

Scrum

Gestão Ágil para Projetos de Sucesso

Edição atualizada



Casa do
Código

RAFAEL SABBAGH

© Casa do Código

Todos os direitos reservados e protegidos pela Lei nº9.610, de 10/02/1998.

Nenhuma parte deste livro poderá ser reproduzida, nem transmitida, sem autorização prévia por escrito da editora, sejam quais forem os meios: fotográficos, eletrônicos, mecânicos, gravação ou quaisquer outros.

Casa do Código

Livros para o programador

Rua Vergueiro, 3185 - 8º andar

04101-300 – Vila Mariana – São Paulo – SP – Brasil

Sobre o autor

Rafael Sabbagh é instrutor oficial (Certified Scrum Trainer) da Scrum Alliance, organização internacional responsável pelo Scrum, e membro de seu Board de Diretores no momento da publicação do livro. Ele atua como Agile Coach e já treinou com sucesso ScrumMasters, Product Owners e membros de Times de Desenvolvimento em mais de quinze países da Europa, Américas e Ásia.

Ele possui grande experiência com projetos de tecnologia. Rafael também apresentou trabalhos em várias edições do Agile Brazil e de Scrum Gatherings da Europa e dos Estados Unidos. Ele é Engenheiro de Computação e Mestre em Administração de Empresas pela PUC-Rio. Rafael é cofundador da Knowledge21.

Knowledge21 é uma empresa brasileira com foco em Transformação Ágil, treinamentos e *coaching* em gestão Ágil e Lean de projetos e em desenvolvimento de produtos. Utilizando abordagens inovadoras e com resultados comprovados, a empresa acompanha equipes, gestores e organizações como um todo para que se tornem mais efetivas e atinjam altíssimos níveis de produtividade.

A empresa é formada por Agile Coaches e Trainers certificados e com vasta experiência de mercado na adoção e prática de métodos Ágeis em diferentes organizações, desde startups a organizações globais. Conheça mais em: <http://knowledge21.com.br>.

Dedicatória

Para as pessoas mais importantes da minha vida:

- meu pai, Miguel Armony, que não está mais aqui, mas se orgulharia mais do que ninguém ao ver este trabalho pronto;
- meus filhos, Clara e Henrique, e minha mulher, Maria José, que são a minha razão de estar aqui;
- minha mãe, Réjane, e meus irmãos, Nathália e Flávio, que, juntos com meu pai, me fizeram quem eu sou.

Agradecimentos

Agradeço aos meus revisores técnicos, Manoel Pimentel, Marcos Garrido e Rodrigo de Toledo, que prontamente aceitaram me ajudar a garantir um livro de alta qualidade, e ainda contribuíram diretamente para o seu conteúdo.

Agradeço à minha mulher, Maria José Levy Ibarra, pelo apoio nessa longa jornada que é escrever um livro.

Agradeço a tantos outros que, com seu apoio e *feedback*, me encorajaram e ajudaram a construir esta nova edição.

Prefácio

Imagine. Você contando aos seus netos que participou da grande transformação do início do século. Transformação na forma em que as pessoas trabalham. Uma mudança de paradigma. Que começou com empresas de TI e, aos poucos, expandiu-se para todas as áreas da sociedade.

É como se o seu bisavô tivesse trabalhado com Taylor e te contasse como se sentiu com as primeiras mudanças daquela transformação do início do século passado.

Foi o Taylor aparecer e a empresa logo implementou as mudanças. O trabalho ficou mais específico, com alocação e especialização de tarefas para cada cargo. Criaram metas para a produtividade e um sistema de recompensas para o cumprimento delas. Ah, a empresa também instalou a tal esteirinha (assembly line, em inglês), meu bisneto. — Conversa imaginária com seu bisavô que trabalhou com Taylor em 1916.

Pois bem. É isso que está acontecendo. Uma profunda transformação na essência de como trabalhamos. Uma transformação dos canais digitais, permitindo novos tipos de inovação e criatividade em vários domínios.

Uma transformação que afeta tanto os pequenos empreendimentos quanto os grandes segmentos da sociedade, tais como governo, transporte, comunicação de massa, arte e a ciência. Esta transformação não só mudará sociedades e economias inteiras, mas mexerá com a própria essência da natureza humana, do nosso comportamento e convívio em sociedade.

E você ali, na crista da onda, no lugar certo, na hora certa. Um protagonista deste admirável novo mundo. Buscando um modelo de transformação

baseada em princípios e valores fortes que impulsiona uma cultura de melhoria contínua. Tentando saber qual a parte do futuro que pode ser introduzida no presente.

Você já escolheu tomar a pílula vermelha (do filme *Matrix*). Não tem mais volta. Agora é fazer o *upload* do conhecimento, praticar e comemorar as conquistas.

Neste excelente livro, Rafael Sabbagh, um dos precursores do Scrum no Brasil, compartilha a teoria e a prática desse *framework* simples, efetivo e poderoso; um conhecimento essencial para a sua jornada.

Boa leitura.

— **Paulo Caroli**

Prefácio da primeira edição

Vivemos em um período de extrema riqueza na quantidade e qualidade de adoções de Agilidade em diferentes organizações. Hoje, é possível ver os assuntos ligados à Agilidade sendo discutidos nos mais diferentes níveis dentro dela. E um dos grandes catalisadores desse movimento de adoção é o Scrum. O Scrum fez com que Agilistas de todo o mundo ganhassem ferramentas para empatizar com os problemas gerenciais e econômicos das organizações.

Se fizermos uma busca rápida pela internet, encontraremos vários artigos, apresentações e vídeos falando sobre Scrum. Na prática, entender a mecânica básica do seu fluxo de trabalho é algo fácil e sem grandes desafios pela sua extrema simplicidade. Mas, segundo o próprio Rafael Sabbagh:

A adoção mundial de Scrum não significa que todos os problemas estão resolvidos. Longe disso, Scrum é apenas uma ferramenta que pode trazer diversos benefícios em comparação a outras formas de se conduzir projetos, mas somente se bem utilizada.

Essa frase resume muito bem o grande desafio em questão. “Ser bem utilizado” é um estado paradoxal em se tratando do Scrum. Por ser um *framework* iterativo e incremental para se desenvolver produtos, ele também estimula que o próprio aprendizado acerca dos seus detalhes seja construído de forma iterativa e incremental. Isso significa que não é necessário ter uma ampla compreensão do Scrum para começar a usá-lo. Logo, por definição, “usar bem” o Scrum é um estado que pode demorar um considerável tempo para ser alcançado.

A dinâmica de *framework*, as peculiaridades dos papéis e suas respectivas interações são uma parte difícil do Scrum. Existe uma grande variedade de possibilidades no uso de seus papéis, cerimônias, artefatos e regras. Na prática, compreender esses detalhes do *framework* é fundamental para permitir uma extensibilidade saudável, de forma que possa ser adotado em qualquer tipo de ambiente organizacional.

Tipicamente, o maior desafio se dá exatamente em função de sua extensibilidade. Na verdade, muito mais difícil do que usar bem o Scrum é estender seus papéis, cerimônias, artefatos e regras de forma a gerar uma congruência do contexto organizacional com os pilares do Scrum e com os valores do Manifesto Ágil.

É comum, por exemplo, que, movidos por um desejo de gerar menos conflitos no processo de adoção, sejamos compelidos a retirar algum dos elementos básicos do *framework*. Outro comportamento comum é usarmos de maneira desenfreada a extensibilidade para fazer o Scrum conviver de forma harmônica com todos os outros papéis, cerimônias, artefatos e regras já existentes na organização. Em ambos os casos, os resultados gerados são muito efêmeros e não promovem um uso sustentável do Scrum.

Adotar Scrum traz uma série de ganhos relacionados, mas também traz uma gama de perdas inerentes. “Perdas”, nesse caso, possui o sentido de “coisas de que precisamos abrir mão” para usá-lo. Como um *framework* Ágil, ele promove invariavelmente uma reflexão organizacional acerca de “o que” e “por que” melhorar. Logo, é comum que, com seu uso, seja necessário abrir mão de algum papel, de alguma cerimônia, de algum artefato ou de alguma regra existente na organização.

Na realidade, ao contrário do que muitas organizações buscam, o Scrum é um meio, e não o fim. Isso significa que ele sempre estará em uma contínua mutação, pois deve ser usado para ajudar a organização a chegar a uma forma melhor no que tange ao *mindset* e aos processos de gestão de projetos e de produtos.

Logo, o propósito do Scrum não é se perpetuar em uma organização. Muito pelo contrário, almeja-se que um dia a organização não mais precise dele.

Uma forma típica de compreender plenamente essa dinâmica é ter muitas

horas de experimentação real do Scrum em algum ambiente organizacional complexo. Na verdade, somente com essa carga de experiência na adoção será possível que muitos de seus detalhes sejam revelados e compreendidos.

Dadas todas essas peculiaridades da adoção de Scrum, é possível afirmar que Rafael Sabbagh conseguiu um feito singular: com este livro, Rafael conseguiu sintetizar de maneira didática todos os principais detalhes do uso de Scrum. Acredito piamente que este livro poderá encurtar o caminho para uma compreensão plena da sua essência e de como adotá-lo de maneira efetiva dentro uma organização.

Tive a honra de revisar este livro. Na verdade, há alguns anos, tive a honra maior de trabalhar junto do Rafael em um complexo processo de adoção de Scrum para uma grande empresa de telecomunicações aqui do Brasil. Lá tive a felicidade de conhecer e ser inspirado pela forma analítica e questionadora do pensamento do Rafael.

Na época, um dos objetivos desse trabalho foi produzir materiais sobre como usar Scrum para aquela organização. Apesar da aspiração que Rafael já tinha para escrever um livro, gosto de pensar que foi ali que ele começou a se concretizar.

Uma marca inconfundível do Rafael é destilar o conhecimento como se fosse uma cebola. Nessa abordagem, cada camada representa um aprofundamento do conhecimento que fora iniciado na camada anterior. Este livro tem essa abordagem da cebola. Dessa forma, tenho plena certeza de que, muito mais do que um simples livro, você, leitor, está munido com uma poderosa ferramenta de trabalho. Portanto, aproveite cada linha deste livro para que ele lhe ajude a transformar o seu dia a dia de trabalho em direção a um estado Ágil de gerenciar e desenvolver produtos.

— Manoel Pimentel Medeiros

Como este livro está organizado

Este trabalho está dividido em seis partes: *Introdução ao Scrum*, *Papéis: o Time de Scrum*, *Artefatos do Scrum*, *Eventos do Scrum*, *Técnicas Complementares* e *Final*.

Na **Parte I — Introdução ao Scrum**, começamos com as principais razões para se escolher e utilizar Scrum. Em seguida, ofereço uma definição formal e sigo explicando o que Scrum é em detalhes, indicando também onde melhor o *framework* pode ser aplicado. No capítulo *Como é o Scrum?*, o leitor pode fazer um passeio por um projeto utilizando Scrum e diversas práticas e artefatos associados, muitos deles opcionais. No capítulo seguinte, explico como se pode contratar projetos com Scrum. A Parte I se encerra com uma explicação das origens do Scrum e quais são as suas principais influências.

A **Parte II — Papéis**: o Time de Scrum trata em detalhes do que é, o que faz e como é cada um dos papéis do Time de Scrum, ou seja, o Time de Desenvolvimento, o Product Owner e o ScrumMaster. Mostro também nessa parte diversas ideias e práticas associadas aos papéis, como resolução de impedimentos, motivação e facilitação, entre outras.

Na **Parte III — Artefatos do Scrum**, explico o que é e como utilizar o Product Backlog, o Sprint Backlog, a Definição de Pronto e o Incremento do Produto. Acrescento aos artefatos básicos do Scrum a Definição de Preparado.

A **Parte IV — Eventos do Scrum** descreve as reuniões de Sprint Planning, de Daily Scrum, de Sprint Review e de Sprint Retrospective, indicando em detalhes boas práticas para sua execução e problemas comuns enfrentados por seus participantes. Adiciono também a reunião de Release Planning, a Release e o Refinamento do Product Backlog. O Sprint também é abordado

na Parte IV, além de fatores determinantes para a escolha de sua duração, motivos para seu cancelamento e a sua relação com a Definição de Pronto.

Na **Parte V — Técnicas complementares**, ofereço alguns conceitos adicionais que podem ser muito úteis na execução do projeto, como: metas de negócios (Meta do Sprint, Meta de Release ou de Roadmap e Visão do Produto), User Stories, Estimativas Ágeis com Story Points (que também incluem Planning Poker e Velocidade do Time de Desenvolvimento) e os Gráficos de Acompanhamento do Trabalho (Release Burndown, Release Burnup e Sprint Burndown).

Na **Parte VI — Final**, o leitor tem as palavras finais escritas pelo Marcos Garrido, Rodrigo de Toledo e Carlos Felippe Cardoso, um glossário dos termos relacionados ao Scrum e a bibliografia utilizada neste livro.

Para mais discussões, material e complementos ao conteúdo deste livro, acesse o *site* do livro em <http://livrodescrum.com.br>.

Você também pode discutir sobre este livro no Fórum da Casa do Código: <http://forum.casadocodigo.com.br>.

Caso você deseje submeter alguma errata ou sugestão, acesse: <http://erratas.casadocodigo.com.br>.

Sumário

Introdução ao Scrum	1
1 Por que Scrum?	3
1.1 Introdução	3
1.2 Entregas frequentes de retorno ao investimento dos clientes	5
1.3 Redução dos riscos do projeto	5
1.4 Maior qualidade no produto gerado	7
1.5 Mudanças utilizadas como vantagem competitiva	8
1.6 Visibilidade do progresso do projeto	9
1.7 Redução do desperdício	9
1.7.1 Produzir apenas o que os usuários vão utilizar	10
1.7.2 Planejar apenas com o nível de detalhe possível	11
1.7.3 Utilizar apenas os artefatos necessários e suficientes	14
1.8 Aumento da motivação e produtividade	15
1.8.1 Trabalho em equipe e autonomia	15
1.8.2 Facilitação e remoção de impedimentos	16
1.8.3 Melhoria contínua	16
1.8.4 Ritmo sustentável de trabalho	17
1.8.5 Realização do trabalho de ponta a ponta	18
2 O que é Scrum?	19
2.1 Introdução	19
2.2 Scrum é Ágil	20
2.2.1 Os valores Ágeis	23

2.2.2	Os princípios Ágeis	27
2.2.3	Os valores do Scrum	31
2.3	Scrum é um framework simples e leve	33
2.4	Scrum se aplica a produtos complexos em ambientes complexos	35
2.4.1	Sistemas Adaptativos Complexos	35
2.4.2	Onde utilizar Scrum?	36
2.5	Scrum é embasado no empirismo	40
2.6	Scrum é iterativo e incremental	41
3	Como é o Scrum?	47
3.1	Introdução	47
3.2	Início do projeto	47
3.3	Planejamento do Sprint	51
3.4	Desenvolvimento	53
3.5	Encerramento do Sprint	55
3.6	Entregas	56
3.7	Final do projeto	58
4	Como contratar projetos com Scrum?	59
4.1	Introdução	59
4.2	Contratos tradicionais	59
4.2.1	Preço Fixo tradicional	61
4.2.2	Tempo e Material tradicional	62
4.3	Princípios básicos para contratos com Scrum	63
4.3.1	Premissas básicas de contratos com Scrum	63
4.3.2	O Triângulo Ágil	64
4.3.3	Escopo não detalhado	65
4.3.4	Ponto de parada	65
4.4	Formatos de contratos com Scrum	67
4.4.1	Preço fixo, prazo fixo, escopo flexível	67
4.4.2	Preço fixo, escopo flexível, prazo fixo com opção de parada	68

4.4.3	Preço fixo, escopo flexível, prazo fixo com opção de parada taxada	69
4.4.4	Preço fixo, escopo flexível, prazo fixo com opção de troca de contexto	70
4.4.5	Incremental com pontos de verificação	70
5	De onde veio o Scrum?	71
5.1	Introdução	71
5.2	Um novo jogo	72
5.3	Lean e Sistema Toyota	74
5.3.1	Introdução	74
5.3.2	Os princípios do Lean	75
5.3.3	Lean e Agilidade	76
	Papéis: o Time de Scrum	85
6	Time de Desenvolvimento	87
6.1	Quem é o Time de Desenvolvimento?	87
6.2	O que faz o Time de Desenvolvimento?	89
6.2.1	Planeja seu trabalho	89
6.2.2	Realiza o desenvolvimento do produto	90
6.2.3	Colabora com o Product Owner durante o Sprint	90
6.2.4	Identifica e informa os impedimentos ao ScrumMaster	91
6.2.5	Obtém feedback sobre o produto	95
6.2.6	Entrega valor com frequência	95
6.3	Como é o Time de Desenvolvimento?	96
6.3.1	Multidisciplinar	96
6.3.2	Auto-organizado	100
6.3.3	Suficientemente pequeno	101
6.3.4	Motivado	102
6.3.5	Orientado à excelência	106
6.3.6	Focado nos objetivos	107

7 Product Owner	109
7.1 Quem é o Product Owner?	109
7.2 O que faz o Product Owner?	113
7.2.1 Gerencia o produto	113
7.2.2 Gerencia as partes interessadas no projeto	114
7.2.3 Mantém os objetivos do produto	115
7.2.4 Gerencia as entregas do produto	116
7.2.5 Colabora com o Time de Desenvolvimento durante o Sprint	118
7.3 Como é o Product Owner?	118
7.3.1 Único	118
7.3.2 Disponível para o trabalho no projeto	120
7.3.3 Competente para a definição do produto	121
8 ScrumMaster	123
8.1 Quem é o ScrumMaster?	123
8.2 O que faz o ScrumMaster?	124
8.2.1 Facilita o trabalho do Time de Scrum	124
8.2.2 Garante a remoção de impedimentos	131
8.2.3 Promove as mudanças organizacionais necessárias .	133
8.2.4 Garante o uso do Scrum	134
8.3 Como é o ScrumMaster?	135
8.3.1 Competente em soft skills	135
8.3.2 Presente	137
8.3.3 Suficientemente neutro	138
9 Onde está o Gerente de Projetos?	141
Artefatos do Scrum	145
10 Product Backlog	147
10.1 O que é o Product Backlog?	147

10.2 Como é o Product Backlog?	150
10.2.1 Ordenado	150
10.2.2 Planejável	152
10.2.3 Emergente	153
10.2.4 Gradualmente detalhado	153
 11 Sprint Backlog	 157
11.1 O que é o Sprint Backlog?	157
11.1.1 O quê?	158
11.1.2 Como?	158
11.2 Como é o Sprint Backlog?	159
 12 Definição de Pronto	 165
12.1 O que é a Definição de Pronto?	165
12.2 Como é a Definição de Pronto?	167
 13 Incremento do Produto	 171
 14 Definição de Preparado	 173
14.1 O que é a Definição de Preparado?	173
14.2 Como é a Definição de Preparado?	175
 Eventos do Scrum	 177
 15 Inicialização	 179
 16 Sprint	 183
16.1 O que é o Sprint?	184
16.2 Como é o Sprint?	186
16.2.1 Duração	186
16.2.2 Incrementos entregáveis	190
16.3 O Sprint pode ser cancelado?	190

17 Sprint Planning	193
17.1 O que é a Sprint Planning?	193
17.1.1 Duração	194
17.1.2 Saídas	194
17.1.3 Preparação	197
17.2 Como é a Sprint Planning?	197
17.2.1 Sprint Planning 1 e Sprint Planning 2	198
17.2.2 Planejamento Intercalado de Sprint	202
17.2.3 Planejamento Just-In-Time de Sprint	205
18 Daily Scrum	207
18.1 O que é a Daily Scrum?	207
18.2 Como é a Daily Scrum?	208
18.3 O que NÃO deve acontecer na Daily Scrum?	210
18.3.1 Disfunção: informe de impedimentos ao ScrumMaster	211
18.3.2 Disfunção: reunião de trabalho	212
18.3.3 Disfunção: prestação de contas a outros	212
18.3.4 Disfunção: falta de interesse e atenção	213
19 Sprint Review	215
19.1 O que é a Sprint Review?	216
19.2 Como é a Sprint Review?	218
19.2.1 Preparação	218
19.2.2 Gestão de expectativas	218
19.2.3 Quem participa	218
19.2.4 Apresentação	219
19.2.5 O que é apresentado	220
19.2.6 Resultados	220
20 Sprint Retrospective	223
20.1 O que é a Sprint Retrospective?	224
20.1.1 Lições aprendidas em times tradicionais	224
20.1.2 Melhoria incremental contínua em times Ágeis	224

20.1.3	A retrospectiva do Sprint	225
20.2	Como é a Sprint Retrospective?	225
20.2.1	Regularidade, frequência e duração	225
20.2.2	Participação do ScrumMaster	226
20.2.3	Participação do Product Owner	227
20.2.4	Dinâmica básica de uma retrospectiva	228
20.2.5	Diferentes possibilidades na retrospectiva	231
20.3	O que NÃO deve acontecer na Sprint Retrospective?	233
20.3.1	Busca de culpados	234
20.3.2	Insegurança e medo de exposição	234
20.3.3	Conflitos da diversidade	235
20.3.4	Pensamento grupal	236
21	Release	239
21.1	O que é a Release?	239
21.2	Como é a Release?	241
21.2.1	Quanto à frequência	241
21.2.2	Quanto a quem a recebe	243
21.2.3	Outras dimensões	244
21.2.4	Conclusão	244
22	Release Planning	247
22.1	O que é a Release Planning?	248
22.2	Plano da Release	248
22.3	Como é a Release Planning?	248
22.3.1	Agendamento e duração	248
22.3.2	A reunião	249
22.3.3	Granularidade dos itens	251
22.3.4	Escopo da Release	252
23	Refinamento do Product Backlog	253
23.1	O que é o Refinamento do Product Backlog?	254
23.2	Como é o Refinamento do Product Backlog?	256
23.2.1	Participantes	256
23.2.2	Quando ocorre	257

Técnicas complementares	259
24 Metas: definindo os objetivos	261
24.1 O que são metas no Scrum?	261
24.2 Meta de Sprint	264
24.2.1 O que é a Meta de Sprint?	264
24.2.2 Como é a Meta de Sprint?	266
24.3 Meta de Release/Roadmap	268
24.3.1 O que é a Meta de Release/Roadmap?	268
24.3.2 Como é a Meta de Release/Roadmap?	269
24.4 Roadmap do Produto	270
24.4.1 O que é o Roadmap do Produto?	270
24.4.2 Como é o Roadmap do Produto?	270
24.5 Visão de Produto	271
24.5.1 O que é a Visão do Produto?	271
24.5.2 Como é a Visão do Produto?	271
25 User Stories: representando o trabalho	279
25.1 O que é a User Story?	279
25.2 Como é a User Story?	281
25.2.1 Cartão	281
25.2.2 Conversas	286
25.2.3 Confirmação	286
25.3 Criando boas User Stories	290
25.4 Épicos e Temas	291
26 Story Points: estimando o trabalho	295
26.1 Para que serve estimar?	295
26.2 Precisão e acurácia de estimativas baseadas em juízo	297
26.3 Precisão e acurácia suspeitas: Lei de Parkinson e Síndrome do Estudante	299
26.4 Estimativas Ágeis	300
26.5 Unidades para as estimativas	301

26.5.1	Unidades mais comuns	301
26.5.2	Estimativas absolutas x relativas	302
26.5.3	Story Points	304
26.6	Como fazer a estimativa?	308
26.6.1	Planning Poker	310
26.7	Velocidade do Time de Desenvolvimento	313
26.8	No estimates: estimativas são desperdício	315
27	Burndown e Burnup: acompanhando o trabalho	319
27.1	Gráfico de Release Burndown	320
27.1.1	O que é o Gráfico de Release Burndown?	320
27.1.2	Como é o Gráfico de Release Burndown?	322
27.2	Gráfico de Release Burnup	324
27.2.1	O que é o Gráfico de Release Burnup?	324
27.2.2	Como é o Gráfico de Release Burnup?	326
27.3	Gráfico de Sprint Burndown	328
27.3.1	O que é o Gráfico de Sprint Burndown?	328
27.3.2	Como é o Gráfico de Sprint Burndown?	329
27.3.3	Linha ideal	332
Final		337
28	Além do Scrum	339
28.1	Os Quatro Domínios da Agilidade	339
28.2	Facilitação	341
28.3	Management 3.0	342
28.4	Lean Kanban	344
28.5	Técnicas para Product Owners	347
28.6	Lean Startup	348
28.7	Escalando Scrum	349
28.8	Testes automatizados e entrega contínua	350
28.9	Conclusão	353

29 Apêndice - Glossário	357
--------------------------------	------------

30 Apêndice - Bibliografia	363
-----------------------------------	------------

Versão: 19.7.6

Parte I

Introdução ao Scrum

CAPÍTULO 1

Por que Scrum?

1.1 INTRODUÇÃO

Scrum existe desde o início dos anos 1990, mas foi só na década seguinte que se tornou popular. Scrum ganhou o mundo, desbancou métodos tradicionais e se tornou a forma mais bem-sucedida de se trabalhar em projetos de desenvolvimento de *software*. Uma pesquisa de 2015 mostra que projetos que utilizam Scrum ou métodos similares têm 3,5 vezes mais chances de sucesso (The Standish Group, 2015).

A adoção de Scrum em larga escala não significa que todos os problemas estão resolvidos. Longe disso. Scrum é apenas uma ferramenta que pode trazer diversos benefícios em comparação a outras formas de se conduzir projetos, mas somente se bem utilizada.

Scrum permite reduzir os riscos de insucesso, entregar valor mais rápido e, desde cedo, lidar com as inevitáveis mudanças de escopo, transformando-

as em vantagem competitiva. Seu uso pode também aumentar a qualidade do produto entregue e melhorar a produtividade das equipes.

Ele vem sendo adotado com sucesso por organizações de diversos tamanhos e tipos. De multinacionais a *startups*, de famosas a desconhecidas. Seu uso não se limita a projetos de desenvolvimento de *software*, embora tenha sido concebido com essa finalidade. Scrum é hoje também usado em diferentes mercados que incluem empresas de *marketing* e de desenvolvimento de *hardware*, por exemplo.

Scrum é aplicado em projetos com características igualmente variadas. Em projetos críticos de centenas de milhares de dólares e em projetos internos simples. Em projetos para produção de *softwares* comerciais, de *sites* da internet, de *softwares* embarcados, de aplicativos para dispositivos móveis, de *softwares* financeiros e de jogos.

Ao aprender Scrum, você passará por termos como facilitação, trabalho em equipe, auto-organização, metas de negócios, motivação, relacionamento com os clientes, entre tantos outros. Scrum utiliza-se de poucos conceitos novos, e essa é uma de suas grandes qualidades: juntar práticas de mercado já conhecidas e consagradas de uma forma organizada e que pode funcionar.

Mas por que Scrum é uma boa escolha para o projeto de desenvolvimento de produtos, para as organizações que estão desenvolvendo esses produtos e para os seus clientes? Uma gama de respostas para essa pergunta é dada ao longo deste livro. Neste capítulo, sumarizo algumas delas.

É importante lembrar de que não existe uma solução única para todos os problemas. Scrum é um *framework* simples e pequeno e, assim, funciona bem em cada contexto se for usado em conjunto com outras técnicas e práticas a serem escolhidas, experimentadas e adaptadas.

Os benefícios no uso do Scrum incluem:

- entregas frequentes de retorno ao investimento dos clientes;
- redução dos riscos do projeto;
- maior qualidade no produto gerado;
- mudanças utilizadas como vantagem competitiva;

- visibilidade do progresso do projeto;
- redução do desperdício;
- aumento da motivação e produtividade.

1.2 ENTREGAS FREQUENTES DE RETORNO AO INVESTIMENTO DOS CLIENTES

Em projetos desenvolvimento de *software*, ainda é comum haver apenas uma entrega, realizada ao seu final ou ao final de uma grande etapa. Assim, investe-se no desenvolvimento por um longo tempo sem qualquer retorno, que ocorrerá apenas como resultado da entrega.

Scrum possibilita que se entreguem partes do produto desde cedo e frequentemente, ao longo do projeto. Cada uma dessas partes, somada às anteriores, representa um produto funcional, gerado a partir das maiores necessidades para os clientes e seus usuários no momento da entrega. Essas partes, uma vez importantes, serão utilizadas. E, uma vez utilizadas, representam retorno ao investimento realizado.

Uma consequência das entregas frequentes desde cedo é que o produto pode ser introduzido no mercado em um curtíssimo prazo. As entregas frequentes também possibilitam o *feedback* rápido sobre o produto em construção, que guia seu desenvolvimento e permite a realização de mudanças necessárias.

1.3 REDUÇÃO DOS RISCOS DO PROJETO

Scrum não oferece um processo formal de gestão de riscos. Em vez disso, a redução dos principais riscos é inerente ao próprio processo de desenvolvimento do produto com Scrum.

É apenas ao entrar em contato com partes funcionais do produto que o usuário começa a ser capaz de entender melhor como esse *software* pode ajudá-lo a resolver suas necessidades. Como consequência, uma das práticas de mercado mais arriscadas é a de se demorar a oferecer ao usuário algo pronto que ele possa ver, utilizar e, então, oferecer o *feedback* necessário para

guiar a construção do produto. Quanto mais se posterga essa possibilidade de *feedback*, maior é o desperdício gerado com a construção de um produto realizada sob premissas erradas.

Com Scrum, o produto é construído em ciclos curtos, incrementalmente e com entregas frequentes, partindo-se das suas partes mais importantes. Busca-se a colaboração com os usuários, clientes e demais partes interessadas durante todo o projeto. Eles aprenderão sobre o que está sendo desenvolvido ao longo do trabalho, à medida que veem ou utilizam as partes do produto demonstradas e, preferencialmente, entregues.

Assim, busca-se, a partir de seu *feedback*, desenvolver e entregar, pouco a pouco, o produto certo. Da mesma forma, quando se entende que decisões equivocadas foram tomadas, esse *feedback* rapidamente gera visibilidade sobre o problema e permite a correção do rumo.

O atraso na entrega é outro dos grandes riscos mais comuns em projetos de desenvolvimento de *software*. De acordo com o Chaos Report de 2015, um estudo do reconhecido grupo de pesquisa norteamericano Standish Group, apenas 11% dos projetos que fixam escopo, orçamento e prazo terminam com o orçamento e no prazo previstos e com o cliente satisfeito (The Standish Group, 2015).

O trabalho com Scrum é priorizado. Uma vez que as partes mais importantes do produto são produzidas primeiro e em ciclos curtos, raramente é necessário se atrasar uma entrega. Ao se chegar à data preestabelecida, as chances são de que um valor suficiente já terá sido produzido para se realizar os objetivos de negócios da entrega. Restarão, talvez, apenas as partes menos importantes do que foi inicialmente pensado ou planejado, e que não são essenciais para os objetivos da entrega. Da mesma forma, ao final do prazo estabelecido para todo o projeto, é natural esperar que a visão inicialmente definida tenha sido realizada por se terem resolvidos os problemas mais importantes.

Gargalos durante o processo de desenvolvimento também representam importantes riscos em projetos de *software*. O time que desenvolve o produto com Scrum deve ser multifuncional, possuindo em seus membros todos os conhecimentos e habilidades necessários para gerar o produto, de ponta a ponta. Esse time produz partes do produto prontas em cada ciclo curto de

desenvolvimento. Caso identifique uma habilidade necessária em que é deficiente, o time deve buscar formas de incorporar o aprendizado suficiente para ser capaz gerar o produto.

O uso de times multifuncionais elimina ou, ao menos, minimiza dependências externas. Ou seja, busca-se eliminar gargalos no desenvolvimento como a espera por entradas ou, na outra ponta, por validação. Dessa forma, reduzem-se os riscos de não se terem as partes do produto prontas em cada ciclo de desenvolvimento.

Além disso, ao trabalharem juntos, o conhecimento vai gradualmente se distribuindo entre os membros do time, que aprendem uns com os outros. O conhecimento compartilhado ajuda a minimizar as dependências internas, reduzindo-se ainda mais possíveis gargalos.

1.4 MAIOR QUALIDADE NO PRODUTO GERADO

As partes do produto só estarão prontas, em cada ciclo de desenvolvimento, se possuírem a qualidade necessária para serem entregues aos clientes do projeto. O time que trabalha com Scrum é integralmente responsável pelo trabalho de geração do produto e, assim, possui todas as habilidades e conhecimentos necessários para fazê-lo.

É parte integral disso, portanto, realizar dentro do ciclo de desenvolvimento qualquer validação que seja necessária para garantir a qualidade desse produto, ou seja, que as partes do produto geradas estejam funcionando. Essa validação inclui, entre outros, quaisquer tipos de testes — técnicos e de negócios — que se façam necessários, a integração com o resto do produto já gerado e integrações com outros produtos, caso exista esse tipo de dependência.

Para o desenvolvimento de *software*, ferramentas de automação para testes e integração contínua auxiliam nessas tarefas. A princípio, não existem times externos responsáveis por avaliar ou garantir a qualidade do produto gerado, já que esse trabalho é de inteira responsabilidade do próprio time que desenvolve o produto.

Ao final de cada ciclo de desenvolvimento, uma validação de negócios da parte do produto gerada ocorre em uma reunião chamada de Sprint Review.

A partir de uma demonstração do que foi produzido pelo time, clientes do projeto e demais partes interessadas fornecem seu *feedback*.

Um *feedback* ainda mais profundo e valioso é obtido dos usuários do produto ao utilizarem as partes do produto já entregues, o que é possibilitado pelas entregas frequentes. Em ambos os casos, o produto será modificado, a partir do *feedback* obtido, para melhor atender aos clientes e usuários do projeto.

Ao se anteciparem as validações do que é produzido, há muito mais chances dos problemas ou modificações necessários serem detectados cedo. Quando essas validações são postergadas, os problemas podem se acumular e gerar atrasos no projeto. As consequências podem ser drásticas: não haverá tempo para se realizarem todas as validações ou todas as mudanças necessárias e, assim, partes do produto serão entregues com baixa qualidade.

1.5 MUDANÇAS UTILIZADAS COMO VANTAGEM COMPETITIVA

Em um projeto com Scrum, os *feedbacks* que levarão a mudanças no produto, ao longo do seu desenvolvimento, são acolhidos como oportunidades de se aumentar o valor entregue, e não como acontecimentos indesejáveis. Aceitar a mudança possibilita uma resposta rápida às mudanças de mercado.

Mas o significado do termo “mudança” vai além de simplesmente “alterar algo que já foi planejado ou realizado”. As necessidades de negócios e consequentes especificações e planos não são definidos detalhadamente no princípio do projeto com Scrum. Ao contrário, esses detalhes do produto são descobertos e reavaliados ao longo do desenvolvimento, a partir do *feedback* frequente dos seus usuários, clientes e demais partes interessadas.

Esse *feedback*, que é oferecido sobre partes prontas e funcionais do produto, possibilita que ele seja incrementalmente construído e modificado para melhor atender às necessidades. Aceitar a mudança, portanto, é essencial para garantir que o produto entregue realmente signifique valor.

1.6 VISIBILIDADE DO PROGRESSO DO PROJETO

Diversas práticas e artefatos do Scrum ou associados a ele visam a garantir a visibilidade (ou transparência) do progresso do projeto para seus participantes e envolvidos.

Em um projeto com Scrum, os clientes e demais partes interessadas estão muito mais próximos, e colaboram com o time na evolução do produto. Ao final de cada ciclo curto de desenvolvimento, é apresentado a eles o que foi desenvolvido e está funcionando. Mais determinante ainda, os clientes e demais partes interessadas recebem com frequência as próximas partes mais importantes do produto. Assim, ao contrário do que ocorre em projetos cuja entrega é realizada após um grande período, o senso de progresso no projeto é real, e não baseado em percentuais ou etapas cumpridas de um longo plano.

O time que desenvolve o produto também tem grande visibilidade de seu próprio progresso. O *feedback* que o time obtém nas reuniões de Sprint Review, diretamente dos clientes e demais partes interessadas, permite descobrir desde cedo, ao final de cada ciclo, se ele está se ele está na direção certa e o quanto eficiente seu trabalho está sendo.

A Daily Scrum é uma reunião diária realizada pelo time exatamente para criar visibilidade sobre o trabalho para os seus próprios membros e, assim, tornarem-se capazes de planejar o que será realizado até o próximo dia.

Além dessas ferramentas, existem outras que podem ser utilizadas para aumentar a visibilidade do progresso do trabalho, como o Quadro de Tarefas e os Gráficos de Acompanhamento do Trabalho, ambos explicados posteriormente neste livro.

As reuniões de Sprint Retrospective, também realizadas ao final de cada ciclo, permitem criar-se visibilidade sobre como o time está realizando seu trabalho. Assim, o time pode inspecionar e adaptar suas práticas para se tornar mais eficiente.

1.7 REDUÇÃO DO DESPERDÍCIO

O desperdício é reduzido e evitado com Scrum principalmente por meio da busca pela simplicidade. A ideia central é que o time produza e utilize, no

processo de desenvolvimento do produto, apenas o que é necessário e suficiente.

Em seguida, identifico como as regras e práticas do Scrum ajudam a evitar alguns tipos de desperdícios:

- produzir apenas o que os usuários vão utilizar;
- planejar apenas com o nível de detalhes possível;
- utilizar apenas os artefatos necessários e suficientes.

1.7.1 Produzir apenas o que os usuários vão utilizar

Um estudo de 2001 menciona os resultados de uma pesquisa em que foram analisados quatrocentos projetos de desenvolvimento de *software* durante quinze anos, em uma época em que pouquíssimos projetos utilizavam Scrum e similares. Em nenhum dos projetos analisados, mais de 20% das funcionalidades entregues foram utilizadas por seus usuários.

Para piorar, apenas entre uma em cada dez e uma em cada cem linhas de código produzidas foram de fato entregues, variando com a complexidade e tamanho do projeto (COHEN et al., 2001). Esses números mostram um gigantesco desperdício gerado no desenvolvimento de *software*.

Em 2002, o presidente do Standish Group apresentou em uma conferência na Itália os resultados de uma pesquisa limitada, que mostrou que mais da metade das funcionalidades produzidas em projetos de *software* nunca ou raramente eram utilizadas (veja a figura 1.1). Embora as origens desses números sejam nebulosas, a pesquisa passou a ser largamente referenciada.

Esse enorme desperdício ocorre principalmente quando se definem os requisitos do produto para um longo período de desenvolvimento, sem ou com pouca possibilidade de *feedback* nesse caminho. Mesmo que se despenda muito tempo em reuniões para levantamento e análise de requisitos, ao se exigir dos clientes ou das pessoas de negócio que decidam os detalhes de um produto de *software* a ser construído durante meses adiante, um grande desperdício será inevitavelmente gerado. Eles listarão tudo o que puderem imaginar, pois somente depois desses meses de desenvolvimento haverá uma nova oportunidade de intervirem no produto.

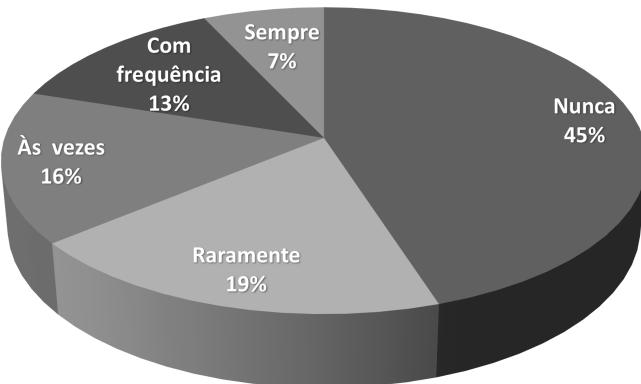


Fig. 1.1: Percentual de uso das funcionalidades em software, de acordo com o presidente do Standish Group em 2002

Com Scrum, os detalhes do produto emergem ao longo de seu desenvolvimento, a partir do *feedback* obtido dos clientes e demais partes interessadas. O trabalho de desenvolvimento do produto é priorizado e realizado a partir de suas necessidades e, assim, as partes entregues do produto deverão ser usadas. Ainda que exista um desperdício, uma vez que partes já prontas do produto são modificadas a partir do *feedback*, este não é comparável àquele gerado quando se tentam prever os detalhes antecipadamente.

Outro princípio importante é que, ao se gerar o produto a partir da solução mais simples que funcione, evita-se o desperdício. Uma solução complexa e custosa, por mais bela que seja, em geral oferece menor retorno sobre o investimento do que uma solução mais simples que funcione.

1.7.2 Planejar apenas com o nível de detalhe possível

Métodos tradicionais buscam planejar, no início do projeto, todo o trabalho a ser realizado no desenvolvimento do produto (ou, ao menos, da próxima entrega) com um alto nível de detalhes. Os planos gerados muitas vezes descrevem cada uma das funcionalidades do produto, quem as produzirão e quando serão produzidas. O resultado é que esses planos, em geral, não traduzem a realidade futura.

Scrum não possui essa fase inicial de planejamento detalhado, mas isso não significa que o projeto não possa ser descrito desde o seu início até o fim. O princípio básico é, no entanto, que se planeje apenas com o nível de detalhes que é possível de se obter.

Para um projeto com Scrum, pode-se utilizar uma visão de produto que traduza seus objetivos. Para realizar essa visão, pode-se definir um plano em alto nível que preveja as principais metas de negócios a serem cumpridas em cada entrega e saber quando isso acontecer aproximadamente. Para o trabalho a ser realizado até a próxima entrega, pode-se criar um plano, no qual serão definidos uma data para a entrega, o trabalho provável a ser realizado e o próximo objetivo ou meta de negócios a ser cumprida a partir desse trabalho, que é um incremento à realização da visão do produto.

Para o ciclo de desenvolvimento atual, o time planeja, com detalhes, que trabalho vai realizar nesse ciclo e como vai realizá-lo, visando a cumprir uma meta de negócios, que por sua vez é um incremento à realização da meta estabelecida para a entrega.

Exceto pelo plano do ciclo de desenvolvimento, qualquer um desses planos é modificado com frequência, de forma a refletir a realidade que se apresenta em cada momento. Esse comportamento é melhor demonstrado pela analogia do horizonte.

Analogia do horizonte

Digamos que você está olhando para uma cidade e pode enxergar desde as construções, carros e pessoas que estão mais próximos de você até o que está no horizonte distante.

Ao olhar para o que está mais próximo, você consegue descrever o que vê com um excelente nível de detalhes e com uma granularidade fina, ou seja, em pequenas porções de informação. O que está um pouco mais distante, você ainda pode descrever com um bom nível de detalhes, ainda que menor, e com uma granularidade um pouco mais grossa.

Mas, à medida que olha para mais longe, os detalhes que você pode usar em sua descrição vão gradualmente diminuindo e a granularidade deve ficar cada vez mais grossa. Se você então tentar descrever tudo o que vê com um alto nível de detalhes, ao caminhar em direção ao horizonte, verá que sua des-

crição não corresponde inteiramente à realidade e que você deverá modificar essa descrição.

Na verdade, para quanto mais distante você estiver olhando, mais incorreta estará essa sua descrição detalhada e maior será o desperdício gerado. Mas se, ao contrário, você descrever apenas o que consegue enxergar, ao caminhar na direção do horizonte poderá refinar a sua descrição, adicionando detalhes à medida que os enxerga.

Um planejamento tradicional geralmente descreve em detalhes o que será feito em todo o projeto ou, ao menos, todo o trabalho até a próxima entrega. É muito comum ver esses planos descritos em gráficos de Gantt, que mostram tarefas detalhadas e alocação de “recursos” no tempo.

Essa prática pode ser comparada a se descrever detalhadamente todo o caminho, desde o que está mais próximo de você até o que está mais próximo da linha do horizonte. O seu resultado é um plano de altíssima precisão, porém com baixíssima chance de acerto, levando assim à geração de desperdício.

Em um planejamento com Scrum, somente se planeja com o nível de detalhes que se pode enxergar. Para planejar-se um trabalho a ser realizado, por exemplo, até o próximo dia, pode-se utilizar um nível de detalhes bastante alto e uma granularidade bem fina. Para as próximas duas semanas — um ciclo de desenvolvimento, por exemplo — pode-se usar um nível de detalhes ainda razoavelmente alto, porém mais baixo do que para apenas um dia.

Já ao se planejar uma entrega que acontecerá daqui a dois meses ou ao se planejar o ano inteiro de projeto, a quantidade de detalhes diminui e a granularidade fica mais grossa quanto mais longe se olha no tempo, de forma que pouquíssimos detalhes podem ser utilizados para planejar o que está mais distante. À medida que o projeto caminha no tempo, esse plano deve ser refinado e mais detalhes devem ser gradualmente obtidos.

A lista de necessidades de negócios do Scrum, chamada de Product Backlog, reflete essa forma de planejamento. Os itens do topo do Product Backlog possuem uma granularidade mais fina. Ou seja, são itens menores e que representam mais detalhes. Ao se descer no Product Backlog, os itens vão ficando cada vez maiores e com menos detalhes. Mais detalhes do que o necessário e suficiente significam geração de desperdício.

1.7.3 Utilizar apenas os artefatos necessários e suficientes

Ferramentas são úteis até o ponto em que começam a gerar desperdício. Aquelas usadas no suporte à gestão do projeto e ao desenvolvimento do produto devem cumprir seu papel sem adicionar burocracia ao trabalho. De forma geral, as ferramentas mais simples e que funcionam geram os melhores resultados.

Ferramentas utilizadas pelo time para esboçar, planejar e monitorar seu trabalho de desenvolvimento do produto são mais bem representadas em meios físicos do que em meios virtuais. Quadros brancos na parede são bons exemplos de meios para essas ferramentas. Eles, enquanto irradiadores de informação, trazem uma visibilidade que não é possível, por exemplo, ao se usarem ferramentas de *software*. Estas últimas, quando mal utilizadas, terminam por impor mais empecilhos do que facilitar o trabalho, gerando um desperdício bastante custoso para o projeto.

Documentos que refletem planos, esquemas e especificações também podem ser úteis no suporte à gestão do projeto e ao desenvolvimento. O time, no entanto, evita o desperdício ao se concentrar em criar e manter apenas aquilo que será, de fato, utilizado.

DOCUMENTAÇÃO SUFICIENTE E NECESSÁRIA

Imagine um projeto em que, por burocracia da organização ou por tentativa de se seguir alguma metodologia específica, é exigida a geração de uma grande quantidade de documentos de especificações. Em muitos casos, esse trabalho é tão grande quanto ou até maior do que o trabalho de se construir o produto e, na realidade, apenas uma pequena parte dessa documentação é realmente necessária. Assim, um grande desperdício é gerado.

Nesse mesmo cenário, imagine que, em algum momento no decorrer do projeto, você necessite de consultar um item específico dessa documentação. As chances são de que, por haver tanto a ser atualizado, esse item não estará mais correspondendo à realidade e, assim, não será mais útil nesse estado. Portanto, um desperdício ainda maior se caracteriza.

Produzir documentação além do suficiente e necessário é um desperdício. A documentação em excesso desloca o foco daquela que é realmente necessária, gerando um desperdício ainda maior.

1.8 AUMENTO DA MOTIVAÇÃO E PRODUTIVIDADE

Diversos fatores potencializam a produtividade de times que utilizam Scrum. Entre esses fatores, podemos citar:

- o trabalho em equipe e a autonomia do time na realização desse trabalho;
- a existência de facilitação e de remoção de impedimentos;
- a melhoria contínua dos processos de trabalho;
- um ritmo sustentável de trabalho;
- realização do trabalho de ponta a ponta.

Um aumento na motivação dos membros do time é, ao mesmo tempo, consequência de todos os fatores descritos e causador de um aumento na produtividade. A estes fatores que aumentam a motivação e, consequentemente, a produtividade, soma-se o time trabalhar em um ambiente que apoie seu trabalho, o que inclui a disponibilidade de todas as ferramentas necessárias para realizar esse trabalho, e a confiança da organização e de sua gerência na capacidade do time em tomar as decisões que lhe cabem.

1.8.1 Trabalho em equipe e autonomia

O time de um projeto com Scrum é propositalmente pequeno, de forma que seus membros possam interagir e se comunicar com eficiência durante todo o dia para realizar o trabalho. Esse trabalho em equipe é considerado essencial para o sucesso do projeto.

A responsabilidade na geração do produto não recai apenas sobre o indivíduo, mas sobre o time como um todo, que tem o poder e deve se organizar

para decidir o quanto é possível de se fazer em cada ciclo de desenvolvimento, definir como vai fazê-lo e monitorar seu próprio progresso.

O time gera cada parte do produto em direção a uma meta de negócios bem definida, que os guia e dá propósito ao seu trabalho. Essa meta é dimensionada pelo próprio time, que a ajusta de forma que possa se comprometer com ela. Os membros do time, então, tornam-se igualmente responsáveis por realizá-la, o que estimula a cooperação entre eles em busca desse objetivo. Assim, a responsabilidade sobre o trabalho não é individual, mas sim coletiva.

Essa autonomia ajuda o time a criar um senso de propriedade sobre seu trabalho, de forma que os próprios membros do time estimulam, cobram e ajudam seus colegas a fazerem o seu melhor para realizarem a meta. Esse comportamento naturalmente leva o time a ser mais motivado e produtivo.

1.8.2 Facilitação e remoção de impedimentos

O time conta com a ajuda de um facilitador para se tornar mais produtivo.

Esse facilitador, chamado de ScrumMaster, ensina Scrum ao time, ajudando a ganhar autonomia e a aprender a se auto-organizar, facilita suas reuniões e remove os impedimentos que ameaçam as metas a serem realizadas.

1.8.3 Melhoria contínua

Um time que utiliza Scrum está sempre em busca de melhorar a forma como realiza seu trabalho, para se tornar mais eficiente. A reunião de Sprint Retrospective, realizada ao final de cada ciclo de desenvolvimento, busca garantir essa prática. Nessa reunião, pontos a melhorar são levantados e discutidos, e planos de ação para colocar as melhorias em prática são definidos.

Na busca por melhorias em seu dia a dia de trabalho, o time é estimulado a experimentar, inspecionar e adaptar-se de acordo. No entanto, tão importante quanto encontrar técnicas ou práticas que melhorem seu trabalho é descobrir cedo as que não funcionam adequadamente, para então modificá-las ou descartá-las. É melhor falhar cedo para aprender com a falha em vez de postergar a validação de uma prática ou técnica em uso que pode não estar funcionando.

A melhoria contínua promovida pelo time o leva a um aumento progressivo da sua eficiência na realização do trabalho, o que por sua vez o leva a um aumento na sua produtividade.

1.8.4 Ritmo sustentável de trabalho

As práticas do uso intensivo de horas-extras e de aceleração forçada do ritmo de trabalho, embora muito utilizadas para se cumprir um trabalho no prazo determinado, geram estresse no time que desenvolve o produto, destroem a motivação e não tardam em derrubar sua produtividade e a qualidade do produto gerado. Times que utilizam Scrum buscam conseguir o efeito contrário com a obtenção de um ritmo constante e sustentável no trabalho de criação do produto.

Cada ciclo de desenvolvimento possui uma duração fixa. De forma similar, todas as reuniões prescritas pelo Scrum possuem uma duração máxima dentro da qual devem ocorrer. São os chamados *timeboxes* que, quando respeitados com rigor, auxiliam o time a alcançar esse ritmo constante e sustentável de trabalho.

Para um ciclo de desenvolvimento do Scrum, por exemplo, deve-se escolher uma duração fixa entre uma e quatro semanas, durante a qual o time trabalha para realizar uma meta de negócios. Nenhuma pressão deve ser exercida sobre o time para que se comprometa com mais trabalho do que acredita que pode realizar nesse *timebox*. Assim, o time naturalmente tende a atingir um ritmo constante e sustentável, pois trabalha sempre em ciclos de desenvolvimento com a mesma duração e tem a autoridade para definir quanto trabalho acredita que será capaz de realizar em cada ciclo.

A reunião de Sprint Planning é outro exemplo de *timebox*. Essa reunião tem como objetivo criar um plano para o ciclo de desenvolvimento, e não deve durar mais que oito horas para ciclos com duração de quatro semanas. As outras reuniões do Scrum também têm duração máxima definida.

Mesmo quando o time, perto do final do período estabelecido, tiver certeza de que não cumprirá com o objetivo do ciclo ou da reunião, seu *timebox* não será estendido. Da mesma forma, disfunções como horas extras e a aceleração forçada do ritmo de trabalho devem ser evitadas. Ao contrário, é mais importante o time chegar ao final do *timebox* e poder falhar.

Essa falha, no entanto, deve gerar um aprendizado que ajude o time a melhorar e, eventualmente, não repeti-la no futuro. Isso reforçará a criação de um ritmo sustentável e poderá levar a um aumento de produtividade.

1.8.5 Realização do trabalho de ponta a ponta

O time que usa Scrum realiza o trabalho de desenvolvimento do produto de ponta a ponta, ou seja, gera, em cada ciclo de desenvolvimento, partes do produto prontas para serem entregues. Esse time não produz partes fragmentadas ou camadas isoladas de um produto, nas quais dificilmente se reconhece o valor produzido.

Ao final de cada ciclo, os membros do time ficam frente a frente com clientes e demais partes interessadas para coletar seu *feedback* e, assim, o valor gerado lhes fica ainda mais evidente. Ao produzir um valor identificável, esse trabalho gera neles um senso de realização, aumentando sua motivação. Além disso, o *feedback* obtido nessa reunião de revisão busca garantir que o time produza apenas o que gera valor e, assim, reduza o desperdício, aumentando sua produtividade.

Para que seja capaz de realizar o trabalho de ponta a ponta, o time deve ser multidisciplinar. Ou seja, deve possuir todos os conhecimentos e habilidades necessários para a geração do produto.

Os membros do time multidisciplinar, ao trabalharem juntos e compartilharem responsabilidade sobre as diversas tarefas a serem realizadas, trocam conhecimentos e são estimulados a desenvolverem diversas competências, o que é outro fator de motivação no trabalho.

CAPÍTULO 2

O que é Scrum?

2.1 INTRODUÇÃO

O capítulo anterior tratou dos benefícios de se utilizar Scrum, ou seja, por que ele pode nos ajudar a obtermos projetos de sucesso. Mas então o que é Scrum, afinal? Busquei criar uma definição resumida, que pode ser lida a seguir:

Scrum é um *framework* Ágil, simples e leve, utilizado para a gestão do desenvolvimento de produtos complexos imersos em ambientes complexos. Scrum é embasado no empirismo, e usa uma abordagem iterativa e incremental para entregar valor com frequência, assim, reduzindo os riscos do projeto.

Vamos entender em seguida o que cada um desses termos significa.

2.2 SCRUM É ÁGIL

Uma definição comum para “ágil” poderia ser: *que se movimenta com facilidade; ligeiro, leve.*

O nome “Ágil” (ou “Agilidade”) foi escolhido para representar um movimento que surgiu em meados dos anos 90 em resposta aos pesados métodos de gerenciamento de desenvolvimento de *software* que predominavam na época, que aqui chamo de “métodos tradicionais”.

O método tradicional que é mais conhecido para o desenvolvimento de *software* é o modelo em cascata, ou *waterfall*. Este foi inicialmente descrito por Royce em 1970 e se caracteriza por uma sequência de fases de desenvolvimento, em que cada fase somente se inicia quando a anterior se encerra, e a saída de uma fase é a entrada da fase seguinte, como mostrado na figura 2.1. Royce, no entanto, criticava o modelo em seu artigo, afirmando que, para o desenvolvimento de *software*, seu uso era arriscado.

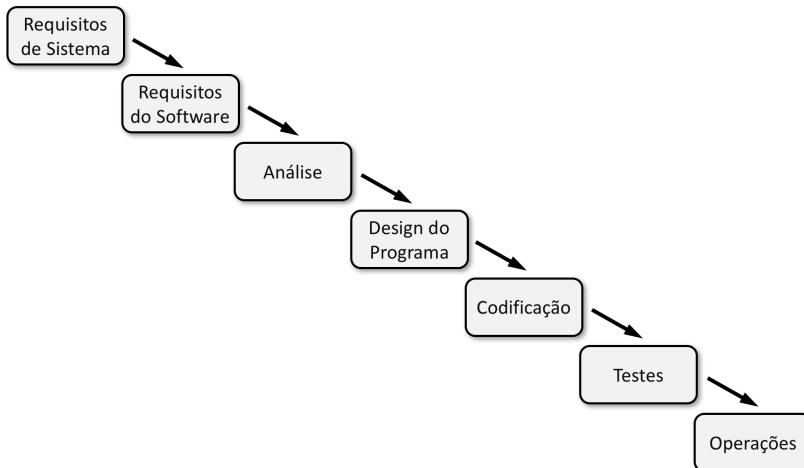


Fig. 2.1: O modelo em cascata ou *waterfall*, conforme definido por Royce (1970)

Os métodos tradicionais são fortemente prescritivos e, de forma geral, se caracterizam pelo foco em planos detalhados e definidos no princípio do

projeto (com custo, escopo e um cronograma detalhado), em microgerenciamento do trabalho com o poder centralizado, em processos cada vez mais complicados e em extensa documentação. Mudanças são fortemente indesejadas.

Acreditava-se que, pelo uso desses métodos, seria possível tratar o desenvolvimento de *software* como um processo previsível, no que, quanto mais falhavam, mais pesados e complexos esses métodos se tornavam.

Scrum é Ágil porque, assim como outros métodos, metodologias e *frameworks*, sua utilização deve seguir os princípios e valores do **Manifesto para o Desenvolvimento Ágil de Software**. Esse manifesto foi criado em fevereiro de 2001 em uma reunião que aconteceu na estação de esqui de Snowbird no estado de Utah, Estados Unidos. Assinaram-no dezessete líderes representantes de ideias, metodologias e processos que, em contraste com as práticas predominantes na época, estavam trazendo valor para seus clientes por meio de abordagens leves e empíricas para projetos de desenvolvimento de *software*.

Nesse encontro histórico, não havia entre os participantes a intenção de unificar suas formas de trabalhar. A expectativa de se chegar a qualquer tipo de consenso era limitada e variada. No entanto, apesar das diferentes práticas defendidas, os participantes encontraram um conjunto de valores em comum e, ao final de três dias, estabeleceram o termo “Ágil” para representar o novo movimento. Neste livro, eu tomo a licença poética de escrever “Ágil” ou “Agilidade” com “A” maiúsculo para identificar tudo o que está relacionado ao movimento Ágil.

O Manifesto Ágil, como ficou conhecido, pode ser encontrado em <http://agilemanifesto.org> em diversas línguas, inclusive em português do Brasil (eu mesmo participei desse esforço de tradução). Ao procurar na internet pelos nomes dos signatários originais do Manifesto e por suas ideias, pode-se ver o quanto representativa a grande maioria deles ainda é.

Tive o prazer de conversar e interagir com um dos signatários do Manifesto, Mike Beedle, durante o Scrum Gathering do Rio de Janeiro de 2014 (<http://scrumrio.com>) . Foi Beedle quem sugeriu o termo “Ágil” durante aquela reunião e ele nos mostrou uma lista dos nomes que foram cogitados na época. Ele é coautor do primeiro livro de Scrum no mundo, publicado

em 2002 (SCHWABER; BEEDLE, 2002) e é um Certified Scrum Trainer da Scrum Alliance (<http://scrumalliance.org>) , assim como eu.

Scrum é a escolha de duas em cada três pessoas que utilizam métodos, metodologias ou *frameworks* que seguem o Manifesto Ágil, de acordo com uma pesquisa realizada em 2015 (VersionOne, 2016).

Discussão: a analogia errada

Desde quase o começo da história do desenvolvimento de *software*, a comparação com a construção civil foi largamente utilizada para descrever esse tipo de projeto. São, no entanto, trabalhos de naturezas muito distintas.

Embora tenham evoluído ao longo dos tempos, projetos de construção existem há eras na história da humanidade e, em linhas gerais, sua forma de execução se manteve a mesma: uma longa fase de definições e especificações no início que tem como saída um plano, seguida de sua fase de execução. Parece natural para o ser humano comparar outros tipos de trabalho com um que lhe seja tão familiar.

Os métodos tradicionais de desenvolvimento de *software* buscaram algo similar com o modelo em cascata e suas fases sequenciais de levantamento e análise de requisitos, especificação, desenvolvimento e testes. Ainda hoje, é comum usarem-se as expressões “engenharia ou engenheiro de *software*”, “arquitetura ou arquiteto de *software*”, e até mesmo “construção de *software*”, todas provindas da analogia com a construção civil.

O livro *Wicked problems, righteous solutions* (DEGRACE; STALL, 1990 apud SUTHERLAND, 2004), no entanto, já descrevia em 1990 as razões por que métodos tradicionais de desenvolvimento de *software* não funcionam, a partir das seguintes prerrogativas básicas:

- requisitos não são completamente compreendidos antes do início do projeto;
- usuários só sabem exatamente o que querem após ver uma versão inicial do produto;
- requisitos mudam frequentemente durante o processo de desenvolvimento;

- novas ferramentas e tecnologias tornam as estratégias de desenvolvimento imprevisíveis.

Os autores definem *software* como um tipo de problema que não pode ser claramente definido e detalhado de antemão, já que isso somente pode ser feito a partir de soluções. Em outras palavras, cada tentativa de se criar uma solução modifica a própria compreensão do problema. Assim, não se pode criar antecipadamente um plano detalhado para como o problema — o *software* — será resolvido.

Sabemos, portanto, que comparar projetos de desenvolvimento de *software* com projetos construção civil não faz sentido. A analogia simplesmente não funciona.

2.2.1 Os valores Ágeis

O Manifesto Ágil reconhece que a utilização de processos, ferramentas, documentação, contratos e planos pode ser importante para o sucesso do projeto, mas são ainda mais importantes os chamados valores Ágeis: os indivíduos e interações entre eles, *software* (ou produto) em funcionamento, colaboração com o cliente e responder a mudanças.

MANIFESTO PARA DESENVOLVIMENTO ÁGIL DE SOFTWARE

Estamos descobrindo maneiras melhores de desenvolver *software* fazendo-o nós mesmos e ajudando outros a fazê-lo. Por meio deste trabalho, passamos a valorizar:

- **Indivíduos e interações** mais do que processos e ferramentas.
- **Software em funcionamento** mais do que documentação abrangente.
- **Colaboração com o cliente** mais do que negociação de contratos.
- **Responder a mudanças** mais do que seguir um plano.

Ou seja, mesmo havendo valor aos itens à direita, valorizamos mais os itens à esquerda.

Apresento a seguir a visão de dois signatários originais do Manifesto, Alistair Cockburn e Jim Highsmith, e de um autor relevante, Craig Larman, acerca do significado de cada um desses valores.

Indivíduos e interações

Para Jim Highsmith, valorizar indivíduos e interações mais do que processos e ferramentas leva em conta que, em última instância, quem gera produtos e serviços são os indivíduos, que possuem características únicas individualmente e em equipe, como talento e habilidade (HIGHSMITH, 2004).

Alistair Cockburn (2007) afirma que as pessoas na equipe são mais importantes do que seus papéis em diagramas de processos. Assim, embora descrições de processos possam ser necessárias para se começar o trabalho, as pessoas envolvidas nos processos não podem ser trocadas como peças.

Na mesma linha, Craig Larman (2003) lembra de que a programação de computadores é uma atividade humana e, assim, depende de questões humanas para seu sucesso. O autor cita como exemplo a vida social e familiar dos membros da equipe que são afetadas, por exemplo, pelo excesso de trabalho.

Highsmith complementa afirmando que são críticas para o sucesso dos projetos, as habilidades, personalidades e peculiaridades dos indivíduos, que são muitas vezes desorganizados e difíceis de entender, ao mesmo tempo em que são inovadores, criativos, exuberantes e apaixonados (HIGHSMITH, 2002).

Cockburn afirma ainda que a qualidade da interação entre os membros do time é importante para a solução de problemas no desenvolvimento do projeto (COCKBURN, 2007). Larman (2003), sobre esse tema, destaca que o uso da comunicação e do *feedback* é uma diretiva essencial para a prática Ágil, especialmente a partir da conversação face a face. Highsmith (2002) lembra sobre a importância do trabalho em equipe, afirmando que habilidades individuais e trabalho em equipe são inseparáveis em projetos de desenvolvimento de *software*.

Outro ponto levantado por Larman (2003) é que as ferramentas utilizadas para auxiliar o desenvolvimento de *software* devem ser as mais simples possíveis que funcionem. Projetos que utilizaram metodologias pesadas em termos de processos e ferramentas, segundo Highsmith (2002), obtiveram sucesso fundamentalmente devido às pessoas envolvidas naqueles projetos.

O autor afirma ainda que processos podem guiar e apoiar o desenvolvimento de *software* e ferramentas podem melhorar sua eficiência, mas é com a capacidade e conhecimento dos indivíduos que se pode contar para tomar decisões críticas para o desenvolvimento do projeto (HIGHSMITH, 2002). Bons processos ajudam a equipe em vez de ditar como seu trabalho deve ser feito, de forma que são os processos que se adaptam à equipe e não o oposto (HIGHSMITH, 2004). Ainda segundo Highsmith (2002), um processo não é um substituto para uma habilidade, de forma que segui-lo por si só não cria bons programas de computador.

Software em funcionamento

Para Alistair Cockburn (2007), *software* em funcionamento é o único indicador do que a equipe de fato construiu. Jim Highsmith (2002) afirma que clientes se interessam por resultados, ou seja, *software* em funcionamento que entregue valor de negócio.

Ainda segundo Cockburn, a documentação pode ser muito útil para o desenvolvimento do projeto, mas deve-se produzir somente a documentação

necessária e suficiente para a realização do trabalho. Highsmith, por sua vez, afirma que *software* em funcionamento não exclui a necessidade de documentação, pois ela pode permitir a comunicação e colaboração, melhorar a transferência de conhecimento, preservar informações históricas, ajudar melhorias em progresso e satisfazer a necessidades legais e regulatórias.

Segundo o autor, a documentação não é desimportante, ela simplesmente é menos importante do que versões em funcionamento do produto. Ainda segundo Highsmith, um erro comum em projetos é a crença de que a documentação substitui a interação, servindo como um meio de comunicação. A documentação, na realidade, facilita a interação, funcionando como seu sub-produto na forma de documentos, rascunhos, desenhos, anotações etc., que podem (ou não) ser utilizados como um registro permanente (HIGHSMITH, 2004).

Highsmith (2004) afirma ainda que a entrega iterativa de versões do *software* em funcionamento possibilita um *feedback* confiável no processo de desenvolvimento de formas que simplesmente com a documentação é impossível. Craig Larman (2003) denomina essa prática como “entrega evolutiva”, na qual há um esforço em se obter o máximo de *feedback* possível sobre o que foi entregue, de forma que a próxima entrega seja planejada dinamicamente, baseada nesse *feedback*.

Colaboração com o cliente

Para Alistair Cockburn, no desenvolvimento Ágil não há “nós” e “eles”, há simplesmente “nós”, o que significa que clientes e desenvolvedores estão do mesmo lado, colaborando para produzir o *software* que traga valor para esses clientes. Nessa dinâmica, ambos são necessários para que se produza *software* de boa qualidade. Segundo o autor, essa colaboração envolve companheirismo, tomada de decisão conjunta e rapidez na comunicação, de forma que muitas vezes pode até tornar contratos desnecessários (COCKBURN, 2007).

Segundo Jim Highsmith (2004), no desenvolvimento de novos produtos como *software*, em que há alta volatilidade, ambiguidade e incertezas, a relação cliente-desenvolvedor deve ser colaborativa, em vez de marcada por disputas de contrato antagônicas.

Responder a mudanças

De acordo com Jim Highsmith, todo projeto deve balancear o planejar com o mudar, dependendo do nível de incerteza inerente a ele. Segundo o autor, em projetos onde há alta incerteza, a investigação e a concepção mental predominam sobre o planejamento e a execução restrita de tarefas planejadas (HIGHSMITH, 2004).

Esse conceito se aplica ao desenvolvimento de *software*, já que, de acordo com Craig Larman (2003), a incerteza é inerente e inevitável em projetos e processos de *software*. Alistair Cockburn afirma que construir um plano é útil, mas seguir o plano só é útil até o momento em que ele ainda está próximo o suficiente da situação atual. Assim, manter-se preso a um plano ultrapassado não funciona em favor do sucesso (COCKBURN, 2007).

Highsmith (2004) destaca ainda que empresas que buscam prosperar na turbulenta economia atual devem alterar tanto seus processos quanto suas perspectivas em consideração à mudança, já que praticamente tudo, exceto a visão do produto, pode mudar em pouquíssimo tempo, como escopo, funcionalidades, tecnologia, arquitetura etc.

Segundo Cockburn (2007), iterações curtas de desenvolvimento permitem que mudanças possam ser mais rapidamente inseridas no projeto, de forma que atendam às novas necessidades dos clientes.

2.2.2 Os princípios Ágeis

Os doze princípios Ágeis foram cunhados a partir do Manifesto por parte de seus autores e podem ser vistos em seguida, cada um com uma pequena explicação adicional:

1) Nossa maior prioridade é satisfazer o cliente por meio da entrega cedo e frequente de *software* com valor.

- O foco no desenvolvimento do produto está na satisfação dos clientes. Gera-se, desde cedo e frequentemente, retorno ao investimento dos clientes no projeto a partir da entrega de partes do produto que atendam às suas necessidades. Esse princípio se opõe à prática de se seguir um plano detalhado, sugerindo que a prioridade está em

se adaptar e buscar, em cada momento, o que de fato trará valor aos clientes, para entregar-lhes o mais cedo e frequentemente possível.

2) Mudanças de requisitos são bem-vindas, mesmo em fases tardias do desenvolvimento. Os processos Ágeis utilizam a mudança em favor da vantagem competitiva para o cliente.

- Aceitar a mudança como natural no processo de desenvolvimento do produto, para melhor atender às necessidades dos clientes. Ao se trabalhar em ciclos curtos de *feedback*, permite-se aos clientes evoluirem o produto à medida que melhor entendem suas necessidades e adaptarem às mudanças de mercado, tornando-se mais competitivos. Esse princípio se opõe a se tratar o processo de desenvolvimento do produto como previsível, cenário irreal no qual a necessidade de mudança poderia e deveria ser prevenida, já que ela seria considerada indesejada e custosa.

3) Entregar *software* em funcionamento com frequência, desde a cada duas semanas até a cada dois meses, com uma preferência por prazos mais curtos.

- Entregar a seus clientes e usuários, com frequência, partes do produto prontas gera, a cada entrega, retorno ao investimento dos clientes e permite obter-se *feedback* sobre o que foi produzido. Assim, pode-se adaptar o produto às necessidades dos clientes incrementalmente, reduzindo-se os riscos do projeto. Esse princípio se opõe a realizar poucas ou, no limite, uma entrega de valor única, apenas ao final do projeto.

4) As pessoas do negócio e os desenvolvedores devem trabalhar em conjunto diariamente ao longo do projeto.

- Pessoas de negócio e desenvolvedores do produto possuem o objetivo comum de garantir a geração de valor para os clientes e, para

realizar esse objetivo, cooperam continuamente durante todo o projeto, interagindo com frequência. Esse princípio se opõe ao cenário de antagonismo comum em projetos de desenvolvimento de *software*, nos quais pessoas de negócios — que frequentemente incluem os próprios clientes do projeto — e desenvolvedores raramente se comunicam ou colaboram entre si e, muitas vezes, estão em lados opostos.

5) Construa projetos em torno de indivíduos motivados. Dê-lhes o ambiente e o suporte que precisam e confie neles para realizarem o trabalho.

- O produto é construído por pessoas. O ambiente, o suporte e a confiança necessários para realizar seu trabalho são fatores fundamentais para sua motivação. Esse princípio se opõe à crença de que o produto se constrói em torno das melhores ferramentas e processos e não das pessoas, apoiando-se nos melhores instrumentos de monitoração e controle externos, por exemplo.

6) O método mais eficiente e efetivo de se transmitir informação para e entre uma equipe de desenvolvimento é a conversa face a face.

- A melhor forma de comunicação entre membros do time que desenvolve o produto e entre esse time e o mundo externo é a comunicação face a face, que é direta, síncrona e enriquecida pela entonação de voz, olhar e linguagem corporal, entre outros fatores. Quando a comunicação presencial não é viável (em um projeto distribuído, por exemplo), é uma boa prática fazer-se o melhor uso possível da tecnologia disponível para se aproximar da comunicação face a face. Esse princípio se opõe à utilização de documentos, e-mails, telefone e teleconferência, entre outros, como formas padrão de comunicação em um projeto.

7) Software em funcionamento é a principal medida de progresso.

- O progresso do projeto ocorre à medida que partes do produto que signifiquem valor são entregues aos clientes do projeto. Esse princípio se opõe à prática de se gerar artefatos como protótipos e extensos

documentos de planos e especificações e, assim, acreditar que se progrediu no projeto, baseando-se em percentuais de completude, por exemplo. Esse princípio também se opõe à geração de quaisquer artefatos e partes do produto — inclusive documentação — que não gerem valor para os clientes do projeto.

- 8) **Os processos Ágeis promovem o desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter indefinidamente um ritmo constante.**
 - Busca-se promover um ritmo constante e sustentável para o trabalho do time que desenvolve o produto, o que se torna possível quando esse ritmo é apoiado por toda a cadeia, incluindo usuários e patrocinadores. No entanto, ao se exigir do time um compromisso com mais trabalho do que ele é capaz de produzir, são muitas vezes adotadas as horas extras, o trabalho em fins de semana e a pressa exagerada para se cumprir o prazo de entrega, por exemplo. Essas práticas podem levar a um desbalanço no fluxo de produção, o que acaba por gerar insatisfação nos membros do time de desenvolvimento, uma menor produtividade e uma menor qualidade no produto gerado.
- 9) **A atenção contínua à excelência técnica e a um bom projeto aumentam a agilidade.**
 - O produto projetado com qualidade e produzido com excelência técnica permite que possa facilmente ser modificado e, assim, que aceite a mudança como natural no processo de seu desenvolvimento. Dessa forma, a alta qualidade no produto gerado é essencial para se manter a Agilidade. Esse princípio se opõe à crença de que, para se obter velocidade e flexibilidade no desenvolvimento do produto, a qualidade deve ser sacrificada. Na realidade, ocorre exatamente o oposto.
- 10) **Simplicidade — a arte de se maximizar a quantidade de trabalho não feito — é essencial.**

- Evita-se o desperdício no desenvolvimento do produto ao não se realizar trabalho que não é necessário. Exemplos comuns de desperdícios incluem desenvolvimento de funcionalidades de que os clientes não precisam ou de soluções desnecessariamente complexas, planejamento com nível de detalhes maior do que se pode ter em um determinado momento e uso ou geração de artefatos desnecessários.
- 11) **As melhores arquiteturas, requisitos e projetos emergem de equipes que se auto-organizam.**
- Equipes com maior autonomia são mais eficientes. Essas equipes auto-organizadas trabalham em direção a metas acordadas, mas têm autonomia para decidirem quanto trabalho serão capazes de realizar, planejarem qual a melhor forma de realizá-lo e monitorarem seu progresso. Assim, se tornam responsáveis e são responsabilizadas por seus resultados. Desse trabalho, emergem, ao longo de todo o projeto, a arquitetura e os requisitos do produto.
- 12) **Em intervalos de tempo regulares, a equipe reflete sobre como se tornar mais efetiva e, então, refina e ajusta seu comportamento de acordo.**
- Para se tornar cada vez mais efetiva, a equipe regularmente inspecciona suas formas de trabalho, identifica pontos de melhoria e se adapta de acordo, promovendo a melhoria incremental contínua. É a inspeção e adaptação que o time realiza em seus processos de trabalho.

2.2.3 Os valores do Scrum

Além dos valores e princípios Ágeis, Scrum também tem o seu próprio conjunto de valores a serem seguidos. Eles complementam as regras do Scrum descritas em seus papéis, artefatos e eventos. Assim, os valores do Scrum representam os pilares para todo o trabalho realizado pelas pessoas que trabalham no projeto, ou seja, o time que desenvolve o produto e a pessoa de negócios que o define, chamada no Scrum de Product Owner.

Os cinco valores do Scrum são: foco, coragem, franqueza, compromisso e respeito (Scrum Alliance, 2012). Esses valores são explicados em detalhes abaixo:

- **foco:** os times mais produtivos trabalham em apenas um projeto de cada vez, evitando a multitarefa. O time trabalha focado em metas claras e realizáveis com as quais se comprometem. Ao mesmo tempo, essas metas, definidas e negociadas com o Product Owner, mantêm o foco do trabalho nas necessidades de negócios mais urgentes em cada momento. Além disso, para auxiliar o time a manter seu foco no seu trabalho, há no Scrum um facilitador que remove impedimentos e protege o time de distrações e interferências externas, chamado de ScrumMaster;
- **coragem:** as pessoas que trabalham no projeto têm coragem para aceitar a mudança como parte natural do processo de desenvolvimento do produto. O Product Owner e a organização têm coragem para confiar no time e deixá-lo livre para realizar o trabalho necessário para realizar as metas acordadas. O time tem coragem para falhar e criar visibilidade sobre os problemas e, assim, aprender com essas falhas e problemas o mais cedo possível. Tanto time quanto Product Owner têm coragem para mostrar e entregar com frequência as partes do produto criadas. O ScrumMaster tem coragem para remover impedimentos e realizar as mudanças necessárias na organização, de forma a ajudar a equipe a se tornar mais produtiva;
- **franqueza:** a franqueza ou transparéncia é necessária para que se possa realizar a inspeção e adaptação. Assim, o time busca melhorar sua forma de trabalhar a partir do *feedback* frequente de seus membros, o que cria visibilidade sobre os problemas e estimula a busca por soluções. O time também busca validar os resultados de seu trabalho a partir do *feedback* frequente sobre o que produzem, o que cria visibilidade sobre as mudanças necessárias. Ele mantém visível o progresso de seu trabalho, para poder tomar medidas de correção necessárias. Ao mesmo tempo, sente-se seguro para estabelecer e informar suas estimativas de quanto trabalho acredita que pode ser realizado. O time também interage com o Product Owner para esclarecer dúvidas e com o

ScrumMaster para ter seus impedimentos removidos o mais cedo possível, sempre que necessário;

- **compromisso:** o time determina como seu trabalho será realizado, monitora seu progresso e realiza as correções de rumo que achar necessárias. Assim, é responsável e responsabilizado pelos seus resultados. Esse controle maior que ele tem sobre seu trabalho o torna mais comprometido com o sucesso do projeto. Em cada ciclo de desenvolvimento, o time se compromete com realizar uma meta de negócio acordada com o Product Owner, se comprometendo com dar o seu melhor para fazê-lo. Por sua vez, o Product Owner se compromete a estabelecer as prioridades de trabalho de forma a satisfazer as necessidades de negócios do projeto, interagindo com quem for necessário para fazê-lo;
- **respeito:** os membros do time e o Product Owner trabalham juntos, compartilhando responsabilidades e, assim, ajudam uns aos outros em seu trabalho. Todos que trabalham no projeto respeitam as opiniões uns dos outros, ouvem e buscam entender os diferentes pontos de vista. O Product Owner respeita as decisões técnicas dos membros do time, que respeitam suas decisões de negócios. Por fim, são respeitados o bem-estar e o direito das pessoas que trabalham no projeto a terem uma vida privada.

2.3 SCRUM É UM FRAMEWORK SIMPLES E LEVE

O Scrum é um *framework* para gestão de projeto e não uma metodologia ou um conjunto de processos. Mas o que isso quer dizer exatamente?

Um *framework* (ou arcabouço) é uma estrutura básica que pretende servir de suporte e guia para a construção, a partir da expansão dessa própria estrutura, de algo com uso prático. Enquanto *framework*, o Scrum não define práticas específicas e detalhadas a serem seguidas. Scrum não prescreve, por exemplo, como o time deve desenvolver o produto, como se deve facilitar seu trabalho ou remover impedimentos, nem como devem ser levantadas as necessidades de negócios ou se deve lidar com os clientes do projeto.

Scrum entende que as práticas necessárias para o sucesso do projeto são

muito específicas para cada contexto, não podendo assim ser prescritas. Não existe aquela solução ou conjunto de soluções que funcionam para todas as situações. Ao contrário, as pessoas gerando o produto, a partir dos papéis, eventos, artefatos e regras do Scrum, avaliam, adquirem e desenvolvem um conjunto de práticas que melhor lhe servirão, o que é constantemente reavaliado. Esse é um trabalho contínuo de descoberta, inspeção e adaptação.

Por ser um *framework*, Scrum pode funcionar bem quando combinado com ou complementado por diferentes métodos e práticas consagrados pelo mercado, que podem ser experimentados e adaptados pelo time para seu contexto específico.

Extreme Programming (XP), por exemplo, é uma metodologia. Embora seja considerado leve, XP prescreve um conjunto mais completo de práticas que devem ser seguidas para se obter sucesso no projeto. Ao usar Scrum, muitos times de desenvolvimento pegam emprestadas diversas práticas do XP, como a programação em par, a integração contínua, o desenvolvimento dirigido por testes etc.

Scrum é simples e leve. Com ele, não se gera ou se aplica nada que não será efetivamente útil e utilizado. Prescrevendo apenas três papéis (Product Owner, Time de Desenvolvimento e ScrumMaster), seis eventos (Sprint e reuniões de Sprint Planning, Daily Scrum, Sprint Review e Sprint Retrospective) e três artefatos (Product Backlog, Sprint Backlog e o Incremento do Produto), Scrum pode ser facilmente explicado e compreendido (neste livro, adicionei alguns eventos e artefatos opcionais).

Mas, ao contrário do que pode parecer, Scrum não é fácil de ser usado. Ao longo deste livro, apresentarei algumas alternativas possíveis de técnicas já consagradas, utilizadas com sucesso por inúmeros Times de Scrum, e que talvez possam ajudar o seu time no uso do *framework*.

Analogia do esqueleto

Podemos imaginar que um *framework* se opõe a uma metodologia assim como o esqueleto está para o corpo humano.

O esqueleto — ou seja, os ossos — fornece uma estrutura, mas não se movimenta sozinho. É necessário que sejam agregados a ele músculos, tendões e outros tecidos para poder se movimentar. O corpo humano, por outro lado,

já possui tudo o que necessita para se movimentar.

O *framework* fornece uma estrutura de trabalho, mas não é útil se aplicado sozinho. Ele necessita de ser complementado por outras técnicas e práticas para que seja possível a realização do trabalho por quem o está usando. Uma metodologia já prescreve grande parte do que é necessário para que sua aplicação seja bem sucedida.

Scrum é um *framework* e não uma metodologia porque, como afirmei anteriormente, acredita-se que as técnicas e práticas necessárias para a realização do trabalho dependem do contexto em que esse trabalho é realizado e não podem ser prescritas, embora existam boas práticas reconhecidas.

2.4 SCRUM SE APLICA A PRODUTOS COMPLEXOS EM AMBIENTES COMPLEXOS

2.4.1 Sistemas Adaptativos Complexos

Scrum foi criado para auxiliar no desenvolvimento de produtos complexos imersos em ambientes complexos. O projeto de desenvolvimento de *software* com Scrum pode ser entendido como um Sistema Adaptativo Complexo. Podemos ver a seguir duas das definições mais comuns para esse tipo de sistema:

Sistemas Adaptativos Complexos são sistemas que envolvem um grande número de componentes, frequentemente chamados de agentes, que se adaptam ou aprendem à medida que interagem (HOLLAND, 2006).

Sistemas Adaptativos Complexos consistem em uma série de componentes ou agentes, que interagem uns com os outros de acordo com conjuntos de regras que requerem que eles examinem e respondam aos comportamentos uns dos outros a fim de melhorar seu comportamento e, portanto, o comportamento do sistema de que fazem parte (STACEY, 1996).

O comportamento de Sistemas Adaptativos Complexos é emergente, não podendo ser previsto a partir de uma análise de causa e efeito, e nem a partir da inspeção individual de seus agentes e de suas propriedades. Ou seja,

entender e analisar os detalhes de cada um de seus agentes não nos permite inferir o que vai acontecer com o sistema como um todo.

No entanto, esses agentes interagem entre si e, a partir dessas interações, emergem padrões que aumentam a previsibilidade do comportamento do sistema no curto prazo. Dessa forma, esses sistemas operam em um estado de instabilidade ordenada. Os agentes de Sistemas Adaptativos Complexos são auto-organizados: em frequentes ciclos de *feedback*, o próprio comportamento emergente do sistema o realimenta e provoca a adaptação de seus agentes e do sistema como um todo. Estímulos externos também provocam o mesmo. Essa adaptação causa a evolução do sistema e a emergência de novos padrões de comportamento (JAIN; MESO, 2004).

Diversos líderes do Movimento Ágil basearam suas ideias de como gerenciar o desenvolvimento de *software* em teorias que estudam os Sistemas Adaptativos Complexos (HIGHSMITH, 2002). Scrum foi criado nesse contexto.

Como veremos em detalhes neste livro, os ciclos frequentes de *feedback*, que guiam a construção do produto e possibilitam a melhoria contínua dos processos de trabalho (inspeção e adaptação), as equipes de trabalho auto-organizadas e a alta interação necessária entre as partes envolvidas no desenvolvimento do produto são, entre outros, elementos do Scrum que podem ter seus paralelos facilmente identificados nas teorias de Sistemas Adaptativos Complexos.

2.4.2 Onde utilizar Scrum?

Cynefin é um modelo criado por Dave Snowden que auxilia na tomada de decisões (SNOWDEN; BOONE, 2007). O modelo categoriza os contextos em que sistemas operam, de forma que se possa selecionar a abordagem mais apropriada.

Nesse modelo, os contextos ordenados são aqueles em que se pode estabelecer a relação entre causa e efeito, e são divididos em óbvio e complicado. Os contextos não ordenados, em que não há relação imediata aparente entre causa e efeito, são o complexo e o caótico. No contexto de desordem, é difícil de reconhecer com o que se está lidando (veja a figura 2.2).

Ao apresentar o modelo Cynefin, meu objetivo é que você perceba que Scrum é adequado para projetos de desenvolvimento de *software*, que pertencem em sua maioria aos contextos complexos. Vou analisar em seguida o significado de cada domínio de contextos definido no modelo, com relação a projetos.

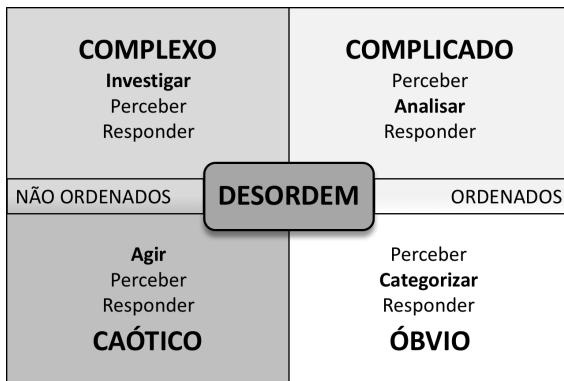


Fig. 2.2: O modelo Cynefin

Contextos óbvios

Esse é o domínio das melhores práticas, onde há estabilidade e as relações entre causa e efeito são evidentes para todos, previsíveis e repetíveis. Assim, para uma dada situação, deve-se avaliar seus fatos (perceber) para definir de que tipo de situação se trata (categorizar) e, então, aplicar a melhor prática conhecida (responder). Esses contextos eram chamados de “simples” no modelo Cynefin original, posteriormente modificado pelo seu autor.

Para um projeto que predominantemente opera nesse domínio, pode-se usar conhecimento do passado para realizar um planejamento completo e detalhado do projeto e, assim, monitorar o seu andamento de acordo com esse plano.

A troca da tubulação de um banheiro com a planta desse banheiro em mãos, por exemplo, é um projeto que pertence a esse contexto, uma vez que existe uma significativa base de conhecimento sobre o assunto. Realizar para

um cliente a digitação dos dados dos produtos de uma loja na internet também caracteriza um contexto óbvio.

Scrum não é geralmente muito útil em projetos desse domínio, pois pode trazer uma sobrecarga de regras, eventos e artefatos que apenas são úteis para ambientes de mudança e imprevisibilidade.

Contextos complicados

Esse é o domínio das boas práticas, onde existe a relação entre causa e efeito, mas nem todos podem vê-la. Diferentemente dos contextos óbvios, pode haver várias respostas viáveis para um mesmo problema e especialistas podem ser necessários para investigar e escolher a que melhor se aplica ao problema em questão. Assim, para uma dada situação, deve-se avaliar seus fatos (perceber) para definir qual das possíveis soluções é a mais adequada (analisar) e, então, aplicá-la (responder).

Para um projeto que predominantemente opera nesse domínio, pode-se realizar análises suficientes dos problemas e escolher, entre as possíveis soluções, aquelas que se julguem mais adequadas. Pode-se realizar um planejamento detalhado do projeto e monitorar o seu andamento de acordo com esse plano.

Definir a planta de uma nova casa, por exemplo, é um projeto complicado, uma vez que escolhas entre diferentes soluções deverão ser adotadas para cada uma das questões do projeto.

Da mesma forma que para contextos óbvios, Scrum pode trazer uma sobrecarga de regras, eventos e artefatos úteis apenas onde há mudança e imprevisibilidade.

Contextos complexos

Esse é o domínio da emergência, onde a relação entre causa e efeito sómente se torna evidente em retrospecto, e não é reproduzível. Em outras palavras, para qualquer ação tomada, não é possível prever o que vai acontecer como consequência. Assim, em vez de se impor um caminho, uma abordagem empírica e adaptativa deve ser utilizada, a partir da qual as práticas vão emergir.

A tolerância a falhas é um aspecto essencial da aprendizagem empírica. Assim, para uma dada situação, deve-se investigar por meio de experimentos (investigar) para entender seus impactos (perceber) e, então, promover o que funciona e abandonar o que não funciona (responder).

Abordagens tradicionais de gestão com *comando-e-controle* não são eficientes nesses contextos de mudança e imprevisibilidade. Nos contextos complexos, aplicam-se a experimentação, a inovação, a criatividade e a emergência de novas formas de se trabalhar. É aqui que operam os Sistemas Adaptativos Complexos, descritos anteriormente, em que se enquadra a grande maioria dos projetos de desenvolvimento de *software*, assim como tantos outros tipos de projetos.

Scrum foi criado para ser utilizado em contextos complexos.

Contextos caóticos

Esse é o domínio das práticas novas, onde as relações entre causa e efeito são impossíveis de serem determinadas, pois mudam constantemente. Não há padrões, apenas turbulência, de modo que qualquer prática utilizada será completamente nova. Quando possível, deve-se agir rapidamente de modo a reestabelecer a ordem (agir), para se entender onde existe a estabilidade (perceber) e, então, trabalhar para trazer a situação do caos para a complexidade (responder), que é um contexto mais gerenciável.

As ações necessárias em uma situação de crise, como o resgate de reféns, por exemplo, caracterizam um projeto caótico. Projetos com foco em pesquisa, no qual o que se descobre em um momento pode mudar todo o trabalho a ser realizado em seguida, são exemplos de projetos que operam em contextos caóticos. Um projeto de desenvolvimento de *software* sobre um sistema legado mal estruturado e repleto de problemas também pode tomar conformações de um contexto caótico.

Scrum não funciona bem para projetos que operam em contextos caóticos. Para que ele seja útil, é necessário estabelecer-se ao menos um horizonte de curto prazo no qual os objetivos a serem realizados sejam bem definidos e estáveis. Nos contextos caóticos, no entanto, os objetivos podem mudar a qualquer momento.

Desordem

Nesse domínio, múltiplas perspectivas se apresentam e não é possível determinar qual dos quatro contextos é o predominante. Assim, nada se sabe sobre a relação entre causa e efeito, e não se pode determinar qual a melhor forma de se trabalhar.

Na desordem, as pessoas competem de forma destrutiva por soluções que estejam em sua zona de conforto, ou seja, em algum dos outros quatro contextos. A saída para essa situação é quebrar o problema em partes e definir o contexto de cada uma delas — óbvio, complicado, complexo ou caótico — para então agir de acordo.

Conclusão

De forma geral, métodos tradicionais de gerência de projetos tratam todos os tipos de projeto como se fossem ordenados, ou seja, pertencendo ao domínio de contextos óbvios ou complicados. Essa prerrogativa, quando falsa, pode levar a grandes e custosos fracassos.

A grande maioria de projetos de desenvolvimento de *software*, assim como tantos outros tipos de projetos, pertence ao domínio de contextos complexos. Gerenciar projetos que requerem uma abordagem empírica e adaptativa como se fossem ordenados simplesmente não é uma boa solução. Scrum foi criado exatamente para lidar com projetos complexos.

2.5 SCRUM É EMBASADO NO EMPIRISMO

É típico se adotar a abordagem de modelagem definida (teórica) quando os mecanismos básicos pelos quais um processo opera são razoavelmente bem compreendidos. Quando o processo é complicado demais para a abordagem definida, a abordagem empírica é a escolha apropriada (OGUNNAIKE; RAY, 1994).

Scrum entende que, por ser complexo e possuir um alto grau de incerteza, o desenvolvimento de produtos como *software* deve ser realizado de forma empírica.

Na abordagem empírica, aprende-se tanto sobre os meios de produção utilizados quanto sobre o produto que está sendo gerado ao se trabalhar em

ciclos sucessivos de *feedback*, experimentando-se, verificando-se o que é válido e o que não é e adaptando-se de acordo.

Ao contrário do que pregam métodos tradicionais, não é viável definir as necessidades de negócios com grande nível de detalhes no início do projeto. Embora se trabalhe sempre para se realizar uma visão de produto suficientemente bem definida, as necessidades de negócios e seus detalhes vão mudar ao longo do projeto, seguindo as mudanças no ambiente e a melhor compreensão das necessidades dos clientes.

Scrum busca maximizar a habilidade do time que desenvolve o produto em responder rapidamente a essas mudanças. Ao mesmo tempo, o time buscará tornar-se cada vez mais produtivo, avaliando quais práticas serão mantidas e quais serão modificadas ou acrescentadas.

Dessa forma, podemos afirmar que o Scrum baseia-se na visibilidade ou transparência, inspeção e adaptação frequentes tanto do produto quanto dos processos de desenvolvimento para que, pela realização de melhorias incrementais contínuas em ambos, busque-se sempre gerar o maior retorno possível para os clientes.

Transparência, inspeção e adaptação são considerados os pilares do Scrum.

2.6 SCRUM É ITERATIVO E INCREMENTAL

Scrum é iterativo. O produto é desenvolvido em ciclos ou iterações, que se repetem em sua forma sucessivamente. Em cada um desses ciclos, é gerado um incremento no produto, que modifica e se soma ao que já se tem pronto do produto até o momento. Cada ciclo é como se fosse um pequeno projeto autocontido, com todas as atividades necessárias para se gerar uma parte do produto estável e funcionando.

São funcionalidades prontas de ponta a ponta, de forma que é possível entregar o produto para uso, caso se decida fazê-lo. Nesse trabalho em iterações, mais do que gerar apenas funcionalidades prontas, o time visa a frequentemente entregar um produto que represente valor para seus usuários.

No Scrum, os ciclos são chamados de Sprints. Eles têm tamanho fixo e acontecem um depois do outro, sem intervalos entre eles, ou seja, tudo em um projeto que utiliza Scrum acontece dentro dos Sprints. Cada Sprint tem um formato bem estabelecido, que é composto por:

- reunião de Sprint Planning, onde se realiza o planejamento do próprio Sprint;
- trabalho de desenvolvimento do produto, que inclui tudo que é necessário para se entregar uma parte do produto pronta ao final do Sprint;
- reunião de Daily Scrum, contida no dia a dia de desenvolvimento, que é uma reunião diária para gerar visibilidade e planejar o próximo dia de desenvolvimento;
- reunião de Sprint Review, onde se obtém *feedback* dos clientes e demais partes interessadas sobre os resultados do trabalho realizado no Sprint;
- reunião de Sprint Retrospective, onde o time avalia o que foi bem e o que precisa ser melhorado na sua forma de trabalho;
- quaisquer reuniões ou sessões programadas adicionais que se fizerem necessárias, como por exemplo a reunião de Release Planning, as sessões de Refinamento do Product Backlog etc.

A figura 2.3 representa a iteração do Scrum, indicando a sucessão de elementos presentes em todos os ciclos (ou Sprints) e a sucessão de ciclos. A reunião de Daily Scrum e quaisquer reuniões adicionais acontecem no dia a dia de desenvolvimento.

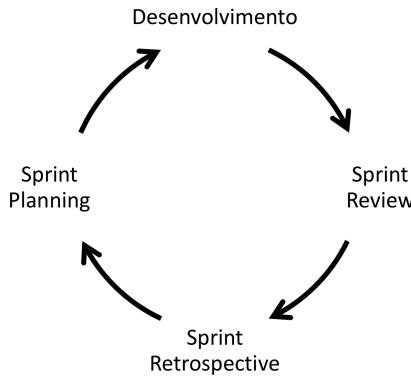


Fig. 2.3: Sucessão de iterações no Scrum

Analogia do bolo

Vamos supor que dois times de confeiteiros vão assar um bolo, cada um utilizando um método diferente. Ambos os bolos possuem várias camadas (veja a figura 2.4).

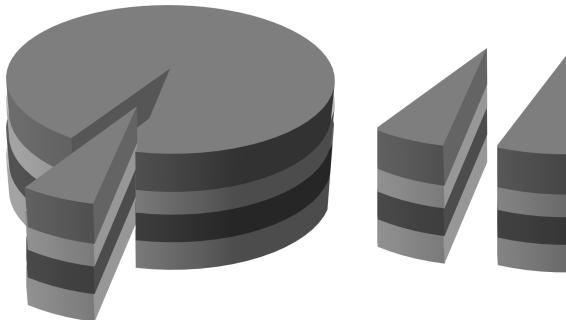


Fig. 2.4: A analogia do bolo: o produto com Scrum é gerado fatia a fatia

O primeiro time usará um método tradicional, em que se produz o bolo camada por camada. Assim, seus membros misturam os ingredientes da primeira camada do bolo e, então, a assam. Em seguida, misturam os ingredi-

entes da segunda camada, assam-na e colocam-na sobre a primeira. Fazem o mesmo com a terceira, quarta e quinta camadas, colocando cada uma sobre a anterior. Por fim, cobrem o bolo de glacê, que é a parte visível para quem comerá o bolo. Somente aí uma fatia do bolo pode ser cortada para ser consumida.

O segundo time tem um objetivo ousado. Deve-se permitir que o bolo seja consumido o mais rápido possível e não apenas no final. Para atingir esse objetivo, o time utiliza uma forma de trabalho bem diferente: eles produzem o bolo fatia por fatia, em fatias bem finas. Cada uma dessas fatias possui apenas um pedacinho de cada uma das camadas, sendo cada pedacinho apenas o suficiente para sustentar o pedacinho acima, tudo coberto pelo glacê. Assim, o bolo já pode ser consumido desde a sua primeira fatia.

É claro que não espero que algum confeiteiro utilize o método do segundo time para produzir um bolo. Mas o desenvolvimento do produto com Scrum se assemelha muito com essa forma de se trabalhar.

Diferentemente do trabalho com métodos tradicionais, o time de desenvolvimento não produz camadas completas do produto, uma após a outra. O time de Scrum gera o produto em incrementos. Cada um desses incrementos é uma fatia fina do produto pronta para ser utilizada, com apenas o necessário de cada uma das camadas do *software* para funcionar.

Essas camadas podem incluir, por exemplo, desde o banco de dados até a interface, que é a parte visível para o usuário — o glacê. Essa entrega de fatias funcionando do produto gera desde cedo retorno ao investimento realizado pelos clientes e possibilita o seu *feedback*, reduzindo os riscos do projeto.

Analogia da iteração

Para determinados tipos de equações matemáticas, não existem fórmulas para se encontrarem suas soluções. Elas somente podem ser resolvidas por meio de iterações.

O uso de iterações para resolver equações matemáticas funciona mais ou menos da seguinte forma: partindo-se de um valor inicial conhecido ou arbitrado, deve-se iterar sobre o problema, utilizando a solução de uma iteração como entrada da próxima iteração. Assim, o resultado é progressivamente melhorado. Esse valor inicial é geralmente um palpite, ou seja, imagina-se ou

sabe-se que a solução é próxima a ele, mas não o suficiente para ele ser considerado satisfatório. O que se quer é chegar em uma solução boa o suficiente.

Em outras palavras, aplica-se o valor inicial dado à equação, obtendo-se desse modo um resultado mais próximo à solução da equação. Aplica-se esse resultado obtido à própria equação, obtendo um novo resultado mais próximo ainda à solução da equação. Aplica-se, assim, sucessivamente cada resultado seguinte à equação, utilizando a saída de uma iteração como a entrada da seguinte. Dessa forma, melhora-se a solução até que ela possa ser considerada boa o suficiente.

Pode-se, por exemplo, estabelecer para a solução uma precisão desejada de cinco casas decimais. Ao verificar que uma nova iteração modificaria apenas a partir da sexta casa, considera-se que a solução é boa o suficiente. Assim, o ponto de parada é aquele em que não compense o retorno sobre o investimento de se executar mais uma iteração.

Para o desenvolvimento de *software*, a ideia é a de que partimos, desde a primeira iteração, de uma solução simples (ou uma hipótese de solução) para a parte mais importante do problema. Utilizamos o *feedback* obtido sobre essa solução para iterarmos sobre ela, melhorando, corrigindo e expandindo a solução. E seguimos iterando. A condição de parada é a mesma que para a equação: não vale a pena realizar próxima iteração se o retorno sobre o investimento de fazê-lo não compensar.

CAPÍTULO 3

Como é o Scrum?

3.1 INTRODUÇÃO

Neste capítulo, juntos passearemos por todas as etapas de um projeto com Scrum, de ponta a ponta. No entanto, não pretendo ser prescritivo nem mostrar todas as possíveis alternativas. A ideia aqui é dar um exemplo genérico do que, em geral, acontece em projetos que utilizam o *framework*, passando por todos os seus papéis, artefatos e eventos, adicionados de mais alguns elementos que podem ser utilizados.

3.2 INÍCIO DO PROJETO

O projeto para o desenvolvimento de um produto é contratado ou solicitado para atender a uma oportunidade, necessidade ou objetivo de negócios, seja de um cliente específico, de um grupo de clientes ou do mercado. A **Visão do**

Produto é uma forma de traduzir esse objetivo a ser realizado.

O **Product Owner** é o responsável por definir o produto, trabalho que ele realiza pouco a pouco, ao longo de todo o projeto, priorizando o que será desenvolvido em cada ciclo de desenvolvimento, de forma a maximizar o retorno sobre o investimento realizado pelos clientes do projeto.

Para guiar esse trabalho, o Product Owner deve criar, comunicar e manter a Visão do Produto relativamente constante ao longo do projeto. O Product Owner é único. Ele trabalha com os clientes do projeto e com quaisquer outras partes interessadas que possam contribuir para o entendimento e definição do produto. O grupo de partes interessadas do projeto também inclui os próprios **usuários do produto**, que receberão ao longo do desenvolvimento partes prontas do produto para serem utilizadas.

A partir da Visão do Produto, o Product Owner pode opcionalmente criar um plano de como se espera que o produto evolua ao longo do tempo, chamado de **Roadmap do Produto**. Este é geralmente expresso por uma linha do tempo com as metas esperadas para cada momento no futuro, que aqui chamo de **Metas de Roadmap**.

Antes do desenvolvimento do produto começar, o projeto geralmente passa por uma fase inicial na qual definições e preparativos básicos são realizados. Essa fase possui uma duração que depende do que e de quanto é necessário se definir e preparar. É chamada por alguns de “Pré-Jogo” ou, por outros, equivocadamente de “Sprint Zero”, já que o trabalho realizado nessa fase de forma alguma caracteriza um Sprint, como será visto mais adiante.

Embora seu uso seja comum em projetos Ágeis, tanto a fase de “Pré-Jogo” quanto os artefatos Visão do Produto e Roadmap do Produto não são parte do *framework* Scrum.

Ainda nessa fase inicial, decide-se quem serão as pessoas a trabalhar no projeto, que formarão o **Time de Scrum**: além do Product Owner, são elas os membros do Time de Desenvolvimento e o ScrumMaster. Esse processo de escolha varia de organização para organização. Entre diferentes possibilidades, pode haver um departamento responsável pela seleção de pessoal, pode-se partir de um Product Owner, de um ScrumMaster ou de um ou mais membros iniciais do Time de Desenvolvimento que escolhem na organização o resto do time, ou pode-se designar para o projeto um time que já trabalha

junto, por exemplo.

O **Time de Desenvolvimento** realiza todo o trabalho de desenvolvimento do produto, de ponta a ponta. Ele é multidisciplinar, o que significa que possui, em seus membros, todo o conhecimento necessário para realizar esse trabalho. O Time de Desenvolvimento é também auto-organizado, ou seja, ele próprio define, dentro do contexto da organização e do *framework* Scrum, como vai realizar o trabalho e gerenciar seu progresso em direção a metas acordadas com o Product Owner.

O **ScrumMaster** é o facilitador do trabalho do Time de Scrum em seu dia a dia, ensinando o Time de Desenvolvimento a assumir a responsabilidade sobre seu trabalho, a crescer em autonomia e, assim, a se auto-organizar. Ele está presente e age como um facilitador em todas as reuniões do Scrum, além de facilitar as interações entre o Time de Desenvolvimento e o Product Owner.

Para realizar esse trabalho, ele deve ser tão neutro quanto possível e possuir *soft skills*, ou seja, competências comportamentais e pessoais. O ScrumMaster também ensina Scrum ao Time de Scrum e, quando necessário, a quaisquer pessoas na organização que podem influenciar o trabalho no projeto. Ele é o responsável por garantir que os **impedimentos** que o Time de Scrum encontre em seu trabalho sejam removidos, atuando sempre que necessário como um agente de mudança na organização. Esses impedimentos geram para o Time de Desenvolvimento o risco de não se realizarem os objetivos.

Ainda nessa fase de Pré-Jogo, pode ser necessário especificar uma arquitetura básica de produto, de forma a reduzir os riscos de decisões tardias que invalidariam o que já foi produzido. Pode ser também necessário criar ou adaptar uma infraestrutura que servirá de suporte para o desenvolvimento do produto. A ideia nessa fase é gerar apenas o mínimo necessário e suficiente para reduzir os riscos, mas sem engessar o desenvolvimento do produto, nem gerar grandes desperdícios.

Antes do início do desenvolvimento, o Product Owner inicia, a partir da Visão do Produto, a criação de uma lista ordenada, incompleta e dinâmica de itens que representam o que ele acredita que será produzido ao longo do projeto. Essa lista é chamada de **Product Backlog** e deve ser atualizada ao

longo de todo o projeto.

Os itens do alto do Product Backlog são os mais importantes naquele momento e, por essa razão, possuem mais detalhes, suficientes para possibilitar que sejam desenvolvidos primeiro. Os itens mais abaixo têm gradativamente menos detalhes.

O Product Backlog inicial pode ser longo, contendo desde itens pequenos e bem detalhados até itens grandes e vagos. Mas ele também pode ser curto e conter apenas uma quantidade de itens necessária para se iniciar o desenvolvimento. O Product Backlog evoluirá ao longo de todo o projeto e será frequentemente modificado com a adição, subtração, detalhamento, divisão, reordenamento e modificação de seus itens.

User Story é a forma preferida de times Ágeis para representar cada um dos itens do Product Backlog que tratam de necessidades de usuários do produto, descrevendo-os sob o ponto de vista desses usuários e de uma forma concisa, simples e leve. A User Story não é, no entanto, parte do *framework* Scrum. Cabe ao Time de Scrum definir a forma que melhor lhe serve para representar os itens do Product Backlog.

O Time de Scrum então está pronto para iniciar o primeiro dos vários ciclos do projeto, nos quais o trabalho de desenvolvimento do produto será realizado de forma incremental. Esses ciclos são chamados de **Sprints**. O projeto com Scrum acontece Sprint após Sprint. Assim, ao terminar um Sprint, inicia-se imediatamente o seguinte. As Sprints têm sempre a mesma duração, escolhida entre uma e quatro semanas.

Os eventos do Scrum — o próprio Sprint e as reuniões de Sprint Planning, de Daily Scrum, de Sprint Review e de Sprint Retrospective — possuem uma duração máxima ou fixa definida, chamada de **timebox**. Os timeboxes são importantes, pois evitam o desperdício, limitando o tempo em que um objetivo deve ser realizado, além de ajudar a criar um ritmo ou uma regularidade no trabalho realizado.

Pode-se ver o ciclo completo do Scrum na figura 3.1 (SCHWABER; BEE-DLE, 2002; SCHWABER, 2004).

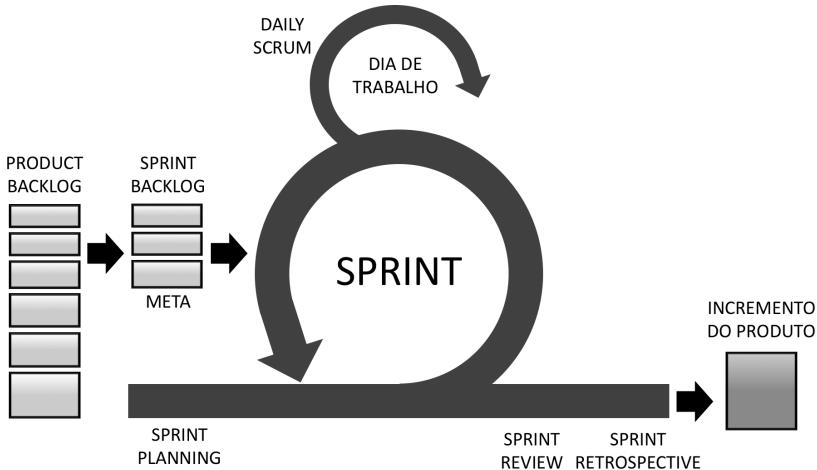


Fig. 3.1: O ciclo do Scrum

3.3 PLANEJAMENTO DO SPRINT

O Sprint se inicia com a reunião de **Sprint Planning**, na qual se planeja o trabalho a ser realizado no próprio Sprint. Nessa reunião, Time de Desenvolvimento e Product Owner colaborar para decidir, a partir dos itens do alto do Product Backlog, o que será desenvolvido no Sprint. Ou seja, facilitados pelo ScrumMaster, eles selecionam um conjunto de itens do alto do Product Backlog que julgam ser capazes de desenvolver na duração do Sprint — o que é apenas uma previsão — e estabelecem um objetivo a ser realizado a partir do desenvolvimento desses itens, chamado de **Meta do Sprint**. O Time de Desenvolvimento, então, se compromete a realizar essa Meta do Sprint.

É importante que os itens do alto do Product Backlog estejam preparados para que a reunião de Sprint Planning seja eficiente e produtiva. Itens que chegam à reunião sem detalhes suficientes ou com uma granularidade muito grossa, por exemplo, podem colocar todo o Sprint em risco.

Para garantir que esses itens estejam preparados para serem discutidos na reunião de Sprint Planning, pode-se criar e usar uma **Definição de Preparado**. São critérios claros que definem qual é o estado necessário de um item

de Product Backlog para ser considerado preparado para ser colocado em desenvolvimento. Caso um Time de Scrum opte pelo uso de uma Definição de Preparado, o Time de Desenvolvimento passa a ter a prerrogativa de recusar, na a reunião de Sprint Planning, um item que não esteja preparado de acordo com essa definição.

Além do detalhamento necessário, por exemplo, outros critérios da Definição de Preparado podem incluir o item ser pequeno o suficiente e possuir os **Critérios de Aceitação** definidos, entre outros.

No primeiro Sprint, o Time de Desenvolvimento ainda não tem dados para gerar métricas sobre sua capacidade de trabalho em um Sprint. Ele pode estimar os itens do Product Backlog individualmente para possibilitar a futura obtenção dessas métricas úteis para o planejamento. **Story Point** é uma unidade muito utilizada por times Ágeis em suas estimativas, mas não é obrigatória.

Na realidade, nenhum tipo de estimativa é obrigatório em um projeto com Scrum. Outra forma de se obter essas métricas é dividir e deixar sempre os itens do alto do Product Backlog bem pequenos, com pequena variação de tamanho entre eles, para então contar o número de itens entregues nos últimos Sprints e calcular a média.

Seja como for, já a partir do segundo Sprint, o Time de Desenvolvimento pode utilizar como referência de quantidade de trabalho para o Sprint em planejamento a média da quantidade de trabalho entregue nos últimos Sprints. Essa quantidade de trabalho que se espera realizar por Sprint é chamada de **Velocidade do Time de Desenvolvimento**.

Além de, juntamente com o Product Owner, selecionar os itens e definir uma meta, o Time de Desenvolvimento também cria um plano de como o que foi selecionado será desenvolvido. Esse plano é geralmente expresso por um conjunto de tarefas, para cada item, a serem realizadas durante o Sprint. Essas tarefas são, de forma geral, os passos técnicos que o Time de Desenvolvimento identifica como necessários para desenvolver o item.

O conjunto de itens selecionados e seu respectivo plano é chamado de **Sprint Backlog**, e é geralmente representado na forma de um **Quadro de Tarefas** (veja a figura 3.2). O Sprint Backlog existe apenas dentro do seu Sprint respectivo.



Fig. 3.2: Exemplo de Quadro de Tarefas representando o Sprint Backlog

3.4 DESENVOLVIMENTO

Após a reunião de Sprint Planning, inicia-se o trabalho de desenvolvimento propriamente dito dos itens do Sprint Backlog. Os membros do Time de Desenvolvimento definiram como o trabalho será realizado durante a reunião de Sprint Planning. Agora, são os responsáveis por se auto-organizar para realizar esse trabalho e por monitorar seu progresso em direção à Meta do Sprint.

O desenvolvimento inicia-se a partir do primeiro item do Sprint Backlog, que é o item mais importante para se realizar a Meta do Sprint. Os diferentes membros do Time de Desenvolvimento realizam as diferentes tarefas necessárias, comunicando-se e colaborando durante todo o processo. Aos poucos, os membros do Time de Desenvolvimento vão migrando para os itens seguintes do Sprint Backlog, sempre preocupando-se em completá-los de acordo com a sua prioridade.

Em cada dia do trabalho de desenvolvimento, os membros do Time de Desenvolvimento se encontram por no máximo quinze minutos, preferencialmente no mesmo horário e no mesmo local, para a reunião de **Daily Scrum**.

O objetivo da reunião é garantir entre eles a visibilidade de seu trabalho e planejar, informalmente, o próximo dia de trabalho. O ScrumMaster pode, se necessário, facilitar essa reunião, mas sua presença não é obrigatória.

A reunião de Daily Scrum é um ótimo momento para se atualizar, quando adotado, o **Gráfico de Sprint Burndown**, uma ferramenta que o Time de Desenvolvimento pode utilizar para monitorar seu progresso no Sprint. Esse gráfico possui no eixo X os dias de trabalho no Sprint, e no eixo Y a quantidade de trabalho restante, que pode ser medida pela quantidade de tarefas restantes ou por algum tipo de estimativa realizada sobre as tarefas, por exemplo. O Time de Desenvolvimento marca no gráfico a quantidade de trabalho restante no dia corrente (veja a figura 3.3).

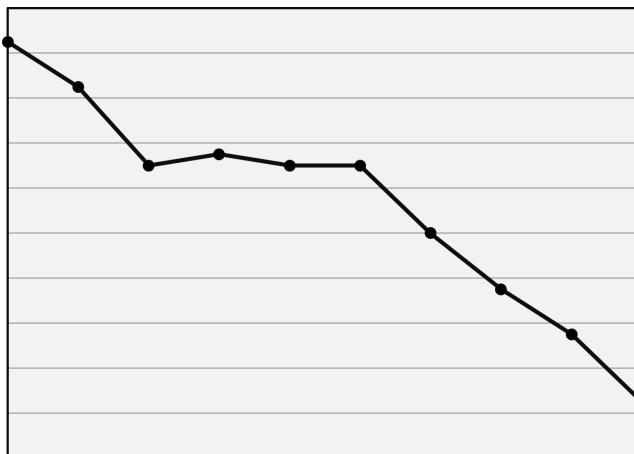


Fig. 3.3: Exemplo de Gráfico de Burndown

Em seu dia a dia de trabalho, é comum os membros do Time de Desenvolvimento se depararem com desafios. Sempre que encontram algo que os impede de realizar seu trabalho, eles avisam ao ScrumMaster, que inicia imediatamente o trabalho de remoção desse impedimento.

Um Product Owner acessível e presente ao longo do Sprint permite que o Time de Desenvolvimento tire dúvidas de negócios que naturalmente surgem sobre os itens em desenvolvimento. Ao mesmo tempo, o Product Owner

mantém contato frequente com os clientes do projeto e demais partes interessadas para entender suas necessidades ou objetivos de negócios mais urgentes, colher seu *feedback* e, assim, atualizar o Product Backlog de acordo, em um trabalho contínuo de **Refinamento do Product Backlog**.

O Product Owner também busca interagir com o Time de Desenvolvimento para chegar ao Sprint com os itens necessários preparados. Para realizar esse Refinamento do Product Backlog, eles podem se encontrar apenas na reunião de Sprint Planning. Entretanto, é preferível que essa preparação ocorra já durante o Sprint anterior, em sessões agendadas ou em um trabalho contínuo. Em qualquer caso, pode-se usar uma Definição de Preparado como critério que define o que significa um item estar preparado.

3.5 ENCERRAMENTO DO SPRINT

Ao final do Sprint, o Time de Desenvolvimento deverá ter gerado, a partir dos itens do Sprint Backlog, um **Incremento do Produto** entregável — ou “potencialmente entregável”, como descreve Schwaber & Sutherland (2013) — que representa valor visível para os clientes do projeto. Ser entregável significa que nenhum trabalho adicional é necessário para que esse Incremento do Produto possa ser entregue aos clientes do projeto. Entregar ou não o Incremento após o final do Sprint, no entanto, é uma decisão do Product Owner.

Caso, no transcorrer do Sprint, a Meta do Sprint tenha perdido o seu sentido, o Product Owner pode decidir pelo cancelamento do Sprint, antecipando o seu encerramento. Essa é uma situação de exceção e raramente deve ocorrer.

No último dia do Sprint, Product Owner e Time de Desenvolvimento se encontram para duas reuniões consecutivas de inspeção e adaptação, ambas facilitadas pelo ScrumMaster. Na primeira reunião, de **Sprint Review**, faz-se a inspeção e adaptação do produto, enquanto que na reunião de **Sprint Retrospective**, faz-se a inspeção e adaptação da forma de trabalhar do Time de Scrum.

O Product Owner convida para a reunião de Sprint Review os clientes do projeto e demais pessoas relevantes que possam prover *feedback* sobre o Incremento do Produto que foi produzido durante o Sprint. Nessa reunião,

o Time de Desenvolvimento e o Product Owner apresentam e demonstram para os presentes o resultado do seu trabalho no Sprint, para então obter *feedback* deles. É uma importante oportunidade para antecipar parte do *feedback* que só virá com o uso após a entrega e, assim, reduzir os riscos do projeto.

A **Definição de Pronto** é um acordo entre Product Owner e Time de Desenvolvimento sobre o que é necessário para que qualquer item ou o Incremento do Produto como um todo seja considerado pronto e, assim, passe a fazer parte do produto em desenvolvimento. A Definição de Pronto é a mesma para todos os itens do Product Backlog que se traduzam em funcionalidades a serem desenvolvidas. Idealmente, ela é utilizada para garantir que o Incremento do Produto gerado no Sprint seja entregável.

O Time de Desenvolvimento usa a Definição de Pronto para guiar o desenvolvimento dos itens definidos para o Sprint Backlog na reunião de Sprint Planning. Apenas os itens prontos de acordo com a Definição de Pronto devem ser apresentados na reunião de Sprint Review.

O *feedback* obtido dos clientes e demais pessoas relevantes presentes na reunião de Sprint Review é utilizado pelo Product Owner como matéria-prima para alterações no Product Backlog. Ou seja, para modificar o produto que está sendo gerado de forma a melhor atender às necessidades dos clientes.

Após a reunião de Sprint Review, Time de Desenvolvimento e Product Owner realizam a reunião de Sprint Retrospective, facilitados pelo Scrum-Master. Nela, ambos identificam o que foi bem no Sprint corrente, e que por essa razão pode ser mantido no próximo Sprint, e o que pode melhorar, buscando formas práticas e traçando planos de ação para realizar essas melhorias. O objetivo desse processo de melhoria contínua é tornar o Time de Scrum cada vez mais efetivo.

Uma vez terminada a reunião de Sprint Retrospective, está encerrado o Sprint. Um novo Sprint se inicia imediatamente após o término da anterior (respeitando, claro, os fins de semana e feriados).

3.6 ENTREGAS

Uma vez que o Incremento do Produto ou a soma de Incrementos do Produto represente valor suficiente e já puder ser utilizado, é importante que chegue a

usuários o mais rápido possível.

Por meio de **Releases** frequentes, o Time de Scrum pode obter *feedback* dos usuários do produto, reduzindo os riscos e produzindo o produto certo. O Time de Scrum pode também dar um senso de progresso do projeto aos seus clientes e demais partes interessadas, e pode prover retorno ao investimento realizado pelos clientes do projeto.

A estratégia de Releases a ser usada no projeto é definida pelo Product Owner. Ele decide quando ou com que frequência as Releases serão realizadas e quem vai recebê-las — por exemplo, algum grupo de usuários que pode experimentar o produto e prover *feedback* ou, sempre que possível, o usuário final. Assim, o Product Owner decide se a Release será realizada cada vez que um item estiver pronto (em entrega contínua), ao final de cada Sprint, sempre que ele julgar adequado ou após alguns Sprints, visando a um objetivo ou meta de negócios definida.

Pode-se realizar, nesse último caso, uma reunião de **Release Planning** para cada Release. Essa reunião é realizada em algum momento antes do início dos trabalhos para a Release correspondente. Portanto, deve acontecer durante o último Sprint da Release anterior (ou durante o Pré-Jogo, para a primeira Release).

Na reunião de Release Planning, Product Owner e Time de Desenvolvimento estabelecem o **Plano de Release**, que contém uma data aproximada para a Release, um objetivo ou meta a ser realizada, chamada de **Meta da Release**, e um conjunto de itens selecionados a partir do alto do Product Backlog. Essa reunião não substitui as reuniões de Sprint Planning que serão realizadas para cada Sprint da Release.

O progresso em direção à data da Release e, portanto, ao cumprimento da Meta da Release é inspecionado Sprint a Sprint e a ferramenta mais utilizada com esse propósito é o **Gráfico de Release Burndown**. Esse gráfico possui no eixo X os Sprints da Release, e no eixo Y a quantidade de trabalho restante, que pode ser medida pela quantidade de itens restantes entre os previstos para a Release ou por estimativas realizadas sobre os itens, por exemplo.

O Product Owner marca no gráfico a quantidade de trabalho restante ao final de cada Sprint, momento em que o Plano da Release é revisto. Outras ferramentas, como o **Gráfico de Release Burnup**, podem ser usadas em seu

lugar.

A reunião de Release Planning, a Meta de Release, o Gráfico de Release Burndown e o Gráfico de Release Burnup não são parte do Scrum.

3.7 FINAL DO PROJETO

Em geral, um projeto possui uma duração determinada. Na realidade, um projeto pode ser encerrado por diversas razões, dependendo do contexto: o tempo de contrato expirou, valor suficiente já foi entregue para os clientes do projeto, o orçamento do projeto acabou ou o contrato foi cancelado, entre outras.

No momento em que o projeto passa a ser considerado terminado, espera-se que as necessidades ou objetivos de negócios de maior importância já tenham sido entregues. O projeto terá sido bem-sucedido se, por meio de suas entregas frequentes, a Visão do Produto for considerada satisfatoriamente realizada.

Após um término normal de projeto, é geralmente necessária sua manutenção, em uma fase que podemos chamar de “pós-jogo”. Embora não haja fórmula definida para esse trabalho, sabe-se que o uso de Scrum, por suas características, não costuma ser a melhor opção para esse trabalho. Métodos orientados a pedidos ou *tickets* como o **Kanban**, criado por David Anderson, são geralmente mais apropriados para essa fase do projeto (veja em [28.4 Lean Kanban](#)).

CAPÍTULO 4

Como contratar projetos com Scrum?

4.1 INTRODUÇÃO

O objetivo deste capítulo é apresentar ideias e conceitos que podem embasar contratos para desenvolvimento de *software* com Scrum. Não pretendo, no entanto, entrar nos detalhes do que deve estar presente nesses contratos, descrever os passos para a sua elaboração, nem entrar no *juridiquês* que deve ser utilizado na sua elaboração.

4.2 CONTRATOS TRADICIONAIS

O que vai ser entregue? Quanto vai custar? Quando será entregue? Essas são perguntas comuns que contratos tradicionais buscam responder, firma-

dos entre clientes e empresas desenvolvedoras de *software*.

O “Triângulo de Ferro” (ou “Triângulo da Gestão de Projetos”), formado pela tríade de restrições “escopo”, “orçamento” (custo para o cliente) e “prazo”, é ainda hoje usado como referência para a formação de muitos desses contratos (veja a figura 4.1). Tendo esse modelo como referência, estabelece-se que o produto deve ser entregue com o exato escopo acordado com o cliente, com o orçamento estabelecido e dentro do prazo acertado.

Alguns autores ainda adicionam ao triângulo a “qualidade” como a quarta restrição. Isso significa para o contrato que o produto entregue deve estar dentro dos padrões de qualidade acordados.

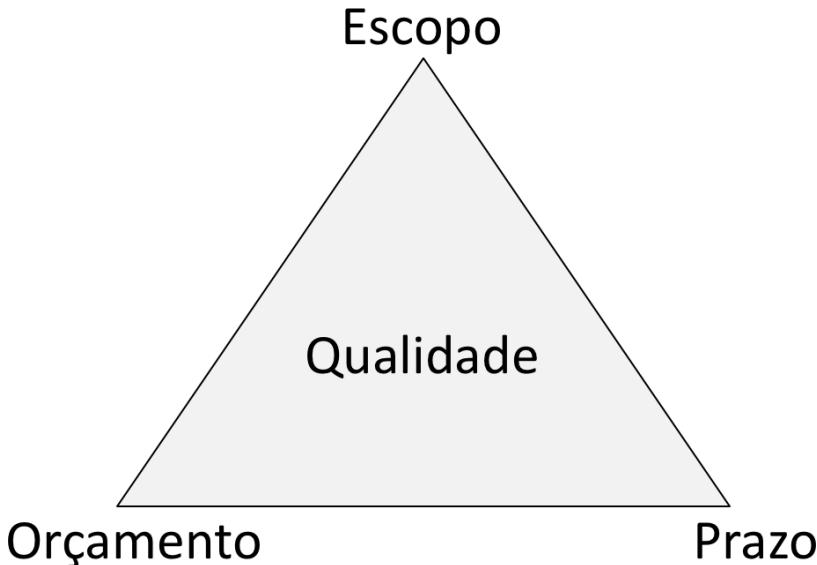


Fig. 4.1: Triângulo da Gestão de Projetos

A necessidade de se estabelecer um contrato formal é mais comum quando há um cliente solicitando, a uma empresa de desenvolvimento de *software*, a geração de um produto. No entanto, os conceitos apresentados neste capítulo também são úteis para quando tratamos de projetos internos, nos quais o time de desenvolvimento (ou seu departamento) estabelece um con-

trato informal com clientes internos. Esses conceitos também são úteis para quando tratamos da geração de produtos para o mercado, quando há um contrato informal com pessoas ou departamentos internos que fazem o papel de clientes.

4.2.1 Preço Fixo tradicional

A forma mais comum de contratação em projetos de *software* é por Preço Fixo. Na forma tradicional desse tipo de contrato, as três restrições são fixadas. O escopo do projeto é esmiuçado em detalhes. Estimativas são realizadas sobre esses detalhes para, em seguida, se estabelecer o prazo do projeto. A esse prazo, em geral, é ainda aplicado um fator de correção, também conhecido informalmente como “gordura”.

A partir do escopo e prazo, o orçamento é fixado. Ou seja, parte-se do escopo para se determinar o prazo e o orçamento para o cliente. Alternativamente, pode-se também partir de um orçamento ou prazo máximo (em geral, estabelecidos pelo cliente) e ajustar as duas outras restrições de acordo. Por exemplo, o cliente exige uma data máxima para a entrega final e, então, detalha-se o escopo para, a partir daí, negociar-se sua redução, juntamente com o orçamento do projeto.

Ao oferecer exatamente o que será entregue, quando será entregue e quanto lhe custará, essa forma de contrato traz uma sensação de segurança para o cliente do projeto. Não é à toa que é de longe o modelo preferido entre os clientes que não conhecem os benefícios do Scrum.

Mas essa segurança é inteiramente falsa: o modelo, para projetos de *software*, simplesmente não funciona. Como mencionei em outro capítulo, o Standish Group divulgou em 2015 um estudo mostrando que apenas 11% dos projetos que fixam escopo, orçamento e prazo terminam com o orçamento e no prazo previstos e com o cliente satisfeito (The Standish Group, 2015).

Assim, ao detalhar e fixar o escopo, o modelo de Preço Fixo tradicional para projetos de *software* garante apenas que há uma enorme chance do contrato ser violado, o que observamos ser o caso mais comum nesse mercado.

4.2.2 Tempo e Material tradicional

Tempo e Material é outra forma conhecida de se contratarem projetos de desenvolvimento de *software*. A forma tradicional desse contrato em geral prevê o detalhamento do escopo no início sem, no entanto, que se fixem o orçamento e o prazo final.

Basicamente, o cliente é cobrado por hora ou dia de trabalho das pessoas contratadas para o projeto até que o escopo estabelecido seja esgotado, sem qualquer garantia de quando isso vai ocorrer. O valor cobrado por dia ou por hora é normalmente estabelecido de acordo com o papel exercido por essas pessoas e por seu nível de senioridade.

Esse modelo é mais usado quando as pessoas contratadas para o projeto estão alocadas no cliente, ou seja, trabalhando em suas dependências durante o projeto. Assim, o cliente acredita exercer um maior controle sobre o trabalho que está sendo realizado.

Ao se utilizar o contrato por Tempo e Material, tanto o custo total do projeto (orçamento) quanto a data de seu término ficam imprevisíveis. O cliente até pode tentar se proteger, ao estabelecer um limite a partir de estimativas, tanto para o orçamento quanto para o prazo final. Mesmo assim, ao se definirem os detalhes do produto de antemão, grande parte dos riscos recaí sobre o cliente, pois ele, ainda assim, pode não receber um produto funcional ao se chegar nesse limite.

Caso opte-se por permitir que o cliente modifique o escopo acordado no contrato estabelecido, em geral o que quer que o cliente peça será feito e o projeto pode facilmente sair do controle na perseguição do produto final. A empresa desenvolvedora de *software*, em princípio, não verá problemas, pois continuará sendo paga, enquanto que novamente os riscos de um projeto des-governado recairão sobre o cliente.

4.3 PRINCÍPIOS BÁSICOS PARA CONTRATOS COM SCRUM

4.3.1 Premissas básicas de contratos com Scrum

No capítulo *O que é Scrum?*, descrevemos *software* como um problema que não pode ser definido antecipadamente (veja *Discussão: a analogia errada*, em [2.2 Scrum é Ágil](#)). Ou seja, somente se pode definir o problema a partir da construção de soluções, em um processo empírico.

Também afirmei que projetos de desenvolvimento de *software* são Sistemas Adaptativos Complexos, que operam em contextos complexos e, portanto, seu comportamento é emergente e não previsível ao longo prazo (veja [2.4 Scrum se aplica a produtos complexos em ambientes complexos](#)).

Como vimos no capítulo *Por que Scrum?*, resultados de estudos apontam que, ao fixar-se o escopo no princípio do projeto, apenas algo entre 20% e 57% das funcionalidades entregues são utilizadas, em diferentes graus. Soma-se a isso o fato de que apenas entre 1% e 10% das linhas de código produzidas são entregues, dependendo da complexidade e tamanho do projeto (veja *Produzir apenas o que os usuários vão utilizar*, em [“1.7 Redução de desperdício”](#)).

Assim, uma das premissas básicas com que trabalhamos para a criação de contratos em projetos de desenvolvimento de *software* é que não é possível prever e detalhar o escopo do projeto em seu princípio. Entendemos que, ao se tentar fazê-lo, um enorme desperdício será gerado, e o prazo e o orçamento estabelecidos muito provavelmente não serão cumpridos.

Essa é uma realidade inerente ao desenvolvimento de *software*, independentemente da metodologia utilizada para o projeto. Dessa forma, podemos concluir que ambos os modelos Preço Fixo tradicional e Tempo e Material tradicional não são modelos adequados para projetos de desenvolvimento de *software*.

No mesmo capítulo *O que é Scrum?*, comparei o formato do projeto de desenvolvimento de *software* com Scrum ao uso de iterações para se resolverem equações matemáticas (veja *Analogia da iteração*, em [2.6 Scrum é iterativo e incremental](#)). Assim, seguimos iterando sobre a solução, melhorando-a cada vez mais a partir de *feedback*, até que ela esteja suficientemente boa, ou seja, não valha a pena executar mais uma iteração.

A premissa básica que essa analogia nos traz, importante para contratos em projetos de desenvolvimento de *software*, é que, uma vez que se prioriza o que se produz, seguir desenvolvendo um produto não tem mais sentido a partir do momento em que o retorno a ser obtido já não compensa o investimento de fazê-lo.

4.3.2 O Triângulo Ágil

Uma vez colocadas as premissas básicas de contratos com Scrum, podemos entender por que o Triângulo de Ferro, que mostrei anteriormente, é inadequado como base para o estabelecimento desses contratos para projetos de desenvolvimento de *software*. Entendemos que não é possível se estabelecer o prazo e o orçamento do projeto, a partir do detalhamento de um todo a ser desenvolvido. Esse todo não existe de antemão e somente pode ser detalhado em retrospecto, após pronto.

Em outras palavras, não há como se calcular o prazo e orçamento do projeto a partir do escopo. Podemos, no máximo, fixar esse prazo, que será o tempo máximo dentro do qual iteraremos sobre o problema a ser resolvido.

Assim, para o estabelecimento de um contrato, em vez de trabalharmos com prazo e orçamento como consequências de um escopo fixo predefinido (como no Triângulo de Ferro), podemos trabalhar com um triângulo invertido. Ou seja, são fixados o prazo e orçamento, e os detalhes do escopo são definidos apenas ao longo do projeto (veja a figura 4.2).

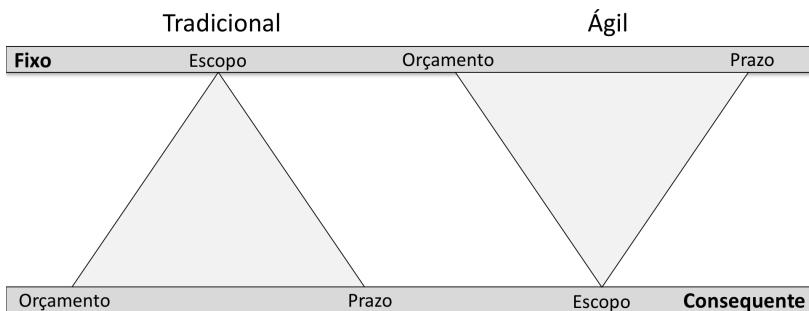


Fig. 4.2: Triângulo de Ferro versus Triângulo Ágil

4.3.3 Escopo não detalhado

Escopo não detalhado não significa ausência de escopo. É importante que se esclareça no contrato, mesmo que em alto nível, qual o problema a ser resolvido (ou a necessidade a ser suprida, ou a oportunidade a ser aproveitada) com o uso do produto, que justifica a própria contratação do desenvolvimento.

A definição clara e direta desse problema é essencial para guiar o desenvolvimento do produto. A forma mais comum de representá-lo é através de uma Visão do Produto. No contrato, essa visão pode ser explicitamente desmembrada, caso se mostre necessário. Esse desmembramento é feito em objetivos um pouco mais granulares a serem distribuídos em um *roadmap* da expectativa da sua evolução, ainda em alto nível, no decorrer do projeto.

Veremos mais adiante neste livro formas sucintas de se definir a Visão do Produto e consequentes objetivos ou metas (veja em [24 Metas: definindo os objetivos](#)).

4.3.4 Ponto de parada

Ao fixar-se o orçamento e o prazo, e flexibilizar-se o escopo, o cliente pode se sentir inseguro sobre o que vai receber como resultado do projeto. Ele pode acreditar que existe o risco de não alcançar a expectativa de ter os seus problemas resolvidos e buscar sentir-se mais seguro exigindo um contrato tradicional, com o escopo detalhado.

Já entendemos neste capítulo que esse tipo de contrato traz apenas uma ilusão de segurança e não deve ser uma opção em projetos de desenvolvimento de *software*. Isso porque aumenta em muito os riscos do cliente, embora ele não necessariamente perceba isso.

Na realidade, o contrato com Scrum traz muito mais visibilidade e maior segurança para o cliente do projeto. Ele vê os seus resultados concretos ao final de cada ciclo de desenvolvimento, na reunião de Sprint Review. A partir de seu *feedback*, o produto é ajustado e direções são definidas. Incrementos do produto funcionando são entregues com frequência e, na mão de seus usuários, geram retorno sobre o investimento realizado pelo cliente, permitindo um *feedback* ainda mais concreto.

Aliada ao *feedback* frequente, a priorização do que é produzido mitiga tanto os riscos de necessidades importantes do cliente ficarem de fora do projeto quanto os riscos de se produzirem partes ou camadas do produto que apenas potencialmente significam valor, que se concretiza apenas após um longo período de desenvolvimento. O trabalho priorizado garante que as necessidades do cliente sejam atendidas desde cedo, desde as mais importantes, com soluções de ponta a ponta.

Veja na figura 4.3 um gráfico que mostra o incremento de valor de negócio no tempo, ou seja, realizado em cada entrega do projeto. As marcações verticais indicam entregas de Incrementos do Produto.

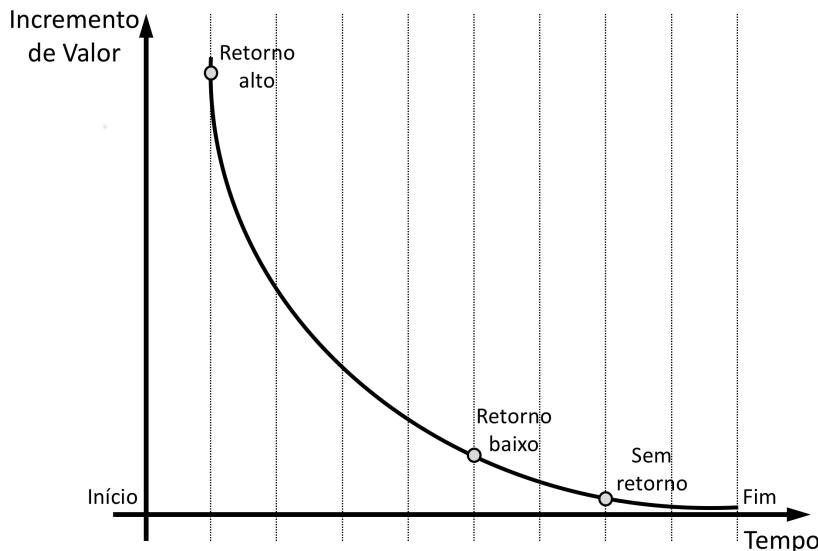


Fig. 4.3: Incremento de valor versus Tempo

De forma simplificada, podemos afirmar que o retorno sobre o investimento envolve o valor de negócio obtido contra o investimento realizado, sendo aqui este último proporcional ao tempo de trabalho até uma entrega. Como se pode observar no gráfico, o projeto com Scrum parte de uma primeira entrega com alto valor de negócio e um consequente alto retorno sobre

o investimento. Isso pois resolve-se em pouco tempo, possivelmente após algumas poucas iterações, a parte mais importante do problema mais importante da Visão do Produto.

A entrega seguinte gera menor valor de negócio, pois consiste de soluções para as próximas questões mais importantes, que incluem novas funcionalidades e melhorias para o que já foi entregue. O trabalho segue de forma iterativa, com melhorias incrementais em como a Visão do Produto é realizada, mas somando cada vez menos valor ao que já foi entregue. Forma-se, assim, uma curva assintótica, cujos pontos se aproximam cada vez mais do eixo, mas sem nunca tocá-lo. Dependendo do projeto, essa curva pode ser mais ou menos acentuada que a da figura 4.3.

Repare no gráfico como o retorno da entrega é altíssimo no princípio do projeto, mas rapidamente decai entrega após entrega, até poder ser considerado baixo. Em algum ponto mais baixo da curva, podemos avaliar que esse retorno se anula e que, por essa razão, já não compensa mais para o cliente seguir com o desenvolvimento do produto. A partir desse ponto, adicionar novas funcionalidades ao produto já não se paga, e o projeto poderia simplesmente ser interrompido.

4.4 FORMATOS DE CONTRATOS COM SCRUM

Nesta seção, busco listar os formatos de contrato mais comuns para projetos de desenvolvimento de *software* com Scrum. Todos partem das mesmas premissas listadas em seções anteriores: orçamento e prazo fixos, escopo flexível.

4.4.1 Preço fixo, prazo fixo, escopo flexível

Fixa-se um prazo e um orçamento para o projeto, mas o escopo não é detalhado em seu princípio e evolui em seu decorrer. Em um contrato desse tipo, o escopo pode ser definido em alto nível através de uma Visão do Produto e, talvez, de um *roadmap*. O prazo e o orçamento podem ser fixados a partir de:

- o valor que o cliente está disposto a pagar pelo projeto (portanto, o orçamento), que compra um certo tempo de trabalho da empresa desenvolvedora do *software*;

- uma data final dada pelo cliente, definida a partir de uma necessidade específica do negócio;
- o tempo que a empresa desenvolvedora do *software*, a partir de seu conhecimento e experiência, acredita ser suficiente para iterar sobre o problema até resolvê-lo satisfatoriamente.

O cliente, por exemplo, necessita do produto pronto até certa data antes do início de um evento relacionado. Ou, em outro exemplo, o cliente possui reservado um determinado orçamento para o projeto, e a empresa desenvolvedora calcula que esse valor é suficiente para X meses de trabalho de uma equipe de Y pessoas. Ou a empresa desenvolvedora simplesmente define, a partir de sua experiência, que em X meses será capaz de resolver o problema satisfatoriamente.

Nos dois últimos casos, o orçamento para o cliente pode ser calculado a partir do prazo que foi fixado, alocando-se um determinado número de pessoas para o projeto. Pode-se também definir o orçamento a partir do valor que se espera gerar para o cliente. Entretanto, esse é um cálculo difícil de se realizar na maioria das vezes.

4.4.2 Preço fixo, escopo flexível, prazo fixo com opção de parada

Esse formato de contrato carrega as mesmas características que o anterior, *preço fixo, prazo fixo, escopo flexível*. Nesse formato, no entanto, o cliente pode escolher interromper o projeto em qualquer momento que desejar.

De forma geral, o cliente escolherá cessar o desenvolvimento se:

- ele estiver insatisfeito com o trabalho realizado até o momento e não quiser gastar mais com um projeto no qual não mais acredita;
- ele estiver suficientemente satisfeito com o produto já entregue até o momento como um resultado final, e acreditar que o retorno sobre o investimento a ser realizado em possíveis futuras entregas não compensará mais.

Enquanto a primeira razão traz uma maior segurança para o cliente, a segunda é tratada no projeto como o objetivo de sucesso. O gráfico de valor

incremental *versus* tempo da figura 4.3 nos mostra que, caso a priorização tenha sido bem feita, o cliente e as demais partes interessadas tenham sido envolvidos no projeto para oferecer *feedback* e entregas frequentes tenham sido realizadas, existe uma grande possibilidade de que se atinja um ponto no qual o retorno sobre o investimento não justifique mais prosseguirem com o projeto, antes do prazo final estabelecido. Assim, naturalmente, o cliente poderia escolher interromper o projeto, satisfeito com o que já está em suas mãos.

4.4.3 Preço fixo, escopo flexível, prazo fixo com opção de parada taxada

Esse formato de contrato adiciona ao anterior, “preço fixo, escopo flexível, prazo fixo com opção de parada”, uma taxa a ser paga pelo cliente caso deseje interromper o projeto antes do prazo final acordado. Essa taxa pode corresponder a um valor fixo, mas é geralmente um percentual do valor restante do contrato ou, talvez, um valor calculado como o suficiente para cobrir o custo da equipe de desenvolvimento (mas não o lucro) pelo tempo restante em que trabalharia no projeto.

Dessa forma, se estabelece uma relação ganha-ganha: o cliente economiza com relação ao total que pretendia gastar no projeto e ainda vê a necessidade de seu envolvimento cessando mais cedo; e a organização que desenvolve o *software* recebe por um trabalho que não vai realizar e pode alocar as pessoas envolvidas naquele projeto em outros.

Esse formato de contrato é uma evolução do já bastante conhecido e usado *Money for nothing and your change for free*, de Jeff Sutherland, um dos criadores do Scrum (SUTHERLAND, 2008). A grande diferença é que o modelo do Jeff estabelece, em adição à opção de parada taxada, um escopo fixo inicial do qual partes podem ser negociadas e trocadas. Acredito que, para os dias de hoje, não passa de uma solução paliativa.

4.4.4 Preço fixo, escopo flexível, prazo fixo com opção de troca de contexto

Adiciona-se ao formato “preço fixo, escopo flexível, prazo fixo com opção de parada” a possibilidade de usar o tempo e orçamento restantes após a decisão de interrupção do projeto para outros fins. Pode-se, por exemplo, utilizar-se o tempo restante para realizar um pequeno projeto ou iniciar um novo projeto a ser continuado em um novo contrato.

4.4.5 Incremental com pontos de verificação

O projeto é executado buscando-se entregar partes prontas do produto o mais frequentemente possível, a partir das mais prioritárias. Para esse formato de contrato, são estabelecidos pontos frequentes de verificação em que o cliente decide se prossegue com o projeto ou não. Ele pode tomar essa decisão ao final de cada Sprint, por exemplo, ou após cada entrega, que pode ser realizada após alguns Sprints.

A diferença fundamental do modelo “incremental com pontos de parada” para o “preço fixo, escopo flexível, prazo fixo com opção de parada” é que, no primeiro, não se estabelece um orçamento nem prazo final para o projeto. No modelo incremental, o produto é desenvolvido com Scrum em incrementos e, após verificar ou receber um ou mais incrementos, o cliente decide se vale a pena continuar com o projeto, seguindo as mesmas razões listadas para o outro modelo.

Esse modelo mantém o prazo fixo, estabelecido como a duração de um Sprint ou a uma data da entrega, e o orçamento é em geral fixado a partir desse prazo.

CAPÍTULO 5

De onde veio o Scrum?

5.1 INTRODUÇÃO

O primeiro Time de Scrum foi criado por Jeff Sutherland em 1993. Jeff trabalhava na Easel Corporation, uma empresa de *software* de Massachussets (Estados Unidos), e aplicou Scrum no projeto mais crítico da organização. Foram obtidos resultados excepcionais com essa nova forma de se trabalhar.

Em 1995, Jeff apresentou o primeiro Time de Scrum a Ken Schwaber, que era CEO da empresa Advanced Development Methods e vinha trabalhando com ideias similares, focadas em abordagens empíricas para o desenvolvimento de *software*. Ken e Jeff trabalharam juntos para criar uma definição formal de Scrum, apresentada por Ken no OOPSLA de 1995, um congresso de Orientação a Objetos.

Em 2001, tanto Jeff Sutherland quanto Ken Schwaber foram signatários do Manifesto Ágil. Como mencionei no capítulo *O que é Scrum?* (veja [2.2 Scrum](#))

é Ágil), esse manifesto deu início ao chamado movimento Ágil, do qual Scrum faz parte. Nesse mesmo ano, Ken Schwaber, em parceria com Mike Beedle, publicou *Agile software development with Scrum*, o primeiro livro a descrever o *framework* Scrum.

5.2 UM NOVO JOGO

Scrum não é uma sigla ou um acrônimo. Assim, não se deve escrever SCRUM, com as letras maiúsculas, ou S.C.R.U.M., com um ponto após cada letra. Scrum é, na verdade, uma das formações em que as equipes se colocam para a reposição de bola em um jogo de rúgbi. Mas como esse nome veio parar em um *framework* Ágil?

Os criadores do Scrum se inspiraram em um artigo dos autores Hirotaka Takeuchi e Ikujiro Nonaka, intitulado *The new new product development game* (ou “O novo jogo no desenvolvimento de novos produtos”). Ele foi publicado em 1986 na renomada revista Harvard Business Review (TAKEUCHI; NONAKA, 1986).

Takeuchi e Nonaka são conhecidos por suas importantes contribuições na área de gestão de conhecimento e aprendizado organizacional. Nesse artigo, eles estudaram empresas do Japão e Estados Unidos, como a 3M, Xerox, Honda, Epson e Hewlett-Packard, que estavam utilizando uma abordagem diferente e com excelentes resultados para equipes de desenvolvimento de novos produtos. Essas equipes possuíam seis características fundamentais:

- instabilidade embutida, ou seja, a alta gerência indica objetivos ou estratégias amplos e desafiadores para criar um elemento de tensão, ao mesmo tempo em que dá à equipe uma grande liberdade para alcançar esses objetivos;
- equipes de projeto auto-organizadas, que possuem:
 - autonomia no seu dia a dia do trabalho, com pouquíssima interferência da alta gerência;
 - autotranscendência, ou seja, uma busca interminável por um limite inalcançável, em que a equipe estabelece objetivos cada vez mais altos durante o processo de desenvolvimento;

- fertilização cruzada, ou seja, a equipe deve possuir uma variedade de especializações, comportamentos e personalidades, de forma que a interação entre seus membros promova a distribuição do conhecimento e o trabalho em equipe, já que cada um entende melhor a posição do outro;
- sobreposição nas fases de desenvolvimento, o que possibilita uma melhor integração entre as diferentes fases e o melhor tratamento de galhos. Embora os membros da equipe possuam horizontes iniciais diferentes, pois seu trabalho começa em fases diferentes do projeto, a equipe acaba por produzir um ritmo comum de trabalho ou um pulso, que funciona como uma força motora;
- múltiplo aprendizado, que permite à equipe responder rapidamente às mudanças de mercado. Por ficarem expostos a diversas fontes de informação, os membros da equipe adquirem um amplo conhecimento e habilidades diversas, gerando uma equipe versátil capaz de resolver uma miríade de problemas rapidamente. Esse aprendizado se dá em duas dimensões: em múltiplos níveis, isto é, o aprendizado individual, o aprendizado do grupo e o aprendizado da organização; e em diversas funções, nas quais especialistas são encorajados a acumular experiências em áreas fora de sua especialidade;
- controle sutil exercido pela alta gerência, que estabelece pontos de verificação para prevenir que a instabilidade, ambiguidade e tensão levem ao caos. A gerência evita o tipo de controle rígido que prejudica a criatividade e espontaneidade. O controle sutil é exercido das seguintes formas:
 - selecionando as pessoas certas para o projeto, enquanto monitoram-se as dinâmicas do grupo e adicionam-se ou removem-se membros quando necessário;
 - criando um ambiente aberto de trabalho;
 - encorajando os engenheiros a interagir com os clientes e fornecedores;

- estabelecendo um sistema de avaliação e recompensa baseado no desempenho do grupo, e não no individual;
 - gerenciando as diferenças de ritmo ao longo do processo de desenvolvimento;
 - tolerando erros, buscando que sejam cometidos o mais cedo possível e que sempre se aprenda com eles;
 - encorajando fornecedores a também se auto-organizarem e envolvendo-os desde cedo no projeto;
- transferência organizacional de aprendizado, ou seja, a transferência do conhecimento acumulado para outras divisões da organização e para outros projetos de desenvolvimento de novos produtos. Essa transferência ocorre ao se designar indivíduos-chave para projetos subsequentes, ou ao se criarem padrões a partir de atividades do projeto. No entanto, a transferência de conhecimento funciona bem quando o ambiente externo é estável. Mudanças no ambiente, no entanto, podem fazer com que essas lições aprendidas sejam inúteis. Nesses cenários, desaprender pode ajudar o time de desenvolvimento a se manter alinhado à realidade de fora de seu ambiente e a realizar melhorias incrementais.

Takeuchi e Nonaka afirmam nesse artigo que a forma dessas equipes de alto desempenho trabalharem lembra muito times de rúgbi, que se movem em direção ao objetivo como um só bloco, passando a bola entre seus membros para trás e para frente — daí o nome Scrum. Essa visão opõe-se à forma tradicional de se trabalhar no desenvolvimento de produtos, em que o projeto progride em fases sequenciais, e as funções são altamente especializadas e segmentadas.

5.3 LEAN E SISTEMA TOYOTA

5.3.1 Introdução

O Sistema Toyota de Produção (STP) foi criado nos anos 1950 pela empresa japonesa Toyota Motor Corporation, com a ambiciosa missão de competir com

as gigantes Ford e General Electric (GE), dos Estados Unidos. Essas empresas dominavam o mercado mundial, produzindo automóveis da forma mais barata possível, utilizando-se da produção em massa.

No cenário do Japão no pós-guerra, com uma economia devastada e um mercado interno muito limitado, a Toyota necessitava de alternativas. A empresa concluiu que deveria produzir pequenos volumes de diferentes modelos, utilizando a mesma linha de montagem (LIKER, 2003; WOMACK et al., 1992).

Os propósitos da Agilidade e do Sistema Toyota de Produção são diferentes, a princípio. Enquanto STP foi originalmente concebido para manufaturas, a Agilidade foi criada no contexto do desenvolvimento de *software*. No entanto, Scrum, assim como todo o movimento Ágil, possui várias de suas ideias enraizadas no STP. Assim, acredito que seja importante entender esses conceitos e compará-los ao Scrum e à Agilidade em geral.

Nos anos 1990, autores americanos escreveram livros e divulgaram, a partir de um ponto de vista norte-americano, o Sistema Toyota de Produção para o mundo. O sistema japonês foi generalizado e rebatizado para *Lean Production* (ou Produção Enxuta), com a justificativa de que nele se usam menores quantidades de tudo em comparação com a produção em massa.

Chamarei, nas seções a seguir, tanto o Sistema Toyota de Produção quanto o *Lean Production* apenas de “Lean” (WOMACK; JONES, 1998; WOMACK et al., 1992), exceto quando pretendo fazer alguma distinção entre os dois.

5.3.2 Os princípios do Lean

Os cinco princípios do Lean são:

- **definir valor** — especificar o que significa valor na perspectiva de seus clientes, e não do produtor. Ou seja, o produto deve atender às necessidades do cliente;
- **identificar a cadeia de valor** — para cada produto (ou família de produtos), identificar todos os passos que são realizados para que se possa oferecer ao cliente o produto final. Os passos que não geram valor para o cliente constituem desperdício e devem ser eliminados;

- **criar fluxo** — os passos que geram valor devem fluir continuamente, sem interrupções nem fronteiras entre departamentos, ou entre a organização e fornecedores;
- **estabelecer a produção “puxada”**
 - os passos devem ser realizados de forma que o produto final seja o que o cliente quer, produzido e entregue quando ele necessita. A produção, assim, é “puxada” pelo cliente, e não “empurrada” pelo produtor;
- **buscar a perfeição** — a organização deve sempre buscar a perfeição, eliminando os desperdícios e melhorando continuamente seus processos.

5.3.3 Lean e Agilidade

Para lhe ajudar a compreender as influências do Lean na Agilidade, traço a seguir um paralelo entre princípios de ambos, além de destacar como Scrum trata deles.

Definir valor

No Lean, o cliente é tratado como parte integral do processo de produção. O valor deve ser definido pela perspectiva desse cliente, de forma a atender às suas necessidades. O produto que ele deseja deve ser produzido e entregue quando ele necessita.

O valor Ágil *colaboração com o cliente mais que negociação de contratos* e o princípio Ágil *pessoas de negócio e desenvolvedores devem trabalhar em conjunto diariamente por todo o projeto* estabelecem que a contribuição dos clientes na produção de valor é essencial para o sucesso do projeto.

No Scrum, o Product Owner é responsável por interagir frequentemente com os clientes e definir, de forma incremental, um produto que lhes signifique valor, ordenando a produção a partir das funcionalidades de maior importância para os clientes. Entregas frequentes são realizadas, *feedback* é obtido dos clientes em cada ciclo de desenvolvimento e em cada entrega, e o produto é ajustado de acordo.

Identificar a cadeia de valor

Os passos na produção que não geram valor para os clientes constituem desperdício — chamados de *muda*, em japonês — e devem ser eliminados. Evitar o desperdício, uma das principais preocupações do Lean, permeia praticamente todos os valores e princípios Ágeis.

Em particular, o princípio Ágil *simplicidade* — *a arte de se maximizar a quantidade de trabalho não feito* — é essencial defende a redução do desperdício pela aplicação da simplicidade, ou seja, a geração, como acessório e como produto do trabalho, de somente o que é realmente necessário e suficiente, e da forma mais simples possível.

Podemos definir a cadeia de valor no Scrum como todas as atividades realizadas para que se possa oferecer aos clientes um Incremento do Produto pronto ao final de um ciclo de desenvolvimento (Sprint), de acordo com a Definição de Pronto. Entre essas atividades, aquelas que não agregam valor ao produto (e que, portanto, são desnecessárias) são desperdício e devem ser eliminadas imediatamente (chamadas de *muda* Tipo 2 no Lean).

No entanto, aquelas que não agregam valor ao produto, mas são necessárias ou inevitáveis, também são desperdício, mas não podem ser eliminadas, ao menos naquele momento (chamadas de *muda* Tipo 1 no Lean). Algumas documentações burocráticas, exigidas pelos clientes ou pela organização, são exemplos desse último tipo de desperdício.

O time trabalha sempre a partir dos itens mais importantes em cada momento. Assim, ao se chegar ao final de um prazo dado — uma data de entrega ou mesmo o final de um Sprint -, os itens mais importantes estarão prontos, restando apenas os menos importantes. A priorização desses itens é continuamente revista e atualizada a partir do *feedback* dos clientes sobre o que foi produzido, garantindo que o desenvolvimento de itens de pouca importância seja postergado ou até mesmo não realizado, o que ajuda a evitar o desperdício.

Nas práticas Ágeis, a busca pela maximização da comunicação dentro da equipe e entre os membros da equipe e os clientes, por meio das entregas frequentes e das reuniões programadas, pretende propiciar inspeções frequentes do processo e do produto. Assim, com o *feedback* obtido por essas inspeções, busca-se permitir a adaptação do processo e do produto com as melhorias,

que dessa forma reduzem o desperdício. Outra questão é a valorização dos indivíduos e de suas interações mais do que processos e ferramentas, pois observa-se que o maior foco nos dois últimos (ou seja, no uso excessivo de ferramentas e de processos complexos) gera desperdícios.

Já o princípio Ágil *software em funcionamento é a medida primária de progresso* é uma resposta a uma das grandes fontes de desperdício em projetos, que é a geração de funcionalidades ou artefatos (como documentos, planos e relatórios) que não criam qualquer valor para os clientes.

Pode-se afirmar também que o princípio Ágil *a atenção contínua à excelência técnica e a um bom projeto aumenta a agilidade* pretende reduzir o desperdício buscando evitar produtos mal projetados e realizados por indivíduos sem o conhecimento e as habilidades necessários.

Criar fluxo e estabelecer a produção “puxada”

Lean define que os passos necessários para a geração de valor devem fluir continuamente, um após o outro, sem paradas, lotes, filas ou fronteiras entre departamentos, de forma que o produto seja produzido e entregue quando o cliente necessita, em uma produção “puxada” — a aplicação do chamado *just-in-time* ou JIT.

Diferentemente do sistema em lote, no JIT cada processo produz somente o que é necessário para o processo seguinte, em um fluxo contínuo. Esse sistema inclui todos os fornecedores no fluxo de produção e elimina a necessidade de estoques.

De forma semelhante, no desenvolvimento Ágil de *software* com Scrum, Incrementos do Produto prontos são gerados por equipes multidisciplinares em iterações curtas, uma após a outra, que objetivam entregas frequentes. Cada Incremento entregue deve suprir as necessidades de negócio mais importantes para os clientes em cada momento, e sua descoberta é um trabalho contínuo de interação do Product Owner com esses clientes.

Kanban (palavra que significa “sinalização” em japonês) é um sistema do Lean usado para auxiliar na criação do fluxo utilizando-se da visibilidade. Os operadores usam sinais visuais no fluxo de produção para determinar que uma ação deva ser tomada, como quando há a necessidade de reposição de uma peça que serve de entrada em um processo ou quando há necessidade

de algum tipo de ajuda.

Sinais visuais comuns nesse sistema são os cartões de sinalização com diferentes cores (e significados), e até mesmo recipientes vazios, indicando que devem ser preenchidos novamente (GROSS; MCINNIS, 2003). Não se deve confundir *kanban* com Kanban (com “K” maiúsculo), um método baseado no Lean criado, em princípio, para a gestão de projetos para desenvolvimento de software (veja em [28.4 Lean Kanban](#)).

O Sprint Backlog do Scrum é frequentemente representado na forma de um Quadro de Tarefas. Embora não seja um *kanban*, esse quadro funciona de forma parecida, criando visibilidade sobre o fluxo de trabalho do Time de Desenvolvimento.

Além do *muda* (não adição de valor), duas importantes categorias de desperdício devem ser igualmente evitadas. Estas não foram observadas pelo *Lean Production*, mas são parte integrante do Sistema Toyota de Produção (LIKER, 2003).

A primeira categoria é o *muri*, definido como uma sobrecarga nas pessoas ou equipamentos além dos limites naturais, de forma que pode resultar em problemas de segurança, de saúde, vida útil reduzida e problemas de qualidade, por exemplo. A outra categoria de desperdício é o *mura*, definido como irregularidades, flutuações ou desbalanços no ritmo de produção. Estes ocorrem porque problemas internos, como: componentes defeituosos ou faltantes, manutenção ou gargalos fazem com que, em alguns momentos, haja mais trabalho do que pessoas ou máquinas podem fazer e, em outros, haja falta de trabalho (HAMPSON, 1999; LIKER, 2003).

Uma interpretação incorreta do Lean o leva a ser utilizado para intensificar o trabalho a um ponto em que o estresse do trabalhador torna-se um sério problema, ameaçando todo o sistema de produção. Dessa forma, Lean chega a ser chamado de “gerência pelo estresse”.

Esse equívoco ocorre porque busca-se realizar melhorias no sistema, principalmente usando-se de um aumento de pressão sobre o sistema produtivo além da sua capacidade, ao reduzir recursos e prazos ou aumentar a quantidade de trabalho, por exemplo. Pretende-se, dessa forma, que o estresse gerado em trabalhadores e máquinas faça com que o sistema se fragmente e, assim, permita a identificação de pontos de melhoria e reestrutura-

ção. Pretende-se também que se force o uso da criatividade, da autorregulação e do trabalho em equipe para resolver os problemas decorrentes desse aumento de pressão, possibilitando a realização do mesmo trabalho com menos recursos.

Esse tipo de abordagem leva essas empresas a praticarem os desperdícios *muri* (sobrecarga nas pessoas ou equipamentos além dos limites naturais) e *mura* (desbalanços no ritmo de produção) mencionados anteriormente e, assim, a enfrentarem seus problemas decorrentes.

Para combater esses desperdícios, o Sistema Toyota de Produção traz o conceito de *heijunka* (ou “nívelamento”, em japonês), fundamental e essencial para o estabelecimento do fluxo contínuo, mas muitas vezes esquecido pelos defensores do *Lean Production*. O *heijunka* é obtido a partir do balanceamento e nívelamento do fluxo de produção, com ritmos e níveis de produção constantes e sustentáveis, evitando-se a formação de gargalos.

Assim, com a prática do *heijunka*, o *mura* é automaticamente eliminado (LIKER, 2003). O *heijunka* somente pode ser realizado dentro das capacidades normais do processo, dos trabalhadores e das máquinas, evitando-se assim o *muri* (HAMPSON, 1999).

Em projetos de desenvolvimento de *software*, é comum a gestão se apoiar no trabalho do time de desenvolvimento em um ritmo irregular e eventualmente insustentável. Isso é refletido principalmente em práticas como as horas extras, o trabalho em fins de semana e a aceleração exagerada do ritmo de produção em momentos críticos do projeto, como antes de uma entrega.

É também comum Scrum ser incorretamente utilizado para tal fim, o que pode ser acentuado por algumas características do *framework*, como a alta visibilidade do trabalho, os ciclos curtos e seu consequente ritmo alto de entregas. Na realidade, a produção realizada em um ritmo constante e sustentável é essencial para o trabalho com Scrum, e é defendida pelo princípio Ágil: *os processos Ágeis promovem o desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.*

Assim, com a prática do Scrum, entende-se que se deve buscar um balançamento do fluxo de trabalho de desenvolvimento similar ao *heijunka*, para evitar os desperdícios *muri* e *mura*. Scrum possui alguns mecanismos que, de

forma simplificada, buscam atingir esse balanceamento. Entre eles, podemos destacar:

- o Sprint, que é um ciclo de desenvolvimento que se repete em formato ao longo de todo projeto e possui sempre a mesma duração. Essa duração fixa e constante ajuda a estabelecer um ritmo regular para o trabalho do Time de Desenvolvimento, uma cadência com a qual se acostuma desde o Time de Scrum até as partes interessadas do projeto, que entendem com que frequência verão os resultados do trabalho e podem oferecer *feedback*;
- o trabalho priorizado, a multidisciplinaridade e a responsabilidade sobre os resultados compartilhada entre os membros do Time de Desenvolvimento. Um Time de Desenvolvimento com Scrum deve possuir entre seus membros todos os conhecimentos e habilidades necessários para a geração do produto. Ao mesmo tempo, apesar desses conhecimentos e habilidades estarem distribuídos de forma distinta entre os diferentes membros do time — podendo haver, por exemplo, especialistas — a responsabilidade sobre cada tarefa é de todos. Ao trabalhar de forma priorizada, um membro do time, em vez de abrir um novo item de trabalho de menor prioridade que possua tarefas dentro de suas áreas de conhecimento, deverá buscar auxiliar outros membros do time no cumprimento de itens de trabalho mais prioritários, mesmo que fora de suas áreas de conhecimento. Dessa forma, distribui-se melhor esse conhecimento entre os membros do time e eliminam-se os gargalos na produção progressivamente.

Lean traz também o conceito de *jidoka*, que significa “automação com um toque humano” em japonês. O *jidoka* estabelece que os próprios trabalhadores devem realizar o controle de qualidade, paralisando imediatamente a produção caso algum defeito seja encontrado para se buscar a causa raiz, resolvê-la e evitar que o problema volte a acontecer. A qualidade, assim, deve estar embutida no próprio processo de produção.

Da mesma forma, as equipes multidisciplinares do Scrum são responsáveis pela qualidade do produto. Isso significa que a habilidade de testar ou

garantir a qualidade deve estar dentro da equipe, não existindo assim equipes externas de qualidade.

Buscar a perfeição

O *kaizen* (“mudar para melhor”, em japonês) é a busca da perfeição por meio de etapas infinitas e frequentes de melhorias contínuas, prática essencial do Lean. Uma das chaves para se viabilizar a realização do *kaizen* é o *hansei*, que significa “reflexão profunda” em japonês.

O *hansei* é uma autorreflexão implacável e obrigatória, utilizada para identificar e reconhecer os erros cometidos ou melhorias que precisam ser feitas, para então se criar um plano de ação que coloque as melhorias em prática. O *hansei* parte da crença de que há sempre algo que pode ser melhorado. Seu objetivo nunca é apontar culpados, mas sim tornar os problemas visíveis para ajudar a equipe a realizar melhorias (LIKER, 2003).

Os CINCO PORQUÊS

Para se descobrir as raízes dos problemas, pode-se fazer uso no Lean de uma ferramenta chamada de *Os Cinco Porquês*. Nessa técnica, uma vez que um problema ocorre, pergunta-se “por quê?” iterativamente cinco vezes, sempre questionando a resposta anterior.

Assim, inicialmente pergunta-se o porquê do problema e obtém-se uma causa imediata. Essa causa é questionada perguntando-se novamente “por quê?”, para o que se obtém outra causa, que por sua vez é questionada novamente e, assim, sucessivamente até que “por quê?” tenha sido perguntado cinco vezes, quando então se pretende ter chegado à raiz do problema. A partir daí, propõem-se ações de melhoria (LIKER, 2003; OSONO et al., 2008).

Essa técnica pode ser usada por times Ágeis, já que descobrir a causa raiz dos problemas, para então resolvê-los, é uma prática essencial da Agilidade, na qual se busca evitar o desperdício e realizar as melhorias contínuas.

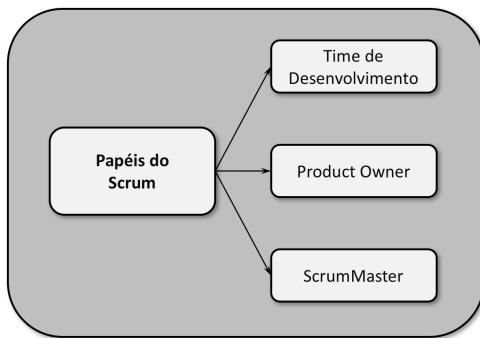
Na Agilidade, a melhoria contínua é expressa no princípio Ágil *em in-*

tervalos de tempo regulares, a equipe reflete sobre como se tornar mais efetiva, para aprimorar e ajustar seu comportamento de acordo. Um dos importantes mecanismos de melhoria contínua no Scrum são as reuniões de Sprint Retrospective, que é basicamente a aplicação do *hansei*.

Parte II

Papéis: o Time de Scrum

Apenas três papéis são definidos pelo Scrum: o Time de Desenvolvimento, o Product Owner e o ScrumMaster. As pessoas que desempenham esses papéis são igualmente responsáveis e responsabilizadas pelos resultados do trabalho e, assim, se comprometem com o projeto.



Os membros do Time de Desenvolvimento, o Product Owner e o ScrumMaster formam o **Time de Scrum**. Enquanto membros de um mesmo time,

eles trabalham juntos, de forma colaborativa, para alcançarem seus resultados.

O QUE É UM TIME?

Um time ou uma equipe é um grupo de pessoas que trabalham juntas e colaboram em busca de um objetivo comum.

Portanto, o Time de Scrum, ou seja, Time de Desenvolvimento, Product Owner e ScrumMaster, devem colaborar lado a lado, em seu dia a dia de trabalho, em busca do objetivo comum de atingirem o sucesso do projeto, em cada passo para chegarem lá.

Ao longo do livro, chamarei de “clientes” as pessoas, grupos ou organizações que solicitam o projeto, ou apenas o patrocinam e recebem o retorno ao investimento, uma vez que o que é produzido lhes é entregue. Chamarei de “usuários” aqueles que, de fato, recebem e usam o produto. Clientes podem também ser usuários.

As demais partes interessadas no projeto contribuem de alguma forma para o projeto, ou simplesmente são informados do seu progresso. Além dos próprios usuários do produto, exemplos de partes interessadas incluem patrocinadores do projeto, executivos e gerentes da organização.

CAPÍTULO 6

Time de Desenvolvimento

6.1 QUEM É O TIME DE DESENVOLVIMENTO?

O Time de Desenvolvimento é um grupo multidisciplinar de pessoas, responsável por realizar o trabalho de desenvolvimento do produto de ponta a ponta. A partir das prioridades definidas pelo Product Owner, o Time de Desenvolvimento gera, em cada Sprint, um Incremento do Produto pronto (veja [13 Incremento do Produto](#)), de acordo com a Definição de Pronto (veja [12 Definição de Pronto](#)), e que significa valor visível para os clientes do projeto.

O Time de Desenvolvimento gerencia o seu trabalho de desenvolvimento do produto, balizado pelo *framework* Scrum. É ele que, mesmo que sujeito aos limites dados pelo contexto do cliente e organizacional, determina tecnicamente como o produto será desenvolvido, planeja esse trabalho e acompanha seu progresso. Para tal, tem propriedade e autoridade sobre suas decisões e, ao mesmo tempo, é responsável e responsabilizado por seus resultados.

Para realizar esse trabalho, o Time de Desenvolvimento:

- planeja seu trabalho, definindo com o Product Owner o que será realizado no decorrer do Sprint, para então detalhar, de forma autônoma, como esse trabalho será realizado;
- realiza as tarefas de desenvolvimento do produto para realizar a Meta do Sprint, garantindo a qualidade do que é produzido, além de acompanhar seu progresso no Sprint em direção a essa Meta;
- colabora com o Product Owner durante o Sprint, sempre que necessário, para ter dúvidas esclarecidas ou solicitar decisões quanto ao produto, e para refinar e aprimorar o Product Backlog, preparando-o para o próximo Sprint;
- identifica e informa ao ScrumMaster sobre impedimentos que obstruam seu trabalho e previne-se deles, quando possível;
- obtém *feedback* dos clientes do projeto e demais partes interessadas sobre o trabalho realizado durante o Sprint, ao apresentar e demonstrar os resultados desse trabalho ao final do Sprint;
- entrega valor com frequência para os clientes do projeto.

O Time de Desenvolvimento é:

- multidisciplinar, possuindo todas as habilidades e conhecimentos necessários para gerar, em cada Sprint, um Incremento do Produto pronto, de acordo com a Definição de Pronto;
- auto-organizado, planejando e executando seu trabalho com autonomia, propriedade e responsabilidade;
- suficientemente pequeno, de forma que seus membros se comuniquem efetivamente e se auto-organizem, sendo capazes de produzir Incrementos do Produto prontos que representem valor visível para os clientes;

- motivado, uma vez que possua o ambiente, apoio e a confiança necessários para realizar seu trabalho;
- orientado à excelência, buscando aprender e melhorar continuamente, realizando seu trabalho com consciência e qualidade;
- focado nos objetivos estabelecidos junto ao Product Owner.

6.2 O QUE FAZ O TIME DE DESENVOLVIMENTO?

6.2.1 Planeja seu trabalho

Na reunião de Sprint Planning, o Time de Desenvolvimento colabora e negocia com o Product Owner para decidirem o que será realizado no decorrer do Sprint que se inicia. Ou seja, a partir das prioridades definidas pelo Product Owner e da capacidade de trabalho percebida do Time de Desenvolvimento, eles decidem juntos quais e quantos itens estarão previstos para serem desenvolvidos nesse Sprint. Também definem a Meta do Sprint que o Time de Desenvolvimento vai se comprometer a realizar por meio desses itens selecionados.

Para auxiliá-lo com o planejamento, o Time de Desenvolvimento pode estimar o tempo ou esforço necessário para o desenvolvimento de cada item do Product Backlog, e usar essas estimativas como parâmetros. Uma vez que se meça, a partir das estimativas, o quanto em média o Time de Desenvolvimento entregou nos últimos Sprints, esse valor pode ser utilizado como um balizador de sua capacidade. É importante notar que essas estimativas, quando usadas, são sempre feitas pelo Time de Desenvolvimento e nunca pelo Product Owner, ScrumMaster ou qualquer outra pessoa.

O Time de Desenvolvimento também planeja como desenvolverá esses itens selecionados, o que em geral é feito quebrando-os em tarefas e, possivelmente, estimando-as. O conjunto formado pelos itens selecionados, suas tarefas correspondentes e o status delas é chamado de Sprint Backlog (veja [11 Sprint Backlog](#)).

6.2.2 Realiza o desenvolvimento do produto

Durante o Sprint, o Time de Desenvolvimento realiza as tarefas necessárias para transformar cada item do Sprint Backlog em uma funcionalidade pronta do produto, de acordo com a Definição de Pronto estabelecida em conjunto com o Product Owner. O Time de Desenvolvimento é responsável por garantir a qualidade dos resultados do seu trabalho, o que em geral é uma parte explícita da Definição de Pronto acordada.

É importante entender que, embora tenha a intenção de completar todo o trabalho planejado, o objetivo do Time de Desenvolvimento é o de realizar a Meta do Sprint, o que pode ocorrer sem que necessariamente tenham sido realizadas todas as tarefas de todos os itens selecionados do Product Backlog.

Aqui, considero **desenvolvimento do produto** toda e qualquer atividade necessária no trabalho de produção do Incremento do Produto, de acordo com a Definição de Pronto. No caso de desenvolvimento de *software*, por exemplo, esse trabalho pode incluir programação, diferentes tipos de testes e de documentação etc.

O Time de Desenvolvimento possui entre seus membros todos os conhecimentos e habilidades necessários para realizar o trabalho de desenvolvimento do produto, evitando assim dependências externas. O Time de Desenvolvimento também possui autonomia, dentro de limites estabelecidos pelo Scrum e pelo contexto organizacional, para realizar seu trabalho da forma que considerar mais apropriada, planejando-o, acompanhando seu progresso e buscando suas próprias soluções para os problemas que surgirão pelo caminho.

6.2.3 Colabora com o Product Owner durante o Sprint

Sempre que necessário, o Time de Desenvolvimento comunica-se e colabora com o Product Owner para que ele esclareça ou tome decisões rápidas sobre itens do Sprint Backlog. Para tornar essa comunicação possível, o Product Owner coloca-se acessível e disponível no decorrer do Sprint.

Caso contrário, o Time de Desenvolvimento pode ter de paralisar o desenvolvimento do item enquanto aguarda por essa interação, o que se torna um impedimento. Outra possível consequência é que o Time de Desenvolvimento termine por tomar decisões sobre o produto que não lhes cabem.

O Product Owner ter acesso a uma prévia e poder dar *feedback* antecipado sobre o trabalho em andamento pode ser saudável e reduzir os riscos de um possível fracasso do Sprint. O Product Owner, no entanto, de forma alguma interfere no planejamento realizado pelo Time de Desenvolvimento sobre como transformar os itens selecionados do Product Backlog em um Incremento do Produto, pois essas decisões não lhe cabe.

É imprescindível o Time de Desenvolvimento e o Product Owner interagirem para refinar e preparar itens no alto do Product Backlog para o Sprint a ser realizado. Diversos Times de Scrum realizam essa atividade durante a reunião de Sprint Planning, no início do Sprint onde os itens serão desenvolvidos.

Outra forma programada e mais efetiva de realizar essa atividade é utilizar sessões de Refinamento do Product Backlog no Sprint anterior, e nelas preparar os itens do Product Backlog para a próxima reunião de Sprint Planning, de acordo com uma Definição de Preparado acordada. Ambos os termos, “Refinamento do Product Backlog” e “Definição de Preparado”, não são parte do Scrum básico (veja [23 Refinamento do Product Backlog](#) e [14 Definição de Preparado](#)).

6.2.4 Identifica e informa os impedimentos ao ScrumMaster

Um impedimento é um obstáculo ou barreira que dificulta significativamente ou impede que o trabalho do Time de Desenvolvimento seja realizado, bloqueando um ou mais itens do Sprint Backlog de forma a ameaçar a Meta do Sprint. A resolução de um impedimento não está ao alcance do Time de Desenvolvimento, está fora do seu contexto de trabalho ou lhe tomará muito tempo. Problemas do dia a dia que o próprio Time de Desenvolvimento pode resolver, sem que ameacem a Meta da Sprint, não caracterizam impedimentos.

O Time de Desenvolvimento busca proteger-se e prevenir-se de impedimentos em seu trabalho. Uma vez que o Time de Desenvolvimento encontra

um impedimento que não pôde ser evitado, ele é imediatamente informado ao ScrumMaster que, então, toma as ações necessárias para removê-lo o mais rapidamente possível. Assim, os membros do Time de Desenvolvimento não esperam até a reunião de Daily Scrum seguinte para comunicar impedimentos ao ScrumMaster, mas sim o fazem ativamente durante o seu trabalho.

Sinalização de impedimentos

Além da comunicação verbal, uma boa prática que pode ser usada pelo Time de Desenvolvimento é a sinalização visual do impedimento no Quadro de Tarefas que representa o seu Sprint Backlog, junto ao item ou à tarefa afetada (veja [11 Sprint Backlog](#)).

Essa sinalização não substitui a ação de se informar o impedimento imediatamente ao ScrumMaster, mas ajuda tanto o ScrumMaster quanto o Time de Desenvolvimento a não negligenciarem e a melhor lidarem com o impedimento.

Um exemplo típico desse tipo de sinalização é a marcação do item ou tarefa impedidos com uma cor diferente, seja colando-se outra nota adesiva sobre o item ou tarefa (quando o Time de Desenvolvimento utiliza um Quadro de Tarefas real), mudando-se a cor do próprio item ou tarefa (no caso de um quadro virtual), ou usando-se algum outro tipo de marcação. A figura [6.1](#) mostra um exemplo de como se pode sinalizar um impedimento em uma tarefa.

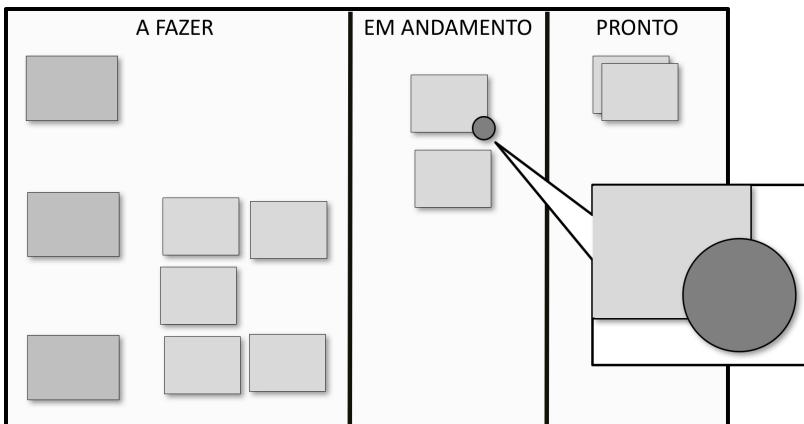


Fig. 6.1: Exemplo de sinalização de um impedimento no Quadro de Tarefas

Prevenção de impedimentos

Em geral, impedimentos podem evidenciar problemas sérios na estrutura da organização, na qualidade dos processos ou na qualidade dos produtos gerados. Assim, é importante que o Time de Desenvolvimento não entenda os impedimentos como um fator natural e aceitável do processo.

Impedimentos também geram desperdícios, pois além de provocar pausas e esperas, aumentam a quantidade de trabalho em progresso, podem provocar a execução de diversas atividades ao mesmo tempo, e estimulam um consumo inapropriado de energia do Time de Desenvolvimento. Esse consumo de energia ocorre desde a identificação do impedimento como tal, o que pressupõe um esforço inicial para resolvê-lo, até a interrupção do trabalho, a sinalização do impedimento ao ScrumMaster e as mudanças de contexto decorrentes de trocas do item em execução.

Considerando esses efeitos extremamente negativos dos impedimentos, é saudável que o Time de Desenvolvimento, antes de tudo, busque evitar a própria ocorrência do impedimento. Para que seja possível fazê-lo, o Time de Desenvolvimento pode utilizar alguns importantes recursos, como os exemplos descritos a seguir (PIMENTEL, 2009):

- **um verdadeiro compromisso com as metas** — quanto menor for o compromisso e, portanto, a motivação do Time de Desenvolvimento acerca da Meta do Sprint, mais facilmente um problema cuja resolução está ao alcance do Time de Desenvolvimento será entendido como impedimento e aceito como tal. Assim, esse problema poderá, mesmo que involuntariamente, ser usado como razão para atrasos ou mesmo para a não realização do trabalho;
- **o uso da dinâmica da iteração e seu *timebox* a seu favor** — solicitações externas ou iniciativas do próprio Time de Desenvolvimento que desviam da Meta do Sprint o foco de seu trabalho também podem transformar-se em impedimentos. Assim, é importante que o Time de Desenvolvimento seja capaz de questionar se a urgência da interrupção é real, ou se ela pode esperar o próximo Sprint, onde o trabalho necessário será devidamente planejado. A Meta do Sprint pode ficar seriamente ameaçada, por exemplo, quando um pedido da alta gerência fizer com que o Time de Desenvolvimento pare o que está fazendo para atendê-lo, abandonando o planejamento. O mesmo problema ocorre quando, no decorrer de um Sprint, o Time de Desenvolvimento resolve testar ou implantar novas ferramentas para melhorar sua produtividade, mas gasta tempo demais nessas atividades não planejadas. É importante lembrar aqui de que, caso a Meta do Sprint perca o sentido, o Sprint provavelmente será cancelado (veja [16.3 O Sprint pode ser cancelado?](#));
- **o acionamento preventivo do ScrumMaster** — assim que identifica um impedimento iminente, o Time de Desenvolvimento aciona o ScrumMaster para ajudá-lo a rejeitar esse impedimento antes mesmo que ele se concretize, protegendo as atividades que estão sendo executadas. Por exemplo, logo que identificar ser futuramente necessário o envolvimento pontual de uma pessoa ou equipe externa de difícil disponibilidade, o Time de Desenvolvimento pode imediatamente solicitar uma ação do ScrumMaster para se antecipar ao problema e garantir a disponibilidade dessa pessoa ou equipe;
- **a identificação de dependências em tempo de planejamento** — ao

se identificarem, em tempo de planejamento (Release Planning, Sprint Planning e Daily Scrum), dependências internas e externas que podem impactar o trabalho do Time de Desenvolvimento durante o Sprint, pode-se buscar meios de evitá-las ou antecipar o envolvimento e a ação de atores que possam ajudar a endereçá-las. Por exemplo, não se recomenda colocar em desenvolvimento uma funcionalidade que depende de uma ferramenta ou de uma pessoa externa que somente poderá se tornar disponível no decorrer do Sprint.

É importante, portanto, que o Time de Desenvolvimento busque se proteger e se prevenir dos impedimentos, tratando-os não somente como algo que prejudica suas pequenas tarefas do dia a dia, mas sim como algo que pode prejudicar e até mesmo comprometer as suas metas.

6.2.5 Obtém feedback sobre o produto

Na reunião de Sprint Review, o Time de Desenvolvimento, em colaboração com o Product Owner, apresenta e demonstra aos clientes e demais pessoas relevantes o trabalho desenvolvido no decorrer do Sprint: o Incremento do Produto pronto de acordo com a Definição de Pronto.

Além do Product Owner e membros do Time de Desenvolvimento, as demais pessoas presentes nessa reunião são preferencialmente as mais adequadas para prover *feedback* sobre o trabalho do Time de Desenvolvimento realizado no Sprint, como clientes ou usuários diretos dos itens que foram desenvolvidos. O *feedback* dessas pessoas é valioso, pois permite a adaptação do produto de acordo com as suas necessidades, servindo de matéria-prima para o Product Owner atualizar o Product Backlog.

6.2.6 Entrega valor com frequência

A partir das prioridades definidas pelo Product Owner para trazer valor para os clientes, o Time de Desenvolvimento gera, em cada ciclo de desenvolvimento, um Incremento do Produto que é um conjunto de funcionalidades prontas de acordo com a Definição de Pronto. Isso que significa que essas funcionalidades já poderiam ser entregues e usadas pelos usuários do projeto.

A decisão de fazer uma Release do produto (ou seja, de fato entregar um ou mais Incrementos para os clientes) pertence ao Product Owner. No entanto, ao gerar valor em cada Sprint, o Time de Desenvolvimento possibilita que essa entrega de valor seja frequente.

6.3 COMO É O TIME DE DESENVOLVIMENTO?

6.3.1 Multidisciplinar

O Time de Desenvolvimento é multidisciplinar, possuindo em seus membros todos os conhecimentos e habilidades necessários para gerar o Incremento do Produto pronto em cada Sprint, de acordo com a Definição de Pronto. A Definição de Pronto, portanto, está intimamente relacionada com a formação do Time de Desenvolvimento.

Não há papéis definidos dentro de um Time de Desenvolvimento. Independente de suas habilidades e conhecimentos individuais, todos são igualmente responsáveis pelo desenvolvimento do produto e pela realização dos objetivos estabelecidos.

Dessa forma, mesmo que neles haja concentração de conhecimentos específicos, não rotulamos membros do time como “testadores”, *front-end*, *back-end*, *designer* de experiência do usuário ou nada do gênero. São todos igualmente “membros do Time de Desenvolvimento” ou, talvez, “desenvolvedores”. Da mesma forma, não há um líder-técnico ou qualquer outro tipo de liderança oficial no time.

Dependências externas, que não estão sob o controle do Time de Desenvolvimento, constituem um risco muito alto para o trabalho realizado durante o Sprint. Se for necessário que o Time de Desenvolvimento espere o resultado do trabalho de pessoas ou times externos para poder realizar o desenvolvimento de algum de seus itens, pode ser impossível realizar a Meta do Sprint. Um Time de Desenvolvimento multidisciplinar reduz esses riscos, já que é capaz de realizar todo o trabalho sem, a princípio, depender de ninguém.

Embora não seja obrigatório, não é incomum membros do Time de Desenvolvimento possuírem conhecimentos na área de negócios do produto em desenvolvimento, e poderem assim colaborar com o Product Owner, e interagirem com os clientes e demais partes interessadas para a definição dos

detalhes do produto. Mesmo nesses casos, o Product Owner mantém total autoridade sobre as decisões relativas ao produto.

Polinização cruzada

O fato de um Time de Desenvolvimento ser multidisciplinar, no entanto, não necessariamente implica em possuir membros multidisciplinares. As pessoas naturalmente possuem suas especialidades, conhecimentos e áreas de interesse específicos. Mas, como com Scrum a responsabilidade sobre os resultados do trabalho recai sobre todo o Time de Desenvolvimento e não sobre determinado indivíduo, os membros do Time de Desenvolvimento são estimulados a trabalhar juntos, trocar conhecimentos e desenvolver habilidades secundárias.

Essa “polinização cruzada” reduz a dependência nas especialidades de determinados membros, diminuindo o risco de não se realizar a Meta do Sprint. A ausência ou sobrecarga de um de seus membros em um determinado momento, por exemplo, tem menos chances de se tornar um impedimento para o trabalho do Time de Desenvolvimento, pois normalmente haverá outros ao menos minimamente capazes de realizar o trabalho ou de ajudá-lo.

O compartilhamento de conhecimentos também reduz as chances de que, em alguns momentos, membros do Time de Desenvolvimento fiquem sem ter o que fazer e, assim, procurem trabalhar em partes específicas de itens de menor importância, ou até mesmo se vejam obrigados a trabalhar em mais de um time ao mesmo tempo. Um especialista que só saiba lidar com bancos de dados, por exemplo, enfrentaria esse problema, já que só teria trabalho em momentos específicos do Sprint.

Em um Time de Desenvolvimento ideal, cada membro seria capaz de utilizar suas diferentes habilidades e conhecimentos para, em conjunto com os outros membros, realizar os diversos tipos de atividades necessárias durante o Sprint. Não haveria distinção entre essas atividades por sua natureza. Ou seja, todos os membros do Time de Desenvolvimento seriam capazes de trabalhar em questões como arquitetura de produto, os diferentes níveis de desenvolvimento, os testes necessários, documentação etc.

Aprendizado

De forma geral, espera-se que uma parte significativa dos conhecimentos e habilidades necessários já exista no Time de Desenvolvimento desde o momento de sua formação, ou seja, selecionam-se as pessoas mais adequadas para o projeto. No entanto, novos desafios que exigem aprendizado surgem naturalmente ao longo do projeto. Por meio da “polinização cruzada”, os conhecimentos e habilidades novos e já existentes são disseminados entre os membros do Time de Desenvolvimento.

A forma como conhecimentos e habilidades são adquiridos e compartilhados entre os membros de um Time de Desenvolvimento varia de time para time. As atividades que promovem esse aprendizado podem ser classificadas em formais e informais.

Muitas organizações contratam treinamentos formais para seus funcionários, o que é sempre limitado pelo orçamento disponível. É comum, por outro lado, os próprios membros de Times de Desenvolvimento da organização promoverem, periódica ou eventualmente, sessões de treinamento para seus colegas. Estas são geralmente em torno de assuntos em que esses indivíduos são especialistas ou sobre os quais aprenderam recentemente, seja por conta própria ou por meio de treinamentos contratados (ARMONY, 2010).

Típico entre desenvolvedores de *software*, o *coding dojo* é outro método formal de treinamento técnico para membros de Times de Desenvolvimento. Por seu aspecto lúdico, a prática do dojo ganha ares informais, embora seja uma forma extremamente disciplinada e eficiente de se cultivar habilidades para o desenvolvimento de *software*.

CODING DOJO

Coding Dojo é um encontro onde um grupo de programadores se reúne para trabalhar em um desafio de programação. Eles estão lá para se divertir e se engajarem em atividades direcionadas a melhorar suas habilidades, proporcionando *feedback* (tanto sobre o código produzido quanto sobre as técnicas usadas), que permite o aprendizado.

Para mais informações, acesse: <http://codingdojo.org/>.

O treinamento informal se dá a partir do compartilhamento de conhecimento com colegas de trabalho ou com outras pessoas de fora da equipe ou organização. Isso pode acontecer no dia a dia de trabalho ou, por exemplo, com a participação em feiras e eventos.

Oriunda do Extreme Programming, a prática da programação em par tem se mostrado uma forma muito eficiente de treinamento informal em projetos de *software*. Entre outros benefícios que ela traz, ao se colocar juntos em um mesmo computador um desenvolvedor menos ou não capacitado em uma determinada matéria e outro mais capacitado, o segundo ensina ao primeiro enquanto ambos executam o trabalho. Essa prática pode acontecer em tempo integral, sistematicamente durante algumas horas por dia ou eventualmente.

Discussão: qualidade e testadores em um Time de Scrum

O Time de Desenvolvimento é integralmente responsável por gerar, em cada Sprint, um Incremento do Produto pronto, de acordo com a Definição de Pronto (veja [12 Definição de Pronto](#)). Esse trabalho inevitavelmente inclui todo e qualquer tipo de teste necessário para garantir a qualidade no desenvolvimento produto. Portanto, a habilidade de realizar esses testes deve estar presente no Time de Desenvolvimento, seja em membros bem definidos ou espalhada pelos diversos membros do time.

Não há papéis definidos em um Time de Desenvolvimento; logo, não existe o papel de “testador” no Scrum. Assim, mesmo quando o conhecimento necessário para realizar os testes se concentra em um membro ou em membros específicos, a responsabilidade pela qualidade do Incremento do Produto gerado não recai sobre ele ou eles. Essa responsabilidade é compartilhada por todo o Time de Desenvolvimento, ainda que, por herança de métodos tradicionais, eles sejam caracterizados como “testadores”.

O mesmo princípio deve ser aplicado a outros conhecimentos que se façam necessários, como por exemplo, o *design* da experiência do usuário (ou *user experience*, em inglês). Assim, esse conhecimento deve existir dentro do Time de Desenvolvimento, em um ou mais membros, mas não debaixo de um papel definido cuja única responsabilidade é de tratar desse assunto. Não há também um time à parte realizando esse trabalho com antecedência. A experiência do usuário deve ser, assim como todo o resto do produto, desen-

volvida iterativa e incrementalmente pelo Time de Desenvolvimento, e está sujeita a *feedback* de uso.

6.3.2 Auto-organizado

Ao contrário do que se espera de times tradicionais, não há gerentes de projeto dizendo para o Time de Desenvolvimento como ele deve realizar seu trabalho, nem pressionando ou cobrando informações sobre o andamento de cada tarefa. Com Scrum, o microgerenciamento realizado por um agente externo dá lugar à auto-organização.

Scrum foi concebido de forma a estimular a auto-organização de seus times. O Time de Desenvolvimento, por ser aquele que realiza o trabalho de desenvolvimento, é o mais indicado para planejar e gerenciar esse trabalho.

Entretanto, essa auto-organização não deve ser confundida com anarquia ou falta de controle. Ainda que, de forma auto-organizada, os membros do Time de Desenvolvimento trabalhem seguindo as necessidades dos clientes quanto ao produto a ser gerado, em direção à Meta do Sprint que negociaram com o Product Owner e com a qual se comprometeram.

Além disso, o Time de Desenvolvimento está inserido no contexto de uma organização. Portanto, deve estar alinhado com as regras e objetivos dessa organização, inclusive no que diz respeito ao uso do *framework* Scrum.

Para facilitar a auto-organização, Scrum estimula a visibilidade ou transparência, possuindo pontos-chave nos quais o andamento do trabalho ou seus resultados são inspecionados. Um exemplo são as reuniões de Daily Scrum, na qual o próprio Time de Desenvolvimento inspeciona seu andamento em direção à Meta do Sprint; outro é de Sprint Review, onde os resultados do trabalho no Sprint são inspecionados pelos clientes e partes interessadas.

Scrum também possui ferramentas que aumentam a visibilidade do andamento do trabalho, como o Sprint Backlog e gráficos que indicam o progresso ou o trabalho restante.

Scrum oferece objetivos claros a serem realizados no curto prazo, traduzidos em cada Sprint na forma da Meta do Sprint (veja em [24.2 Meta de Sprint](#)). Esse objetivo comum entre os membros do Time de Desenvolvimento fornece o alinhamento necessário para a auto-organização acontecer. A responsabilidade compartilhada dos membros do Time de Desenvolvimento sobre esse

objetivo, em oposição à responsabilidade individual sobre as suas partes, é outro elemento essencial, pois estimula a preocupação com o todo. Não há o “meu” e o “seu”, mas sim o “nossa”.

A presença dos membros do Time de Desenvolvimento na reunião de Sprint Review reforça esse senso de responsabilidade dos membros do Time de Desenvolvimento sobre os resultados de seu trabalho, já que lá estarão frente a frente com clientes e partes interessadas para os quais o trabalho está sendo realizado. Além disso, como mencionei anteriormente, não há líderes-técnicos ou quaisquer outros tipos de lideranças oficiais dentro do Time de Desenvolvimento. Isso porque, caso existissem, contariam com uma carga diferenciada de responsabilidade e prejudicariam a auto-organização.

Além disso, é importante que o Time de Desenvolvimento possua o espaço necessário para criar e desenvolver, dentro de sua capacidade e ritmo, seus próprios meios para realizar metas estabelecida para cada Sprint, tornando-se o principal responsável por essas metas.

Para que a auto-organização se torne possível, é imprescindível que o Time de Desenvolvimento realize o trabalho de forma colaborativa, maximizando a comunicação entre seus membros. Essa é uma das grandes justificativas do porquê de o Time de Desenvolvimento dever se manter pequeno em número de membros.

O Time de Desenvolvimento conta ainda com o apoio de um facilitador, o ScrumMaster, para estimulá-lo a tomar responsabilidade sobre seus problemas e sobre suas soluções, aumentando sua autonomia. O ScrumMaster ainda, entre outras coisas, remove impedimentos que atrapalhem o trabalho do time, ensina o uso de Scrum e facilita seu acesso ao meios necessários para realizar esse trabalho.

6.3.3 Suficientemente pequeno

O Time de Desenvolvimento deve ser suficientemente pequeno. De forma geral, recomenda-se que o tamanho do Time de Desenvolvimento seja algo entre 3 e 9 membros. Esses números não incluem o ScrumMaster e o Product Owner, a menos que também exerçam o papel de membro do Time de Desenvolvimento.

Esses parâmetros servem apenas como orientação. Times de Desenvolvimento menores do que três pessoas e não muito maiores do que nove também podem se beneficiar do uso do Scrum. Na realidade, ao escolher o número de membros do Time de Desenvolvimento espera-se que:

- os membros do Time de Desenvolvimento sejam capazes de se comunicar efetivamente para se auto-organizarem. O número de canais de comunicação cresce exponencialmente com o número de pessoas envolvidas. Assim, times grandes têm maiores dificuldades de comunicação e coordenação, o que dificulta e até pode impossibilitar a auto-organização;
- o Time de Desenvolvimento possua todos os conhecimentos e habilidades necessários para produzir um Incremento do Produto pronto, de acordo com a Definição de Pronto. Esse trabalho se traduz em funcionalidades prontas de ponta a ponta, que representam valor para os clientes, em vez de partes de funcionalidades ou camadas. Pode ser mais difícil encontrar todos os conhecimentos e habilidades necessários em um número muito pequeno de pessoas;
- o Time de Desenvolvimento seja capaz de produzir valor visível suficiente em cada Sprint, para que se possa obter *feedback* dos clientes e demais partes interessadas na reunião de Sprint Review.

Para times pequenos demais, é possível que o Scrum gere uma sobrecarga com suas regras, papéis e eventos que supere seus benefícios.

Quando o projeto é muito grande, pode-se utilizar, de forma coordenada, mais de um Time de Desenvolvimento trabalhando no mesmo produto. Nesse caso, cada um desses Times de Desenvolvimento do projeto deve ser suficientemente pequeno, de acordo com a definição dada nesta seção.

6.3.4 Motivado

A falta de motivação no trabalho é historicamente relacionada a baixos rendimentos, faltas, atrasos, tédio, frustração, insatisfação e ineficiência entre trabalhadores (MOTTA, 1991).

Ao buscar fatores que influenciam a motivação dos membros de um Time de Desenvolvimento, deve-se levar em conta que os seres humanos são complexos por natureza e, assim, são em cada instante influenciados por suas necessidades, desejos, aspirações, preferências, problemas, frustrações e toda a sorte de emoções. O homem, portanto, possui momentos bons e momentos ruins na realização do seu trabalho.

Em um Time de Desenvolvimento, a motivação no trabalho é essencial para gerar compromisso de seus membros com as metas e com a qualidade do produto de seu trabalho. A motivação também pode aumentar a estabilidade da composição do Time de Desenvolvimento, já que pode levar a uma menor rotatividade de pessoal, ajudando a não se perder o conhecimento e ritmo de trabalho adquiridos.

Os princípios Ágeis apontam explicitamente o ambiente, o suporte e a confiança necessários para realizar o trabalho como essenciais para a motivação do indivíduo. Outro fator que pode ser entendido como motivador, também apontado pelos princípios Ágeis, é a existência de um ritmo sustentável de trabalho, segundo o qual se evita a prática de horas extras e a aceleração forçada do ritmo de trabalho, por exemplo, diante da perspectiva de atraso em uma entrega prevista.

O ScrumMaster, enquanto uma liderança motivadora, também exerce influência sobre a motivação dos membros do Time de Desenvolvimento, facilitando a auto-organização e encorajando-os à auto-observação, autoavaliação e autorreforço (veja *Liderança no time auto-organizado*, em 8.2 *O que faz o ScrumMaster*).

A partir de reconhecidas teorias sobre a motivação no trabalho, podemos também destacar exemplos de fatores motivacionais relevantes a Times de Desenvolvimento (ARMONY, 2010).

Teoria da fixação de objetivos

De acordo com essa teoria, a fixação de objetivos desafiadores, claros e realizáveis, o compromisso com esses objetivos e a disponibilidade de *feedback* mostrando o progresso em direção a esse objetivo são poderosos mecanismos motivacionais (LOCKE, 1996), tanto para indivíduos quanto para grupos (LOCKE; LATHAM, 2006).

Os objetivos no Scrum são as Metas estabelecidas para os Sprints. Adicionalmente, pode-se estabelecer metas para as Releases (ou marcos no Roadmap do Produto) e uma Visão do Produto. O *feedback* sobre o progresso do Time de Scrum em direção aos objetivos vem, por exemplo, por meio dos Gráficos de Burndown ou Burnup de Trabalho e da reunião de Sprint Review.

De acordo com o princípio Ágil *software em funcionamento é a principal medida de progresso*, podemos também afirmar que os *feedbacks* dos clientes e demais partes interessadas sobre Incrementos do Produto entregues também indicam o progresso em direção aos objetivos estabelecidos.

Teoria das características do trabalho

Segundo essa teoria, o indivíduo pode se motivar em seu trabalho se ele percebe esse trabalho como compensador ou importante, acredita que é diretamente responsável por seus resultados e é capaz de determinar se esses resultados foram ou não satisfatórios (HACKMAN et al., 1975).

Para se chegar a esses estados psicológicos, é importante que determinadas características estejam presentes no trabalho do indivíduo. A primeira delas é o indivíduo necessitar de uma variedade de habilidades e conhecimentos para executar seu trabalho, o que ocorre naturalmente em um Time de Desenvolvimento multidisciplinar. Outra dessas características é o indivíduo executar uma parte inteira e identificável do trabalho do início ao fim, característica importante do Scrum em que se produzem, em cada Sprint, funcionalidades prontas e de ponta a ponta.

O indivíduo deve também perceber seu trabalho como tendo um impacto significativo sobre a vida ou trabalho de outras pessoas, o que ocorre no Scrum a partir do *feedback* provindo das reuniões de Sprint Review e das entregas frequentes. Esse *feedback*, somado à autorregulação do time, dá uma noção ao indivíduo sobre a sua eficácia e desempenho, mais uma característica necessária. Por fim, a autonomia na realização do trabalho, outra dessas características, é natural em um time auto-organizado.

Teoria ERC

Segundo essa teoria, o homem é motivado por três categorias de necessidades que devem ser satisfeitas, ordenadas da seguinte forma: as necessidades

de existência, as de relacionamento e as de crescimento (ALDERFER, 1969; ALDERFER et al., 1974).

A existência de condições básicas de trabalho, como salários em dia, ambiente de trabalho adequado, suporte técnico efetivo e infraestrutura suficiente é uma necessidade de existência de membros do Time de Desenvolvimento.

As necessidades de relacionamento do indivíduo podem ser expressas pela intensa comunicação e pelo bom relacionamento entre os membros do Time de Scrum, que leva ao compartilhamento de pensamentos e sentimentos relevantes.

Já as necessidades de crescimento do indivíduo podem ser atendidas a partir da perspectiva de evolução de carreira na organização. Ou seja, o indivíduo se motiva diante da possibilidade clara de ascensão profissional advinda do reconhecimento do trabalho realizado ou do potencial percebido.

A Teoria ERC é uma evolução da conhecida Teoria das Necessidades de Maslow.

Discussão: avaliação e recompensa

As avaliações e recompensas relativas ao trabalho realizado (bônus, por exemplo), quando dirigidas ao indivíduo, estimulam-no a trabalhar como indivíduo. Quando dirigidas ao time, estimulam seus membros a trabalharem como um time. Deixar a critério de um gerente julgar a contribuição individual de membros do time somente gera conflitos e desentendimentos, reduzindo a efetividade do time, já que cada membro passa a trabalhar em detrimento do outro.

Ao recompensar o indivíduo, o que se está incentivando é a competição entre os membros do time em vez da colaboração para se realizar os objetivos comuns (HACKMAN, 1987). E essa competição não necessariamente é pelo melhor trabalho, mas muitas vezes por uma maior visibilidade diante do gerente que está julgando. Assim, existe uma ampla preferência por que times Ágeis sejam avaliados e recompensados como um time, e não como indivíduos.

Tenho visto, no entanto, algumas abordagens de avaliação cruzada entre membros do Time de Desenvolvimento funcionando muito bem. Nelas,

cada membro avalia seus colegas de acordo com parâmetros estabelecidos pelo próprio time. O conjunto das avaliações realizadas, então, serve de base para o sistema de recompensas, sendo em geral cruzado com a avaliação externa da contribuição do Time de Desenvolvimento como um todo.

Recomendo que esses parâmetros de avaliação digam mais respeito a questões qualitativas, como a contribuição do indivíduo para a efetivação de valores como colaboração, comunicação, foco etc., em vez de dados quantitativos, como a quantidade de trabalho realizada por cada um, número de erros gerados etc.

6.3.5 Orientado à excelência

A atenção contínua à excelência técnica e a um bom projeto aumenta a Agilidade. Esse princípio Ágil reforça a ideia de que um Time de Desenvolvimento deve possuir pessoas tecnicamente qualificadas que trabalhem de forma a produzir com consciência e entregar Incrementos do Produto de alta qualidade. Não existe mágica: não há método ou processo que faça com que maus profissionais produzam bons resultados.

Problemas técnicos podem se acumular no projeto devido à falta de qualificação ou até mesmo por falta de compromisso com a qualidade. Times de Desenvolvimento são orientados à excelência técnica e, assim, não apenas possuem os conhecimentos e habilidades necessários, mas também buscam aprender continuamente como podem melhor realizar seu trabalho. Times de Desenvolvimento também se preocupam em não deixar soluções malfeitas ou temporárias no produto gerado.

Em intervalos de tempo regulares, a equipe reflete sobre como se tornar mais efetiva e, então, refina e ajusta seu comportamento de acordo. Esse princípio Ágil diz respeito à busca do Time de Desenvolvimento pela melhoria contínua. Ainda que se considere suficientemente qualificado, o Time de Desenvolvimento está sempre buscando melhores formas de fazer seu trabalho, de se tornar mais efetivo e de produzir com mais qualidade.

Diversas práticas do Scrum reforçam essa ideia de melhoria contínua. Entre elas, a reunião de Sprint Retrospective.

6.3.6 Focado nos objetivos

Um objetivo ou meta no Scrum é uma necessidade de negócios ou do usuário a ser atendida como resultado de um trabalho. Esses objetivos são estabelecidos pelo Product Owner e ajustados em conjunto com o Time de Desenvolvimento. Nessa colaboração, o Product Owner sabe que forçar o Time de Desenvolvimento além de sua capacidade de produção não trará os resultados que ele almeja e que um Time de Desenvolvimento comprometido buscará dar o melhor de si para entregar valor para os clientes do projeto.

Cada Sprint possui um objetivo definido, chamado de Meta do Sprint, que pode traduzir o valor dessas necessidades de negócios ou do usuário relativo ao trabalho realizado no Sprint. Além da Meta do Sprint, pode-se definir uma Visão do Produto e uma Meta para cada Release ou marco do Roadmap do Produto. Estas provêm um foco e uma direção para o trabalho do Time de Desenvolvimento, incentivando a auto-organização. Além disso, elas motivam e oferecem ao Time de Desenvolvimento alguma flexibilidade sobre as entregas correspondentes.

O Time de Desenvolvimento mantém seu foco de trabalho nos objetivos estabelecidos e, assim, dedica o máximo do seu tempo útil ao projeto. Um Time de Desenvolvimento multitarefa, que trabalha em mais de um projeto ao mesmo tempo ou em outras atividades além do projeto, encontra grandes desafios para conseguir manter seu foco nesses objetivos.

Para ajudar a manter seu foco na Meta do Sprint, o Time de Desenvolvimento pode se comprometer formalmente com a mesma na reunião de Sprint Planning, tornando-se responsável por realizá-la e sendo responsabilizado por seus resultados.

CAPÍTULO 7

Product Owner

7.1 QUEM É O PRODUCT OWNER?

O Product Owner, também chamado de P.O., é a pessoa responsável pela definição do produto, trabalho que é realizado de forma incremental ao longo de todo projeto. Seu objetivo primário é garantir e maximizar, a partir do trabalho do Time de Desenvolvimento, o retorno sobre o investimento para os clientes do projeto, satisfazendo suas necessidades com relação ao produto em desenvolvimento.

Para realizar esse trabalho, o Product Owner:

- gerencia o produto, inserindo, detalhando, removendo e priorizando os itens de trabalho do produto no Product Backlog, a partir do contato frequente com os clientes do projeto e demais partes interessadas;

- gerencia os clientes e demais partes interessadas em sua relação com o projeto, descobrindo quem são essas partes interessadas que devem influenciar as decisões sobre o produto, balanceando suas necessidades com relação ao produto, comunicando-se com elas para descobrir essas necessidades, e influenciando-as para maximizar sua colaboração com relação ao projeto;
- mantém os objetivos do produto, comumente expressos na forma de uma Visão do Produto, definindo-os junto aos clientes do projeto e demais partes interessadas, comunicando-os a todos os envolvidos e garantido que o trabalho seja realizado em direção a eles;
- gerencia as entregas do produto, definindo a melhor estratégia para a realização dessas entregas e seus objetivos;
- participa do Sprint nos eventos do Scrum, realizando com o Time de Desenvolvimento o planejamento do Sprint na reunião de Sprint Planning, a apresentação e coleta de *feedback* dos clientes e demais partes interessadas na reunião de Sprint Review e o processo de melhoria contínua na reunião de Sprint Retrospective;
- colabora com o Time de Desenvolvimento, sempre que necessário, mostrando-se disponível para esclarecer dúvidas e tomar decisões quanto aos detalhes do produto, e para refinar e preparar o Product Backlog para o desenvolvimento.

O Product Owner é:

- único para um Time de Scrum, pois deve haver apenas um foco de decisões sobre o produto para o Time de Desenvolvimento;
- disponível para colaborar com o Time de Desenvolvimento, para estar presente nas reuniões do Scrum em que sua presença é obrigatória, para interagir com os clientes e demais partes interessadas, e para manter e priorizar o Product Backlog;
- competente para a definição do produto, com conhecimento e poder suficiente para tomar decisões rápidas e adequadas.

Discussão: de onde vem o Product Owner?

A escolha mais comum para Product Owners em projetos de desenvolvimento de *software* se dá entre alguém do próprio cliente e alguém da organização contratada para gerar o produto. Escolher para Product Owner o próprio cliente ou alguém designado por ele é uma opção adotada por muitas organizações e defendida por vários especialistas em Scrum.

Nesse caso, o Product Owner define o produto a ser desenvolvido a partir de suas próprias necessidades ou de sua organização. Naturalmente, essa escolha busca garantir que o Product Owner possua o conhecimento do negócio e que tome decisões de produto que favoreçam o próprio cliente.

Minha experiência, no entanto, mostra que essa não é a melhor opção. Na realidade, já vi muitos projetos fracassarem como decorrência dessa escolha. Apresento aqui diversas razões pelas quais defendo que o Product Owner, em princípio, deve ser alguém próximo ao time que está desenvolvendo o produto.

Esse é um papel que exige grande preparo, e raramente um Product Owner do cliente saberá exercê-lo. Na maioria das vezes, ele simplesmente não tem as habilidades e os conhecimentos necessários, que obrigatoriamente incluem o próprio *framework* Scrum e saber definir o produto de forma Ágil, ou seja, iterativa e incremental. Esse problema é mais comum e mais grave do que parece.

Defende-se que, para resolver essa questão, o ScrumMaster pode ensinar Scrum e outras técnicas ao Product Owner, que também pode realizar treinamentos específicos. Mas eu pergunto: quanto esforço do projeto se quer investir em uma formação tão complexa e tão específica, que engloba uma série de habilidades, conhecimentos e práticas, para a princípio ser utilizada apenas na duração desse projeto?

Esse é um papel que exige enxergar os problemas de fora, talvez como um consultor. O cliente está imerso no contexto de seus problemas, e geralmente não os identifica com clareza, raramente sabendo o que realmente precisa. Por conta disso, não é nada incomum ele trazer soluções “prontas” para problemas que ele não entendeu ainda, fazendo com que o time desenvolva soluções que não entregam muito valor. O time pode fazer exatamente o que o cliente pediu, mas esse cliente não ficará satisfeito, já que os problemas reais não

serão resolvidos.

Esse é um papel que exige disponibilidade e colaboração. Mesmo que esse Product Owner do cliente possua os conhecimentos e habilidades necessários, é bem possível que ele não possua tempo disponível para executar o seu papel, pois provavelmente estará envolvido em outras questões de seu próprio dia a dia de trabalho no cliente. Além disso, sua relação com o Time de Desenvolvimento poderá se converter em uma relação de cobrança, e não de colaboração como se espera de membros de um mesmo Time de Scrum. O resultado disso pode ser um Product Owner que: pouco interage com o Time de Desenvolvimento; cobra do time em vez de colaborar; possivelmente não está presente em todas as reuniões do Scrum em que sua presença é necessária; e não consegue dedicar tempo para pensar no produto e refinar o Product Backlog.

Esse é um papel que exige equilíbrio. Quando o projeto possui vários clientes, que podem até mesmo pertencer a diferentes organizações, fica claro o desvio na escolha de um Product Owner como um dos clientes. Somente pode haver um Product Owner no Time de Scrum e, assim, o cliente que for escolhido para o papel poderá tomar decisões que o favoreçam em detrimento dos outros.

Recomendo fortemente, para a grande maioria dos casos, que o Product Owner seja designado pela própria organização ou grupo contratado para gerar o produto. Deve ser alguém formado para tal, com as habilidades e conhecimentos necessários. Ele realizará seu trabalho a partir do contato frequente com os clientes do projeto e com as demais partes interessadas, balanceando as necessidades das partes interessadas, mas tomando decisões alinhadas com ou em direção aos objetivos da sua própria organização.

Lembro de que, para ser de fato um Product Owner, essa pessoa deve ter a autoridade para tomar as decisões necessárias sobre o produto. Ou seja, mesmo que não seja alguém do cliente, o Product Owner é sempre quem de fato define o produto.

7.2 O QUE FAZ O PRODUCT OWNER?

7.2.1 Gerencia o produto

O Product Owner mantém um contato frequente com os clientes e demais partes interessadas ao longo de todo o projeto para identificar e entender os objetivos ou necessidades de negócios mais prioritárias do produto em cada momento. Isso, claro, dentro dos objetivos estabelecidos para o produto, estes em geral expressos na forma de uma Visão do Produto (veja em [24.5 Visão do Produto](#)).

Ele decide quais dessas necessidades farão parte do produto, e as insere como itens em uma lista ordenada, chamada de Product Backlog. Ele ordena, atualiza e reordena esses itens. Também detalha os itens progressivamente à medida que ganham importância, e remove aqueles que não são mais necessários e que não deverão mais ser desenvolvidos. Ele também garante a visibilidade do Product Backlog ao Time de Desenvolvimento.

O Product Owner realiza a gestão do produto e é o único que pode alterar o Product Backlog ou, ao menos, que tem a palavra final e a responsabilidade sobre ele. Nesse trabalho de gestão do produto, é importante que o Product Owner desenvolva uma estratégia de negócios para o desenvolvimento do produto em direção aos objetivos traçados para ele, utilizando-se dessa estratégia para tomar as decisões necessárias quanto ao que será desenvolvido.

Essa estratégia, assim como o próprio desenvolvimento do produto, evolui de forma iterativa e incremental. Um formato possível de estratégia de negócios é o Roadmap do Produto, que descreve o futuro do produto em alto nível. No Roadmap, apontam-se uma data provável para cada Release ou para marcos do projeto, e a Meta a ser realizada em cada uma dessas Releases ou marcos (veja em [24.4 Roadmap do Produto](#)).

Além dos clientes do projeto e demais partes interessadas, o Product Owner também colabora com o próprio Time de Desenvolvimento para tomar as decisões quanto ao produto. Durante todo o projeto, o Time de Desenvolvimento oferece suas ideias e sua perspectiva quanto ao que será desenvolvido. Embora a decisão do que será feito caiba ao Product Owner, não é incomum membros do Time de Desenvolvimento possuírem conhecimento do negócio

e o ajudarem diretamente, tanto nessa tomada de decisão quanto no contato com os clientes e demais partes interessadas.

O Time de Desenvolvimento também influencia diretamente as decisões do Product Owner, oferecendo uma visão do que é tecnicamente possível, do custo de desenvolvimento de cada item do Product Backlog e de que trabalho técnico adicional poderá ser necessário. Com relação a esta última questão, cabe ao Time de Desenvolvimento identificar itens de trabalho técnico — como o teste de uma nova ferramenta ou a melhoria de alguma parte do produto, por exemplo —, e sugerir ao Product Owner sua inserção no Product Backlog. Cabe ao Product Owner decidir, após entender o que for necessário junto ao Time de Desenvolvimento, se esses itens serão realmente desenvolvidos e quando.

Discussão: proxy do cliente?

Ao contrário do que já vi e ouvi por aí ao longo dos anos, o Product Owner **não é**:

- o *proxy* do(s) cliente(s);
- um canal de comunicação para o(s) cliente(s);
- o representante do(s) cliente(s);
- um tomador de pedidos ou de *tickets* do(s) cliente(s).

Ao contrário, o Product Owner é aquele que define o produto. Ele possui autoridade e conhecimento das técnicas necessárias para fazê-lo de forma a satisfazer às necessidades dos clientes, e entende que não será capaz de atingir esse objetivo caso simplesmente atenda a seus desejos e vontades. O cliente, de forma geral, não sabe o que precisa e, assim, o Product Owner é responsável por descobrir, definir e validar soluções junto aos usuários, clientes e partes interessadas ao longo de todo o projeto.

7.2.2 Gerencia as partes interessadas no projeto

Além dos clientes, as partes interessadas no projeto são os usuários, patrocinadores e quaisquer pessoas que tenham algum tipo de influência nas defi-

nições do projeto ou tenham interesse direto, seja no seu andamento, no seu sucesso ou em seus impactos.

O Product Owner faz a gestão dos clientes do projeto e demais partes interessadas em sua relação com o produto que está sendo desenvolvido. Este trabalho inclui:

- identificar corretamente quem são os clientes e pessoas relevantes do projeto, para interagir com as pessoas certas e, assim, definir as necessidades de negócios;
- entender as necessidades dos diferentes clientes e demais partes interessadas com relação ao produto;
- gerenciar as expectativas dos clientes e demais partes interessadas com relação ao produto, garantindo que entendam o que está para ser demonstrado ou entregue, e o que será realizado em seguida;
- balancear e gerenciar conflitos entre as necessidades e expectativas dos diferentes clientes e demais partes interessadas;
- ajudar os clientes e demais partes interessadas a descobrir e entender o que lhes trará maior retorno em cada momento do projeto;
- obter a colaboração direta dos clientes e demais partes interessadas sempre que se mostrar necessária como, por exemplo, para ajudar o ScrumMaster na remoção de impedimentos organizacionais ou relacionados ao ambiente do cliente;
- Comunicar-se frequentemente e pró-ativamente com os clientes e demais partes interessadas para realizar esse trabalho.

7.2.3 Mantém os objetivos do produto

O Product Owner é o responsável por definir claramente os objetivos do produto, que são comumente expressos na forma de uma Visão de Produto. Embora não faça parte do *framework* Scrum, a definição de uma Visão do Produto é uma prática importante para o trabalho no projeto de desenvolvimento de um produto.

A Visão do Produto responde de forma curta e direta à pergunta: “por que esse produto está sendo desenvolvido?”. Ou, ainda melhor, “que problema será resolvido com o desenvolvimento desse produto?”.

Ela serve de guia para o trabalho do Time de Scrum e alinha o entendimento e as expectativas quanto ao produto entre as diferentes partes interessadas do projeto e o Time de Scrum. Para tornar esse alinhamento possível, o Product Owner comunica a Visão a todos os envolvidos, assegurando que ela seja compreendida da mesma forma por todos e durante todo o projeto.

O Product Owner estabelece a Visão do Produto junto aos clientes e demais partes interessadas do projeto antes do início do desenvolvimento do produto e, portanto, antes do uso do Scrum. Se possível, é importante também envolver o Time de Desenvolvimento nas atividades necessárias para se chegar a essa definição, pois isso aumenta seu senso de propriedade sobre o produto que está sendo gerado.

O Product Owner é o responsável por manter a Visão do Produto, gerenciando o Product Backlog de forma a garantir que todo o trabalho realizado pelo Time de Desenvolvimento caminhe em direção a ela.

Leia mais detalhes sobre a Visão do Produto em [24.5 Visão de Produto](#).

7.2.4 Gerencia as entregas do produto

A gestão das entregas do produto ou Releases é uma atribuição do Product Owner. Ele decide qual é a melhor estratégia do projeto para se realizarem essas entregas de Incrementos do Produto e a modifica quando necessário. Ele leva em conta também que os princípios Ágeis pregam as entregas desde cedo e frequentes para possibilitar o *feedback* rápido e, assim, reduzir os riscos do projeto.

A estratégia de Releases é condicionada por questões de negócios e técnicas. Por um lado, as Releases são alinhadas com a estratégia de negócios do produto de forma a contribuir com ela. Ao mesmo tempo, para decidir como e quando serão realizadas, o Product Owner também considera questões técnicas, que são apontadas pelo Time de Desenvolvimento e, em muitos casos, pelos próprios clientes ou usuários.

Existem diferentes possibilidades para a estratégia para a realização das Releases de um projeto. Por exemplo, elas podem ser realizadas quando o

Product Owner julgar adequado, após alguns Sprints, ou ao final de cada Sprint ou em entrega contínua. As Releases também podem ser realizadas diretamente para seus usuários finais, ou para um grupo selecionado desses usuários ou de usuários intermediários, possivelmente internos, dependendo da estratégia do produto.

Leia mais detalhes sobre entregas em [21 Release](#).

Participa do Sprint com o Time de Desenvolvimento

Na reunião de Sprint Planning, o Product Owner colabora e negocia com o Time de Desenvolvimento para decidirem o que será realizado durante o Sprint que se inicia. Ou seja, a partir das prioridades definidas pelo Product Owner e da capacidade de trabalho do Time de Desenvolvimento, eles decidem juntos quantos itens estarão previstos para serem desenvolvidos nesse Sprint, desde o item de maior importância, e qual será a Meta do Sprint que o Time de Desenvolvimento vai se comprometer a realizar a partir do desenvolvimento dos itens selecionados.

Enquanto o Time de Desenvolvimento planeja como desenvolverá esses itens selecionados (em geral, quebrando-os em tarefas), surge um número de questões sobre o Incremento do Produto a ser gerado. Assim, é importante que o Product Owner, caso não esteja presente nessa parte da reunião, ao menos se mostre disponível e acessível para esclarecer essas dúvidas.

Na reunião de Sprint Review, Product Owner e Time de Desenvolvimento, facilitados pelo ScrumMaster, buscam *feedback* de clientes e demais partes interessadas sobre o Incremento do Produto que foi gerado durante o Sprint. Esse *feedback* servirá de matéria-prima para o Product Owner decidir o que será feito em futuros Sprints. A apresentação para clientes e demais partes interessadas é uma colaboração entre Product Owner e Time de Desenvolvimento, e seu formato varia de time para time.

Na reunião de Sprint Retrospective, Product Owner e Time de Desenvolvimento, facilitados pelo ScrumMaster, buscam identificar pontos de melhorias na sua forma de trabalhar, e traçam respectivos planos de ação a serem postos em prática já no Sprint seguinte.

7.2.5 Colabora com o Time de Desenvolvimento durante o Sprint

Durante o Sprint, o Product Owner se coloca disponível e acessível, permitindo assim ao Time de Desenvolvimento que solicite esclarecimentos ou uma decisão rápida sobre algum item do Sprint Backlog sempre que necessário. De outra forma, ele estaria gerando um impedimento ou estimulando o Time de Desenvolvimento a tomar decisões que não lhe cabem sobre o produto.

É também saudável o Product Owner interagir com o Time de Desenvolvimento durante o Sprint para, juntos, refinarem e prepararem itens no alto do Product Backlog para o Sprint seguinte. Enquanto diversos Times de Scrum realizam essa atividade durante a reunião de Sprint Planning, cada vez mais times usam as sessões de Refinamento do Product Backlog, durante o Sprint, para essas atividades (veja [2.3 Refinamento do Product Backlog](#)).

No entanto, caso o Product Owner interfira no planejamento já realizado para o Sprint, isso poderá desviar o foco do Time de Desenvolvimento e, assim, comprometer a realização do trabalho. Por essa razão, o Time de Desenvolvimento, amparado pelo ScrumMaster, rejeita quaisquer alterações no Sprint Backlog que modifiquem, durante o Sprint, a Meta do Sprint já acordada.

Da mesma forma, o Product Owner que busca informações sobre o andamento do trabalho pode ter o intuito de exercer pressão sobre o Time de Desenvolvimento, interferindo em sua auto-organização. Diante de problemas como esses, o ScrumMaster deve trabalhar para ensinar ao Product Owner como ele melhor pode realizar seu trabalho, sempre em colaboração com o Time de Desenvolvimento.

O Time de Desenvolvimento, no entanto, pode buscar o *feedback* do Product Owner no decorrer do Sprint sobre itens em andamento ou já prontos. Isso pode ajudá-lo a melhorar a sua entrega e a realizar a Meta da Sprint.

7.3 COMO É O PRODUCT OWNER?

7.3.1 Único

O papel de Product Owner é exercido por apenas uma pessoa em um Time de Scrum. A existência de mais de um Product Owner interagindo com o Time

de Desenvolvimento geraria dúvidas e conflitos sobre quem tem o poder de decisão sobre o produto em desenvolvimento. Quando houvesse discordâncias, por exemplo, o Time de Desenvolvimento não saberia quem ouvir e, assim, o processo decisório sobre o produto se tornaria desnecessariamente lento e confuso.

De forma similar, a existência de Product Owners substitutos, que compreenderiam quando o Product Owner não pudesse estar presente, confundiria o Time de Desenvolvimento e o próprio trabalho de definição do Produto. Como na situação anterior, as expectativas e interpretações diversas de diferentes Product Owners poderiam tornar as Metas dos Sprints difíceis de serem compreendidas e, afinal, realizadas.

Um revezamento de Product Owners ao longo do projeto também poderia confundir o Time de Desenvolvimento e trazer problemas parecidos.

Ausências do Product Owner por doença ou férias, no entanto, são situações infreqüentes e, nesses casos, dificilmente há alguma solução diferente de simplesmente substituí-lo provisoriamente. Assim, o uso de substitutos dever acontecer apenas para o caso de exceção, ou seja, quando for estritamente necessário.

Não existem também, ao mesmo tempo, um Product Owner do cliente e outro da organização que está gerando o produto, trabalhando como um representante ou como um canal de comunicação. O Product Owner é único: ou ele é uma pessoa escolhida no cliente, ou ele é uma pessoa escolhida na organização.

O Product Owner também não é um comitê ou um departamento da organização. Grupos de pessoas, mesmo que tomem decisões únicas e centralizadas, em geral demoram um tempo demasiadamente longo para chegar a elas. Com o Product Owner único, o Time de Desenvolvimento enxerga apenas um foco para a tomada de decisões sobre o produto e para o esclarecimento de dúvidas.

O Product Owner é a única pessoa responsável por gerenciar o Product Backlog. Suas decisões sobre as necessidades de negócios com relação ao produto devem ser respeitadas, de forma que ninguém além dele pode mudar as prioridades por ele estabelecidas.

Para realizar seu trabalho, no entanto, o Product Owner é aconselhado,

influenciado e conta com o apoio de outras pessoas ou equipes envolvidas no projeto além dos próprios clientes do projeto e o Time de Desenvolvimento. Pode também existir, por exemplo, uma equipe de analistas de negócios que auxilie o Product Owner em seu trabalho.

Em projetos maiores, pode haver a necessidade de existir mais de um Time de Scrum trabalhando no mesmo produto. Nesses casos, dependendo do número de times envolvidos no projeto, apenas um Product Owner dificilmente terá a capacidade e disponibilidade suficientes para fazer parte de todos os Times de Scrum, e realizar todo o trabalho necessário. Ele conseguirá, talvez, participar de dois ou três times.

Nesses casos, o Product Owner continua sendo único para um Time de Scrum: cada Time de Desenvolvimento do projeto colaborará apenas com um Product Owner, que em geral será responsável por uma área definida do produto em desenvolvimento. Além disso, todos os Times de Scrum trabalharão no mesmo Product Backlog. Os diferentes Product Owners do projeto coordenarão seu trabalho e tomarão decisões sobre o produto em conjunto. Esse assunto, no entanto, está fora do escopo deste livro e não será aqui discutido em detalhes.

7.3.2 Disponível para o trabalho no projeto

O trabalho do Product Owner envolve principalmente:

- estar presente na reunião de Sprint Planning. O Product Owner define, junto com o Time de Desenvolvimento, qual a Meta a ser realizada no Sprint e qual o trabalho a ser feito para se realizar essa Meta;
- colocar-se acessível e disponível para tomar decisões e esclarecer dúvidas do Time de Desenvolvimento sobre o produto durante todo o Sprint, sempre que solicitado;
- interagir com o Time de Desenvolvimento para que, juntos, preparem itens para o próximo Sprint;
- estar presente na reunião de Sprint Review para, junto com o Time de Desenvolvimento, obter *feedback* dos clientes do projeto e demais partes interessadas sobre o trabalho realizado no Sprint;

- estar presente na reunião de Sprint Retrospective para ajudar o Time de Scrum a melhorar seu trabalho e a se tornar mais produtivo;
- interagir frequentemente com os clientes e demais partes interessadas ao longo de todo o projeto para entender suas necessidades de negócios, obter seu *feedback* sobre o trabalho já entregue e atualizar o Product Backlog com as novas informações.

Todas essas atividades podem demandar do Product Owner uma dedicação considerável de tempo, embora não necessariamente exclusiva. Assim, um Product Owner ocupado e ausente pode não ter disponibilidade suficiente para realizar seu trabalho, o que poderá trazer consequências desastrosas para o projeto.

Observadas todas essas questões envolvidas em seu trabalho, o Product Owner poderia trabalhar, ao mesmo tempo, em mais de um projeto. Ou seja, ele pode fazer parte de mais de um Time de Scrum. No entanto, dependendo da complexidade do produto e do projeto, essa pode se tornar uma tarefa muito difícil.

7.3.3 Competente para a definição do produto

O Product Owner define qual o produto a ser desenvolvido, Incremento a Incremento. Ele é, de fato, o gerente do produto em um projeto que utiliza Scrum.

Para tomar as decisões necessárias, não se espera que o Product Owner saiba tudo sobre o produto a ser desenvolvido. Afinal, os detalhes de um novo produto são “alvos em movimento”, já que ele se desenvolve a partir do *feedback* frequente dos clientes e das demais partes interessadas.

Também não se espera, inicialmente, que o Product Owner necessariamente tenha um grande conhecimento sobre o negócio dos clientes. Esse é um cenário bastante comum quando o Product Owner é escolhido na organização a desenvolver o produto. Nesses casos, ele deverá construir esse conhecimento aos poucos, ao longo do projeto. Espera-se certamente que o Product Owner possua o conhecimento necessário para tomar decisões suficientemente bem direcionadas, que logo receberão *feedback* e deverão ser ajustadas de acordo, para receberem *feedback* novamente.

Um Product Owner que tome decisões demasiadamente erradas, no entanto, pode ser prejudicial para o projeto, gerando muito desperdício com grandes correções de rumo. O *framework* Scrum busca mitigar esse risco rápida e frequentemente através das reuniões de Sprint Review, onde a qualidade das decisões do Product Owner torna-se visível, além das entregas frequentes.

O Product Owner não é um intermediário para os clientes do projeto, mas sim aquele que de fato define o produto a ser desenvolvido, com o propósito de atender as necessidades desses clientes. Ele assume essa responsabilidade, pois entende que não será capaz de realizar esse trabalho caso simplesmente obedeça às vontades dos clientes, atuando como um tomador de pedidos. Assim, ele deve possuir o poder de tomar decisões que considere as mais adequadas em cada momento.

Quando há alguma dúvida ou questão a ser resolvida, o Product Owner, se considerar necessário, pode consultar os clientes do projeto ou quaisquer outros especialistas relevantes. No entanto, ele deve estar preparado para rapidamente oferecer respostas ao Time de Desenvolvimento, antecipando-se às possíveis dúvidas. Ele deve decidir, em vez de ficar esperando por decisões de outros, de forma a não criar impedimentos para o trabalho do Time de Desenvolvimento, que de outra forma ficaria em espera.

CAPÍTULO 8

ScrumMaster

8.1 QUEM É O SCRUMMASTER?

O ScrumMaster trabalha para facilitar e potencializar o trabalho do Time de Scrum. Ou seja, utilizando-se de seu conhecimento de Scrum, habilidade de lidar com pessoas, técnicas de facilitação e outras técnicas, ele ajuda o Product Owner e Time de Desenvolvimento a serem mais eficientes na realização do seu trabalho.

Para realizar esse trabalho, o ScrumMaster:

- facilita o trabalho do Time de Scrum em seu dia a dia e nos eventos do Scrum, de forma a aumentar a autonomia de seus membros para que juntos desenvolvam o produto, comuniquem-se efetivamente e busquem continuamente melhorar seus processos de trabalho, realizando-o com qualidade e produtividade;

- garante a remoção dos impedimentos que atrapalham o trabalho do Time de Desenvolvimento e ajuda a prevenir que os impedimentos aconteçam, quando possível;
- promove as mudanças organizacionais necessárias para que o Time de Scrum possa realizar seu trabalho com efetividade;
- garante o uso do Scrum, assegurando que o *framework* seja compreendido e adequadamente utilizado pelo Time de Scrum.

O ScrumMaster é:

- competente em *soft skills*, ou seja, possui competências comportamentais e pessoais como comunicação, facilitação e política. Ele é também corajoso, proativo e autoconfiante para realizar as mudanças necessárias, remover impedimentos e proteger o trabalho do Time de Desenvolvimento;
- presente durante o trabalho do Time de Desenvolvimento, para observar, identificar, ajudar a criar visibilidade e a resolver problemas, para remover impedimentos e para proteger o Time de Desenvolvimento de interrupções;
- suficientemente neutro, visando a aumentar a responsabilidade e capacidade do Time de Scrum de resolver seus próprios problemas e chegar a suas próprias soluções.

8.2 O QUE FAZ O SCRUMMASTER?

8.2.1 Facilita o trabalho do Time de Scrum

O ScrumMaster é um facilitador para o trabalho do Time de Scrum, ou seja, dos membros do Time de Desenvolvimento e do Product Owner. Ele promove a autonomia, a relação de trabalho construtiva e a comunicação entre os membros do Time de Scrum no seu dia a dia, de forma a se tornarem cada vez mais efetivos.

Facilitador hábil

Na definição clássica de Roger Schwarz (2002), facilitação de um grupo é: *um processo pelo qual uma pessoa cuja escolha é aceitável para todos os membros do grupo, que é suficientemente neutra e que não possui autoridade considerável no processo decisório do grupo, diagnostica e intervém para ajudar o grupo a melhorar como ele identifica e resolve problemas e toma decisões, para aumentar e efetividade do grupo.* A partir dessa definição, Schwarz criou o modelo do “facilitador hábil”.

Listo a seguir algumas características do trabalho do ScrumMaster, enquanto facilitador, utilizando o modelo criado por Schwarz. Em particular, adoto aqui a abordagem de facilitação desenvolvimentista, na qual o facilitador ensina o time a melhorar seus processos de trabalho e a depender cada vez menos dele.

O facilitador habilita o time a aumentar sua efetividade. A principal atribuição do ScrumMaster enquanto facilitador é ajudar o Time de Scrum a aumentar a sua efetividade. Para tal, ele ajuda a habilitar os membros do Time de Desenvolvimento e o Product Owner a melhorarem:

- **seus processos:** como trabalham juntos, ou seja, como se comunicam, como é o seu processo criativo, como identificam e resolvem problemas, como lidam com conflitos e como se relacionam com seu entorno;
- **sua estrutura:** características relativamente estáveis importantes para o funcionamento do grupo, como a existência de metas claras a serem realizadas, de uma composição do time adequada para a realização de suas tarefas, de tarefas motivadoras, de valores e crenças consistentes com um time efetivo, de normas ou acordos explícitos derivados desses valores e crenças, e de tempo suficiente para realizar seu trabalho e para melhorar sua forma de trabalhar;
- **seu contexto:** como o contexto no qual o time está inserido (por exemplo, sua organização ou setor) influencia a sua efetividade, como a existência de objetivos organizacionais claros, de uma cultura organizacional que apoie o uso do Scrum e de práticas de gestão que tornem o time mais efetivo, da premiação de comportamentos do time (e não do

indivíduo) consistentes com os objetivos do time, de informações necessárias para realizar o trabalho, de *feedback* sobre o seu trabalho, de acesso a treinamento e consultoria que se façam necessários para possibilitar a resolução de problemas, aumentar seus conhecimentos e desenvolver habilidades necessárias para realizar seu trabalho, de acesso ao material e tecnologia necessários e de um ambiente físico adequado.

O ScrumMaster, enquanto facilitador, intervém diretamente no processo, estrutura e contexto do Time de Scrum para ajudá-lo a se tornar mais efetivo. Mas, ao mesmo tempo, o ScrumMaster estimula o Time de Scrum a refletir sobre como pode melhorar seu processo, estrutura e contexto, e ensina-o a desenvolver as habilidades necessárias para atuar neles por si só, pois a responsabilidade de aumentar sua efetividade pertence ao próprio time.

Essas mudanças, no entanto, só podem ocorrer se o Time de Scrum possuir a autoridade necessária para realizá-las e se todos os seus membros compartilharem a responsabilidade por essas mudanças. Embora o Time de Scrum não possua controle direto sobre seu contexto, ele pode influenciá-lo e provocar mudanças que o possibilitem a se tornar mais efetivo.

O facilitador é neutro. O ScrumMaster, enquanto facilitador, é uma pessoa suficientemente neutra, que tem como objetivo aumentar a responsabilidade e capacidade do grupo de resolver seus próprios problemas. Assim, ele não interfere diretamente no conteúdo das discussões do grupo. Para que essa neutralidade seja possível, o ScrumMaster preferencialmente não exerce também o papel de membro do Time de Desenvolvimento ou Product Owner, que têm suas opiniões e interesses e, assim, são invariavelmente parciais. O ScrumMaster, portanto, trabalha para o Time de Scrum como um todo.

O facilitador habilita o time a aumentar sua autonomia. Uma parte importante do trabalho do facilitador é habilitar o time a realizar suas próprias escolhas (e decisões) a partir de informações suficientes: é o que chamamos de escolhas livres e informadas. Ao realizar escolhas livres e informadas, os membros do time naturalmente tornam-se comprometidos com essas escolhas e, ao mesmo tempo, revisam-nas e as mantêm atualizadas. O trabalho do ScrumMaster, enquanto facilitador, é chave para possibilitar a auto-organização do Time de Desenvolvimento.

Um elemento importante para possibilitar as escolhas livres e informadas é a transparência. Para assegurá-la, o ScrumMaster nunca deve fazer arranjos com determinados membros ou com terceiros, seja para ocultar informações do Time de Scrum (ou de alguma parte dele) ou para se comportar de uma determinada forma ou de outra. Ao contrário, o ScrumMaster deve ajudar o Time de Scrum a garantir a visibilidade das informações disponíveis.

A princípio, o time é mais efetivo se é internamente comprometido com suas escolhas. No entanto, mesmo que o ScrumMaster esteja realizando a facilitação de forma efetiva, o time pode fazer escolhas ruins, já que elas são livres, ainda que informadas. E ainda que o ScrumMaster identifique que uma determinada escolha não é apropriada, sua intervenção não é a princípio recomendável.

Seu papel é o de estimular o time a refletir sobre o resultado de suas escolhas e ajustá-las de acordo. Ao intervir com suas próprias opiniões ou, pior, tomar a decisão pelo time, o ScrumMaster estará reduzindo tanto a autonomia do time quanto sua própria credibilidade enquanto facilitador.

Para aumentar a autonomia do Time de Scrum e desenvolver sua efetividade, as intervenções do ScrumMaster, enquanto facilitador, têm sempre o objetivo de reduzir a dependência do Time de Scrum no próprio ScrumMaster. Assim, podemos afirmar que o ScrumMaster trabalha em direção a se tornar cada vez menos necessário, até mesmo habilitando os membros do Time de Scrum a atuarem como facilitadores eles mesmos.

O facilitador não é intermediário ou representante. O ScrumMaster também não atua como um intermediário entre membros do Time de Scrum, ou entre o Time de Scrum e pessoas externas a ele. Ao contrário, ele estimula o time a desenvolver as habilidades de se comunicar e lidar diretamente com quem se fizer necessário.

O ScrumMaster também não atua como representante do Time de Scrum ou de qualquer de seus membros diante de outros membros ou de pessoas externas. Dessa forma, não é o ScrumMaster, por exemplo, que reporta ao resto da organização os resultados do trabalho ou o desempenho do Time de Scrum ou de algum de seus membros.

Ele não tem autoridade para utilizar qualquer informação obtida a partir da facilitação para influenciar decisões externas sobre membros do time

como, por exemplo, bonificações ou punições. Caso o fizesse, a confiança do Time de Scrum no ScrumMaster ficaria comprometida, assim como sua neutralidade e, consequentemente, sua eficiência como facilitador. Esse tipo de informação, caso necessária, é fornecida diretamente pelo Time de Scrum.

Da mesma forma, o ScrumMaster não atua como intermediário entre os membros do Time de Desenvolvimento e Product Owner ou entre diferentes membros do Time de Desenvolvimento. Ele, ao contrário, estimula que se comuniquem diretamente.

O facilitador é um especialista no processo. Como um especialista no processo, o ScrumMaster sabe quais elementos mais contribuem para tornar o Time de Scrum mais eficiente. Uma parte importante desses elementos são as regras do Scrum, que devem ser corretamente compreendidas e utilizadas pelo Time de Scrum. Assim, o ScrumMaster identifica quaisquer comportamentos disfuncionais do time e desvios no uso do Scrum utilizando sua expertise no processo.

Embora não interfira diretamente no conteúdo das discussões do grupo, quando uma discussão envolve processos do time ou organizacionais, o ScrumMaster pode nesse momento deixar sua neutralidade de lado — justamente por ser um especialista no processo — e, então, interferir nesse conteúdo.

Facilitador de eventos

Michael Wilkinson (2004), no livro *The secrets of facilitation: the S.M.A.R.T. guide for getting results with groups*, define uma sessão facilitada como: *um encontro altamente estruturado no qual o facilitador guia os participantes por meio de uma série de passos predefinidos para que cheguem a um resultado que é criado, compreendido e aceito por todos os participantes*. Ainda segundo o autor, técnicas de facilitação são apropriadas sempre que um grupo necessita de compreensão comum e aceitação para obter sucesso em se chegar a um determinado resultado.

Além de facilitar o dia a dia de trabalho do Time de Scrum, o ScrumMaster tem a importante função de atuar como um facilitador nos eventos do Scrum. Ele estimula os envolvidos a participarem ativamente das discussões, ajuda-os a manter o foco nos objetivos do evento e a chegarem a suas próprias

conclusões.

Podemos interpretar qualquer um dos eventos do Scrum como uma sessão facilitada, que possui um caminho específico a se percorrer e um resultado específico a ser alcançado. O resultado esperado, por exemplo, da reunião de Sprint Planning é um plano para o Sprint, ou seja, a Meta do Sprint e o Sprint Backlog. O resultado esperado da reunião de Sprint Review é o *feedback* dos clientes e demais partes interessadas sobre o Incremento do Produto gerado pelo Time de Desenvolvimento durante o Sprint.

As características de um facilitador de eventos descrito por Wilkinson são bastante similares às do “facilitador hábil”, descrito anteriormente. De acordo com o autor, o facilitador de eventos não interfere no conteúdo das discussões, nem oferece soluções ou suas próprias opiniões. Ele, na realidade, serve ao grupo guiando os participantes por meio de passos predefinidos para que alcancem resultados específicos, utilizando seu conhecimento de dinâmicas de grupo e dos passos do processo.

O facilitador cria um ambiente em que os participantes se sentem à vontade para contribuir. Em geral, ele responde a questionamentos com perguntas e sabe realizar as perguntas certas, que estimularão os participantes a chegarem a suas próprias conclusões.

O facilitador procura prevenir comportamentos disfuncionais que possam ocorrer durante a sessão, ajudando o grupo, por exemplo, a estabelecer suas regras básicas de conduta. Ele reconhece e lida com comportamentos disfuncionais quando ocorrem, buscando soluções a partir de suas causas. O facilitador também ajuda o grupo a manter seu foco nos resultados da reunião e a construir um consenso em torno das soluções geradas.

Ao final, os resultados da sessão facilitada devem ter sido criados, compreendidos e aceitos por todos os participantes.

Liderança no time auto-organizado

Times auto-organizados são usados em diferentes áreas de trabalho, e não apenas no desenvolvimento de *software*. Embora possa parecer paradoxal, a prática mostra que times auto-organizados muitas vezes possuem líderes. Sua função primária, no entanto, é a de facilitar a auto-organização, e não a de gerenciar a equipe. O líder de um time auto-organizado, portanto, tem o

importante papel de *lidar o time em direção a que o time lidere a si mesmo*.

O ScrumMaster possui muitas das características de um líder de time auto-organizado. Esse tipo de líder facilita a auto-organização principalmente ao encorajar o time à auto-observação e autoavaliação, de forma que o time monitore, torne consciente e avalie seu desempenho; e do encorajamento do time ao autorreforço, de forma que seus membros demonstrem entre si que valorizam um bom trabalho feito por um colega e valorizam o alto desempenho do time.

Essas ações de encorajamento são mais efetivas quando o líder consegue criar uma relação de confiança e de atenção com os membros do time. Isso facilita o levantamento de informações que determinem os tipos de ações que devem ser encorajadas.

Para serem efetivos, líderes de times auto-organizados realizam tanto funções voltadas internamente quanto voltadas externamente ao time, atuando na organização que o contém e atravessando constantemente a fronteira entre os dois lados. O líder efetivo é capaz de construir relacionamentos tanto com membros do time quanto com outros indivíduos, ou unidades da organização que podem afetar o desempenho do time (como a alta gerência, a manutenção e outros grupos) e de buscar entender ambas as perspectivas.

Nesse trabalho, o líder de um time auto-organizado não tem poder nem dá ordens: ele na verdade obtém acesso a informações-chave de ambos os contextos por meio do bom relacionamento, da percepção política e da habilidade em realizar perguntas. Ele utiliza essas informações-chave para alinhar as necessidades do time com as da organização e vice-versa, de forma a fazer com que tanto o time quanto a organização se comportem de modo a facilitar a efetividade de ambos.

Esse trabalho envolve, por exemplo, evitar que uma decisão tomada pelo time, que de fato cabe ao time, seja rejeitada pela organização. Esse trabalho também permite ao líder promover o poder de ação e de decisão do time, o que o leva a assumir uma maior responsabilidade por seus resultados.

As informações desta seção foram baseadas nos trabalhos de Cummings (1978), Manz & Sims (1987) e Druskat & Wheeler (2003).

8.2.2 Garante a remoção de impedimentos

Impedimentos ameaçam o cumprimento da Meta do Sprint, pois dificultam significativamente ou obstruem o trabalho do Time de Desenvolvimento. O ScrumMaster é o responsável pela resolução desses impedimentos, removendo-os ele mesmo ou mobilizando as pessoas e os recursos necessários para tal.

Um impedimento tipicamente acontece em um trabalho que já foi iniciado pelo Time de Desenvolvimento e, assim, gera uma espera sobre um item ou itens que têm prioridade de desenvolvimento naquele momento. Por essa razão, uma vez identificado pelo Time de Desenvolvimento, o impedimento é imediatamente sinalizado para o ScrumMaster, que por sua vez toma ações rápidas e efetivas para a sua remoção. É importante que as soluções adotadas atuem diretamente sobre as possíveis causas raízes em vez de se utilizarem soluções paliativas.

O ScrumMaster estimula o Time de Desenvolvimento a identificar se há impedimentos por vir. Assim, quando o Time de Desenvolvimento se antecipa a um impedimento, o ScrumMaster trabalha imediatamente para ajudar a evitar que ele aconteça.

No entanto, deve-se saber distinguir problemas e questões do dia a dia de impedimentos no trabalho, sobre os quais atua o ScrumMaster. A resolução de problemas está ao alcance e no contexto de atuação do Time de Desenvolvimento. Por essa razão, são tratados pelo próprio Time de Desenvolvimento, ainda que com a ajuda do ScrumMaster.

Natureza dos impedimentos e sua resolução

De acordo com sua natureza, os impedimentos podem ser classificados em organizacionais, básicos, administrativos ou de nível de serviço (PIMENTEL, 2009):

- **organizacionais** — ocorrem quando a própria estrutura ou funcionamento da organização os impõe ao Time de Desenvolvimento. Podem acontecer em pelo menos três situações distintas:
 - o Time de Desenvolvimento necessita, mas não obtém a ajuda ou

intervenção de outra pessoa, equipe ou área da organização. Nesses casos, o ScrumMaster usa o acesso e influência que construiu e está construindo na organização para buscar, o mais rapidamente possível, o apoio necessário para o Time de Desenvolvimento realizar seu trabalho e monitorar se esse apoio foi realmente obtido. Esse tipo de impedimento se configura, por exemplo, quando o equipamento de trabalho de um membro do Time de Desenvolvimento apresenta defeito, mas a equipe de suporte não resolve o problema em tempo hábil;

- outra pessoa, equipe ou área da organização intervém no trabalho do Time de Desenvolvimento e demanda sua ajuda ou de algum de seus membros, assim desviando seu foco da Meta do Sprint. O ScrumMaster tem o importante papel de proteger o Time de Desenvolvimento de interferências externas que coloquem em risco a Meta do Sprint. De forma geral, o modo mais efetivo de se realizar essa proteção em ambientes hierárquicos é utilizando política e o ensino do Scrum e de seus benefícios.

Esse tipo de impedimento ocorre, por exemplo, quando um membro da organização em uma posição hierarquicamente superior aborda diretamente um membro do Time de Desenvolvimento para realizar alguma tarefa externa ao projeto;

- questões de cunho político ou burocrático cultivadas pela organização obstruem o trabalho do Time de Desenvolvimento. Como um agente de mudanças organizacional, o ScrumMaster trabalha para modificar as práticas organizacionais em benefício da eficiência do Time de Desenvolvimento em seu trabalho. Ele, por exemplo, pode se unir a ScrumMasters de outros times para realizar as mudanças organizacionais necessárias.

A adoção de métodos ou práticas pesadas que conflitem com os valores e princípios Ágeis, como as que geram uma produção excessiva de documentação, constitui um exemplo desse tipo de impedimento;

- **básicos** — são impedimentos que ocorrem pela falta de um ou mais

elementos essenciais à realização do trabalho em algum determinado momento. O ScrumMaster trabalha para garantir que o Time de Desenvolvimento possua acesso aos meios necessários para realizar suas tarefas.

Esse tipo de impedimento inclui, por exemplo, a falta de ferramentas ou equipamentos necessários para a realização do trabalho, a falta de um ambiente adequado ou de elementos essenciais nesse ambiente (como cadeiras ou mesas, por exemplo), e até mesmo a impossibilidade de acesso a dados ou informações necessárias;

- **administrativos** — são impedimentos provenientes de ocorrências administrativas, como absenteísmo (atrasos, licenças, folgas, faltas por motivos diversos etc.), demissões e restrições de horários de trabalho. Essas questões estão muitas vezes ligadas a áreas em que atua o Scrum-Master, como políticas organizacionais ou problemas intraequipe que podem estar gerando desmotivação;
- **nível de serviço** — aqui são identificados impedimentos oriundos de problemas na sustentação de algum serviço em operação. O Scrum-Master trabalha para criar visibilidade para o problema e monitora sua resolução junto à equipe que tem a atribuição de resolvê-lo, sendo muitas vezes necessário estimular o envolvimento do Time de Desenvolvimento nesse trabalho.

Exemplos desse tipo de impedimento incluem, para projetos de desenvolvimento de *software*, os erros em ambientes de produção e problemas com o servidor de aplicações, com o servidor de integração ou com ferramentas diversas de apoio.

8.2.3 Promove as mudanças organizacionais necessárias

O ScrumMaster atua como um agente de mudanças na organização onde está inserido o Time de Scrum. Ou seja, ele trabalha para promover as mudanças no contexto do Time de Scrum necessárias para que a equipe possa realizar seu trabalho com mais efetividade.

Essas mudanças podem envolver, por exemplo, desde educar em Scrum pessoas de quem o Time de Scrum depende para realizar seu trabalho, de forma que possam colaborar em vez de gerar impedimentos, até trabalhar para modificar regulamentos que gerem desperdícios ou estruturas ineficientes que atrapalhem o trabalho do Time de Scrum.

Ao identificar um impedimento ao trabalho do Time de Desenvolvimento, o ScrumMaster vai em busca de sua causa raiz. Quando esse impedimento tem origem na burocracia ou na estrutura da própria organização, o ScrumMaster atuará como um agente de mudança para promover as mudanças necessárias e, assim, evitar que o impedimento ocorra novamente.

Um ScrumMaster efetivo é persistente e perseverante, e não aceita que não consiga modificar a organização em que trabalha. Por mais difícil e lento que esse trabalho possa parecer em algumas organizações, realizar as mudanças necessárias é uma parte essencial das atribuições do ScrumMaster.

O ScrumMaster é capaz transitar entre o Time de Scrum e a organização que o contém, construindo relacionamentos com pessoas-chave ou unidades da organização que podem afetar o desempenho do Time de Scrum. A partir do bom relacionamento, da percepção política e da habilidade em realizar perguntas, ele obtém acesso a informações-chave e as utiliza para alinhar as necessidades e objetivos do Time de Scrum e os da organização que o contém e vice-versa (veja *Liderança no time auto-organizado* em [8.2 O que faz o ScrumMaster?](#)).

8.2.4 Garante o uso do Scrum

O ScrumMaster é responsável por garantir que os valores, práticas e regras do Scrum estejam sendo entendidos e seguidos por todo o Time de Scrum. Ele ensina o Scrum para o Time de Desenvolvimento, para o Product Owner e para demais pessoas da organização que julgue necessário, de forma que apoiem o trabalho do Time de Scrum. O ScrumMaster fica atento a desvios na prática do Scrum, e toma medidas que considere necessárias para estimular o Time a corrigir esses desvios.

É responsabilidade do ScrumMaster que todos os eventos do Scrum aconteçam, e que os horários e *timeboxes* sejam respeitados. É também de sua

responsabilidade que o Time de Desenvolvimento esteja atualizando o Sprint Backlog e que esteja monitorando seu progresso em direção à Meta do Sprint.

O ScrumMaster também se assegura de que o Product Owner esteja atualizando e preparando o Product Backlog e que, para fazê-lo, esteja interagindo com os clientes do projeto e com o Time de Desenvolvimento, de forma que se chegue à Sprint Planning com os itens mais prioritários preparados para entrarem em desenvolvimento.

Um bom ScrumMaster, no entanto, não age como polícia ou fiscal dos processos, forçando os envolvidos a usarem o Scrum corretamente, ou mesmo realizando partes do trabalho por eles para que o bom uso do Scrum aconteça. Na realidade, como descrito anteriormente, um dos importantes objetivos do ScrumMaster é tornar-se cada vez menos necessário, aumentando a autonomia do Time de Scrum.

Com esse propósito, o ScrumMaster ensina e habilita o Time de Desenvolvimento e Product Owner a utilizarem o Scrum, de forma que cada vez menos dependam dele para que os valores, práticas e regras do Scrum sejam seguidos e usados.

Como um exemplo de disfunção, vejo muitas vezes ScrumMasters avisando todos os dias ao Time de Desenvolvimento quando é hora de realizar a sua reunião diária. Ao agir dessa forma, no entanto, em vez de habilitá-lo e torná-lo independente, o ScrumMaster pode estar levando o Time de Desenvolvimento a sempre depender dele para avisá-lo da reunião, o que provavelmente acontecerá até o final do projeto.

O ScrumMaster fará melhor seu trabalho se, utilizando diferentes técnicas, for capaz de mostrar e convencer o Time de Desenvolvimento da importância de manter a disciplina da realização da reunião, de forma que crie o hábito e se auto-organize para fazê-lo.

8.3 COMO É O SCRUMMASTER?

8.3.1 Competente em soft skills

Para desempenhar bem o papel de ScrumMaster, o indivíduo deve possuir competências comportamentais e pessoais que lhe permitam realizar o seu trabalho. Essas competências são chamadas de *soft skills*.

De forma geral, um ScrumMaster tímido ou reservado encontra dificuldades para desempenhar o seu papel. A facilitação do trabalho do Time de Scrum, parte essencial do trabalho do ScrumMaster, inclui promover a detecção e visibilidade dos problemas e assegurar a boa relação entre os membros do Time de Desenvolvimento, e entre esses e o Product Owner.

O ScrumMaster utiliza-se de boa comunicação e de voz ativa, mas sem comandar, para realizar esse trabalho. Ele também se usa de suas habilidades de negociação e de política para estimular a resolução de conflitos e promover a tomada de decisão do Time de Scrum em direção ao consenso.

Um ScrumMaster autoconfiante, corajoso e proativo atua como um agente de mudanças na organização e, utilizando-se de persistência e perseverança, promove as mudanças necessárias para que o Time de Scrum possa entregar valor para seus clientes. O ScrumMaster faz uso dessas mesmas habilidades para se articular em diferentes níveis organizacionais com o propósito de realizar a remoção de impedimentos e de defender o Time de Desenvolvimento de interferências externas e de mudanças que possam afetar a Meta do Sprint.

A seguir, podemos ver exemplos de *soft skills* desejáveis em um ScrumMaster:

- habilidade de comunicação;
- saber ouvir;
- flexibilidade e adaptabilidade;
- capacidade de negociação e habilidade política;
- capacidade de resolução de problemas e pensamento crítico;
- habilidade de ensinar, por meio de *coaching* ou *mentoring*;
- habilidade de facilitação;
- ser colaborativo;
- autoconfiança;

- autoconsciência;
- paciência;
- persistência e perseverança;
- sociabilidade;
- gestão de seu tempo;
- atitude positiva.

Discussão: o ScrumMaster necessita de conhecimento técnico?

As habilidades necessárias para o trabalho técnico — mas especificamente, na área de desenvolvimento de *software* — e as habilidades necessárias para o trabalho do ScrumMaster são muito diferentes. Concordo que ter conhecimento técnico pode ajudar o ScrumMaster a se inserir mais rapidamente no contexto de trabalho do time, além de facilitar sua leitura de cada situação. Mas ele não realmente necessita desse conhecimento para realizar seu trabalho.

O que vejo na prática é que um bom ScrumMaster consegue se inserir em seu trabalho mesmo sem esse conhecimento. Talvez isso não ocorra tão rapidamente, mas acredito que isso não traga prejuízos relevantes.

Um problema comum com o ScrumMaster técnico, que queremos evitar, é ele buscar usar esse conhecimento para interferir nas discussões (e, portanto, no conteúdo) do trabalho do Time de Desenvolvimento, prejudicando a autonomia do time. Tendo um *background* técnico forte, essa foi uma questão com a qual tive que tomar muito cuidado no meu início de carreira como ScrumMaster.

8.3.2 Presente

O ScrumMaster presente toma ciência imediatamente de obstáculos que estejam impedindo o trabalho do Time de Desenvolvimento e pode rapidamente tomar ações para removê-los, antes que ameacem a Meta do Sprint. O ScrumMaster atento a desvios na prática do Scrum pode agir para corrigi-los, ensi-

nando e habilitando o Time de Desenvolvimento e o Product Owner a usarem o Scrum corretamente.

O ScrumMaster que acompanha o trabalho do Time de Scrum é capaz de detectar comportamentos disfuncionais, conflitos ou problemas de relacionamento entre os membros do Time de Desenvolvimento ou entre o Time de Desenvolvimento e o Product Owner, e atuar o mais rápido possível para ajudar a resolver esses problemas. Ele, presente, pode proteger o Time de Desenvolvimento de interferências externas que ameacem a Meta do Sprint, o que pode acontecer em qualquer momento. Além disso, ele atua como um facilitador nos eventos do Scrum, e não há como fazê-lo sem estar neles presente.

Em suma, o ScrumMaster idealmente se faz presente sempre que o Time de Scrum necessitar dele.

Para muitos times, o ScrumMaster estar presente sempre que necessário pode significar um trabalho em tempo integral. Nesses casos, embora seja comum o uso de ScrumMasters em tempo parcial ou trabalhando em mais de um projeto ao mesmo tempo, um ScrumMaster ausente pode trazer sérios problemas para o projeto, pois ele será necessário em momentos em que não poderá estar presente. Em vez de economizar dinheiro, a organização estará desperdiçando a oportunidade de ter alguém especializado em potencializar o trabalho do Time de Scrum.

De forma geral, o ScrumMaster em tempo integral é importante para: times que trabalham juntos há pouco tempo, times que estão começando a utilizar Scrum, times jovens, times inseridos em organizações hierárquicas e burocráticas e para times disfuncionais, ou seja, aqueles que enfrentam problemas graves na forma como operam.

8.3.3 Suficientemente neutro

O ScrumMaster é um facilitador e, como tal, atua de forma mais neutra possível com relação ao conteúdo das discussões do Time de Scrum. Conforme descrito anteriormente, ele tem como objetivo aumentar a responsabilidade e capacidade do Time de Scrum de resolver seus próprios problemas e chegar às suas próprias conclusões. Com esse objetivo, ele evita interferir diretamente nesse conteúdo.

O ScrumMaster trabalha para o Time de Scrum como um todo, ou seja, para os membros do Time de Desenvolvimento e para o Product Owner. É um erro comum, no entanto, o ScrumMaster manter seu foco apenas no Time de Desenvolvimento, deixando assim o Product Owner desassistido em termos de facilitação. Ao agir dessa forma, o ScrumMaster assume incorretamente uma postura parcial diante do Product Owner e do resto da organização, enquanto alguém que trabalha apenas em prol do Time de Desenvolvimento.

Discussão: o ScrumMaster também pode ser...?

Vejo muitas adoções de Scrum onde o ScrumMaster trabalha parte de seu tempo como desenvolvedor, ou seja, como membro do Time de Desenvolvimento, e possui conhecimento técnico necessário para tal. Embora essa divisão de trabalho não seja proibida, a atuação dupla pode sacrificar a neutralidade do ScrumMaster e, assim, seu trabalho como facilitador.

Não é incomum que as sugestões ou decisões desse ScrumMaster acabem por ter, para os outros membros do time, um peso maior do que suas próprias opiniões e que, assim, invariavelmente terminam por acatá-las. Esse tipo de situação pode ficar mais evidente em momentos de crise, embora não unicamente.

Quando esse cenário ocorre, o ScrumMaster está fazendo o oposto do que deveria ser seu trabalho: habilitar o Time de Desenvolvimento a aumentar sua autonomia. Além disso, principalmente devido à falta de tempo e diferenças no tipo de trabalho, observa-se que as atividades de remoção de impedimentos, facilitação e atuação como agente de mudanças organizacionais podem ser prejudicadas quando o ScrumMaster compartilha de uma atuação técnica.

Mesmo assim, esse muitas vezes pode ser o ponto de partida mais viável para diversos contextos. Ao não entender bem o papel, a gerência não pode aceitar a contratação de um ScrumMaster em tempo integral, e acabar empurrando esse papel para um membro do Time de Desenvolvimento.

Se você estiver nessa situação, não recomendo de forma alguma recusar a adoção de Scrum. Entenda bem seu papel, identifique as limitações que a jornada dupla está produzindo e, enquanto agente de mudança, trabalhe dentro da organização para pouco a pouco viabilizar o seu trabalho e de outros ScrumMasters em tempo integral.

No entanto, recomendo fortemente que o ScrumMaster nunca seja o Product Owner do projeto. Essa é uma recomendação comum entre especialistas em Scrum e constava como regra na definição original do *framework*.

Caso os papéis de ScrumMaster e Product Owner se concentrem em um único indivíduo, o conflito de interesses fica exacerbado. Não há ninguém para facilitar a relação entre o Product Owner e o Time de Desenvolvimento. Não há ninguém para defender o Time de Desenvolvimento se, disfuncionalmente, o próprio Product Owner buscar interferir em seu trabalho durante o Sprint.

Nesses casos, a concentração de poder pode deixar esse indivíduo com características próximas às de um Gerente de Projetos tradicional, exercendo comando e controle sobre o Time de Desenvolvimento. Em suma: não faça isso!

CAPÍTULO 9

Onde está o Gerente de Projetos?

Com Scrum, o trabalho de gestão do projeto é distribuído pelos seus três papéis, ou seja, pelos membros do Time de Scrum. Assim, não há um Gerente de Projetos como é tradicionalmente definido.

De forma sintética, identifico na tabela a seguir alguns pontos-chave cujo gerenciamento cabe a diferentes papéis do Scrum.

Ponto-chave a ser gerenciado	Product Owner	Time de Desenvolvimento	ScrumMaster
Retorno sobre o investimento	X		
Necessidades/objetivos de negócios	X		
Clientes e demais partes interessadas	X		
Visão do Produto	X		
Releases	X		
Tarefas de desenvolvimento do produto		X	
Qualidade interna do produto		X	
Qualidade externa do produto	X	X	
Estimativas ou previsões		X	
Processos (funcionamento do Scrum)			X
Impedimentos no trabalho			X
Relacionamento e motivação do Time		X	X
Riscos	X	X	X
Comunicação	X	X	X

Essa divisão, na prática, significa que questões como escopo, prazo, qualidade, risco, processos e gestão do trabalho de desenvolvimento do produto não ficam sob a responsabilidade de um único papel centralizador. Ao contrário, elas são distribuídas entre o Product Owner, o Time de Desenvolvimento e o ScrumMaster. Cada um em suas funções, eles trabalham de forma colaborativa, e estão igualmente comprometidos com a satisfação das necessidades dos clientes do projeto.

À gestão do projeto, o *framework* Scrum adiciona ainda as tarefas de facilitação (ao papel do ScrumMaster) e de definição do produto (ao papel do Product Owner), que em geral não existem na definição do papel do Gerente de Projetos. Na prática de projetos tradicionais, em geral, a questão da facilitação não é tratada e a responsabilidade da definição do produto é passada para os clientes. Assim, não se deve e não é possível mapear o papel do Gerente de Projetos tradicional para o papel do ScrumMaster ou do Product Owner.

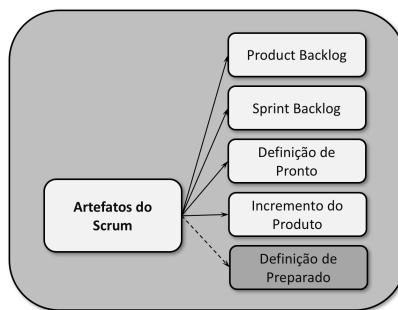
As atribuições de um gerente funcional, que é responsável por formação de equipes, contratações, promoções, demissões, férias, planos de carreira etc., são muito específicas ao contexto das organizações e, assim, observam-se diferentes soluções para cada uma delas.

Por essa razão, essas questões não são tratadas pelo *framework* Scrum. Outras funções gerenciais como geração de contratos, precificação, gestão de portfólio etc. também estão fora do escopo do Scrum.

Parte III

Artefatos do Scrum

O Scrum originalmente define o uso de apenas quatro artefatos: o Product Backlog, o Sprint Backlog, a Definição de Pronto e o Incremento no Produto. Adiciono à lista a Definição de Preparado, artefato que pode ser importante para o trabalho de um Time de Scrum.



CAPÍTULO 10

Product Backlog

10.1 O QUE É O PRODUCT BACKLOG?

O Product Backlog é uma lista ordenada ou priorizada de itens sobre os quais o Time de Desenvolvimento trabalhará no decorrer do projeto, desde os itens do topo da lista, buscando realizar os objetivos do produto, comumente representados por uma Visão de Produto. Ele é atualizado, reordenado e refinado de acordo o nível de detalhes que é possível de se ter em cada momento do projeto.

O Product Backlog é gerenciado pelo Product Owner, e contém o que ele define o que será desenvolvido pelo Time de Desenvolvimento para o produto. Esses itens de trabalho são expressos na forma de necessidades do usuário, objetivos de negócios dos clientes e demais partes interessadas ou funcionalidades do produto. O Product Backlog pode também conter melhorias a serem realizadas no produto, correções de problemas, questões técnicas, pes-

quisas que forem necessárias etc.

Os itens do Product Backlog são organizados pelo Product Owner de acordo com a ordem em que serão desenvolvidos pelo Time de Desenvolvimento, de forma a maximizar o retorno ao investimento dos clientes. Assim, os itens do topo do Product Backlog são colocados em desenvolvimento primeiro.

Por essa razão, esses itens do topo são de granularidade mais fina e possuem uma descrição detalhada e, possivelmente, uma estimativa com maior precisão do esforço necessário para seu desenvolvimento. Na outra ponta, os itens da parte de baixo do Product Backlog são de granularidade mais grossa, e possuem menos detalhes e estimativas mais grosseiras.

O Product Backlog está em constante evolução e, assim, nunca está terminado ou completo. Conforme o produto evolui, o Product Backlog é frequentemente modificado com a adição, subtração, reordenamento e modificação de seus itens. O produto evolui à medida que o ambiente muda, e à medida que tanto clientes quanto Time de Desenvolvimento vão conhecendo e entendendo melhor esse produto que está sendo construído.

O Product Owner é o único responsável por gerenciar o Product Backlog. Ou seja, ele é responsável por criar, atualizar, reordenar e dar visibilidade a ele. Outros contribuem e influenciam as opiniões do Product Owner, mas ninguém além dele pode modificar o Product Backlog ou, ao menos, não sem o seu conhecimento e aprovação. Para realizar esse trabalho, o Product Owner interage frequentemente com os clientes do projeto, com as demais partes interessadas e com o Time de Desenvolvimento.

O Scrum não prescreve nenhum formato ou padrão para o Product Backlog. O importante é que ele tenha o formato de itens em uma sequência, de forma que jamais haja itens com a mesma prioridade e seja facilmente reordenável. Existem softwares especializados que permitem essa manipulação do Product Backlog, mas uma planilha ou até mesmo notas adesivas em um quadro branco podem funcionar muito bem (veja a figura 10.1).

<input type="checkbox"/> Lorem ipsum dolor sit amet consectetur	3
<input type="checkbox"/> Adipiscing elit duis molestie nibh quis ultricies	5
<input type="checkbox"/> Adipiscing sem odio eleifend lectus ut luctus	2
<input type="checkbox"/> Augue sapien mattis	Loreum ipsum dolor sit amet consecteturetra
<input type="checkbox"/> Mollis elementum aliquet eget fringilla temp	Adipiscing elit duis molestie nibh quis ultricies
<input type="checkbox"/> Erat vestibulum consequat purus in ante	Adipiscing sem odio eleifend lectus ut luctus
<input type="checkbox"/> Eleifend id pretium tellus sagittis vestibulum	Augue sapien mattis ante proin massa est
<input type="checkbox"/> Iaculis lacinia neque condimentum semper	Mollis elementum aliquet eget fringilla temp
<input type="checkbox"/> Quam imperdier a proin pulvinar odio eu	Erat vestibulum consequat purus in ante
<input type="checkbox"/> Fringilla varius ante nunc faucibus odio	Eleifend id pretium tellus sagittis vestibulum
<input type="checkbox"/> Euismod tempor erat purus at purus aliquam	Iaculis lacinia neque condimentum semper
<input type="checkbox"/> A lectus arcu non feugiat ipsum cras dapibus	Quam imperdier a proin pulvinar odio eu
	Fringilla varius ante nunc faucibus odio
	Euismod tempor erat purus at purus aliquam
	A lectus arcu non feugiat ipsum cras dapibus

PRODUCT BACKLOG

Fig. 10.1: Três exemplos de formatos de Product Backlog

Discussão: Bug tracking?

O Product Backlog é a única fonte de trabalho a ser realizado no produto pelo Time de Desenvolvimento.

Esse trabalho também inclui as correções de erros (ou *bugs*) encontrados no sistema. Ou seja, erros encontrados no sistema em funcionalidades já desenvolvidas deverão dar origem a itens de correção no Product Backlog, adicionamos na ordem de sua importância.

Para o desenvolvimento de *software*, o uso de ferramentas de *bug tracking*, portanto, se sobreporia ao uso do Product Backlog. Logo, não é recomendado.

10.2 COMO É O PRODUCT BACKLOG?

O Product Backlog é ordenado, planejável, emergente e gradualmente detalhado.

10.2.1 Ordenado

Os itens do Product Backlog são ordenados ou priorizados de acordo com o grau de importância de seu desenvolvimento. Ou, colocando de outra forma, de acordo com o grau de importância de sua entrega pelo Time de Desenvolvimento. Esse ordenamento ou priorização tem o propósito de satisfazer os clientes do projeto. Em outras palavras, garantir e maximizar o retorno sobre o investimento realizado por eles no projeto.

Na reunião de Sprint Planning, o Product Owner e o Time de Desenvolvimento selecionam um número de itens do topo do Product Backlog para serem colocados em desenvolvimento. Como vemos na figura 10.2, os itens seguintes a estes poderão estar presentes ainda na entrega ou Release atual, mas em futuros Sprints. Os itens mais abaixo no Product Backlog poderão estar presentes em entregas ou Releases futuras.

Quanto mais alto no Product Backlog o item estiver, mais chances de ser desenvolvido ele terá, e mais cedo isso poderá acontecer. Por outro lado, quanto mais baixo no Product Backlog o item estiver, menor a sua ordem de desenvolvimento e, assim, menores são suas chances de ser realizado, já que mudanças podem fazê-lo perder o sentido ou ser significativamente modificado.

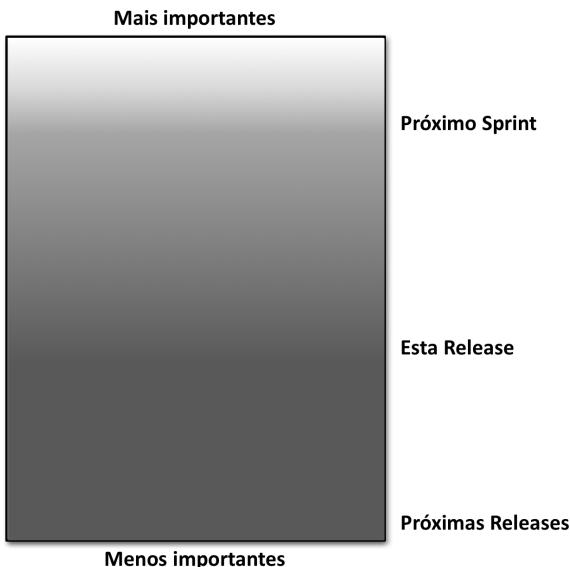


Fig. 10.2: O Product Backlog é ordenado

A ordenação dos itens do Product Backlog é um trabalho contínuo, que acontece ao longo de todo o projeto. Para maximizar o retorno sobre o investimento, o Product Owner considera o valor de negócios e o custo de desenvolvimento de cada item, as interdependências entre os itens, os riscos associados a cada item, questões técnicas fornecidas pelo Time de Desenvolvimento, a estratégia de negócios dos clientes e a estratégia de negócios da organização, entre outros fatores.

O trabalho de ordenação não é simples e requer conhecimento do negócio, das oportunidades e riscos do mercado, da concorrência, da capacidade de entrega do Time de Desenvolvimento e de uma série de outros fatores. O Product Owner busca todo o conhecimento e a assistência necessários para realizar esse trabalho.

A melhor e menos complexa forma do Product Owner maximizar o retorno dos clientes do projeto é manter o foco em entregar para esses clientes as funcionalidades que eles mais necessitam em cada momento do projeto, e

obter *feedback* o mais cedo possível sobre essas funcionalidades, para assim realizar as mudanças necessárias. Ao entregar o que os clientes mais precisam, espera-se que os clientes ou usuários usem o que foi entregue, aumentando as chances de se obter o *feedback* necessário.

RETORNO SOBRE O INVESTIMENTO

Retorno sobre o investimento ou ROI (de *Return On Investment*, em inglês) é uma medida de desempenho de um investimento, em geral calculado a partir da relação entre o valor ganho ou perdido e o valor investido. Caso se obtenha um ganho de X ao investir Y, o ROI correspondente será de $(X - Y) / Y$.

Ao trabalharmos com Scrum, no entanto, não fazemos contas matemáticas para calcular o ROI geralmente. Entretanto, utilizamos um conceito mais amplo e subjetivo do termo, de difícil mensuração numérica, em que o custo é baseado em estimativas realizadas pelo Time de Desenvolvimento para o esforço necessário para gerar o valor.

O retorno sobre o investimento em projetos em que o Scrum é usado acontece sempre que são entregues aos clientes funcionalidades que lhes signifiquem valor de negócio, em geral quando chegam nas mãos de seus usuários.

É importante que o Product Owner defina uma estratégia para o desenvolvimento do produto focada em objetivos de negócios, que evoluí de acordo com as informações que estiverem disponíveis em cada momento. Dessa forma, o Product Owner considera o Product Backlog como um todo e em longo prazo, e não apenas seus itens individualmente.

10.2.2 Planejável

Um Product Backlog planejável permite que Time de Desenvolvimento e o Product Owner sejam capazes de, a partir das informações contidas no Product Backlog, realizar o planejamento do desenvolvimento de uma parte relevante dele — em geral, o Sprint atual ou a próxima Release.

De forma geral, o Product Backlog ser planejável significa que ele permite:

- estabelecer o escopo mais provável do Sprint atual ou da próxima Release. Ou seja, o que é mais provável que esteja dentro de um entregável ou de uma entrega, em geral com uma data determinada;
- definir quando os próximos Incrementos do Produto a serem desenvolvidos poderão ser entregues para os clientes (a data da entrega), dado um escopo provável;
- calcular a capacidade de entrega do Time de Desenvolvimento, em geral a partir da média do que o Time de Desenvolvimento foi capaz de entregar por Sprint, também chamada de Velocidade do Time de Desenvolvimento (veja em [26.7 Velocidade do Time de Desenvolvimento](#));
- monitorar o progresso do trabalho em direção a uma entrega, dado o trabalho e o tempo restantes. Gráficos de Acompanhamento do Trabalho são ferramentas que podem ser usadas com esse propósito (veja em [26.7 Burndown e Burnup: acompanhando o trabalho](#)).

10.2.3 Emergente

O Product Backlog é uma lista dinâmica, que reflete o ambiente de mudança a partir do refinamento gradual dos detalhes do produto que está sendo desenvolvido.

À medida que o Time de Desenvolvimento trabalha desde o topo do Product Backlog, transformando-o em produto funcionando, o Product Owner adiciona, remove, modifica, refina e reordena seus itens. Esse é um trabalho contínuo, que ocorre desde o início até o final do projeto.

O Product Backlog nunca está completo. Ele evolui em um processo contínuo de descoberta, refletido na inspeção e adaptação do produto realizada a partir do *feedback* dos clientes e demais partes interessadas sobre os Incrementos do Produto a eles mostrados ou entregues.

10.2.4 Gradualmente detalhado

O Product Backlog é progressivamente refinado ao longo de todo o projeto, de forma que seus itens possuam um detalhamento adequado. Itens do alto do

Product Backlog possuem granularidade mais fina e, assim, representam mais detalhes. Esses são os próximos itens a serem desenvolvidos; logo, devem possuir os detalhes de negócios suficientes e necessários para que possam ser colocados em desenvolvimento.

Itens abaixo no Product Backlog possuem granularidade gradativamente mais grossa e, assim, representam menos detalhes, de forma que itens da parte de baixo, em geral, não possuem detalhe algum. Mais detalhes do que o necessário e suficiente é, geralmente, desperdício.

À medida que um item ganha importância, ele sobe no Product Backlog, pode evoluir em itens menores e os detalhes de negócios que serão necessários para seu desenvolvimento são esclarecidos e elaborados.

A Analogia do horizonte, descrita em [1.7 Redução do desperdício](#), descreve bem esse comportamento.

Discussão: qual o melhor Product Backlog?

Na figura 10.3, apresento três possíveis formatos para o Product Backlog.

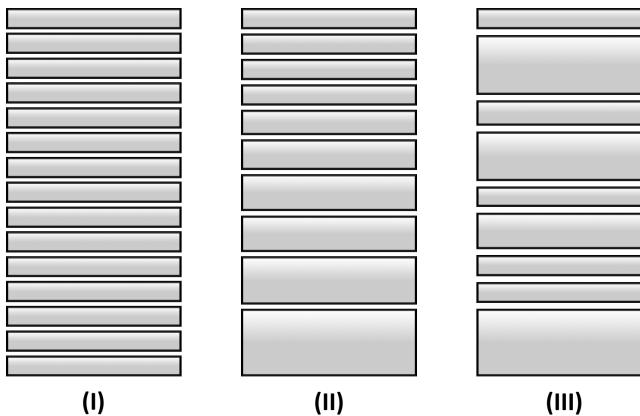


Fig. 10.3: O Product Backlog é gradualmente detalhado

No Product Backlog (I), todos os itens têm granularidade fina e muitos detalhes. Mais detalhes do que o suficiente e necessário constituirão desper-

dício, já que, pela natureza emergente do Product Backlog, os detalhes mudarão.

No Product Backlog (III), ao menos um item de seu alto não possui detalhes suficientes para ser colocado em desenvolvimento, o que significa um alto risco para o trabalho do Time de Desenvolvimento. Para corrigir esse problema, seria necessário fatiar esse item em um ou mais alguns poucos itens pequenos e detalhados, que carregariam as partes mais importantes e manteriam a alta prioridade. O resto seria movido, em um item, mais para baixo no Product Backlog. Além desse item, diversos outros mais abaixo possuem mais detalhes do que o necessário, o que novamente constitui desperdício, já que há uma grande chance de que os detalhes mudem até o momento do seu desenvolvimento. Refinar esse Product Backlog se faz necessário.

O Product Backlog (II) é o que considero o ideal, pois é gradualmente detalhado, ou seja, possui detalhes suficientes e necessários, dependentes da ordem do item.

CAPÍTULO 11

Sprint Backlog

11.1 O QUE É O SPRINT BACKLOG?

O Sprint Backlog é uma lista de itens selecionados do alto do Product Backlog para o desenvolvimento do Incremento do Produto no Sprint (**o quê**), adicionada de um plano de como esse trabalho será realizado (**como**).

O Sprint Backlog existe apenas no contexto de seu Sprint correspondente. Assim, ele é criado na reunião de Sprint Planning, e deixa de existir após as reuniões de Sprint Review e Sprint Retrospective de seu Sprint.

O Sprint Backlog pertence ao Time de Desenvolvimento e é uma importante ferramenta utilizada por seus membros para organizarem o trabalho durante o Sprint. Por essa razão, o Sprint Backlog deve ser de alta visibilidade.

11.1.1 O quê?

A lista de itens escolhida para o Sprint Backlog é o resultado da colaboração entre Time de Desenvolvimento e Product Owner, realizada na reunião de Sprint Planning. Esses itens são retirados do alto do Product Backlog, de acordo com a ordenação realizada pelo Product Owner, e movidos para o Sprint Backlog que está sendo criado.

Essa lista representa uma previsão de que itens os membros do Time de Desenvolvimento acreditam que conseguirão entregar naquele Sprint. Para construí-la, o Time de Desenvolvimento coopera com o Product Owner para obter os detalhes suficientes e necessários para ser capaz de escolher uma quantidade de itens, desde o topo do Product Backlog, que acredite que preencha sua capacidade de trabalho em um Sprint. Com frequência, Times de Desenvolvimento utilizam sua Velocidade medida como parâmetro para essa escolha (veja [26.7 Velocidade do Time de Desenvolvimento](#)).

Um Product Backlog bem ordenado terá em seu topo itens que, em conjunto, levam a se realizar a necessidade de negócios mais importante para os clientes naquele momento (veja *Ordenado*, em [10.2 Como é o Product Backlog?](#)). Essa necessidade de negócios, após ajustada e dimensionada de acordo com o que o Time de Desenvolvimento acredita ser capaz de entregar, pode se transformar na Meta do Sprint. Assim, os itens para o Sprint Backlog serão desenvolvidos, do mais importante ao menos importante, de forma a se realizar a Meta do Sprint (veja [24.2 Meta de Sprint](#)).

11.1.2 Como?

O plano de como os itens do Sprint Backlog serão desenvolvidos é geralmente expresso por um conjunto de tarefas correspondente a cada item, além da indicação do andamento de cada tarefa. Ou seja, se a tarefa ainda não foi iniciada, se está em andamento ou se já está terminada.

Estimativas do tempo de desenvolvimento de cada tarefa podem ser adicionadas para que o Time de Desenvolvimento possa acompanhar seu progresso em direção ao final do Sprint com o uso de Gráficos de Sprint Burn-down. Porém, diversos times simplesmente contam o número de tarefas restantes com esse propósito (veja [27.3 Gráfico de Sprint Burndown](#)).

Em geral, essas tarefas são estabelecidas na própria reunião de Sprint Planning pelo Time de Desenvolvimento. Elas representam os pequenos passos, normalmente técnicos, necessários para que o item correspondente esteja pronto, de acordo com a Definição de Pronto (veja [12 Definição de Pronto](#)). Assim, quando todas as tarefas de um item estão terminadas, o item está pronto.

O formato de tarefas, embora largamente utilizado, não é prescrito pelo Scrum e há times que optam por não quebrar os itens em tarefas.

11.2 COMO É O SPRINT BACKLOG?

O Sprint Backlog pode ser representado simplesmente por uma lista priorizada dos itens escolhidos para o Sprint. Mas, como mencionei anteriormente, geralmente o Time de Desenvolvimento quebra esses itens do Sprint Backlog em tarefas. Nesses casos, um Quadro de Tarefas, como o da figura [12.1](#), é considerado uma boa representação de Sprint Backlog.



Fig. 11.1: Quadro de Tarefas representando o Sprint Backlog

Os itens do Sprint Backlog estão dispostos verticalmente de acordo com a sua prioridade de desenvolvimento, ou seja, serão desenvolvidos de cima para baixo. A cada item corresponde uma raia horizontal, demarcada verticalmente apenas pelo item mais abaixo ou pelo fim do quadro.

Estarão dispostas em cada raia as tarefas correspondentes àquele item, distribuídas em diferentes colunas de acordo com seu status de andamento: “a fazer”, “em andamento” e “pronto” (ou quaisquer termos semelhantes). Esse comportamento está ilustrado na figura 11.2.

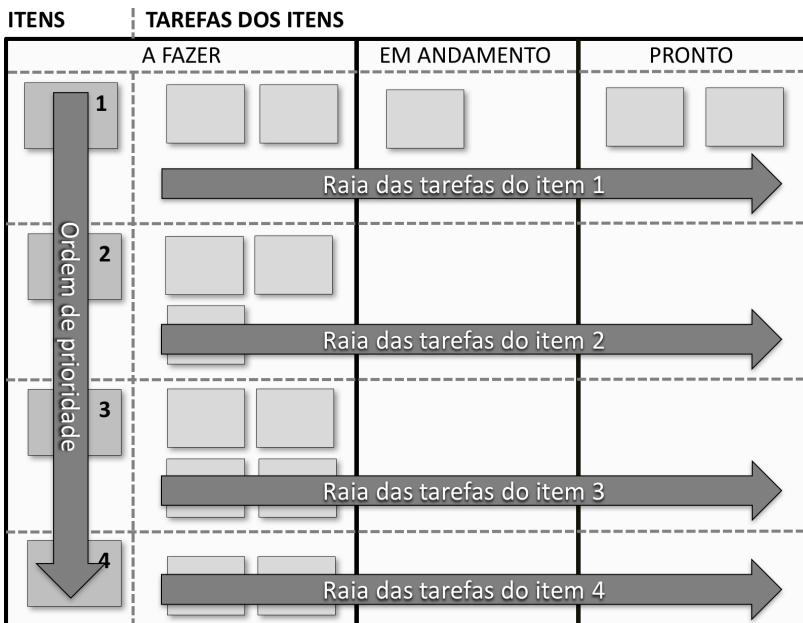


Fig. 11.2: Como utilizar o quadro de tarefas

O Sprint Backlog pertence ao Time de Desenvolvimento, que é responsável por seu uso e manutenção. O Time de Desenvolvimento usa o melhor de seu conhecimento no momento do planejamento para criá-lo. É importante, no entanto, que o Sprint Backlog sempre reflita o momento atual e, assim, espera-se que ele evolua e seja atualizado no decorrer do Sprint, de forma a refletir o trabalho restante.

Assim, o Sprint Backlog não é estático: embora não se devam modificar os itens escolhidos para o Sprint, o Time de Desenvolvimento atualiza suas tarefas sempre que houver qualquer mudança: progressão no status de andamento de uma tarefa, reestimativa, adição ou remoção de uma tarefa. No entanto, uma quantidade de mudanças que dificulte ou até mesmo torne impossível de se realizar a Meta do Sprint indicam, em geral, um planejamento para o Sprint mal executado.

O Sprint Backlog não deve ser utilizado por outras pessoas como ins-

trumento de controle — nem mesmo pelo Product Owner, pelo ScrumMaster ou por algum gerente externo. É um artefato para maximizar a visibilidade do trabalho, de forma que o próprio Time de Desenvolvimento possa acompanhar seu progresso em busca da Meta do Sprint, facilitando a auto-organização.

Discussão: trabalho priorizado

Durante o Sprint, os membros do Time de Desenvolvimento trabalham a partir do item mais importante e em direção ao menos importante, visando a realizar a Meta do Sprint. Embora normalmente trabalhem em tarefas diferentes ou, no máximo, em pares, eles colaboram buscando tornar prontos — de acordo com a Definição de Pronto — os itens mais importantes primeiro.

Em meu trabalho, no entanto, não é incomum eu observar times em que cada membro ou conjunto de membros atua sobre itens diferentes, de acordo com áreas de conhecimento ou algum outro critério de divisão (veja a figura 11.3). Esse comportamento é considerado disfuncional e traz diversos prejuízos.

Além de não estimular o trabalho em equipe e o compartilhamento de conhecimento, essa abordagem leva à abertura do desenvolvimento de vários itens ao mesmo tempo sem, no entanto, estimular seu fechamento. Há, assim, o risco de se chegar ao final do Sprint sem vários dos itens prontos e, ao não privilegiar os itens mais importantes, aumenta o risco de não se realizar a Meta do Sprint.

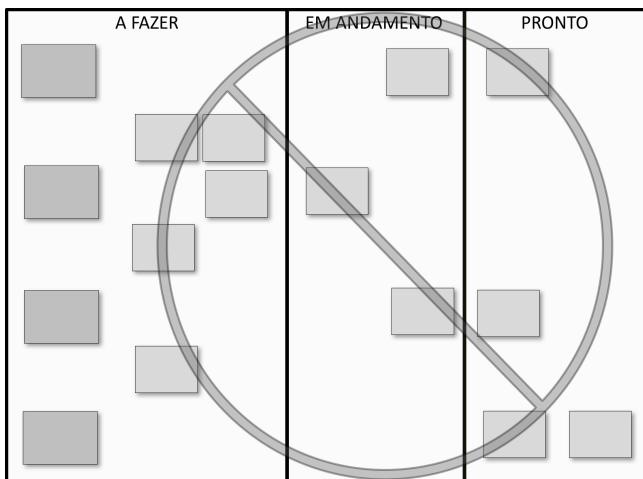


Fig. 11.3: Divisão indesejável de trabalho

O método Kanban (veja em [28.4 Lean Kanban](#)) traz uma frase que traduz muito bem como esperamos resolver esse problema: *pare de começar e comece a terminar!*

Discussão: quadro real x quadro virtual

A representação do Sprint Backlog é muito mais eficiente em um Quadro de Tarefas real do que em um quadro virtual de um programa de computador, que necessita de ser acessado para mostrar a informação. O quadro virtual pode fazer sentido apenas quando se utiliza um time distribuído, ou seja, cujos membros não trabalham juntos na mesma sala.

O quadro real deve ser bem visível, localizado na própria sala de trabalho do Time de Desenvolvimento, e notas adesivas são usadas geralmente para representar os itens e suas tarefas correspondentes. Esse quadro funciona como um “irradiador de informação”, termo originado no Sistema Toyota de Produção (veja em [5.3 Lean e Sistema Toyota](#)). Com o uso de um irradiador de informação, a informação chega aos olhos de quem é importante chegar, sem haver um esforço significativo para buscá-la.

Ao escrever em uma nota adesiva para o quadro real, pense que os outros membros do time devem entender facilmente o que está escrito, pois seu conteúdo pertence a todos. Escreva com letra grande e legível. Utilize marcadores fortes em vez de canetas comuns, de forma que se possa ler bem à distância. Use notas adesivas de boa marca, com uma cola que não sai facilmente, e retire cada folha do bloco com cuidado suficiente para não entortá-la, de forma que depois não caia facilmente do quadro.

CAPÍTULO 12

Definição de Pronto

12.1 O QUE É A DEFINIÇÃO DE PRONTO?

A Definição de Pronto é um acordo formal entre Product Owner e Time de Desenvolvimento sobre o que é necessário para se considerar que um trabalho realizado no Sprint está “pronto”. É um conjunto de critérios compartilhados para que, quando o Time de Desenvolvimento ou um de seus membros afirmem que um item ou o Incremento do Produto está pronto, todos compreendam o que isso significa.

A mesma Definição de Pronto se aplica a cada item do Sprint Backlog e ao Incremento do Produto como um todo, que é o conjunto dos itens desenvolvidos no Sprint. Ela ajuda o Time de Desenvolvimento no momento de planejar o trabalho a ser realizado no Sprint e guia o Time de Desenvolvimento durante a realização desse trabalho.

PRONTO?

O time entrega uma funcionalidade que considera pronta. Ao começar a experimentar o que lhe foi entregue, o cliente encontra vários erros. Ele, então, pergunta a um dos desenvolvedores:

— *Você testou isso?*

Ao que o desenvolvedor responde:

— *Não...*

— *Mas você não disse que está pronto?* — se espanta o cliente.

— *Sim, está pronto. Só falta testar.* — afirma o desenvolvedor.

— *Se falta testar, não está pronto!* — responde o cliente.

Situações como essa são comuns em projetos de desenvolvimento de software. Isso ocorre porque não há uma compreensão conjunta do que “estar pronto” significa.

Uma Definição de Pronto ideal estabelece que o resultado do trabalho de um Time de Desenvolvimento em um Sprint seja entregável. Dessa forma, idealmente nenhum desenvolvimento, teste, integração ou quaisquer tarefas adicionais devem ser necessárias para que o Incremento do Produto produzido no Sprint possa ser entregue aos clientes do projeto. Entregar ou não ao final do Sprint, no entanto, é uma decisão que cabe ao Product Owner. Mesmo que já seja possível, ele pode decidir por acumular alguns Incrementos do Produto para só então fazer uma entrega.

Diferentemente dos Critérios e Testes de Aceitação, a Definição de Pronto não é específica para um item ou para um grupo de itens. Ela é a mesma para todos os itens desenvolvidos (veja *Confirmação*, em [25.1 O que é a User Story?](#)). Uma boa Definição de Pronto exige, preferencialmente de forma explícita, que cada item passe nos Testes de Aceitação — e, portanto, nos Critérios de Aceitação — acordados especificamente para o item. Uma boa Definição de Pronto também garante que o Incremento do Produto tenha o nível de qualidade exigido para ser entregue.

12.2 COMO É A DEFINIÇÃO DE PRONTO?

A Definição de Pronto é criada antes do início do desenvolvimento do produto. Em geral, antes mesmo do primeiro Sprint. Entretanto, ela pode ser modificada e evoluir de forma a acomodar novas necessidades identificadas ao longo do projeto para o desenvolvimento do produto.

Para a construção de uma Definição de Pronto efetiva é importante que se considerem as restrições organizacionais que afetam o trabalho do Time de Scrum, sejam restrições de negócio, de processo, tecnológicas ou culturais. Devido a essas restrições, muitos Times de Scrum iniciam o projeto com uma Definição de Pronto distante da ideal: algum trabalho adicional ainda será necessário para tornar o resultado do Sprint, o Incremento do Produto, entregável.

Exemplos dessas atividades adicionais incluem: diversos tipos de testes realizados por uma equipe externa (como exploratórios, de integração, de estabilidade e de segurança), outras validações, como auditorias externas, integrações com outros produtos em desenvolvimento, homologações por parte do cliente e alguns tipos de documentação. A necessidade dessas atividades não permite que o Incremento do Produto “pronto” seja entregável.

A existência de um trabalho adicional ao “pronto” em cada Sprint é uma disfunção e traz riscos consideráveis ao projeto. Esse trabalho, ao ser realizado em Sprints seguintes, desorganiza e adia a entrega de valor.

Uma alternativa comum é realizá-lo em um Sprint adicional imediatamente anterior à entrega, que é chamado por muitos de “Sprint de estabilização”. Nessa abordagem, diversos problemas serão descobertos tarde e cada entrega poderá acumular problemas comuns a projetos em cascata.

Na medida do possível, o Time de Scrum trabalha ao longo do projeto para reduzir a disfunção, transferindo progressivamente esse trabalho para dentro dos Sprints e, assim, tornando sua Definição de Pronto cada vez mais estrita, à medida que se consegue aproximar, de fato, o “pronto” do “entregável”.

Em casos muito particulares — em se tratando, por exemplo, de projetos de alto risco em que são necessários testes adicionais e auditorias externas — o trabalho adicional antes da entrega será sempre necessário e existirá durante todo o projeto.

A Definição de Pronto varia de time para time, de projeto para projeto. O Time de Desenvolvimento possui, entre seus membros, todas as habilidades e conhecimentos necessários para entregar, ao final do Sprint, um Incremento do Produto pronto de acordo com a Definição de Pronto. Assim, é fácil entender como a Definição de Pronto e a formação do Time de Desenvolvimento são intrinsecamente relacionadas.

A figura ?? mostra um exemplo de Definição de Pronto possível para um sistema de *software*. Como no exemplo, a Definição de Pronto geralmente se parece com uma lista de atividades ou um processo pelo qual passam os itens de trabalho no Sprint. Essa lista é definida de acordo com as necessidades do produto — das quais qualidade sempre faz parte — e em conformidade com as convenções e padrões organizacionais. Essas atividades incluem tudo o que deve ser realizado para se construir o produto, como por exemplo, diferentes tipos de testes que se considerem necessários, documentação que faça parte do produto, quaisquer integrações que devam ser realizadas etc.

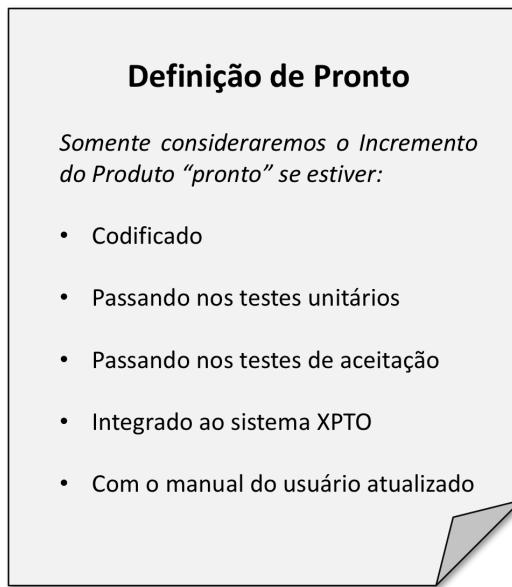


Fig. 12.1: Um exemplo de Definição de Pronto

As atividades da Definição de Pronto, portanto, são utilizadas como referência quando o Time de Desenvolvimento seleciona quanto trabalho prevê que conseguirá produzir no Sprint. Servem da mesma forma para quando o Time de Desenvolvimento estabelece o plano para o Sprint, quebrando os itens do Sprint Backlog em tarefas, por exemplo.

A Definição de Pronto é criada, compreendida e compartilhada por todos os membros do Time de Scrum. Dessa forma, mantê-la visível para eles é essencial.

CAPÍTULO 13

Incremento do Produto

Em cada Sprint do projeto, o Time de Desenvolvimento trabalha nos itens selecionados do alto do Product Backlog para o Sprint Backlog, do mais importante para o menos importante, visando a realizar a Meta do Sprint. O Incremento do Produto é o resultado desse trabalho, ou seja, é a soma de todos os itens completos em um Sprint. Portanto, é um incremento de funcionalidades do produto prontas, de acordo com a Definição de Pronto (veja [12 Definição de Pronto](#)), produzido em cada Sprint do projeto.

O Incremento do Produto é composto por novas funcionalidades e por melhorias no que foi produzido anteriormente, oriundas de itens do Product Backlog. A natureza incremental desse trabalho faz com que um item já considerado “pronto” jamais volte para o desenvolvimento.

Quando há necessidade de mudanças, um novo item relativo a esse trabalho é criado no Product Backlog. No entanto, itens não “prontos” ao final do

Sprint não fazem parte do Incremento do Produto, e devem voltar ao Product Backlog a critério e escolha do Product Owner.

Espera-se que o Incremento do Produto seja entregável (ou, como alguns dizem, “potencialmente entregável”), de forma que o Product Owner possa decidir por fazer uma entrega para os clientes do projeto ao final do Sprint. No entanto, embora deva-se fazer entregas frequentes, o Product Owner pode optar por acumular alguns Incrementos do Produto para só então fazer uma entrega (um Release). Ou seja, entregável é diferente de entregue.

Entre as razões para tal, um Incremento nem sempre representa valor suficiente para ser utilizado por seus usuários. Assim, pode não haver sentido entregá-lo ao final do Sprint em que foi produzido. E, mesmo que represente valor suficiente, os clientes podem não conseguir absorver mudanças tão frequentemente. Ou pode haver questões políticas, burocráticas ou técnicas que impedem o Product Owner de realizar a entrega ao final de cada Sprint.

O Time de Desenvolvimento e o Product Owner demonstram o Incremento do Produto para os clientes do projeto e pessoas relevantes ao final de cada Sprint, na reunião de Sprint Review. O principal objetivo dessa reunião é obter *feedback* dessas pessoas sobre o trabalho realizado e dar a elas um senso de progresso do projeto.

Para tornar esse *feedback* possível, o Incremento do Produto deve representar valor visível para os presentes. Ou seja, deve conter funcionalidades que possam ser experimentadas e usadas.

CAPÍTULO 14

Definição de Preparado

14.1 O QUE É A DEFINIÇÃO DE PREPARADO?

A Definição de Preparado é um artefato utilizado para garantir que os itens a serem considerados na reunião de Sprint Planning estejam preparados segundo um critério bem definido. É um acordo formal entre Product Owner e Time de Desenvolvimento sobre o estado em que um item do Product Backlog deve estar para ser considerado preparado para entrar no Sprint Backlog.

Mesmo que o item, no momento da reunião de Sprint Planning, seja de alta prioridade, ele poderá ser rejeitado pelo Time de Desenvolvimento se não estiver preparado de acordo com a Definição de Preparado.

A Definição de Preparado não é um artefato oficial do Scrum e, portanto, é opção do Time de Scrum usá-la ou não.

É comum Times de Scrum realizarem esse trabalho de preparação apenas durante a reunião de Sprint Planning. Diversos times optam, no entanto, por

anticipá-lo, realizando-o ao longo do Sprint anterior, em sessões de Refinamento do Product Backlog (veja 23 *Refinamento do Product Backlog*). Essas sessões realizadas pelo Product Owner e membros do Time de Desenvolvimento são importantes para diminuir os riscos de um Sprint mal planejado, já que de outra forma detalhes demais são deixados para serem discutidos apenas na reunião de Sprint Planning.

Esse trabalho excessivo torna a reunião longa, cansativa e ineficiente, originando um Sprint Backlog mal formulado e colocando em risco todo o trabalho do Sprint. O principal resultado esperado das sessões de Refinamento do Product Backlog é que um número suficiente de itens a serem considerados na reunião de Sprint Planning seguinte esteja preparado para ser desenvolvido. Nesses casos, a Definição de Preparado funciona como um critério de entrada para o Sprint, como se pode observar na figura 14.1.

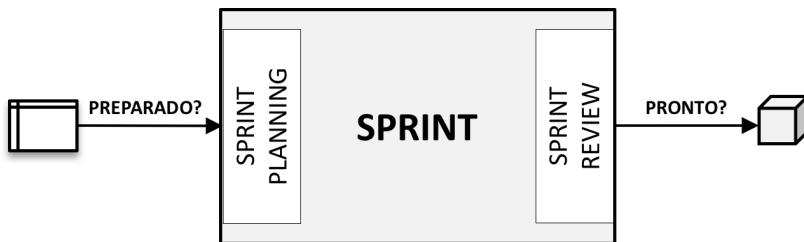


Fig. 14.1: Fluxo do item: Definição de Preparado e Definição de Pronto

Pode-se observar nessa mesma figura o fluxo de um item do Product Backlog através de um Sprint. Caso o item do Product Backlog esteja preparado de acordo com a Definição de Preparado, ele pode ser aceito para discussão na reunião de Sprint Planning e, consequentemente, pode fazer parte do Sprint Backlog. Se o item estiver pronto de acordo com a Definição de Pronto ao final do Sprint, ele é aceito na reunião de Sprint Review e sai do Sprint como parte do Incremento do Produto gerado no Sprint.

14.2 COMO É A DEFINIÇÃO DE PREPARADO?

A Definição de Preparado, assim como a Definição de Pronto, é criada antes do início do desenvolvimento do produto, geralmente antes mesmo do primeiro Sprint. Entretanto, ela pode ser modificada e evoluir de forma a acomodar novas necessidades identificadas ao longo do projeto. A Definição de Preparado tem geralmente o formato de uma lista de critérios, condições ou, ainda, passos de um processo. Veja um exemplo na figura 14.2.

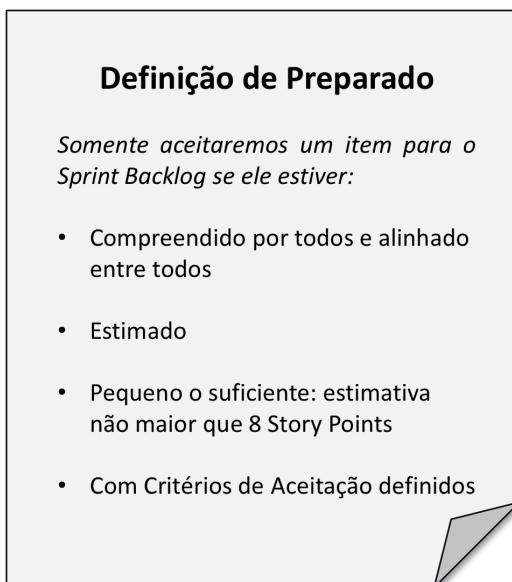


Fig. 14.2: Exemplo de Definição de Preparado

É comum a Definição de Preparado para itens do Product Backlog conter os seguintes tópicos:

- deve ter havido um alinhamento sobre o item, de forma a gerar uma compreensão compartilhada sobre o que o item representa;
- o item deve possuir Critérios de Aceitação, discutidos, compreendidos e acordados entre Product Owner e Time de Desenvolvimento;

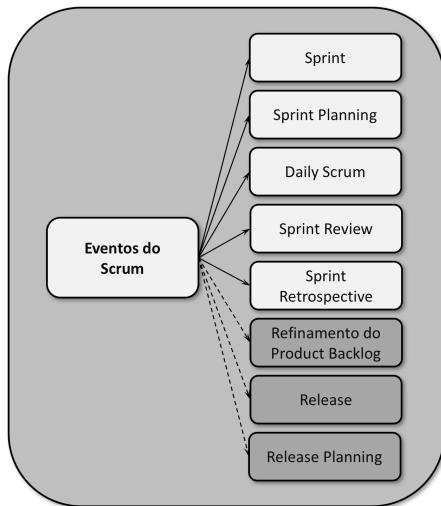
- o item deve ter sido estimado pelo Time de Desenvolvimento;
- o item deve ser pequeno o suficiente, de acordo com algum critério estabelecido pelo Time de Desenvolvimento (um valor máximo para sua estimativa, por exemplo).

Assim como a Definição de Pronto, a Definição de Preparado é criada, compreendida e compartilhada por todos os membros do Time de Scrum. Assim, deve se manter visível para todos eles.

Parte IV

Eventos do Scrum

Os eventos do Scrum são o próprio ciclo de desenvolvimento, chamado de Sprint, e as reuniões ou cerimônias realizadas durante o ciclo, que são: Sprint Planning, Daily Scrum, Sprint Review e Sprint Retrospective.



Adiciono a essas uma fase curta de inicialização do projeto, as sessões de Refinamento do Product Backlog, as Releases e as reuniões de Release Planning.

Timebox

Os eventos do Scrum possuem uma duração definida, chamada de *timebox*. O *timebox* pode definir o tempo máximo em que um evento deve ocorrer, como nas reuniões, ou o tempo exato, como no Sprint. O objetivo dos *timeboxes* é limitar o tempo em que um objetivo deve ser alcançado, de forma que não se gere desperdício com trabalhos intermináveis. Os *timeboxes* ajudam a criar um ritmo ou uma regularidade no trabalho do Time de Desenvolvimento.

CAPÍTULO 15

Inicialização

Ken Schwaber (2004), que com Jeff Sutherland desenvolveu e formalizou o framework Scrum, declarou em um de seus livros: *o mínimo necessário para se iniciar um projeto com Scrum consiste de uma visão e um Product Backlog. A visão descreve por que o projeto está sendo realizado e qual é o estado final desejado.*

Não existe oficialmente no Scrum uma fase de inicialização de projeto. O estabelecimento de uma Visão do Produto e a criação de um Product Backlog inicial são as únicas atividades estritamente indispensáveis para se produzirem as entradas necessárias para o projeto. É comum, no entanto, esse trabalho de preparação ser enquadrado em uma fase de inicial extraoficial e a ele se somarem outras atividades específicas de cada contexto.

Essa fase inicial recebe os mais variados nomes: Sprint Zero, Pré-Jogo, *Discovery*, *Inception*, entre outros. Desses, o mais comum e o mais incorreto é Sprint Zero. Como veremos adiante, um Sprint possui duração fixa, todos os

Sprints possuem a mesma duração e cada Sprint sempre gera valor visível para os usuários, o que deixa claro que Sprint Zero é uma denominação incorreta.

Seja quais forem as atividades realizadas nessa fase não-oficial de inicialização, ela deve ser curta — de alguns dias a poucas semanas. Esse trabalho de preparação deve ser realizado sempre na base do mínimo estritamente necessário e suficiente, com o objetivo de não se gerar desperdícios e não se adiar demais o início da produção de valor.

As atividades mais comuns e, de forma geral, necessárias em uma fase de inicialização são:

- criação de uma Visão de Produto;
- criação de um Product Backlog inicial, em geral suficiente para o primeiro ou para os primeiros Sprints. Esse Product Backlog pode, no entanto, ser estendido para mais adiante, desde que com a granularidade adequada;
- formação do Time de Scrum, ou seja, a definição de quem é o Scrum-Master, o Product Owner e os membros do Time de Desenvolvimento.

Além disso, não é incomum outras atividades se somarem a essas, dependendo de cada contexto. A seguir, listo algumas apenas como exemplo:

- criação ou adaptação de uma infraestrutura inicial que servirá de suporte para o início do desenvolvimento do produto, sempre na base do mínimo suficiente e necessário, e que evoluirá ao longo do projeto;
- definição de uma arquitetura inicial, sempre na base do mínimo suficiente e necessário, e que evoluirá ao longo do projeto. Essa arquitetura inicial visa a reduzir os riscos de decisões tardias que invalidariam o que já foi produzido, mas sem correr o risco de engessar o desenvolvimento;
- definição da duração do Sprint, que se manterá a mesma ao longo do projeto;
- investigações de que tecnologias serão úteis ou necessárias para o projeto;

- aquisição de tecnologias e equipamentos necessários para o projeto;
- criação da Definição de Pronto;
- criação da Definição de Preparado;
- refinamento, em uma ou mais sessões, dos itens iniciais do Product Backlog para prepará-los para o primeiro Sprint;
- negociação de detalhes do contrato e sua assinatura;
- identificação de partes interessadas que deverão ser levadas em conta no desenvolvimento do produto;
- realização de treinamentos necessários. Para times iniciando com Scrum, esses treinamentos podem ser essenciais;
- realização de atividades de formação de time (*teambuilding*).

No entanto, ao contrário dessa prática de realização de uma fase de inicialização, é mais correto nos termos do Scrum (e talvez mais eficiente e menos arriscado) que essas atividades preparatórias extras sejam progressivamente realizadas dentro dos primeiros Sprints, de forma que não se postergue a produção de valor.

CAPÍTULO 16

Sprint

- **Objetivo:** produzir valor entregável, de forma a realizar a Meta do Sprint
- **Quando:** durante todo o desenvolvimento do produto, um atrás do outro
- **Duração:** fixa de uma a quatro semanas
- **Participantes obrigatórios:** Time de Desenvolvimento, Product Owner e ScrumMaster
- **Saídas esperadas:** um Incremento do Produto pronto, de acordo com a Definição de Pronto, que realize a Meta do Sprint

16.1 O QUE É O SPRINT?

O Sprint é o ciclo de desenvolvimento, a iteração, que se repete ao longo de todo o projeto, um atrás do outro. Em cada Sprint, o Time de Scrum planeja, desenvolve um Incremento do Produto, coleta *feedback* e busca melhorar sua forma de trabalhar.

Assim, ocorrem dentro de cada Sprint a reunião de Sprint Planning, em seu primeiro dia; as reuniões de Daily Scrum, em cada dia do Sprint; e as reuniões de Sprint Review e de Sprint Retrospective, em seu último dia, além do trabalho de desenvolvimento propriamente dito e quaisquer outras atividades ou reuniões realizadas com a participação do Time de Desenvolvimento. Veja como é um Sprint na figura 16.1.

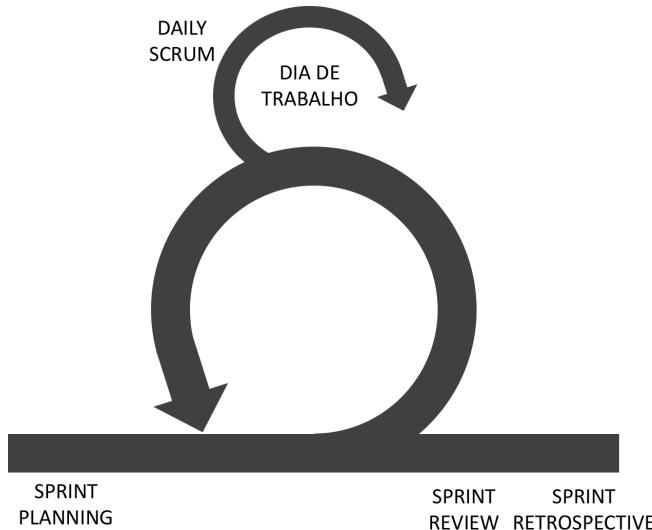


Fig. 16.1: O Sprint

De seu início até o final, o projeto com Scrum funciona inteiro dentro de Sprints, que acontecem um após o outro, sem paradas ou intervalos (veja a figura 16.2). Desse modo, não há intervalos entre Sprints, nem se interrompe um Sprint por alguns dias para resolver quaisquer questões, relativas

ao projeto ou não. Tudo o que acontece em um projeto com Scrum relativo ao trabalho do Time de Desenvolvimento é trazido para dentro dos Sprints.

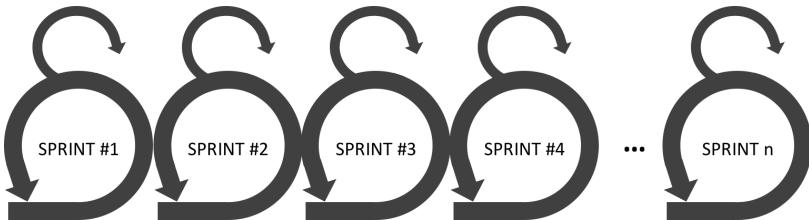


Fig. 16.2: Scrum: um Sprint após o outro

O Sprint possui como finalidade cumprir um objetivo bem definido, negociado e acordado entre o Time de Desenvolvimento e o Product Owner durante a reunião de Sprint Planning, que determina o que deve ser realizado a partir do que estará pronto ao final do Sprint. É a chamada Meta do Sprint, que guia o trabalho do Time de Desenvolvimento, levando seus membros a trabalharem juntos, e não em diferentes iniciativas (veja [24.2 Meta de Sprint](#)).

Durante o Sprint, nenhuma mudança que ameace a Meta pode ocorrer e a Meta não pode ser modificada. Em casos de exceção, quando a Meta perde seu sentido no decorrer do ciclo, o Sprint pode ser cancelado (veja [16.3 O Sprint pode ser cancelado?](#)).

Para realizar a Meta do Sprint, o Time de Desenvolvimento desenvolve, ao longo do Sprint, um Incremento do Produto pronto, de acordo com a Definição de Pronto, a partir dos itens do Sprint Backlog. Esses itens são escolhidos e retirados do alto do Product Backlog durante a reunião de Sprint Planning, e são desenvolvidos no Sprint na ordem de sua prioridade, ou seja, do mais importante para o menos importante.

SPRINT = PROJETO

Cada Sprint pode ser entendido como um “miniprojeto”, no qual se realiza o planejamento e o desenvolvimento de determinados itens visando-se a realizar um objetivo bem definido — a Meta do Sprint. Essa meta pode traduzir um objetivo ou necessidade de negócios que será realizado a partir do desenvolvimento dos itens planejados.

Podemos dizer que essa Meta corresponde a uma (mini) visão do produto a ser gerado nesse “miniprojeto”. O “miniproduto” desse “miniprojeto” é o Incremento do Produto. Ao final de cada “miniprojeto”, o trabalho realizado é revisado, *feedback* é colhido para os próximos “miniprojetos” e o time levanta melhorias a realizar. O desenvolvimento do produto com Scrum acontece, portanto, em “miniprojetos”, um atrás do outro.

16.2 COMO É O SPRINT?

16.2.1 Duração

Os Sprints são *timeboxes* e têm duração fixa e constante, o que significa que eles não devem durar nem mais, nem menos que o estabelecido, e que essa duração não varia de Sprint para Sprint. A ideia por trás desse conceito é a de criar ritmo e disciplina no trabalho do Time de Desenvolvimento, além de regularidade na obtenção de *feedback* e na participação do Product Owner, dos clientes e de demais partes interessadas.

A duração fixa do Sprint também viabiliza o cálculo da Velocidade do Time de Desenvolvimento (veja 26.7 *Velocidade do Time de Desenvolvimento*), o que facilita o seu planejamento.

Quando Scrum foi criado, a literatura determinava que os Sprints tivessem obrigatoriamente um mês de duração. A prática do Scrum em projetos com diferentes características fez com que oficialmente se aceitassem Sprints de três, de duas e, mais recentemente, de uma semana. A duração dos Sprints, portanto, é fixada em algum valor entre uma e quatro semanas.

Para se definir a duração dos Sprints, é comum o uso de semanas (dias

corridos) ou de dias úteis. A medida em semanas cria uma maior regularidade, o que ajuda a dar ritmo no trabalho do Time de Desenvolvimento e pode, por exemplo, facilitar a agenda do Product Owner e favorecer a presença de determinadas pessoas nas reuniões de Sprint Review, como clientes e outras partes interessadas.

Adicionalmente, pode ser psicologicamente interessante iniciar o Sprint em uma segunda-feira e terminá-lo em uma sexta-feira, trazendo mais facilmente aos envolvidos um sentimento de fechamento. Mas nem sempre isso é possível. Quando o Product Owner atua em mais de um projeto, por exemplo, os Sprints de projetos diferentes podem começar em dias diferentes e terminar em dias diferentes, para que ele possa estar presente em todos os eventos que lhe cabem.

Já a medida da duração do Sprint em dias úteis facilita a medição da Velocidade do Time de Desenvolvimento e o seu uso para o planejamento, já que o número de dias de trabalho do Sprint não varia com o advento de feriados, mas pode sacrificar a regularidade.

Pode ser necessário modificar a duração do Sprint dentro do mesmo projeto conforme as condições do projeto e do ambiente mudam. Essa necessidade é geralmente detectada pelo Time de Scrum em uma reunião de Sprint Retrospective, e raramente deve ocorrer. A nova duração escolhida para o Sprint deve também ser fixa e constante.

Apresento aqui alguns fatores que podem influenciar a escolha da duração do Sprint de um projeto.

Frequência das mudanças

A frequência com que mudam as necessidades de negócios e suas prioridades é um fator importante na escolha da duração do Sprint. Sprints menores aumentam a capacidade do Time de Desenvolvimento para responder a essas mudanças, e podem ser utilizados quando a natureza do projeto ou dos clientes tornam as necessidades de negócios mais voláteis. Sprints maiores podem ser usados em projetos com necessidades de negócios mais estáveis, em que o Product Owner encontrará menos problemas em planejar para um tempo mais longo.

Em qualquer um dos casos, o planejamento deve ser feito para um pe-

ríodo curto o suficiente para que sejam mínimas as chances da Meta do Sprint perder o sentido em seu decorrer.

Frequência de feedback

Quanto menor for o Sprint, mais frequente será o *feedback* dos clientes e demais partes interessadas sobre o que foi produzido. Quanto mais frequente for esse *feedback*, maiores serão as chances de se efetuarem correções de curso no trabalho realizado, o que diminui os riscos de desperdício do trabalho do Time de Desenvolvimento. Outro benefício é que os clientes e demais partes interessadas também verão os progressos do projeto mais frequentemente.

Da mesma forma, quanto menor for o Sprint, mais frequentes serão as retrospectivas do Time de Scrum, possibilitando-o experimentar e aprender mais rapidamente com soluções que possam trazer melhorias a seus processos. Além desse benefício, a sensação de sucesso e de trabalho cumprido que vem com o desfecho de um Sprint bem-sucedido pode ser um forte motivador para o trabalho do Time de Desenvolvimento. Assim, quanto mais frequentemente isso ocorrer, melhor.

Sobrecarga

A prática mostra que, mesmo para Sprints curtos como os de uma semana, é comum a disfunção de se ter a reunião de Sprint Planning ocupando boa parte de um dia de trabalho para alguns Times de Scrum. O mesmo pode ocorrer com as reuniões de Sprint Review e de Sprint Retrospective juntas. Nesses casos, o custo de se iniciar e de se encerrar o Sprint pode ser alto demais para que valha a pena se realizar um Sprint tão curto. É recomendável o endereçamento da disfunção antes de se utilizar um Sprint tão curto.

Ritmo

Sprints mais curtos podem ajudar a criar um ritmo para o trabalho do Time de Desenvolvimento. Esse ritmo ajuda a se manter o foco do trabalho, aumentando o compromisso do Time de Desenvolvimento com a Meta de cada Sprint.

No entanto, Sprints de uma semana podem acabar ditando um ritmo forte

e estressante demais para alguns Times de Desenvolvimento, que podem assim preferir Sprints com duração maior.

Tamanho do projeto

Para projetos curtos, é mais apropriado se estabelecer Sprints curtos, pois dessa forma o número de oportunidades de *feedback* dos clientes e demais partes interessadas é maior. Para um projeto com um ou dois meses de duração, por exemplo, Sprints de uma ou duas semanas são mais adequados.

Valor de negócio

A duração do Sprint escolhida é tal que o Time de Desenvolvimento pode produzir um Incremento do Produto que signifique valor de negócio visível para os clientes, que lhes será demonstrado durante a reunião de Sprint Review. Dependendo da natureza do projeto, um Sprint muito curto pode não ser suficiente para que seja possível se produzir valor visível suficiente.

Iniciantes

Para Times de Scrum que estão em seus primeiros passos na utilização de Scrum, não é incomum a escolha de Sprints mais curtos — de uma semana, por exemplo — para aumentar a frequência de *feedback*. Esse *feedback* frequente, tanto externo (nas reuniões de Sprint Review) quanto interno (nas reuniões de Sprint Retrospective), permite ao time iniciante evoluir mais rapidamente ao identificar onde e como melhorar.

No entanto, a menos que haja um acompanhamento especializado, recomendo de forma geral começar-se com Sprints de duas semanas para evitar que o time enfrente um período inicial de falhas sucessivas, de forma a dar tempo para que o time se acostume com o ritmo de trabalho e com as práticas mais essenciais do Scrum.

Foco

Em Sprints muito longos, como os de quatro semanas, o foco do Time de Desenvolvimento na Meta do Sprint tende, com o passar do tempo, a se enfraquecer. Sprints mais curtos favorecem a manutenção do foco na Meta

do Sprint.

16.2.2 Incrementos entregáveis

Em cada Sprint, é gerado pelo Time de Desenvolvimento um Incremento do Produto pronto, de acordo com a Definição de Pronto. Idealmente, esse Incremento está sempre apto a ser entregue para os clientes, ou seja, é entregável (ou “potencialmente entregável”).

No entanto, a entrega ou Release em si somente é feita quando o Product Owner julga que há valor suficiente para que seja utilizado. Isso, em muitos casos, ocorre apenas ao somar-se mais de um Incremento do Produto, ou seja, a partir dos resultados de mais de um Sprint (veja [13 Incremento do Produto](#)).

Há casos, no entanto, em que é necessário algum trabalho adicional antes que um conjunto de Incrementos do Produto possa, de fato, ser entregue. Isso aumenta os riscos do projeto ao se postergar a detecção de problemas (veja [12 Definição de Pronto](#)).

16.3 O SPRINT PODE SER CANCELADO?

Caso julgue que a Meta do Sprint perdeu o seu sentido, o Product Owner pode decidir pelo cancelamento do Sprint. Se houver, por exemplo, uma mudança na legislação ou regulamentação, uma ação da concorrência ou qualquer mudança drástica de prioridades que torne a Meta do Sprint obsoleta, não faz sentido prosseguir com o Sprint até o seu final.

Assim, se a Meta perder seu sentido, o Product Owner informará ao Time de Desenvolvimento sobre o cancelamento do Sprint. Os itens já prontos (de acordo com a Definição de Pronto) serão revistos, ao antecipar-se a reunião de Sprint Review. Uma reunião de Sprint Retrospective também será realizada em seguida.

Idealmente, um novo Sprint será iniciado em seguida. No entanto, como é uma situação de exceção, é razoável que Time de Desenvolvimento e Product Owner decidam como proceder, já que podem preferir não modificar o calendário já previsto de Sprints. Em vez de iniciar um novo Sprint com o tamanho regular, é comum optarem por iniciar um Sprint maior que contenha

os dias que sobraram, ou realizar um Sprint curto com esses dias restantes, de forma a manter as datas de início e término dos próximos Sprints.

No entanto, escolhe-se a duração fixa dos Sprints como curta o suficiente para que sejam mínimas as chances de a Meta perder o sentido nesse período, de forma que cancelamentos de Sprint sejam raros. É importante que seja dessa forma, pois cancelamentos de Sprint são traumáticos para todo o Time de Scrum e devem ser evitados, na medida do possível.

Discussão: falhar ou não falhar?

Alguns autores e praticantes de Scrum defendem que, no momento em que o Time de Desenvolvimento detecta que não conseguirá realizar a Meta do Sprint, o Product Owner seja chamado para se negociar uma redução da Meta, ou mesmo o cancelamento do Sprint.

A Meta do Sprint traz flexibilidade suficiente para que não realizá-la seja a exceção, e não a regra (veja [24.2 Meta de Sprint](#)). Na minha opinião, admitir a redução da Meta ou o cancelamento do Sprint é um comportamento que dá margem a se transformar um resultado indesejado — não se realizar a Meta — em uma parte normal do trabalho.

Ou seja, em vez de falhar e aprender com a falha e, assim, tornar-se capaz de definir Metas de tamanho mais adequado à sua capacidade de trabalho, o Time de Desenvolvimento pode, como consequência, sentir-se suficientemente confortável com o cancelamento de Sprints ou reduções da Meta para ameaçar seu compromisso com as Metas dos Sprints seguintes.

CAPÍTULO 17

Sprint Planning

- **Objetivo:** planejar o ciclo de desenvolvimento (Sprint) que se inicia
- **Quando:** no primeiro dia do Sprint, iniciando-o
- **Duração:** máxima proporcional a 8 horas para Sprints de 1 mês
- **Participantes obrigatórios:** Product Owner, Time de Desenvolvimento e ScrumMaster
- **Saídas esperadas:** Meta do Sprint e Sprint Backlog

17.1 O QUE É A SPRINT PLANNING?

O planejamento do ciclo de desenvolvimento ou Sprint atual é realizado na reunião de Sprint Planning. Esta inicia-se logo no primeiro momento do primeiro dia do Sprint.

A reunião de Sprint Planning conta com a participação obrigatória do ScrumMaster, que atua como um facilitador, do Product Owner e dos membros do Time de Desenvolvimento. Os dois últimos colaboram e negociam o que será desenvolvido e qual o objetivo deve ser realizado a partir do produto do trabalho realizado nesse Sprint — a Meta do Sprint.

Os membros do Time de Desenvolvimento estabelecem como o que foi negociado será desenvolvido. Essa lista de itens retirados do alto do Product Backlog para desenvolvimento, somada ao plano de como esse trabalho será realizado, é chamada de Sprint Backlog.

Todos os membros do Time de Desenvolvimento participam da reunião com igual poder de opinião e decisão, quaisquer que sejam suas áreas de conhecimento ou de atuação.

17.1.1 Duração

A reunião de Sprint Planning é um *timebox* de 5% do tempo do Sprint. Ou seja, ela deve durar no máximo oito horas para Sprints de um mês, e proporcionalmente menos para Sprints mais curtas. Como para qualquer *timebox*, essa duração máxima deve ser rigorosamente respeitada.

Ao se aliar a prática de sessões de Refinamento do Product Backlog, contando com a participação do Product Owner e dos membros do Time de Desenvolvimento, ao uso da Definição de Preparado, pode-se reduzir significativamente a duração da reunião de Sprint Planning. No entanto, quando essas boas práticas não são usadas, a reunião tende a ficar mais longa e menos produtiva (veja [23 Refinamento do Product Backlog](#)).

17.1.2 Saídas

Meta do Sprint. Time de Desenvolvimento e Product Owner negociam e estabelecem um objetivo para o Sprint corrente, que determina o que deve ser realizado nesse Sprint. É a chamada Meta do Sprint, que estabelece uma direção que leva os membros do Time de Desenvolvimento a trabalharem juntos e não em diferentes iniciativas. Em geral, é uma necessidade de negócios ou do usuário que está dentro da estratégia do produto desenvolvida pelo Pro-

duct Owner para se realizar a Visão do Produto (veja o capítulo [24 Metas: definindo os objetivos](#)).

Nesses termos, uma boa Meta de Sprint é qualitativa, e não quantitativa. Ela pode ser, por exemplo, *permitir que a loja virtual disponibilize novos produtos*, ao passo que *desenvolver os itens A, B, C e D do Product Backlog* não seria uma boa meta.

A Meta do Sprint, enquanto um objetivo de negócios ou do usuário, é o resultado da colaboração entre Product Owner e Time de Desenvolvimento. Normalmente, é estabelecida após a seleção dos itens para o Sprint Backlog. Assim, o Product Owner estabelece qual a direção a se seguir, e o time estipula quanto trabalho acredita ser possível ser realizado no Sprint.

No entanto, é comum o Product Owner trazer para a reunião uma ideia da Meta que esperava ver realizada ao final do Sprint, para que então seja negociada com o Time de Desenvolvimento. Dessa forma, o Time de Desenvolvimento ajuda o Product Owner a estabelecer uma meta realista e se compromete a realizá-la.

Uma vez estabelecida para o Sprint, a Meta do Sprint não deve ser modificada. Porém, caso ela perca o sentido em termos de negócios, o Sprint pode ser cancelado pelo Product Owner (veja [16.3 O Sprint pode ser cancelado?](#)).

Sprint Backlog. O Time de Desenvolvimento e o Product Owner colaboram na reunião de Sprint Planning para estabelecer o que será desenvolvido no Sprint corrente. São itens selecionados do alto do Product Backlog — portanto, os mais importantes naquele momento — para se gerar o Incremento do Produto de forma a se realizar a Meta do Sprint.

A partir desses itens selecionados, o Time de Desenvolvimento também pode criar um plano mais detalhado de como o trabalho será realizado. Esse plano é geralmente expresso a partir da quebra de cada item em tarefas e, possivelmente, de estimativas para essas tarefas.

O conjunto de itens selecionados para o Sprint, adicionados do plano de como serão desenvolvidos, é chamado de Sprint Backlog (veja [11 Sprint Backlog](#)). Na figura [17.1](#), podemos ver um exemplo de Sprint Backlog ao final de uma reunião de Sprint Planning.

O Time de Desenvolvimento monitora seu progresso, verificando o trabalho restante em cada momento do Sprint. Uma forma comum de se viabilizar

esse acompanhamento é a prática de se estimar as tarefas do Sprint Backlog usando-se alguma unidade de tempo ou tamanho, somar-se essas estimativas para determinar o trabalho restante e visualizá-lo por um Gráfico de Sprint Burndown (veja [27.3 Gráfico de Sprint Burndown](#)).

“Horas” e “tamanhos de camisa” (ou *T-Shirt Sizing*) são unidades muito utilizadas para essas estimativas. No entanto, uma prática comum consiste em apenas contar o número de tarefas restantes sem estimá-las, o que em geral fornece informação suficiente.



Fig. 17.1: Sprint Backlog recém-criado na reunião de Sprint Planning

O Time de Desenvolvimento trabalha na reunião de Sprint Planning para criar o melhor plano possível com todo o conhecimento de que dispõe nesse momento. Ainda assim, provavelmente o plano do Sprint Backlog evoluirá durante o Sprint, à medida que o Time de Desenvolvimento adquirir maior conhecimento ao executá-lo.

17.1.3 Preparação

Preparar os itens previamente à reunião de Sprint Planning pode aumentar as chances de sucesso do Sprint, ou seja, de o Time de Desenvolvimento realizar a Meta do Sprint definida. Portanto, é importante que o Product Owner e o Time de Desenvolvimento colaborem para garantir um número suficiente de itens do alto do Product Backlog preparados para a reunião de Sprint Planning. Esse trabalho de preparação pode ser realizado em sessões de Refinamento do Product Backlog durante o Sprint anterior.

É igualmente importante que se definam quais os critérios para considerar um item preparado para entrar em desenvolvimento, no que é chamado de Definição de Preparado. “Preparado” pode significar, por exemplo, que o item já tenha sido alinhado entre Product Owner e Time de Desenvolvimento, já possua Critérios de Aceitação definidos, já esteja estimado, seja pequeno o suficiente e possua detalhes suficientes e necessários para o desenvolvimento (veja [23 Refinamento do Product Backlog](#) e [14 Definição de Preparado](#)).

17.2 COMO É A SPRINT PLANNING?

A reunião de Sprint Planning é dividida em duas atividades. Na primeira, o Product Owner e o Time de Desenvolvimento se reúnem para estabelecer o que será desenvolvido no Sprint corrente. Na segunda atividade, o Time de Desenvolvimento planeja como o que foi estabelecido será desenvolvido, geralmente por meio de tarefas de desenvolvimento.

No formato tradicional da reunião de Sprint Planning, essas duas atividades são realizadas em dois momentos distintos, um após o outro, chamados não oficialmente de “Sprint Planning 1” e “Sprint Planning 2”. Entretanto, existem algumas variações para o formato tradicional da reunião de Sprint Planning.

No Planejamento Intercalado de Sprint, o Time de Desenvolvimento quebra em tarefas item a item, do alto para baixo do Product Backlog, até que julgue que já selecionou itens suficientes para o Sprint Backlog. Então, ele negocia a Meta do Sprint com o Product Owner.

No Planejamento Just-In-Time, o Time de Desenvolvimento apenas negocia e seleciona os itens para o Sprint Backlog, deixando para quebrá-los em

tarefas apenas no momento do desenvolvimento de cada um deles durante o Sprint. A seguir, descrevemos esses diferentes formatos utilizados para a reunião de Sprint Planning.

17.2.1 Sprint Planning 1 e Sprint Planning 2

O quê?

Na primeira parte da reunião de Sprint Planning, o Product Owner e o Time de Desenvolvimento colaboram, com a facilitação do ScrumMaster, para definir o que será desenvolvido no Sprint corrente. Eles escolhem, a partir do topo do Product Backlog, quais itens farão parte do Sprint Backlog e definem a Meta do Sprint.

A Sprint Planning 1 deve durar no máximo a metade do tempo previsto para toda a reunião de Sprint Planning. Recomenda-se não mais que quatro horas para Sprints de um mês, e proporcionalmente menos para Sprints menores.

O Product Owner apresenta ao Time de Desenvolvimento os itens mais importantes do Product Backlog. Ele percorre cada item, desde o topo do Product Backlog, e responde a perguntas do Time de Desenvolvimento sobre o seu conteúdo, propósito e importância, tanto individualmente quanto relacionando-o aos outros itens já descritos.

Sessões de Refinamento do Product Backlog realizadas no Sprint anterior tornam a reunião de Sprint Planning mais objetiva e eficiente, já que os itens chegam preparados para discussão (veja [23 Refinamento do Product Backlog](#)). Ainda assim, durante a reunião de Sprint Planning podem acontecer o detalhamento e a evolução de um ou mais itens em itens menores, a retirada ou inserção de algum item e reestimativas de itens que se façam necessárias, caso se usem estimativas.

É importante o Product Owner não tentar impor mais trabalho do que o Time de Desenvolvimento acredita que é capaz de realizar, seja forçando a inserção de mais itens no Sprint Backlog ou impondo a redução de estimativas. É também igualmente importante que o Product Owner entenda que é o Time de Desenvolvimento quem melhor pode dizer quanto é capaz de produzir.

Essa colaboração se encerra quando Product Owner e Time de Desenvolvimento concordam que itens além dos que já foram apresentados estarão acima de quanto o Time de Desenvolvimento acredita ser capaz de produzir. Essa decisão é tomada a partir da experiência do time com Sprints anteriores, que pode ser apoiada no cálculo da Velocidade do Time de Desenvolvimento.

Os itens selecionados fazem parte do Sprint Backlog do Sprint corrente. Essa lista de itens não gera um compromisso, mas trata-se apenas de uma previsão do Time de Desenvolvimento sobre o quanto acredita que será possível desenvolver durante o Sprint.

Uma vez que os itens estão escolhidos, o Time de Desenvolvimento e o Product Owner negociam e estabelecem a Meta do Sprint, que determina qual objetivo deve ser realizado por meio do desenvolvimento dos itens selecionados. Na prática, o trabalho de seleção de itens para o Sprint não é necessariamente uma transposição direta de itens desde o alto do Product Backlog para o Sprint Backlog. Os itens são escolhidos e negociados de forma a fazermem sentido juntos, formando em sua maioria um conjunto coerente dirigido à Meta do Sprint. Assim, algumas mudanças na priorização original do Product Backlog poderão ocorrer.

Os resultados obrigatórios da Sprint Planning 1 são os itens escolhidos para o Sprint Backlog e a Meta do Sprint. A reunião se encerra no momento em que o Time de Desenvolvimento, diante do Product Owner, se compromete com a Meta acordada, tornando-se, assim, responsável por realizá-la.

Como?

A segunda parte da reunião de Sprint Planning também dura no máximo metade do tempo previsto para toda a reunião. Durante esse tempo, o Time de Desenvolvimento planeja como será feito o desenvolvimento dos itens escolhidos para o Sprint Backlog.

Embora não haja um formato prescrito pelo Scrum, esse plano é geralmente expresso por tarefas a serem realizadas pelo Time de Desenvolvimento durante o Sprint. Dessa forma, os membros do Time de Desenvolvimento trabalham na Sprint Planning 2 percorrendo item a item entre os escolhidos para o Sprint Backlog, e quebrando cada um em um conjunto de tarefas correspondentes.

Essas tarefas representam os pequenos passos que serão seguidos para que o item esteja pronto, de acordo com a Definição de Pronto, o que sempre inclui estar de acordo com seus Critérios de Aceitação acordados. Para criá-las, o Time de Desenvolvimento reflete sobre quais serão os passos necessários para transformar o item em uma parte funcionando do produto.

Assim, os membros do Time de Desenvolvimento provavelmente observarão atentamente tanto os Critérios de Aceitação dos itens quanto a Definição de Pronto ao quebrar cada item em tarefas. Assim, eles detalharam as atividades técnicas que acreditam serem necessárias para o desenvolvimento.

As tarefas são geralmente pequenas, representando no máximo algumas horas de trabalho. Tarefas maiores que um dia de trabalho são de difícil acompanhamento e devem ser evitadas. Caso elas existam, a visibilidade que se ganharia durante a reunião de Daily Scrum seria prejudicada, já que um membro do Time de Desenvolvimento poderia informar ao resto do time que ainda está trabalhando na mesma tarefa por vários dias seguidos.

Uma vez que todos os itens selecionados para o Sprint estejam quebrados em tarefas, podem ser adicionadas estimativas para o tempo de desenvolvimento de cada tarefa. Essas estimativas possuem o único propósito de servirem ao Time de Desenvolvimento, de forma que seja possível acompanhar o progresso de seu trabalho em direção ao final do Sprint por meio de um Gráfico de Sprint Burndown (veja [27.3 Gráfico de Sprint Burndown](#)).

Diferentes unidades podem ser utilizadas para essas estimativas. A maioria dos times usa simplesmente “horas”. O principal problema com estimativas em “horas” é que o tempo estimado em horas por um desenvolvedor provavelmente seria diferente do estimado por outro para uma mesma tarefa. Por essa razão, o Time de Desenvolvimento acaba sendo forçado a estabelecer no planejamento — e, portanto, prematuramente — quem desenvolverá cada tarefa. Essa definição deveria ser realizada ao longo do Sprint e conforme necessário, já que cada tarefa pertence ao Time de Desenvolvimento como um todo, e não a um ou a outro membro com conhecimentos específicos.

Já com o uso de uma escala mais simples como “tamanhos de camisa” (*T-Shirt Sizing*), ou seja, Pequeno, Médio e Grande (P, M, G), esse problema é minimizado devido à menor precisão. Pode-se ver na figura [17.2](#) um exemplo de um trecho do Quadro de Tarefas com as tarefas estimadas em Tamanhos

de Camisas.

Alternativamente, prefiro quebrar cada item em tarefas pequenas e utilizar a contagem do número de tarefas restantes em cada dia para traçar o Gráfico de Sprint Burndown, sem a necessidade de estimativas. O resultado é muito parecido para um investimento e uma quantidade de problemas muito menores.

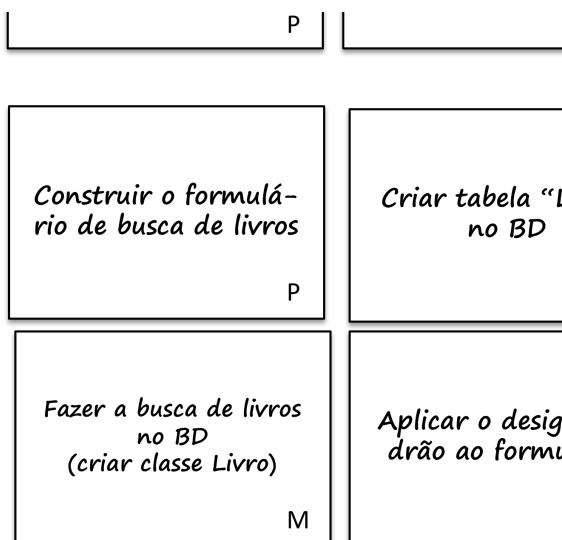


Fig. 17.2: Trecho do Quadro de Tarefas com estimativas em Tamanhos de Camisas

É importante destacar que todos os membros do Time de Desenvolvimento participam com poder igual de opinião e decisão sobre a quebra dos itens em tarefas e das estimativas, se utilizadas.

O trabalho de definição e estimativa das tarefas não deve ser exaustivo nem completo. O Time de Desenvolvimento o faz da melhor forma possível, com as informações e conhecimento que tem sobre os itens no momento da reunião. Inevitavelmente, à medida que o Time de Desenvolvimento avança no Sprint e entende melhor o trabalho que está realizando, novas tarefas sur-

girão para os itens do Sprint Backlog, outras não mais serão necessárias e desaparecerão, e estimativas, caso presentes, serão modificadas.

A participação do Product Owner na Sprint Planning 2 não é obrigatória. No entanto, é altamente recomendado que, no mínimo, ele fique acessível e disponível para o Time de Desenvolvimento. O Product Owner pode ser requisitado pois, durante a reunião, dúvidas sobre os itens e sobre a Meta do Sprint invariavelmente surgirão, e elas devem ser sanadas o mais rapidamente possível.

Ao final da Sprint Planning 2, o Sprint Backlog inicial estará pronto, contendo os itens escolhidos na Sprint Planning 1 e o plano de como esses itens serão desenvolvidos. Pode ser interessante nesse momento que o Time de Desenvolvimento já crie o Gráfico de Sprint Burndown, caso ele seja usado.

17.2.2 Planejamento Intercalado de Sprint

Também chamado de Planejamento Baseado em Compromisso, o Planejamento Intercalado é uma alternativa ao formato oficial da reunião de Sprint Planning a se considerar, pois traz algumas vantagens concretas.

Em um Planejamento Intercalado, Product Owner e Time de Desenvolvimento trabalham juntos por toda a reunião, facilitados pelo ScrumMaster. Observe que, ao contrário do que ocorre no formato tradicional da reunião de Sprint Planning, o Product Owner deve obrigatoriamente estar presente durante toda a reunião. Métricas como Velocidade e estimativas não são usadas, apenas a experiência, o conhecimento e um acordo entre os envolvidos.

As partes “o quê” e “como” do Sprint Backlog são definidas de forma intercalada ao longo da reunião. Ou seja, o que será desenvolvido e como será desenvolvido não estão mais separados.

Seleciona-se o item mais importante do Product Backlog (ou seja, aquele que está no topo), que então é lido em voz alta. O Product Owner tira dúvidas sobre o item até que o Time de Desenvolvimento esteja confortável com sua própria compreensão do item. Esse é geralmente um trabalho rápido se o Product Backlog estiver previamente refinado (veja [23 Refinamento do Product Backlog](#)).

Então, o Time de Desenvolvimento realiza as discussões técnicas necessárias e estabelece um plano de como o item será desenvolvido, o que geral-

mente é expresso por tarefas a serem realizadas durante o Sprint. O item e seu plano são adicionados ao Sprint Backlog.

Segue-se para o item seguinte, que também é lido, discutido e quebrado em tarefas (ou em alguma outra forma de plano). Ao fazê-lo, o Time de Desenvolvimento tem uma maior compreensão sobre o desenvolvimento do item, e pode decidir melhor se vai adicioná-lo ao Sprint Backlog ou não.

Essa decisão é basicamente baseada no que o Time de Desenvolvimento pode compreender naquele momento sobre o item e em experiências passadas. Caso decida adicioná-lo, segue-se para o item seguinte, até que o Time de Desenvolvimento julgue que já selecionou itens suficientes para o Sprint.

Em seguida, o Product Owner e o Time de Desenvolvimento, tendo em vista os itens selecionados, vão negociar e estabelecer a Meta do Sprint.

PLANEJAMENTO INTERCALADO DE SPRINT: PASSOS

- 1) O próximo item mais importante (mais ao alto) do Product Backlog é lido, e dúvidas do Time de Desenvolvimento são tiradas pelo Product Owner;
- 2) o Time de Desenvolvimento gera um plano de como o item será desenvolvido, geralmente quebrando-o em tarefas;
- 3) a partir das tarefas, o Time de Desenvolvimento decide se acredita que desenvolver o item durante o Sprint está dentro da sua capacidade de trabalho;
- 4) em caso positivo, o item é removido do Product Backlog e adicionado ao Sprint Backlog, que está sendo criado para o Sprint, e volta-se ao passo 1. Em caso negativo, o item não é selecionado e segue-se para o passo 5;
- 5) Product Owner e Time de Desenvolvimento estabelecem a Meta do Sprint.

Há diversas vantagens nessa abordagem. As principais são:

- o resultado é, a princípio, a melhor previsão possível que o Time de Desenvolvimento pode dar naquele momento, já que a produzirá sobre um plano mais detalhado;
- a reunião é mais dinâmica, já que a participação de todos é ativa o tempo todo e as atividades são mais variadas, uma vez que se intercalam partes diferentes do planejamento. O dinamismo pode levar a um maior engajamento dos participantes;
- há um maior foco dos membros do Time de Desenvolvimento. Ao se iniciar a discussão sobre o que é o item, alguns participantes já começam a pensar em como desenvolvê-lo. Ao não criarem esse plano imediatamente, poderão ter maior dificuldade em manter seu foco ao seuirem na discussão dos próximos itens, o que muitas vezes ocorre ao se utilizar o formato tradicional da reunião;
- obtém-se uma melhor compreensão conjunta sobre cada item, já que Time de Desenvolvimento e Product Owner trabalham juntos do início ao fim da reunião;
- a colaboração mais próxima entre Product Owner e Time de Desenvolvimento aumenta o espírito de equipe e a responsabilidade conjunta sobre o Sprint.

O Planejamento Intercalado pode também ser útil quando apenas usado para se iniciar o projeto. Nesse momento, o Time de Desenvolvimento pode não ter noção da sua capacidade de trabalho, caso não venha trabalhando junto desde outro projeto.

Nessa primeira reunião de Sprint Planning, o Time de Desenvolvimento não tem parâmetros para determinar quanto trabalho vai aceitar para o Sprint Backlog, e o Planejamento Intercalado pode ser útil. Já a partir do terceiro ou quarto Sprints, o volume médio de trabalho completado pelo time pode trazer uma melhor noção dessa capacidade, facilitando o planejamento.

A Velocidade do Time de Desenvolvimento é uma forma de se representar essa capacidade (veja [26.7 Velocidade do Time de Desenvolvimento](#)). Com

o histórico gerado após alguns Sprints, o time já está apto a medi-la. Pode-se dessa forma realizar o Planejamento Intercalado nos primeiros Sprints e utilizar-se da quantidade de trabalho entregue em cada um deles para se calcular a Velocidade. Depois desses primeiros Sprints, ou quando a Velocidade se estabilizar, pode-se passar a usar o formato de planejamento tradicional, em duas partes.

17.2.3 Planejamento Just-In-Time de Sprint

A máxima do Planejamento *Just-In-Time* é a redução do desperdício. Esse formato considera que o melhor momento para se realizar o planejamento de como os itens serão desenvolvidos é imediatamente antes do início do desenvolvimento de cada item. Ou seja, o último momento possível.

Se o planejamento de todos os itens for realizado na reunião de Sprint Planning, o Time de Desenvolvimento pode não ter dados suficientes para criar um bom plano. Mudanças significativas no plano terão de ser feitas no decorrer do Sprint, o que caracterizará o desperdício.

Na reunião de Sprint Planning de um Planejamento *Just-In-Time*, o Time de Desenvolvimento e o Product Owner apenas selecionam os itens do Product Backlog para o Sprint Backlog e criam a Meta do Sprint (“o quê?”). Sua execução é similar ao Sprint Planning 1, descrito na seção anterior. Assim, nessa reunião não é definido o plano de como será realizado o desenvolvimento dos itens escolhidos para o Sprint Backlog (“como?”).

Após a reunião de Sprint Planning, planeja-se acerca do primeiro e mais importante item do Product Backlog, em geral quebrando-o em tarefas a serem realizadas pelo Time de Desenvolvimento. O item é desenvolvido e, quando estiver pronto, de acordo com a Definição de Pronto, o Time de Desenvolvimento seguirá para planejar acerca do item seguinte, o desenvolverá e seguirá assim pelos outros itens do Sprint Backlog.

Times com maior maturidade podem fazer melhor uso do Planejamento *Just-In-Time*. No entanto, embora colabore significativamente para a redução do desperdício, essa modalidade de planejamento pode aumentar os riscos de surpresas perigosas no Sprint, uma vez que detalhes importantes surgirão apenas no seu decorrer. Já quando o planejamento é realizado no início do

Sprint, problemas podem ser identificados cedo e, então, tratados imediatamente.

CAPÍTULO 18

Daily Scrum

- **Objetivo:** planejar o próximo dia de desenvolvimento
- **Quando:** em cada dia de desenvolvimento do Sprint
- **Duração:** máxima de 15 minutos
- **Participantes obrigatórios:** Time de Desenvolvimento
- **Saídas esperadas:** plano informal para o próximo dia de trabalho

18.1 O QUE É A DAILY SCRUM?

A Daily Scrum é uma reunião curta realizada diariamente pelo Time de Desenvolvimento. Ela tem a duração máxima ou o *timebox* de quinze minutos, e acontece preferencialmente no mesmo local e na mesma hora, para que o time se acostume a realizá-la.

A reunião de Daily Scrum é essencial para a auto-organização do Time de Desenvolvimento. Ela tem os objetivos de proporcionar, entre os membros do Time de Desenvolvimento, visibilidade no trabalho realizado e a realizar, promover a comunicação sobre esse trabalho, dar visibilidade a quais obstáculos estão atrapalhando o trabalho e servir de oportunidade para decisões rápidas com relação ao progresso do Sprint. Assim, essa reunião produz um plano informal para o próximo dia de trabalho do Time de Desenvolvimento, ou seja, para até a próxima reunião de Daily Scrum.

18.2 COMO É A DAILY SCRUM?

Existe um padrão utilizado por Times de Desenvolvimento para endereçar essas questões na reunião de Daily Scrum. Cada membro se dirige a seus colegas e responde a três perguntas:

- *O que eu fiz desde a última reunião de Daily Scrum (para ajudar o Time de Desenvolvimento a realizar a Meta do Sprint)?*
- *O que eu pretendo fazer até a próxima reunião de Daily Scrum (para ajudar o Time de Desenvolvimento a realizar a Meta do Sprint)?*
- *Que impedimentos estão em meu caminho (atrapalhando o Time de Desenvolvimento a realizar a Meta do Sprint)?*

As perguntas têm o propósito estimular a reflexão e criar alinhamento sobre o trabalho a ser realizado até a próxima reunião de Daily Scrum, mantendo-se o foco na Meta do Sprint. Pode-se também adicionar à última questão quais impedimentos foram resolvidos desde a última Daily Scrum.

O ScrumMaster participa, quando necessário, como um facilitador na reunião de Daily Scrum. Ele ajuda o Time de Desenvolvimento a manter o foco e a utilizar apenas os quinze minutos previstos, o que pode ser particularmente importante para times com pouca maturidade no uso de Scrum. Sua presença na reunião, no entanto, não é obrigatória.

O Product Owner, em geral, não participa da reunião de Daily Scrum, a menos que seja convidado pelo Time de Desenvolvimento.

A reunião de Daily Scrum é também conhecida como Daily Meeting (reunião diária) ou Stand-up Meeting (reunião em pé). Esta última designação provém do Extreme Programming, uma metodologia Ágil de desenvolvimento de *software*, e indica uma prática potencialmente útil: a realização da reunião com todos os participantes de pé, com o objetivo de não permitir que eles se sintam confortáveis o suficiente para quebrar o *timebox*. Porém, cabe destacar que não há obrigatoriedade no Scrum de se realizar essa reunião em pé.

Discussão: Product Owner na Daily Scrum

A princípio, o Product Owner não tem muito o que fazer na reunião de Daily Scrum. Ela é uma reunião do Time de Desenvolvimento para o Time de Desenvolvimento. Ao planejarem o trabalho, os membros do time passarão pelas tarefas realizadas e a realizar, o que deve ser desinteressante para a maioria dos Product Owners dado o teor técnico da conversa.

Um Product Owner presente na reunião pode também ter a intenção de praticar o microgerenciamento do trabalho do time, buscando uma maior sensação de controle a partir da visibilidade de detalhes do dia a dia no Sprint. Esse comportamento de “chefe” é altamente disfuncional e contrário aos princípios do Scrum.

Mesmo um Product Owner bem-intencionado pode trazer problemas para a reunião. Já observei reuniões em que os membros do time, talvez por desconhecimento ou falta de maturidade, respondiam às três perguntas dirigindo-se ao Product Owner, acabando por fazer uma reunião de status.

Em outros casos, no entanto, observei Product Owners presentes no dia a dia do Time de Desenvolvimento que participavam da Daily Scrum de forma benéfica e natural, mostrando-se disponíveis para tirar dúvidas (o que deve ser feito após a reunião) e buscando incentivar um espírito de equipe com sua presença e colaboração.

18.3 O QUE NÃO DEVE ACONTECER NA DAILY SCRUM?

A reunião de Daily Scrum é importante para ajudar o Time de Desenvolvimento a organizar e a acompanhar seu trabalho durante o Sprint. Entretanto, há diversas disfunções comuns que podem atrapalhar a sua execução e reduzir o seu valor. Listo algumas delas nesta seção.

Discussão: Daily Scrums disfuncionais

O ScrumMaster olha no relógio. É hora da Daily Scrum. O time não se mexe. O ScrumMaster pensa: *ai meu deus, lá vou eu de babá de novo*. Puxa um por um pelo braço e encontra resistência. *Vamos lá, pessoal, hora da Daily Scrum. Vamos!*

O Time de Desenvolvimento entra na sala de reunião. Eles já conhecem bem a rotina. Quinze minutos. Todos de pé. Responder às três perguntas. O primeiro puxa a palavra e começa, monotônico: *desde ontem fiz isso, isso e aquilo, até amanhã vou fazer aquilo, aquilo e aquilo e não há impedimentos*. O próximo membro do time vai pela mesma linha, quase como um robô: ...e *não há impedimentos*. E o próximo. E o próximo.

Enquanto isso, outros membros do time, desinteressados, consultam o celular ou simplesmente viajam, quase dormem. Ninguém entende bem o que está fazendo ali. A reunião é muito chata e sem propósito. Afinal, cada um faz apenas a sua parte no dia a dia do trabalho e, ao não dividirem responsabilidades, não se interessam pelo que os outros estão fazendo.

Na sala ao lado, outro time e seu ScrumMaster também estão fazendo a Daily Scrum. Cada membro do time se vira para o ScrumMaster e responde às três perguntas. O ScrumMaster faz caras e bocas, comenta, sugere e anota. A reunião termina e o ScrumMaster está preocupado com o andamento da Sprint. Pensa em como ele vai cobrar mais empenho do time para que os itens planejados estejam completos ao final da Sprint.

Por fim, mais para o final do dia, um terceiro time termina a Daily Scrum com quarenta e poucos minutos. Eles se perderam em discussões técnicas e esclarecimentos de questões de negócios. Parecia que eles queriam resolver todos os seus problemas na reunião. Alguns debates foram bem acalorados e completamente infrutíferos. E isso é uma constante para esse time.

Mostrei aqui alguns cenários de Daily Scrums bastante disfuncionais. O fato é que a Daily Scrum não é uma reunião de status, e não é uma reunião de prestação de contas, nem entre os membros do time, nem para o ScrumMaster, nem para o Product Owner. Também não é uma reunião de resolução de problemas.

A Daily Scrum é uma reunião de **planejamento**. O time, preferencialmente em frente ao seu Quadro de Tarefas, planeja o que vai fazer até o próximo dia de trabalho, até a próxima Daily Scrum.

Algo como: *eu vou fazer essa tarefa aqui. E eu vou fazer aquela. Vamos puxar essa juntos? Não, essa aqui está impedida, mas o ScrumMaster já está resolvendo.* O Time de Desenvolvimento conversa e cria visibilidade sobre o seu trabalho para o próprio time, de forma a conseguir se planejar. É claro, o time tira muito mais benefício e tem muito mais interesse na reunião se de fato trabalhar como time. Ou seja, todos juntos, a partir do item mais prioritário.

E as famosas três perguntas? Elas podem ser um bom guia para times que estão começando com Scrum. Mas elas são o meio, e não o objetivo. Já vi muitos times maduros ignorando essas perguntas-padrão e tirando muito mais benefício da reunião.

18.3.1 Disfunção: informe de impedimentos ao ScrumMaster

Informar impedimentos ao ScrumMaster não é um dos objetivos da reunião de Daily Scrum. Informa-se o impedimento ao ScrumMaster assim que ele é identificado, de forma que possa ser tratado o mais rapidamente possível. O objetivo de se informar, durante a reunião, quais impedimentos surgiram desde a última Daily Scrum é apenas o de dar visibilidade a todos os membros do Time de Desenvolvimento sobre o que está atrapalhando o trabalho.

Ao se deparar com um impedimento, é um erro comum o membro do Time de Desenvolvimento aguardar até a reunião de Daily Scrum para solicitar a ajuda do ScrumMaster. Esse comportamento acaba por atrasar a sua resolução, que pode atrapalhar por mais tempo do que o necessário o trabalho do Time de Desenvolvimento, e ameaçar que se consiga realizar a Meta do Sprint.

Outra questão é que, ao informar um impedimento para o ScrumMaster, o membro do Time de Desenvolvimento fornece a ele o máximo de deta-

lhes possível, de forma a ajudá-lo na remoção do impedimento. Essa é uma atividade que consome tempo, e isso poderia comprometer o *timebox* de 15 minutos da reunião ou atrapalhar o seu andamento.

18.3.2 Disfunção: reunião de trabalho

A reunião de Daily Scrum não deve ser transformada em reunião de trabalho. Os membros do Time de Desenvolvimento se limitam a informar sucintamente seus colegas sobre seu trabalho realizado e definir o trabalho a realizar. Ao se dar visibilidade sobre o trabalho, é comum surgirem assuntos relacionados, como discussões técnicas ou dúvidas de negócios, e é natural o Time de Desenvolvimento querer aproveitar para discuti-los.

Entretanto, esse tipo de desvio deve ser evitado para que o foco da reunião seja mantido. Contudo, é normal nessas situações ser agendada uma reunião de trabalho para acontecer logo após a reunião diária, contando inclusive com a presença do Product Owner, quando necessário.

18.3.3 Disfunção: prestação de contas a outros

A reunião de Daily Scrum não deve servir como instrumento de cobrança externa sobre o Time de Desenvolvimento. Nesta reunião, cada membro do Time de Desenvolvimento não está prestando contas do seu trabalho para o ScrumMaster, para o Product Owner ou para qualquer parte interessada do projeto, mas sim apenas para os outros membros do Time de Desenvolvimento.

Infelizmente, não é incomum ver o ScrumMaster ou o Product Owner repassando a lista de atividades e perguntando a cada membro do Time de Desenvolvimento o que cada um fez, o que pretende fazer e quais são os impedimentos. O Time de Desenvolvimento, na verdade, não responde por suas tarefas ao ScrumMaster ou ao Product Owner, e a responsabilidade sobre a reunião de Daily Scrum é somente do Time de Desenvolvimento. O ScrumMaster pode estar presente na reunião de Daily Scrum apenas como facilitador.

18.3.4 Disfunção: falta de interesse e atenção

Todos os membros do Time de Desenvolvimento devem manter o foco e prestar total atenção ao que seus colegas estão relatando durante a reunião de Daily Scrum. Nenhum outro trabalho ou conversa paralela devem acontecer durante a reunião. Para que se alcance esse nível de concentração, recomenda-se que conversas paralelas, telefones e computadores abertos sejam evitados durante a reunião.

O trabalho realizado no Sprint pertence ao Time de Desenvolvimento como um todo. Não há uma divisão de responsabilidades por área de conhecimento ou senioridade, de forma que todos são igualmente responsáveis pela execução e pelos resultados do trabalho. Quando essa divisão acontece, é comum o baixo interesse de membros do time no trabalho de seus colegas e, assim, a reunião de Daily Scrum torna-se ineficiente.

CAPÍTULO 19

Sprint Review

- **Objetivo:** obter *feedback* sobre o Incremento do Produto desenvolvido no Sprint (inspeção e adaptação do produto)
- **Quando:** no último dia de cada Sprint, antes da reunião de Sprint Retrospective
- **Duração:** máxima proporcional a 4 horas para Sprints de 1 mês
- **Participantes obrigatórios:** clientes do projeto, Time de Desenvolvimento, Product Owner e ScrumMaster, podendo também estar presentes usuários e quaisquer outras partes interessadas que possam prover *feedback*
- **Saídas esperadas:** *feedback* como matéria-prima para o Product Owner atualizar o Product Backlog, visibilidade sobre o produto para clientes e demais partes interessadas

19.1 O QUE É A SPRINT REVIEW?

Na reunião de Sprint Review (ou revisão do Sprint), o Time de Desenvolvimento e o Product Owner trabalham em conjunto, com a facilitação do ScrumMaster, para demonstrar o trabalho pronto produzido durante o Sprint, para clientes do projeto e demais partes interessadas. Seu principal objetivo é a obtenção de *feedback* antecipado dessas pessoas sobre o Incremento do Produto produzido, já que um *feedback* mais concreto e profundo só poderá ser obtido após a entrega para seus usuários.

O Product Owner utilizará o *feedback* obtido nessa reunião como matéria-prima para modificar o Product Backlog para Sprints futuros. É, portanto, uma reunião de inspeção e adaptação do produto.

As presenças do Time de Desenvolvimento, do Product Owner e do ScrumMaster são obrigatórias nessa reunião. As demais pessoas convidadas podem ser clientes, usuários, gerentes e outros para os quais as funcionalidades desenvolvidas no Sprint são relevantes, e cujo *feedback* é considerado importante.

A reunião de Sprint Review acontece no último dia do Sprint, antes da reunião de Sprint Retrospective. Ela tem a duração máxima de quatro horas para Sprints de um mês, ou proporcionalmente menos para Sprints mais curtas.

Discussão: aprovação ou feedback?

A reunião de Sprint Review começa. O Time de Desenvolvimento e o Product Owner apresentam para os clientes o que fizeram nesse Sprint. Eles observam, mas pouco falam, exceto por algumas poucas perguntas básicas. Ao final, aprovam (e até mesmo aplaudem) e se despedem.

Todos pensam: *a reunião foi um sucesso*. Mas será que foi mesmo? Na realidade, acredito que esse seja um dos piores cenários possíveis para uma reunião de Sprint Review. Pior que isso, talvez, só se não houver clientes presentes.

Será que esses clientes entenderam o que lhes foi demonstrado? Será que realmente se importaram com o que estavam vendo? Eles certamente vão se importar no futuro, possivelmente quando usuários usarem o *software* e

notarem que não atende bem às suas necessidades.

O propósito da reunião de Sprint Review não é o de se obter a aprovação formal dos clientes sobre o que foi feito no Sprint, ou seja, polegar para cima ou carimbo de “aceito” no contrato. Não é uma sessão de testes de aceitação, tampouco. A aprovação para se concretizar uma entrega deve ser feita em outro momento, fora do contexto do Scrum.

O objetivo da reunião de Sprint Review é de se obter *feedback* do cliente sobre o Incremento do Produto gerado no Sprint e, com isso, poder frequentemente fazer ajustes de direção, diminuindo os riscos do projeto. É trabalho — e obrigação — do Time de Desenvolvimento e do Product Owner puxarem esse *feedback* dos clientes e demais partes interessadas presentes. Convidá-los a usarem o produto ali mesmo. Instigar. Fazer perguntas. Mostrar alternativas.

Algum cliente achou que algo não estava exatamente como ele queria? Excelente! Deixemos a postura defensiva de lado. Não tenhamos medo. Nós não fizemos errado. Não estragamos tudo. Na realidade, já esperávamos por isso. Faremos de tudo para acertar, claro, mas não é possível ler a mente de ninguém. E, mesmo que fosse, isso de nada adiantaria, pois o cliente só saberá exatamente o que ele precisa após ver algo pronto. O produto, na cabeça do cliente, é construído aos poucos, incrementalmente.

Mesmo quando der tudo errado e clientes entenderem que tudo o que foi feito no Sprint não serve para nada, pelo menos obteremos esse *feedback* antes de gastarmos meses trabalhando naquilo. Gestão de riscos pura, não?

Ou seja, o espírito da Sprint Review não é:

— *Cliente, o que fizemos está aprovado?*

Mas talvez algo assim:

— *Cliente, agora que você está tendo a oportunidade de ver funcionando (e experimentar!) esse Incremento do Produto que fizemos para você nesse Sprint, o que podemos modificar ou adicionar a ele para melhor atender às suas necessidades?*

19.2 COMO É A SPRINT REVIEW?

19.2.1 Preparação

Não é necessário haver grandes preparações para essa reunião. A demonstração realizada para os clientes e demais partes interessadas é informal, e deve manter seu foco inteiramente no produto funcionando, ou seja, no Incremento do Produto gerado.

É interessante, no entanto, haver ao menos uma pequena sessão no final do trabalho do Sprint, na qual Time de Desenvolvimento e Product Owner alinharão o entendimento sobre o que ocorreu durante o Sprint, e definirão o que será apresentado e quem fará essa demonstração.

19.2.2 Gestão de expectativas

É saudável que Product Owner e Time de Desenvolvimento se preocupem com a gestão das expectativas dos clientes e demais partes interessadas que estarão presentes. Ao entenderem claramente qual é a Meta do Sprint e qual é a capacidade de trabalho do Time de Desenvolvimento, por exemplo, essas pessoas vão compreender melhor o que o Time de Desenvolvimento foi capaz de produzir no Sprint.

Assim, como parte da preparação para o encontro, pode ser interessante o Product Owner deixar seus objetivos claros ao realizar o convite para as pessoas que quer que participem da reunião. Da mesma forma, antes do início da demonstração, o Time de Desenvolvimento eo Product Owner podem informá-los qual era a meta a ser realizada nesse Sprint (veja [24.2 Meta de Sprint](#)).

19.2.3 Quem participa

Além do ScrumMaster e Product Owner, todos os membros do Time de Desenvolvimento estão presentes na reunião de Sprint Review. Afinal, eles têm igual responsabilidade sobre os resultados de seu trabalho.

A presença de pessoas relevantes na reunião cria a oportunidade para valiosos *feedbacks* sobre o produto. O *feedback* reduz os riscos do projeto ao conformar os rumos dos próximos Incrementos do Produto de acordo com

os objetivos e necessidades dos clientes e demais partes interessadas. Eles podem vislumbrar com mais clareza ao verem (e, talvez, experimentarem) o Incremento do Produto pronto e funcionando. Por essa razão, pode ser ainda mais interessante haver usuários reais entre os presentes.

Em alguns casos, infelizmente, os únicos participantes da reunião de Sprint Review são o Time de Desenvolvimento, que faz a demonstração, e o Product Owner, que fornece o *feedback* sobre o que foi produzido. Nesse cenário, confiança em excesso poderá estar sendo depositada na representatividade do Product Owner, pois ele será o único a ver o Incremento do Produto pronto, e a ser capaz de dar *feedback* sobre ele.

Assim, as pessoas para as quais o produto desse Sprint é relevante somente poderão fornecer algum tipo de *feedback* após a Release, o que aumenta consideravelmente os riscos do projeto.

O ScrumMaster possui o importante papel de facilitar essa reunião, mas ele não participa diretamente na apresentação do que foi feito, nem dá suas opiniões sobre o Incremento do Produto gerado. Esse simplesmente não é o seu papel.

19.2.4 Apresentação

A apresentação do que foi feito é realizada a partir de uma colaboração entre o Time de Desenvolvimento e o Product Owner. Esse comportamento reflete uma proximidade e colaboração saudáveis acontecendo no trabalho de ambos. No entanto, não é errado apenas o Product Owner apresentar para os presentes o que foi feito durante o Sprint, com os membros do Time de Desenvolvimento dando o apoio necessário. Ou, de outra forma, apenas o Time de Desenvolvimento ou alguns de seus membros realizarem a demonstração, com o apoio do Product Owner ao seu lado ou junto aos outros presentes.

Uma vez que o foco da apresentação é colocado na Meta realizada, o Incremento do Produto pode ser demonstrado como um todo. Alternativamente, muitos times apresentam e demonstram os itens desenvolvidos durante o Sprint um a um, do mais importante ao menos importante. Clientes e demais presentes trabalham colaborativamente com o Time de Desenvolvimento e com o Product Owner, fazendo perguntas e obtendo respostas sobre

o que lhes está sendo demonstrado, e apresentando suas ideias sobre o que esperam do produto nos próximos Sprints.

É uma excelente prática convidar os presentes a experimentarem diretamente o produto. Isso os estimulará a fornecerem *feedback* e permitirá que este seja mais profundo e preciso. No entanto, a Sprint Review não é uma reunião para testes do produto e, assim, não deve ser utilizada para substituir práticas de testes que devam ser realizadas ao longo do Sprint.

É igualmente interessante perguntar aos presentes o que esperam ver pronto nas próximas reuniões de Sprint Review. Ou seja, estimulá-los a elaborar sobre quais são as próximas necessidades de negócios mais importantes a serem atendidas.

19.2.5 O que é apresentado

O Incremento do Produto funcionando, ou seja, o valor gerado para os clientes do projeto, é o foco da demonstração. Evitam-se *slides*, documentos que não fazem parte do produto, explicações excessivas sobre planos ou sobre o processo de desenvolvimento, intenções não realizadas, desculpas e justificativas.

Da mesma forma, questões técnicas somente serão discutidas se forem de interesse das pessoas presentes. Caso contrário, há o risco de que considerem a reunião uma perda de tempo, o que dificultará o seu comparecimento nas próximas.

Apresentam-se apenas os itens do Sprint Backlog que estejam prontos, de acordo com a Definição de Pronto.

19.2.6 Resultados

A partir do que foi e do que não foi gerado no Sprint, o Product Owner e Time de Desenvolvimento esperam que se tenha realizado a Meta do Sprint. Ou seja, que o problema proposto para o Sprint tenha sido resolvido.

É importante observar que, para realizar essa Meta, o Time de Desenvolvimento não necessariamente deverá ter completado todos os itens planejados. Salvo tenha havido algum impedimento prevenindo um item importante

de ser desenvolvido, geralmente os itens não prontos ao final do Sprint são os de mais baixa importância para se realizar a Meta do Sprint.

Na figura 19.1, pode-se ver um Quadro de Tarefas representando o Sprint Backlog de um Sprint possivelmente bem-sucedido, apesar de um dos itens não ter sido completado.

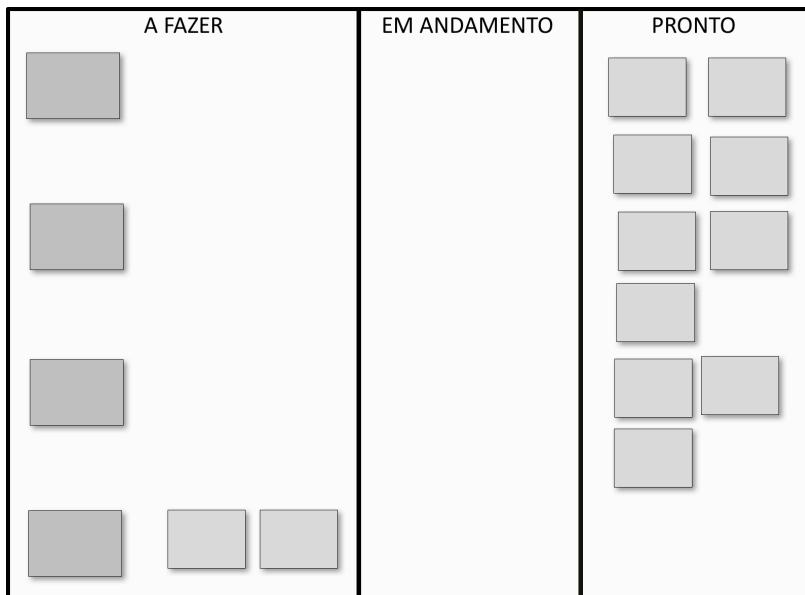


Fig. 19.1: Quadro de Tarefas (Sprint Backlog) ao final de um Sprint

O *feedback* obtido a partir da interação entre Time de Desenvolvimento, Product Owner e demais presentes na reunião de Sprint Review é usado pelo Product Owner como matéria-prima para adicionar, remover ou modificar itens do Product Backlog. É dessa forma que o produto é construído incrementalmente para melhor atender às necessidades dos seus clientes.

Caso haja, na reunião de Sprint Review, itens do Sprint não prontos de acordo com a Definição de Pronto, eles podem retornar ao Product Backlog e reaparecerem no próximo ou em algum dos próximos Sprints. Pode-se também decidir que sejam eliminados, se assim fizer sentido a partir do *feedback*.

obtido ou de outras questões de negócio. É papel do Product Owner, e apenas dele, decidir o destino desses itens.

[Discussão: foi ou não foi?]

Em uma visão mais tradicional do Scrum, o Product Owner verifica na reunião se cada item planejado e apresentado está de fato pronto, de acordo com a Definição de Pronto. A partir do que foi e não foi gerado no Sprint, o Product Owner estabelece, em frente aos presentes, se o Time de Desenvolvimento conseguiu ou não realizar a Meta do Sprint.

Para mim, um Product Owner ter alguma surpresa na Sprint Review é um indicador de problema, que precisa ser tratado urgentemente na próxima reunião de Sprint Retrospective. Um bom Product Owner colabora com o Time de Desenvolvimento durante todo o Sprint e tem uma excelente visão do que está acontecendo. Surpresas são exceções.

Dessa forma, em vez de usar a Sprint Review para um julgamento do trabalho do time por parte do Product Owner e, talvez, dos clientes e demais partes interessadas presentes, prefiro usar a reunião para que os presentes colaborarem de modo a se obter *feedback* sobre o Incremento do Produto gerado.

CAPÍTULO 20

Sprint Retrospective

- **Objetivo:** melhoria incremental contínua na forma como o Time de Scrum faz o seu trabalho (inspeção e adaptação)
- **Quando:** no último dia de cada Sprint, após a reunião de Sprint Review
- **Duração:** máxima proporcional a 3 horas para Sprints de 1 mês
- **Participantes obrigatórios:** Time de Desenvolvimento, Product Owner e ScrumMaster
- **Saídas esperadas:** planos de ação para melhorias a serem realizados já no próximo Sprint

20.1 O QUE É A SPRINT RETROSPECTIVE?

Em intervalos regulares, a equipe reflete sobre como se tornar mais efetiva e, então, refina e ajusta seu comportamento de acordo — Princípio Ágil.

20.1.1 Lições aprendidas em times tradicionais

Os acertos e erros em projetos que utilizam métodos tradicionais são geralmente levantados e discutidos apenas uma vez ao final do projeto, na esperança de que essas lições aprendidas possam ser úteis em trabalhos futuros.

A reunião de Lições Aprendidas ou *postmortem*, por exemplo, busca cumprir esse papel, realizando uma verdadeira autópsia no projeto sem, no entanto, ter contribuído em nada para a sua saúde enquanto o projeto ainda estava vivo. Além de não ser útil para o próprio projeto, apenas uma pequena parte dessas lições pode ser aproveitada, se tanto, pois novos projetos envolvem novas condições, pessoas e desafios.

20.1.2 Melhoria incremental contínua em times Ágeis

Times que utilizam Scrum estão sempre buscando melhores formas de fazer seu trabalho, aprendendo com seus acertos e erros ao longo de todo o projeto. O objetivo da reunião de Sprint Retrospective (ou retrospectiva do Sprint) é estimular o Time de Scrum a realizar essa prática, que é chamada de melhoria incremental contínua.

O princípio Ágil citado no início deste capítulo reflete exatamente isso: o Time de Scrum, ou seja, Time de Desenvolvimento, Product Owner e Scrum-Master se reúnem periodicamente com os objetivos de identificar pontos de melhoria em como fazem seu trabalho (inspeção) e de traçar planos de ação para executar essas melhorias (adaptação).

Por buscar a geração de melhorias já no Sprint seguinte do mesmo projeto, a prática da retrospectiva é muito mais objetiva e eficiente do que a prática de se avaliar as lições aprendidas somente ao final do projeto.

20.1.3 A retrospectiva do Sprint

Na reunião de Sprint Retrospective, o Time de Scrum inspeciona o Sprint que está se encerrando quanto a seus processos de trabalho, dinâmicas, pessoas, relacionamentos, comportamentos, práticas, ferramentas utilizadas e ambiente, e planeja as melhorias necessárias.

Facilitados pelo ScrumMaster, o Time de Desenvolvimento e o Product Owner identificam o que foi bem, e que por essa razão se quer manter no próximo Sprint, e o que se pode melhorar. Eles buscam as causas raízes dos problemas enfrentados no período, e traçam planos de ação com formas práticas para se realizarem as melhorias.

A reunião é uma colaboração e nunca deve se configurar como um espaço para trocas de acusações ou discussões improdutivas.

20.2 COMO É A SPRINT RETROSPECTIVE?

20.2.1 Regularidade, frequência e duração

O Time de Scrum realiza a reunião de Sprint Retrospective no último dia de todo e cada Sprint do projeto, logo após a reunião de Sprint Review, com o objetivo de garantir a melhoria incremental contínua. A obrigatoriedade dessa prática visa a assegurar regularidade e frequência na realização dessas melhorias, dando oportunidades ao Time de Scrum de promover mudanças para corrigir problemas antes que estes afetem negativamente os resultados de seu trabalho em Sprints futuros.

Em projetos com prazos curtos demais e ritmo de trabalho consequentemente acelerado, é comum o Time de Desenvolvimento sofrer pressões internas e externas para suprimir quaisquer esforços que não façam parte do trabalho de desenvolvimento propriamente dito. Nesse cenário, a reunião de Sprint Retrospective é uma das primeiras a ser repetidamente cancelada, e até mesmo definitivamente descartada. Sem a reunião, o Time de Scrum dificilmente realiza a inspeção e a adaptação. Assim, sequer consegue avaliar o porquê de não ser capaz de realizar as tarefas em um ritmo sustentável, fechando-se em um círculo vicioso.

Como guardião das práticas do Scrum, é papel do ScrumMaster estimu-

lar o Time de Scrum a realizar as retrospectivas com a regularidade exigida pelo *framework*. A reunião de Sprint Retrospective deve durar no máximo três horas para Sprints de um mês, ou proporcionalmente menos para Sprints menores.

20.2.2 Participação do ScrumMaster

Facilitação

O ScrumMaster atua como um facilitador na reunião de Sprint Retrospective, trabalhando junto ao Time de Scrum para minimizar as dificuldades de comunicação e de colaboração, ingredientes necessários para a geração de ideias. A partir das questões levantadas nessa reunião, o ScrumMaster incentiva o Time de Scrum a encontrar soluções para trabalhar com qualidade e tornar-se cada vez mais efetivo, buscando sempre manter um ritmo sustentável de trabalho.

A variação na forma é, em geral, positiva para o resultado das reuniões de Sprint Retrospective. Assim, nesse trabalho de facilitação, o ScrumMaster pode introduzir diferentes técnicas e práticas, propondo de tempos em tempos sua experimentação ao Time de Scrum, visando a tornar as retrospectivas mais efetivas.

Quando há vários Times de Scrum trabalhando próximos, pode também ser interessante ocasionalmente permutarem-se os ScrumMasters entre diferentes Times de Scrum nas reuniões de Sprint Retrospective, de forma a estimular a variação de práticas.

O ScrumMaster, enquanto facilitador, estimula todos os membros do Time de Desenvolvimento e o Product Owner a participarem igualmente das atividades. Aqueles pouco participativos, por exemplo, e aqueles que agem de forma oposta, deixando pouco espaço para os outros se manifestarem, representam um especial desafio à facilitação.

O ScrumMaster também cuida da agenda da reunião, assegurando-se que nela aconteçam todas as atividades necessárias para se chegar a resultados, e garantindo que o seu *timebox* seja cumprido.

Neutralidade

O ScrumMaster, no entanto, não participa diretamente na realização das atividades, ou interfere com suas opiniões nas ideias e no processo decisório do Time de Scrum. Para que as melhorias sejam corretamente identificadas e colocadas em prática, é importante que os membros do Time de Desenvolvimento e o Product Owner sejam responsáveis pelo processo de geração de ideias, sentindo-se, assim, igualmente responsáveis por seus resultados.

Impedimentos

O ScrumMaster também exerce na reunião o importante papel de não deixar passarem despercebidos pontos que exigirão sua atuação direta. Impedimentos dificultam consideravelmente ou obstruem o trabalho do Time de Desenvolvimento em direção à Meta do Sprint. Garantir sua remoção é responsabilidade do ScrumMaster.

Embora sejam identificados normalmente pelo Time de Desenvolvimento no seu trabalho diário e imediatamente comunicados ao ScrumMaster, não é incomum impedimentos emergirem na reunião de Sprint Retrospective na forma de pontos a serem melhorados. A responsabilidade pela resolução dees impedimentos detectados na reunião também recai sobre o ScrumMaster.

20.2.3 Participação do Product Owner

O Product Owner é membro do Time de Scrum e, como tal, colabora com as melhorias incrementais contínuas e participa das reuniões de Sprint Retrospective. No entanto, alguns comportamentos disfuncionais podem prejudicar a sua participação nessas reuniões. Por essa razão, não é incomum encontrarmos recomendações equivocadas de que o Product Owner não deve participar delas.

Enquanto facilitador, todavia, é papel do ScrumMaster trabalhar para garantir uma participação positiva e efetiva de todos os membros do Time de Scrum nas reuniões de Sprint Retrospective.

Na presença de um Product Owner não colaborativo, que se coloque em um nível hierarquicamente superior ou que seja (ou haja) como um cliente a

cobrar por resultados, os membros do Time de Desenvolvimento podem se sentir intimidados caso questões a serem levantadas toquem em pontos sensíveis ou estejam diretamente relacionadas com o trabalho do Product Owner. Assim, essas questões podem acabar não se revelando durante a reunião, tornando-a ineficiente.

A intimidação pode também acontecer na direção contrária. Os membros do Time de Desenvolvimento podem entender que devem aproveitar a presença do Product Owner e exercer um foco excessivo em problemas relacionadas ao trabalho dele. O Product Owner, nesse caso, passa a ser o centro das atenções durante a reunião ou, pior, um alvo de reclamações e acusações. Esse comportamento é comum quando o Product Owner é ausente durante o Sprint, o que, de fato, configura-se como um problema a ser resolvido.

A ocorrência de algum desses casos depende de que tipo de relação o Product Owner e o Time de Desenvolvimento cultivam. De forma geral, quanto mais próximos eles trabalharem, e quanto mais interagirem e colaborarem no seu dia a dia, mais a participação do Product Owner na reunião será vista como natural e positiva.

20.2.4 Dinâmica básica de uma retrospectiva

Descrevo em seguida um exemplo de como pode se parecer a dinâmica básica de uma reunião de Sprint Retrospective. É bom destacar que existem diversas formas de se conduzir essa reunião, mas o objetivo final é sempre o de se realizarem melhorias contínuas a partir de planos de ação realizáveis.



Fig. 20.1: Quadro básico de retrospectiva

- 1) Os membros do Time de Desenvolvimento, Product Owner e o ScrumMaster se reúnem em uma sala privada para realizar a reunião;
- 2) Time de Desenvolvimento e Product Owner repassam rapidamente os pontos levantados na reunião de retrospectiva anterior. O objetivo é verificar se e como as ações definidas para realizar as melhorias foram executadas, e que pontos importantes ainda restaram para serem tratados em Sprints futuros;
- 3) Time de Desenvolvimento e Product Owner utilizam alguns minutos para relembrar os fatos mais marcantes do Sprint que está se encerrando. Para viabilizar essa atividade, o ScrumMaster pode sugerir o uso de diferentes práticas;
- 4) mais alguns minutos são então reservados para que os membros do Time de Desenvolvimento e o Product Owner reflitam e escrevam (em geral, em notas adesivas) o que consideram que foi bem no Sprint que está se encerrando, e o que consideram que pode ser melhorado para futuros Sprints;

- 5) em seguida, os membros do Time de Desenvolvimento e o Product Owner colam suas notas adesivas em um quadro com as colunas “o que foi bem” e “o que pode melhorar” (um exemplo desse quadro pode ser visto na figura 20.1);
- 6) o ScrumMaster ou algum dos outros participantes agrupa itens parecidos ou relacionados. Os membros do Time de Desenvolvimento e o Product Owner ordenam esses grupos de itens de acordo com sua importância;
- 7) Time de Desenvolvimento e Product Owner, em seguida, discutem em torno dos grupos de itens levantados. Se houver itens demais para serem discutidos dentro do *timebox* da reunião e para serem tratados no próximo Sprint, os participantes devem concentrar-se apenas nos itens mais importantes, em geral não mais que dois ou três. Para definir quais são os itens mais importantes, pode-se usar como critério o número de participantes que apontaram o item ou alguma forma de votação;
- 8) os membros do Time de Desenvolvimento e o Product Owner repassam rapidamente cada item da coluna “o que foi bem”, comprometendo-se a manter ou repetir em Sprints futuros esses pontos identificados como positivos, caso faça sentido. Os presentes refletem sobre como tornar isso possível;
- 9) com relação aos pontos a melhorar, Time de Desenvolvimento e o Product Owner buscam identificar as causas raízes dos problemas e, a partir daí, formulam as ações necessárias (planos de ação) para colocar as melhorias correspondentes em prática, de forma a tornar seu trabalho mais eficiente já nos próximos Sprints. Planos de ação são mais efetivos quando indicam “o quê”, “quem” e “quando”. Ou seja, qual é a questão a ser tratada, quem ficará responsável por ela e em que prazo ela será tratada;
- 10) após repassar todos os pontos, traçar planos de ação adequados e registrá-los de alguma forma — em um papel à parte, por exemplo, ou até mesmo em notas adesivas coladas sobre os itens da coluna “o que pode melhorar” -, os presentes dão a reunião como encerrada.

20.2.5 Diferentes possibilidades na retrospectiva

O estabelecimento de uma rotina repetitiva pode ser uma receita de fracasso em médio prazo para retrospectivas de um Time de Scrum. A escolha de técnicas apropriadas para cada situação e a simples variação nas práticas utilizadas podem tornar a reunião mais dinâmica, o que mantém os membros do Time de Scrum envolvidos e interessados. O ScrumMaster, enquanto facilitador, geralmente escolhe e sugere como se dará a reunião.

O livro *Agile retrospectives: making good teams great*, de Esther Derby e Diana Larsen (2006), oferece, além de um *framework* para a execução das retrospectivas, diversas práticas que podem ser usadas. Mostro a seguir algumas das práticas mais populares entre times Ágeis:

- **coluna “a fazer”**
: adiciona-se mais uma coluna ao quadro da retrospectiva, onde serão colocadas notas adesivas com as ações de melhorias planejadas. As melhorias podem estar alinhadas aos pontos da coluna “O que pode melhorar” correspondentes;
- **coluna “delta”**
: a coluna “o que pode melhorar” é trocada pela coluna “delta”, que simboliza mudança. Assim, o foco se desloca do que pode melhorar para a melhoria em si, ou seja, as notas adesivas coladas nessa coluna já descrevem as mudanças a serem realizadas;
- **começar-parar-tentar**: em vez do quadro tradicional, divide-se o quadro nas colunas “Começar a fazer”, “Parar de fazer” e “A tentar”. Os membros do Time de Desenvolvimento colam na primeira coluna notas adesivas com ações que consideram que devem começar a fazer, na segunda coluna ações que realizaram no último Sprint, mas que consideram que devem parar de fazer e, na terceira, ações a se tentar no próximo Sprint;
- **classificação dos pontos**: existem diversas possíveis classificações para os pontos positivos e os pontos a melhorar que podem ser úteis para o Time de Desenvolvimento, entre as quais destaco:

- classificar os pontos a melhorar entre “Time” e “ScrumMaster” (ou “Organização”), o que significam, respectivamente, pontos sobre os quais o próprio Time de Desenvolvimento deve agir e questões organizacionais a respeito das quais somente o Scrum-Master pode fazer algo;
 - classificar os pontos a melhorar entre “Ação” e “Gradual”, ou seja, entre pontos para os quais é possível se traçar planos de ação mais precisos, ou objetivos e pontos para os quais as mudanças são mais subjetivas e graduais, como mudanças comportamentais;
 - classificar tanto os pontos positivos quanto os pontos a melhorar entre “Causado pelo Scrum”, quando a questão foi causada pelo uso do Scrum (repare que aqui geralmente são pontos positivos), e “Visível pelo Scrum”, nos casos em que foi o Scrum que tornou a questão visível;
- **linha do tempo:** o Time de Desenvolvimento traça uma linha do tempo com os acontecimentos mais importantes do Sprint e, a partir deles, identifica os pontos positivos e os pontos a melhorar. Observe que o próprio Gráfico de Sprint Burndown pode ser usado como essa linha do tempo, com a vantagem que seus pontos de inflexão para cima e linhas retas podem indicar impedimentos ou problemas que ocorreram durante o Sprint. Nesse caso, pode ser interessante que os membros do Time de Desenvolvimento coloem notas adesivas sobre pontos relevantes no próprio Gráfico de Sprint Burndown, indicando esses acontecimentos (veja a figura 20.2);

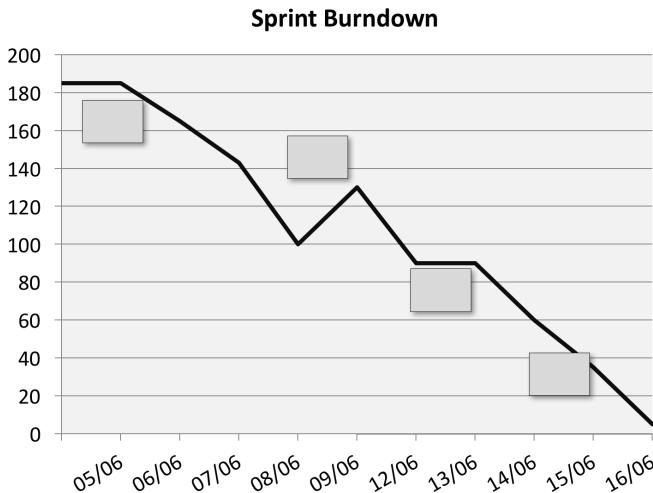


Fig. 20.2: Sprint Burndown utilizado como linha do tempo para a retrospectiva

- **foco único:** caso tenha havido algum problema importante suficiente que tenha causado grande impacto no Sprint e que o Time de Desenvolvimento acredite que poderá se repetir, a busca de soluções para esse problema pode ser o tema único (ou, ao menos, central) da reunião.

20.3 O QUE NÃO DEVE ACONTECER NA SPRINT RETROSPECTIVE?

A reunião de Sprint Retrospective não deve ser utilizada para se identificarem ações de melhoria no produto, trabalho que acontece na reunião de Sprint Review. O principal foco da reunião de Sprint Retrospective é a identificação do que precisa ser melhorado na forma de trabalho do Time de Scrum. Assim, é necessário que falhas ocorridas durante o Sprint que se encerra sejam identificadas e exploradas com franqueza.

Dessa forma, embora não se devam apontar erros individuais nessa reunião, o trabalho de inspeção envolve a capacidade dos participantes de expor,

em algum grau, suas próprias limitações e pontos fracos e, principalmente, as do Time de Scrum como um todo. Diante dessa necessidade, o Time de Scrum enfrenta importantes desafios que interferem em sua capacidade de executar uma reunião de Sprint Retrospective eficaz. Descrevo a seguir algumas disfunções comuns.

20.3.1 Busca de culpados

Norm Kerth (2001), em seu livro sobre retrospectivas de projetos, definiu o que chamou de Principal Diretiva das Retrospectivas.

PRINCIPAL DIRETIVA DAS RETROSPECTIVAS

Independente do que descobrirmos, entendemos e verdadeiramente acreditamos que todos fizeram o melhor trabalho que puderam, dados o que sabiam no momento, suas competências e habilidades, os recursos disponíveis e a situação que se apresentou.

Os objetivos de reuniões de Sprint Retrospective são sempre construtivos. O foco dos participantes da reunião deve se manter no desejo objetivo de melhoria, aprendendo-se com os erros cometidos.

Buscar culpados para os problemas que aconteceram durante o Sprint ou “lavar roupa suja” tem efeitos extremamente negativos sobre o andamento da reunião e sobre seus resultados. Em hipótese alguma deve-se fazer acusações, apontar nomes ou citar erros individuais, relacionando-os com problemas que ocorreram no Sprint.

O ScrumMaster, como facilitador, trabalha para que a busca de culpados não aconteça, de forma que os membros do Time de Desenvolvimento e Product Owner se sintam mais seguros para falar livremente sobre os problemas.

20.3.2 Insegurança e medo de exposição

Algum nível de exposição dos presentes é necessário para uma participação efetiva na reunião de Sprint Retrospective. No entanto, falar sobre o que não

funcionou bem em seu próprio trabalho e no trabalho de outros que estão presentes não é uma tarefa fácil para a maioria das pessoas.

Muitos não estão acostumados com o nível de exposição necessário e se sentem desconfortáveis com a possibilidade de haver qualquer tipo de enfrentamento. Essa dificuldade é ainda mais evidente em indivíduos habituados até então com o trabalho no estilo comando e controle.

Os membros do Time de Scrum poderão se sentir ainda mais intimidados caso haja algum cliente ou parte interessada importante do projeto presente na reunião, que esteja direta ou indiretamente relacionado a causas de problemas ocorridos durante o Sprint. Nesses casos, os membros do Time de Scrum não se sentem em um ambiente seguro para falar de problemas e, assim, não será surpreendente se até mesmo problemas bastante visíveis não forem mencionados nela. Por essa razão, somente estão presentes na reunião pessoas diretamente convidadas pelo Time de Scrum.

Os membros do Time de Desenvolvimento, no entanto, devem se sentir confortáveis para discutir problemas com o Product Owner. Caso não haja um ambiente de confiança que permita essa transparência, uma disfunção está configurada e cabe ao ScrumMaster, enquanto facilitador, trabalhar para ajudar os envolvidos a resolvê-la.

De forma geral, em uma retrospectiva saudável os membros do Time de Scrum conseguem levantar os pontos a melhorar sem inibição e sem medo de sofrer qualquer retaliação, pois sabem que todos os presentes compreendem que identificar possíveis melhorias é positivo para toda a organização. Porém, essas questões são naturalmente diferentes para diferentes de Time de Scrum, de acordo com as relações estabelecidas entre os envolvidos no projeto.

20.3.3 Conflitos da diversidade

A diversidade de personalidades presentes em um Time de Scrum e, portanto, em uma reunião de Sprint Retrospective, favorece a manifestação de uma variedade de conflitos durante a reunião. As pessoas naturalmente têm diferentes opiniões, pontos de vista e interesses e, quanto mais heterogêneo for o grupo, mais essas diferenças se acentuam.

Um ambiente hostil ou simplesmente não amigável pode levar os membros do Time de Desenvolvimento ou Product Owner a adotarem posturas

defensivas e a se fecharem, deixando consciente ou inconscientemente de trazer à tona pontos que ameacem a sua imagem ou a de colegas mais próximos perante o resto do Time de Scrum.

O ScrumMaster atua dentro e fora das retrospectivas, buscando dinâmicas que aumentem o entrosamento dos membros do Time de Scrum e que os estimulem a construir objetivos comuns, levando-os a funcionar como um verdadeiro time.

20.3.4 Pensamento grupal

Uma armadilha relativamente comum em grupos muito coesos é a tendência de seus membros criarem pressões intensas sobre si mesmos para obter e manter o consenso ou a conformidade. Essas pressões internas anulam a motivação do grupo em buscar, por meio do pensamento crítico, caminhos alternativos de ação. Assim, o grupo acaba por tomar decisões disfuncionais.

Esse processo nocivo é chamado de pensamento grupal ou *groupthink* (JANIS, 1982). Um Time de Scrum ou Time de Desenvolvimento que incorre em pensamento grupal pode apresentar alguns dos seguintes sintomas:

- acredita ser invulnerável a falhas, apresentando um otimismo excessivo e assumindo riscos desnecessários;
- trata membros que questionam ou discordam da opinião da maioria como desleais e exerce pressões para que se adequem às opiniões do resto do time;
- acredita possuir uma moralidade inerente, independente das ações de seus membros, o que os leva a ignorar consequências morais e éticas de suas decisões;
- cria estereótipos de pessoas externas que podem representar uma ameaça a seu trabalho, caracterizando-as como más demais para que seja possível negociar com elas ou tão fracas e tolas que não representam uma ameaça a ser considerada;

- acredita existir uma unanimidade em torno da visão da maioria de seus membros, de forma que o silêncio de uma minoria diante de discussões ou argumentações é tratado como concordância;
- induz seus membros a se autocensurarem por temerem a desaprovação dos outros membros ou por minimizarem a relevância de suas questões diante do aparente consenso do time;
- induz o surgimento de membros que protegem o time de informações externas que possam ameaçar os valores já estabelecidos;
- cria explicações coerentes e aceitáveis de forma a ser capaz de ignorar qualquer *feedback* negativo que, caso fosse levado em consideração, levaria seus membros a reconsiderarem suas concepções ou suposições.

Times auto-organizados possuem uma tendência natural a serem muito coesos e, assim, são particularmente vulneráveis ao pensamento grupal (MANZ; NECK, 1997). A reunião de Sprint Retrospective é, provavelmente, o momento em que esses fatores podem se tornar mais visíveis. Por essa razão, é quando o ScrumMaster pode atuar com maior intensidade para estimular o time a detectar, criar visibilidade e resolver o problema.

CAPÍTULO 21

Release

- **Objetivo:** entregar Incremento(s) do Produto gerado(s) para uso e *feedback*
- **Quando:** frequentemente, quando já se produziu valor suficiente para ser utilizado e gerar *feedback*
- **Participantes obrigatórios:** Time de Desenvolvimento e Product Owner
- **Saídas esperadas:** produto utilizável, em funcionamento

21.1 O QUE É A RELEASE?

Release é a entrega de um ou mais Incrementos do Produto prontos, gerados pelo Time de Desenvolvimento em um ou mais Sprints sucessivos, para que

sejam utilizados e que se obtenha *feedback*. Em conjunto e somados ao que já foi entregue anteriormente, esses Incrementos do Produto formam um produto que possui valor suficiente para ser usado. É comum também chamar-se de Release o objeto da entrega, ou seja, o Incremento ou Incrementos do Produto a serem entregues.

Projetos com Scrum realizam Releases frequentes, com intervalos máximos de alguns poucos Sprints entre elas.

A realização de Releases ao longo do projeto tem três objetivos principais:

- **obter *feedback* frequentemente** — uma vez realizada a Release, o Product Owner busca impressões e opiniões sobre o que foi recebido e utilizado e o que se espera da próxima Release junto a clientes e usuários do produto. A partir desse *feedback*, o Product Owner realizará mudanças no Product Backlog e, assim, o produto é construído incrementalmente;
- **dar um senso mais concreto de progresso a clientes e a demais partes interessadas** — ao receber uma Release, os clientes do projeto e demais partes interessadas têm visibilidade do estado atual do projeto, isto é, o que já está pronto e o que vislumbram que ainda há de necessidades a serem atendidas para se realizar a Visão do Produto. O senso de progresso decorrente de uma Release tende a ser mais concreto do que o obtido a partir de reuniões de Sprint Review, já que se trata de uma entrega para uso e não apenas uma demonstração;
- **prover retorno ao investimento dos clientes** — ao se realizar Releases frequentemente para usuários finais do produto, em cada entrega é gerado um retorno ao investimento realizado pelos clientes do projeto. No entanto, dependendo de características do produto e de sua estratégia de negócios, pode fazer sentido realizar entregas com menor frequência, ou até mesmo apenas ao final do projeto, para quem de fato vai usar o produto. Pode-se, no entanto, realizar Releases para usuários intermediários, o que em geral não constitui retorno sobre o investimento.

A estratégia de Releases do projeto é de inteira responsabilidade do Product Owner, que deve defini-la e, sempre que necessário, modificá-la. Essa

estratégia estabelece, por exemplo, quais dos objetivos entre os descritos anteriormente cada Release busca alcançar, o que inclui a frequência das Releases e quem vai recebê-las. A estratégia de Releases também garante que elas sejam tecnicamente viáveis e estejam alinhadas com a estratégia de negócios do produto e da organização.

21.2 COMO É A RELEASE?

Consideram-se duas dimensões essenciais em uma estratégia de Releases: quando ou com que frequência serão realizadas, e quem vai recebê-las. Assim, classificamos as Releases de um projeto quanto à frequência e quanto a quem as recebe.

21.2.1 Quanto à frequência

Com relação a quando ou com que frequência deverão ser realizadas, as Releases podem ser classificadas em: Release por valor, Release por Sprint, Release por item e Release por plano.

Release por valor

Dependendo da natureza do negócio, a Release pode ser realizada quando o Product Owner julgar que os Incrementos do Produto já gerados nos Sprints pelo Time de Desenvolvimento representam valor de negócio suficiente para que valha a pena entregá-los para que usuários os utilizem. Dessa forma, o Time de Desenvolvimento segue gerando Incrementos do Produto prontos, Sprint a Sprint, até que o Product Owner julgue que já se tem o suficiente para se realizar uma Release.

Release por Sprint

Pode-se realizar uma Release ao final de cada Sprint, aproveitando a própria cadência do Scrum. Nesse cenário, a reunião de Sprint Planning já é suficiente para que os clientes e demais partes interessadas entendam, por meio do Product Owner, o que será recebido e quando.

Assim, o Time de Desenvolvimento trabalha para que o resultado final

do Sprint seja imediatamente entregue, dependendo apenas da concordância do Product Owner ao final do Sprint. Realizar a Release em cada Sprint é extremamente Ágil, pois antecipa o *feedback* dos usuários sobre o que foi produzido e maximiza as oportunidades de melhorias no produto.

Release por item

Em um cenário ainda mais Ágil, os itens desenvolvidos pelo Time de Desenvolvimento são entregues durante o próprio Sprint, no que se pode caracterizar como entrega contínua. Entregar o item, portanto, é parte da Definição de Pronto usada, o que pode ou não envolver uma aprovação imediata do Product Owner. Esse tipo de abordagem é muitas vezes inviável dependendo do negócio e do ambiente, mas seu uso é cada vez mais frequente.

Release por plano

Para cada Release, pode-se criar um plano de alto nível do que será desenvolvido. Nesse caso, é comum também realizar uma reunião de Release Planning para cada Release, logo antes de seu início.

Nessa reunião, é estabelecido o Plano da Release, que contém uma data aproximada para a Release, uma Meta a ser realizada e um conjunto de itens selecionados a partir do alto do Product Backlog. O progresso em direção à data da Release e, portanto, ao cumprimento da Meta da Release, é inspecionado em cada Sprint. As ferramentas mais utilizadas com esse propósito são o Gráfico de Release Burndown e o Gráfico de Release Burnup (veja [27 Burndown e Burnup: acompanhando o trabalho](#)).

Nesses casos, é comum estabelecer-se o tamanho da Release a partir do número de Sprints que acontecerão até a entrega. É igualmente comum nesses casos chamar-se também de Release todo o período de trabalho realizado desde o início do primeiro Sprint planejado para a Release até a entrega propriamente dita. Dizemos, por exemplo, que *essa Release durará cinco Sprints*, ou que *estamos trabalhando na quinta Release*.

21.2.2 Quanto a quem a recebe

Uma Release deve idealmente ser realizada para os usuários finais, de forma a se obter *feedback* sobre os Incrementos do Produto gerados e proporcionar retorno ao investimento para os clientes do projeto. Quando esse cenário não é possível, a Release pode ser feita para um grupo de usuários intermediários ou de usuários finais selecionados. Assim, garante-se no mínimo um *feedback* para se reduzirem os riscos do projeto.

Em um mesmo projeto pode aparecer, em momentos distintos, qualquer um dos três cenários definidos em seguida, mas ao menos uma Release para os usuários finais obrigatoriamente ocorrerá no final do projeto.

Release para os usuários finais

O Incremento ou Incrementos do Produto são disponibilizados para os usuários finais do produto. Tendo as funcionalidades que mais necessitam em suas mãos, estes as utilizarão, gerando retorno ao investimento feito pelos clientes do projeto e possibilitando a obtenção de *feedback*.

Sempre que é possível de ser adotado, esse cenário representa o menor risco, pois Releases frequentes para usuários reais os permitem oferecer *feedback* a partir do uso real do produto.

Release para usuários intermediários

Quando não é possível realizar a Release para os usuários finais do produto, um grupo de pessoas pode ser escolhido para representá-los. Esses usuários intermediários são geralmente selecionados na própria organização que gera o produto ou nos clientes.

Eles podem ser especialistas no negócio do produto, especialistas acerca de seus usuários finais, especialistas em usabilidade ou quaisquer pessoas capacitadas a utilizar o produto e a verificar o que é necessário para atrair os usuários e satisfazer suas necessidades. Eles receberão essa Release interna e serão responsáveis por prover *feedback* sobre o que lhes foi entregue.

Esse cenário é comum nos produtos em que os usuários reais são anônimos e não há sentido ou não é interessante disponibilizar um produto parcial. Produtos de prateleira constituem exemplos comuns.

Release para usuários finais selecionados

Em produtos com um grande número de usuários, um grupo menor e representativo desses usuários reais pode ser escolhido para receber a Release. Esse grupo utilizará cada conjunto de Incrementos do Produto entregue com a consciência de que se trata de um produto parcial e será responsável por prover *feedback* sobre ele.

Em muitos casos, o benefício e estímulo para que continuem usando o produto e provendo *feedback* é a possibilidade de experimentar o produto antes de todos, ou a possibilidade de utilizá-lo de graça, por exemplo. Outros chegam até mesmo a pagar para fazer parte de programas desse tipo.

Os usuários selecionados com esse propósito são geralmente chamados de *beta-testers*, *beta-users* ou grupos de foco.

21.2.3 Outras dimensões

Existem ainda outras possíveis dimensões de Releases de que não tratarrei neste livro.

Uma dimensão de modelo comercial, relativa a Releases propositalmente limitados com relação ao produto vendido, por exemplo, contemplaria versões de demonstração, com funcionalidade reduzida para se gerar expectativa nos usuários sobre a versão final. Contemplaria também versões de *shareware*, que podem ser livremente distribuídas, mas usadas apenas por um período até que o produto pare de funcionar, forçando a decisão de compra.

21.2.4 Conclusão

Idealmente, as Releases são realizadas por item (ou seja, em entrega contínua) e para o usuário final. Quando isso não é possível, o Product Owner deve definir uma estratégia de Releases conjugando para quem e com que frequência as Releases serão realizadas, de forma a se aproximarem, o máximo possível, do usuário final.

Assim, Releases podem ser realizadas para usuários diferentes em frequências diferentes. Uma grande empresa que vende jogos de computador de prateleira, por exemplo, utiliza uma estratégia de Releases que ilustra bem esse ponto:

- **release por Sprint para usuários intermediários** — a partir de um dado momento, mas ainda cedo no desenvolvimento do jogo, uma equipe interna a empresa recebe as entregas internas ao final de cada Sprint e seu trabalho é jogar o jogo para prover *feedback* sobre a sua usabilidade;
- **release por valor para usuários finais selecionados** — frequentemente, mas com menor incidência, por decisão direta do Product Owner, o jogo é entregue a usuários selecionados, os *beta-testers*, que proverão *feedback* sobre o jogo como um todo e permitirão a coleta de métricas de uso;
- **release por valor para os usuários finais** — por fim, uma entrega é realizada para os usuários finais ao se disponibilizar o produto nas lojas.

CAPÍTULO 22

Release Planning

- **Objetivo:** planejamento da próxima Release
- **Quando:** antes do início do trabalho para a Release (em geral, ao final do último Sprint da Release anterior ou antes do primeiro Sprint do projeto)
- **Duração:** não há duração estabelecida, mas é importante se definir um *timebox*
- **Participantes obrigatórios:** Product Owner, Time de Desenvolvimento e ScrumMaster
- **Saídas esperadas:** Plano da Release

22.1 O QUE É A RELEASE PLANNING?

Em projetos que utilizam Releases por plano como parte de sua estratégia de Releases, pode-se realizar as reuniões de Release Planning (veja [21.2 Como é a Release?](#)). Product Owner e Time de Desenvolvimento, facilitados pelo ScrumMaster, encontram-se para criar o Plano da Release, que indica qual o objetivo a ser realizado a partir do que é entregue na Release e quando será realizado.

22.2 PLANO DA RELEASE

O Plano da Release contém:

- a Meta da Release, que é um objetivo de negócios a ser realizado por meio da entrega, e representa um incremento à realização da Visão do Produto (veja [24.3 Meta de Release/Roadmap](#));
- a data exata ou aproximada em que a entrega será realizada;
- um conjunto de itens selecionados do Product Backlog para a Release. Visando a realizar a Meta da Release, esses itens serão desenvolvidos do mais importante (granularidade mais fina) ao menos importante (granularidade mais grossa), até se chegar à data da Release.

A abordagem de uso de um Plano de Release e de reuniões de Release Planning, conforme descritas neste capítulo, somente é viável se o Time de Desenvolvimento atribuir estimativas para os itens do Product Backlog, o que pode ser feito nessa mesma reunião.

22.3 COMO É A RELEASE PLANNING?

22.3.1 Agendamento e duração

Como o Time de Desenvolvimento trabalha continuamente dentro de Sprints, um atrás do outro, não existe um intervalo entre dois Sprints que possa ser utilizado para a realização da reunião de Release Planning. Assim,

a reunião de planejamento para a próxima Release é geralmente realizada durante o último Sprint da Release atual, preferencialmente perto do seu final.

O uso desse tempo reduz um pouco o quanto o Time de Desenvolvimento será capaz de produzir nesse último Sprint. Logo, pode-se levá-lo em consideração em seu planejamento. Alternativamente, alguns times preferem realizar a reunião de Release Planning imediatamente antes da reunião de Sprint Planning do primeiro Sprint da Release a ser planejada.

Caso se trate da primeira Release do projeto, em geral a reunião de Release Planning é realizada antes do início dos Sprints, ou seja, antes do início do desenvolvimento do produto. No entanto, como nesse momento se conhece pouco sobre a capacidade de produção do Time de Desenvolvimento, é também comum esperarem-se dois ou três Sprints para se adquirir mais parâmetros, e só então realizar o planejamento da primeira Release do projeto.

Não há duração oficial estabelecida para a reunião de Release Planning. Recomenda-se, no entanto, que se estabeleça um tempo máximo de duração para ela (*timebox*) e que esse tempo não ultrapasse um dia de trabalho do Time de Desenvolvimento.

22.3.2 A reunião

Podemos identificar dois diferentes cenários para uma reunião de Release Planning. No primeiro cenário, é dada uma data para a Release, mas não se sabe qual a Meta a ser realizada. É o caso em que é necessário fazer uma entrega até uma determinada data. No segundo cenário, é dada uma necessidade de negócios (que será a própria Meta da Release) e pergunta-se quando será possível realizá-la.

Em ambos cenários, itens suficientes do Product Backlog devem estar estimados para que seja possível realizar o planejamento.

Dada a data da Release

A partir da data da Release, dados um Product Backlog estimado, o tamanho constante do Sprint e a Velocidade do Time de Desenvolvimento (veja [26.7 Velocidade do Time de Desenvolvimento](#)), é possível se determinar o resto

do plano: um conjunto de itens selecionados do Product Backlog para a Release e a Meta da Release. Vou explicar com um exemplo.

Digamos que o Product Owner estabeleça, a partir de questões de negócios, a necessidade de uma entrega a ser realizada daqui a dois meses. Digamos também que o tamanho do Sprint utilizado pelo Time de Desenvolvimento seja de duas semanas. Assim, podemos calcular quantos Sprints serão executados até a data dessa Release:

$$\begin{aligned} n. \text{ Sprints} &= (\text{Tempo Total / Tam. do Sprint}) = (8 \text{ semanas} / 2 \text{ semanas}) \\ &= 4 \text{ Sprints} \end{aligned}$$

Sabemos, portanto, que quatro Sprints serão executados até a data dessa Release. Também podemos estimar quantos pontos o Time de Desenvolvimento entregará nessa Release:

$$\begin{aligned} n. \text{ pontos} &= (\text{Velocidade x n. Sprints}) = (30 \text{ pontos/Sprint} \times 4 \text{ Sprints}) \\ &= 120 \text{ pontos} \end{aligned}$$

Para determinar o escopo provável da Release, parte-se do topo do Product Backlog e se desce, somando-se as estimativas de cada item dadas pelo Time de Desenvolvimento até se chegar em algo próximo (um pouco mais ou um pouco menos) que esses 120 pontos.

É importante entender que essa extração traz uma previsão cada vez mais baixa sobre a precisão quanto mais distante for a data da Release, gerando um planejamento que deve ser revisto Sprint a Sprint.

Dada a Meta da Release

Em outro caso, digamos que, em vez de fornecer uma data, o Product Owner deseja saber em quanto tempo uma determinada necessidade de negócios poderá ser atendida. Essa necessidade de negócios é a própria Meta da Release.

Primeiramente, o Product Owner deve garantir que o Product Backlog esteja ordenado e com os itens adequados para atender a essa necessidade. O

Product Owner partirá então do topo do Product Backlog e descerá, somando as estimativas de cada item dadas pelo Time de Desenvolvimento, até considerar que já percorreu os itens necessários para atender a essa necessidade de negócios.

Digamos que, por exemplo, a soma dessas estimativas seja 184 pontos. Se o Time de Desenvolvimento produz, em média, 30 pontos por Sprint (Velocidade), poderemos então estimar em quantos Sprints ele será capaz de queimar esses pontos:

$$\begin{aligned} n. \text{ Sprints} &= (\text{n. pontos} / \text{Velocidade}) = (184 \text{ pontos} / 30 \text{ pontos/Sprint}) \\ &= \text{aproximadamente 6 Sprints} \end{aligned}$$

Dessa forma, podemos prever que o Time de Desenvolvimento será capaz de queimar esses pontos em seis Sprints (parte inteira), ou seja, daqui a três meses. É sempre importante lembrar de que essa estimativa de data e de escopo é de baixa precisão, e deve ser revista em cada Sprint.

22.3.3 Granularidade dos itens

Como resultado da reunião de Release Planning, um conjunto de itens terá sido selecionado a partir do alto do Product Backlog. Na realidade, esses itens não são separados do Product Backlog, de forma que não existe um Release Backlog. Faz-se apenas algum tipo de marcação nos itens que pertencem à Release planejada, como uma etiqueta.

Como são parte do Product Backlog, esses itens prováveis para a Release têm diferentes níveis de granularidade. Os itens que estão no alto do Product Backlog e que, de acordo com o plano, serão desenvolvidos nos primeiros Sprints da Release, devem ser menores e representar mais detalhes. Possuem, portanto, granularidade mais fina e estimativas de maior precisão.

Os itens que supostamente serão desenvolvidos nos Sprints seguintes da Release, portanto mais abaixo no Product Backlog, serão maiores e mais vagos, com granularidade mais grossa e estimativas de menor precisão.

À medida que o trabalho de desenvolvimento avança Sprint a Sprint em direção à Release, os itens do alto vão sendo retirados do Product Backlog

para desenvolvimento. Assim, os itens seguintes que chegam ao topo do Product Backlog serão gradativamente detalhados, refinando-se a sua granularidade e suas estimativas.

22.3.4 Escopo da Release

Embora persiga-se realizar uma meta de negócios clara, o escopo da Release é aberto. Novos itens surgirão durante a Release e outros desaparecerão. Itens existentes vão evoluir, ganharão mais detalhes, serão divididos em itens menores, reestimados e reordenados. Os itens menos importantes, resultados dessa evolução, serão relegados a partes mais baixas do Product Backlog.

Assim, ao se atingir a data prevista para a Release, os itens que restarão na lista de itens prevista inicialmente para a Release serão os de menor importância. Logo, há menores chances de a Meta da Release não ter sido realizada.

É importante destacar que a reunião de Release Planning de forma alguma substitui as reuniões de Sprint Planning. O escopo de um Sprint futuro durante o trabalho para a Release está em aberto até que se realize a sua reunião de Sprint Planning. Ou seja, o conjunto de itens que entrará em cada Sprint Backlog será definido na sua reunião de Sprint Planning respectiva, e não na reunião de Release Planning.

CAPÍTULO 23

Refinamento do Product Backlog

- **Objetivo:** refinamento do Product Backlog e sua preparação para o desenvolvimento
- **Quando:** pelo Product Owner, sempre que necessário. Durante o Sprint, é um trabalho contínuo, eventual ou realizado em sessões agendadas entre Product Owner e Time de Desenvolvimento
- **Duração:** não há duração estabelecida, mas em geral o Time de Desenvolvimento não utiliza no total mais do que 5-10% do seu esforço em cada Sprint;
- **Participantes obrigatórios:** Product Owner e Time de Desenvolvimento
- **Saídas esperadas:** o Product Backlog ordenado, planejável, emergente

e gradualmente detalhado. Espera-se obter uma quantidade suficiente de itens preparados para o próximo Sprint

23.1 O QUE É O REFINAMENTO DO PRODUCT BACKLOG?

O Product Backlog é progressivamente atualizado e detalhado ao longo de todo o projeto. Esse trabalho é chamado de Refinamento do Product Backlog e visa a garantir que ele seja:

- **ordenado**, para maximizar o retorno sobre o investimento dos clientes do projeto;
- **planejável**, de forma que o Time de Desenvolvimento e o Product Owner sejam capazes de planejar o desenvolvimento do produto;
- **emergente**, refletindo o dinamismo do ambiente de mudanças no qual o projeto está imerso;
- **gradualmente detalhado**, de forma que seus itens possuam um detalhamento adequado e que, assim, um número suficiente de itens esteja preparado para o Sprint seguinte.

O trabalho de Refinamento do Product Backlog inclui a adição de novos itens, a remoção de itens que não farão mais parte do produto, o desmembramento de itens maiores em itens menores, a junção de itens menores em um item maior que perdeu prioridade, o detalhamento de itens, seu reordenamento e, possivelmente, a criação de algum tipo de estimativa.

Discussão: Sprint Planning ineficiente

O Product Owner inicia o Sprint Planning apresentando os itens mais importantes do Product Backlog, um a um. O Time de Desenvolvimento, que ainda não conhecia esses itens, faz perguntas de todo o tipo, que são prontamente respondidas pelo Product Owner.

Os Critérios de Aceitação de cada item ainda não foram discutidos e, para alguns dos itens, sequer foram criados. O Product Owner não teve tempo de

prepará-los previamente. Como é a primeira vez que tem acesso a esses itens, o Time de Desenvolvimento também ainda não os estimou.

Time de Desenvolvimento e Product Owner então consideram item a item para discutir e preparar os Critérios de Aceitação. É um trabalho detalhado, que gera muitas ideias e discussões e, assim, mais um tempo considerável é necessário para esclarecimentos.

Em seguida à definição dos Critérios de Aceitação de um item, o Time de Desenvolvimento realiza a atividade de Planning Poker para estimá-lo. Nela surgem novas dúvidas que, novamente, são prontamente esclarecidas pelo Product Owner.

O Time de Desenvolvimento logo identifica que um dos itens mais importantes é muito grande e que provavelmente deve ser fatiado em itens menores. Product Owner e Time de Desenvolvimento então colaboram para dividi-lo, e decidem que uma de suas partes não tem importância para esse Sprint e deve voltar para o Product Backlog. Não demoram a identificar outro item a ser fatiado e novamente realizam esse trabalho.

Para um dos itens mais importantes, o Time de Desenvolvimento entende que o Product Owner não possui detalhes suficientes para que seja colocado em desenvolvimento. O Product Owner procura negociar e até mesmo convencê-los a aceitar para o Sprint esse item importante. Porém, o Time de Desenvolvimento recusa, pois considera muito arriscado que um item com pouquíssimos detalhes faça parte de seu Sprint Backlog. Essa negociação eleva o nível de estresse da reunião e o ScrumMaster, enquanto facilitador, faz o seu melhor para que o conflito seja resolvido.

Após algumas horas de reunião, mais próximo do final do dia, o Time de Desenvolvimento e o Product Owner estão exaustos. Tantos detalhes foram discutidos e tantas negociações ocorreram que todos sentem que o resto da reunião não será mais produtivo. Os itens que restam são discutidos mais rapidamente e, pelo cansaço, o Time de Desenvolvimento está mais propenso a aceitá-los com uma compreensão menor dos detalhes. Uma Meta para o Sprint é rapidamente negociada e, em seguida, o Time de Desenvolvimento parte para a segunda parte da reunião, onde quebrará os itens em tarefas.

A exaustão toma conta do Time de Desenvolvimento, e o que seus membros mais querem é chegar ao fim do dia. As tarefas são geradas sem o cuidado

e reflexão adequados, e o Sprint Backlog resultante não representa um bom plano para o Sprint.

Como consequência, o Sprint é bastante conturbado. Surpresas no trabalho surgem a todo momento, muitos esclarecimentos são necessários e uma quantidade grande de novas tarefas são diariamente adicionadas ao Sprint Backlog. Ao final, o Time de Desenvolvimento não realiza a Meta do Sprint satisfatoriamente.

O que levou o Time de Desenvolvimento ao fracasso nesse Sprint? Será que uma preparação não o teria ajudado?

23.2 COMO É O REFINAMENTO DO PRODUCT BACKLOG?

23.2.1 Participantes

Pela natureza emergente do Product Backlog, o Refinamento do Product Backlog é um trabalho contínuo, realizado pelo Product Owner ao longo de todo projeto. O Product Owner tem a prerrogativa de alterar o Product Backlog sempre que se fizer necessário, adicionando, removendo, dividindo e detalhando seus itens.

No entanto, o Product Owner não realiza todo o Refinamento do Product Backlog sozinho. Embora o Product Owner deva prepará-los da melhor forma possível, o Time de Desenvolvimento é quem utilizará os itens do Product Backlog para realizar seu trabalho no Sprint seguinte, transformando-os em um Incremento do Produto pronto.

Dessa forma, enquanto usuário ou “consumidor” desses itens, é importante que o Time de Desenvolvimento interaja com o Product Owner durante o Sprint para que, juntos, preparem um número de itens suficiente para serem trabalhados no Sprint seguinte. Assim, é comum Product Owner e membros do Time de Desenvolvimento realizarem em conjunto uma ou mais sessões de Refinamento do Product Backlog ao longo do Sprint, visando a preparar itens para o Sprint seguinte. Essas sessões são também conhecidas como *Backlog Grooming*.

É importante destacar que não é uma boa prática o Product Owner chegar a uma sessão de Refinamento do Product Backlog realizada em conjunto com

o Time de Desenvolvimento sem trazer itens com algum nível de preparação. Em geral, ele o faz usando o máximo de informações de que puder dispor. De outra forma, a sessão pode tornar-se cansativa e ineficiente.

Ao final das sessões de Refinamento, os itens mais importantes do Product Backlog possuem os detalhes necessários para serem colocados em desenvolvimento. Prepara-se durante o Sprint um número de itens que se julgue suficiente para estar no Sprint seguinte, mas estes somente serão de fato escolhidos na reunião de Sprint Planning.

Frequentemente, a criação de estimativas é parte das sessões de Refinamento do Product Backlog. Esse trabalho de estimar é realizado unicamente pelo Time de Desenvolvimento. É importante, no entanto, que o Product Owner esteja presente para esclarecer dúvidas que invariavelmente vão surgir.

Alguns Times de Desenvolvimento preferem realizar as sessões de Refinamento do Product Backlog com apenas alguns de seus membros, que em seguida transmitem os resultados para o restante do time que seguiu com o trabalho do Sprint durante a sessão. Entretanto, recomendo a participação de todos, pois normalmente leva a um melhor compartilhamento das informações e a uma sensação maior de propriedade sobre os resultados do trabalho.

23.2.2 Quando ocorre

Cada Time de Scrum possui seu próprio processo para a colaboração entre Product Owner e Time de Desenvolvimento nas atividades de Refinamento do Product Backlog. Para alguns, esse é um trabalho contínuo, que ocorre durante todo o Sprint. Para outros, é um trabalho eventual, muitas vezes solicitado pelo Product Owner, que traz novos itens a serem preparados.

Outros Times de Scrum preferem agendar sessões de trabalho em dias e horários específicos, também durante o Sprint. Essas podem ser curtas sessões diárias, o que somente é possível com uma alta disponibilidade do Product Owner, sessões semanais ou até mesmo uma ou mais sessões próximas ao fim do Sprint.

Quando itens são preparados cedo demais, um ambiente com mudanças muito frequentes pode diminuir suas chances de pertencerem ao trabalho a ser realizado no Sprint seguinte, levando à geração de desperdício. Nesses casos, uma preparação mais próxima ao final do Sprint pode ser mais produtiva.

Mas, se houver apenas uma sessão de Refinamento ao final do Sprint, pode não haver tempo suficiente para reflexões sobre decisões de seus detalhes que podem ser necessárias.

Adicionalmente, uma sessão de Refinamento do Product Backlog pode ser realizada juntamente com, ou imediatamente antes, de uma reunião de Release Planning, exatamente para viabilizar o trabalho a ser realizado nessa reunião, preparando-se itens com estimativas, por exemplo.

Seja qual for o formato escolhido, essa colaboração não deve ocupar muito tempo do trabalho do Time de Desenvolvimento. Usualmente, recomenda-se que não se use mais que 5 a 10% do esforço do Time de Desenvolvimento no Sprint com essas atividades.

Parte V

Técnicas complementares

Nesta parte do livro, ofereço algumas técnicas e práticas comumente utilizadas por Times de Scrum, detalhadas e enriquecidas com exemplos, que podem ajudar times a obterem sucesso com o uso do *framework*. São elas:

- metas, para se definirem os objetivos a serem realizados no projeto;
- User Stories, provindas da metodologia Extreme Programming, que permitem representar-se o trabalho a ser realizado pela perspectiva do usuário;
- Story Points, que podem ajudar a estimar o trabalho com melhor precisão;
- Gráficos de Burndown e Burnup, que trazem visibilidade quanto ao cumprimento do trabalho planejado.

CAPÍTULO 24

Metas: definindo os objetivos

24.1 O QUE SÃO METAS NO SCRUM?

Seja no contexto de um Sprint, de uma Release ou do projeto como um todo, o Time de Desenvolvimento com Scrum trabalha desde o item de maior importância em direção ao item de menor importância em cada momento, em busca de um objetivo ou meta clara e realizável.

Idealmente, as metas no Scrum não são quantitativas. Diferentes índices de produtividade ou uma quantidade de trabalho a ser realizado não são boas escolhas, portanto. Preferimos, na verdade, que essas metas reflitam necessidades de negócios ou do usuário, que podem ser realizadas com maior ou menor grau de sucesso como resultado do trabalho cumprido, trazendo uma flexibilidade mais realista para o desenvolvimento de *software*.

Assim, ao trabalhar sobre os itens seguindo sua ordem de prioridade, o Time de Desenvolvimento, ao chegar ao prazo estabelecido, pode ter reali-

zado a meta definida havendo desenvolvido mais ou menos itens, mas res- tando sempre os de menor importância.

Essas metas ou objetivos funcionam no Scrum ao mesmo tempo como guias e motivadores. Elas são definidas pelo Product Owner e ajustadas com o Time de Desenvolvimento. Juntos, eles as tornam factíveis, de forma que o Time de Desenvolvimento possa se comprometer a realizá-las e trabalhar em direção a elas.

As metas ou objetivos geralmente usados no Scrum são:

- **Meta do Sprint (contexto de um Sprint)** — é parte integrante do Scrum e é obrigatoriamente estabelecida na reunião de Sprint Planning para cada Sprint do projeto. Cada Meta de Sprint, conforme explicarei mais adiante, pode significar um incremento à realização da Meta da Release ou de Roadmap, e cada funcionalidade desenvolvida pelo Time de Desenvolvimento significa um incremento à realização da Meta do Sprint;
- **Meta da Release ou de Roadmap (contexto de uma Release ou de um marco do Roadmap)** — a Meta da Release, caso usada, existe para cada Release do projeto e é estabelecida antes do início da Release (possivelmente durante a reunião de Release Planning). Caso o Time de Scrum realize Releases com alta frequência — a cada Sprint ou em entregas contínuas, por exemplo —, as Metas de Release podem ser substituídas por metas de mais alto nível, que aqui chamo de Metas de Roadmap. Elas são expressas em um Roadmap do Produto. Cada Meta de Release ou de Roadmap significa um incremento à realização da Visão do Produto;
- **Visão do Produto (contexto do produto)** — é o objetivo maior e mais geral a ser realizado por meio do desenvolvimento do produto. Ela é estabelecida antes de se iniciarem os Sprints e existe ao longo de todo o desenvolvimento.

A Visão do Produto não é parte do *framework* Scrum. No entanto, uma vi- são bem definida é importante para prover contexto para o trabalho do Time de Scrum e alinhamento entre os membros do Time de Scrum, os clientes do

projeto e as demais partes interessadas. Da mesma forma, as Metas de Release ou de Roadmap não são obrigatórias nem são parte do Scrum, mas facilitam a ligação entre as Metas dos Sprints e a Visão do Produto.

O Roadmap do Produto é um plano em alto nível que pode ser usado para se ter uma visão de como se dará a evolução do produto no tempo. Esse plano é expresso em uma linha do tempo com datas e objetivos a serem realizados. Cada um desses objetivos pode ser expresso como uma Meta de Release ou de Roadmap.

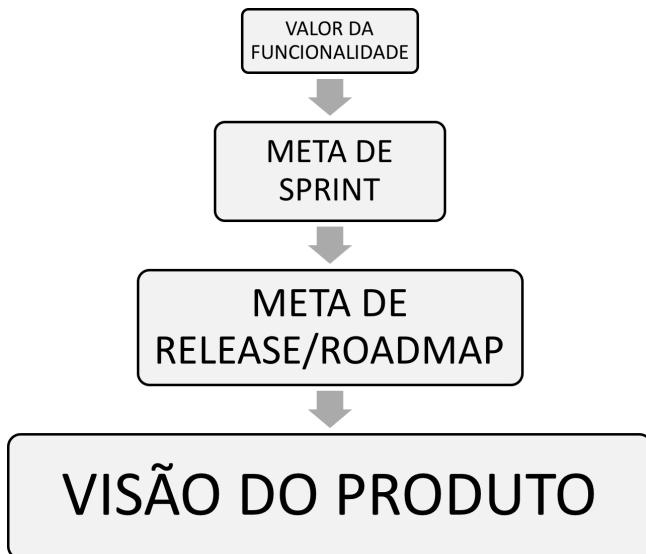


Fig. 24.1: Do valor da funcionalidade à Visão do Produto

O conjunto formado por cada Meta de Release ou de Roadmap, estabelecida para cada Release ou marco do Roadmap, e cada Meta de Sprint, estabelecida no início de cada Sprint, representa a estratégia de negócios do Product Owner para se realizar a Visão do Produto, por meio do trabalho do Time de Desenvolvimento. Assim, cada Meta de Release ou de Roadmap definida representa um incremento à realização da Visão do Produto e cada Meta de Sprint acordada entre Product Owner e Time de Desenvolvimento é, preferi-

velmente, um incremento à realização da Meta da Release ou de Roadmap.

O valor de negócio de cada funcionalidade produzida durante o Sprint pelo Time de Desenvolvimento é, também preferivelmente, um incremento à realização da Meta do Sprint. Logo, é um pequeno incremento à realização da Visão do Produto (veja a figura 24.1).

Essa estratégia de negócios existe sob a responsabilidade do Product Owner e é, assim como o próprio desenvolvimento do produto, iterativamente definida, detalhada, reavaliada e modificada de acordo com o *feedback* obtido sobre o produto e mudanças em seu contexto.

24.2 META DE SPRINT

24.2.1 O que é a Meta de Sprint?

A Meta do Sprint é um objetivo determinado para o Sprint, a ser realizado a partir do desenvolvimento de itens do Sprint Backlog. Scrum define a Meta do Sprint de forma ampla, como qualquer nexo escolhido entre esses itens que leve os membros do Time de Desenvolvimento a trabalharem juntos, e não em diferentes iniciativas.

De forma mais estrita, recomendo fortemente que esse nexo traduza o valor produzido pelo Time de Desenvolvimento no Sprint. Ou seja, recomendo que a Meta do Sprint traduza uma necessidade ou objetivo de negócios ou do usuário a ser realizado como resultado do trabalho realizado no Sprint. Tratarei, neste capítulo, da Meta da Sprint seguindo essa definição mais estrita.

A Meta do Sprint é estabelecida e acordada entre Product Owner e Time de Desenvolvimento durante a reunião de Sprint Planning de cada Sprint. O Product Owner prioriza o Product Backlog e chega à reunião capaz de definir a necessidade dos clientes mais importante naquele momento. Colabora-se, então, para ajustá-la de acordo com a capacidade de produção do Time de Desenvolvimento, para então se estabelecer a Meta do Sprint. Essa meta determina qual objetivo de negócios ou do usuário deve ser realizado a partir do Incremento do Produto que estará pronto ao final do Sprint.

Esse Incremento do Produto é gerado ao longo do Sprint a partir dos itens selecionados para o Sprint Backlog, que serão desenvolvidos do que mais contribui para a realização da Meta do Sprint ao que menos contribui para tal.

A Meta do Sprint oferece alguma flexibilidade para o trabalho do Time de Desenvolvimento no Sprint, que se faz necessária devido à natural incerteza inerente ao planejamento do desenvolvimento de *software*.

Assim, embora o Time de Desenvolvimento faça o seu melhor para concluir todo o trabalho planejado, muitas vezes ele não será capaz de fazê-lo. Ao sobrarem itens não prontos ao término do tempo de desenvolvimento do Sprint, esses serão sempre os menos importantes, ou seja, os que menos contribuem para a realização da meta. Assim, o sucesso do Sprint não reside em completar-se tudo o que foi planejado, mas sim em realizar-se satisfatoriamente a Meta do Sprint a partir do que o Time de Desenvolvimento produziu no Sprint.

A Meta do Sprint guia, dá propósito, motiva e mantém o foco do trabalho do Time de Desenvolvimento no Sprint. Dessa forma, ela estimula e reforça a auto-organização, já que o Time de Desenvolvimento define como vai realizá-la e se torna responsável por fazê-lo acontecer. Ao mesmo tempo, essa meta representa um valor visível produzido pelo Time de Desenvolvimento durante o Sprint sobre o qual será oferecido *feedback* na reunião de Sprint Review por clientes e demais partes interessadas.

A Meta do Sprint representa um incremento à realização da Meta da Release ou de Roadmap e, assim, é também um incremento à realização da Visão do Produto, que é o objetivo mais amplo a ser realizado no projeto. O trabalho sobre os itens do Sprint Backlog representa um pequeno incremento à realização dessa visão.

A Meta do Sprint, por sua vez, é um alvo muito imediato e mais tangível tanto para o Time de Desenvolvimento quanto para o Product Owner. Ela provê uma conexão entre o valor de negócios dos itens do Product Backlog e a Meta da Release ou de Roadmap, que por sua vez a conecta com a Visão do Produto.

Uma vez definida na reunião de Sprint Planning, não pode mais ser alterada. Para garantir a integridade do Sprint, o ScrumMaster tem o papel de assegurar que não seja feita durante o Sprint nenhuma mudança que possa afetar a sua meta. No entanto, caso a sua Meta perca o sentido, o Sprint pode ser cancelado pelo Product Owner (veja [16.3 O Sprint pode ser cancelado?](#)).

24.2.2 Como é a Meta de Sprint?

Conforme expliquei anteriormente, a Meta do Sprint funciona como um tema ou um nexo para o Sprint, que permeia o trabalho do Time de Desenvolvimento em seu decorrer e leva seus membros a trabalharem juntos. Ela não é uma quantidade de itens a serem desenvolvidos. Ela também não é um item selecionado do Product Backlog, nem mesmo o item mais importante. Tampouco é a reescrita dos itens do Sprint Backlog por extenso, como “desenvolver X, Y e Z”.

Recomendo que essa meta seja, na realidade, uma e apenas uma necessidade de negócios ou do usuário razoavelmente ampla a ser realizada incrementalmente a partir do desenvolvimento dos itens selecionados para o Sprint Backlog, do mais importante para o menos importante.

Alguns Times de Desenvolvimento definem como Meta do Sprint um aprendizado a se alcançar ou a realização de alguma melhoria interna. Por exemplo: *buscamos aprender como realizar Testes de Aceitação automatizados com a ferramenta Cucumber*. Embora itens de aprendizado possam fazer parte do Sprint, desaconselho essa prática, exatamente porque está desalinhada com a minha recomendação de que a Meta do Sprint represente um objetivo que represente valor para os clientes ou usuários.

Caso os itens selecionados para o Sprint Backlog tenham pouca ou nenhuma relação uns com os outros, será muito difícil se estabelecer uma Meta coerente para o Sprint. Na realidade, essa desconexão entre os itens escolhidos para o Sprint pode sinalizar que não há uma estratégia clara de produto sendo utilizada. Ou seja, não se está caminhando para realizar incrementalmente uma Visão de Produto, o que é responsabilidade do Product Owner.

Como consequência, além da falta de direção, o trabalho sem um propósito claro, realizado a partir de uma sequência de partes desconexas ou pouco relacionadas, é geralmente desmotivador para os membros do Time de Desenvolvimento.

Um formato que proponho para as Metas de Sprint é:

(Por meio deste Sprint) possibilitaremos a <QUEM> <O QUÊ>

Onde <QUEM> pode ser uma categoria de usuários ou um cliente ou uma parte interessada específica, e <O QUÊ> define qual é o objetivo ou necessidade de negócios ou do usuário a ser atendida. Podemos ver exemplos seguintes para produtos de *software*:

- *Possibilitaremos ao Comprador comprar um livro.*
- *Possibilitaremos à Livraria realizar uma venda básica de um livro.*
- *Possibilitaremos ao Administrador um login mais seguro.*

O primeiro exemplo mostra a perspectiva de um usuário, o Comprador, enquanto o segundo mostra a perspectiva do negócio, representado pela Livraria. Ambas são válidas e levarão, provavelmente, ao mesmo resultado. Como exemplo, imagino que os itens mais importantes do Sprint Backlog tratarão do que é essencial para uma compra básica, de forma a se realizar minimamente essa meta. Assim, poderemos ter a busca pelo título do livro, a inserção de dados básicos de entrega e o pagamento com cartão de crédito.

Itens mais abaixo tratarão de questões relacionadas com gradativa menor importância para essa meta, como por exemplo, a busca pelo autor e o pagamento com débito em conta, entre outros. Ao serem desenvolvidos, estes enriquecerão a qualidade da realização da Meta da Sprint.

Entre os itens, pode haver também, por exemplo, ajustes de Sprints anteriores, como correções de problemas ordenadas de acordo com sua importância relativa aos outros itens, e até mesmo itens pouco ou nada relacionados, estes em geral com menor prioridade.

Já o terceiro exemplo traz uma outra possibilidade de meta válida, definida pela perspectiva do usuário Administrador.

24.3 META DE RELEASE/ROADMAP

24.3.1 O que é a Meta de Release/Roadmap?

A Meta da Release é um objetivo ou necessidade de alto nível, de negócios ou do usuário, a ser realizada por meio do trabalho do Time de Desenvolvimento para uma entrega (Release). Ela representa um incremento à realização da Visão do Produto.

A Meta da Release geralmente existe quando se trabalha com uma Release planejada. Ela forma, juntamente com a data da Release e um conjunto de itens selecionados, o Plano da Release (veja [22.1 O que é a Release Planning?](#)).

A Meta de Roadmap é um objetivo ou necessidade de alto nível, de negócios ou do usuário, que representa um incremento à realização da Visão do Produto, e é similar à Meta da Release. Sua diferença é que ela representa um marco no Roadmap do Produto (uma data, aproximada ou exata) e não é necessariamente realizada no momento de uma entrega.

A Meta de Roadmap é útil quando não faz sentido se ter uma Meta de Release. Um exemplo disso ocorre quando se realizam Releases em toda e cada Sprint, caso em que a Meta da Release e a Meta do Sprint seriam a mesma. Outro exemplo se dá quando se trabalha com entrega contínua, em que cada Meta de Release corresponderia ao valor de negócios do item entregue. Nesses casos, a granularidade da Meta da Release seria fina demais, perdendo-se seu sentido (veja [21.2 Como é a Release?](#)).

A Meta de Release também pode representar um marco no Roadmap do Produto. Nesses casos, cada um desses marcos corresponde a uma entrega. Veja no exemplo da figura [24.2](#) um Roadmap de um produto de *software* para vendas pela internet.

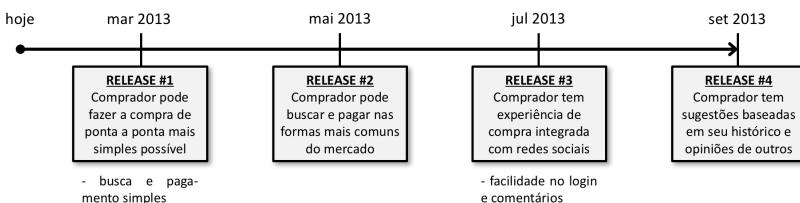


Fig. 24.2: Roadmap do Produto com Metas de Release

Assim como a Visão do Produto, a Meta da Release ou de Roadmap fornece contexto, orientação, motivação e inspiração para todo o trabalho de desenvolvimento do produto até o momento da Release ou da chegada do marco do Roadmap.

24.3.2 Como é a Meta de Release/Roadmap?

Não existem formatos prescritos para as Metas de Release ou de Roadmap. Essas Metas são necessidades a serem atendidas em uma dada janela de tempo, e essas necessidades podem ser descritas a partir de seus usuários ou de seus clientes.

Um formato que sugiro neste livro para as Metas de Release ou de Roadmap, similar ao da Meta do Sprint, é:

(Por meio da Release)/(Até <QUANDO>), possibilitaremos a <QUEM> <O QUÊ>.

Onde <QUANDO> se refere a um identificador de Release (número ou nome, por exemplo) ou à data da Release ou do marco do Roadmap; <QUEM> pode ser uma categoria de usuários ou um cliente ou uma parte interessada específica; e <O QUÊ> define qual é a necessidade a ser atendida. Pode-se ver dois exemplos a seguir para produtos de *software* distintos, o primeiro pela perspectiva do usuário e o segundo pela perspectiva do negócio:

- *Por meio da Release #3, possibilitaremos ao Comprador ter uma experiência de compra integrada com redes sociais.*
- *Até o final do primeiro semestre, possibilitaremos à loja online oferecer seus produtos para os usuários do sistema.*

24.4 ROADMAP DO PRODUTO

24.4.1 O que é o Roadmap do Produto?

O Roadmap do Produto é um plano em alto nível de como o produto vai evoluir ao longo do tempo até um momento futuro determinado, que pode ser o final previsto para o projeto ou algum momento futuro relevante. Assim, ele é útil apenas quando é possível se imaginar, em alto nível, como o produto evoluirá no futuro, mesmo que apenas em termos de objetivos a serem realizados.

Em um projeto com Scrum, o Product Owner é o responsável por criar, manter e comunicar o Roadmap do Produto. O Product Owner geralmente cria o Roadmap do Produto em uma ou mais sessões de trabalho em que podem estar presentes diferentes partes interessadas e o Time de Desenvolvimento.

O Roadmap do Produto facilita o diálogo entre o Time de Scrum, os clientes e as demais partes interessadas sobre a evolução do produto, indicando a todos o que se pretende realizar em alto nível, ao longo do projeto. Ele é também útil na estratégia da organização, uma vez que ajuda a coordenar o desenvolvimento e entregas de produtos relacionados, atividades necessárias para essas entregas e outras atividades estratégicas que impactam ou são impactadas por elas.

24.4.2 Como é o Roadmap do Produto?

O Roadmap do Produto é uma linha do tempo que contém marcos, que são datas exatas ou aproximadas no futuro, e metas do produto a serem realizadas até cada uma delas (veja a figura 24.3).

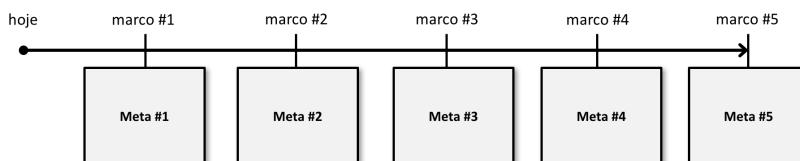


Fig. 24.3: Roadmap do Produto

O Roadmap do Produto em um projeto que utiliza Scrum mostra o caminho incremental para a realização da Visão do Produto. Assim, sugiro que se indique em cada um de seus marcos a meta de negócios ou do usuário que se quer realizar, que é a da Meta de Release ou de Roadmap esperada.

Pode-se também adicionar a cada marco objetivos parciais, sejam técnicos, de usuário ou de negócios. Embora não seja recomendável, é também aceitável haver mais de uma Meta de Release ou de Roadmap por marco, em geral direcionadas a diferentes classes de usuários, clientes ou demais partes interessadas.

24.5 VISÃO DE PRODUTO

24.5.1 O que é a Visão do Produto?

A Visão do Produto é um objetivo ou necessidade de negócios ou de usuário de alto nível que fornece contexto, alinhamento, orientação, motivação e inspiração para o trabalho de desenvolvimento do produto durante todo o projeto. É o objetivo maior a ser realizado pelo Time de Scrum, que se traduz na satisfação dos clientes (PICHLER, 2010).

A Visão do Produto é estabelecida antes que o desenvolvimento do produto se inicie e, de forma geral, permanece estável durante todo o projeto. Ela é criada, gerenciada e compartilhada pelo Product Owner, que garante que o Product Backlog esteja sempre alinhado com ela. No entanto, o Time de Desenvolvimento, os clientes e quaisquer outras pessoas relevantes interessadas podem estar diretamente envolvidos no refinamento dessa Visão.

24.5.2 Como é a Visão do Produto?

A Visão do Produto provê um alinhamento entre todos os envolvidos no projeto, desde os clientes e demais partes interessadas até o Time de Scrum, sobre o que deve ser realizado. Para que esse entendimento comum seja possível, é necessária uma Visão do Produto de fácil compreensão e, assim, é desejável que seja concisa e clara, contendo apenas o necessário e suficiente.

Assim, um documento de dezenas de páginas contendo detalhes do que será o produto não representa uma boa Visão de Produto. Aqui apresento

algumas técnicas úteis para se representar essa visão.

Teste do Elevador

Geoffrey Moore, no seu livro *Crossing the chasm* (2001), apresenta um modelo interessante para a Visão do Produto, o chamado “Teste do Elevador” ou “Elevator Pitch”. A ideia original é que seja possível explicar o que é o produto para um investidor durante a subida de um elevador, ou seja, em um tempo bastante curto, de forma a convencê-lo a investir no produto.

Jim Highsmith adaptou o modelo para a Visão do Produto em projetos Ágeis. Aqui, apresento essa versão com pequenas modificações:

Para (cliente-alvo),
que (problema que o cliente-alvo enfrenta),
o (nome do produto) **é um** (categoria do produto)
que (benefício-chave, razão convincente para utilizar).
Ao contrário de (alternativa primária competitora),
nossa produto (diferenciação primária).

Um exemplo de aplicação para um *site* de relacionamentos pode ser visto em seguida:

Para usuários da internet solteiros em busca de um relacionamento sério,
que estão insatisfeitos com encontros malsucedidos e a oferta de numerosas opções pouco criteriosas,
o LoveFinder **é um** *site* de relacionamentos amorosos
que os ajuda a encontrar sua alma gêmea.
Ao contrário de *sites* de encontros populares,
nossa produto oferece pretendentes compatíveis com as preferências do usuário, apresentando detalhes suficientes que incluem fotos.

Outro exemplo, agora para um aplicativo móvel de viagens, pode ser visto a seguir:

Para turistas usuários de smartphone,
que estão cansados de serem oferecidos sempre as mesmas viagens óbvias e sem graça,
o MyTrip é um aplicativo móvel de viagens
que sugere roteiros diários flexíveis de acordo com seu perfil.
Ao contrário de guias de viagens com roteiros predefinidos,
nossa produto elabora trajetos personalizados e adaptáveis.

O “Teste do Elevador” é um enunciado de Visão do Produto curto e efetivo ao informar, de forma priorizada, quem são os clientes-alvo do produto, qual o problema desses clientes será resolvido, um nome e categorização que fornecem um posicionamento do produto no mercado, uma razão convincente para utilizar o produto e a diferenciação do produto diante das alternativas existentes.

Para **cliente-alvo**, usamos o usuário final. Caso haja dois lados de usuários, dou em geral preferência a se utilizar o usuário mais na ponta. Assim, para um aplicativo para conectar passageiros a motoristas, por exemplo, escolho sempre o passageiro, e não o motorista.

Para o **problema enfrentado**, busco sempre ir na “dor” do cliente-alvo, e não em uma necessidade. Assim, no primeiro exemplo, o problema não seria que *querem receber boas opções para encontros*. Utilizar a “dor” que estão enfrentando, como no exemplo, causa uma empatia muito maior.

Escolha um **nome** relevante para o produto e uma **categoria** que ajude a melhor se localizar qual o tipo de produto. Como guia, pergunte-se em que categoria da loja de aplicativos de celular esse produto estaria relacionado ou em que departamento estaria disponível caso fosse vendido em uma loja de departamentos.

O **benefício-chave** é a principal razão pela qual o cliente-alvo compraria o produto, ou seja, o valor central que o produto traz para o usuário. A **alternativa primária** pode ser o nome de um produto concorrente direto ou a forma atual de se resolver problemas similares, como por exemplo, ...*ao contrário do trabalho manual*.

A **diferenciação primária** define como esse produto se diferencia da alternativa primária competidora. Em vez de listar os aspectos negativos da

alternativa primária junto a ela, é preferível listar na diferenciação primária os aspectos positivos do produto em comparação à alternativa primária.

Caixa da Visão do Produto

A criação da Caixa da Visão do Produto é uma atividade lúdica, em geral realizada pelo Time de Desenvolvimento em conjunto com o Product Owner, para criarem uma caixa para o produto a ser desenvolvido. O resultado desse trabalho representa o que seria a caixa do produto caso ele fosse um produto de prateleira, vendido em uma loja, embora não haja nenhuma necessidade de que de fato o seja.

A parte da frente da caixa tem o objetivo de atrair o “comprador”, ao passar pela loja e ver a caixa na prateleira. Assim, desenha-se essa parte da frente com um nome atrativo, uma imagem relacionada relevante e alguns tópicos curtos e diretos para vender o produto.

A parte de traz da caixa, uma vez que o possível comprador já está com a caixa em suas mãos, vai ajudá-lo na decisão de compra. Assim, constará nessa parte da caixa uma descrição mais detalhada dos atributos do produto, que pode ser um parágrafo ou tópicos, e requisitos ou restrições de uso (veja a figura 24.4).

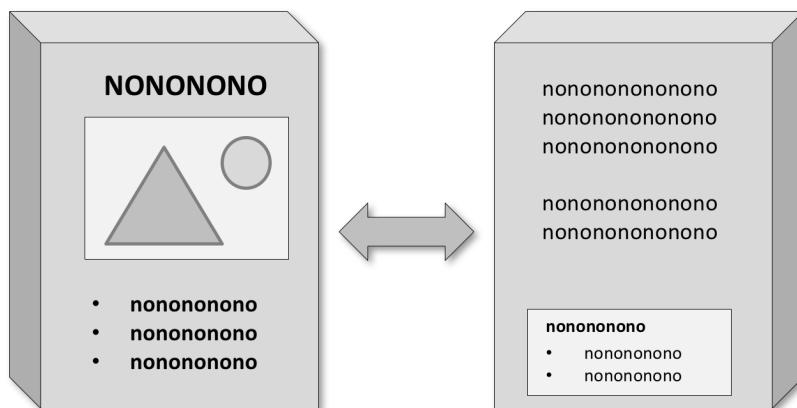


Fig. 24.4: Caixa da Visão do Produto

Ao criar a caixa, os membros do Time de Desenvolvimento e Product Owner colaboraram para criar um alinhamento, ainda em alto nível, sobre o que é o produto a ser desenvolvido. Geralmente, utilizo a Caixa da Visão do Produto em conjunto com o Teste do Elevador, de forma a complementá-lo. Já vi diversos times deixarem a caixa visível, ao alcance para consulta durante todo o desenvolvimento do produto.

É/Não é/Faz/Não faz

Essa é uma técnica que criei para realizar exercícios sobre o papel do Scrum em minhas aulas. Um grande amigo, Paulo Caroli, que escreveu o prefácio desta edição do livro, esteve presente em uma dessas aulas e adaptou a técnica para a definição da Visão de Produto em seu trabalho. Hoje, usamos essa técnica na minha empresa para esse fim e para diversos outros, como por exemplo, contratar novos colaboradores.

Divide-se um quadro em quatro quadrantes: é, não é, faz, não faz, e utilizam-se notas adesivas (veja a figura 24.5). Um grupo de pessoas deve colaborar em torno desse quadro. Caso possível, esse grupo pode idealmente ser formado por clientes, partes interessadas e Product Owner, mas em geral utilizamos Product Owner e Time de Desenvolvimento, ou Product Owner e um ou poucos clientes.

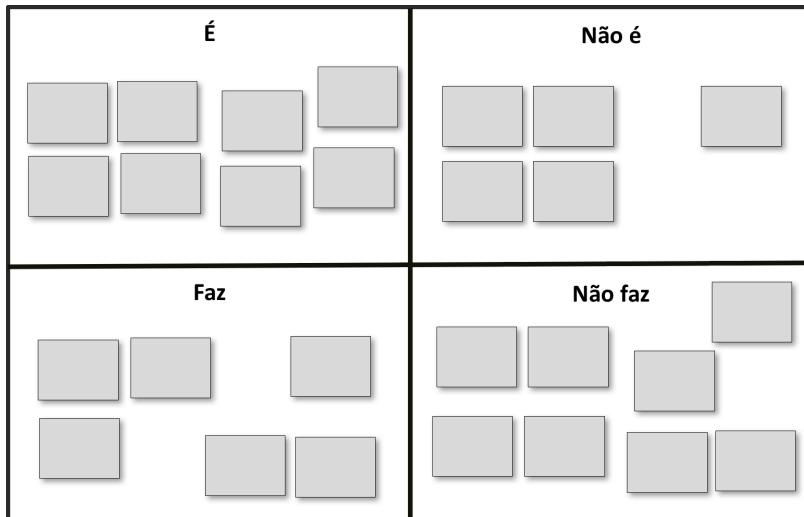


Fig. 24.5: Quadro do É/Não é/Faz/Não faz

Colaborativamente, os presentes escrevem em notas adesivas:

- **é** — o que esperam que o produto seja, em termos de suas características.
- **não é** — o que esperam que o produto não seja. Aqui, busca-se listar aquelas características que erradamente se imaginaria que o produto terá, delimitando assim a definição do produto.
- **faz** — o que esperam que o produto seja capaz de realizar, em alto nível. Aqui, busca-se preferivelmente listar os problemas que o produto deve resolver, mas também pode-se listar suas funcionalidades em alto nível.
- **não faz** — o que esperam que o produto não vai realizar, em alto nível. Aqui, busca-se preferivelmente listar os problemas que erradamente se imaginaria que o produto deve resolver, mas também pode-se listar as funcionalidades em alto nível que erradamente se imaginaria que o produto deve ter, delimitando assim sua atuação.

Geralmente, uso essa técnica em conjunto com o Teste do Elevador, de forma a complementá-lo. Entretanto, raramente utilizo-a ao mesmo tempo que a Caixa da Visão do Produto.

CAPÍTULO 25

User Stories: representando o trabalho

25.1 O QUE É A USER STORY?

User Story, ou “história de usuário”, é uma descrição concisa e simples de uma necessidade do usuário do produto sob o seu ponto de vista. As User Stories foram criadas pelos autores do Extreme Programming para substituir integralmente, em projetos Ágeis, o que chamamos de “requisitos” em projetos tradicionais (representados, por exemplo, por Casos de Uso), mas com uma abordagem diferente.

Um dos princípios por trás das User Stories é o de que o produto poderia ser inteiramente representado por meio das necessidades de seus usuários (JEFFRIES et al., 2000). O produto desenvolvido com Scrum é descrito por

meio de itens do Product Backlog. Logo, de acordo com esse princípio, cada um desses itens deveria ser representado no formato de User Stories. Ou seja, uma User Story pode representar um e apenas um item do Product Backlog.

Em seu uso com Scrum, as User Stories são, a princípio, escritas pelo Product Owner e refinadas para o trabalho em conjunto com o Time de Desenvolvimento.

Veja um exemplo de User Story na figura 25.1.

	<p><i>Eu, Comprador de Livros, quero buscar um livro pelo título para escolher comprá-lo</i></p>

Fig. 25.1: Exemplo de User Story, representando uma necessidade do usuário

As User Stories não fazem parte do *framework* Scrum e, assim, seu uso é opcional.

A User Story é apenas uma promessa de uma conversa, um lembrete de que mais detalhes serão necessários para que ela seja colocada em desenvolvimento e, antes disso, ela não deve ser considerada suficiente para a realização do trabalho. Ela é o começo, mas deve ser seguida por uma série de conversas entre as pessoas de negócios (no Scrum, em geral, apenas o Product Owner) e os membros do Time de Desenvolvimento, para então se iniciar o seu desenvolvimento. Essas conversas visam a definir e capturar os detalhes de negócios necessários para o desenvolvimento da funcionalidade que atenderá a essa necessidade do usuário.

Os detalhes de negócios podem ser documentados de diferentes formas e anexados à sua User Story correspondente. Acredita-se, no entanto, que apenas Critérios e Testes de Aceitação da User Story, descritos mais adiante,

já representem documentação suficiente, uma vez que devem cobrir todos os aspectos de negócios do requisito.

Cada item do Product Backlog (e, portanto, do Sprint Backlog) que levará ao desenvolvimento de uma funcionalidade, pode ser representado no formato de User Story. Recomenda-se ainda que todas e quaisquer questões técnicas, sempre que possível, façam parte de User Stories. O mesmo serve para necessidades de pesquisa ou de investigação por parte do Time de Desenvolvimento.

No entanto, algumas questões técnicas que perpassam mais de uma User Story, como a refatoração de uma classe ou a população de um banco de dados, por exemplo, assim como a correção de problemas, dificilmente podem ser descritas sob a perspectiva do usuário. Dessa forma, dificilmente podem ser encaixados em uma User Story.

Embora pertençam ao Product Backlog, não devem, assim, ser representadas por User Stories. Para estas, alguns times utilizam um formato parecido e as chamam de “histórias técnicas”. Porém, não as recomendo. Prefiro que não se use, por exemplo, *eu, enquanto banco de dados, necessito ser populado...*, já que bancos de dados não são usuários e isso apenas criaria confusão.

25.2 COMO É A USER STORY?

A User Story possui três aspectos críticos, chamados de “os três C’s”: o Cartão, as Conversas e a Confirmação. Veremos em seguida o que cada um desses aspectos significa (COHN, 2004).

25.2.1 Cartão

O Cartão é a descrição da necessidade do usuário, ou seja, a própria User Story. Essa descrição é concisa e suficiente para apenas identificar qual é de que se trata essa necessidade. O nome “Cartão” é dado porque inicialmente as User Stories eram escritas em cartões de índice ou fichas.

Não existe formato obrigatório para se escrever User Stories. O padrão mais conhecido utiliza-se de três parâmetros: “QUEM”, “O QUÊ” e “POR QUÊ” (COHN, 2004).

QUEM define quem é o usuário que tem a necessidade. Pode ser representado por um tipo de usuário do produto (como “Comprador de Livros” ou “Administrador do Sistema”, por exemplo), por uma *persona* ou até mesmo por um usuário específico, uma pessoa.

“QUEM” ajuda a criar no leitor da User Story uma imagem mental desse usuário e a gerar uma empatia com ele, permitindo que o leitor se coloque em seu lugar. User Stories que começam “Eu, usuário...” ou “Eu, cliente...”, portanto, não atingem esse objetivo e devem ser evitadas.

Personas são frequentemente utilizadas com esse propósito. Uma *persona* é um usuário fictício que representa um grupo distinto de usuários do produto, que tem comportamentos e objetivos específicos em comum. Assim, para um produto, são criadas diferentes *personas* que representem grupos de usuários que terão diferentes objetivos atendidos. Personas são largamente usadas por profissionais de *Marketing* e de *design* de experiência do usuário.

A *persona* possui um nome, uma imagem ou foto, um objetivo claro, características, comportamentos e motivações, além de um contexto que a humanize, traga empatia e, possivelmente, a coloque no contexto do uso do produto sem, em princípio, mencionar o produto. O nome da persona pode ser totalmente inventado ou pode-se pegar emprestado o nome de alguma personalidade conhecida que compartilhe características com essa *persona*. Alternativamente, pode-se usar parcialmente o nome da personalidade conhecida, de forma a não se fechar demais em suas características.

Em geral, para um produto criam-se apenas três ou quatro *personas*. No entanto, quando há mais de uma ponta no produto (por exemplo, em um aplicativo que conecta motoristas a passageiros), pode-se criar três ou quatro personas por ponta, se assim fizer sentido.

Na figura 25.2, descrevemos “Sara Sozinha”, uma *persona* de um site de relacionamentos.

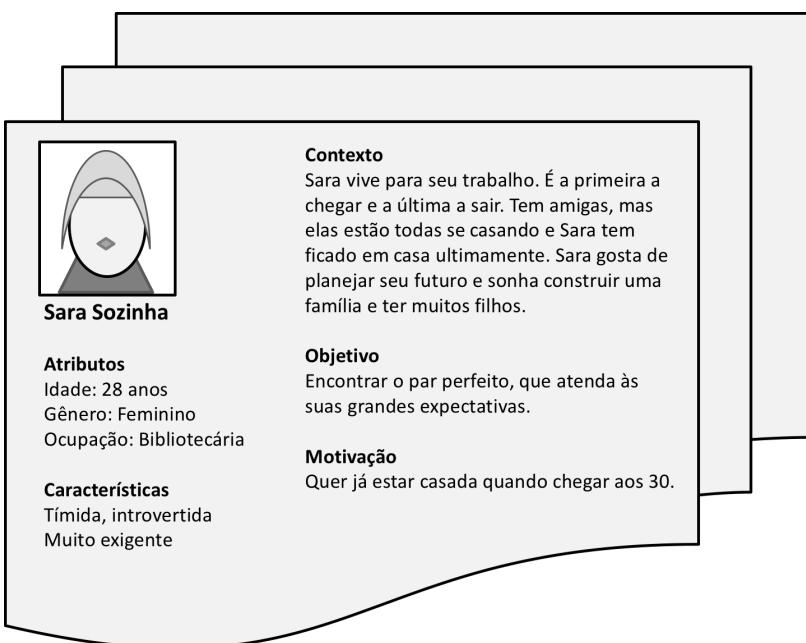


Fig. 25.2: Exemplo de persona

Uma User Story para essa *persona* poderia ser algo assim: *para encontrar alguém com interesses parecidos com os meus, eu, Sara Sozinha, quero comparar os interesses de diferentes candidatos.*

O QUÊ define qual é a necessidade do usuário. Muitas vezes, em projetos tradicionais, os requisitos do produto são representados apenas por essa parte.

POR QUÊ define qual o benefício do usuário ao ter a funcionalidade desenvolvida para atender a essa necessidade. Aqui, deve-se ter cuidado para não se remeter a um objetivo mais amplo (por exemplo, a Visão do Produto ou a Meta do Sprint). O “por quê” é o valor imediato obtido pelo usuário, ou seja, o que ele passa a ser capaz de realizar ao ter esse “o quê” atendido.

Na figura 25.3, vemos o padrão mais comum para escrita de User Stories:

	<i>Eu, enquanto <QUEM>, quero <O QUÊ> para <POR QUÊ></i>

Fig. 25.3: Padrão de escrita de uma User Story

Esse formato, embora largamente utilizado, não é obrigatório. Podemos ver na figura 25.4 uma outra forma de dizer o mesmo, mas com ênfase no benefício do usuário.

	<i>Para <POR QUÊ>, enquanto <QUEM>, eu quero <O QUÊ></i>

Fig. 25.4: Outro padrão de escrita de uma User Story

Outra forma simples de se escrever a User Story é: “Um <QUEM> pode <O QUÊ> para <POR QUÊ>”. Por exemplo, se imaginarmos como produto um aplicativo móvel para viagens, uma de suas User Stories poderia ser: *Um Viajante de Turismo quer mostrar sua viagem para seus amigos para lhes causar admiração*.

Ao escrever uma User Story em algum desses formatos, o Product Owner

realiza uma profunda reflexão para estabelecer quem é o usuário da funcionalidade a ser desenvolvida e qual o benefício que esse usuário obtém. Isso lhe permite manter os itens do Product Backlog alinhados à Visão do Produto. Ao mesmo tempo, a User Story ajuda o Time de Desenvolvimento a manter seu foco no usuário e em qual benefício é esperado que ele obtenha.

Discussão: Ponto de vista do usuário

O ponto de vista do usuário é o de quem tem um problema ou uma solução? Um problema, certo? Vejamos, a seguir, o exemplo de User Story que utilizamos anteriormente:

Eu, Comprador de Livros, quero buscar um livro pelo título para escolher comprá-lo.

Essa é a forma mais comum de se escreverem User Stories e é perfeitamente aceitável. No entanto, buscar um livro por título é um problema ou uma solução? Uma solução. Logo, essa User Story não está realmente expressa sob a perspectiva do usuário. Qual seria o problema do usuário, então? Se a reescrevermos da seguinte forma:

Eu, Comprador de Livros, quero encontrar um livro de que sei o título para escolher comprá-lo.

Nesse caso, o problema do usuário é encontrar um livro cujo título ele conhece. Uma entre as diferentes possíveis soluções é buscar livros por título, com uma caixa de texto e um botão, por exemplo. Outra seria mostrar simplesmente uma lista de todos os livros ordenados por título para que ele possa encontrar o que precisa. Essa alternativa poderia, por exemplo, ser mais adequada para uma versão inicial do sistema devido a seu custo mais baixo.

Expressar a User Story a partir do problema e não de uma solução particular abre espaço para discussões sobre as possíveis soluções para o mesmo problema. Assim, pode fazer com que soluções mais adequadas sejam encontradas. A solução escolhida pode ser integralmente descrita nos Critérios de Aceitação, descritos mais adiante.

25.2.2 Conversas

São conversas sobre a User Story, nas quais a solução de negócios e os detalhes dessa solução são discutidos, negociados, definidos e, então, documentados na forma de Critérios e Testes de Aceitação. Elas geram uma compreensão compartilhada sobre o que é necessário para que a funcionalidade a ser desenvolvida gere valor de negócio e retorno ao investimento.

As conversas são imprescindíveis para o trabalho do Time de Desenvolvimento, uma vez que o Cartão não possui detalhes que permitam que o item seja desenvolvido. Em geral, participam dessas conversas o Product Owner, membros do Time de Desenvolvimento e outras pessoas de negócios envolvidas, caso haja. Qualquer pessoa capaz de contribuir com detalhes pode ser convidada a participar.

Tomemos como exemplo a User Story do exemplo anterior: *um Viajante de Turismo quer mostrar sua viagem para seus amigos para lhes causar admiração*. O Time de Desenvolvimento e Product Owner, por meio de conversas, podem concluir que permitir o compartilhamento de fotos e vídeos na rede social do momento diretamente do aplicativo é a solução de negócios desejada. A partir daí, escrevem os Critérios de Aceitação correspondente, que serão transformados em Testes de Aceitação automatizados no decorrer do Sprint.

As conversas, que podem ocorrer em qualquer momento, são geralmente parte das sessões de Refinamento do Product Backlog (veja [23 Refinamento do Product Backlog](#)).

25.2.3 Confirmação

A confirmação é o resultado das conversas descritas anteriormente. São critérios informais e testes deles derivados que documentam os detalhes da User Story, definindo seus limites. Ou seja, são regras que estabelecem como a funcionalidade deve se comportar uma vez implementada. Esses critérios são chamados de Critérios de Aceitação e os testes, de Testes de Aceitação.

Enquanto todas as User Stories devem satisfazer à mesma Definição de Pronto, cada User Story possui seu próprio conjunto de Critérios e Testes de

Aceitação. Critérios de Aceitação são, em geral, expressos por enunciados pequenos e de fácil entendimento. São usados para determinar quando a funcionalidade produzida pelo Time de Desenvolvimento está completa e, assim, nada mais deve ser adicionado a ela.

Os Critérios de Aceitação ajudam o Product Owner a determinar para o Time de Desenvolvimento o que ele precisa para que essa funcionalidade propicie valor. A partir desses critérios, o Time de Desenvolvimento gera os Testes de Aceitação.

Os Critérios de Aceitação são negociados e definidos antes do desenvolvimento da funcionalidade, geralmente em sessões de Refinamento do Product Backlog.

Como exemplo, vamos imaginar que a seguinte User Story é forte candidata a ser desenvolvida no próximo Sprint:

Eu, Comprador de Livros, quero utilizar meu cartão de crédito no pagamento dos livros escolhidos, para ter praticidade e segurança no pagamento.

Assim, em uma das sessões de Refinamento do Product Backlog, os seguintes Critérios de Aceitação podem ser definidos para a User Story, entre outros:

- **Critério #1:** somente podemos aceitar cartões de crédito com bandeiras com que temos convênio.
- **Critério #2:** somente podemos aceitar cartões de crédito com data de expiração no futuro.

Já vi vários times utilizando como confirmação, em uma abordagem mais simples, apenas o quadro branco preenchido como resultado das conversas realizadas entre Product Owner e Time de Desenvolvimento. Esse quadro

é preservado e permanece visível durante todo o Sprint e, a partir dele, são escritos os Testes de Aceitação e as funcionalidades são desenvolvidas.

Testes de Aceitação são criados a partir de aplicações de exemplos aos Critérios de Aceitação, a partir dos quais se verificam se entradas dadas estão produzindo os resultados esperados. Eles servem para verificar se a funcionalidade está se comportando conforme esperado, sob o ponto de vista do usuário.

Ao se trabalhar com Testes de Aceitação automatizados e executados frequentemente, reduzem-se os riscos de que mudanças no produto, muito comuns em projetos Ágeis, gerem erros que passem desapercebidos. Existem inúmeras ferramentas para fazer essa automatização e diversas inclusive permitem sua escrita em linguagem humana.

A seguir, vemos a aplicação de Testes de Aceitação aos Critérios de Aceitação definidos anteriormente. Para efeitos didáticos, os Testes de Aceitação estão descritos aqui de uma forma textual simplificada. Eles são, na realidade, um *script* que é percorrido por uma ferramenta, simulando usuários realizado todas as ações imaginadas e verificando se o sistema está respondendo como deveria.

Para a User Story e os Critérios do exemplo anterior:

Critério #1: somente podemos aceitar cartões de crédito com bandeiras com que temos convênio.

Testes de aceitação:

- Comprador de Livros utiliza cartão de crédito Visa
 - Aceitou = correto.
 - Recusou = errado, deve ser corrigido!
- Comprador de Livros utiliza cartão de crédito MasterCard
 - Aceitou = correto.
 - Recusou = errado, deve ser corrigido!

- Comprador de Livros utiliza cartão de crédito Amex
 - Recusou = correto.
 - Aceitou = errado, deve ser corrigido!

Critério #2: somente podemos aceitar cartões de crédito com data de expiração no futuro.

Testes de aceitação:

- Comprador de Livros utiliza cartão de crédito com expiração em 01/01/2020
 - Aceitou = correto.
 - Recusou = errado, deve ser corrigido!
- Comprador de Livros utilizou cartão de crédito com expiração em 01/01/2000
 - Recusou = correto.
 - Aceitou = errado, deve ser corrigido!
- Comprador de Livros utilizou cartão de crédito com expiração hoje
 - Aceitou = correto.
 - Recusou = errado, deve ser corrigido!

Nesses exemplos, pode-se ver claramente que as bandeiras de cartão aceitas são Visa e MasterCard, enquanto que Amex não. Pode-se ver também que cartões de crédito expirando no dia da compra serão aceitos.

Para alguns Critérios de Aceitação, como por exemplo *o botão deve ser azul* ou *o gráfico resultante deve ser composto por dois retângulos e um triângulo*, não é possível gerar Testes de Aceitação e, assim, não podem ser automatizados. Por meio de testes exploratórios, os membros do Time de Desenvolvimento realizam a verificação desses critérios.

Os Critérios e Testes de Aceitação cobrem cada aspecto de negócios da User Story e, assim, são geralmente suficientes para documentá-las.

25.3 CRIANDO BOAS USER STORIES

Bill Wake (2003), autor do livro *Extreme programming explored*, descreveu em seu blog quais seriam as características de boas User Stories. Ele formou o acrônimo INVEST (“investir”, em inglês) com a primeira letra de cada uma dessas características, afirmando que devemos “investir em boas User Stories”.

De acordo com Wake, uma User Story deve ser:

- **independente** — User Stories devem ser tão independentes ou desacopladas umas das outras quanto possível. User Stories com sobreposições de conceitos com outras User Stories devem ser evitadas. Busca-se ainda ser viável alterar livremente a ordem que serão desenvolvidas e, ao fazê-lo, não ser necessário alterar suas estimativas, o que nem sempre é viável.

Cabe adicionar que uma User Story se traduzirá sempre em uma funcionalidade de ponta a ponta, que representa valor para o usuário. Além disso, deve ser possível entender uma User Story sem ser necessário ler quaisquer outras;

- **negociável e negociada** — seus detalhes serão discutidos, negociados e definidos entre as pessoas de negócios (em geral o Product Owner, no Scrum) e o Time de Desenvolvimento. São as “Conversas”, dos três Cs;
- **valiosa** — devem representar valor de negócio para os clientes do projeto. Ao dividir-se uma User Story, o resultado dessa divisão deve ser User Stories menores que também representem funcionalidades de ponta a ponta, e não partes de trabalho técnico;
- **estimável** — o Time de Desenvolvimento deve possuir detalhes suficientes — tanto técnicos quanto de negócios — para estimar o trabalho de se transformar a User Story em parte do produto, de forma que o Product Owner possa ordená-la apropriadamente.

Assim, conversas suficientes devem ser realizadas entre Product Owner e os membros do Time de Desenvolvimento para se obter os detalhes de negócios da User Story a ser desenvolvida. Os membros do Time de

Desenvolvimento devem discutir possíveis soluções e executar pequenos experimentos, caso necessário, para reduzir os riscos técnicos da User Story. É claro que o nível de detalhes necessário depende de quão próxima a User Story está de ser colocada em desenvolvimento.

Estimativas fornecem o custo de desenvolvimento de uma User Story. Mesmo caso não se utilizem estimativas, os detalhes são necessários para dividir as User Stories de forma que fiquem pequenas o suficiente para serem desenvolvidas;

- **pequena (*small*, em inglês) ou dimensionada apropriadamente** — apenas User Stories pequenas e com um bom nível de detalhes podem ser colocadas em desenvolvimento. Quanto mais para o alto do Product Backlog a User Story estiver, de menor granularidade ela será.

User Stories pequenas próximas a seu desenvolvimento têm maior precisão em suas estimativas, permitem um melhor ordenamento do Product Backlog e representam um menor risco ao possibilitarem que mais itens façam parte de um Sprint;

- **testável** — deve ser possível verificar e confirmar que a User Story está pronta, ou seja, que foi transformada em parte do produto e está funcionando conforme esperado. A verificação é realizada por meio dos Critérios e Testes de Aceitação. É a “Confirmação”, dos três C’s.

25.4 ÉPICOS E TEMAS

O Product Backlog é ordenado, de forma que seus itens são tão menores e mais detalhados quanto mais acima estiverem nessa lista. Os itens da parte superior do Product Backlog são pequenos e possuem detalhes suficientes que os permitem serem aceitos no próximo Sprint para desenvolvimento. Os itens mais abaixo são cada vez maiores e menos detalhados.

Épicos são User Stories que representam itens grandes demais ou sem detalhes suficientes para serem desenvolvidos. Enquanto épicos, essas User Stories somente necessitarão de mais detalhes e serão fatiados em User Stories menores quando ganharem prioridade.

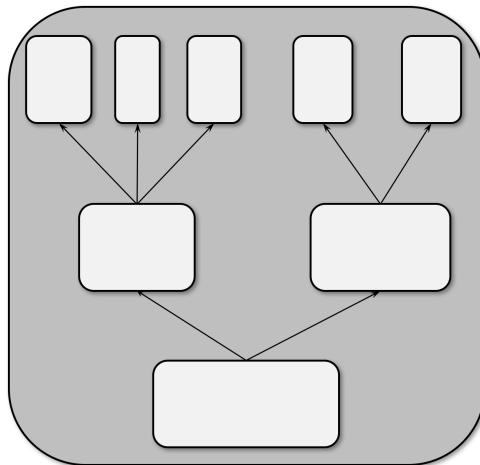


Fig. 25.5: Épicos evoluem para User Stories menores

A medida que o problema endereçado pelo épico ganha prioridade, User Stories menores evoluem a partir dele (veja a figura ??). Nesse momento, não se deve fatiar inteiramente o épico em todas as User Stories possíveis. Ao contrário, fatia-se o épico em uma ou em algumas poucas User Stories que representem apenas as partes mais importantes do problema.

Assim, por ter menor importância naquele momento, o resto segue no Product Backlog como um épico de menor prioridade, até que mais fatias ganhem importância, ou até que o problema tenha sido inteiramente endereçado, momento em que o épico deixa de existir.

Temas são coleções ou conjuntos de User Stories que estão relacionadas e, assim, podem ser agrupadas. Razões para agrupar User Stories em temas incluem, entre outras:

- juntas definem uma meta de negócios específica;
- são provenientes de um mesmo épico;
- pertencem a um Time de Desenvolvimento em um ambiente com múltiplos times trabalhando a partir de um mesmo Product Backlog.

O agrupamento de User Stories em temas relacionados a metas de negócios pode facilitar o trabalho de ordenação do Product Backlog em planejamentos mais longos, mantendo-se o foco em quais metas são mais importantes de serem realizadas primeiro em vez de manter-se o foco em itens individuais.

Cada tema geralmente possui um curto título que o identifica.

CAPÍTULO 26

Story Points: estimando o trabalho

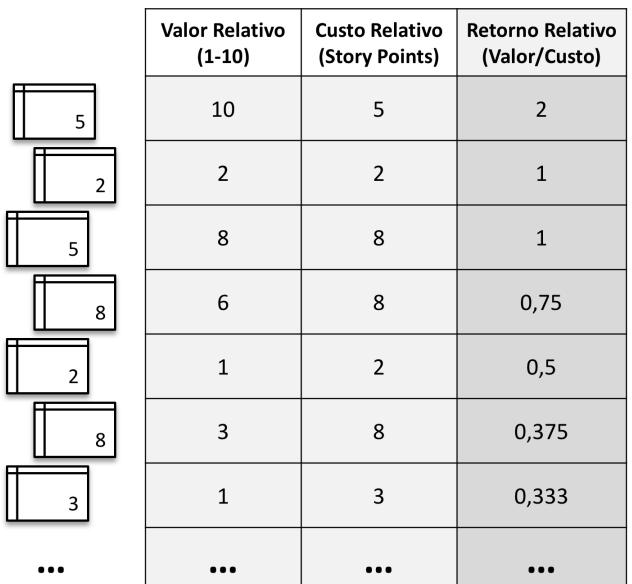
26.1 PARA QUE SERVE ESTIMAR?

Entendemos por estimar o ato de julgar e formar uma opinião sobre o tempo ou esforço que será necessário realizar um determinado trabalho. A estimativa é uma ferramenta que pode ser utilizada para se ajudar a melhor planejar. Assim, estimar-se cada um dos itens do Product Backlog até um determinado horizonte facilita ao Time de Desenvolvimento e Product Owner planejarem o que caberá no próximo Sprint ou na próxima Release.

Estimar os itens do Product Backlog individualmente ainda pode ajudar a se definir a ordem em que os itens do Product Backlog serão desenvolvidos. Uma vez que sua estimativa representa o custo para se desenvolver o item,

pode-se, a partir dessa estimativa (custo) e de alguma medida do valor de negócio do item (valor), calcular o retorno sobre o investimento individual do item dividindo-se, de forma simplificada, seu valor pelo seu custo. Assim, esse retorno individual pode ser usado como um dos critérios na ordenação do Product Backlog.

No exemplo da figura 26.1, a ordenação foi determinada a partir do retorno individual de cada item, utilizando-se o valor de negócio relativo (arbitrado em uma escala relativa de 1 a 10) e o custo relativo de cada item, dado pela estimativa de seu tempo em Story Points, unidade que definiremos mais adiante.



Valor Relativo (1-10)	Custo Relativo (Story Points)	Retorno Relativo (Valor/Custo)
10	5	2
2	2	1
8	8	1
6	8	0,75
1	2	0,5
3	8	0,375
1	3	0,333
...

Fig. 26.1: Utilizando-se o retorno para priorizar os itens

Por fim, uma vez que a estimativa do Product Backlog é feita por todo o Time de Desenvolvimento, o simples exercício de se reunir e discutir para realizar as estimativas leva o Time de Desenvolvimento a uma compreensão compartilhada e mais profunda do que é cada item e de como ele será desen-

volvido. Ou seja, a importância não está somente nos números, mas também no caminho para se chegar a eles.

26.2 PRECISÃO E ACURÁCIA DE ESTIMATIVAS BASEADAS EM JUÍZO

Ao estimarmos o tempo ou esforço que levaremos para realizar um determinado trabalho no nosso dia a dia, não estamos usando cálculos avançados nem aplicando conceitos de probabilidade e estatística. Na realidade, construímos essa estimativa simplesmente utilizando-se de nossos conhecimentos, bom-senso, experiências passadas, contribuições de terceiros e demais informações disponíveis.

De forma simplificada, podemos definir uma estimativa a partir de uma faixa ou intervalo de valores com tamanho maior ou igual a zero, em torno de um **valor de referência**, dados em alguma unidade. Por exemplo, podemos estimar que uma determinada tarefa durará entre 3 e 5 horas ou 4 +- 1 horas.

Essa faixa de valores é chamada de **intervalo de confiança** da estimativa. O intervalo de confiança é tal que existe uma determinada probabilidade de que o valor real estará dentro dele. De forma prática, escolhemos para nossas estimativas um intervalo de confiança maior, quanto maior for a incerteza relacionada à essa estimativa.

Definimos a **precisão** de uma estimativa como a granularidade dessa estimativa, que é inversamente relacionada ao tamanho do intervalo em torno do valor de referência. Assim, quanto menor é o intervalo de confiança usado para a estimativa, mais precisa ela é (veja a figura 26.2).

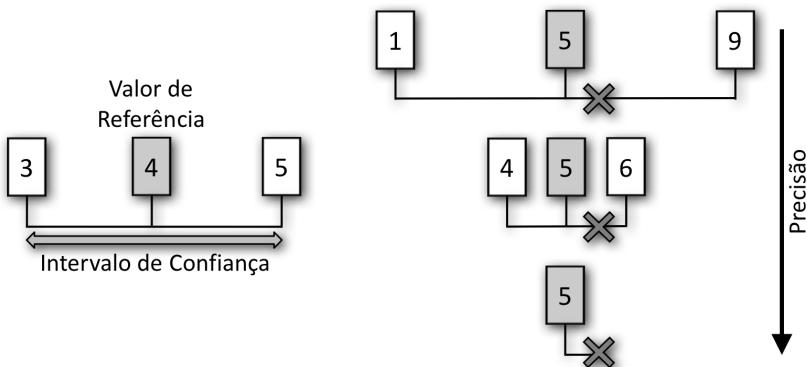


Fig. 26.2: A estimativa e a precisão de uma estimativa

Por exemplo, a estimativa de que uma atividade durará exatamente 5 horas é mais precisa do que uma estimativa de que essa atividade vai durar 5 ± 1 horas (ou seja, entre 4 e 6 horas). Ela é mais precisa ainda do que uma estimativa de que essa mesma atividade durará 5 ± 4 horas (ou seja, entre 1 e 9 horas).

É importante observar que também podemos aumentar a precisão da estimativa utilizando unidades com maior granularidade. Por exemplo, a estimativa de 5 horas, 3 minutos e 10 segundos é mais precisa do que a estimativa de que a atividade vai durar 5 horas.

Repare que a precisão independe totalmente do valor real verificado posteriormente. Ou seja, a precisão não é diferente para qualquer dessas estimativas se a atividade, na realidade, durar 4 ou 20 horas, por exemplo.

O grau de **acurácia** ou de exatidão de uma estimativa é definido por quanto perto do valor real ela está. Assim, quanto menor a diferença entre a estimativa e o valor real verificado posteriormente, maior terá sido o seu grau de acurácia.

O cálculo de um valor numérico para o grau de acurácia pode ser complicado, já que a estimativa é, na realidade, um intervalo. Aqui, porém, usarei uma definição propositalmente simplificada: consideraremos a estimativa “acurada” se ela contiver o valor real verificado posteriormente (ou seja, a es-

timativa se mostrou certa), e “não acurada” caso não o contenha (ou seja, a estimativa se mostrou errada).

Imagine, por exemplo, uma atividade para cuja duração diferentes estimativas foram realizadas. Digamos que a sua execução tanha durado 5,5 horas. Podemos afirmar que a estimativa de que ela duraria exatamente 5 horas não foi acurada, de acordo com as nossas definições, pois não contém o valor real. A estimativa de que ela duraria $5 + - 1$ horas tem menor precisão, mas foi acurada. A estimativa de que ela duraria $5 + - 3$ horas também foi acurada, mas tem uma precisão bem menor.

Em outro exemplo, ao estimarmos tendo como unidade dias, se afirmarmos em nosso planejamento que uma determinada funcionalidade ficará pronta no dia 18 de novembro, essa será uma estimativa de alta precisão independente da data em que a funcionalidade de fato fique pronta. Caso essa funcionalidade fique pronta apenas no dia 25 de novembro, essa terá sido uma estimativa sem acurácia, de acordo com nossa definição de acurácia.

Se tivéssemos afirmado, no entanto, que a funcionalidade estaria pronta no mês de novembro (o que na realidade significa entre os dias 1 e 30 de novembro), nossa estimativa teria sido de mais baixa precisão, porém acurada.

Ou seja, repare pelos dois exemplos anteriores que, ao reduzirmos a precisão de uma estimativa, aumentamos as chances de ela ser acurada. Uma precisão cada vez menor (ou intervalo de confiança cada vez maior) é como um alvo cada vez maior que, assim, aumentam-se as chances de se acertar nele, ou seja, de se ter acurácia. Portanto, precisão e chance de acurácia têm uma relação inversa.

26.3 PRECISÃO E ACURÁCIA SUSPEITAS: LEI DE PARKINSON E SÍNDROME DO ESTUDANTE

A Lei de Parkinson e a Síndrome de Estudante ilustram que obter um alto grau de acurácia (ou seja, de acerto) em uma estimativa de alta precisão para o tempo de realização de alguma tarefa é geralmente um resultado artificial. Ou seja, ao se estimar o tempo necessário para se realizar uma determinada tarefa em X horas e, de fato, a tarefa estiver completa após exatas X horas de trabalho, há possivelmente algo de errado (COHN, 2005).

A **Lei de Parkinson** foi criada por Cyril Northcote Parkinson nos anos 1950 e publicada em um artigo bem-humorado na revista britânica *The Economist* e depois em um livro. Essa “lei” afirma que *a utilização de um recurso se expande até a sua disponibilidade*. Ou, para o nosso caso específico, *o trabalho se expande de modo a preencher o tempo disponível para sua realização*. Ou seja, ao se predeterminar o tempo para a realização de uma tarefa, o tempo usado para a sua realização naturalmente se expandirá até a duração preestabelecida, mesmo que menos tempo seja suficiente.

Por exemplo, digamos que um Gerente de Projetos, em um projeto tradicional, tenha estimado que um desenvolvedor demorará um certo número de horas para realizar uma determinada tarefa. Ao longo do trabalho, o desenvolvedor naturalmente vai preenchendo todo esse tempo disponível para a execução da tarefa. Se há a percepção de que lhe sobrará tempo, ele possivelmente vai utilizar uma maior minúcia e adicionamos atributos não essenciais ao trabalho realizado.

A **Síndrome do Estudante** está relacionada a como o estudante realiza seus trabalhos de casa, ou seja, deixando-os para a última hora. Quando possuímos um prazo para realizar uma determinada tarefa, é um comportamento comum deixarmos sua realização para a última hora, comprometendo-se esse prazo ou, ao menos, deixando de lado o cuidado e a qualidade necessários e só assim conseguirmos terminar a tempo.

Assim, de acordo com a Lei de Parkinson e a Síndrome do Estudante, um Gerente de Projetos tradicional que frequentemente acerta estimativas impostas para o trabalho de seus funcionários não tem razão para comemorar. O fato de existir essa estimativa, na realidade, acaba por produzir artificialmente o “acerto”.

A Lei de Parkinson e a Síndrome de Estudante não possuem caráter científico, mas ilustram bem o comportamento humano.

26.4 ESTIMATIVAS ÁGEIS

Abordagens tradicionais para projetos usam um alto grau de precisão em seu planejamento, o que, a uma primeira vista, pode parecer benéfico. Datas e durações exatas para o cumprimento de tarefas são marcadas em cronogra-

mas, que vemos muitas vezes representados por meio de gráficos de Gantt em sistemas de gerenciamento de projeto.

Sabemos, no entanto, que essas datas e durações estimadas serão quase que certamente erradas, pois a precisão utilizada ignora a realidade de incerteza que permeia os projetos de desenvolvimento de *software*. Assim, podemos afirmar que abordagens tradicionais usam estimativas de alta precisão, mas de baixa acurácia.

Para o desenvolvimento Ágil, ao utilizarmos estimativas para nosso planejamento, acreditamos que ter maior acurácia e, portanto, maior acerto, é mais importante do que ter precisão máxima. Assim, em oposição aos planos com altíssima precisão usados em projetos tradicionais, reconhecemos que, quanto menos detalhes possuirmos acerca de um trabalho a ser realizado, menor deve ser a precisão utilizada para se obter uma acurácia razoável.

Ainda entendemos que, em um ambiente de mudanças como é o desenvolvimento de *software*, quanto mais distantes no tempo estivermos da realização de um trabalho, menos detalhes teremos acerca dele. Logo, menor deverá ser a precisão utilizada para se estimá-lo, de modo a se obter estimativas mais acuradas.

26.5 UNIDADES PARA AS ESTIMATIVAS

26.5.1 Unidades mais comuns

Para estimarmos o tempo para a realização de itens do Product Backlog de um projeto, podemos utilizar como unidades:

- **tempo real** — dias ou horas reais de trabalho, ou seja, uma estimativa de quanto tempo em dias ou horas a realização da atividade vai durar. É a unidade mais tradicional de estimativas;
- **tempo ideal** — dias ou horas ideais de trabalho, ou seja, quanto tempo se levaria para realizar uma atividade caso todo o foco de trabalho estivesse nessa atividade e não existissem quaisquer interrupções, como conversas, leitura de e-mails, idas ao banheiro, reuniões, cafezinho etc.;
- **Story Points**: unidade relativa de tempo ou esforço criada pelo Time de Desenvolvimento. É a unidade mais usada por equipes Ágeis.

Em vez de estimarmos o tempo para a realização dos itens de trabalho, podemos estimar o esforço. Nesse caso, substituímos horas ou dias por pessoas-horas ou pessoas-dias, ou seja, o número de dias ou horas, em geral ideais, necessários para completar o item que devem ser distribuídos entre as pessoas executando o trabalho.

Assim, 10 pessoas-horas podem significar tanto uma pessoa realizando o item em dez horas, como duas pessoas realizando o item em cinco horas ou quatro pessoas o realizando em duas horas e meia.

Para o uso de Story Points, no entanto, basta simplesmente definir o Story Point como esforço em vez de tempo. O uso de Story Points como tempo e o uso de Story Points como esforço trazem o mesmo efeito por duas razões:

- 1) uma estimativa do trabalho do Time de Desenvolvimento trata do trabalho do time como um todo, e não de um membro individual;
- 2) a estimativa com Story Points somente faz sentido se o time mantém sua formação constante e, assim, o número de pessoas para realizar o trabalho é sempre o mesmo.

Para efeitos de simplicidade, tratarei de Story Points neste capítulo como tempo, e não esforço.

26.5.2 Estimativas absolutas x relativas

Para se estimar o tempo necessário para a realização de uma atividade, pode parecer natural ou intuitivo que utilizar dias ou horas seja a melhor opção. Entretanto, a experiência nos mostra que existem alternativas mais interessantes.

De forma geral, apesar de possuírem baixa acurácia, as estimativas em dias ou horas reais para o desenvolvimento de *software* carregam em si um senso de obrigação ou promessa irreal. Em vez de serem entendidas como meras previsões para facilitar o planejamento, elas são tratadas como compromissos por quem espera os resultados.

Por exemplo, quando alguém diz que demorará cinco dias para executar uma determinada atividade, a expectativa criada é de que, após cinco dias, a

atividade esteja pronta e essa pessoa será questionada caso isso não ocorra. Utilizar horas ideais ou dias ideais acentua ainda mais o problema da cobrança.

Ao se dizer que uma atividade levará n dias ideais, é difícil justificar o porquê dessa atividade não estar pronta após n dias, pois a sensação que fica é de que o tempo não foi bem utilizado. Além disso, estimar em horas ou dias que não são, de fato, horas ou dias não traz nenhum benefício real para o planejamento.

Para melhorar essa questão, as estimativas em horas ou dias reais podem ser explicitamente expressas com precisões menores, usando-se intervalos de confiança. Pode-se, por exemplo, estimar *entre quatro e seis dias* em vez de *cinco dias*, e as chances de acertar (de ter acurácia, portanto) aumentarão. No entanto, lidar no planejamento com esse tipo de estimativas com intervalos é muito trabalhoso.

Além disso, conforme expliquei anteriormente, possuir um valor em dias ou horas como referência para a realização de uma atividade pode levar a disfunções como a Lei de Parkinson e a Síndrome de Estudante.

A abordagem mais utilizada por times Ágeis é o uso de estimativas relativas. Verifica-se na prática que é muito mais fácil estimar realizando comparações do que fazê-lo em termos absolutos. Assim, em vez de estimar que uma determinada atividade exigirá seis dias para ser realizada, estabelecemos que ela vai demorar, por exemplo, cerca de duas vezes mais tempo que uma outra determinada atividade, e metade do tempo de uma terceira — estas últimas duas sendo utilizadas como referência.

Contanto que as referências sejam propagadas de estimativa em estimativa, as estimativas se manterão consistentes. A abordagem de estimativas relativas evita ao mesmo tempo os efeitos da Lei de Parkinson, da Síndrome do Estudante e do tratamento incorreto da estimativa como compromisso ou obrigação simplesmente ao se deixar de expressar as estimativas em dias ou horas e, assim, desacoplar a execução de um item de trabalho de uma data exata de entrega (DE TOLEDO, 2009).

26.5.3 Story Points

Story Point é uma unidade relativa usada pelo Time de Desenvolvimento para estimar o tempo ou esforço necessário para se desenvolver um item de trabalho. No caso de Scrum, esse desenvolvimento se refere a transformar um item do Product Backlog em pronto, de acordo com a Definição de Pronto.

Em vez de utilizar horas ou dias, o Time de Desenvolvimento cria sua própria escala a partir de um ou mais pontos de referência selecionados. Assim, passa a estimar os itens de trabalho dentro dessa escala, comparando-os com esses pontos de referência e com os últimos itens estimados.

Escalas simples de um a dez ou potências de dois podem funcionar, mas as mais utilizadas por times Ágeis são a sequência de Fibonacci e a chamada “tamanhos de camisa” (*T-Shirt Sizing*), ambas explicadas em seguida.

Na sequência de Fibonacci, cada número é igual a soma dos dois números anteriores, começando com zero e um. Assim, temos: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 e assim por diante. Para efeitos práticos, usamos a escala da seguinte forma:

1 — 2 — 3 — 5 — 8 — 13 — 21 — 34 — 55 — 89...

Pode-se utilizar uma versão adaptada, criada por Mike Cohn (2005) para maior simplicidade e menor sensação de alta precisão nos números maiores:

1 — 2 — 3 — 5 — 8 — 13 — 20 — 40 — 100

Nessa versão, 40 e 100 simplesmente significam que um item de trabalho é grande demais. Para os tamanhos de camisa (*T-Shirt Sizing*), a escala é a seguinte:

PP — P — M — G — GG — XG

P significa mais ou menos o dobro do tempo que **PP**, **M** significa mais ou menos o dobro do tempo que **P** e mais ou menos quatro vezes o tempo

que **PP**, e assim por diante. Na maioria dos casos, uma escala menor e mais simples é suficiente:

P — M — G

Outras escalas que passem a ideia de tamanho relativo podem funcionar bem. Eu já vi times utilizando raças de cachorro (do chihuahua ao dogue alemão), vegetais (da azeitona à melancia, por exemplo) e muitas outras alternativas criativas.

Uma vez selecionada a escala, escolhem-se um ou mais itens de referência para se criarem os pontos da escala. Depois de se escolher esse item ou itens de referência, os itens seguintes a serem estimados são comparados com ele ou eles.

Uma técnica simples para se criar os pontos da escala é a de se escolher como referência de unidade um item do Product Backlog que o time concorde que é menor naquele momento. Esse item de referência provavelmente está próximo ao topo do Product Backlog, onde estão os itens menores e mais detalhados. Caso o Time de Desenvolvimento julgue haver vários itens igualmente pequenos, ele escolherá entre eles como referência aquele mais conhecido, mais detalhado e com menos risco envolvido em seu desenvolvimento.

Dessa forma, o item de referência passa a ter a estimativa de 1 Story Point ou “P” ou “chihuahua”, por exemplo, dependendo da escala escolhida. Assim, os próximos itens a serem estimados serão comparados com essa referência. Por exemplo, um item que, na estimativa do time, demande cerca de duas vezes o tempo para se realizar o item de referência, receberá 2 Story Points.

Em vez da unidade, muitos times preferem atribuir a esse item de referência o segundo ponto da escala (ou seja, 2 Story Points, por exemplo) para que no futuro haja espaço para itens menores.

Alternativamente, pode-se conseguir melhores resultados ao se escolherem dois itens de referência de tamanho médio em vez de apenas um e pequeno. Atribuem-se a esses itens 5 ou 8 Story Points (ou “M”, por exemplo). De outra forma, pode-se escolher um item pequeno e um item grande, atribuindo-se a eles 2 e 13 Story Points (ou “P” e “G”, respectivamente). Os

próximos itens a serem estimados são comparados com esses dois itens de referência, em uma triangulação.

Uma vez criada a referência ou referências, parte-se do alto do Product Backlog estimando-se o primeiro item, comparando-o com o item ou itens de referência. Digamos que o Time de Desenvolvimento escolheu um item como referência e atribuiu 2 Story Points a ele. Assim, se o Time de Desenvolvimento julga, por exemplo, que primeiro item do Product Backlog levará mais que o dobro do tempo que o item de referência, esse item é então estimado em 5 Story Points ou “M” etc. Para o item seguinte, faz-se uma triangulação com os dois itens já pontuados, ou seja, compara-se com eles esse item seguinte.

Assim, se o Time de Desenvolvimento julga que o desenvolvimento desse item seguinte vai demorar bem mais o do item de referência e um pouco mais que o do outro item já estimado, pode-se estimar esse item seguinte em 8 ou “G” etc., dependendo da escala escolhida.

A partir daí, o Time de Desenvolvimento segue estimando os itens a partir do alto do Product Backlog, Sprint após Sprint, sempre os comparando com os últimos itens estimados. É importante ressaltar que o item de referência é escolhido apenas uma vez, geralmente antes do princípio do projeto, e não Sprint a Sprint.

O uso da sequência de Fibonacci como escala para as estimativas traz uma vantagem interessante. Cada número da escala carrega em si um intervalo de confiança e, assim, o tratamento da precisão e acurácia da estimativa já está embutido nessa escala.

Por exemplo, ao se estimar um item do Product Backlog em 3 Story Points, implicitamente se está afirmado que esse item deverá ser algo entre 2 e 5 Story Points — um intervalo de confiança de tamanho $(5 - 2) = 3$. No entanto, ao se estimar outro item como 8 Story Points, implicitamente se está afirmado que esse item deverá ser algo entre 5 e 13 Story Points — um intervalo de confiança de tamanho $(13 - 5) = 8$ (veja a figura 26.3).



Fig. 26.3: Fibonacci, precisão e acurácia

Ao usar a sequência de Fibonacci, podemos verificar que quanto maior for a estimativa de um item, maior será seu intervalo de confiança e, assim, maior será a incerteza associada a ela e menor será sua precisão. De fato, sabemos intuitivamente que, para termos mais acurácia, a precisão que podemos utilizar para estimar itens menores é maior do que para itens maiores.

Como vimos anteriormente, quanto menor for a precisão de uma estimativa, maior é a possibilidade de essa estimativa ser acurada. Portanto, a sequência de Fibonacci automaticamente reduz a precisão de uma estimativa quanto maior ela for, aumentando assim sua possibilidade de acerto, ou seja, de acurácia.

Discussão: Story Points representam complexidade, risco, esforço ou tamanho?

Existe muita confusão acerca do significado dos Story Points. Eles representam a complexidade de uma funcionalidade? Talvez com um componente do risco envolvido em realizá-la, derivado da incerteza? Ou será que representam o esforço necessário para desenvolver a funcionalidade? Ou talvez o tamanho? Mas então o que exatamente significa tamanho de uma funcionalidade?

Story Points foram criados pelos mesmos inventores do Extreme Programming (JEFFRIES et al., 2000), uma metodologia Ágil que muitas vezes anda de mãos dadas com Scrum. Seu propósito era o de burlar estimativas em unidades tradicionais de tempo, como horas ou dias, pois a gerência equivocadamente entendia essas estimativas como promessas, e não como estimativas.

Ou seja, era comum ouvir-se *se você disse que ficaria pronto em três dias, por que não está pronto três dias depois?*. Assim, Story Points foram inicialmente criados para que se pudesse continuar realizando estimativas necessá-

rias para o planejamento, mas se ofuscando o tempo para evitar o compromisso com uma alta precisão e, dessa forma, evitar a pressão da gerência sobre essas estimativas.

Story Points representam, nada mais, nada menos, que tempo.

Utilizamos Story Points justamente para estimarmos quanto trabalho é possível ser feito em um determinado período de tempo (um Sprint ou uma Release, por exemplo) ou quanto tempo o time vai demorar para entregar um determinado objetivo realizado.

É claro que, para isso, levamos em consideração parâmetros como complexidade, incerteza e tamanho, e sabemos que custo e esforço são derivados do tempo. Mas Story Points representam, simplesmente, tempo. Esse é, porém, um tempo que não é medido em horas ou dias, mas sim em uma escala relativa criada pelo próprio Time de Desenvolvimento. Assim, n Story Points nada mais significam que n dias ou horas vezes algum fator.

Como afirmei anteriormente, podemos alternativamente dizer que Story Points representam esforço, o que traz o mesmo efeito para Times de Desenvolvimento com Scrum já que, para que essas estimativas tenham sentido, o time deve manter sua formação constante.

É importante ressaltar que, ao utilizar Story Points, buscamos justamente mascarar e evitar expressar a estimativa em horas. Logo, não faz sentido realizar sistematicamente qualquer tradução dos pontos em dias ou horas (como, por exemplo, *para nosso time, 3 Story Points significam dois dias de trabalho*) ou descobrir em qualquer momento o fator de conversão de um para outro.

Hoje, no entanto, os próprios criadores dos Story Points não mais as recomendam. Devido a seu uso exageradamente incorreto, preferem alternativamente manter os itens do Product Backlog bem pequenos e, se necessário, contar o número médio de itens do Product Backlog por Sprint para se realizar o planejamento, como veremos adiante.

26.6 COMO FAZER A ESTIMATIVA?

Qualquer que seja o método usado para realizar as estimativas, é importante que os princípios básicos a seguir sejam respeitados:

- as estimativas dos itens do Product Backlog somente podem ser realizadas pelo Time de Desenvolvimento. Elas não devem ser realizadas pelo Product Owner, pelo ScrumMaster ou por quaisquer outras partes interessadas no projeto. Acredita-se que as melhores estimativas são aquelas feitas pelas pessoas que realizam o trabalho a ser estimado. Essa premissa, apesar de representar o senso comum, é ignorada em um grande número de projetos, nos quais frequentemente Gerentes de Projeto criam as estimativas para suas equipes;
- as estimativas são definidas por um consenso de todo o Time de Desenvolvimento. Como cada item do Product Backlog deve ser desenvolvido por todo o Time de Desenvolvimento, a estimativa de um item do Product Backlog não pode pertencer a apenas um ou a alguns de seus membros. Ela deve ser fruto da colaboração entre todos;
- o processo de definição das estimativas não pode ser custoso nem demorado. Se esse processo não for rápido e objetivo, o tempo despendido nele será relevante, de forma que a própria atividade de estimar necessitará de estimativas, o que não faz o menor sentido.

Os membros do Time de Desenvolvimento se reúnem para a realização de estimativas de itens do Product Backlog, em geral em sessões de Refinamento do Product Backlog ou, no mais tardar, na reunião de Sprint Planning. O Product Owner, presente durante a atividade, pode esclarecer dúvidas que inevitavelmente surgirão quanto aos itens, enquanto que o ScrumMaster atua como facilitador. Nem Product Owner, nem ScrumMaster participarão da escolha dos valores para as estimativas.

Não é recomendável gastar tempo demasiadamente longo com a estimativa de cada item. O processo de estimar é idealmente rápido e objetivo, de forma que o tempo total gasto não seja relevante. Além disso, a partir de algum momento, mais conversas sobre um item já não trarão melhores estimativas. É considerado desperdício, por exemplo, discutir profundamente e em detalhes como o item será desenvolvido.

O uso da sequência de Fibonacci com escala de estimativas facilita significativamente o processo de se chegar a um consenso. Como a escala não é linear, quanto maior é o item, menor é o número de escolhas possíveis para sua

estimativa. Assim, não se desperdiça tempo com estimativas de mais baixa precisão.

Por exemplo, faz mais sentido gastar mais tempo se discutindo se um item do Product Backlog deve ter a estimativa de 2 ou 3 Story Points (mais alta precisão) do que se outro item deve ter 15 ou 16 Story Points (mais baixa precisão). Esta última discussão não é possível ao se utilizar a sequência de Fibonacci — ou o item será estimado em 13 ou em 21.

26.6.1 Planning Poker

O Planning Poker é a técnica mais conhecida e utilizada por times Ágeis para se chegar um consenso quanto às estimativas de itens do Product Backlog. Foi criado por James Greening (2002), um dos signatários do Manifesto Ágil, e satisfaz todos os princípios descritos anteriormente se usado corretamente. Apesar de sua popularidade, o Planning Poker não é parte integrante do Scrum e seu uso é opcional.

Para o Planning Poker, cada membro do Time de Desenvolvimento deve possuir um conjunto de cartas com as alternativas possíveis da escala utilizada para as estimativas. Assim, se a sequência de Fibonacci adaptada for escolhida, cada um possuirá as cartas 1, 2, 3, 5, 8, 13, 20, 40, 100. Alternativamente, cada membro pode escrever as alternativas em notas adesivas ou a sua escolha diretamente em um papel em branco. Existem também diversos aplicativos disponíveis para *tablets* e celulares com essa finalidade.

Alguém — o Product Owner, o ScrumMaster ou um membro do Time de Desenvolvimento — lê o próximo item do Product Backlog a ser estimado. Os membros do Time de Desenvolvimento refletem brevemente sobre esse item e esclarecem dúvidas com o Product Owner.

Cada membro do Time de Desenvolvimento então escolhe a carta com sua estimativa para o trabalho necessário para o se desenvolver o item, comparando-o mentalmente com estimativas realizadas anteriormente. Para a estimativa, cada um pensa no trabalho do Time de Desenvolvimento como um todo, e não apenas em seu próprio.

É importante que o valor escolhido não seja imediatamente discutido nem mostrado aos colegas para não influenciar seu julgamento inicial. Ao

verem prematuramente as estimativas de seus colegas, os membros do Time de Desenvolvimento que ainda não revelaram sua carta podem modificar sua escolha de forma a se aproximarem das estimativas já mostradas, sem sequer pensarem ou discutirem o porquê. É uma tendência natural de conformidade que o ser humano possui, chamada de ancoragem.

Em seguida, cada um coloca sua escolha sobre a mesa, de forma que todos possam vê-la (veja a figura 26.4). Difícilmente um consenso será atingido nessa primeira rodada de votação. As diferenças, na realidade, vão gerar conversas que levarão a uma melhor compreensão do item, o que também contribui para a identificação de problemas mais cedo. Mais rodadas de votação são então realizadas, sempre facilitadas pelo ScrumMaster, até que um consenso seja atingido. A necessidade de um consenso faz com que todo o Time de Desenvolvimento seja responsável pela estimativa, e não apenas um indivíduo.

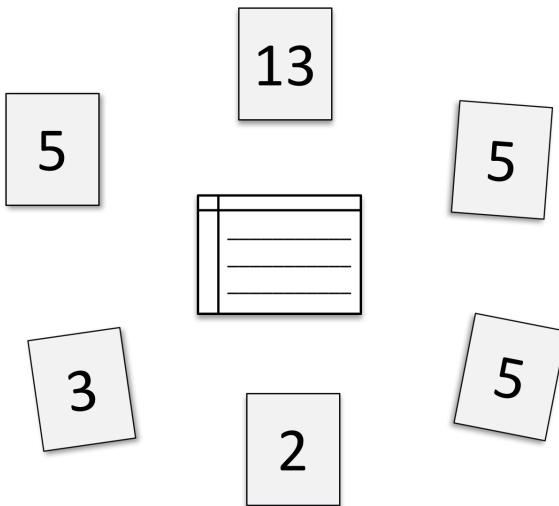


Fig. 26.4: Planning Poker

Após cada rodada de votação, as pessoas que deram a estimativa mais alta e a mais baixa começam a conversa, justificando o porquê de suas escolhas.

O fato de terem que justificar suas estimativas leva os membros do Time de Desenvolvimento a realmente se engajarem em buscar estimativas que lhes façam sentido.

Um membro do Time de Desenvolvimento que estimou o item em 2 Story Points pode dizer, por exemplo: *esse item é fácil, já fiz algo muito parecido antes*. O outro membro que escolheu 8 Story Points pode, então, dizer *sim, mas você está se esquecendo que testar isso é bem complicado e que é necessário construir um ambiente para esse tipo de teste. Mas como você já fez algo parecido, talvez demore menos do que pensei*. Em uma segunda rodada, eles terão estimativas mais próximas (3 e 5 Story Points, provavelmente), assim como o resto do Time de Desenvolvimento, e o consenso poderá até mesmo ser atingido sem uma nova votação.

Pode ser uma boa ideia determinar o tempo máximo para cada discussão, ou seja, estabelecer-se um *timebox*. Além disso, espera-se que o consenso seja atingido com não mais que duas ou três rodadas de votação. Caso o time necessite de mais rodadas, esse pode ser um indicador da necessidade de uma atuação mais efetiva do facilitador. No caso de impasses, o Time de Desenvolvimento pode ter seus próprios acordos de como proceder. Se o consenso não é possível, por exemplo, o time pode determinar em seu acordo a escolha da estimativa mais alta ou, talvez, de uma intermediária.

PLANNING POKER

A atividade de Planning Poker se resume aos seguintes passos:

- 1) alguém — ScrumMaster, Product Owner ou membro do Time de Desenvolvimento — lê o próximo item a ser estimado. O ScrumMaster facilita a sessão;
- 2) os membros do Time de Desenvolvimento rapidamente refletem sobre esse item e esclarecem dúvidas com o Product Owner;
- 3) cada membro do Time de Desenvolvimento então escolhe uma carta com a sua estimativa para o trabalho necessário para o Time de Desenvolvimento desenvolver o item, sem mostrar a carta imediatamente;
- 4) todos mostram as suas cartas, colocando-as sobre a mesa ao mesmo tempo. Dessa forma, evita-se a ancoragem, ou seja, que se aceite sem reflexão a influência dos outros;
- 5) os membros do Time de Desenvolvimento que deram a estimativa mais alta e a mais baixa conversam diante dos outros e defendem suas estimativas, justificando o porquê dos valores escolhidos;
- 6) uma ou mais rodadas de escolha de estimativas para o item são realizadas, até os membros Time de Desenvolvimento chegarem a um consenso.

26.7 VELOCIDADE DO TIME DE DESENVOLVIMENTO

Conhecer a Velocidade do Time de Desenvolvimento pode ajudar ao Time de Desenvolvimento e ao Product Owner na tarefa de planejar.

O Time de Desenvolvimento pode utilizar Story Points ou alguma unidade para estimar o tempo necessário para realizar um determinado tra-

lho. Para efeitos didáticos, chamaremos essa unidade simplesmente de “Pontos” nesta seção. Ela será usada para a estimativa de cada item do Product Backlog.

Digamos que o Time de Desenvolvimento tenha produzido, nas últimas três Sprints, itens cujas estimativas somadas correspondem a 30, 32 e 29 pontos, respectivamente. Podemos tirar a média desses valores e dizer que a Velocidade do Time de Desenvolvimento é de aproximadamente 30 pontos por Sprint (veja a figura 26.5).

A Velocidade do Time de Desenvolvimento é, portanto, a taxa média de pontos dos itens produzidos por Sprint. Assim, podemos esperar que o Time de Desenvolvimento entregue algo próximo de itens correspondentes a 30 pontos no próximo ou nos próximos Sprints ou, ao menos, sabemos que essa é a nossa melhor previsão possível.

Espera-se uma Velocidade relativamente constante ao longo do projeto, dado que a composição do Time de Desenvolvimento seja mantida e o mesmo ocorra com o tamanho dos Sprints. O primeiro Sprint se inicia, é claro, sem nenhuma referência de Velocidade. A partir do segundo ou terceiro Sprints, no entanto, já há sentido em se utilizarem para seu cálculo os valores das entregas anteriores, ainda que a média não tenha se estabilizado.

No entanto, é considerada uma boa prática usarem-se apenas os valores somados das estimativas dos itens entregues nos três últimos Sprints para se calcular essa média em vez de todos os valores históricos. Assim, se produz um valor mais realista e atualizado para a Velocidade do Time de Desenvolvimento.

Essa Velocidade pode servir como referência no planejamento do próximo Sprint. Por exemplo, durante a reunião de Sprint Planning seguinte, o Time de Desenvolvimento pode não aceitar para o Sprint Backlog um número de itens cujas estimativas somem muito mais do que 30 pontos e, se for necessário, negociará com o Product Owner para reduzir o número de itens e se aproximar desse valor.

O Product Owner, por sua vez, poderá contestar a escolha do Time de Desenvolvimento, para o Sprint Backlog, de um número de itens cujas estimativas somem muito menos do que esses 30 pontos. Caso necessário, ele negociará com o Time de Desenvolvimento um aumento no número de itens

para se aproximar desse valor.

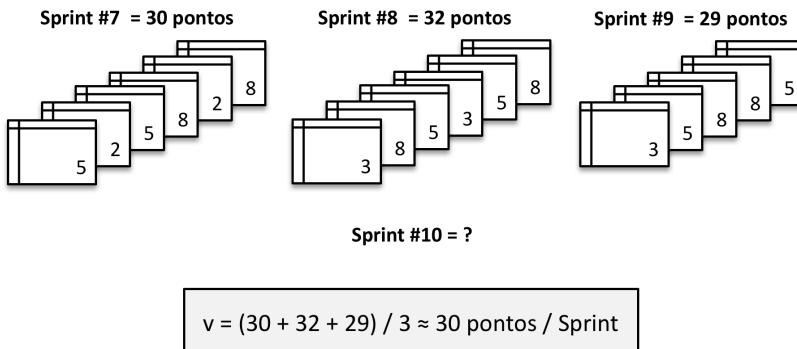


Fig. 26.5: Medir a Velocidade do Time de Desenvolvimento pode aumentar a previsibilidade

A Velocidade também pode ser útil para o planejamento da próxima Release, mesmo que seja necessária uma extração de baixa precisão. Tendo-se estimativas para os itens do Product Backlog e conhecendo-se a Velocidade do Time de Desenvolvimento, pode-se planejar qual meta de negócios (Meta da Release, no caso) poderá ser realizada dado um prazo ou, dada uma meta de negócios, quando ela poderá ser realizada. Essa técnica é geralmente usada na reunião de Release Planning (veja [22.1 O que é a Release Planning?](#)).

26.8 NO ESTIMATES: ESTIMATIVAS SÃO DESPERDÍCIO

Para facilitar o planejamento, existe uma alternativa importante à abordagem de se estimar os itens do Product Backlog individualmente. O movimento *no estimates* (ou “sem estimativas”) ganhou muito espaço nos últimos anos.

O princípio básico de se trabalhar sem estimativas reside na realidade de que, apesar de que as estimativas nos trazem uma sensação de segurança, nossas estimativas são muito limitadas no desenvolvimento de *software* e, basicamente, têm uma distância muito grande da realidade. Há uma grande desconexão entre o que cremos que vai acontecer e o que realmente acontece. Assim, essa segurança é falsa.

Além disso, como vimos anteriormente, o comportamento das pessoas se modifica de forma disfuncional devido à simples existência de uma estimativa. Dessa forma, estimativas constituem desperdício. A partir desses conceitos, existem diferentes abordagens para o trabalho sem estimativas.

A abordagem mais comum de *no estimates* estabelece que, para viabilizar o planejamento sem estimativas, deve-se utilizar a boa prática de se manter os itens do alto do Product Backlog consistentemente pequenos ou, ao menos, com tamanhos razoavelmente próximos. Recomenda-se que, para cada User Story, não sejam necessários mais do que dois ou três dias de desenvolvimento. Assim, pode-se determinar a Velocidade do Time de Desenvolvimento apenas fazendo-se a média da quantidade de itens entregues nos últimos Sprints, e usar esse valor para planejar o próximo Sprint (veja a figura 26.6).

Dessa forma, um Time de Desenvolvimento que tenha entregue 6, 7 e 5 itens nos últimos três Sprints tem a Velocidade de 6 itens por Sprint, e escolherá algo em torno dessa quantidade de itens na reunião de Sprint Planning. Essa forma simplificada de se trabalhar é suficiente para muitos contextos, nos quais se estimar além desses limites pode ser considerado desperdício.

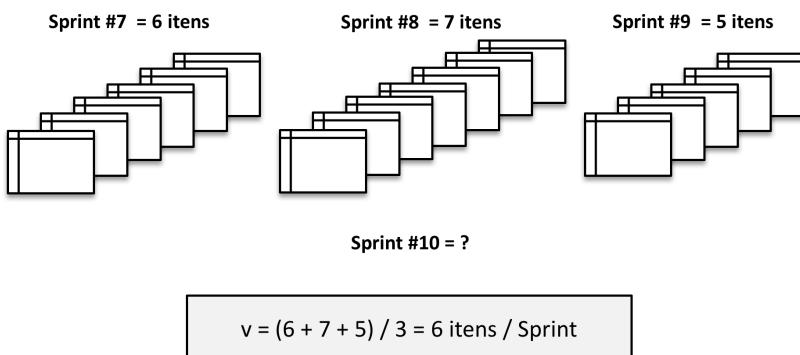


Fig. 26.6: Medindo a Velocidade do Time de Desenvolvimento sem estimativas

Ao se trabalhar para realizar as metas, o foco desvia-se do plano estabelecido para o valor a ser alcançado. Desse modo, as metas podem ser atingidas

com maior ou menor grau de sucesso a partir de mais ou menos itens produzidos. Assim, diminui-se a importância da pontuação cumprida.

Considera-se também que o planejamento para um horizonte muito maior que um Sprint possui uma precisão tão pequena que não vale a pena ser realizado. Logo, evitam-se planejamentos detalhados de Releases.

Como mencionei anteriormente, os próprios criadores dos Story Points não mais as utilizam e, de forma geral, são adeptos das ideias de *no estimates*.

Há times que sequer calculam sua Velocidade. Eles consideram que trabalhar com qualquer conceito relacionado a estimativas é desperdício. Apenas com seu conhecimento e experiência, são capazes de estabelecer que objetivo acreditam que entregarão naquele Sprint ou naquela Release, trabalhando sempre do item mais importante para o menos importante.

CAPÍTULO 27

Burndown e Burnup: acompanhando o trabalho

Os Gráficos de Acompanhamento do Trabalho são ferramentas para a visualização do progresso do cumprimento de uma quantidade mensurável estimada de trabalho em um determinado tempo. Eles servem para criar visibilidade de uma forma simples, ajudando a rapidamente identificar problemas e, assim, reduzir os riscos do projeto.

Os Gráficos de Burndown de Trabalho são os mais utilizados por Times de Scrum com esse propósito. A ideia de *burndown* (queima) é que o trabalho previsto é “queimado” em direção a um determinado momento no tempo, formando um gráfico de inclinação negativa, ou seja, para baixo.

Os Gráficos de Burndown de Trabalho envolvem, portanto, alguma unidade de trabalho restante no eixo Y e alguma unidade de tempo no eixo X

(veja a figura 27.1). Unidades usadas para trabalho restante incluem número de tarefas restantes, quantidade de horas das tarefas restantes, número de itens restantes e quantidade de Story Points dos itens restantes. Unidades comuns para tempo incluem dias, semanas e Sprints, entre outros.

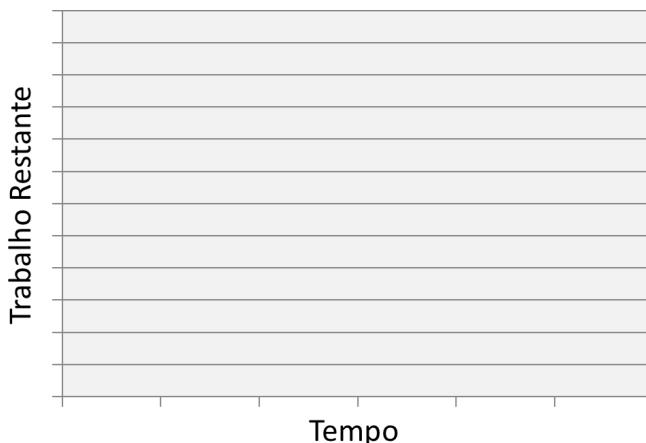


Fig. 27.1: Gráfico de Burndown de Trabalho: Trabalho Restante x Tempo

Os Gráficos de Release Burndown e de Sprint Burndown são os dois tipos de Gráficos de Burndown de Trabalho mais utilizados por Times de Scrum. Os Gráficos de Burnup de Trabalho, diferentemente dos de Burndown, mostram a quantidade de trabalho já realizado no tempo e o trabalho total a se realizar.

O Gráfico de Burnup de trabalho mais usado por Times de Scrum é o Gráfico de Release Burnup. Os Gráficos de Acompanhamento do Trabalho não são parte do *framework* Scrum, sendo seu uso opcional.

27.1 GRÁFICO DE RELEASE BURNDOWN

27.1.1 O que é o Gráfico de Release Burndown?

O Release Burndown é um gráfico mantido pelo Product Owner e utilizado tanto pelo Product Owner quanto pelo Time de Desenvolvimento para mo-

nitorar o progresso no desenvolvimento em direção a uma entrega (Release). O Gráfico de Release Burndown não é parte do *framework* Scrum, mas pode ser útil quando se usa um Plano de Release, geralmente estabelecido em uma reunião de Release Planning.

Ao avaliar o gráfico, o Product Owner pode decidir mudanças de rumo, como mudanças no escopo ou na Meta da Release, adiamento da Release, e até mesmo seu cancelamento.

O Gráfico de release Burndown mostra a quantidade de trabalho restante previsto para uma Release (eixo Y), correspondente aos itens do Product Bac-klog selecionados para essa Release (a soma de suas estimativas, por exemplo), ao final de cada Sprint dessa Release (eixo X). O Gráfico de Release Burndown é uma maneira rápida e prática de se visualizar o andamento da Release.

A figura 27.2 mostra um exemplo de Release Burndown antes do início do último Sprint de uma Release. Nesse exemplo, a quantidade inicial de trabalho é de 235 Story Points, a serem queimados em seis Sprints.

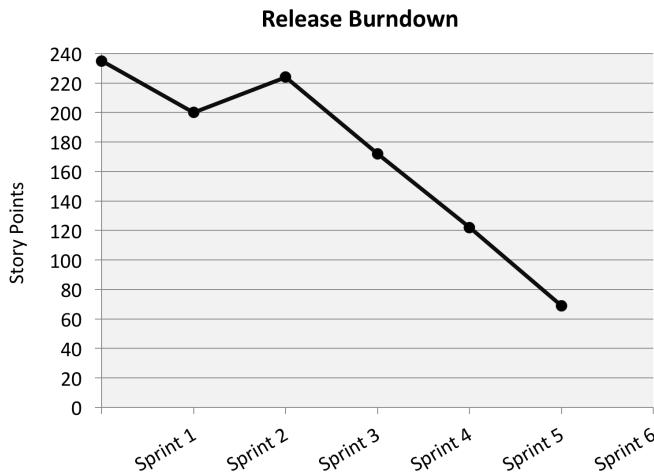


Fig. 27.2: Um exemplo de Release Burndown ao final do penúltimo Sprint da Release

Nos exemplos desta seção, usarei Story Points como unidade de “trabalho restante previsto”. Os Story Points são atribuídos pelo Time de Desenvolvimento aos itens do Product Backlog selecionados para a Release. Entretanto, existem diversas outras unidades possíveis, como tempo real e tempo ideal (veja [26.5 Unidades para as estimativas](#)).

Para “tempo”, utilizamos como marcos o final de cada Sprint previsto para a Release. Assim, cada ponto do gráfico mostrará a soma dos Story Points restantes estimados dos itens do Product Backlog selecionados para a Release, ao final de cada Sprint dela.

No exemplo da figura [27.2](#), podemos verificar pelo gráfico que, nos Sprints 3, 4 e 5, o Time de Desenvolvimento “queimou” 52, 50 e 53 Story Points respectivamente, atingindo quase uma constância. Definimos que, antes do início do sexto e último Sprint da Release atual, a Velocidade do Time de Desenvolvimento (ou seja, a taxa esperada de “queima” de Story Points por Sprint) é a média inteira dos pontos entregues nesses três últimos Sprints — ou seja, 52 pontos por Sprint.

O Gráfico de Release Burndown é criado na reunião de Release Planning, ou logo antes do primeiro Sprint, e é atualizado ao final de cada Sprint com o valor da quantidade restante de trabalho previsto naquele momento.

27.1.2 Como é o Gráfico de Release Burndown?

O exemplo da figura [27.3](#) mostra um Gráfico de Release Burndown em Story Points que acaba de ser atualizado logo após o final do terceiro Sprint da Release, de um total de seis previstos.

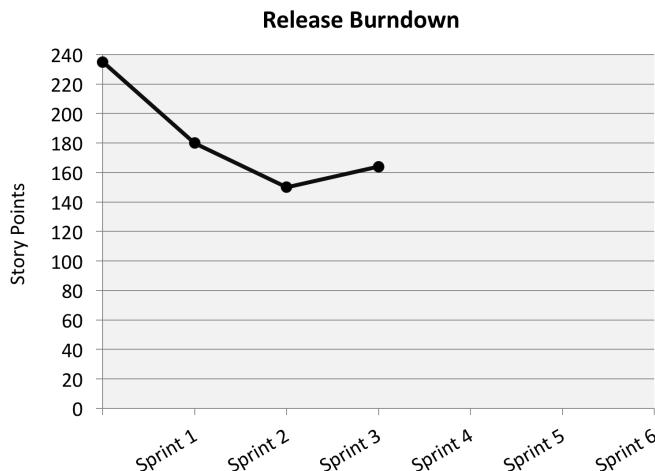


Fig. 27.3: Release Burndown com inclinação para cima no final do terceiro Sprint

Observe que a inclinação súbita para cima, de 150 para 164 Story Points, serve como sinal de alerta, indicando que mais Story Points entraram do que foram “queimados” na Release durante Sprint 3. Essa subida pode acontecer quando há uma combinação de um ou mais dos seguintes fatores: mudanças no escopo da Release com a inclusão de novas funcionalidades, mudanças nas estimativas dos itens restantes do Product Backlog e parte significativa do Sprint não foi entregue.

Uma forma prática de se verificar o andamento da Release é traçando-se uma linhareta iniciando três Sprints antes do Sprint que acaba de se encerrar, que no exemplo é o ponto inicial do gráfico em ([Início], 235), e o ponto atual, que é ([final do Sprint 3], 164), e prolongando essa linha até encontrar o eixo X.

Esse ponto no eixo X é o momento em que se projeta que o Time de Desenvolvimento terá “queimado” todos os Story Points planejados, mantendo-se a Velocidade atual. Pode-se ver pela projeção da figura 27.4 que, no momento atual, as maiores chances são que o trabalho estará longe de terminado ao final do Sprint 6. Esse cenário de possível atraso provavelmente le-

vará Time de Desenvolvimento e Product Owner e, possivelmente, Product Owner e clientes, a negociarem sobre o futuro da Release.

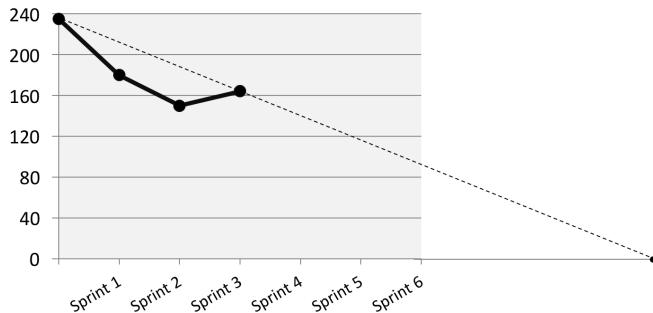


Fig. 27.4: Verificando o andamento da Release

É importante destacar que a projeção mostra apenas uma tendência e que não se espera que o Time de Desenvolvimento complete todos os itens definidos inicialmente para a Release. O Time de Desenvolvimento trabalha a partir dos mais importantes em direção aos menos importantes previstos para a Release.

Esses itens vão evoluindo para itens menores e mais detalhados à medida que o trabalho segue e eles ganham importância, e partes de menor importância são movidas para baixo no Product Backlog. Novos itens também surgirão. Dessa forma, espera-se que os que sobrarem ao final da Release sejam os menos importantes, assim aumentando as chances de que a Meta da Release tenha sido realizada.

27.2 GRÁFICO DE RELEASE BURNUP

27.2.1 O que é o Gráfico de Release Burnup?

O Release Burnup, assim como o Release Burndown, é um gráfico mantido pelo Product Owner e usado tanto pelo Product Owner quanto pelo Time de Desenvolvimento para monitorar o progresso no desenvolvimento em direção a uma entrega (Release). O Gráfico de Release Burnup também não é

parte do *framework* Scrum, mas pode ser útil quando se utiliza um Plano de Release, geralmente estabelecido em uma reunião de Release Planning.

O Gráfico de release Burnup mostra duas linhas. Uma das linhas representa a quantidade de trabalho total previsto para a Release, correspondente aos itens escolhidos para o Plano de Release. A outra linha representa a quantidade de trabalho já realizado na Release (apenas itens prontos de acordo com a Definição de Pronto).

Assim, à medida que o Time de Desenvolvimento trabalha nos Sprints dessa Release, a quantidade de trabalho realizado sempre aumenta, aproximando-se da quantidade de trabalho total prevista. Esse, no entanto, pode variar, quando novos itens são introduzidos, quando itens previstos são divididos em itens menores e quando estimativas são refeitas.

Dessa forma, o Gráfico de Release Burnup mostra mais informações que o de Burndown: ele mostra a variação no total de quantidade de trabalho previsto para a Release.

A figura 27.5 mostra um exemplo de Release Burnup ao final do quinto Sprint da Release. Repare que, nesse exemplo, a quantidade total de trabalho, representada pela linha pontilhada, sobe ao longo da Release, o que fica visível com o gráfico.

Nos exemplos desta seção, utilizo Story Points como unidade de “trabalho realizado” e “trabalho total”, que são atribuídos pelo Time de Desenvolvimento aos itens do Product Backlog selecionados para a Release. No entanto, outras unidades podem ser usadas.

Para “tempo”, é comum se usarem os Sprints previstos para a Release. Assim, cada ponto do gráfico mostrará a soma dos Story Points dos itens já desenvolvidos do Product Backlog nessa Release, ao final de cada Sprint dessa Release.

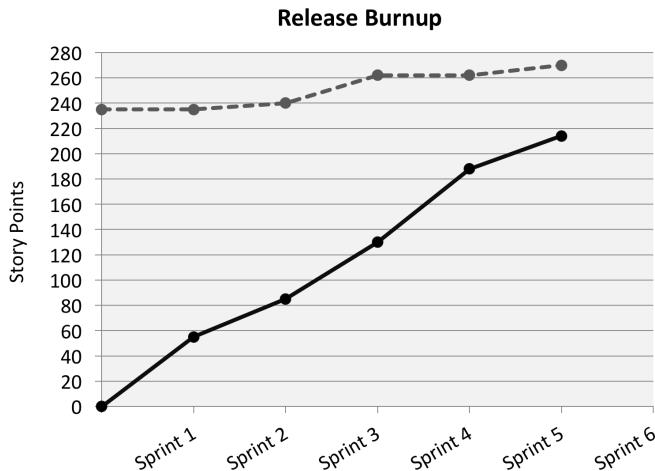


Fig. 27.5: Um exemplo de Release Burnup no final do penúltimo Sprint da Release

Outra forma comum de se definir o trabalho já realizado e o trabalho total é utilizando-se o valor de negócios de cada item. Esse valor de negócio é estabelecido pelo Product Owner ou diretamente pelos clientes do projeto. É uma unidade relativa, ou seja, obtém-se o valor de negócio de um item comparando-o com o valor de negócio de outros.

Nesse caso, o trabalho total ao final de cada Sprint é representado no Release Burnup pelo valor de negócios total esperado para aquele momento. E o trabalho já realizado é representado pelo valor de negócios já gerado até aquele momento. O Gráfico de Release Burnup é criado na reunião de Release Planning, ou logo antes do primeiro Sprint, e é atualizado ao final de cada Sprint.

27.2.2 Como é o Gráfico de Release Burnup?

Para efeitos didáticos, o exemplo da figura 27.6 mostra um Gráfico de Release Burnup e um Gráfico de Release Burndown juntos. Os gráficos mostram o trabalho em Story Points e foram atualizados logo após o final do quarto

Sprint (de um total de cinco) da Release. O gráfico pontilhado é o total de trabalho previsto, o gráfico na descendente é o Gráfico de Burndown, e o gráfico na ascendente é o Gráfico de Burnup.

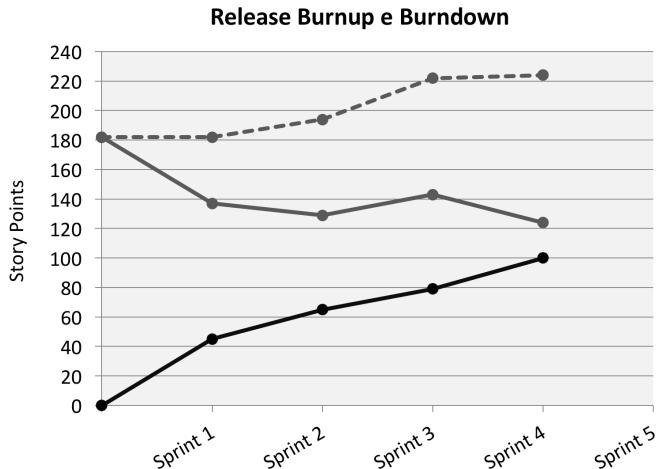


Fig. 27.6: Release Burnup e Burndown juntos

À medida que o trabalho é realizado Sprint a Sprint, o Gráfico de Burnup acumula mais pontos e sobe, aproximando-se da linha que representa o trabalho total. Quanto mais próximo dessa linha o Gráfico de Burnup estiver ao final da Release, maiores são as chances de se ter realizada a Meta da Release.

Observe que a inclinação súbita para cima do Gráfico de Burndown durante o Sprint 3 é melhor explicada pelo Gráfico de Burnup, onde se pode ver que a quantidade de trabalho total cresceu mais do que o trabalho realizado naquele Sprint. Esse é um sinal de alerta importante para o Product Owner, já que pode significar uma ameaça à Meta da Release.

Novamente é importante destacar que não se espera que o Time de Desenvolvimento complete todos os itens definidos inicialmente para a Release, mas sim que realize sua meta, trabalhando a partir dos itens mais importantes.

27.3 GRÁFICO DE SPRINT BURNDOWN

27.3.1 O que é o Gráfico de Sprint Burndown?

O Sprint Burndown é um gráfico mantido e usado pelo Time de Desenvolvimento para monitorar seu progresso no desenvolvimento em direção ao final de um Sprint. Ele mostra a quantidade de trabalho restante estimado para um Sprint (eixo Y) em cada dia de trabalho para o desenvolvimento do produto no Sprint (eixo X).

Embora não seja parte integrante do *framework* Scrum, o Gráfico de Sprint Burndown é a maneira mais rápida e prática de se visualizar o andamento do Sprint. A figura 27.7 mostra um exemplo de Sprint Burndown ao final de um Sprint bem-sucedido.

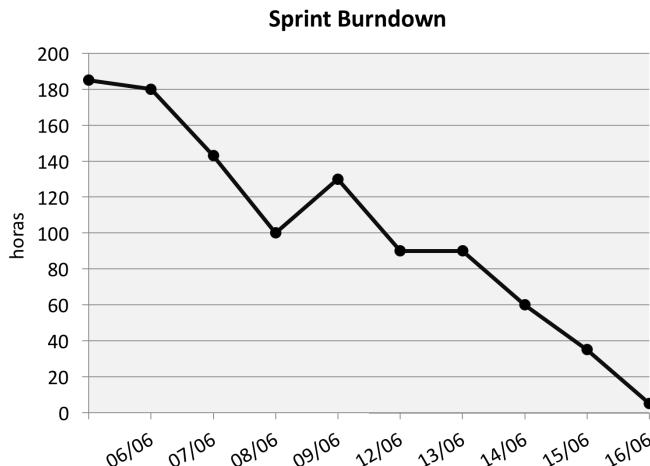


Fig. 27.7: Gráfico de Sprint Burndown ao final de um Sprint bem sucedido

A estratégia mais comum para a definição de um plano para o Sprint é a de se quebrar os itens selecionados para o Sprint Backlog em tarefas. Nesse caso, o Time de Desenvolvimento pode estimar as tarefas individualmente e utilizar essas estimativas para traçar o Gráfico de Sprint Burndown.

Se o Time de Desenvolvimento estima as tarefas em horas de trabalho, então cada ponto do gráfico mostrará a soma das horas estimadas de todas as

tarefas restantes (tanto aquelas não iniciadas quanto as que já estão sendo desenvolvidas) que fazem parte do Sprint Backlog versus o tempo, representado por cada dia útil de desenvolvimento do Sprint.

Caso o Time de Desenvolvimento use uma unidade não numérica para as estimativas das tarefas, como por exemplo (P)equeno, (M)édio e (G)rande (os “tamanhos de camisas” ou *T-Shirt Sizing*), pode-se estabelecer uma proporção entre os valores da escala (no exemplo, $1G = 2M = 4P$) e somar os valores das tarefas restantes em termos da menor unidade da escala (no caso, P) para traçar os pontos do gráfico. Outra forma é simplesmente designar um valor para cada unidade da escala (como $P = 1$, $M = 2$ e $G = 4$).

Se o Time de Desenvolvimento não estima as suas tarefas, pode-se somar o número de tarefas restantes em cada dia e marcar no gráfico. Outra alternativa é utilizar-se a estimativa de cada item restante do Sprint Backlog (em Story Points, por exemplo). Nesse caso, geralmente não se consideram pontos parciais. Assim, somente se consideram os pontos de um item como “queimados” quando o item está pronto, de acordo com a Definição de Pronto, o que gerará saltos no gráfico.

Caso o Time de Desenvolvimento utilize uma estratégia diferente e não divida os itens do Sprint Backlog em tarefas, é importante que ele busque outra forma de contabilizar o trabalho restante em cada momento para ser capaz de monitorar seu progresso em direção ao final do Sprint.

O Gráfico de Sprint Burndown é criado ao final da reunião de Sprint Planning, ou antes da primeira reunião de Daily Scrum, e é atualizado diariamente com o valor do trabalho restante naquele momento. Alguns Times de Desenvolvimento preferem atualizá-lo logo antes de cada reunião de Daily Scrum, enquanto que outros preferem fazê-lo no início ou no final do dia. O último ponto do gráfico é marcado quando o desenvolvimento se encerra no Sprint.

27.3.2 Como é o Gráfico de Sprint Burndown?

No exemplo da figura 27.8, o Time de Desenvolvimento usa horas de trabalho para estimar suas tarefas. Vamos imaginar que, nesse exemplo, o Gráfico de Sprint Burndown é atualizado logo antes do início da reunião de Daily Scrum e que essa reunião ocorre no meio da manhã de cada dia de desenvolvimento (de um total de nove).

A reunião de Sprint Planning foi realizada na primeira metade do primeiro dia, e o primeiro ponto (sobre o eixo Y) reflete a soma do total de horas de todas as tarefas definidas na reunião. O segundo ponto do gráfico (dia 06/06) reflete a quantidade de horas estimadas para as tarefas restantes (ainda não desenvolvidas ou em andamento) logo antes da primeira reunião de Daily Scrum, no segundo dia do Sprint, momento em que algumas tarefas já foram realizadas. O gráfico acaba de ser atualizado no quinto dia do Sprint e a reunião de Daily Scrum está para começar. Observe que o último ponto do gráfico (16/06) será atualizado no último dia do Sprint, logo antes da reunião de Sprint Review.

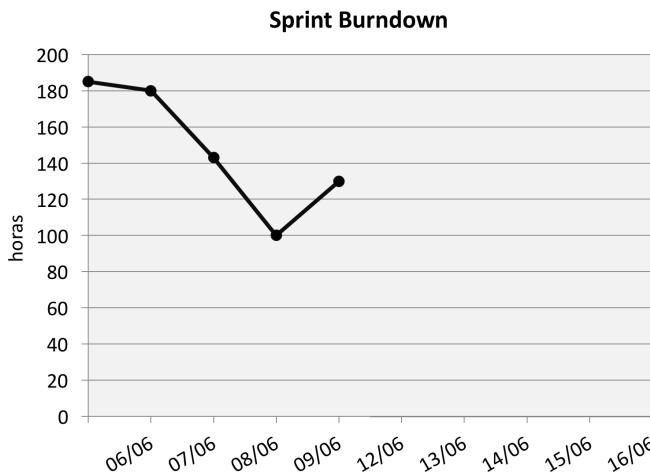


Fig. 27.8: Gráfico de Sprint Burndown com inclinação para cima

A inclinação súbita para cima no dia 09/06, de 100 para 135 horas de trabalho, serve como sinal de alerta, podendo indicar que ocorreu algum problema entre a reunião de Daily Scrum dos dias 08/06 e 09/06. A reestimativa de tarefas já existentes ou a introdução de novas tarefas são perfeitamente normais e esperadas, mas aqui alguma dessas questões pode ter ocorrido em excesso, pode ter ocorrido algum impedimento ao trabalho do Time de Desenvolvimento, ou pode ter ocorrido mais de um desses problemas ao mesmo tempo.

Nesse outro exemplo da figura 27.9, vemos pelo Gráfico de Sprint Burn-

down que o Time de Desenvolvimento trabalha em Sprints de quatro semanas, e utiliza o número de tarefas como unidade de trabalho restante. Vemos, no exemplo, que a quantidade de tarefas restantes se manteve a mesma por três dias seguidos, formando uma linha reta.

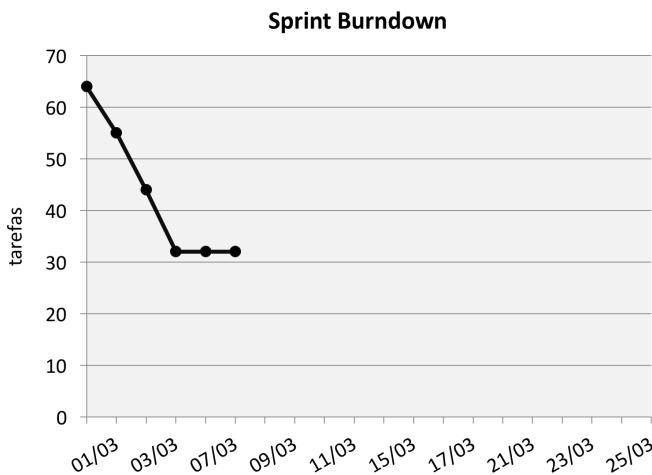


Fig. 27.9: Gráfico de Sprint Burndown com linha reta

Da mesma forma que no caso anterior, esse comportamento do gráfico também serve como sinal de alerta, já que o Time de Desenvolvimento está cada vez mais distante de “queimar” o trabalho estimado restante.

É importante destacar que, mesmo que o Time de Desenvolvimento chegue ao final de um Sprint com tarefas sobrando, ainda assim poderá ter cumprido seu compromisso. Embora dê o seu melhor para completar o trabalho, o compromisso do Time de Desenvolvimento é o de cumprir a Meta do Sprint. Mesmo que nem todos os itens selecionados tenham sido completados, ainda assim essa Meta poderá ter sido realizada (veja [24.2 Meta de Sprint](#)). Não é incomum se chegar ao final de um Sprint sem que o Sprint Burndown chegue ao zero do eixo Y, e mesmo assim pode-se considerar o Sprint um sucesso.

Nos dois exemplos da figura [27.10](#), vemos que, ao final do Sprint, não foi “queimado” todo o trabalho estimado.

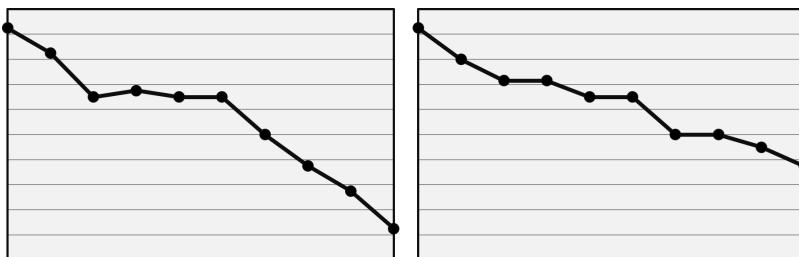


Fig. 27.10: Dois exemplos de gráficos que não chegam a zero trabalho restante

No entanto, são dois cenários com resultados bastante distintos. No primeiro caso, o Time de Desenvolvimento provavelmente entregou valor suficiente para realizar a Meta do Sprint. Mas, no segundo caso, há uma grande chance de a Meta do Sprint não ter sido realizada, já que muito deixou de ser feito.

O Gráfico de Sprint Burndown é criado pelo Time de Desenvolvimento, para o Time de Desenvolvimento. Ele serve para sinalizar para o Time de Desenvolvimento o quanto perto ou quanto longe ele está de cumprir as tarefas planejadas para o Sprint. Também não deve, de forma alguma, servir como instrumento de cobrança para o Product Owner, o ScrumMaster ou qualquer parte interessada do projeto exercer pressão sobre o Time de Desenvolvimento.

27.3.3 Linha ideal

A linha ideal serve como guia visual para o comportamento do Gráfico de Sprint Burndown ao longo do Sprint. A linha ideal é opcional e, caso utilizada, é traçada no momento que o Gráfico de Sprint Burndown é criado.

Ela é uma linha diagonal que liga a quantidade inicial de trabalho restante sobre o eixo Y ao último dia do Sprint sobre o eixo X. Ou seja, ela liga (*[primeiro dia do Sprint], [trabalho total]*) a (*[último dia do Sprint], [zero trabalho restante]*). A figura 27.11 mostra a linha ideal.

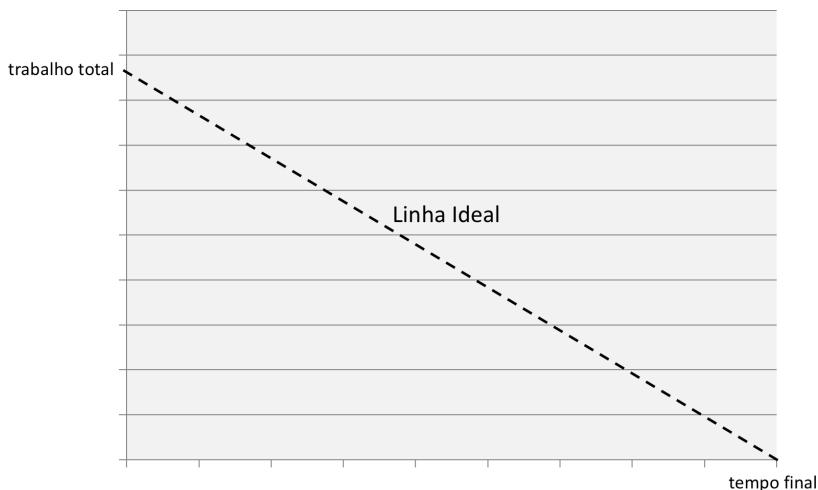


Fig. 27.11: Linha ideal

É natural, ao longo do Sprint, o gráfico oscilar em torno da linha ideal. Um exemplo de oscilação saudável pode ser visto na figura 27.12. Repare que o gráfico oscila em torno da linha ideal sem, no entanto, se distanciar muito dela.

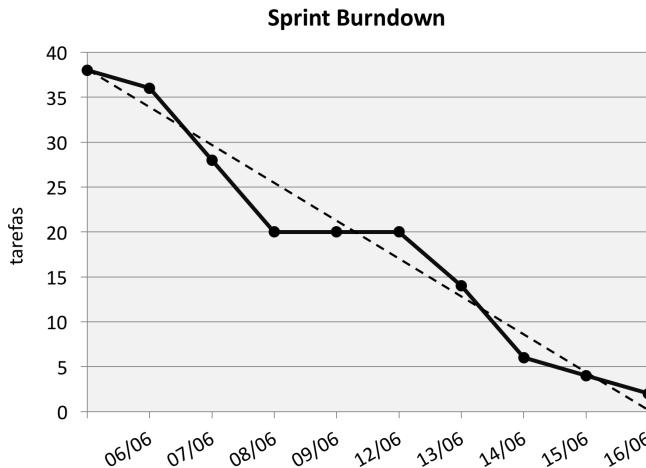


Fig. 27.12: Oscilação natural do gráfico em torno da linha ideal

No entanto, se o ponto do gráfico referente ao momento atual estiver muito acima da linha, o Time de Desenvolvimento ficará em alerta. Quanto mais próximo do final do Sprint isso acontecer, em mais risco a Meta do Sprint estará. No exemplo da figura 27.13, a situação parece crítica a três dias do final do Sprint.

Por diversas razões, a linha ideal é muito criticada por vários autores e praticantes de Scrum e, assim, muitos preferem não a utilizar. Alguns acreditam que essa linha somente serve de instrumento de pressão sobre o Time de Desenvolvimento e a chamam de “linha da opressão”. Outros acreditam que, ao buscar adequar-se ao comportamento da linha ideal, o Time de Desenvolvimento acaba por não ser honesto em suas estimativas.

Outra questão é que, ao traçar-se a linha ideal, está se considerando uma irreal precisão na definição e nas estimativas iniciais das tarefas. Um percentual significativo de tarefas surge no decorrer do Sprint e diversas tarefas são reestimadas, fazendo com que o ponto inicial da linha ideal (e, portanto, a própria linha) não tenha significado real.

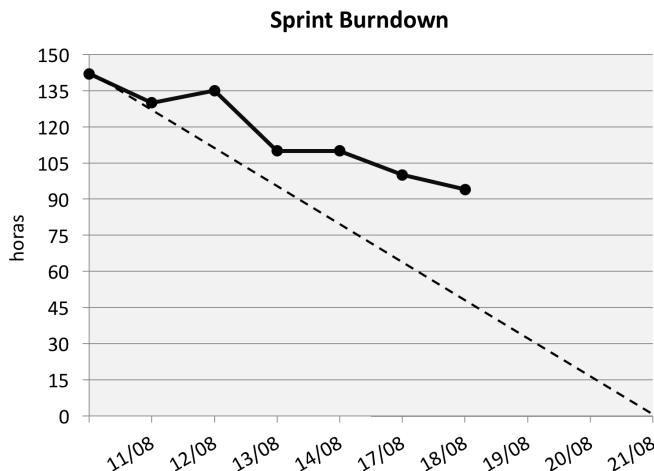


Fig. 27.13: Nesse momento, a distância do ponto atual à linha ideal pode indicar uma situação crítica

Entretanto, a linha ideal pode cumprir seu propósito se compreendida apenas como um guia visual para o comportamento do Gráfico de Sprint Burndown. Ela também pode ser usada para o Gráfico de Release Burndown, ligando a quantidade inicial de trabalho previsto sobre o eixo Y ao final do último Sprint da Release sobre o eixo X. Ou seja, ela liga (*[início do primeiro Sprint da Release]*, *[trabalho total]*) a (*[final do último Sprint da Release]*, *[zero trabalho restante]*).

Parte VI

Final

CAPÍTULO 28

Além do Scrum

— Por Marcos Garrido, Rodrigo de Toledo e Carlos Felippe Cardoso

28.1 Os QUATRO DOMÍNIOS DA AGILIDADE

Quando escrevemos o capítulo “Além do Scrum” na primeira versão deste livro, dividimos os próximos passos na caminhada do aprendizado em quatro diferentes domínios. Originalmente, o nosso objetivo era fazer um de/para das funções tradicionalmente encontradas nas empresas para os papéis do Scrum, facilitando o processo de adoção do Scrum nas organizações.

No entanto, a identificação dos quatro domínios acabou por ampliar nossos horizontes. Nos últimos anos, evoluímos muito nesse assunto e a divisão da Agilidade em quatro domínios se tornou uma ferramenta poderosa. Ela nos ajuda a fazer retrospectivas, mapeamento de conhecimento, abordagem para *coaching*, aplicação da transformação Ágil, *assessments* etc.

Os quatro domínios foram originalmente desenhados pensando no uso de Scrum para desenvolvimento de *software*. Entretanto, eles podem e devem ser compreendidos como domínios genéricos de conhecimento, aplicáveis a diversos contextos organizacionais, na área da tecnologia da informação ou em áreas diversas. A ideia é que os domínios possam ser aplicados a qualquer estrutura na qual haja trabalho criativo, ou como nós mesmos chamamos, trabalhadores do conhecimento.

Os quatro domínios são:

- **domínio de negócios** — nesse domínio, tratamos de produtos, cálculo de ROI, gestão de portfolio, contratos e metas. É o domínio ligado à estratégia organizacional.
- **domínio cultural** — esse é o domínio que mais demanda esforço em uma transformação Ágil. O *mindset* Ágil é fundamental para que se consiga realizar a transformação com sucesso. Os assuntos compreendem: melhoria contínua, motivação dos indivíduos, autonomia, liderança e experimentação.
- **domínio organizacional** — o domínio organizacional aborda temas relativos ao processo e ao fluxo de trabalho: distribuição dos times, frequência de entrega, cerimônias, até a estrutura física.
- **domínio técnico** — os assuntos técnicos são relativos à execução: qualidade, automação, padrões, ferramentas. Repare que é exatamente aqui que não há itens específicos de computação, justamente para que o domínio seja usado em outros contextos também.

A figura 28.1 relaciona, para cada domínio da Agilidade, algumas das habilidades e técnicas necessárias para que se possa evoluir nos caminhos da Agilidade no nível do indivíduo, do time e da organização.

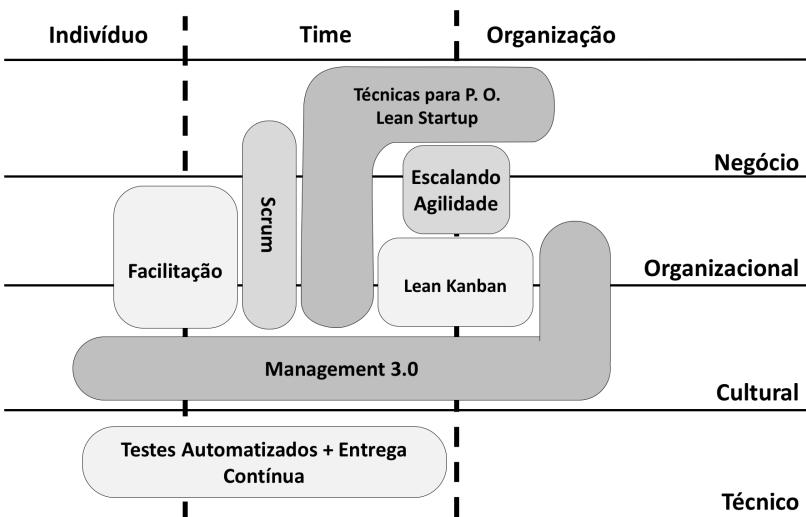


Fig. 28.1: Domínios da Agilidade

A seguir, vamos descrever essas habilidades e técnicas selecionadas.

28.2 FACILITAÇÃO

O ScrumMaster trabalha como um facilitador para ajudar o Product Owner e Time de Desenvolvimento a serem mais eficientes na realização do seu trabalho. Ele se utiliza de suas habilidades de facilitação no dia a dia de trabalho do Time de Scrum e durante suas reuniões.

No capítulo 8, há uma descrição detalhada do papel de facilitador que é exercido pelo ScrumMaster (veja [8.1 Quem é o ScrumMaster?](#)).

Apesar da importância do tema, poucos ScrumMasters parecem investir seu tempo na aquisição e desenvolvimento de habilidades em relação às técnicas de facilitação, de forma que seu trabalho não fique apenas focado em reforçar o correto uso de práticas Ágeis.

As técnicas de facilitação fornecem ferramentas para que o ScrumMaster seja um especialista em:

- identificar e lidar com comportamentos disfuncionais;

- trabalhar para que todos os envolvidos entendam seus próprios objetivos e quais resultados lhes são esperados;
- preparar corretamente o ambiente de trabalho e em que ocorrerão as reuniões;
- atuar proativamente quando os envolvidos perdem seu foco;
- trabalhar para que o grupo chegue a decisões informadas a partir do consenso, maximizando a participação de todos os envolvidos;
- criar as condições para que o processo de melhoria contínua funcione adequadamente e de forma eficiente.

Embora o ScrumMaster seja naturalmente um facilitador, todos os membros de um Time de Scrum podem se beneficiar do conhecimento de técnicas de facilitação: Product Owners podem usar técnicas de facilitação para compreender melhor as necessidades de seus clientes, usuários e partes interessadas. Membros do Time de Desenvolvimento podem utilizar técnicas de facilitação para resolver conflitos e problemas do dia a dia, de forma rápida e eficiente, sem que, para isso, tenham de recorrer o tempo todo ao Scrum-Master.

28.3 MANAGEMENT 3.0

Desde que Adam Smith, em seu livro *A riqueza das nações*, de 1776, discutiu a importância da especialização do trabalho, passando pela publicação dos princípios da administração científica por Frederic Taylor em 1911, e chegando até os dias de hoje, as organizações são desenhadas da mesma maneira. O modelo vigente segue uma orientação *top-down*, na qual as decisões são tomadas na cúpula, e então irradiadas para os níveis inferiores da organização.

Mesmo organizações mais abertas, que já adotaram métodos Ágeis, em sua maioria foram desenhadas seguindo essa lógica *top-down*. Dessa forma, os benefícios no uso dos métodos Ágeis acabam ficando restritos ao dia a dia dos times. Ao mesmo tempo, gestores que lidam com times Ágeis no cotidiano, muitas vezes se veem limitados pelas técnicas tradicionais de gestão.

Management 3.0 é um conjunto de teorias e técnicas, divididas em seis visões de gestão, que visa a fornecer a gestores e líderes os conhecimentos necessários para levar a Agilidade para as camadas superiores de gestão. Altos níveis de eficiência podem ser alcançados por meio da formação de times Ágeis empoderados, inovadores, motivados e trabalhando em estruturas que permitam a comunicação eficiente em todas as camadas da organização.

As seis visões do Management 3.0 são descritas a seguir:

- **energizar as pessoas** — como as pessoas são a parte mais importante das organizações, seus gerentes devem fazer o possível para mantê-las ativas, criativas e motivadas. Para isso, faz-se necessário entender a diferença entre motivação intrínseca e extrínseca e conhecer técnicas para identificar o que motiva as pessoas à sua volta;
- **empoderar times** — auto-organização é um valor importante para times Ágeis. Gerentes devem entender como times auto-organizados funcionam, quais são os desafios envolvidos e como construir relações de confiança mútua que levem a times auto-organizados de sucesso;
- **alinhar restrições** — restrições são importantes. Gestores devem entender a diferença entre restrições e regras, de forma que os times possam trabalhar de maneira auto-organizada, seguindo em direção aos objetivos propostos pelos gestores. Além disso, quando não há uma meta ou propósito claro, o resultado pode não ser o esperado. Também é importante que os gestores aprendam quando gerenciar e quando liderar, sabendo utilizar critérios diferentes para criar metas úteis;
- **desenvolver competências** — times só conseguem atingir suas metas se seus membros forem capazes o suficiente. Embora times auto-organizados possam lidar com suas lacunas técnicas, gerentes devem contribuir para o desenvolvimento das competências necessárias dos membros do time;
- **crescer a estrutura organizacional** — o desenho da estrutura organizacional impacta de forma significativa em como a organização funciona. Vários times operam no contexto de uma organização complexa, e

por isso é importante considerar uma estrutura que privilegie a comunicação. Gestores precisam entender a diferença entre times funcionais e multifuncionais, e conhecer técnicas para desenhar áreas que funcionam e se comunicam com eficácia;

- **melhorar continuamente** — é um dos maiores desafios nas organizações. Pessoas, times e organizações precisam melhorar continuamente. Na prática, isso significa que gestores e líderes devem agir como agentes da mudança, tentando mudar os complexos sistemas sociais à sua volta.

28.4 LEAN KANBAN

Os métodos Ágeis trouxeram uma grande mudança nos rumos da computação. O Scrum, seu maior ícone, tem liderado essa nova forma de pensar, quebrando diversos paradigmas das décadas passadas. O método Kanban surgiu depois dessa grande onda, em 2007, explicitando ainda mais essas quebras de paradigmas. Conceitos como fluxo contínuo, foco, visibilidade e melhoria contínua são reforçados.

A **visibilidade** é sempre o primeiro passo: colocar na parede em forma de cartões o que está sendo efetivamente feito pelo time. Há então dois movimentos nessa visibilidade. Um é interno ao time, pois muitas vezes apenas três colunas não representam as etapas de trabalho de um time. É comum, portanto, surgirem outras colunas para etapas, como “em teste”, “integração”, “especificação”, entre outras, além de colunas para representar as filas que ocorrem entre essas fases.

Um outro movimento da visibilidade é para fora do time, pois a tendência é que a gestão visual extrapole as paredes do time e comece o cobrir todo o fluxo, “da concepção ao retorno financeiro” (*from concept to cash*). Atividades que são realizadas antes e depois da implementação do time passam a ser também mapeadas, como por exemplo: concepção, análise, homologação e subida à produção.

Limitar a quantidade de trabalho (*limited WIP — work in progress*) talvez seja a maior contribuição do Kanban para a Agilidade de um time ou de uma empresa. A afirmação de que quanto menos trabalho fazemos ao mesmo

tempo, maior é a nossa eficiência, pode parecer um pouco contraintuitiva, mas como demonstra a Lei de Little, essa restrição é a chave para diminuir tempo de entrega. O time é mais focado e o limite faz com que não entre em zonas de conforto, como começar algo menos importante ou não correr atrás de um impedimento.

Finalmente, o **fluxo** é uma consequência das ações anteriores (visibilidade e quantidade de trabalho limitado). Com objetivo de manter o fluxo contínuo de entrega, diversos mecanismos emergem. Um dos possíveis efeitos é o de que o Sprint deixe de existir e a entrega passe a ser item a item. Mas observe que é uma opção avançada, logo, muitos times que adotam Kanban escolhem permanecer com iterações de *timebox* fixo.

Além desses pontos, há muito o que se aprender e alguns assuntos importantes são tratados somente em treinamentos avançados de Kanban. Por exemplo, como abordar problemas usando um pensamento sistêmico, melhoria contínua baseada nos cinco porquês e relatórios A3, criação de *design* de sistemas de trabalho, modelos econômicos e métricas. Tais assuntos avançados podem se tornar verdadeiras alavancas de melhoria contínua dos times e suas empresas.

Scrum e Kanban

— Por Rafael Sabbagh

A partir da disseminação de suas práticas, Scrum evoluiu nos últimos anos para tornar-se ainda mais leve e ainda menos prescritivo, de forma a adaptar-se a seus diversos usos. Surgiram novas formas de se conduzirem suas reuniões e de se usarem seus artefatos. Várias de suas regras estritas tornaram-se recomendações, enquanto outras foram flexibilizadas.

Por exemplo, o tamanho da equipe, que era de entre cinco e nove membros, agora pode ser tão curto quanto três membros. Há também times de dois utilizando Scrum com sucesso, assim como times com mais — mas não muito mais — do que nove membros. Por outro lado, brotaram novas formas de se usar Scrum em projetos maiores, com diversas equipes trabalhando no mesmo produto.

Outro exemplo é a duração dos Sprints, que era originalmente de um mês, e logo passou a poder ser tão curta quanto duas semanas. Posteriormente,

verificou-se que vários times são bem-sucedidos com Sprints de apenas uma semana.

Novas práticas passaram a ser importantes, como as sessões de refinamento do Product Backlog. Estas servem para a preparação do que deve ser feito no próximo Sprint, e são realizadas em conjunto pelo Product Owner e membro do Time de Desenvolvimento, antecipando algo que acontecia apenas na reunião de Sprint Planning.

Esse próprio Sprint Planning passou a ser flexibilizado por alguns times. Em vez de planejarem para todo o Sprint em seu princípio, esses times quebram em tarefas cada um dos itens de trabalho previamente selecionados apenas na medida do necessário, durante todo Sprint.

Kanban, um *framework* criado em 2007 por David Anderson, é visto por muitos como uma evolução do Scrum. Com Kanban, rompem-se as iterações e prega-se o trabalho em um fluxo contínuo, junto com o que se associam outras práticas importantes.

Scrum e Kanban têm raízes comuns no Sistema Toyota de Produção, que mencionamos no capítulo *De onde veio o Scrum?* (veja [5.3 Lean e Sistema Toyota](#)). Embora seja muito mais leve e menos prescritivo, Kanban em geral tem maiores chances de sucesso quando complementado com práticas importantes do Scrum.

Entre essas práticas desejadas, podemos citar: o desenvolvimento incremental do produto; a priorização do trabalho a partir do que é mais importante para os clientes; o trabalho em equipe, que se auto-organiza para realizá-lo; a presença de um facilitador, que auxilia a equipe a se auto-organizar; as reuniões frequentes de retrospectiva para a equipe tornar-se cada vez mais efetiva; e o trabalho em direção a metas de negócios claras e desafiadoras, mas alcançáveis.

Dessa forma, talvez o Kanban possa ser visto não como uma evolução que visa a substituir seu antecessor, o Scrum, já que há tantas equipes no mundo utilizando Scrum com sucesso e não parece haver motivos para isso mudar tão cedo. Kanban é sim uma evolução, mas que na realidade flexibiliza Scrum ainda mais em uma direção que Scrum hoje já caminha naturalmente, tornando-o menos prescritivo e abrindo-o para uma gama maior de práticas e possibilidades.

Seja como for, adicionalmente ao que prega qualquer dos dois *frameworks*, o conjunto total de práticas usadas varia de projeto para projeto e de equipe para equipe. Uma dessas práticas possíveis para um Time de Scrum poderia ser, por exemplo, o trabalho em iterações tão curtas que se transformam em um fluxo contínuo. Assim, deixando de lado antagonismos, marketing e preciosismos, talvez possamos afirmar que, na realidade, Kanban e Scrum são — ou serão — uma coisa só.

28.5 TÉCNICAS PARA PRODUCT OWNERS

A função do Product Owner é a chave do sucesso de qualquer projeto com Scrum. O Product Owner determina a Visão do Produto e direciona o Time de Desenvolvimento, fazendo sua conexão com clientes e negócios. Times de Scrum podem avançar extremamente rápido e criar produtos de alta qualidade técnica. Investe-se muito tempo e esforço em tornar os Times de Desenvolvimento cada vez mais produtivos. No entanto, nada é mais improdutivo do que criar um produto ou serviço que não atende às necessidades dos clientes, ou seja, que não gera retorno sobre o investimento (ROI).

As três atividades mais importantes de um bom Product Owner são:

- **fatiar** — épicos carregam muita incerteza. Fatiar ajuda a reduzir essa incerteza, uma vez que se reduz a quantidade de detalhes necessários para um mesmo item. Além disso, aumenta-se a precisão das estimativas e permite-se um encaixe mais fácil nos Sprints, o que implica no ganho de flexibilidade na gestão do Product Backlog. O Product Owner tem à sua disposição um conjunto de critérios para fatiar os itens de Backlog: estratégia, desempenho, passos de um fluxo de trabalho, valor de negócio e complexidade, por exemplo. Após fatiar os itens, pode-se utilizar a técnica de descarte para eliminar complexidade desnecessária, conforme explicado mais à frente;
- **priorizar** — uma priorização bem-feita permite que os itens mais importantes para o cliente possam ser entregues primeiro. Ao entregar os itens mais importantes mais cedo, aprende-se mais sobre o produto, garante-se o ROI, reduz-se o *time-to-market* e o cliente fica mais satisfeito. No entanto, priorizar não é uma tarefa fácil. É preciso entender

profundamente sobre o mercado e quais são as reais necessidades do cliente. É uma arte. Mas, como toda arte, existem técnicas apuradas que suportam todo o processo, sempre a partir do ponto de vista do cliente. Há inúmeras maneiras para se calcular e maximizar o ROI do Product Backlog e, como cada projeto possui características específicas, não existe uma resposta única que atenda a todos os casos;

- **descartar** — assim como o Product Owner deve priorizar os itens de Backlog para descobrir o que há de mais importante para ser feito, descobrir quais itens devem ser descartados é fundamental para manter o foco na entrega de valor. Ao fatiar épicos, por exemplo, o Product Owner pode buscar a simplificação, descartando itens que entregam pouco ou nenhum valor e, consequentemente, não contribuem para a satisfação do cliente. Pode-se utilizar também o conceito de MVP (*Minimum Viable Product* ou produto mínimo viável) para ajudar a eliminar deta-lhes dos itens.

Treinamentos devem formar o Product Owner, ensinando técnicas como: construção da Visão do Produto, trabalho com épicos e personas, escrita de User Stories efetivas, fatiamento de User Stories, atribuição de valor de negócios, criação de critérios de aceitação, priorização, Lean Startup e MVP.

28.6 LEAN STARTUP

Cenários em que o cliente é desconhecido (quando se constrói um produto novo para se lançar no mercado) são complexos exatamente porque a ausência do cliente torna o processo de descoberta do produto muito mais difícil de ser executado. No entanto, mesmo quando ele é conhecido (e muitas vezes o próprio contratante do projeto), o processo de descoberta do produto é crucial para o sucesso do projeto.

Product Owners tem à disposição um conjunto de técnicas de descoberta de produto que permitem entender as necessidades do cliente. A fase de descoberta é, na verdade, um grande processo de aprendizado, pois é por meio da tentativa e erro que se descobre qual é o produto ideal a ser construído.

Quanto mais cedo for feita, melhor, pois é possível entregar valor para o cliente desde o início do projeto. Quando é feita tarde demais, corre-se o

risco de jogar fora todo o esforço empregado até então. Uma das formas mais discutidas ultimamente para entender o que o cliente quer é o Lean Startup.

Lean Startup é uma abordagem para a idealização de novos produtos e serviços com base na geração de aprendizado sobre o mercado, o produto e o cliente a partir da validação de hipóteses. Hipóteses são crenças que o responsável pela concepção do produto tem, baseado em sua experiência pessoal e observação, mas que precisam ser validadas, ou seja, precisam ser transformadas em fatos.

A validação das hipóteses é potencializada a partir do lançamento de um MVP (*Minimum Viable Product* ou produto mínimo viável). O MVP representa a versão mais simples possível que pode ser desenvolvida com o objetivo de gerar aprendizado sobre o usuário, de forma que as primeiras crenças sobre o produto possam ser validadas.

Embora tenha sido pensado para atender às necessidades de validação de *startups*, o Lean Startup é aderente ao trabalho do Product Owner, que pode passar a trabalhar com hipóteses em vez de requisitos, de forma que ele se certifique de que o seu Product Backlog refletia o problema que o cliente realmente precisa resolver.

28.7 ESCALANDO SCRUM

Scrum foi concebido para times pequenos, ou seja, que possuem entre 3 e 9 membros. Assim, para empresas de médio e grande porte que possuem times grandes, pode ser necessário adotar técnicas adicionais para se utilizar o *framework*.

Nos últimos anos, surgiu um número de *frameworks* cujo objetivo é contornar essa limitação. LeSS, SAFe, Nexus, Scrum at Scale e Enterprise Scrum são alguns dos mais conhecidos.

Um mecanismo mais simples e que tem o mesmo objetivo é conhecido como Scrum of Scrums (SofS), e é usado quando há múltiplos times trabalhando em um mesmo projeto. SofS é uma reunião adicional à reunião de Daily Scrum, em que representantes dos times se encontram e cada um informa o que seu time realizou desde a última reunião de SofS, o que seu time pretende realizar até a próxima, e quais impedimentos enfrentaram ou estão

enfrentando.

Os presentes mantêm o foco dessa interação nas interdependências, ou seja, no que o trabalho de um time afetou ou afetará o trabalho de outros times. Os times envolvidos devem decidir em conjunto qual a frequência e duração das reuniões, em geral diárias com quinze minutos de duração ou semanais, com uma duração mais longa. O representante de cada time que comparece à reunião de SofS deve ser escolhido por seu respectivo time e é sempre um de seus membros, não o ScrumMaster ou o Product Owner.

Já o contexto em que múltiplos times trabalham em múltiplos projetos é um pouco diferente. Esse cenário, apesar de ser bem comum em empresas, não apresenta uma nomenclatura formal na comunidade Ágil. Por essa razão criamos e adotamos o termo Scrum and Scrum (SandS) para a sua descrição (SOUZA et al., 2012).

Tanto SofS como SandS demandam mecanismos específicos para controle de prioridade e dependência (ou independência) dos times. No SofS, é comum encontrar uma hierarquia de Product Owners responsável pela manutenção do Product Backlog, que normalmente é compartilhado pelos times.

No SandS, times trabalham em projetos independentes e não há coordenação alguma entre os seus Product Owners. Porém, por se tratarem de times que trabalham na mesma empresa ou gerência, sempre há dependências e impedimentos comuns que devem ser trabalhados. Assim, fazem-se necessários o entendimento, a classificação e a visualização dessas dependências e impedimentos, de forma que possam ser tratados em conjunto pelos times envolvidos.

28.8 TESTES AUTOMATIZADOS E ENTREGA CONTÍNUA

A Agilidade traz a quebra de uma série de paradigmas, inclusive na forma de construirmos e testarmos o *software*. Ela deixa muito claro que os times não podem abrir mão da qualidade, que é dita como um valor inegociável. Passamos também a construir o *software* por fatias (uma nova funcionalidade completa, da tela ao banco de dados), e não mais por camadas (camada de banco de dados, de acesso aos dados etc.). Isso causa um profundo impacto

no ciclo de testes tradicional de *software*.

Muitos times Scrum atacam esta questão com analistas de testes em sua composição. A grande questão é que a presença desse profissional pode não ser suficiente para que consigamos, em tempo hábil para a Sprint Review, testar todas as novas fatias entregues e retestar o sistema, que pode ter sido impactado pela adição das novas funcionalidades.

A automação de boa parte destes testes realizados pelos analistas de testes e pelos demais membros do time exponencia o benefício de um *software* testado, pois em apenas alguns minutos de centenas a milhares de testes podem ser rodados.

Para sistemas de *software*, existem várias camadas de testes possíveis de serem feitas: testes unitários, testes integrados, testes de serviço, testes de carga, testes funcionais, testes de interface e testes de aceitação, por exemplo. Há também uma série de técnicas e práticas também nessa área, incluindo TDD (*Test Driven Development*), ATDD (*Acceptance Test Driven Development*) e BDD (*Behavior Driven Development*).

Para automatizar o processo de execução dos testes a cada nova alteração que seja feita no código do *software*, podemos adotar a prática da Integração Contínua, na qual todos os testes, de qualquer tipo, são executados e um *feedback* (e-mail, tela etc.) é dado ao Time de Scrum.

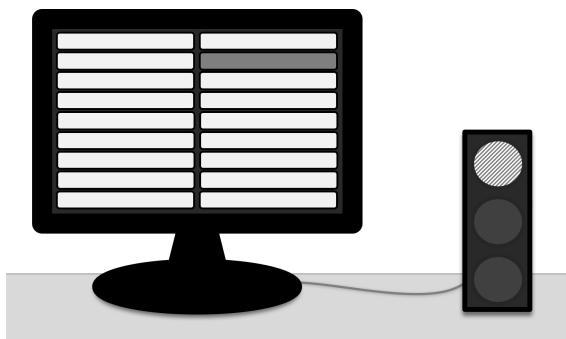


Fig. 28.2: Possível feedback proveniente do Jenkins, uma ferramenta de Integração Contínua, em que algum teste de um dos projetos listados apresentou um erro

O desenvolvimento de *software* tem evoluído também na busca da redução do tempo de *feedback*. Os produtos desenvolvidos hoje em dia passaram a focar nessa questão, por exemplo, evoluindo das entregas anuais (*waterfall*) para entregas trimestrais (RUP), entregas mensais/semanais (Scrum/XP), até as entregas contínuas (Kanban/Lean Startup).

O valor de reduzir o tempo de *feedback* na construção de um produto é inegável: antecipação do valor; correção de desvios; e adaptação a mudanças.

Quando se trata do trabalho do time, uma série de evoluções também ocorreram. Podemos classificar essas evoluções em níveis de maturidade e automação:

Nível 1 — Integração Contínua

- **Objetivo:** responder o mais rápido possível à pergunta “estamos construindo da maneira correta?“;
- **nível de automação:** simples;
- **como alcançar:** montar um ambiente de Integração Contínua que valide que o *software* foi construído da maneira correta, propiciando segurança para o time. Esse ambiente utiliza-se de práticas como testes automatizados, além de práticas de codificação e versionamento de código.

Nível 2 — Pipeline de Build e Implantação Contínua

- **Objetivo:** responder o mais rápido possível à pergunta “estamos construindo o *software* de acordo com o que o cliente definiu?”. O *software* pode parecer estar correto segundo a Integração Contínua, mas não necessariamente está de acordo com o que o cliente (ou Product Owner) definiu;
- **nível de automação:** médio;
- **como alcançar:** automação do processo de implantação (*deploy*) e gerenciamento de dependências, práticas de *review* contínuo e montagem de ambiente intermediário que simule o ambiente de produção

para que se possa validar a nova versão antes mesmo da liberação para produção.

Nível 3 — Entrega Contínua

- **Objetivo:** responder o mais rápido possível à pergunta “estamos construindo o *software* que o usuário precisa?”. O *software* pode estar ok segundo a Integração Contínua e ter recebido o aceite do Product Owner ou cliente. Mas o que realmente agrega valor é entregar o *software* para o usuário final. É fundamental diminuir o tempo que levamos para entregar para os reais consumidores;
- **nível de automação:** extremo;
- **como alcançar:** automação do ciclo de Release, práticas DevOps, definição de métricas, montagem de ambiente de produção automatizado e preparado para escalabilidade e resolução automatizada de problemas.

Para evoluir e atingir a maturidade da Entrega Contínua, há uma sequência de conhecimentos acumulados: testes automatizados, TDD, versionamento, integração contínua, definição de ambientes de testes e homologação, práticas de *blue-green deployment*, *canary releasing*, *smoke tests* etc. O perfil técnico deve buscar continuamente absorver esses conhecimentos.

28.9 CONCLUSÃO

Shu-ha-ri é conceito provindo das artes marciais japonesas, trazido para o contexto da Agilidade por Alistair Cockburn. Shu-ha-ri são os três níveis ou etapas necessários para o completo desenvolvimento de uma habilidade, conforme descritos a seguir (veja a figura 28.3):

- **shu:** nessa fase, o estudante é um aprendiz, e seu objetivo é compreender os fundamentos para futuramente dominá-los, sem promover modificações ou customizações e praticando o que aprendeu com alto grau de fidelidade;

- *ha*: uma vez que dominou os fundamentos e adquiriu experiência, o aprendiz possui maturidade suficiente para compreender o que está por trás das técnicas que aprendeu e, a partir daí, pode promover melhorias. No uso do Scrum, esse é o momento em que o profissional passa a adicionar técnicas e práticas avançadas ao *framework*, de forma a buscar ferramentas que lhe permitam ser cada vez mais eficiente no seu papel;
- *ri*: o terceiro e último estágio é conhecido como o estágio da transcendência ou libertação. Nessa fase, o profissional é como um faixa preta de judô, que domina as técnicas que utiliza e seus fundamentos, de modo que consegue agir naturalmente ao colocar em prática o que aprendeu. A partir do domínio dos fundamentos, o profissional passa a produzir conhecimento original, contribuindo para a comunidade com novas técnicas, práticas e teorias.



Fig. 28.3: Shu ha ri: etapas para o completo desenvolvimento de uma habilidade

Ensinar Scrum e ajudar o leitor a praticá-lo bem é o principal objetivo deste livro. No entanto, um livro pode apenas levar o leitor ao *shu* e, no máximo, indicar caminhos possíveis que o ajudem a seguir para o *ha*. As habilidades e técnicas mostradas neste capítulo complementam esses caminhos já mostrados ao longo do livro, sugerindo alguns dos passos seguintes que um verdadeiro praticante de Scrum pode seguir para obter sucesso em seus projetos a partir do uso de práticas Ágeis.

Assim, para chegar ao *ha*, pratique o Scrum. Experimente novas técnicas. Participe de treinamentos. E, se esbarrar em obstáculos muito altos, busque a ajuda de um Agile Coach profissional. Não tenha medo de errar. Ao contrário, aprenda com seus erros. Persiga a melhoria contínua. E, afinal, chegue ao *ri*, em que usar Scrum ou outra técnica reconhecida não mais terá importância.

CAPÍTULO 29

Apêndice - Glossário

Em ordem alfabética.

Clientes do projeto: pessoas, grupos ou organizações que solicitam o projeto ou apenas o patrocinam, e recebem o retorno ao investimento, uma vez que o que é produzido lhes é entregue.

Critérios de Aceitação: critérios que documentam os detalhes da User Story, expressos por enunciados pequenos e de fácil entendimento, que são utilizados para determinar quando a funcionalidade produzida pelo Time de Desenvolvimento está completa e, assim, nada mais deve ser adicionado a ela.

Daily Scrum: reunião curta, realizada diariamente pelo Time de Desenvolvimento, com o propósito de planejar o próximo dia de trabalho, proporcionando visibilidade ao trabalho realizado e a realizar, promovendo a comunicação sobre esse trabalho, dando visibilidade a quais obstáculos atrapalharam o desenvolvimento e servindo de oportunidade para decisões rápidas com relação ao progresso do Sprint.

Definição de Preparado: acordo formal entre o Product Owner e o Time de Desenvolvimento sobre o estado em que um item do Product Backlog deve estar para estar qualificado para discussão na reunião de Sprint Planning, que visa a garantir que o item chegue à reunião preparado segundo um critério bem definido.

Definição de Pronto: acordo formal entre o Product Owner e o Time de Desenvolvimento sobre o que é necessário para se considerar que um item ou o Incremento do Produto produzido no Sprint está “pronto”.

Gráfico de Release Burndown: gráfico mantido pelo Product Owner, usado tanto por ele quanto pelo Time de Desenvolvimento para monitorar o progresso no desenvolvimento em direção a uma entrega (Release), através da “queima” do trabalho previsto para a Release.

Gráfico de Release Burnup: gráfico mantido pelo Product Owner, utilizado tanto por ele quanto pelo Time de Desenvolvimento para monitorar o progresso no desenvolvimento em direção a uma entrega (Release), através do acúmulo de trabalho realizado em direção ao trabalho previsto para a Release.

Gráfico de Sprint Burndown: gráfico mantido e usado pelo Time de Desenvolvimento para monitorar seu progresso no desenvolvimento em direção ao final de um Sprint, através da “queima” do trabalho previsto para o Sprint.

Impedimento: obstáculo ou barreira que dificulta significativamente ou impede que o trabalho do Time de Desenvolvimento seja realizado, bloqueando um ou mais itens do Sprint Backlog de forma a ameaçar a Meta do Sprint.

Incremento do Produto: incremento de funcionalidades do produto prontas, de acordo com a Definição de Pronto, produzida pelo Time de Desenvolvimento em cada Sprint a partir dos itens do Product Backlog.

Meta de Release: objetivo ou necessidade de negócios de alto nível a ser realizada por meio do trabalho do Time de Desenvolvimento para uma Release.

Meta de Roadmap: objetivo ou necessidade de negócios de alto nível a ser realizada por meio do trabalho do Time de Desenvolvimento até uma determinada data ou marco no Roadmap do Produto.

Meta de Sprint: objetivo ou necessidade de negócios bem definida, obrigatoriamente estabelecida e acordada entre Product Owner e Time de Desenvolvimento durante a reunião de Sprint Planning, que deve ser realizada pelo Time de Desenvolvimento em seu trabalho no Sprint.

Partes interessadas: pessoas, grupos ou entidades que podem afetar, ser afetadas por ou se entenderem afetadas por decisões, atividades ou resultados de um projeto. Partes interessadas possuem algum interesse em jogo no projeto. Clientes, usuários, o próprio time do projeto, departamentos determinados da organização, gestores são exemplos de possíveis partes interessadas. Termo traduzido de *stakeholders*, em inglês.

Plano da Release: plano que indica qual o objetivo a ser realizado por meio da Release e quando será realizado, contendo assim a Meta da Release, a data exata ou aproximada em que a entrega será realizada e um conjunto de itens selecionados do Product Backlog para a Release.

Product Backlog: lista ordenada, planejável, emergente e gradualmente detalhada, criada e mantida pelo Product Owner, que evolui ao longo de todo o projeto e contém o que se acredita que será desenvolvido pelo Time de Desenvolvimento para se realizar a Visão do Produto, como necessidades ou objetivos de negócios dos clientes, melhorias a serem realizadas no produto, correções de problemas, questões técnicas, pesquisas que forem necessárias etc.

Product Owner: pessoa responsável por definir o produto a ser desenvolvido, de forma a garantir e maximizar, a partir do trabalho do Time de Desenvolvimento, o retorno sobre o investimento do projeto.

Quadro de Tarefas: quadro frequentemente utilizado para representar o Sprint Backlog, dividido em colunas que contêm os itens selecionados para o Sprint e as suas tarefas correspondentes. As tarefas são distribuídas entre as colunas “A Fazer”, “Fazendo” e “Feito” (ou termos similares), de acordo com seu andamento no Sprint. Os itens e as tarefas são muitas vezes escritos em notas adesivas. Alguns softwares simulam o Quadro de Tarefas, representando-o virtualmente.

Refinamento do Product Backlog: trabalho de criação, detalhamento e modificação de itens do Product Backlog realizado pelo Product Owner de forma contínua ao longo de todo o projeto e também realizado em sessões

ao longo do Sprint em conjunto pelo Product Owner e Time de Desenvolvimento, com o propósito de preparar itens para o Sprint seguinte.

Release: entrega de um ou mais Incrementos do Produto prontos gerados pelo Time de Desenvolvimento em um ou mais Sprints sucessivos, que em conjunto possuem valor suficiente para serem utilizados.

Release Planning: reunião na qual se planeja a próxima Release, a partir da criação do Plano da Release, que contém a Meta da Release, uma data e um conjunto de itens selecionados do Product Backlog para a Release.

Roadmap do Produto: plano em alto nível de como o produto evoluirá ao longo do tempo até um momento futuro determinado, expresso por uma linha do tempo com marcos, que são datas no futuro e objetivos do produto a serem realizados.

ScrumMaster: pessoa que facilita e potencializa o trabalho do Time de Scrum, e nesse trabalho ensina Scrum para o Time de Scrum, remove impedimentos que previnem o Time de Scrum de realizar seu trabalho e promove as mudanças organizacionais necessárias.

Sprint: ciclo de desenvolvimento de duração fixa, em que o Incremento do Produto pronto é gerado pelo Time de Desenvolvimento a partir dos itens mais importantes do Product Backlog, e que vai desde a reunião de planejamento até as reuniões de encerramento.

Sprint Backlog: lista de itens selecionados na reunião de Sprint Planning do alto do Product Backlog para o desenvolvimento do Incremento do Produto no Sprint (o quê), adicionada de um plano de como esse trabalho será realizado (como), geralmente expresso por um conjunto de tarefas correspondente a cada item.

Sprint Planning: reunião realizada no primeiro momento do primeiro dia do Sprint, na qual Product Owner e Time de Desenvolvimento planejam o que será desenvolvido no Sprint corrente e Time de Desenvolvimento planeja como o que foi selecionado será desenvolvido.

Sprint Retrospective: última reunião realizada no Sprint, em que o Time de Scrum inspeciona o Sprint que está se encerrando quanto a seus processos de trabalho, dinâmicas, comportamentos e ambiente, e planeja as melhorias necessárias a serem executadas no próximo Sprint.

Sprint Review: reunião realizada no último dia do Sprint em que o Time

de Scrum demonstra o Incremento do Produto gerado no Sprint para obter *feedback* dos clientes e demais partes interessadas, e modificar o que será produzido em seguida de acordo.

Partes interessadas do projeto: pessoas, grupos ou organizações que contribuem de alguma forma para o projeto ou simplesmente são informados do seu progresso. Além de clientes e usuários, esse grupo pode incluir patrocinadores do projeto, executivos e gerentes da organização etc.

Story Point: unidade relativa criada pelo Time de Desenvolvimento para ser utilizada em estimativas do esforço necessário para desenvolver um item de trabalho.

Timebox: é uma alocação máxima ou fixa de tempo dentro da qual uma atividade deve ocorrer. Uma vez alcançado o tempo máximo definido, a atividade deve ser terminada imediatamente, mesmo que ainda haja trabalho a ser realizado. A ideia é que o *timebox* tenha o valor educativo de ensinar, a partir das consequências do trabalho incompleto, a se realizar em próximos *timeboxes* o trabalho previsto dentro do tempo determinado.

Time de Desenvolvimento: grupo multidisciplinar e auto-organizado de pessoas, responsável por realizar o trabalho de desenvolvimento do produto propriamente dito.

Time de Scrum: time formado pelo Time de Desenvolvimento, Product Owner e ScrumMaster.

User Story: descrição concisa, simples e leve de uma necessidade do usuário do produto sob o ponto de vista desse usuário, que funciona como um convite para uma conversa entre as pessoas de negócios e desenvolvedores para definirem os detalhes do que deve ser desenvolvido.

Usuário do produto: pessoa que recebe e utiliza o produto incrementalmente gerado no decorrer do projeto.

Velocidade do Time de Desenvolvimento: é a média da quantidade de trabalho produzido pelo do Time de Desenvolvimento nos últimos Sprints, medida pela taxa média de queima de Story Points por Sprint ou de itens por Sprint.

Visão do Produto: objetivo ou necessidade de negócios de alto nível que fornece contexto, orientação, motivação e inspiração para todo o trabalho de

desenvolvimento do produto, alinhando o entendimento de todos os envolvidos no projeto sobre o que deve ser realizado.

CAPÍTULO 30

Apêndice - Bibliografia

ARMONY, R. S. 196 f. *Fatores críticos para a prática de valores Ágeis em equipes de tecnologia da informação.* Dissertação (Mestrado em Administração de Empresas) — Instituto de Administração e Gerência, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, 2010.

COCKBURN, A. *Agile software development: the cooperative game.* 2. ed. Massachusetts: Addison-Wesley, 2007.

COHEN, D.; LARSON, G.; WARE, B. *Improving software investments through requirements validation.* In: Proceedings of 26th Annual NASA Goddard Software Engineering Workshop. Greenbelt, MD, Estados Unidos: IEEE, nov. 2001, p. 106-114. Disponível em <<http://www.sente.com/Presentations/software%20investments%20sew26.pdf>>. Acesso em 30 mar. 2016.

COHN, M. *User Stories applied: for Agile software development.* Massa-

chusetts: Addison-Wesley, 2004.

COHN, M. *Agile estimating and planning*. Englewood Cliffs: Prentice Hall PTR, 2005.

CRISPIN, L.; GREGORY, J. *Agile testing: a practical guide for testers and Agile teams*. Boston: Addison-Wesley Professional, 2009.

CUMMINGS, T. G. *Self-regulating work groups: A socio-technical synthesis*. The Academy of Management Review, v. 3, n. 3, p. 625-634, jul. 1978.

DEGRACE, P.; STAHL L. H. *Wicked problems, righteous solutions: a catalogue of modern software engineering paradigms*. Nova Jersey: Yourdon Press, 1990.

DERBY, E.; LARSEN, D. *Agile retrospectives: making good teams great*. Texas: Pragmatic Bookshelf, 2006.

DRUSKAT, V. U.; WHEELER, J. V. *Managing from the boundary: the effective leadership of self-managing work teams*. Academy of Management Journal, Mississippi State, MS, Estados Unidos, v. 46, n. 4, p. 435-457, 2003.

GREENING, J. *Planning Poker or how to avoid analysis paralysis while release planning*. 2002. Disponível em: <<http://www.renaissancesoftware.net/files/articles/PlanningPoker-v1.1.pdf>> . Acesso em: 15 dez. 2012.

GROSS, J. M.; MCINNIS, K. R. *Kanban made simple: demystifying and applying Toyota's legendary manufacturing process*. Nova Iorque: AMACOM, 2003.

HACKMAN, J. R.; OLDHAM, G.; JANSON, R.; PURDY, K. *A new strategy for job enrichment*. California Management Review, Berkeley, CA, Estados Unidos, v. 17, n. 4, p. 55-71, 1975.

HAMPSON, I. *Lean Production and the Toyota Production System or, the case of the forgotten production concepts*. Economic and Industrial Democracy, v. 20, p. 369-391, 1999.

HIGHSMITH, J. *Agile software development ecosystems*. Boston: Addison-Wesley, 2002.

HIGHSMITH, J. *Agile project management: creating innovative products*. Massachusetts: Addison-Wesley, 2004.

HOLLAND, J. H. *Studying Complex Adaptive Systems*. Journal of Systems Science and Complexity, v. 19, p. 1-8, 2006.

JAIN, R.; MESO, P. *Theory of Complex Adaptive Systems and Agile software development*. In: Proceedings of the Tenth Americas Conference on Information Systems, New York, NY, Estados Unidos: AMCIS, ago. 2004. Disponível em: <<http://aisel.aisnet.org/amcis2004/197>> . Acesso em: 3 maio 2016.

JANIS, I. L. *Groupthink: psychological studies of policy decisions and fiascoes*. 2. ed. Boston: Houghton Mifflin, 1982.

JEFFRIES, R.; ANDERSON, A.; HENDRICKSON, C. *Extreme Programming installed*. Massachusetts: Addison-Wesley, 2000.

KERTH, N. *Project retrospectives: a handbook for team reviews*. Nova Iorque: Dorset House, 2001.

LARMAN, C. *Agile and iterative development: a manager's guide*. Massachusetts: Addison-Wesley, 2003.

LIKER, J. K. *The Toyota way: 14 management principles from the world's greatest manufacturer*. Blacklist: McGraw-Hill Professional Publishing, 2003.

LOCKE, E. A. *Motivation through conscious goal setting*. Applied & Preventive Psychology, Amsterdã, Holanda, v. 5, p. 117-124, 1996.

LOCKE, E. A.; LATHAM, G. P. *New directions in goal-setting theory*. Current Directions in Psychological Science, Nova Iorque, NY, Estados Unidos, v. 15, n. 5, p. 265-268, out. 2006.

MANZ, C. C.; NECK, C. P. *Teamthink: beyond the groupthink syndrome in self-managing work teams*. Team Performance Management, v. 3, n. 1, p. 18-31, 1997.

MANZ, C. C.; SIMS, H. P. Jr. *Leading workers to lead themselves: the external leadership of self-managing work teams*. Administrative Science Quarterly, v. 32, n. 1, p. 106-129, mar. 1987.

MOORE, G. A. *Crossing the chasm: marketing and selling high-tech products to mainstream customers*. Nova Iorque: PerfectBound, 2001.

MOTTA, P. R. *Gestão contemporânea: a ciência e a arte de ser dirigente*. Rio de Janeiro: Record, 1991.

OGUNNAIKE, B. A.; RAY, W. H. *Process dynamics, modeling and control*. Nova Iorque: Oxford University Press, 1994.

OSONO, E.; SHIMIZU, N.; TAKEUCHI, H. *Relatório Toyota: contradições responsáveis pelo sucesso da maior montadora do mundo*. Tradução de Carlos Szlak. São Paulo: Ediouro, 2008.

PICHLER, R. *Agile product management with Scrum: creating products that customers love.* Boston: Addison-Wesley, 2010.

PIMENTEL, M. *Tenha nojo dos impedimentos.* InfoQ Brasil, dez. 2009. Disponível em: <<http://www.infoq.com.br/articles/tinha-nojo-impedimentos>> . Acesso em: 17 set. 2011.

ROYCE, W. *Managing the development of large software systems: concepts and techniques.* In: Proceedings of IEEE WESCON. Piscataway, NJ, Estados Unidos: IEEE Press, ago. 1970, p. 1-9.

SCHWABER, K. *Agile project management with Scrum.* Redmond: Microsoft Press, 2004.

SCHWABER, K.; BEEDLE, M. *Agile software development with Scrum.* Upper Saddle River: Prentice Hall, 2002.

SCHWABER, K.; SUTHERLAND, J. *The Scrum guide — the definitive guide to Scrum: the rules of the game.* Jul. 2013. Disponível em: <<http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf>> . Acesso em: 7 mai. 2016.

SCHWARZ, R. *The skilled facilitator: a comprehensive resource for consultants, facilitators, managers, trainers and coaches.* 2. ed. San Francisco: Jossey-Bass, 2002.

SCRUM ALLIANCE: *Core Scrum.* Agile Atlas, dez. 2012. Disponível em: <<http://agileatlas.org/atlas/scrum>> . Acesso em: 25 abr. 2013.

SNOWDEN, D. J.; BOONE, M. E. *A leader's framework for decision making.* Harvard Business Review, Boston, MA, Estados Unidos, v. 85, n. 11, p. 68-76, nov. 2007.

SOUZA, D.; DE TOLEDO, R.; OLIVEIRA, J. *Estudo sobre dependências e suas consequências no cenário Scrum and Scrum (SandS).* 3rd Brazilian Workshop on Agile Methods (WBMA'2012), set. 2012. Disponível em: <<http://www.ime.usp.br/char126kon/wbma2012.pdf>> . Acesso em: 25 abr. 2013.

STACEY, R. *Complexity and creativity in organizations.* San Francisco: Berrett-Koehler, 1996.

SUTHERLAND, J. *Agile contracts: Money for Nothing and Your Change for Free.* Scrum Inc., oct. 2008. Disponível em: <<https://www.scruminc.com/agile-contracts-money-for-nothing-and/>> . Acesso em: 14 mai. 2016.

SUTHERLAND, J. *Agile development: lessons learned from the first Scrum*. Cutter Agile Project Management Advisory Service: Executive Update, v. 5, n. 20, p. 1-4., 2004.

TAKEUCHI, H.; NONAKA, I. *The new new product development game*. Harvard Business Review, Boston, MA, Estados Unidos, v. 64, n. 1, p. 137-146, jan./fev. 1986.

THE STANDISH GROUP. *2015 CHAOS Report*. 2015. Disponível em: <<https://www.standishgroup.com/store/services/chaos-report-2015-blue-pm2go-membership.html>> . Acesso em: 04 mar. 2016.

DE TOLEDO, R. *Por que usar "story points"?* Blog Visão Ágil, jan. 2009. Disponível em: <<http://visaoagil.files.wordpress.com/2009/01/storypoints.pdf>> . Acesso em: 17 set. 2011.

VERSIONONE. *10th annual state of Agile survey*. 2016. Disponível em: <<http://versionone.com/pdf/VersionOne-10th-Annual-State-of-Agile-Report.pdf>> . Acesso em: 27 abr. 2016.

WAKE, W. *INVEST in good stories, and SMART tasks*. XP123, aug. 2003. Disponível em: <<http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/>> . Acesso em: 14 dez. 2012.

WILKINSON, M. *The secrets of facilitation: The S.M.A.R.T. Guide to Getting Results With Groups*. San Francisco: Jossey-Bass, 2004.

WOMACK, J. P.; JONES, D. T. *A mentalidade enxuta nas empresas: elimine o desperdício e crie riqueza*. Tradução de Ana Beatriz Rodrigues e Priscilla Martins Celeste. 4. ed. Rio de Janeiro: Campus, 1998.

WOMACK, J. P.; JONES, D. T.; ROOS, D. *A máquina que mudou o mundo*. Tradução de Ivo Korytowski. 17. ed. Rio de Janeiro: Campus, 1992.