar explicitamente os requisitos funcionais visados por cada qualidade.

A terceira alternativa é colocar os requisitos de qualidade na definição de feito. Uma vez que as regras na definição de feito se aplicam a TODAS as iterações, você está indicando que quer sempre que esse requisito seja obedecido, independentemente de quais requisitos funcionais você anexe à próxima iteração.

4.4 Tornando as Restrições Explícitas

As restrições são um tipo importante de requisito. Glinz define restrições como requisitos que limitam o espaço da solução além do que é necessário para atender aos requisitos funcionais e de qualidade fornecidos [Glinz2014]. O produto deve ser construído dentro das restrições. As restrições limitam o que você pode decidir e, portanto, influenciam e moldam o produto.

Elas são determinadas por sua administração ou por outros stakeholders fora de seu escopo de controle, por exemplo, autoridades reguladoras, sua empresa mãe ou um arquiteto corporativo.

Note que embora muitas restrições sejam certamente legítimas, muitas vezes vale a pena para o Product Owner ou desenvolvedores verificar sua validade e negociar com pessoas ou organizações que colocam tais restrições em seu desenvolvimento; questionar suas razões e motivações.

Às vezes você descobrirá que algumas das restrições são puro folclore que – uma vez questionadas e sugeridas alternativas – podem ser negociadas com os stakeholders responsáveis e flexibilizadas, permitindo mais liberdade na implementação. Portanto, em uma terminologia Ágil: As restrições também podem ser negociáveis, da mesma forma que a funcionalidade. Entretanto, se as outras partes insistirem nessas restrições, então os desenvolvedores terão que aceitá-las.

Neste Manual, incluímos requisitos legais ou (mais gerais) qualquer tipo de requisitos de conformidade como categorias de requisitos de qualidade (ver capítulo 4.1). Elas também poderiam ser incluídas neste capítulo sobre restrições, uma vez que qualquer solução tem que ter estas qualidades. Em comparação com as outras categorias de restrições, tais requisitos de conformidade muitas vezes não são negociáveis.

Figura 12 mostra uma maneira de categorizar as restrições: Elas podem ser classificadas como restrições de produto ou como restrições de processo. Somente as restrições do produto referem-se aos requisitos funcionais ou de qualidade do produto, limitando assim sua implementação. As restrições de processo não têm relação direta com o produto. Elas impõem limites à organização que desenvolve o produto, ou ao processo de desenvolvimento utilizado para o desenvolvimento do produto. Assim, eles têm apenas um efeito indireto sobre o próprio produto.

Figura 18 sugere algumas subcategorias para estas duas categorias. Alguns exemplos são discutidos no texto a seguir. Mais detalhes sobre como formular tais restrições, e mais exemplos, podem ser encontrados em [RoRo2013].

As restrições do produto podem exigir uma determinada infraestrutura, ou seja, um ambiente tecnológico e/ou físico no qual o produto deve ser instalado. Outros exemplos incluem o uso obrigatório de software de prateleira (o que significa uma decisão de compra em oposição ao desenvolvimento de subsistemas dentro do projeto).

Figura 18: Categorização das restrições

A restrição para reutilizar componentes ou subsistemas existentes de produtos predecessores ou outros produtos desenvolvidos pela empresa é uma restrição que é frequentemente introduzida. O motivo da reutilização é óbvio: você não quer gastar dinheiro se tiver à sua disposição soluções aceitáveis (parciais).

Restrições relativas ao ambiente operacional previsto do produto descrevem quaisquer características do local de trabalho que possam ter um efeito sobre o projeto. Os projetistas de produtos devem saber, por exemplo, que o local de trabalho é ruidoso, portanto, os sinais de áudio podem não funcionar.

Por outro lado, quando o produto for destinado a operar em ambientes silenciosos, o nível de ruído produzido pelo produto não deve exceder um certo nível de decibéis. Se o local de trabalho estiver ao ar livre onde possa estar molhado e frio, então os usuários devem poder usar o produto usando luvas.

Similar para sistemas envolvendo elementos de hardware, restrições físicas tais como aquelas relacionadas ao tamanho ou peso do dispositivo – pense em telefones celulares ou outros dispositivos portáteis – também podem ser muito relevantes (ou seja, relevantes tanto para o projeto do hardware quanto para o software que ele é capaz de suportar).

As restrições mais comuns ao produto, no entanto, limitam a tecnologia que os desenvolvedores estão autorizados a utilizar.

Por exemplo:
– Como arquiteto da empresa, quero que você desenvolva o produto em C# para que nosso pessoal existente possa manter o produto.
– Como administrador do banco de dados, quero que a equipe do produto utilize ORACLE, uma vez que temos um excelente suporte de linha direta para este produto.

Note que você não precisa escrever restrições como histórias. Pode ser suficiente informar à equipe que C# e ORACLE são restrições não negociáveis.

Restrições de processo são frequentemente chamadas de restrições organizacionais, uma vez que elas restringem tanto aspectos de gestão como orçamento, cronograma ou as habilidades dos membros da equipe disponíveis para o projeto ("Você tem que trabalhar com esta equipe. Não temos orçamento para contratar pessoal adicional e nenhum orçamento para pessoas externas. ") ou aplicam certas políticas e regulamentos. Talvez você tenha que seguir um processo de desenvolvimento que prescreva certas funções, atividades obrigatórias a serem realizadas durante o desenvolvimento e um conjunto de documentos ou outros artefatos a serem produzidos e mantidos.

As restrições, como outros tipos de requisitos, têm uma descrição: elas podem conter uma fundamentação ou motivação descrevendo o motivo pelo qual a restrição está em vigor. E eles também devem ter critérios de aceite – assim como para os requisitos funcionais ou de qualidade.

Se você trabalhou em uma organização por algum tempo, é provável que tenha aprendido sobre as preferências tecnológicas da organização e estará ciente das regras e restrições organizacionais. No entanto, é importante tornar essas restrições explícitas para que todos os outros membros da equipe estejam cientes delas. As mais limitantes devem ser conhecidas no início do projeto. Outras devem ser capturadas assim que forem descobertas.

Tais restrições são normalmente aplicáveis a uma gama mais ampla de projetos. Os conjuntos de tecnologia básica, bem como os modelos de processo, normalmente são definidos por um período mais longo em uma empresa. Portanto, assim que essas restrições são capturadas, elas podem ser facilmente reutilizadas em diferentes desenvolvimentos de produtos.

4.5 Sumário

Os requisitos e restrições de qualidade são tão importantes para o sucesso do projeto quanto os requisitos funcionais. Para um Product Owner não é difícil encontrar requisitos relevantes nestas categorias, pois há muitas listas de verificação disponíveis no domínio público, sugerindo categorias para qualidades e restrições.

Os requisitos de qualidade podem começar vagos. Antes de estarem prontos para o desenvolvimento, eles precisam ser mais precisos, até o nível dos testes de aceite, assim como acontece com os requisitos funcionais.

Acrescentar precisão aos requisitos de qualidade é muitas vezes alcançado através da derivação de requisitos funcionais correspondentes que atendem as qualidades originalmente exigidas. Certifique-se de que tais decisões sejam registradas e que os requisitos de qualidade originais não sejam descartados, pois com o tempo você poderá descobrir melhores maneiras de cumprir as qualidades.

Alguns requisitos de qualidade podem ser simplesmente anexados a histórias de usuários já descobertas, por exemplo, adicionando performance ou aspectos especiais de segurança a funções individuais. Muitos requisitos de qualidade dizem respeito a aspectos transversais, o que significa que eles são relevantes para muitos dos requisitos funcionais.

Para essas, sugerimos que você mantenha uma lista separada, sempre visível para os desenvolvedores, pois elas devem ser sempre cumpridas. Uma alternativa é incluí-las na definição de pronto, o que tem o mesmo efeito de ser sempre válido.

Uma abordagem semelhante pode ser adotada para restrições técnicas, organizacionais e legais. Certifique-se de que elas sejam explicitamente conhecidas pelos desenvolvedores. Se não forem específicas do projeto, mas regras mais gerais da empresa, você poderá mantê-las em um local central para todos os projetos, reutilizando-as em vários projetos de desenvolvimento.

5 Priorização e Estimativa de Requisitos

As abordagens Ágeis visam maximizar o valor de negócio geral ao longo do tempo e otimizar permanentemente o processo geral de criação de valor de negócio [Leffingwell2010]. Este processo de agregação de valor constante é mostrado em Figura 19. Cada iteração deve resultar em valor agregado – às vezes mais, às vezes menos.

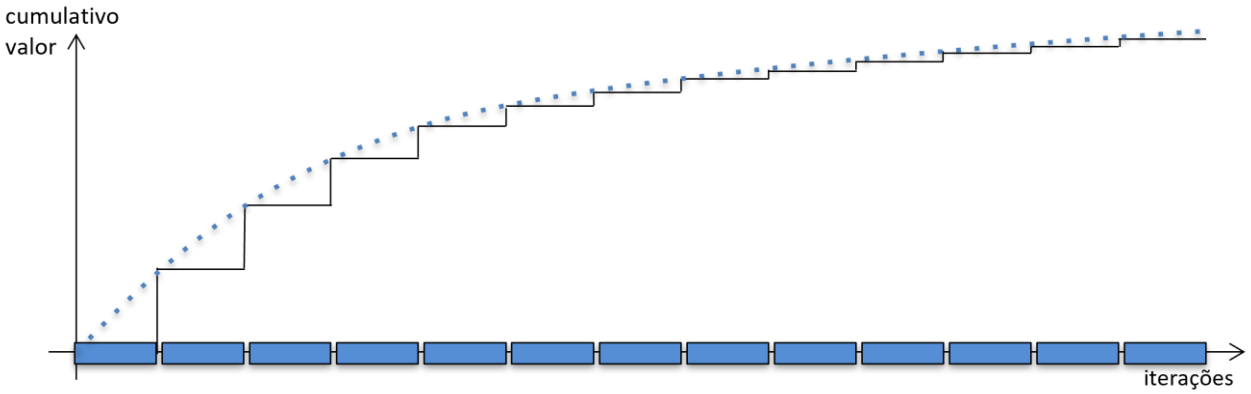


Figura 19: Desenvolvimento Ágil = criação de valor constante

Cada iteração deve proporcionar um incremento de produto potencialmente liberável que aumenta o valor do produto em geral. (Comentário: algumas versões do Scrum e de outras abordagens ágeis se referem a um "produto potencialmente embarcável" ou "incremento de produto potencialmente utilizável").

[LeSS] explica este objetivo da seguinte forma: "Potencialmente embarcável é uma declaração sobre a qualidade do software e não sobre o valor ou a comercialização do software. Quando um produto é potencialmente embarcável, isso significa que todo o trabalho que precisa ser feito para os recursos atualmente implementados foi feito e, tecnicamente, o produto pode ser embarcado, mas isso não significa que os recursos implementados sejam valiosos o suficiente para que o cliente queira uma nova versão. Este último é determinado pelo. Dono do Produto."

Ao planejar e alcançar esta constante adição de valor, todos os requisitos (sejam eles grosseiros ou refinados) devem ser pedidos principalmente com base no valor agregado que podem trazer para o negócio. Mas o valor de negócio pode significar muitas coisas diferentes para organizações diferentes. O esclarecimento deste termo "valor de negócio" é um dos tópicos centrais deste capítulo e será discutido nos capítulos 5.1 a 5.3.

Naturalmente, a criação de valor tem que ser equilibrada com o esforço de criá-lo e o momento em que ele será entregue. Portanto, os desenvolvedores têm que apoiar o Product Owner com estimativas sobre os esforços necessários para criar o valor de negócio. A estimativa dos itens de backlog é o segundo tópico central deste capítulo e será discutido no capítulo 5.4. Com base na relação valor/efeito, o Product Owner pode selecionar as histórias que devem ser assumidas pelos desenvolvedores na próxima iteração.

5.1 Determinação do Valor de Negócio

Como mencionado acima, "valor" pode significar muitas coisas diferentes em ambientes diferentes. Aqui estão alguns aspectos a serem considerados ao estabelecer o valor de negócio e ao colocar os itens de backlog em ordem por esse valor.

- **Valor para o cliente ou outros stakeholders**
  Se você desenvolver um produto para um cliente qualquer ou específico, a opinião dele sobre o que é mais ou menos importante influenciará definitivamente quando você escolher itens de backlog. Nem todos os stakeholders consideram o dinheiro como um critério de valor. O valor para o Greenpeace, por exemplo, pode ser qualquer coisa boa que você faça para proteger o meio ambiente. Portanto, quaisquer que sejam os valores de seus clientes ou stakeholders importantes, serão considerados.
- **Valor para a organização**
  Apesar de ter clientes específicos que irão usar ou comprar o produto, a própria organização pode (ou deve) ter objetivos estratégicos que deseja alcançar, por exemplo, criar uma plataforma reutilizável para um determinado domínio, para que futuros projetos possam ser entregues mais rapidamente e mais baratos. De fato, qualquer tipo de otimização e automação dos processos internos de negócio pode ser uma força motriz para a criação de valor para a organização. Se os itens de backlog estiverem fortemente relacionados a tais objetivos estratégicos, então seu valor de negócio será considerado muito alto.
- **Ameaça à existência**
  Não ter ou oferecer um determinado recurso ou funcionalidade pode ser uma ameaça para o produto ou para a organização em geral. Exemplos típicos de tais ameaças são requisitos legais (p. ex., proteção de dados). Tal característica pode não agregar valor de negócio em um sentido comercial, mas deve ser implementada para garantir a existência futura do produto ou da empresa.
- **Valor financeiro esperado de uma funcionalidade** (volume de vendas, receita total, retorno sobre o investimento)
  O objetivo da maioria das organizações comerciais é ganhar dinheiro (lucro). Portanto, as funcionalidades e histórias serão naturalmente mais bem classificadas se prometerem mais vendas ou um rápido retorno sobre o investimento.
- **Metas de curto prazo do projeto ou metas de lançamento** (versus metas de médio prazo do produto)
  Às vezes é importante ser capaz de demonstrar funcionalidades ou pelo menos maquetes de características em uma próxima feira ou uma apresentação importante. Portanto, os Product Owners podem valorizar tais resultados mais do que aqueles que contribuem para a estratégia de produto a longo prazo. Por outro lado, uma organização pode querer investir em uma estrutura de desenvolvimento que não cria imediatamente valor de negócio, mas reduz os custos de desenvolvimento a longo prazo e melhora a relação de valor agregado para os próximos incrementos de produtos.

- **Custos do atraso**
  Este é um critério muito interessante para a determinação do valor de negócio. A questão-chave é: Qual é o custo do atraso do envio de uma história? Por exemplo, um novo recurso de um portal de compras on-line deve aumentar o volume de vendas em US$ 500.000 por mês significa que a empresa perde US$ 500.000 se o recurso for adiado por um mês. Reinertsen [Reinertsen2008] considera o custo do atraso como um ponto de vista que pode resumir todos os outros aspectos mencionados neste capítulo.
- **Tempo de mercado**
  Certas funcionalidades podem vir com uma janela de oportunidade. Por exemplo: Se esta funcionalidade estiver disponível dentro deste período, então criará um aumento significativo nos negócios. Se chegar tarde demais, o valor pode ser significativamente menor. Por exemplo, as feiras de negócios são uma boa oportunidade para vender novos produtos ao mercado. Se o produto não estiver pronto quando a feira abrir, os clientes poderão comprar outro produto e não terão necessidade de comprar o produto no futuro próximo, mesmo que o produto tenha mais e melhor funcionalidade. Alguns métodos, portanto, sugerem colocar um atributo em cada item de backlog especificando "melhor antes". Desta forma, todos os stakeholders sabem explicitamente sobre a janela de oportunidade.
- **Frequência de requisitos**
  Se você desenvolver um produto para um mercado de massa, pode ser importante compreender a demanda ao determinar o valor do negócio. Muitos clientes pediram por isso? Ou era apenas um pequeno grupo? Qual a receita que você espera obter com base no número de clientes que solicitaram o recurso?
- **Dependências de negócios e dependências técnicas**
  Às vezes é preciso priorizar um elemento de backlog porque é um pré-requisito para um ou mais itens de backlog, o que significa que os outros itens não podem ser desenvolvidos se este não estiver disponível. Um exemplo no estudo de caso iLearn seria: o desenvolvimento de uma conta de usuário não cria valor de negócio, mas você não pode desenvolver recursos personalizados se ainda não tiver desenvolvido o recurso de conta de usuário. Estas dependências também podem ser técnicas, por exemplo, o desenvolvimento de um recurso requer o estabelecimento de uma certa infraestrutura ou certas ferramentas têm que ser compradas e exploradas antes que você possa entregar a funcionalidade. Estes pré-requisitos (funcionalidades) não criarão valor de negócio, mas sem ter estes pré-requisitos feitos, você não poderá desenvolver os itens realmente valiosos do backlog.

Além disso, algumas das qualidades podem ser consideradas como tendo alto valor. Você pode dar prioridade aos itens de backlog que, por exemplo:

- Melhoram a usabilidade
- Melhoram a robustez
- Reduzem os custos de manutenção
- Minimizan o impacto sobre o sistema atual

Trabalhar em tais melhorias de qualidade não costuma criar novas funcionalidades vendáveis, de modo que não geram receita direta. Mas eles podem ser considerados muito importantes por certos grupos de stakeholders e, portanto, serem de prioridade alta nos itens de backlog.

O valor entregue só pode ser medido no lado do usuário final porque o usuário final do produto decidirá se quer usar (e comprar) o produto e se recomendará o produto a outros clientes potenciais. Como resultado disso, a receita da empresa produtora pode aumentar.

Se o cliente é interno, não há receita para medir, então tipicamente o valor dos incrementos do produto entregue é determinado pela classificação do incremento do produto entregue e da versão do produto resultante sprint a sprint e comparando-o ao roteiro do produto com base nas características planejadas e entregues e nas capacidades do produto.

5.2 Valor de Negócio, Risco

Um critério importante para priorizar os itens em atraso é que alguns possuem maior risco que outros.

[DeMaLi2003] dá uma definição cíclica dos riscos e problemas:

- Um risco é um problema potencial.
- Um problema é um risco que se manifestou.

Há muitas categorias de riscos no desenvolvimento de produtos. A característica em si pode ser arriscada, porque, por exemplo, pode não ser aceita pelo público-alvo. O risco pode estar na implementação de um recurso, por exemplo, se a equipe quiser usar determinada tecnologia, enquanto nem todos os membros da equipe são proficientes com a tecnologia.

Ou o risco pode estar na própria tecnologia, que pode ser muito nova (e, portanto, perigosa de usar) ou muito antiga ou desatualizada. Para uma visão abrangente dos riscos, especialmente os cinco principais riscos que afetam cada projeto de TI, consulte [DeMaLi2003].

Talvez os itens de backlog arriscados não proporcionem alto valor de negócio com base nos critérios definidos no último capítulo. Mas se você quiser lidar com os riscos a fim de evitar surpresas mais tarde, então você pode querer escolher itens de backlog que vêm com um risco no início do processo de desenvolvimento. Uma vez lidado com esses itens, o resto do trabalho é menos arriscado.

Há quatro alternativas que você pode escolher quando tiver itens de risco de atraso:

1. Evite o risco: Isto significa não lidar com itens de backlog que são arriscados. Evitar tais itens implica em perder as oportunidades associadas com os itens. Portanto, evitar não deve ser sua escolha ao lidar com itens de risco.
2. Mitigar os riscos: Como gerente, você pode colocar dinheiro e/ou tempo de lado para lidar com os riscos assim que eles se tornarem problemas. Como Product Owner (responsável pela Engenharia de Requisitos) você pode, portanto, adiar o estudo detalhado de tais itens até que eles se tornem importantes para o negócio.

3. Reduzir os riscos: além da mitigação, esta é sua segunda opção óbvia para lidar com itens de risco. Mas isto significa tomar medidas agora para reduzir o risco. Você normalmente decompõe um item de risco em itens menores (p. ex., spikes) que lhe permitem aprender mais sobre suas partes de risco. Por exemplo, você desenvolve um protótipo de UI para garantir que o público-alvo o aceite, ou desenvolve um protótipo para ganhar experiência com uma nova estrutura.
4. Espera-se que o risco não se transforme em um problema. Semelhante à primeira alternativa, esta não é uma escolha viável. Imagine que você tem doze riscos com uma probabilidade de apenas dez por cento cada um. A matemática mostra que a chance de que uma dessas coisas o atinja já é de 75%.

Como Product Owner, você só quer optar pelas alternativas dois e três. Do ponto de vista dos requisitos, a alternativa três é a mais importante. É preciso encontrar maneiras de decompor um requisito de forma a reduzir o risco. Às vezes, você pode usar um spike ou desenvolver um protótipo para reduzir o risco antes de avançar para o desenvolvimento real da funcionalidade.

[DeMaLi2003] conclui: "O verdadeiro motivo pelo qual precisamos fazer o gerenciamento de riscos não é para evitar riscos, mas para permitir a adoção de riscos agressivos."

Sugestões para o exercício:
Discuta quais (combinação de) critérios são usados em sua organização para determinar o valor (negócio).

5.3 Expressando Prioridades e Organizando o Backlog

Uma vez determinado o valor que significa para você, você tem que expressar estas prioridades e ordenar o backlog de acordo com as prioridades dadas aos itens de backlog. Há muitos métodos diferentes para atribuir valor aos itens de backlog, alguns deles muito simples, outros altamente complexos. No capítulo seguinte, discutiremos as abordagens populares.

Um método é usar o MoSCoW. Este método de priorização foi desenvolvido por [ClBa1994] para alcançar um entendimento comum com os stakeholders sobre a importância que atribuem à entrega de cada requisito. O próprio termo MoSCoW é um acrônimo derivado da primeira letra de cada uma das quatro categorias de priorização (Must have, Should have, Could have, and Won't have), com os o's adicionados para tornar a palavra pronunciável.

As categorias são tipicamente entendidas como:

- **Must have**: (*Terá*) os requisitos assim rotulados são essenciais para que o período de entrega atual seja um sucesso. Se um requisito obrigatório não for incluído, a entrega do projeto deve ser considerada um fracasso (nota: os requisitos *devem ser* rebaixados de obrigatórios, mediante acordo com todos os stakeholders relevantes; por exemplo, quando novos requisitos são considerados mais importantes).

- **Should have**: (*deveria ter*) os requisitos são importantes, mas não necessários para a entrega no período de entrega atual. Embora os requisitos *Should have* possam ser tão importantes quanto os *Must have*, muitas vezes eles não são tão críticos em termos de tempo ou pode haver outra maneira de satisfazê-lo, de modo que ele possa ser adiado para um prazo de entrega futuro.
- **Could have**: (*poderia ter*) Estes requisitos são desejáveis, mas não necessários, e podem melhorar a experiência do usuário ou a satisfação do cliente com pouco custo de desenvolvimento. Estes serão normalmente incluídos se o tempo e os recursos permitirem.
- **Won't have** (desta vez): (*não terá*) estes requisitos foram acordados pelos stakeholders como os itens menos críticos e de menor retorno, ou não são apropriados naquele momento. Como resultado, requisitos *Won't have* não serão planejados no cronograma para o próximo período de entrega. Os requisitos *Won't have* são descartados ou reconsiderados para inclusão em um período posterior.

Um esquema mais simples para expressar prioridades poderia ser usar três categorias (em vez das quatro de MoSCoW), rotuladas A(lto), M(édio) e B(aixo) ou, alternativamente, A, B e C.

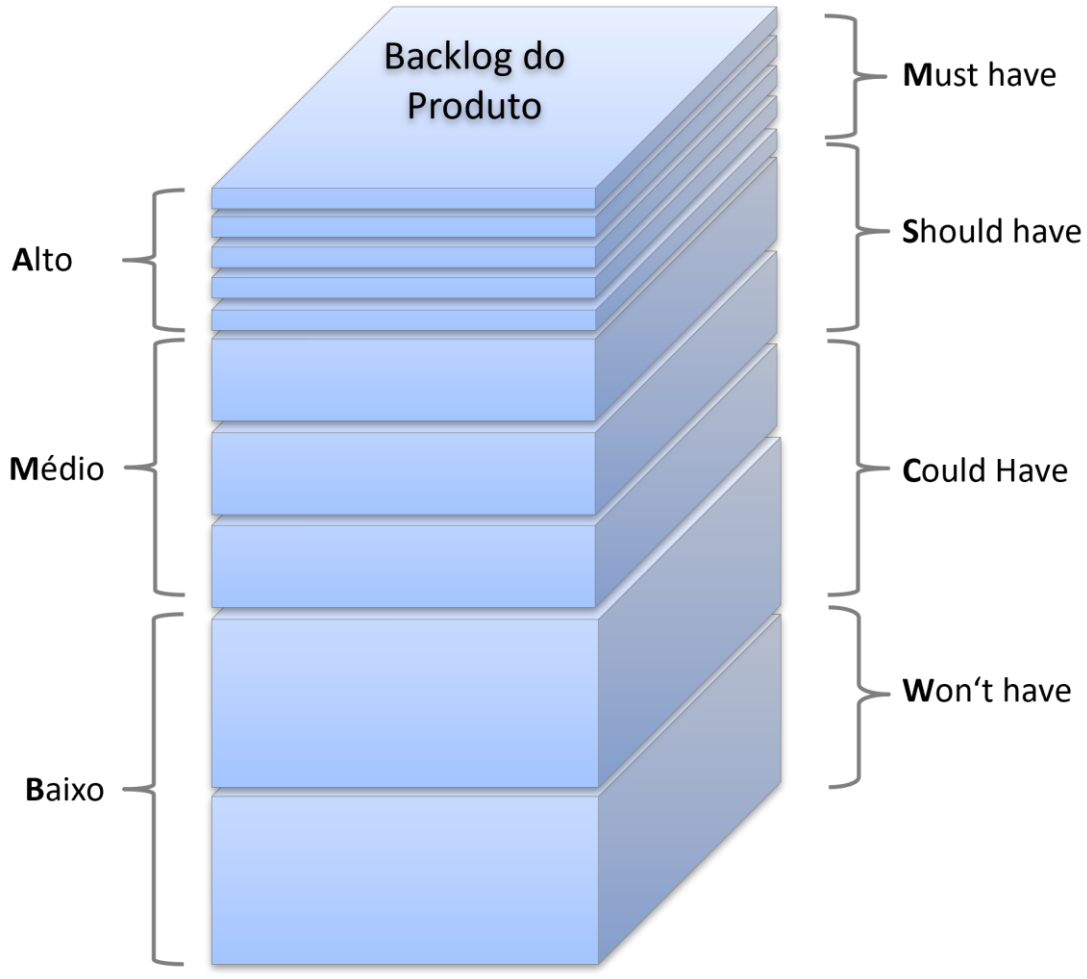


Figura 20: MoSCoW ou prioridades altas/médias/baixas

Figura 20 mostra um backlog onde os itens são anotados como alto, médio e baixo ou MoSCoW. Observe que quanto maior o valor dado ao requisito, mais detalhado ele já deve ser descrito, já que é um candidato potencial para a(s) próxima(s) iteração(ões) (ou uma das próximas).

Algumas empresas utilizam uma gama de números entre 1 e 100, interpretando-o de uma forma que um número maior significa mais valor de negócio. Assim, você pode expressar diferenças maiores, por exemplo, dando prioridade 87 a um item com prioridade 87 e 38 a outro, indicando claramente o quanto mais importante é o item com prioridade 87.

Figura 21 mostra uma gama de números dados a itens menores ou maiores em atraso. Note que se um item de tamanho médio tem valor 95 ou um grande épico tem valor 76 como na figura abaixo, esta é uma mensagem clara para o Product Owner para começar a trabalhar nesse item para levá-lo à definição de pronto, para que itens tão importantes possam ser tratados em uma iteração de curto prazo.

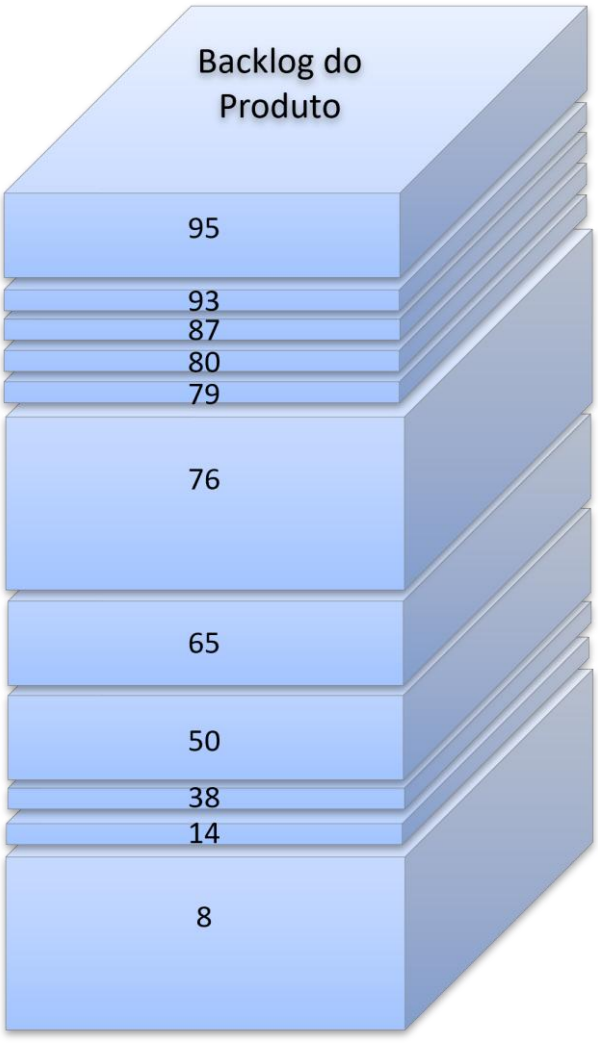


Figura 21: Utilização de uma gama de números para indicar o valor de negócio

A maneira mais simples seria classificar todos os itens de backlog em uma sequência linear (ou seja, colocar os cartões de histórias em uma fila da esquerda para a direita).

Quanto mais à esquerda, mais importante é considerado o item de atraso. Quanto mais à direita você coloca, menos importante este item é considerado. Isto é mostrado na Figura 22.

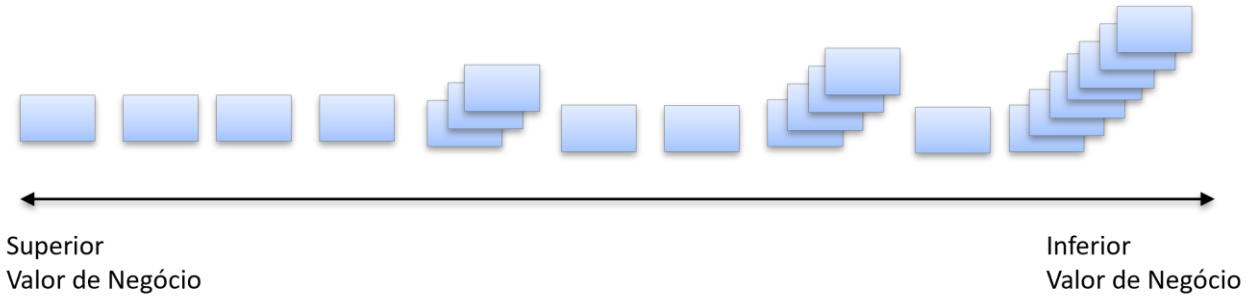


Figura 22: Separação linear por grupos de valores de negócio

Observe que apenas os itens mais à esquerda têm que ser claramente linearizados, já que os desenvolvedores os escolherão para a próxima iteração. Quanto mais à direita um item é colocado, menos importante é sua posição exata. Assim, você pode colocar conjuntos de itens em pilhas sem decidir explicitamente seu valor exato.

O Product Owner tem tempo para o refinamento antes de serem escolhidos para a implementação. Fazer a classificação da esquerda para a direita rapidamente e concentrar-se apenas naqueles itens que prometem alto valor de negócio.

Naturalmente, você poderia aplicar algoritmos muito mais complexos para determinar o valor. Você pode, por exemplo, escolher alguns dos critérios mencionados no capítulo 5.1 e atribuir um peso a cada um deles para equilibrar os valores relativos um ao outro. Você pode então classificar individualmente os itens de backlog de produto dentro de cada critério e calcular o valor resultante. Figura 23 demonstra isto com três critérios e uma classificação de 0 a 5 dentro de cada critério. Como você pode ver, a História 3 se revela a mais valiosa, baseada nessa abordagem combinatória de receita, risco e usabilidade.

Itens de Backlog	Receita no 2º trimestre		Minimizando o Risco Técnico		Melhorando a Usabilidade		Geral Valor
	Peso = 5	Valor da Receita	Peso = 4	Valor do Risco	Peso = 2	Valor da Usabilidade	
Item 1	3	15	0	0	0	0	**15**
Item 2	0	0	3	12	1	2	**14**
Item 3	4	20	2	8	2	4	**32**
Item 4	2	10	2	8	3	6	**24**
Item 5	0	0	2	8	5	10	**18**
...							
...							

Figura 23: valor de negócio calculado com base em múltiplos critérios

5.4 Estimar Histórias de Usuário e outros Itens do Backlog

Para o Product Owner, este capítulo é apenas informativo. Ele ou ela só é responsável por determinar a ordem dos itens de backlog com base no valor e no risco, conforme discutido no último capítulo. É tarefa dos desenvolvedores elaborar estimativas para cada item de backlog. O Product Owner não deve influenciar o processo de estimativa, apenas estar ciente dos resultados.

Mesmo em um mundo perfeito e Ágil, as previsões são úteis e valiosas (se aplicadas corretamente) para determinar quanto trabalho pode ser "feito" dentro de uma iteração previamente especificada (período).

Nenhum elemento não estimado pode entrar em um sprint no Scrum por dois motivos [Cohn2006]:

1. Não está claro se o elemento pode ser completado dentro do sprint e como resultado disso, o software pode não estar funcionando no final do sprint.
2. Sem discussão e estimativa, a equipe não terá nenhum ponto de referência (planejado vs. realizado) para o aprendizado futuro com relação aos próximos sprints.

A maioria das pessoas não gosta de fazer estimativas. Em muitas organizações não-Ágeis, estimativas imprecisas foram tipicamente usadas contra em uma etapa posterior. Se sua estimativa fosse muito alta, então poderá ser visto como muito defensivo ou muito ansioso. Se sua estimativa fosse muito baixa, então você poderá ser desafiado porque não viu os esforços reais por trás do trabalho que tinha que ser feito.

As organizações Ágeis tentam superar esta antipatia estabelecendo um tipo diferente de cultura de estimativa. Uma cultura que ajuda a evitar apontar o dedo. Os princípios desta cultura serão discutidos neste capítulo.

Antes de tudo, a razão para ter melhores estimativas é o uso de iterações curtas no desenvolvimento Ágil. É muito mais fácil dar estimativas mais precisas para as próximas duas a quatro semanas em comparação com estimativas para trimestres ou para anos.

É claro que as organizações de desenvolvimento que trabalham em grandes projetos com múltiplas equipes também precisam de previsões para priorizar e planejar o trabalho adequadamente. A estimativa e o planejamento em larga escala serão discutidos em mais detalhes no capítulo 6. Neste capítulo vamos nos concentrar em estimativas a curto prazo, por exemplo, estimativas para as próximas iterações.

Os métodos Ágeis sugerem algumas boas práticas que ajudam a ter estimativas melhores e precisas:

1. Todos os envolvidos no processo de estimativa devem ter o mesmo entendimento do trabalho que precisa ser "feito". Isto é conseguido através do envolvimento dos desenvolvedores no refinamento do backlog de produtos. Os desenvolvedores ajudam o Product Owner a refinar épicos, funcionalidades e histórias pouco claras ou qualquer tipo de requisito nesses níveis de granularidade, obtendo assim mais informações sobre o trabalho a ser feito. A criação de um entendimento comum sobre o que "pronto" realmente significa nesse contexto evita as típicas armadilhas da estimativa (esquecer o esforço necessários para a documentação, testes ou a preparação da implementação).
2. A estimativa é feita por quem faz o trabalho; os desenvolvedores multifuncionais. Isso ajuda a colocar todas as pessoas envolvidas no mesmo nível de conhecimento por meio da troca de conhecimento e do compartilhamento de suposições sobre o trabalho a ser feito. Naturalmente, é preciso considerar uma troca entre o envolvimento de todos os membros da equipe no processo de estimativa e o envolvimento de apenas alguns deles. O envolvimento de todos significa que todos fazem parte do processo e, portanto, se sentem comprometidos com o resultado. Mas isto pode levar muito tempo que, de outra forma, poderia ser gasto no desenvolvimento de funcionalidades. Se apenas alguns desenvolvedores participam do processo de estimativa, então os outros podem não se sentir comprometidos. Uma boa prática é convidar toda a equipe e deixar que a ela decida quem realmente é necessário estimar. Em todos os casos, a estimativa deve ser feita por grupos e não por indivíduos. Mais adiante, neste capítulo, sugeriremos técnicas para acelerar a estimativa.
3. A estimativa deve ser feita em relação ao trabalho já realizado ou, no início, em relação ao trabalho pequeno que todos os envolvidos possam concordar. A estimativa por analogia ou afinidade é provavelmente mais precisa do que a estimativa absoluta. Olhando para Figura 24, é fácil afirmar que a rocha à direita tem mais que o dobro de tamanho comparando com a rocha à esquerda. Seria muito mais difícil estimar o tamanho ou peso exato dos dois. As estimativas relativas oferecem precisão suficiente para o planejamento.

Figura 24: Estimativas relativas

4. A estimativa deve ser feita usando uma unidade artificial (geralmente chamada de pontos de história) que represente esforço, complexidade e risco em um só. O uso de uma unidade artificial como pontos de história é necessário para que todos se familiarizem com a nova forma de estimar e a cultura associada e se afastem do comportamento tradicional.

Várias técnicas suportam a estimativa relativa. As técnicas mais conhecidas são T-Shirt Sizing ou o Planning Poker [Cohn2006].

Para todas estas técnicas é relevante que primeiro se chegue a um acordo sobre um item de referência (ou história de referência). Vamos assumir que a maçã em Figura 25 é a referência escolhida. Agora você pode estimar o tamanho de todas as outras frutas em comparação com aquela maçã. Eles são aproximadamente do mesmo tamanho? Eles são muito menores? Ou muito maior?

As estimativas relativas eliminam o medo entre os desenvolvedores de que eles tenham que ser exatos.

Os indicadores de T-Shirt Sizing variam de extra pequeno a extragrande (XXS, XS, S, M, M, L, XL, XXL). Alguns métodos sugerem não utilizar todos esses indicadores de tamanho ao fazer as estimativas, pois eles podem já ser muito precisos. Pense em um subconjunto XS, L e XXL como demonstrado em Figura 25. É claro, uma cereja é maior que uma amora, mas ambas são definitivamente menores que maçãs ou laranjas. E os melões são definitivamente maiores do que as laranjas, que são de tamanho semelhante ao das maçãs.

Figura 25. Redução de T-Shirt Sizing

Em Planning Poker, os desenvolvedores estimam os itens do backlog com base em um conjunto de cartas com números inspirados na sequência Fibonacci, representando o tamanho relativo (cf. Figura 26).

Se você concordou com uma história de referência de tamanho médio, por exemplo 5 pontos de história, a equipe agora decide o tamanho de outros itens de atraso com relação à história de referência. Depois de todos terem selecionado secretamente uma carta de pôquer, eles olham para os valores: se houver três "5" e dois "3" na mesa, então o item é marcado como um "5" e assim por diante. Se os números se desviarem uns dos outros, então os membros da equipe com a estimativa mais baixa e a mais alta discutem a lógica e as suposições por trás deles. Não há nenhuma tentativa de convencer em nenhum caso. Também se deve evitar falar sobre os números em si. Afinal, os números resultam das suposições, que são, portanto, o tópico central de discussão. Em seguida, inicia-se a próxima rodada de estimativa. Se a equipe não chegar a um acordo sobre um valor comum em três rodadas, o requisito será enviado de volta ao Product Owner para esclarecimento.

Para as próximas iterações, você pode querer estar na faixa de 1 a 13. Um "20", "40" ou "100" é uma indicação para que o Product Owner refine esse item. Estes números não significam literalmente "20", "40" ou "100", mas "muito grande", "grande demais" e "enorme" – mas são pelo menos indicadores de "quanto grande demais" em comparação com os itens entre 1 e 13. Se a equipe não tem nenhum entendimento do valor, então eles devem escolher o "?" ao invés de expressar seu medo, escolhendo "100".

Alguns conjuntos incluem o "0" que geralmente significa "0": "Pare de falar; isto não é um esforço relevante e não vale a pena incluir no plano."

Figura 26: Cartas do Planning Poker

A vantagem do Planning Poker é que ele é uma técnica muito boa para equipes novas e inexperientes encontrarem suas estimativas, pois evita a fixação por membros individuais da equipe. A desvantagem é que isso consome muito tempo. [Nota: O livro "Thinking, Fast and Slow" (Pensando rápido e devagar), de D. Kahneman [Kahneman2013] oferece uma ótima introdução sobre ancoragem e outros efeitos psicológicos relacionados ao pensamento e ao julgamento.]

O T-Shirt Sizing ou o Planning Poker geralmente levam bastante tempo, já que cada item de backlog é discutido e estimado individualmente. Para superar essa desvantagem, as equipes mais experientes podem usar técnicas aprimoradas.

Uma simplificação da técnica de Planning Poker baseia-se nos mesmos princípios, mas usa uma maneira diferente de determinar a estimativa correta. Em vez de cada membro da equipe fazer uma estimativa pessoal, um conjunto de cartas de pôquer é espalhado em uma mesa e os requisitos de referência são colocados num "recipiente" correspondente representado pela carta de pôquer. Depois disso, os requisitos são selecionados pelos membros da equipe em uma abordagem de rodízio, em que eles podem colocar um novo requisito no "recipiente" correspondente ou reatribuir um requisito já colocado em outro recipiente. Se um requisito for reatribuído várias vezes, ele será removido e enviado de volta ao Product Owner. Essa abordagem é muito mais rápida, mas precisa de uma equipe experiente o suficiente para discordar das atribuições feitas por outros membros em vez de concordar facilmente ("fixação").

O próximo passo da evolução é normalmente chamado "Estimativa de Afinidade" ou "Estimativa de Barreira". É usado ao estimar um número maior de requisitos, por exemplo, para estimativas aproximadas na preparação do Planejamento de Lançamento. Diferentemente da abordagem anterior, os requisitos não serão atribuídos por meio de um rodízio, mas cada membro recebe um número de requisitos e os atribui silenciosamente aos "recipientes" representados pelo conjunto de cartas de pôquer (cf. Figura 27). Após a atribuição silenciosa, todos os envolvidos podem inspecionar os requisitos atribuídos e marcar os que forem questionados.

Normalmente isto leva a uma cota de 20–30% de requisitos que precisam ser discutidos e 70–80% que são aceitos por todos os membros da equipe.

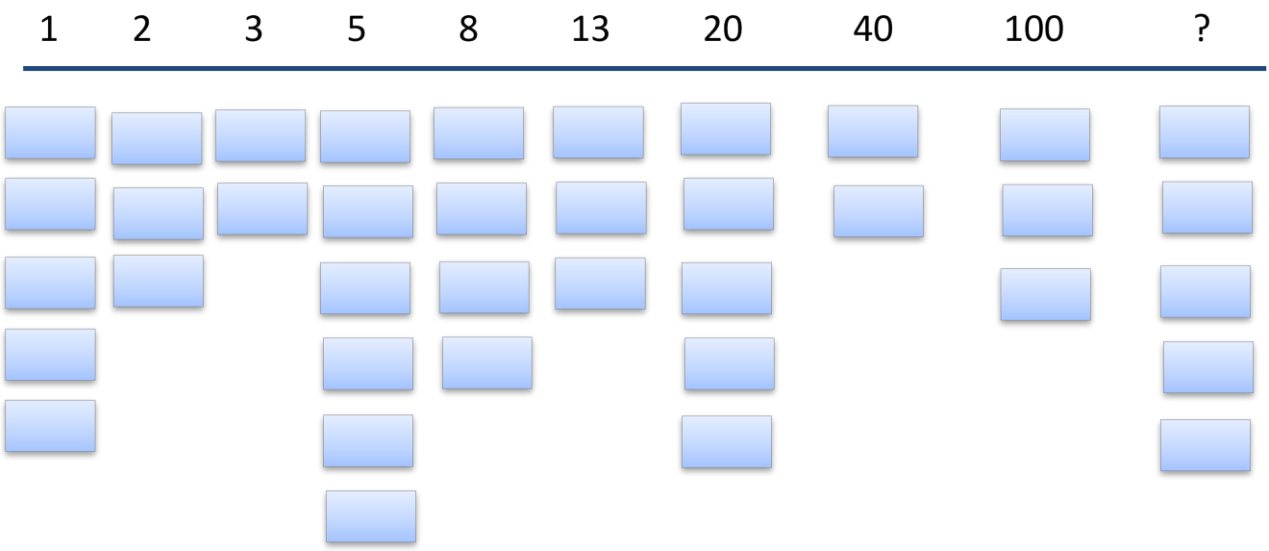


Figura 27: Estimativa de Barreira ou Estimativa de Afinidade

Algumas observações finais sobre a estimativa:

Dentro de uma equipe, o processo de estimativa mudará com o tempo. A equipe aprenderá com base nos resultados das iterações concluídas e alterará sua definição de pronto para incluir regras mais precisas.

Embora as estimativas relativas tenham muitas vantagens e funcionem bem dentro de uma equipe (como discutido anteriormente), há alguns inconvenientes quando se trata de estimativas além dos limites da equipe. Isto será discutido no capítulo 6 (Escala).

Sugestões para o exercício:
Escolha um estudo de caso e use uma maneira rápida para estimar o tamanho dos itens de backlog. Discuta suas descobertas, especialmente discutir o que funcionou e o que não funcionou ao fazer as estimativas.

5.5 Escolhendo uma estratégia de desenvolvimento

Diferentes estratégias podem ser aplicadas ao selecionar o que deve ser escolhido para lançamentos antecipados, com base no valor conhecido, risco e esforço necessários para desenvolver um item em atraso. Dois conceitos são típicos para um desenvolvimento Ágil: desenvolver um produto mínimo viável (MVP) e desenvolver um produto mínimo comercializável (MMP).

Produto Mínimo Viável

Um produto mínimo viável é a versão de um novo produto que permite a uma equipe coletar a quantidade máxima de aprendizagem validada sobre os clientes com o menor esforço.

O termo foi cunhado por Frank Robinson em 2001 e popularizado por Steve Blank, e Eric Ries [Ries2011].

Reunir insights de um MVP é muitas vezes menos caro do que desenvolver um produto com mais funcionalidades. O desenvolvimento de um produto com mais características aumentará os custos e riscos se o produto falhar, por exemplo, devido a suposições incorretas.

O MVP é uma ideia chave da metodologia Lean Startup desenvolvida por Eric Ries, que se baseia no ciclo Build-Measure-Learn (ver Figura 28).

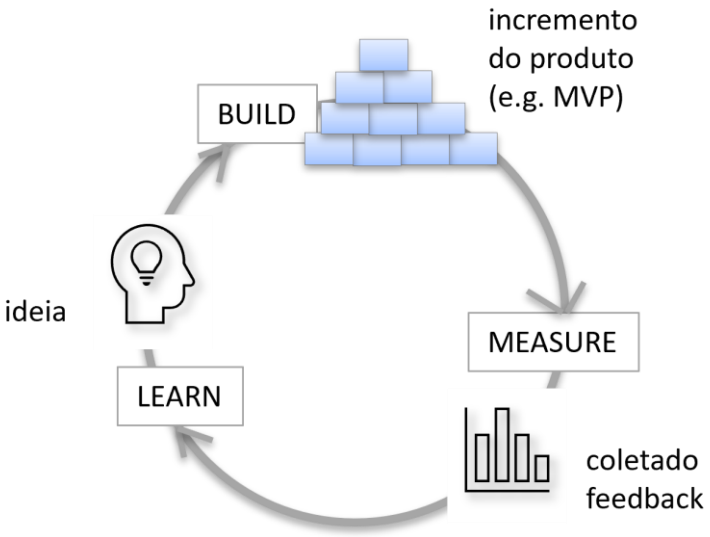


Figura 28: O ciclo "Build-Measure-Learn" de desenvolvimento enxuto

Assim, um MVP é um veículo de aprendizado, permitindo que você teste uma ideia dando rapidamente aos seus participantes-alvo algo tangível que lhe permita coletar dados dos quais você possa obter insights sobre seu mercado-alvo.

Roman Pichler [Pichler2016] observa que "O MVP é chamado de mínimo, pois você deve gastar pouco tempo e esforço para criá-lo". Mas isto não significa que tenha de ser rápido e mal feito. O tempo necessário para criar um MVP e o quão rico em recursos deve ser, depende de seu produto e do mercado. Mas, tente manter o conjunto de recursos o menor possível para acelerar o aprendizado e evitar desperdiçar tempo e dinheiro – sua ideia pode se revelar errada!

O MVP não é necessariamente um produto de software implementável. Algumas vezes, protótipos de papel e maquetes clicáveis podem ser usados para obter insights, desde que ajudem a testar a ideia e a adquirir os conhecimentos relevantes.

Para o sistema iLearnRE um MVP poderia ser apenas a publicação de vídeos de introdução e resumo para cada objetivo de aprendizado, a fim de obter insights sobre o comportamento do usuário e o aceite da UI.

Produto Mínimo Comercializável

O próximo passo deve ser criar um produto mínimo comercializável (MMP). Ela se baseia na ideia de que menos é mais: O MMP descreve o produto com o menor conjunto possível de funcionalidades que atendem às necessidades dos usuários iniciais (inovadores e primeiros usuários) e que, portanto, pode ser comercializado. Estudos demonstraram que a maioria de nossos produtos de software contém muitas funcionalidades que nunca ou raramente são utilizadas. Portanto, parece senso comum concentrar-se em funcionalidades que são populares para a maioria de seus participantes e retardar características que não são consideradas tão populares. Descobrir estas funcionalidades não é simples, mas os MVPs são uma excelente maneira de atingir este objetivo. Talvez alguns de seus MVPs sejam protótipos descartados, criados apenas para fins de aprendizagem. Mas se você fizer isso corretamente, você os desenvolverá de forma que possam ser reutilizados ou transformados no primeiro MMP.

Se você combinar estes dois conceitos, você tem uma estratégia que é mostrada em Figura 29. Desenvolver um par de MVPs para testar o mercado e obter dados reais como feedback. Em seguida, decida o número mínimo de funcionalidades que um produto tem que ter para ser útil para pelo menos um grupo chave de seus stakeholders. Então você continuamente acrescenta funcionalidades que prometem mais valor de negócio.

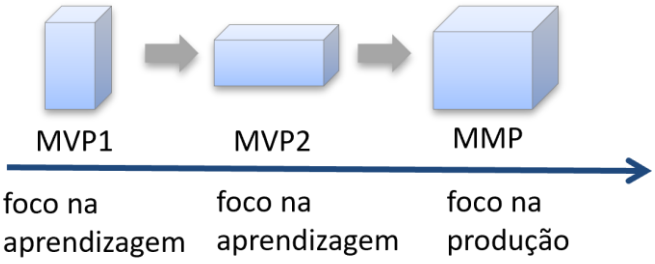


Figura 29: Combinando MVP e MMP

Redução de Risco

O desenvolvimento de MVPs está muito próximo da ideia de uma estratégia de redução de risco. A maioria das vezes os MVPs são desenvolvidos para reduzir o risco de ter as funcionalidades erradas para seus stakeholders. Mas você também pode criar MVPs (ou spikes) para reduzir os riscos técnicos. É melhor falhar rapidamente (seja em funcionalidade ou em tecnologia) do que desenvolver um produto completo e depois descobrir que ele não tem sucesso no mercado.

Em nosso estudo de caso iLearnRE, testar a performance da plataforma de vídeo planejada sob carga pode ser uma versão antecipada viável.

Frutos fáceis de colher ou ganhos rápidos

O oposto de uma estratégia orientada para o risco é buscar primeiro os frutos mais fáceis de colher. Comece publicando recursos que são fáceis e rápidos de implementar a fim de gerar negócios antecipados – ganhando algum dinheiro que lhe permite investir em recursos mais complexos. Mas cuidado com o adiamento de partes com risco, pois elas podem arruinar a estrutura de um produto baseado em resultados de baixo custo.

O aviso do Professor Kano

O professor Kano realizou estudos sobre a satisfação do cliente em relação às funcionalidades entregues. Como já incluído no syllabus CPRE Foundation Level [IREB2022] você deve ser capaz de distinguir três categorias de requisitos: fatores básicos (também conhecidos como insatisfatórios), fatores de performance (também conhecidos como satisfatórios) e fatores de empolgação (também conhecidos como excitantes ou encantadores).

Kano adverte que todo lançamento bem-sucedido de um produto deve incluir funcionalidades das três categorias. Quando você regularmente fornece apenas fatores básicos, seus clientes ficarão insatisfeitos. Você tem que incluir alguns fatores de performance, por exemplo, funcionalidades que os clientes pedem explicitamente, mesmo que não sejam absolutamente necessários. E você também deve tentar inovar, incluindo funcionalidades que eles não pediram, mas que os encantarão assim que os receberem.

Criar tal mistura de funcionalidades para cada lançamento é difícil de se conseguir. Esta é a razão pela qual você deve testar continuamente seu mercado com MVPs como mencionado acima e coletar dados reais antes de avançar em direção ao desenvolvimento de recursos demorados e caros.

WSJF

Outra estratégia interessante para o desenvolvimento é a abordagem Weighted Shortest Job First (WSJF). Baseia-se na relação entre o custo do atraso e o esforço estimado para o desenvolvimento [Reinertsen2008].

$$WSJF =$$

O custo do atraso é muito mais do que o benefício (valor de negócio) se o respectivo requisito for desenvolvido. Também inclui a perspectiva do que acontece se o requisito não for desenvolvido (p. ex., perda de participação de mercado, penalidades contratuais) ou se o seu desenvolvimento reduzirá o risco para toda a implementação (prova de conceito) ou abrirá uma nova oportunidade (p. ex., o uso de estruturas que diminuirão o esforço de desenvolvimento no futuro).

O WSJF pode ajudar a determinar quais requisitos (ou quais partes) devem ser desenvolvidos primeiro sem conhecer todos os detalhes exatamente usando apenas as relações entre os requisitos relativos ao Custo de Atraso e Duração (esforço de desenvolvimento).

Item de Backlog	Valor de Negócio		Criticidade do tempo		RR / OE		CoD		Duração		WSJF
Item 1	8	+	5	+	13	=	26	/	2	=	**13**
Item 2	1	+	13	+	1	=	15	/	3	=	**5,0**
Item 3	8	+	1	+	8	=	17	/	2	=	**8,5**
Item 4	13	+	8	+	5	=	26	/	1	=	**26**
Item 5	5	+	2	+	2	=	9	/	3	=	**3,0**

Figura 30: Exemplo WSJF

A tabela é construída da seguinte forma:

- Preencher a coluna com os itens/requisitos que serão classificados (em nosso exemplo itens 1 – item 5)
- Preencha as colunas (exceto CoD) da esquerda para a direita, por coluna:
  - Valor do Negócio – que valor é agregado se o item for desenvolvido?
  - Criticidade de Tempo – que valor se perde se o item não for desenvolvido?
  - RR (Risk Reduction) / OE (Opportunity Enablement) – quanto risco pode ser reduzido ou quantas oportunidades podem ser aproveitadas se o item estiver sendo desenvolvido?
- Encontre por coluna o elemento que tem o MENOR valor por coluna e atribua-lhe um "1"
- Avalie todos os outros itens da coluna como um fator em relação ao "1" (você pode usar qualquer número, mas o uso da sequência de Fibonacci é uma boa prática)
- Calcular o Valor de CoD como uma soma das colunas anteriores
- Calcular o WSJF como a proporção de CoD / Duração

O item com a maior relação WSJF deve ser desenvolvido primeiro seguido pelo item com a segunda maior relação e assim por diante.

Usar essa abordagem normalmente significa que os grandes blocos serão desenvolvidos mais tarde, já que os grandes blocos normalmente têm uma proporção baixa. Portanto, a sugestão ao Product Owner é dividir grandes pedaços e identificar aquelas peças que oferecem alto valor para, respectivamente, baixo esforço e adiar ainda mais as peças menos valiosas.

5.6 Sumário

Prioirzar o backlog é um processo iterativo de duas etapas. Como Product Owner, você fará a priorização do backlog com base no valor de negócio durante o primeiro passo. Você já viu várias maneiras de definir o que significa o valor de negócio em sua organização. Como Product Owner, você não deve subestimar os riscos. Às vezes é preciso equilibrar valor com risco para não colocar em risco o desenvolvimento de seu produto. O valor pode ser expresso em várias escalas como MoSCoW, ou Alto, Médio e Baixo. Ou você simplesmente coloca todos os itens em uma sequência linear com base em seu valor. Assim não é necessário usar números.

O segundo passo é que os desenvolvedores lhe forneçam estimativas para cada item de backlog. O Ágil fez muitas coisas para tornar o processo de estimativa menos ameaçador:

- As pessoas certas (aquelas que fazem o trabalho) estimam.
- A estimativa é feita como um exercício de grupo, não por uma única pessoa.
- A estimativa deve ser feita relativamente; comparar o tamanho e o esforço dos itens em vez de dar-lhes um valor absoluto.

Vários processos podem ser usados para estimar, como o Shirt Sizing ou o uso de Fibonacci no Planning Poker. Para acelerar o processo, pode-se utilizar a Estimativa de Barreira ou Estimativa de Afinidade.

Quando os itens do backlog são suficientemente pequenos e bem compreendidos, as estimativas serão precisas para permitir o planejamento da iteração. Quando os itens ainda forem muito grandes ou não forem totalmente compreendidos, a equipe indicará isso com um valor mais alto, dando um recado ao Product Owner de que esses itens precisam de esclarecimento e/ou refinamento.

Assim que os itens forem estimados, o Product Owner poderá alterar a ordem do backlog mais uma vez, por exemplo, trocar um grupo de itens mais baratos por um item mais caro.

Com base no valor determinado e nas estimativas, várias estratégias diferentes podem ser aplicadas para determinar a sequência na qual os itens devem ser atribuídos às iterações. Estratégias como a criação de uma série de produtos mínimos viáveis (MVPs), seguidos por um produto mínimo comercializável (MMP) antes de adicionar cada vez mais características apoiam o princípio Ágil de entrega antecipada e entrega frequente. Mas também colher os frutos mais fáceis ou reduzir o risco no início são alternativas viáveis.

Uma organização pode adotar uma estratégia de ganho antecipado de valor de negócio, por exemplo, se sua meta principal for entregar um produto com antecedência e estabelecer participação no mercado. Uma estratégia de redução antecipada de riscos pode ser preferida se um fornecedor quiser evitar a todo custo que um produto seja devolvido devido, por exemplo, a um desempenho ou segurança inadequados.

6 Escalando o RE@Agile

A Engenharia de Requisitos é mais fácil para produtos que são pequenos o suficiente para serem manuseados por uma única equipe em um único local. Todos os capítulos até agora fizeram essa suposição de forma implícita: mostramos como os requisitos mais importantes (ou seja, os que proporcionam o maior valor de negócio) podem ser implementados por essa equipe sem a necessidade de distribuir os requisitos entre várias equipes (de desenvolvimento). Quando essa suposição não se sustenta mais – ou seja, quando precisamos de mais de uma equipe para alcançar nossos objetivos e visões de negócios – precisamos considerar o escalonamento do nosso desenvolvimento.

Neste capítulo, discutimos por que o desenvolvimento de produtos às vezes precisa ser escalonado e porque produtos precisam ser desenvolvidos por mais de uma equipe, seja no mesmo local ou distribuída geograficamente. Ao escalar, é provável que o Product Owner do produto geral (como o papel responsável pela gestão de requisitos) enfrente desafios mais relacionados a aspectos de gerenciamento do que a aspectos de requisitos. Discutiremos como os dois fatores, time-to-market e complexidade (seja complexidade funcional ou requisitos de qualidade desafiadores), justificam e impulsionam o processo de escalonamento. Mas, as restrições organizacionais e técnicas também influenciarão a forma como escalamos.

Neste capítulo, abordaremos os seguintes aspectos:

- O que significa escalonamento e como ele afeta os requisitos e as equipes (capítulo 6.1)?
- Como (re)organizamos os requisitos e as equipes expressivas (capítulo 6.2)?
- Como são definidos e utilizados os lançamentos e roteiros no planejamento a longo prazo (capítulo 6.3)?
- Como os requisitos são validados em ambientes dimensionados (capítulo 6.4)?

6.1 Escalonando Requisitos e Equipes

Usamos o termo escalonamento para descrever uma mudança no tamanho, seja do sistema ou do produto, ou do número de pessoas envolvidas.

Desde 2010, várias estruturas de escalonamento Ágil foram desenvolvidas para resolver esses problemas. Entre eles estão: Nexus [Nexus Guide], SAFe [SAFe1] [SAFe2], LeSS [LeSS], Scrum@Scale [S@S Guide], BOSSA Nova [BOSSANOVA], Scrum of Scrums [SofS], Spotify [Spotify2012], embora existam outros. As estruturas de escalonamento variam em seu nível de maturidade, o número de boas práticas, diretrizes e regras, e o grau de adaptabilidade às necessidades específicas de uma organização. Não discutiremos cada estrutura em detalhes, mas as usaremos como exemplos, especialmente quando apresentam abordagens alternativas para lidar com os requisitos do grande público.

Em Figura 31 são mostradas as forças motrizes para o escalonamento, bem como as restrições que podem ser encontradas no caminho.

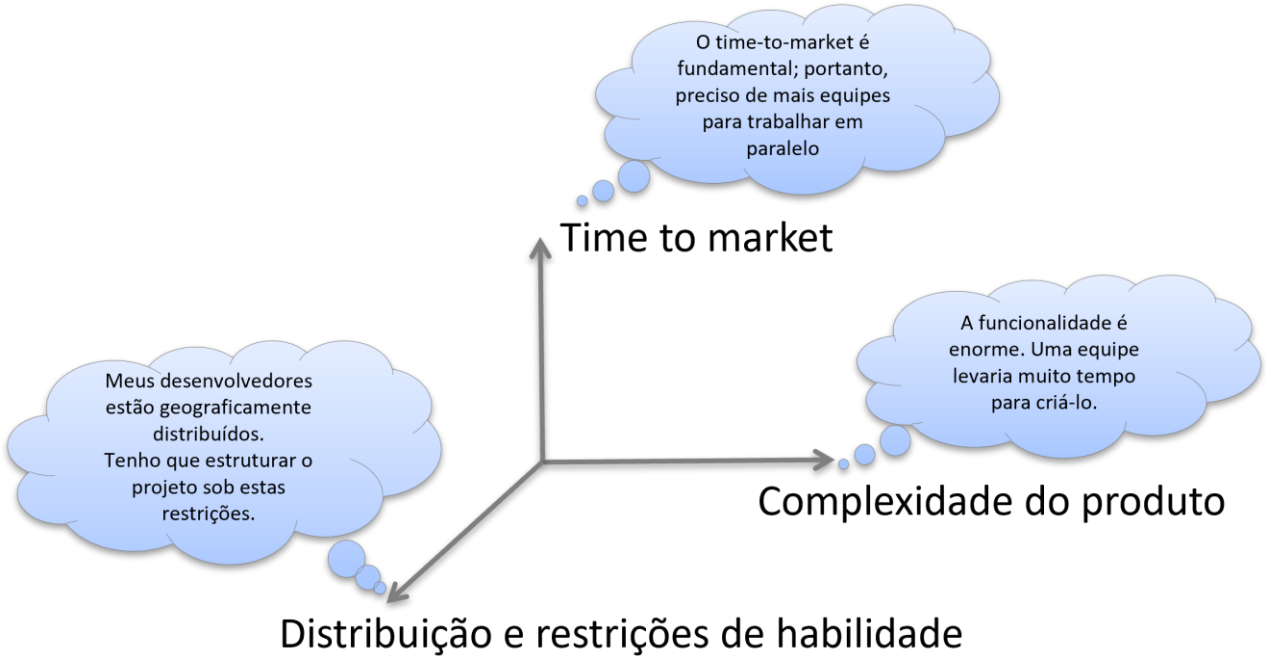


Figura 31: Três dimensões que podem desencadear escalonamento

As duas primeiras dimensões na figura acima são:

- Time-to-market: Uma equipe levaria muito tempo para implementar todos os requisitos necessários para um produto satisfatório. A fim de acelerar a liberação, você coloca várias equipes para trabalhar.
- Complexidade do produto: O domínio do produto ou as tecnologias utilizadas para a implementação são tão complexas que uma equipe não pode lidar com todos os aspectos. Portanto, você decide trabalhar com várias equipes, cada uma focando em diferentes aspectos do produto.

Em ambos os casos, você é imediatamente confrontado pelo fato de ter que coordenar o trabalho de mais de uma equipe. Isto torna o desenvolvimento mais difícil em comparação com o trabalho com uma única equipe alocada.

Há uma terceira dimensão mostrada na figura acima:

- Você pode ter que trabalhar com várias equipes por razões organizacionais ou políticas: você pode ter pessoas em diferentes localizações geográficas ou trabalhando em várias empresas, ou equipes organizadas em torno de conjuntos específicos de habilidades especializadas. Consideramos todos estes aspectos como restrições que às vezes não podem ser evitadas, embora não recomendaríamos necessariamente a escolha destas estruturas organizacionais onde elas ainda não estão presentes. Mais sobre critérios bons e ruins para estruturação de equipes no capítulo 6.2.

No entanto, tenha cuidado com o escalonamento quando não for absolutamente necessário: trabalhar com mais de uma equipe sempre introduz sobrecarga de comunicação e coordenação.

Portanto, se os motivos para escalonamento mencionados acima não se aplicarem, provavelmente você não deve escalar de forma alguma!

Se, no entanto, você optar por escalar, duas coisas sempre serão verdadeiras: você será obrigado a adicionar hierarquia aos requisitos e hierarquia à organização. São necessários requisitos de granularidade grosseira ao discutir o produto como um todo; serão necessários requisitos de granularidade refinada nas equipes que implementam algum aspecto do produto. E as próprias equipes precisarão organizar sua cooperação para funcionar com sucesso dentro de uma equipe maior.

Como diferentes estruturas de escalonamento abordam esses dois aspectos e que terminologia eles sugerem para hierarquias de requisitos e hierarquias de equipes é discutido nos capítulos seguintes.

6.1.1 Organização de requisitos em larga escala

No capítulo 3 discutimos o tema da granularidade dos requisitos e introduzimos os termos *requisitos de granularidade grosseira*, *requisitos de granularidade média* e *requisitos de granularidade refinada*. Deliberadamente escolhemos esta terminologia mais geral, uma vez que as estruturas de escalonamento (e ferramentas ágeis de requisitos) diferem significativamente nos termos específicos que utilizam.

A representação hierárquica dos requisitos reflete uma das ideias-chave do backlog de produto: requisitos de granularidade grosseira ainda podem ser vagos ou imprecisos até que eles (ou partes deles) se tornem relevantes para uma próxima iteração e, portanto, precisam de mais detalhes e precisão. Assim, são elaborados requisitos mais refinados e é mantida uma relação com seus pais maiores. A hierarquia resultante cumpre duas finalidades:

- Ele fornece uma visão geral de todos os requisitos conhecidos.
- Ele permite o detalhamento seletivo dos elementos que provavelmente serão mais desenvolvidos em breve.

Figura 32: Terminologia para requisitos em diferentes níveis de granularidade em métodos e ferramentas selecionadas

Para os fins deste handbook, o IREB escolheu um dos conjuntos de termos mais populares para requisitos em diferentes níveis de granularidade, que contém três termos: Épicos (para requisitos de granularidade grosseira), Funcionalidades (granularidade média) e Histórias de Usuários (granularidade refinada).

Algumas estruturas e ferramentas de dimensionamento não dão nomes explícitos aos diferentes níveis de requisitos, mas simplesmente os chamam de *itens de backlog* e permitem seu refinamento até que sejam pequenos o suficiente para serem implementados em uma única iteração.

Outras ferramentas começam com uma abordagem de dois níveis, mas depois permitem que o número de níveis seja ampliado. O Jira da Atlassian, por exemplo, usa épicos e histórias como padrão, mas permite que essa hierarquia seja ampliada (versões recentes sugerem chamar os maiores requisitos de temas e o próximo nível de *iniciativas*). O LeSS chama os requisitos no nível acima das histórias de usuário de *funcionalidades* e no nível mais alto de *áreas de requisitos*.

O framework SAFe oferece um metamodelo de requisitos abrangente [SAFeMDM] com quatro níveis de requisitos e um esquema de nomenclatura rigoroso: épicos, funções, funcionalidades e histórias de usuários. Figura 33 mostra uma versão simplificada deste metamodelo. A distinção entre os níveis não se baseia tanto no conteúdo, mas sim no tamanho.

Uma história deve ser pequena o suficiente para caber em uma iteração (ou sprint); uma funcionalidade deve ser pequena o suficiente para caber em uma versão. As funções e os épicos são tão grandes que abrangem mais de uma versão (mais sobre o planejamento de versões no capítulo 6.3).

Observe que, em cada nível, o SAFe distingue as funcionalidades de negócio – aqueles que criam valor de negócio – das funcionalidades facilitadoras – os pré-requisitos estruturais necessários sem os quais o valor de negócio não pode ser alcançado. Discutiremos esta distinção com mais detalhes no capítulo 6.2.3.

SAFe também usa termos específicos para os critérios de aceite em diferentes níveis de granularidade, como mostrado abaixo.

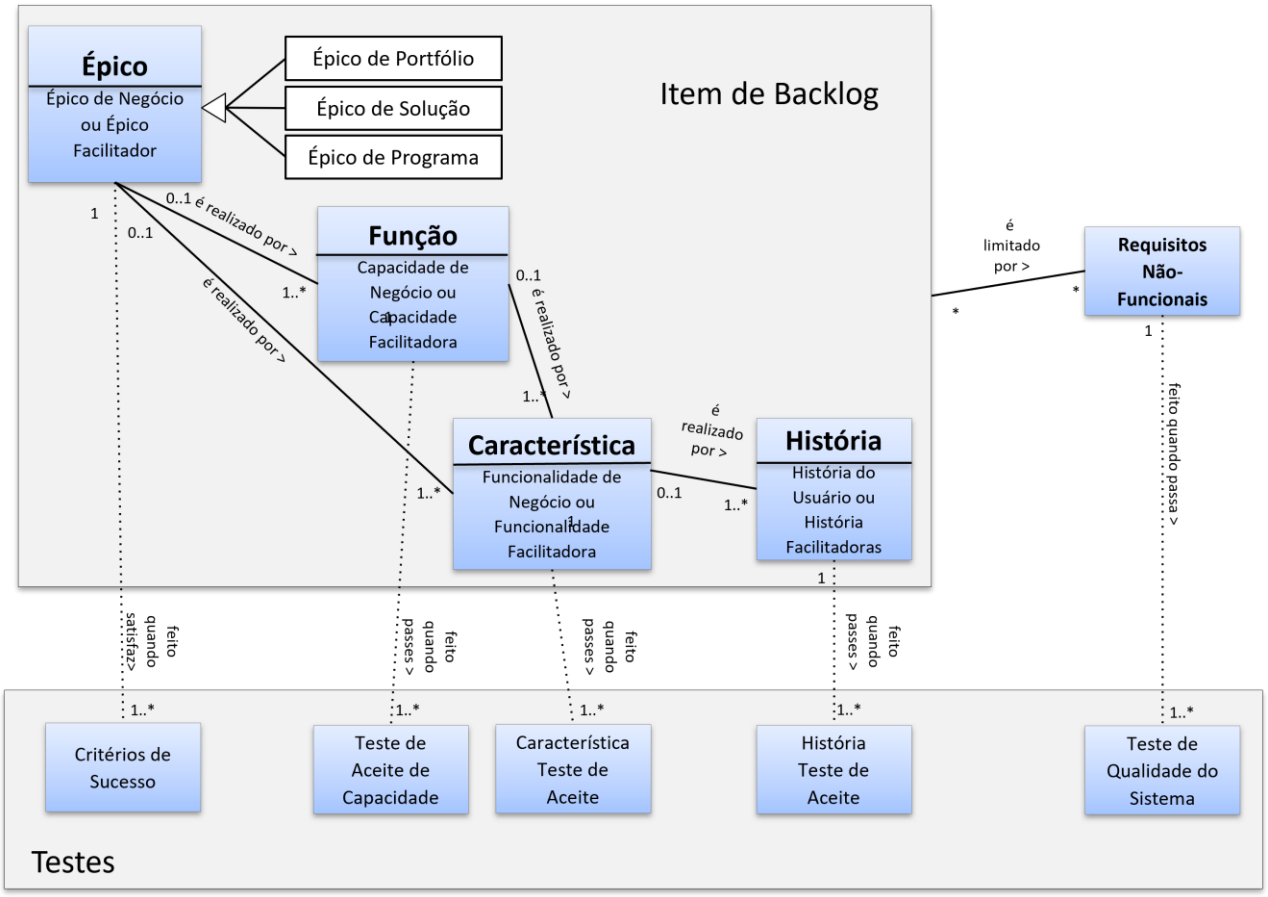


Figura 33: Terminologia de requisitos de SAFe

Embora muitas das ferramentas de requisitos Ágeis atuais não sejam capazes de lidar com os quatro níveis de granularidade desse metamodelo de forma imediata, a maioria delas fornece os meios para personalizar a hierarquia.

Para evitar longas discussões sobre terminologia (e guerras de metodologia entre equipes!), sugerimos que escolha uma terminologia interna para os níveis de granularidade que deseja usar e, em seguida, mantenha essa terminologia em todos os demais projetos. Muitas vezes, seja a estrutura de escalonamento ou as ferramentas que você utiliza, vão ditar a terminologia.

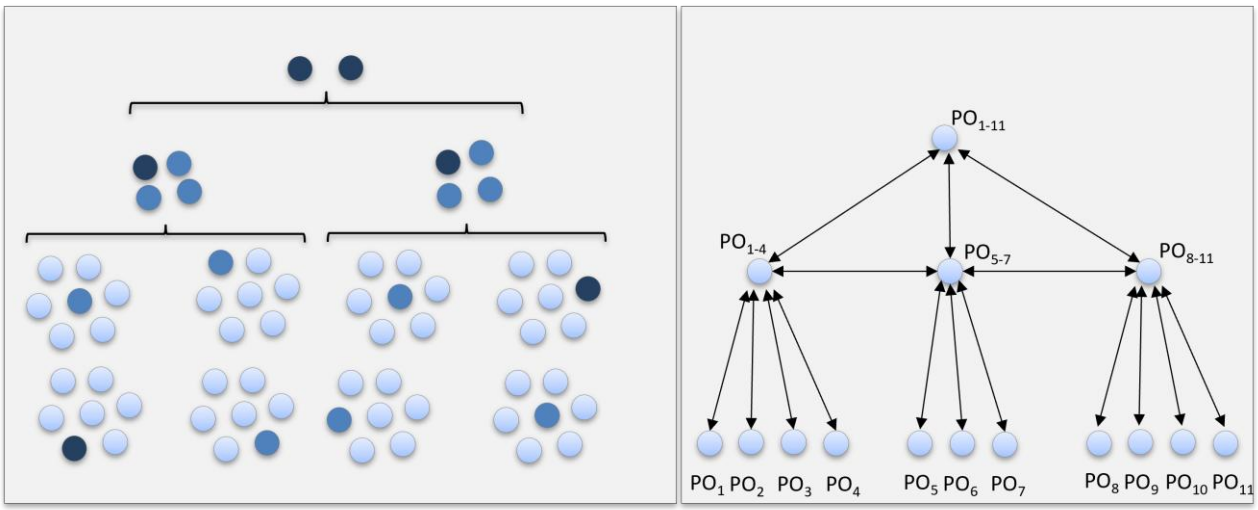
6.1.2 Organização das Equipes

Todas as estruturas de escalonamento concordam que ...

- ... independentemente dos cargos específicos, a responsabilidade é necessária em todos os níveis da organização.
- ... o trabalho tem que ser devidamente coordenado entre as equipes.

Além desses pontos gerais, no entanto, os conceitos e a terminologia diferem em abordagens específicas.

Quando Scrum é usado para várias equipes, uma técnica frequentemente usada para coordenar essas equipes é chamada scrum of scrum. [SofS] A única diferença para o trabalho dentro de uma equipe é que cada equipe escolha uma pessoa (um embaixador) para representá-la em reuniões de coordenação que normalmente acontecem duas ou três vezes por semana. Durante o curso de um projeto, a equipe pode indicar diferentes pessoas, escolhendo as que melhor represente de acordo com os tópicos em discussão.



Scrum of Scrums para coordenação de desenvolvimento

Hierarquia de responsabilidade pelas requisitos

Figura 34: Scrum of Scrums como modelo para organizar a responsabilidade pelos requisitos

Além da coordenação geral dos desenvolvedores, a hierarquia de requisitos discutida no capítulo 6.1.1 precisa de uma hierarquia correspondente de responsabilidade de requisitos (Figura 34, à direita). Requisitos de granularidade grosseira e média devem ser da propriedade de alguém, trabalhos de refinamento devem ser atribuídos a equipes individuais e as dependências entre as equipes devem ser identificadas.

A organização dos papéis em diferentes níveis da hierarquia organizacional difere entre as estruturas: desde a democracia básica até estruturas claramente hierárquicas.

Entre as abordagens mais democráticas estão Nexus e BOSSA Nova. Eles não sugerem ter *hierarquias de PO*. Para essas duas estruturas o Product Owner faz parte da equipe e a equipe decide como coordenar não apenas o desenvolvimento, mas também os requisitos. Assim, o Nexus se aproxima da ideia de um scrum of scrums (ou seja, equipes auto-organizadas) com sua *Equipe de Integração Nexus*, que existe para coordenar, treinar e

supervisionar a aplicação do Nexus e a operação do Scrum para que os melhores resultados sejam derivados. A Equipe de Integração Nexus é composta pelo Product Owner, um Scrum Master, e membros da Equipe de Integração Nexus. Mas note que a Equipe de Integração Nexus não é uma autoridade decisória: semelhante a um scrum master de uma equipe individual, a equipe de integração assegura principalmente que a comunicação necessária ocorra entre as equipes a fim de resolver problemas compartilhados.

Uma democracia ainda mais básica é defendida pela BOSSA Nova [BOSSANOVA]. Aqui, uma sociocracia [SOCIOCRACIA] é proposta como a forma ideal para a organização em geral. As equipes selecionam seus embaixadores no círculo de coordenação, e cada círculo de coordenação seleciona seu embaixador nos círculos de coordenação de nível superior, e assim por diante.

Outras estruturas estabelecem estruturas de gerenciamento de requisitos mais claras com autoridade decisória bem definida. Eles frequentemente atribuem cargos fixos aos coordenadores de requisitos em níveis mais altos. Como vimos acima com hierarquias de requisitos, a terminologia exata usada nas hierarquias organizacionais também varia entre as diferentes estruturas. Figura 35. dá uma visão geral de alguns dos títulos de cargos e nomes de papéis utilizados em estruturas selecionadas.

Framework / Requisitos Coordenação ...	Nexus	Scrum@Scale	SAFe	LeSS
... no Nível de Portfólio (organização geral)			Proprietário do ÉPICO	
... de Equipes de Equipes de Equipes		EMS (Meta Scrum Executivo)	Solução Painel de Controle	
... de Equipes de Equipes	Nexus Equipe de Integração	Chief Product Owner	Produto Painel de Controle	Product Owner
... em Nível de Equipe	Product Owner	Product Owner	Product Owner	Área do Product Owner

Figura 35: Responsabilidade de nomes de papéis para requisitos

Algumas frameworks (Scrum@Scale, Nexus, SAFe) reservam o nome do papel "Product Owner" para a equipe individual e propõem novos nomes para as funções de coordenação de nível superior. Scrum@Scale usa o termo "Chief Product Owner", por exemplo.

Em SAFe, o *Gerente de Produto* é responsável pela saída de várias equipes, que juntas formam um Agile Release Train. Onde vários Agile Release Trains trabalham em conjunto para atender aos requisitos de uma solução ainda maior, eles são gerenciados por um *Gerente de Soluções*. No maior nível de granularidade, agilidade corporativa, a *Epic Owners* tem a responsabilidade geral dos requisitos e juntos representam a *Gestão de Portfólio*.

LeSS vai no sentido oposto e afirma que mesmo para grandes equipes a responsabilidade é do Product Owner. As equipes individuais podem então designar *Product Owners de Área* para gerenciar os requisitos da parte do produto designada para equipes menores.

Você deve se lembrar: Os cargos não importam, desde que haja alguém (ou um pequeno grupo) que seja responsável pela gestão dos requisitos. Todas as estruturas sugerem trabalhar com um único backlog de produtos, independentemente do tamanho da equipe (ver mais detalhes sobre o backlog lógico no capítulo 6.2). Partes desse único backlog podem então ser atribuídas a subequipes.

Seja qual for o mecanismo utilizado, certifique-se de que as subequipes (ou seus representantes) se comuniquem regularmente sobre sobreposições, dependências e prioridades para obter o melhor resultado.

6.1.3 Organização de Ciclos de Vida/Iterações

Em nossa definição no capítulo 1.3 afirmamos que RE@Agile é um processo iterativo. Para projetos grandes, a maioria dos frameworks de dimensionamento sugere dois tipos diferentes de iterações:

- Iterações curtas (frequentemente chamadas sprints): onde os desenvolvedores individuais tentam implementar os itens backlog alocados na reunião de planejamento de sprints. Estas pequenas iterações normalmente duram entre duas e quatro semanas.
- Iterações mais longas (muitas vezes chamadas de releases): destinadas principalmente a garantir a integração dos resultados de múltiplas equipes. Os releases podem conter uma série de iterações curtas. Diferentes frameworks estabelecem regras diferentes para a frequência de integração, desde a integração em cada iteração até a integração pelo menos em cada versão. As iterações de release não devem durar mais de dois a três meses.

Para saber mais sobre planejamento de release e roadmap, consulte o capítulo 6.3.

6.2 Critérios para a estruturação de Requisitos e Equipes expressivas

No desenvolvimento de produtos em larga escala, a maioria das equipes tem que trabalhar juntas no mesmo produto. Na prática, cada equipe desenvolve uma parte específica do produto que deve ser integrada com outras partes para construir uma solução de trabalho. Somente o produto integrado tem valor para os stakeholders.

Quando escalamos o desenvolvimento de produtos para várias equipes, não é suficiente para todos os Product Owners simplesmente se encontrarem e discutirem de alguma forma qual equipe deve desenvolver qual parte do produto, e então esperar pelo melhor resultado! São necessárias estruturas e práticas sofisticadas para apoiar a colaboração da equipe, gerenciar as mudanças de requisitos e permitir a entrega rápida do produto. Caso contrário, os desenvolvedores podem desperdiçar esforços coordenando com equipes que não são relevantes para seu trabalho.

Do ponto de vista dos requisitos, temos que fechar o ciclo: desde o requisito inicial (negócio) exigido pelos stakeholders, passando pela divisão de requisitos complexos em partes menores gerenciáveis pelos desenvolvedores e, em seguida, garantindo que os resultados reunidos se combinem para formar uma solução que possa ser liberada para a empresa.

6.2.1 Backlog com foco no produto

Os Product Owners precisam de um entendimento compartilhado do produto e de seu contexto de negócio. Isso é importante, pois eles precisam trabalhar de forma colaborativa nos requisitos em diferentes níveis de abstração e concordar com as prioridades de cada equipe, que também devem refletir os profissionais de negócios em geral. Além disso, as equipes em um ambiente ágil devem identificar as sobreposições e dependências de requisitos para minimizar as interrupções durante o desenvolvimento.

Para apoiar este tipo de foco de produto, os requisitos devem ser gerenciados usando um backlog lógico. A ideia principal é que cada requisito seja mantido em um único lugar, evitando redundâncias e contradições. Isso ainda pode ser alcançado mesmo quando se subdivide o backlog em backlogs de equipe, conforme ilustrado em Figura 36.

Enquanto refinam os requisitos de granularidade grosseira, os Product Owners podem trabalhar em itens do backlog ainda não associados a nenhuma equipe (veja (a) na Figura 36) ou podem dividir requisitos complexos e entregar os itens do backlog resultantes às equipes para refinamento adicional (veja (b) e (c) na Figura 36). Para garantir a rastreabilidade entre os requisitos em diferentes níveis de abstração, os Product Owners devem vincular os itens do backlog.

Por exemplo, considerando um requisito complexo que descreve a conexão de um dispositivo de hardware especializado com um aplicativo de computador usando um protocolo proprietário. Este requisito é armazenado inicialmente no backlog de produto (ver (a) em Figura 36). Assumindo que as equipes A e B desenvolvem o sistema, enquanto a equipe A tem experiência com o dispositivo de hardware. Assim, o requisito complexo pode ser dividido em um requisito menor que se concentra na interface do dispositivo de hardware, que é gerenciado no backlog da Equipe A, e outro requisito que descreve o manuseio da conexão dentro do aplicativo (consulte (c) na Figura 36), que é gerenciado no backlog da Equipe B.

Dependendo da ferramenta usada para o gerenciamento de backlogs, é possível definir filtros de equipe no backlog comum do produto ou criar backlogs (virtuais) para cada equipe. Independentemente da ferramenta escolhida, todos os itens de backlog juntos formam um backlog lógico.

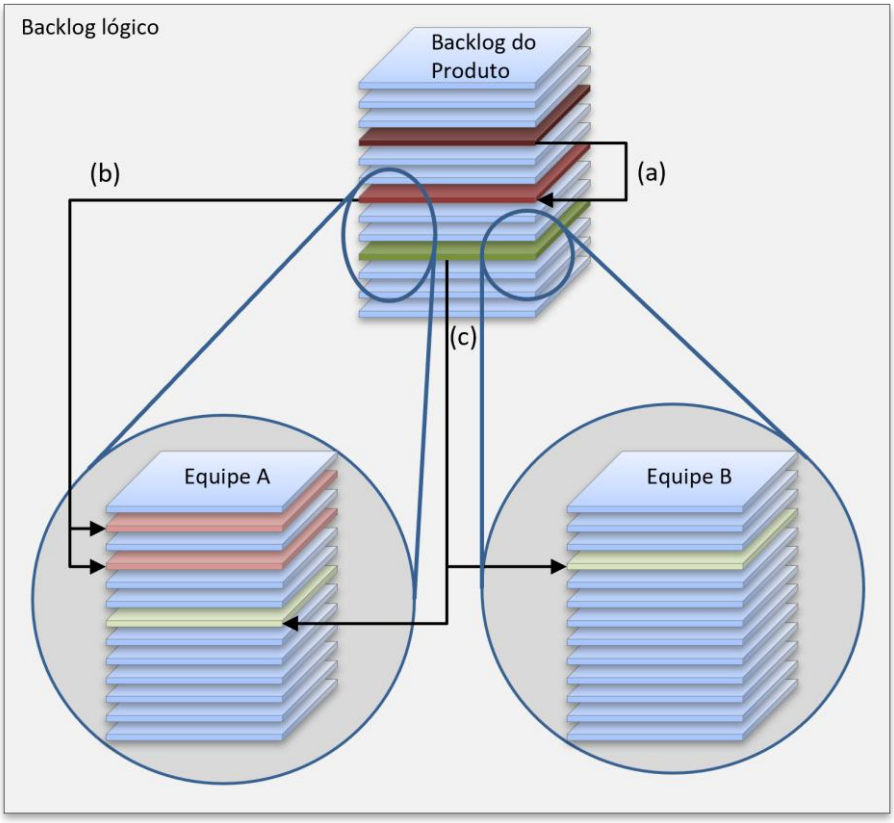


Figura 36: Ideia chave da abordagem lógica do backlog.

Em frameworks escaláveis como Nexus, SAFe e Less, recomenda-se também um backlog lógico do produto. Em SAFe, o backlog lógico é dividido em diferentes backlogs que são ligados de acordo com sua escala (p. ex., Backlog do Portfólio, Backlog da Solução, Backlog de Programa, vários Backlogs de Equipe). Cada backlog contém requisitos de granularidade apropriada de acordo com a escala.

Por exemplo, os itens do backlog do programa são refinados nos backlogs da equipe, enquanto os itens adicionais que surgem do contexto local da equipe também podem ser adicionados diretamente aos backlogs da equipe.

6.2.2 Equipes auto-organizadoras e tomada de decisões colaborativas

O desenvolvimento de produtos terá dificuldade para reagir às mudanças em tempo hábil se cada equipe depender de uma complicada rede de interações com outras equipes para aprovar qualquer decisão. É preciso uma estrutura de equipe que permita que elas se auto-organizem em torno da criação de valor: para responder melhor ao feedback dos stakeholders, para tomar decisões sensatas de forma independente e para fornecer funcionalidades de ponta a ponta [Anderson2020].

Os benefícios das equipes auto-organizadoras são um dos princípios do Ágil [AgileManifestoPrinciples]. A comunicação localizada e direta dentro das equipes (intra-equipe) permite otimizações e tomadas de decisão eficazes, enquanto a comunicação entre as diferentes equipes (entre-equipes) é mais lenta e deve, em geral, ser mantida a um mínimo [Reinertsen2008].

No entanto, sempre haverá a necessidade de colaboração em uma rede de equipes que trabalham para atingir uma meta compartilhada. É necessário um nível de comunicação e coordenação que, inevitavelmente, restringirá o nível de liberdade usufruído por cada equipe.

Para trabalhar nos requisitos de forma colaborativa e tomar decisões razoáveis de forma autônoma, as equipes precisam de um panorama geral dos requisitos das outras equipes com as quais precisam colaborar, sem, no entanto, ficarem sobrecarregadas com todos os detalhes. Os Product Owners devem, portanto, encontrar um nível adequado de detalhes, suficiente para que as equipes entendam o impacto de suas decisões sobre outras equipes.

6.2.3 Compreender a divisão dos requisitos baseadas em funcionalidades

A divisão dos requisitos é necessária no desenvolvimento Ágil para dividir os requisitos maiores em outros mais detalhados, que podem ser implementados em uma iteração. Como discutido no capítulo 3.4, existem diferentes técnicas de divisão que devem ser aplicadas no desenvolvimento Ágil, independentemente de quantas equipes estejam envolvidas. Mas a divisão de requisitos é muito mais fundamental no desenvolvimento de produtos em larga escala, pois permite a auto-organização de equipes que devem ser capazes de implementar os requisitos independentemente umas das outras.

Para fornecer incrementos de produtos que possam ser enviados com o mínimo de dependência de outras equipes, as equipes em um ambiente Ágil devem trabalhar em recursos de ponta a ponta com fraco acoplamento. Em nosso contexto, o termo "funcionalidades de ponta a ponta" refere-se a um conjunto de funções coerentes que executam uma tarefa específica que fornece valor de negócio aos stakeholders. Dependendo do nível de abstração em que a divisão está ocorrendo, no entanto, a definição de tarefas pode variar desde funções específicas do usuário até processos de negócio completos.

Para identificar as funcionalidades de ponta a ponta, os Product Owners devem decompor o escopo do produto em unidades de funcionalidade pouco agrupada e internamente consistentes (ou seja, limites funcionais), conforme representado em Figura 37.

Se o escopo for dividido de acordo com esses limites funcionais, os Product Owners atribuídos a uma unidade específica poderão trabalhar nos requisitos associados com um grau maior de independência. As equipes correspondentes costumam ser chamadas de equipes de funcionalidades [Larman2016].

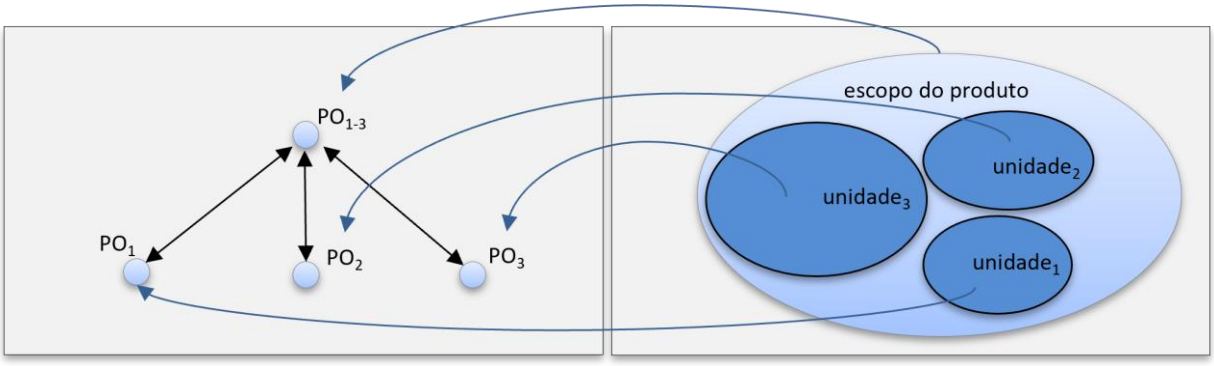


Figura 37: O escopo é dividido em unidades menores de funcionalidade ponta a ponta e compartilhado entre os Product Owners.

Um Product Owner, e geralmente uma equipe Ágil, são designados a uma unidade de funcionalidade de ponta a ponta. Os limites entre as unidades ajudam a estabelecer os caminhos de comunicação. Os limites devem ser claros a fim de permitir uma colaboração eficaz; os Product Owners podem se concentrar nos requisitos detalhados atribuídos a sua unidade em vez de gastar muito tempo tentando entender todo o escopo e contexto de negócio. Eles só têm que colaborar com outros Product Owners nos requisitos que afetam as unidades adjacentes. Os requisitos podem ser organizados hierarquicamente com base em unidades independentes, conforme discutido no capítulo 6.2.1.

A divisão do escopo de um produto pode ser feita de acordo com as linhas do processo de negócios, conforme discutido no capítulo 3.2. Se um processo de negócios consistir em várias linhas de processo, cada linha poderá ser apoiada por recursos de produtos de nível empresarial de ponta a ponta. Idealmente, as diferentes linhas de processo devem ser fracamente acopladas em um processo de negócios, o que geralmente permite que os Product Owners trabalhem de forma independente nos requisitos de seus recursos. Neste caso, eles só têm que concordar em funcionalidades que afetam a interação das linhas de processo.

Os casos de uso são uma abordagem para a estruturação dos requisitos, nem sempre tipicamente associados com Ágil, mas, no entanto, recomendados por vários autores (p. ex. Jacobsen, Cockburn, Leffingwell). Os casos de uso veem o sistema como uma caixa preta e consideram as ações que ocorrem entre um ator (humano ou outro sistema) e a solução.

Os casos de uso podem ser usados como parte das atividades iniciais para definir o escopo e estruturar um projeto, conforme discutido no capítulo 2.1.5.3, ou elaborados como parte do desenvolvimento contínuo do produto. Ao contrário das linhas de processo, um caso de uso pode ser visto no nível do usuário como uma funcionalidade ponta a ponta do produto. Os Product Owners devem concordar apenas com os requisitos que se relacionam a vários casos de uso (p. ex., interfaces ou entidades de negócio comuns).

6.2.4 Considerações quando não é possível a divisão de requisitos baseados em funcionalidades

Infelizmente, em muitos casos, não é tão fácil decompor os requisitos com base em unidades fracamente acopladas de funcionalidade ponta a ponta. Devido ao projeto estrutural (p. ex., tecnologia, infraestrutura, componentes do sistema, plataforma comum, camadas estruturais como front-end e back-end), bem como as considerações organizacionais (habilidades especiais, localização da equipe, subcontratados), as unidades de funcionalidade podem se sobrepor, conforme ilustrado na Figura 38. Isso significa que diferentes equipes em um ambiente ágil devem trabalhar juntas para implementar recursos específicos e seus respectivos Product Owners precisam colaborar mais de perto com relação aos requisitos [Figura 38]. Alternativamente, uma equipe dedicada pode ser estabelecida para trabalhar especificamente na sobreposição, e colaborar com cada uma das equipes originais focadas em uma unidade de funcionalidade.

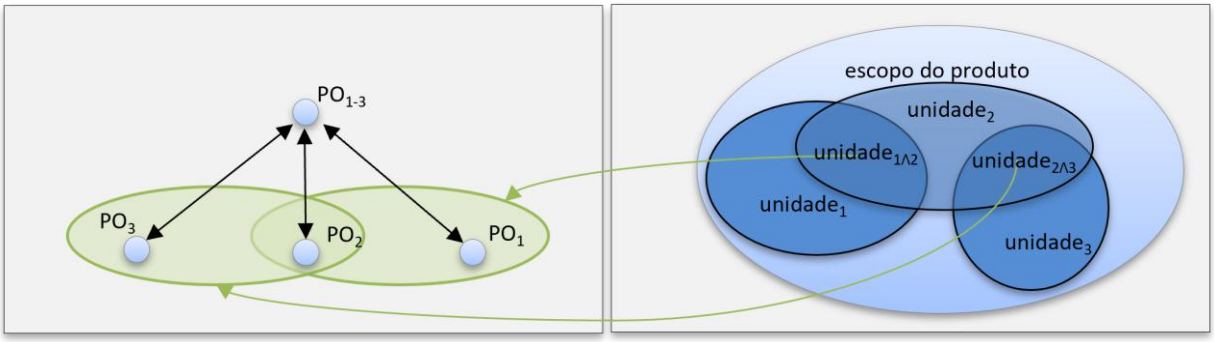


Figura 38: As unidades de interseção indicam uma estreita colaboração dos Product Owners no que diz respeito aos requisitos.

Para implementar recursos de forma colaborativa, as equipes em um ambiente ágil precisam de um entendimento compartilhado dos requisitos e do contexto de negócio. Elas também devem concordar com os requisitos sobrepostos (transversais), as restrições e as interfaces técnicas comuns para que os resultados de diferentes equipes possam ser integrados aos incrementos de trabalho. A integração e o teste de características tornam-se mais complexos e a sincronização das equipes usando backlogs e roteiros é ainda mais essencial (consulte o capítulo 6.3).

A localização distribuída das equipes em diferentes fusos horários apresenta desafios particulares de comunicação e requer um maior esforço de coordenação. Se os desenvolvedores de várias equipes distribuídas precisarem implementar certas funcionalidades em conjunto, por exemplo, os Product Owners devem gastar mais tempo na decomposição dos requisitos dessas funcionalidades a fim de minimizar a comunicação onerosa. As reuniões (virtuais ou físicas!) devem ser organizadas explicitamente com esforço de planejamento adicional e em momentos potencialmente inconvenientes. Diferentes línguas ou culturas faladas podem apresentar outros problemas.

Equipes distribuídas em locais diferentes, mas no mesmo fuso horário ou em fusos horários adjacentes, não têm todas essas dificuldades, mas ainda assim exigem algum esforço para organizar uma comunicação eficaz, seja através de reuniões virtuais ou físicas ou usando outras ferramentas de colaboração. A videoconferência e as ferramentas de colaboração podem ser de grande utilidade aqui.

Uma forma especial de equipes distribuídas são equipes terceirizadas. Tais equipes não são necessariamente distribuídas geograficamente, mas sim organicamente, ou seja, os membros da equipe são funcionários de outra organização que está em alguma relação contratual com outras equipes.

Idealmente, os Product Owners não devem ser subcontratados, pois os conflitos de interesse podem impedi-los de assumir total responsabilidade pelo produto. Os subcontratados geralmente têm suas próprias metas, que, às vezes, podem não estar totalmente correlacionadas com a visão ou as metas gerais do produto.

Cada equipe deve fornecer valor para os incrementos do produto. Algumas equipes não implementam recursos, mas se concentram em gerenciar a infraestrutura ou ajudar outras equipes a integrar os resultados nos incrementos de produtos. Por exemplo, o SAFe propõe ter uma equipe de sistema dedicada que fará a integração de todos os artefatos da equipe para um incremento de produto que pode ser liberado. O Framework Nexus propõe a existência de uma "Equipe de Integração Nexus", que não está realizando o trabalho, mas sim fornecendo consultoria aos desenvolvedores sobre como fazer isso por conta própria. Portanto, eles agregam valor implicitamente ao incremento do produto.

Mais detalhes sobre desenho e práticas organizacionais Ágeis podem ser encontrados em [Anderson2020].

Finalmente, devemos estar cientes da observação de Conway que descreveu um padrão muito comum conhecido como "Lei de Conway". Ele ressalta que a estrutura organizacional exerce uma influência sobre o design do sistema e a estrutura do produto. Em seu artigo [Conway1968], Conway afirma que as organizações que criam novos sistemas ou produtos tendem a estruturá-los da mesma forma que elas próprias estão organizadas e se comunicam atualmente. A estrutura de equipe resultante geralmente não é a ideal no que diz respeito ao desenvolvimento e à entrega eficientes em um contexto Ágil de grande escala.

6.2.5 Exemplo de empresa de telecomunicação

Neste exemplo, ilustramos a abordagem mencionada anteriormente para a divisão de requisitos baseada em recursos e discutimos a influência do contexto organizacional na estrutura das equipes em um ambiente ágil e sua capacidade de fornecer funcionalidades de produtos funcionais aos clientes.

Considere o exemplo de uma empresa de telecomunicações que procura desenvolver e lançar dois novos produtos de banda larga para seus clientes:

1. Um novo produto de alta velocidade VDSL (internet através da linha telefônica) "VDSL100"
2. Um produto de fibra para casa (internet por fibra óptica) "FTTH1000". Em uma primeira fase, os Product Owners analisaram os dois novos produtos e juntos estabeleceram a hierarquia de requisitos de acordo com os principais processos de negócio, como mostrado em Figura 39:

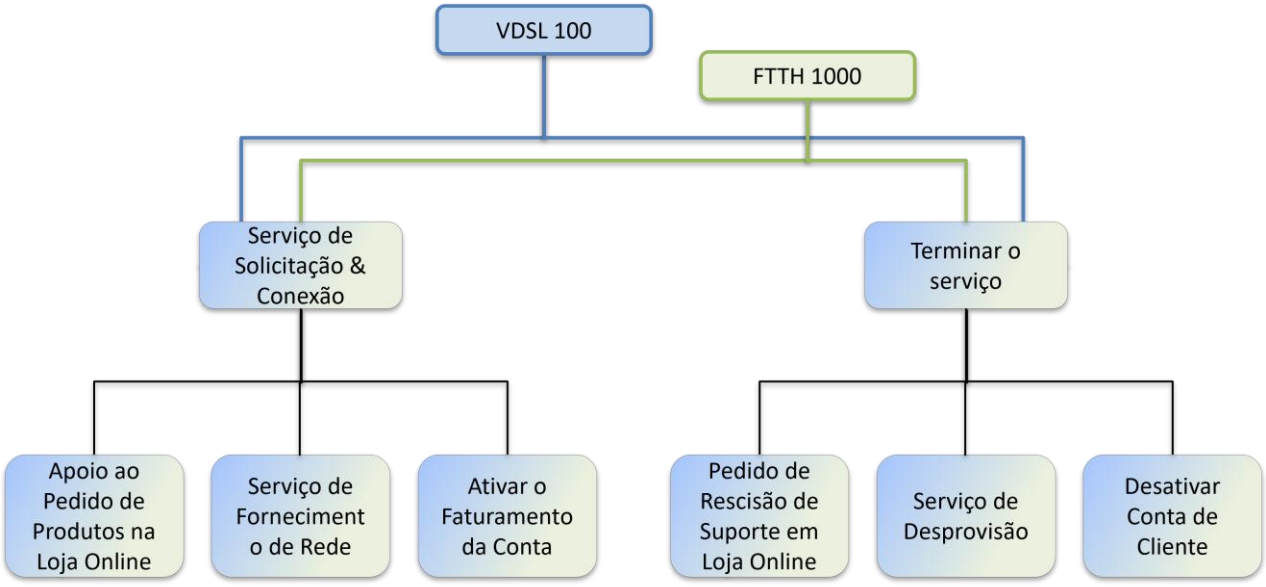


Figura 39: Estrutura dos requisitos dos produtos de banda larga

Mesmo que os detalhes para cada requisito possam variar entre os dois produtos, a organização dos requisitos de granularidade grosseira é a mesma.

Para fornecer os dois produtos a seus clientes, a empresa de telecomunicações deve ampliar seu sistema de TI. Por razões relacionadas à história da organização, os principais sistemas de TI, bem como os recursos e habilidades da equipe de TI, estão organizados da seguinte forma (1) Loja Online e Portal de Atendimento ao Cliente, (2) Conta de Sistema de Conta do Cliente e Faturamento e (3) Sistemas de Provisionamento e Instalação de Rede.

Ou seja, a loja on-line e o portal de atendimento ao cliente são considerados um único produto de TI, com uma camada de tecnologia completa de front-end, lógica de negócio e outra de persistência. Este também é o caso das contas do cliente e do sistema de faturamento. Os desenvolvedores normalmente são especialistas em um ou outro desses sistemas, mas não em ambos. A rede e os sistemas de provisionamento são mais diversificados, mas são tratados de forma semelhante por funções técnicas especializadas.

Como a organização pretende fazer a transição para uma abordagem Ágil em escala, os líderes da empresa de telecomunicações se reúnem com os Product Owners para discutir a melhor estrutura para as equipes em um ambiente Ágil.

A primeira estrutura de equipe proposta e os requisitos do produto atribuído são mostrados em Figura 40:

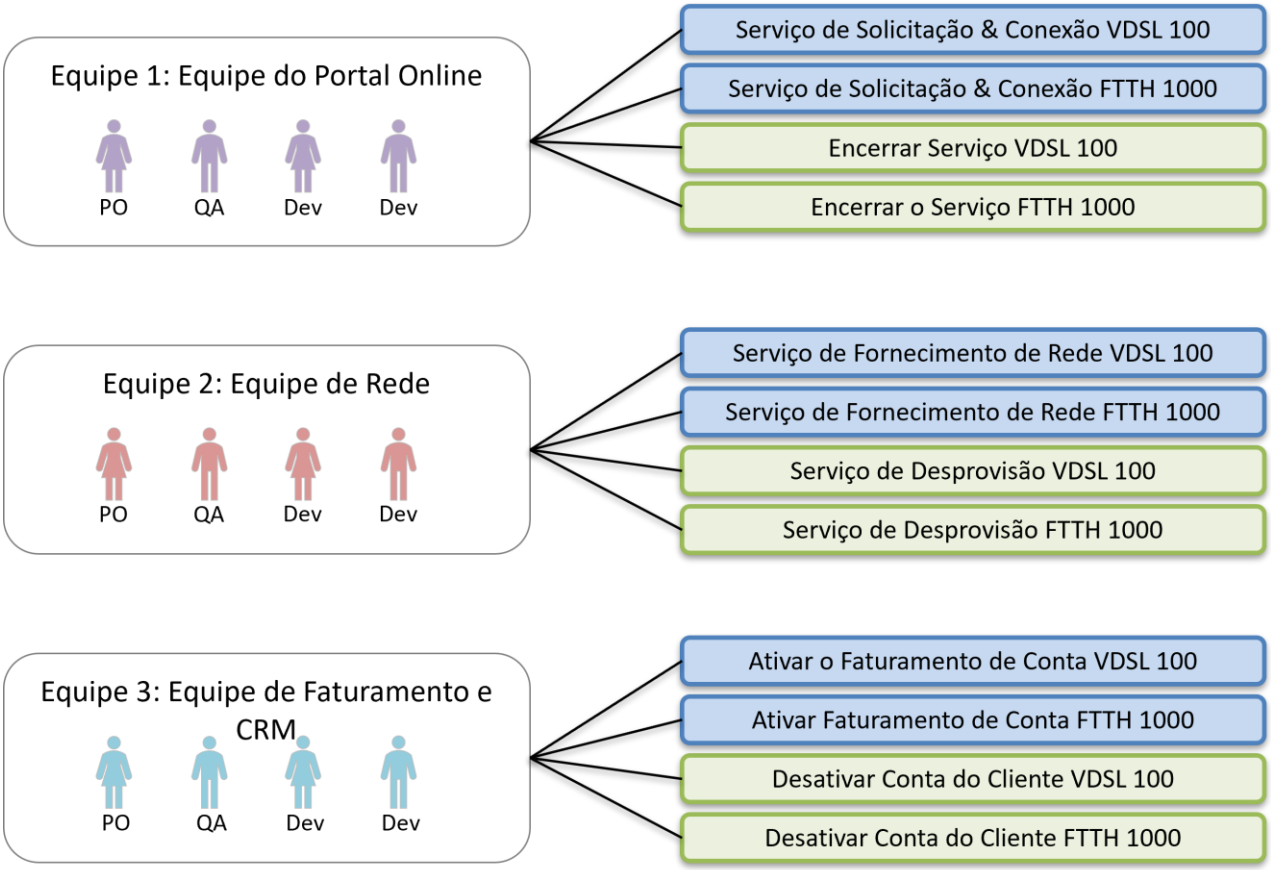


Figura 40: Estrutura da equipe compatível com a estrutura organizacional

A composição das equipes em um ambiente Ágil corresponde de perto à estrutura organizacional existente. Os membros da equipe são especialistas no sistema correspondente e trabalham em requisitos que atendem a esse sistema. A comunicação entre as equipes é necessária principalmente para garantir que os sistemas trabalhem juntos para lançar com sucesso os dois serviços. Nenhuma equipe é capaz de fornecer, de forma independente, recursos de trabalho que apoiem totalmente um cliente interessado em qualquer um dos produtos. Além do Product Owner de cada equipe, especializado nos requisitos desse sistema, outros Product Owners poderão ser necessários para coordenar a entrega dos requisitos do processo de ponta a ponta, de granularidade grosseira.

Para reduzir o esforço de comunicação entre as equipes, uma segunda configuração das equipes em um ambiente Ágil, mostrada em Figura 41, é então proposta:

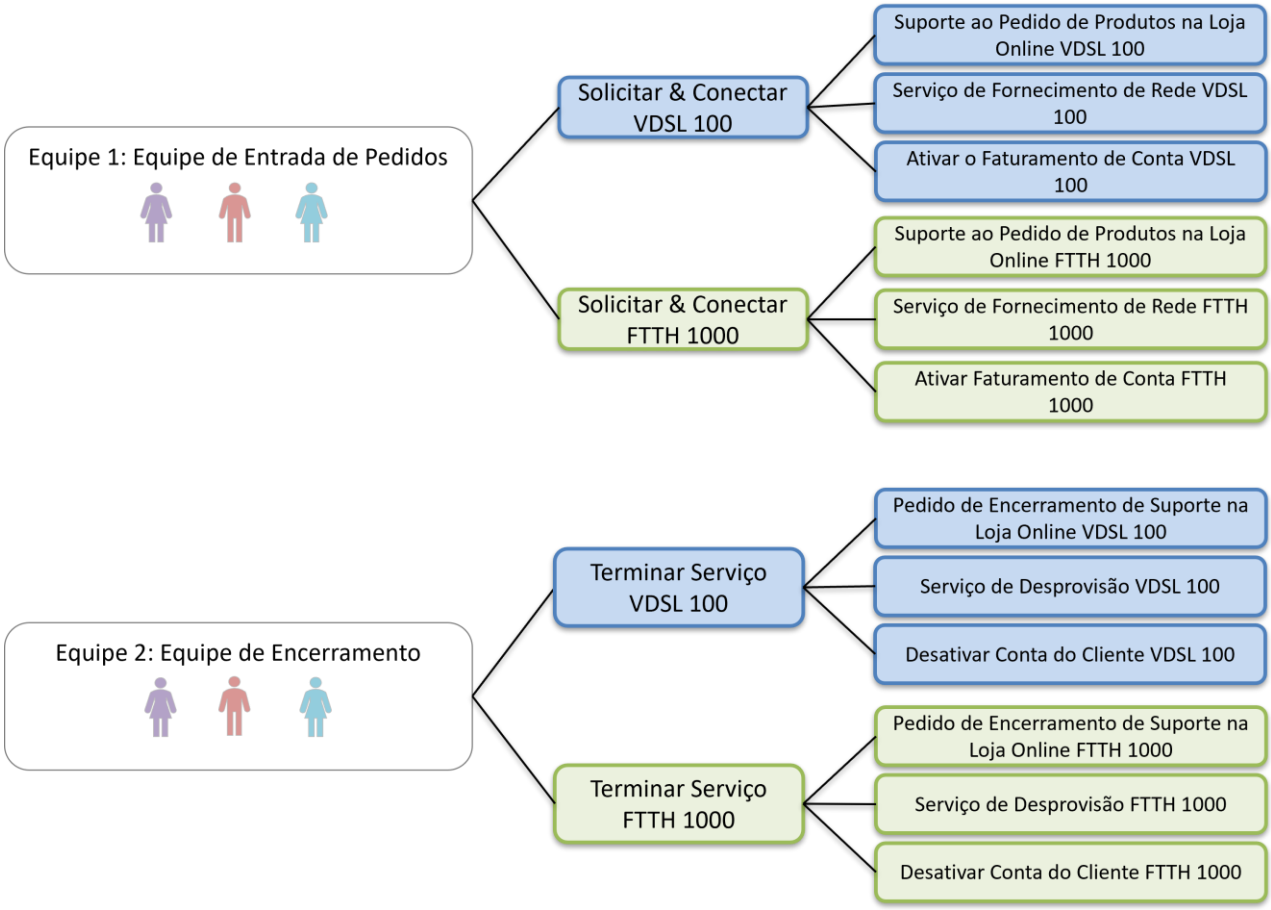


Figura 41: Estrutura da equipe de acordo com os serviços de conexão e encerramento

Cada equipe é responsável por um processo chave de negócio e os especialistas de cada um dos respectivos sistemas são misturados em cada equipe Ágil. Assim, uma equipe é capaz de entregar uma funcionalidade de processo ponta a ponta (p. ex., encomendar um produto de banda larga) e fornecer valor aos clientes (conforme as equipes de funcionalidades discutidas no capítulo 6.2.3). Do ponto de vista dos requisitos, o esforço de coordenação é reduzido, pois cada Product Owner pode projetar seu produto com maior autonomia. A coordenação é necessária principalmente no nível do produto (VDSL 100, FTTH 1000), por exemplo, para garantir um modelo de produto consistente em todos os diferentes processos. Como a solução integrada inclui três sistemas únicos de TI, a comunicação entre as equipes será necessária para coordenar mudanças e lançamentos dentro de um determinado sistema.

Outra composição de equipes em um ambiente Ágil é discutida, Figura 42, enfatizando também o conceito de equipes com capacidades completas de ponta a ponta:

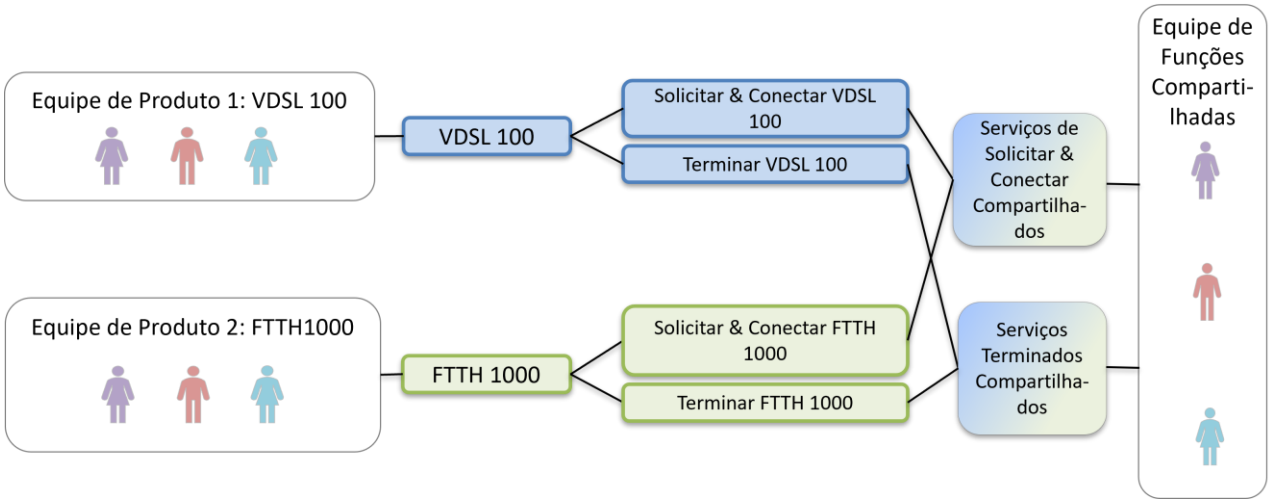


Figura 42: Estrutura de equipe com capacidades completas de ponta a ponta

Aqui cada equipe de produto Ágil é capaz de entregar um produto comercializável com todas as suas funcionalidades (VDSL 100, FTTH 1000). Com experiência em todos os sistemas e todos os processos de negócio, cada equipe é capaz de fornecer valor de negócio de forma independente. De uma perspectiva Ágil, esta estrutura de equipe deve ser preferida. Na prática, porém, essas equipes correm um alto risco de duplicar a funcionalidade ao trabalharem sobre os requisitos sobrepostas. Para resolver esse problema, sugere-se uma equipe de funções compartilhadas, especializada apenas nessas sobreposições, com a tarefa de encontrar soluções genéricas para os dois produtos: aproveitar os sistemas e serviços existentes sempre que possível ou desenvolver funcionalidades facilitadoras quando apropriado para dar suporte a esses e outros produtos (consulte a diferença entre funcionalidades de negócios e funcionalidades facilitadoras em 6.1.1).

Então, qual abordagem devemos escolher? Infelizmente, não há uma resposta simples. Como discutido acima, a abordagem preferida dependerá de muitos fatores: a estrutura organizacional existente, sua disposição para mudanças, restrições técnicas e estruturais, assim como o grau de funcionalidade compartilhada entre os diferentes produtos e processos. De preferência, primeiro estruturaríamos os requisitos e, em seguida, teríamos como objetivo criar equipes de funcionalidades na medida do possível, mas, na verdade, é preciso buscar um equilíbrio após a consideração cuidadosa de todos esses fatores.

6.3 Roteiros e Planejamento em Larga Escala

No desenvolvimento de produtos em larga escala, os Product Owners gerenciam os requisitos no backlog focado no produto, conforme discutido no capítulo anterior 6.2.1. Em contraste com o backlog, um roadmap é usado para planejar o desenvolvimento do produto incrementalmente. Um roteiro é uma previsão de como o produto crescerá [Pichler2016].

Os roteiros não alteram o conteúdo dos itens do backlog, mas os organizam em uma linha do tempo. Ele responde à pergunta sobre quando podemos esperar, grosso modo, quais funcionalidades.

Um roteiro é um meio útil para comunicar metas e decisões (estratégicas) aos desenvolvedores e outros stakeholders. Ele divide uma meta de longo prazo em iterações gerenciáveis, representa as dependências entre as equipes e fornece orientação e transparência aos stakeholders.

Um roteiro é o resultado de um exercício de planejamento, como mostrado em Figura 43. A base para o planejamento é, por um lado, o acúmulo de produtos encomendados e estimados e, por outro lado, os desenvolvedores disponíveis e sua capacidade.

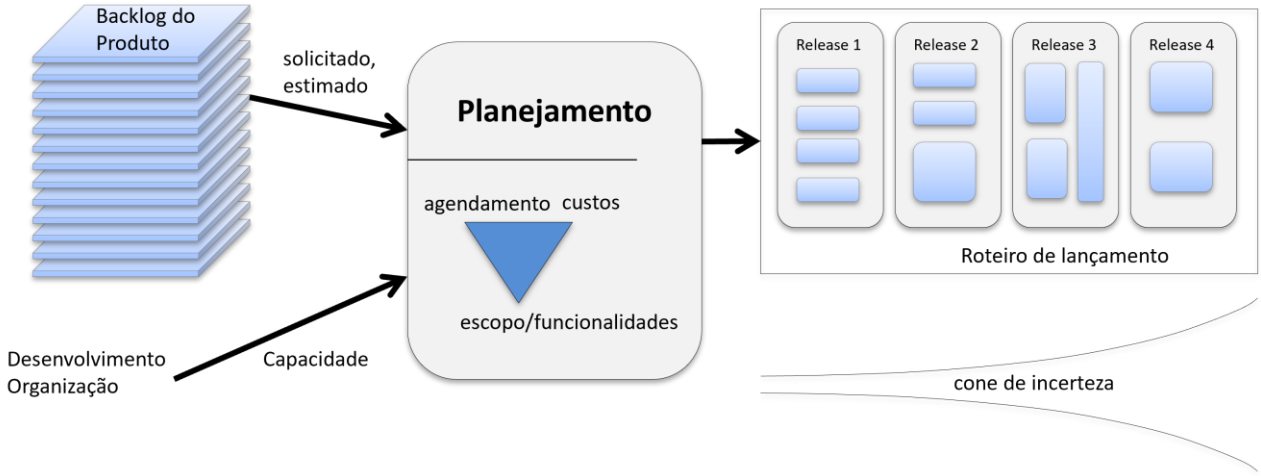


Figura 43: Exercício de planejamento

Com essas informações, o Product Owner enfrenta o triângulo típico do gerenciamento de projetos, tendo que equilibrar o escopo (funções ou funcionalidade do produto), os custos (recursos disponíveis) e o cronograma (datas de entrega). Deliberadamente, desenhamos o triângulo de ponta-cabeça para indicar que, em projetos ágeis, muitas vezes os custos e o cronograma são fixos e, portanto, os recursos planejados são a única variável.

No início do desenvolvimento Ágil de produtos, pouco se sabe sobre o produto ou sobre o trabalho realizado pelas equipes. Assim, o escopo do produto, assim como as estimativas de custo, estão sujeitos a um alto nível de incerteza. À medida que mais iterações são concluídas e mais feedback é coletado dos stakeholders, a incerteza diminui gradualmente levando a um planejamento mais confiável e a um roteiro estável. Esse princípio é conhecido como o cone da incerteza [Boehm 1981]. No entanto, o cone de incerteza também mostra que as versões a serem publicadas em breve oferecem maior certeza quanto às funcionalidades que serão incluídas, enquanto as versões mais adiante só podem ser definidas de forma vaga (ver Figura 43). Embora esse princípio seja geralmente verdadeiro para todos os projetos de desenvolvimento ágil, ele se torna ainda mais importante no desenvolvimento de produtos em larga escala, pois os riscos decorrentes da complexidade do produto e o potencial de desalinhamento entre várias equipes – e, consequentemente, a necessidade de mais planejamento – são ainda maiores.

6.3.1 Representação de roteiros

Um roteiro mostra metas estratégicas, marcos e requisitos de granularidade grosseira (p. ex., conjuntos de recursos). Marcos importantes podem ser internos ou determinados por eventos externos, como uma feira ou a introdução de nova regulamentação no mercado.

A representação de um roteiro depende de seu objetivo, grupo alvo e horizonte de planejamento. Para os clientes, os patrocinadores da gerência e a empresa, um roteiro de produto de longo prazo contendo metas estratégicas e requisitos de produto de granularidade grosseira geralmente é suficiente, com recursos normalmente descritos em linguagem de negócio [Pichler2016].

Em SAFe, o roteiro de produtos é chamado de "Roteiro de Soluções" e representa marcos de longo prazo, temas estratégicos e lançamentos. Um 'roteiro de soluções' normalmente fornece uma visão de um a três anos, com o nível de granularidade maior a curto prazo e depois reduzindo para o longo prazo.

SAFe divide uma 'Solução' em 'Incrementos de Programa' menores que oferecem valor aos clientes na forma de recursos de trabalho. Para representar o horizonte de planejamento mais curto, SAFe introduz o "Roteiro de Incremento do Programa", compreendendo até quatro iterações. Isto oferece uma visão mais detalhada do trabalho a ser feito nos próximos meses.

Outro tipo de roteiro, conhecido em SAFe como 'Program Board' [Leffingwell2017], concentra-se na entrega. Isto proporciona aos desenvolvedores e seus Product Owners uma visão dos tempos de backlog de granulação refinada (p. ex., histórias ou tarefas) e as dependências entre eles.

Um roteiro de produtos de nosso estudo de caso iLearnRE contendo objetivos estratégicos e características de granularidade grosseira é mostrado em Figura 44.

Você pode ver aqui os três próximos lançamentos: o primeiro já está comprometido; os outros dois são previsões. Cada lançamento é atribuído a um horizonte de planejamento pré-definido. As funcionalidades são descritas em termos de negócio e não como épicos e histórias.

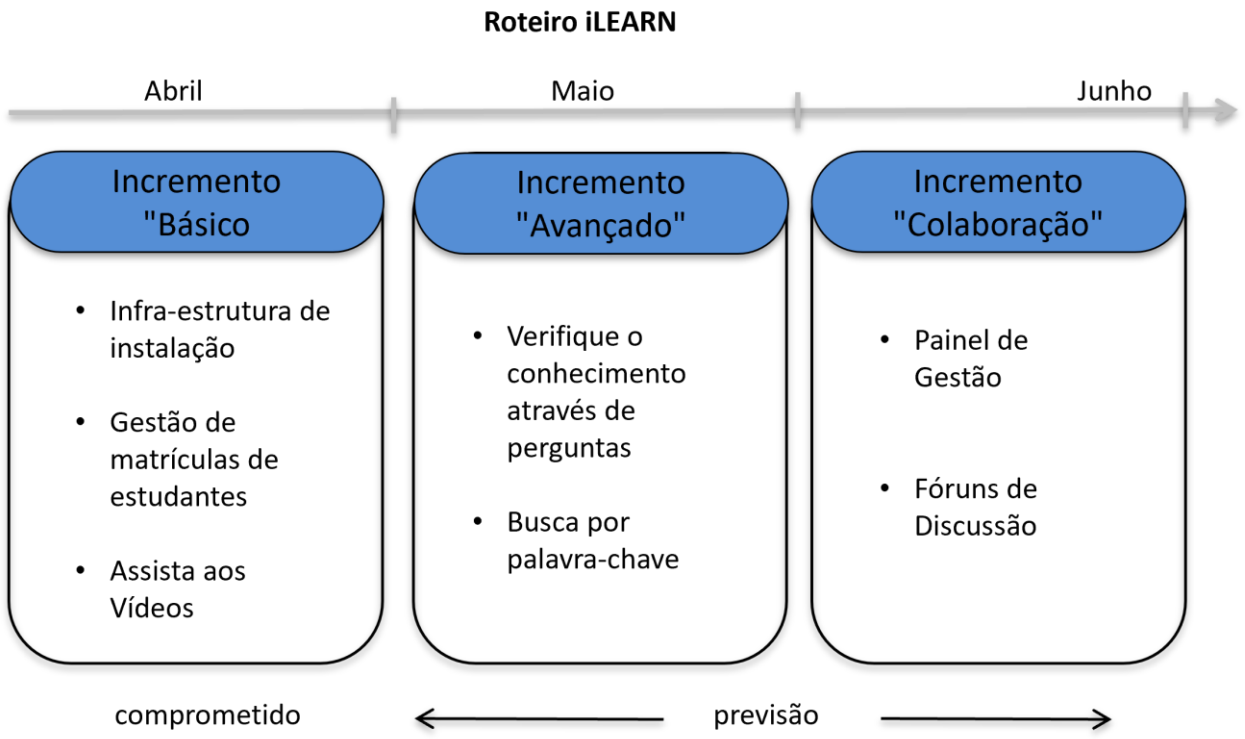


Figura 44: Um roteiro para o estudo de caso iLearnRE

No capítulo 3 introduzimos os mapas de histórias como uma forma de estruturar o backlog de seus produtos. Estes mapas podem ser ampliados para exibir o roteiro dos próximos lançamentos simplesmente usando o eixo vertical para alinhar épicos, funcionalidades e histórias a certos lançamentos, criando assim um backlog de lançamentos individuais. Isto é mostrado na Figura 45. Os itens no mapa da história podem ser mais grosseiros se o lançamento ainda estiver algum tempo à frente.

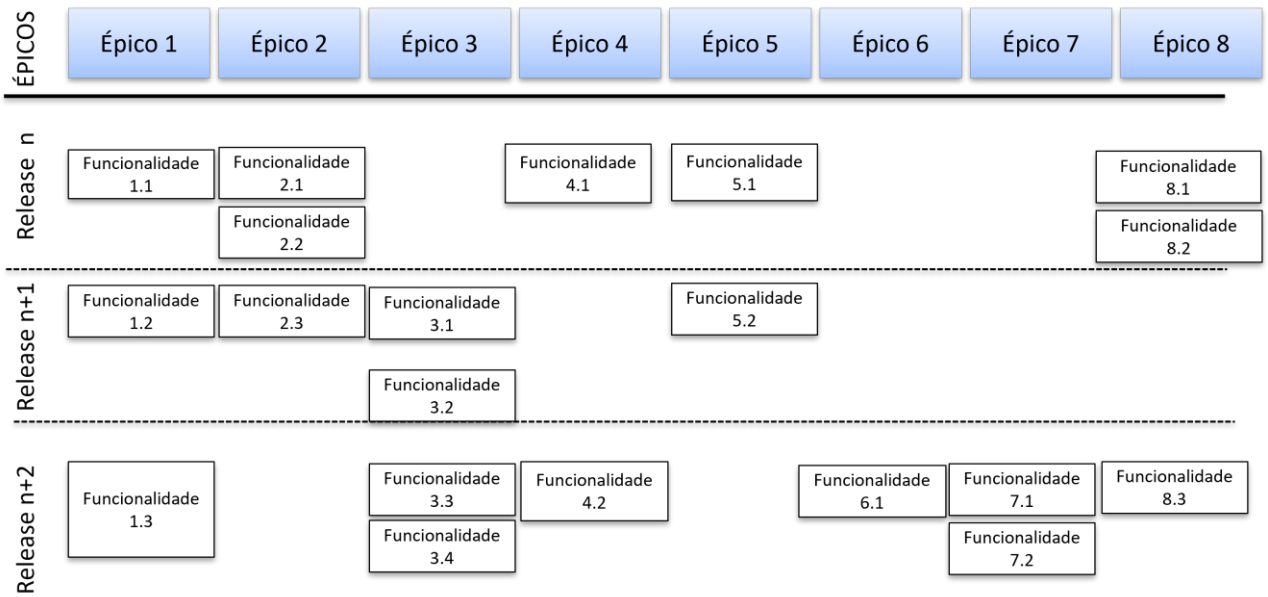


Figura 45: Mapas de histórias com sobreposição de release

Usar histórias e épicos para representar o roteiro do produto tem vários inconvenientes. Para a maioria dos stakeholders de negócio pode ser difícil entender como o produto está evoluindo como um todo, uma vez que muitos detalhes estão incluídos. Além disso, esses roteiros são propensos a mudanças e devem ser atualizados regularmente, o que consome muito tempo.

Figura 46 mostra um roteiro que não só inclui as iterações planejadas, mas também, no eixo vertical, um alinhamento dos itens do backlog para várias equipes, conforme discutido no capítulo 6.2. Os Program Boards são roteiros de entrega com granularidade refinada que são usados em SAFe durante o 'Planejamento de Incremento do Programa'. Eles contêm a linguagem dos desenvolvedores expressa por itens de backlog.

O quadro representa as funcionalidades a serem implementadas (F1...F4). As funcionalidades são divididas em itens de backlog, aqui codificados por cores. A prioridade deles é indicado pelo número. O quadro é usado para identificar as dependências críticas entre os itens de trabalho, conforme indicado pelas setas.

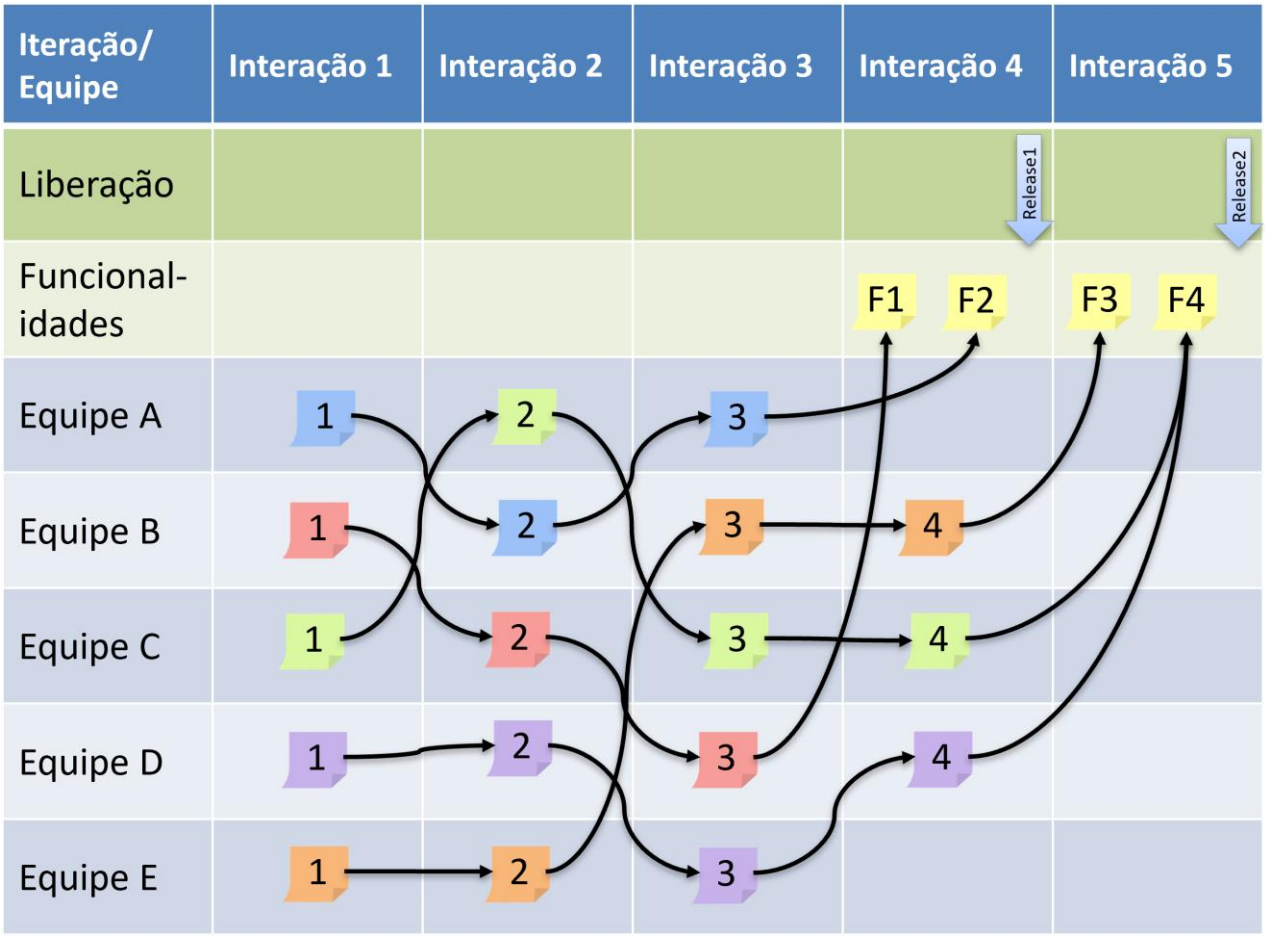


Figura 46: Um roteiro com dependências explícitas

Se suas equipes estiverem no mesmo local, você pode ser capaz de manter seu roteiro fisicamente na parede.

Se tiver que trabalhar com equipes distribuídas, você encontrará dezenas de ferramentas de roteiro para dar suporte ao planejamento visual de várias versões, muitas das quais são capazes, em maior ou menor grau, de se integrar às ferramentas usadas para gerenciar o próprio backlog.

Ao contrário do SAFe, outras estruturas como Less e Nexus não sugerem nenhum uso específico de roteiros. Isso não significa que os roteiros não possam ser usados dentro dessas estruturas, mas sim que cabe aos desenvolvedores decidir se um roteiro é necessário e que tipo de roteiro dará melhor suporte ao planejamento e ao trabalho de integração.

6.3.2 Sincronização de equipes com roteiros

O desenvolvimento Ágil está focado em iterações curtas com ciclos rápidos de feedback, portanto, a situação ideal é aquela em que o produto possa ser desenvolvido com a estreita colaboração de pequenos grupos em um ritmo curto.

É também fundamental que se estabeleça um *ritmo regular* para as iterações de desenvolvimento e releases [DeMarco et al.2008]. Ciclos irregulares irritam a equipe, dificultam o planejamento e tornam mais difícil acompanhar a velocidade dos desenvolvedores.

Este ritmo também é chamado de cadência. Na música, uma cadência é uma configuração melódica que cria uma sensação de resolução ou de finalidade. Para o desenvolvimento de software, esse senso de resolução é criado em vários níveis de abstração: dentro dos desenvolvedores por meio de reuniões diárias, para os desenvolvedores como um todo ao entregarem ao Product Owner no final de uma iteração de sprint e, potencialmente, para a organização de desenvolvimento em escala ao criar um incremento de produto que pode ser enviado para cada ciclo de lançamento.

Se você tiver apenas uma equipe, a entrega de um novo incremento de produto após cada iteração pode ser feita sem se alinhar com outras equipes. Portanto, nenhuma outra cadência além da cadência de iteração (em Scrum o comprimento da sprint) é necessária. Se você tiver várias equipes trabalhando no mesmo produto, você precisa integrar todos os produtos entregues em equipe a um novo incremento de produto. Como os testes de ponta a ponta e o trabalho necessário para empacotar todos os produtos em um lançamento podem envolver algum esforço adicional, uma cadência adicional para lançamentos do cliente pode ser introduzida.

Neste sentido, uma organização Ágil em larga escala pode ser comparada a uma grande orquestra que executa música complexa. Uma organização Ágil e funcional em grande escala mostra uma espécie de harmonia. Se a organização não está funcionando bem, então a harmonia não é visível, assim como uma orquestra que não está tocando no tempo. Se você tiver que trabalhar com várias equipes, então as durações de iteração para cada equipe não precisam ser idênticas, mas os ciclos devem ser compatíveis no sentido de que podem ser sincronizados numa cadência maior. Assim, por exemplo, equipes individuais podem escolher uma sprint de duas ou quatro semanas dentro de um ciclo de release de quatro (ou oito) semanas (ver Figura 47).

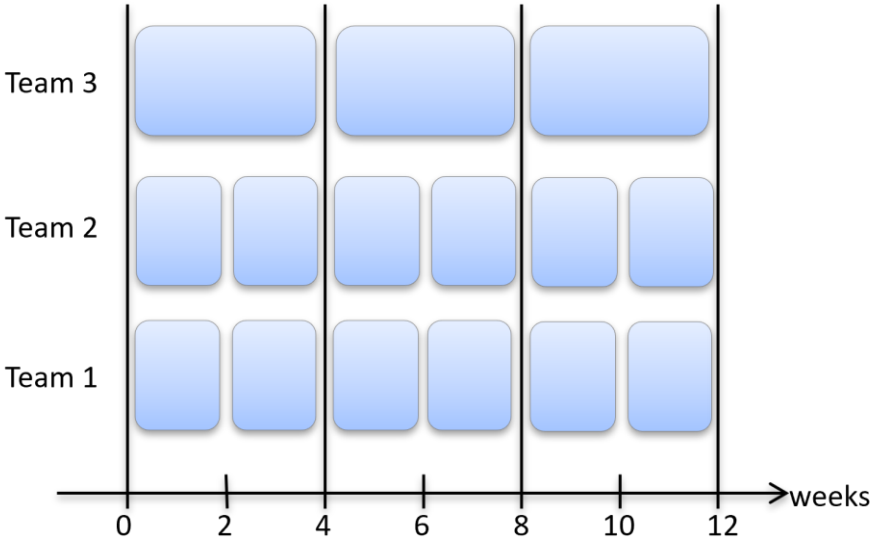


Figura 47: Comprimento de iteração diferente, mas compatível

A integração e os testes manuais provavelmente levarão a ciclos de release mais longos. A automação pode ajudar a encurtar os ciclos de release: abordagens de integração contínua e capacidades de implementação contínua podem permitir que as equipes implementem recursos em ciclos mais curtos.

6.3.3 Desenvolvimento de roteiros

No desenvolvimento de produtos em larga escala, o trabalho de requisitos é realizado por diferentes papéis de Product Owner com base em uma hierarquia de requisitos, conforme discutido no capítulo 6.2.1. As responsabilidades com respeito aos roteiros também serão diferentes em cada nível da hierarquia. Em um nível superior, por exemplo, os Product Owners podem ser responsáveis pelo roteiro do produto e, em um nível inferior, eles podem estar mais focados no roteiro de entrega.

Para desenvolver um *roteiro* de longo prazo para *o produto*, o Product Owner deve primeiro definir uma visão e uma estratégia do produto (consulte o capítulo 2). Isso é necessário para que os stakeholders certos sejam envolvidos no trabalho do roteiro do produto (gerenciamento de stakeholders).

Depois de estabelecer uma visão e uma estratégia do produto, os Product Owners devem obter requisitos de granularidade grosseira (consulte o capítulo 3) envolvendo-se com os stakeholders necessários. Neste momento, não há necessidade de investir tempo em requisitos detalhados. Mais tarde, durante o refinamento do backlog, mais detalhes serão descobertos.

Para obter apoio total para o desenvolvimento do produto, vários stakeholders devem ser envolvidos desde cedo e devem compreender os objetivos de negócio do produto. O roteiro do produto deve, portanto, ser adaptado aos seus interesses particulares e necessidades de informação e deve ser compartilhado e validado regularmente com eles. Os stakeholders comumente são, por exemplo, os executivos e a gerência sênior, os vendedores e o marketing, bem como os desenvolvedores.

Os Product Owners atribuem requisitos de granularidade grosseira ao longo de um amplo horizonte de planejamento, ao mesmo tempo em que mostram metas estratégicas na linha do tempo. Em um roteiro inicial do produto, os Product Owners devem evitar prazos difíceis.

Em vez disso, as funcionalidades devem ser planejadas mensalmente ou trimestralmente. À medida que o desenvolvimento do produto amadurece, datas e prazos concretos podem ser adicionados.

Para criar um *roteiro de entrega* de médio prazo, os Product Owners devem refinar os itens do backlog do roteiro de produtos existente. Esses itens precisam ser estimados de forma aproximada pelos desenvolvedores, mesmo que as estimativas ainda sejam imprecisas (p. ex., T-Shirt Sizing) nesse estágio. A estimativa deve ser boa o suficiente apenas para fornecer uma visão geral das próximas iterações.

A experiência prática mostra que na maioria das estimativas de grande escala, os erros para cada estimativa individual se anulam mutuamente, o que significa que a estimativa global é razoavelmente precisa, apesar dos erros individuais.

No capítulo 5.4 discutimos técnicas de estimativa de itens de backlog. Você também pode aplicar as mesmas técnicas para estimativa e planejamento a longo prazo. Este trabalho de estimativa está além do escopo da Engenharia de Requisitos tradicional, mas se torna importante nos contextos RE@Agile porque o trabalho de requisitos vai de mãos dadas com o planejamento. Muito mais sobre esse tema pode ser encontrado em [Cohn2006].

A criação e a atualização dos roteiros de entrega geralmente ocorrem em eventos de planejamento presenciais conhecidos como Big Room Plannings (ou PI Planning no SAFe), realizados em intervalos regulares. Nesses eventos, os desenvolvedores planejam, estimam e priorizam recursos em colaboração. Os Product Owners preparam os itens de backlog antecipadamente e os alinham à visão, bem como ao roteiro de produtos existente. As equipes trabalham umas com as outras para identificar os riscos e as dependências importantes. O roteiro de entrega é atualizado para mostrar os itens refinados de backlog, as dependências entre eles e como eles se alinham com a visão do produto.

6.3.4 Validação de roteiros

O *roteiro do produto* também deve ser revisto a partir da perspectiva do negócio: o feedback dos clientes, as mudanças no mercado, as próximas ideias e tendências do mercado, bem como produtos similares que entram no mercado, devem ser todos considerados. Para este fim, o MMP (como introduzido no capítulo 5.5) é um bom ponto de partida. Os intervalos de validação dependem da estabilidade do mercado: em um mercado altamente dinâmico, por exemplo, o roteiro do produto deve ser revisto pelo menos mensalmente, caso contrário, intervalos trimestrais podem ser suficientes. Os principais stakeholders devem ser mantidos envolvidos no desenvolvimento do roteiro para aumentar o aceite e para comunicar as mudanças.

A fim de reduzir o cone de incerteza, os *roteiros de entrega* também devem ser atualizados regularmente, com base no feedback dos stakeholders sobre os incrementos integrados do produto (ver MVP no capítulo 5.5), ou sobre os resultados das iterações.

Os intervalos de validação dependem da maturidade do desenvolvimento do produto e de mudanças no roteiro do produto. Em um processo de desenvolvimento maduro, por exemplo, em que os desenvolvedores seniores já trabalham juntos há algum tempo no mesmo produto, o roteiro de entrega pode precisar apenas ser revisto após um lançamento. No início do desenvolvimento do produto, o roteiro de entrega deve ser validado após a integração do primeiro incremento do produto. A validação dos roteiros de entrega pode ser incluída nos eventos regulares de planejamento descritos acima.

6.4 Validação do Produto

Uma das principais ideias do desenvolvimento Ágil é desenvolver uma pequena parte do produto, gerar feedback envolvendo os stakeholders e adaptar o desenvolvimento do produto de acordo com as descobertas e os insights obtidos. Assim, seguindo o princípio do ciclo Build-Measure-Learn [Ries2011], a validação do produto torna-se um passo importante para obter feedback rápido. Cada vez que um novo incremento de produto é lançado, os Product Owners utilizam esse incremento para verificar seu valor de negócio e para examinar se os requisitos do produto foram corretamente compreendidos.

A validação em nível de produto é um método importante no desenvolvimento de produtos em larga escala, pois garante que os Product Owners compartilhem a responsabilidade total, desde os requisitos comerciais até a integração do produto. É o produto inteiro que tem valor para os stakeholders, não apenas pequenas partes de produto.

Em Scrum, uma revisão de sprint é um meio adequado de discutir um incremento de produto (e os possíveis requisitos resultantes) com os stakeholders relevantes. No desenvolvimento de produtos em larga escala, uma ideia semelhante pode ser usada: mas, em vez de analisar uma única parte desenvolvida do produto por uma equipe, todos os produtos da equipe são integrados a um incremento de produto que vale a pena validar. O incremento do produto é demonstrado em uma *revisão do produto (demonstração),* mostrando as funcionalidades de ponta a ponta. Assim, os stakeholders ficam com uma melhor impressão de todo o produto [SAFe1], [Larman2016], [LeSS].

Para coordenar o trabalho de integração que é a base para a validação em nível de produto, um roteiro de entrega mostrando os marcos de lançamento pode ser usado para sincronizar as equipes (ver capítulo 6.2.3).

Os desafios do desenvolvimento de produtos em larga escala (como mencionado no capítulo 6.1) também devem ser considerados na validação do produto. Isto significa que você deve envolver um alto número de stakeholders e usuários de forma eficaz e comunicar seu feedback de volta aos desenvolvedores. Além disso, você deve alcançar uma compreensão geral do produto integrado, considerando diferentes perspectivas e conhecimentos dos stakeholders.

Ao envolver muitas pessoas em uma grande revisão do produto, é muito importante encontrar o nível certo de detalhes nas discussões para manter todos os participantes interessados. Uma abordagem é usar um padrão de colaboração divergente e convergente [Design Council]. Na parte divergente da revisão, a sala está dividida em múltiplas áreas onde as equipes demonstram diferentes funcionalidades do incremento do produto.

Como em um bazar, as pessoas passeiam, assistem a demonstrações de seu interesse e dão feedback à equipe correspondente. Em seguida, na parte convergente da revisão, as pessoas se reúnem para resumir suas descobertas e discutir aspectos importantes e compartilhar novas ideias.

As análises de produtos estão presentes em vários frameworks de dimensionamento. Na Nexus e Less a reunião de revisão é chamada de *Sprint Review*. Em SAFe, é conhecido como *System Demo*. De acordo com o guia do Nexus, a revisão deve ser feita em um período, usando, como regra geral, cerca de quatro horas para um período de um mês.

Outra abordagem para a validação de produtos no desenvolvimento em larga escala é aquela baseada na *análise de dados* [Maalej et al.2016]. O incremento integrado do produto é entregue aos usuários e, com base em seu comportamento, são feitas medições se as funcionalidades do produto têm um impacto positivo, neutro ou negativo. Os frameworks de análise de dados são normalmente usados para analisar sistematicamente os dados de feedback.

> Por exemplo, os Product Owners podem usar os resultados para identificar funcionalidades potencialmente mal projetadas. Para entender melhor os problemas identificados, eles podem precisar aplicar novamente técnicas regulares de elicitação e análise de requisitos.

Entretanto, o feedback dos stakeholders foi coletado, os Product Owners adaptam e priorizam novamente os itens de backlog e adicionam novos itens sempre que necessário. Alguns itens podem ser removidos do backlog se tiver sido demonstrado na validação do produto que as funcionalidades correspondentes não geram o valor pretendido. Mudanças no backlog de produto podem, por sua vez, desencadear mudanças no produto e no roteiro de entrega, conforme discutido no capítulo 6.3.4.

Lista de Abreviaturas

DSDM	Dynamic Systems Development Method
DoD	Definition of Done
DoR	Definition of Ready
LeSS	Large Scale Scrum (https://less.works)
MMP	Minimum Marketable Product
MVP	Minimum Viable Product
PO	Product Owner
RE	Requirements Engineering
ROI	Return on Investment
SAFe	Scaled Agile Framework (www.scaledagileframework.com)
WSJF	Weighted Shortest Job First

Referências

[AgileAlliance] Glossary of the Ágil Alliance: Definition of term "Definition of Ready": https://www.agilealliance.org/glossary/definition-of-ready, Última revisão em abril de 2024.

[AgileManifestoPrinciples] https://agilemanifesto.org/principles.html, Última revisão em abril de 2024.

[Alexander2005] Alexander, I. F.: A Taxonomy of Stakeholders – Human Roles in System Development. International Journal of Technology and Human Interaction, Vol 1, 1, 2005, páginas 23-59.

[AmLi2012] Ambler, S., Lines, M.: Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise, IBM Press, 2012

[Anderson2020] Anderson, J.: Agile Organizational Design – Growing Self-Organizing Structure at Scale. Leanpub, 2020.

[Beck2002] Beck, K.: Test Driven Development: By Example. Addison-Wesley 2002.

[BiKo2018] Bittner, K.; Kong, P.; West, D.: The Nexus Framework for Scaling Scrum, Addision Wesley, 2018

[Boehm 1981] Boehm Barry W.: Software Engineering Economics Published 1981 by Prentice Hall

[BOSSANOVA] https://www.agilebossanova.com/#bossanova, última visita em abril de 2024.

[ClBa1994] Clegg, D.; Barker, R. (09-11-2004). Case Method Fast-Track: A RAD Approach. Addison-Wesley.

[[Clements et al.2001]P. Clements et al.: Evaluating Software Architectures, SEI Series in Software Engineering, 2001

[Cohn2004] Cohn, M.: User Stories Applied For Agile Software Development, Addison-Wesley, 2004

[Cohn2006] Cohn, M.: Agile Estimation and Planning, Addison Wesley, 2006

[Conway1968] Conway, Melvin E.: How Do Committees Invent? Datamation Magazine, 1968. http://www.melconway.com/Home/Committees_Paper.html. Última visita em abril 2024.

[Cooper2004] Cooper, A.: The Inmates are Running the Asylum: Why High Tech Products Drive Us Crazy and How to Restore the Sanity

[DeMaLi2003] DeMarco, T.; Lister, T.: Waltzing with Bears – Managing Risks on Software Projects, Dorset House, 2003

[DeMarco et al.2008] DeMarco, T.; Hruschka, P. Lister, T.; McMenamin, S.; Robertson, J+S.: Adrenaline Junkies and Template Zombies – Understanding Patterns of Project Behavior, Capítulo 31: Ritmo, Dorset House, 2008

[Design Council] A study of the design process https://www.designcouncil.org.uk/fileadmin/uploads/dc/Documents/ElevenLessons_Design_Council%2520%25282%2529.pdf. Última visita em abril 2024.

[Doran1981] Doran, G. T: There's a S.M.A.R.T. way to write management's goals and objectives, Management Review. AMA FORUM.AMA FORUM. 70 (11): 35–36 1981.

[Glinz2014] Glinz, M.: A Glossary of Requirements Engineering Terminology. Standard Glossary for the Certified Professional for Requirements Engineering (CPRE) Studies and Exam, Version 2.0, 2020. https://www.ireb.org/en/downloads/#cpre-glossary. Última visita em abril 2024.

[HaHP 2000] Hatley, D., Hruschka, P., Pirbhai, I.: Process for System Architecture and Requirements Engineering, Dorset House, N.Y. 2000

[HeHe2010] Heath, C., Heath, D.: Switch: How to Change Things When Change Is Hard. Crown Business, 2010

[Highsmith2001] Highsmith, J.: Design the Box. Agile Project Management E-Mail Advisor 2001, http://www.joelonsoftware.com/articles/JimHighsmithonProductVisi.html. Última visita em abril 2024.

[Hruschka2017] https://b-agile.de/downloads/articles/story_splitting.pdf. Última visita em abril 2024.

[IREB2022] IREB e.V.: CPRE Foundation Level Syllabus, 2022. https://www.ireb.org/en/downloads/#cpre-foundation-level-syllabus-3-0. Última revisão em março de 2023.

[ISO25000] ISO/IEC 25000:2014: Systems and software engineering: System and Software Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE: https://www.iso.org/standard/64764.html. Última visita em abril 2024.

[ISO25010] ISO/IEC 25010:2011: Systems and software engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and software quality models: https://www.iso.org/standard/35733.html. Última visita em abril 2024.

[ISO25012] ISO/IEC 25012:2008: Engenharia de software – Requisitos e avaliação da qualidade de produtos de software (SQuaRE) – Modelo de qualidade de dados: https://www.iso.org/standard/35736.html. Última visita em abril 2024.

[Jacobson1992] Jacobson, I. Object-oriented Software Engineering – A Use-Case Driven Approach, ACM Press, 1992

[Jacobson2011] https://www.ivarjacobson.com/publications/white-papers/use-case-ebook. Última visita em abril 2024.

[HaCh1993] Hammer, M., Champy, J.: Re-Engineering the corporation. Harper, 1993

[Jeffries2001] Jeffries, R.: Essential XP: Card, Conversation, Confirmation, 2001, https://ronjeffries.com/xprog/articles/expcardconversationconfirmation/ Última revisão em abril de 2024.

[Kahneman2013] Kahneman D.: Thinking, Fast and Slow. Farrar, Straus and Giroux, 2013.

[KnLe2017] Knaster, R.; Leffingwell, D.: SAFe 4.0 Distilled, Addison Wesley, 2017

[Kniberg] Kniberg, H.: Scaling Agile @ Spotify with Henrik Kniberg https://www.youtube.com/watch?reload=9&v=jyZEikKWhAU&feature=youtu.be ,and https://www.youtube.com/watch?v=4GK1NDTWbkY&t=156s. Última visita em abril 2024.

[Larman2016] Larman, C: Large-Scale Scrum: More with LeSS, Addison Wesley, 2016

[Lawrence1] Lawrence, R: How to Split a User Story http://agileforall.com/resources/how-to-split-a-user-story. Última visita em abril 2024.

[Lawrence2] Lawrence, R: Why Most People Split Workflows Wrong http://agileforall.com/why-most-people-split-workflows-wrong/. Última visita em abril 2024.

[Leffingwell2007] Leffingwell, D.: Scaling Software Agility – Best Practices for Large Enterprises, Addison Wesley, 2007

[Leffingwell2010] Leffingwell, D.: Agile Software Requirements – Lean Requirements Practices for Teams, Programs, and the Enterprise, Addison Wesley, 2010

[Leffingwell2017] Leffingwell, D. et al.: SAFe Reference Guide, Scaled Agile, Inc. 2017

[LeSS] Large-Scale Scrum: https://less.works Última revisão em abril de 2024.

[Maalej et al.2016] Maalej, W., Nayebi, M., Johann T., Ruhe, G.: Toward Data-Driven Requirements Engineering. IEEE Software (Volume 33, Issue 1), 2016

[MaKo2016] Maher, R., Kong, P.: Cross-Team Refinement in Nexus (Refinamento entre equipes no Nexus), https://www.scrum.org/resources/cross-team-refinement-nexus. Última visita em abril 2024.

[McPa1984] McMenamin, S., Palmer, J: Essential Systems Analysis, Yourdon Press, 1984

[Meyer2014] Meyer, B.: Agile! The Good, the Hype and the Ugly, Springer, 2014.

[Nexus Guide] https://www.scrum.org/resources/nexus-guide. Última visita em abril 2024.

[OsPi2010] Osterwald, A., Pigneur, Y.: Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers. John Wiley and Sons, 2010

[Patton2014] Patton, J.: User Story Mapping –Discover the Whole Story, Build the Right Product, O'Reilly, 2014

[Pichler2016] Pichler, R.: Strategize – Product Strategy and Product Roadmap Practices for the Digital Age, Pichler Consulting 2016.

[Primer2017] CPRE RE@Agile Primer https://www.ireb.org/en/downloads/tag:re-agile-primer. Última visita em abril 2024.

[Reinertsen2008] Reinertsen, D.: Principles of Product Development Flow: Second Generation Lean Product Development. Celeritas Publishing 2008

[Ries2011] Ries, E.: The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, Crown Business, New York, NY 2011

[RoRo2013] Robertson S. Robertson J.: Mastering the Requirements Process – Getting Requirements Right, 3rd edition, Addison Wesley, 2013

[RoRo2017] Robertson S. Robertson J.: Volere Requirements Specification Template, https://www.volere.org/requirements-auditing-is-the-specification-fit-for-its-purpose/. Última visita em abril 2024.

[Robertson2003] Robertson, S.: Stakeholders, Goals, Scope: The Foundation for Requirements and Business Models, 2003, https://www.volere.org/wp-content/uploads/2018/12/StkGoalsScope.pdf. Última visita em abril 2024.

[SAFe1] https://www.scaledagileframework.com/roadmap/. Última visita em abril 2024.

[SAFe2] https://www.scaledagileframework.com/pi-planning/. Última visita em abril 2024.

[SAFeMDM] https://www.scaledagileframework.com/safe-requirements-model/. Última visita em abril 2024.

[S@S Guide] Sutherland, J. and Scrum, Inc: Scrum@Scale Guide: https://www.scrumatscale.com/scrum-at-scale-guide/, última revisão em abril de 2024.

[SOCIOCRACIA] https://sociocracy30.org. Última visita em abril 2024.

[SofS] https://scrumguide.de/scrum-of-scrums/. Última visita em abril 2024.

[Spotify2012] https://blog.crisp.se/wp-content/uploads/2012/11/SpotifyScaling.pdf

[Wake2003] Wake, B: INVEST in Good Stories, and SMART Tasks, 2003, https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/. Última visita em abril 2024.

[WyHT2017] Wynne, M., Hellesøy, A., Tooke, S.: The Cucumber Book – Behaviour-Driven Development for Testers and Developers, The Pragmatic Programmers, 2017

[Yakima 2016] Yakima, A.: The Rollout – A Novel about Leadership and Building a Lean-Agile Enterprise with SAFe