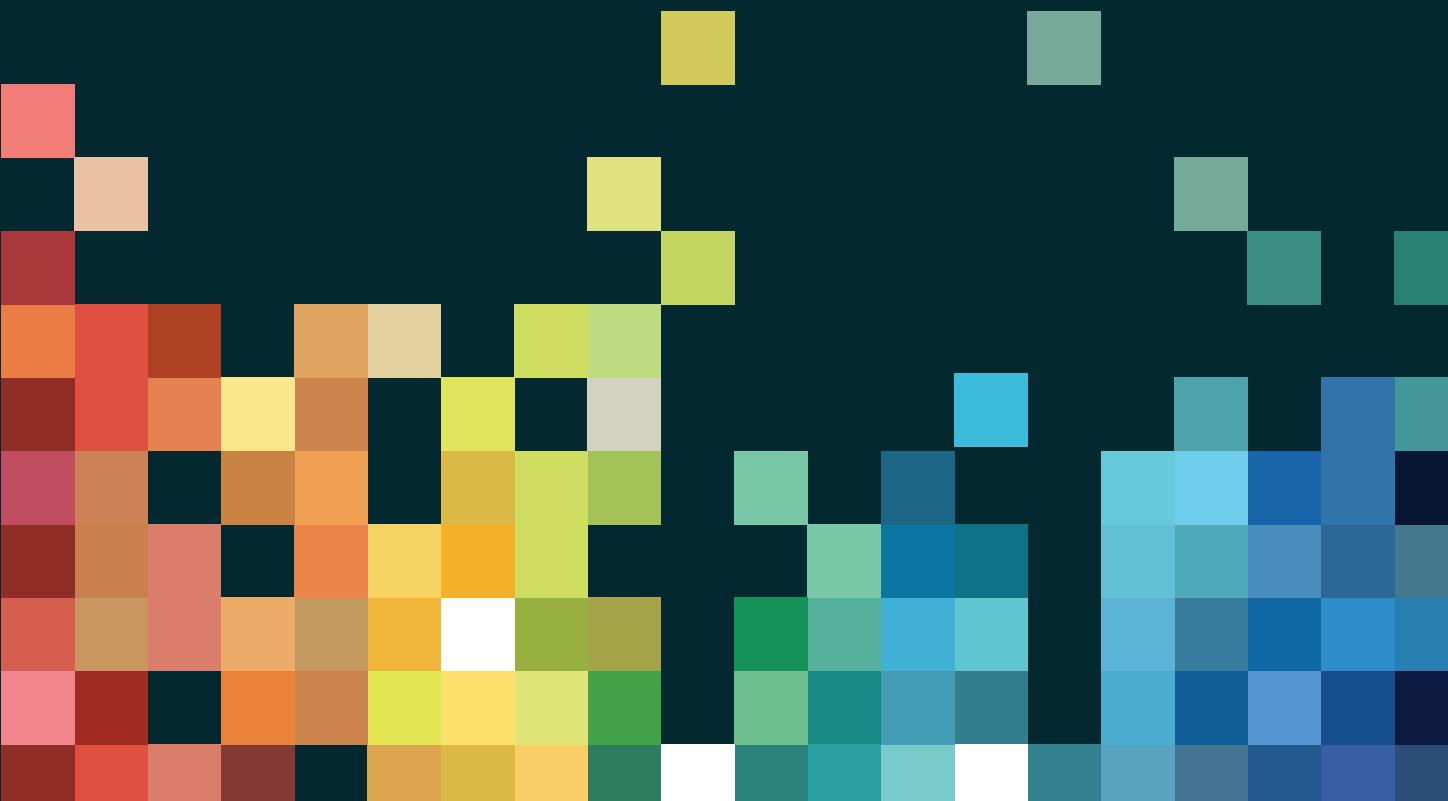


David J. Anderson

KANBAN

Mudança Evolucionária de Sucesso para
Seu Negócio de Tecnologia

Prefácio de Donald G. Reinersten



Elogios para *Kanban*

O trabalho de David com Sistemas Kanban tem uma influência significante em como eu abordo desenvolvimento de software e mudou a maneira como eu penso em processos. Em vez de visualizar trabalho em termos de estórias, pontos, e timeboxes, eu agora enxergo WIP, fluxo, e cadências. Este livro é um marco por trazer esta perspectiva para um público maior, e é leitura obrigatória para qualquer pessoa a procura de maneiras de criar organizações de desenvolvimento bem sucedidas e sustentáveis.

—Karl Scotland
Consultor Prático Sênior, EMC Consulting

Kanban é um assunto complicado para se escrever, visto que a implementação de qualquer pessoa será adaptada para o seu fluxo de trabalho e gargalos específicos, mas David consegue fornecer um framework teórico sólido enquanto mantém uma aderência estrita aos resultados práticos do mundo real.

—Chris Simmons
Gerente de Desenvolvimento, Sophos

O livro *Kanban* de David Anderson vai além do nível introdutório de como kanban conduz mudança, e fornece uma explicação clara dos aspectos práticos, através de exemplos ricos e dicas práticas. Kanban para o trabalho de conhecimento suporta poderosamente a tendência emergente de autonomia no espaço de trabalho, um dos desenvolvimentos de gerenciamento mais interessante da nossa equipe.

—Christina Skaskiw
Consultora Ágil

A melhor metodologia de mudança para software que vi nos últimos dez anos.

—David A. Bulkin
Vice Presidente, Lithespeed, LLC

Kanban

*Mudança Evolucionária de Sucesso
para Seu Negócio de Tecnologia*

David J. Anderson



Sequim, Washington



Blue Hole Press
72 Buckhorn Road
Sequim, WA 98382
www.blueholepress.com

Copyright © 2011 por David J. Anderson

Todos os direitos reservados. Nenhuma parte desta publicação pode ser reproduzida ou transmitida em qualquer forma ou por qualquer meio, eletrônico ou mecânico, incluindo cópia, reprodução, ou qualquer sistema de armazenamento ou recuperação de informação, sem a permissão prévia por escrito do editor.

Impresso nos Estados Unidos da América ou no Reino Unido

*Aplicado para
Library of Congress Cataloging-in-Publication Data; disponível em www.loc.gov*

ISBN: 978-0-9845214-6-3

Projeto gráfico da capa copyright © 2011 by Diego Mocotó
Projeto da capa e do interior by Vicki L. Rowland
Fotos na página 12 utilizadas com permissão, Thomas Blomseth

Tradução Andrea Pinto
10 9 8 7 6 5 4 3 2 1

❖ Índice Analítico ❖

Prefácio _____ **xiii**

PARTE UM ❖ INTRODUÇÃO

CAPÍTULO 1	
Resolvendo um dilema do Gerente Ágil	1
Minha Busca pelo Ritmo Sustentável	2
Minha Busca pela Gestão de Mudança Bem Sucedida	3
Do <i>Drum-Buffer-Rope</i> para o Kanban	6
Surgimento do Método Kanban	7
Adoção do Kanban pela Comunidade	8
O Valor do Kanban é Contra-intuitivo	8

CAPÍTULO 2	
O que é o Método Kanban?	11

O que é um Sistema Kanban?	13
Kanban Aplicado ao Desenvolvimento de Software	13
Por que Usar um Sistema Kanban?	15
Kanban em Ação	16
Um Modelo para o Método Kanban	17
Kanban como um Sistema Complexo Adaptável para <i>Lean</i>	17
Comportamento Emergente com Kanban	18
Kanban como um Doador de Permissões	19

PARTE DOIS ❖ BENEFÍCIOS DO KANBAN

CAPÍTULO 3	
Uma Receita para o Sucesso	25
Implementando a Receita	26
Foco na Qualidade	27

Reducir Trabalho-em-Progresso e Entregar Frequentemente _____	29
<i>WIP, Lead Time, e Defeitos</i> _____	30
Quem é melhor? _____	33
Entregas frequentes aumentam a confiança _____	34
Conhecimento implícito _____	35
Equilibrando a demanda e o rendimento_____	35
Crie folga _____	36
Priorize _____	37
Influência _____	37
Construindo Maturidade _____	38
Ataque as Fontes de Variabilidade para Melhorar Previsibilidade _____	38
Receita para o Sucesso e Kanban _____	39

CAPÍTULO 4

Do Pior ao Melhor em Cinco Trimestres _____	41
O Problema _____	42
Visualizar o Fluxo de Trabalho _____	43
Fatores Influenciando o Desempenho _____	43
Torne Explícitas as Políticas do Processo _____	45
Estimativa Era um Desperdício _____	45
Limitar Trabalho em Progresso (<i>WIP</i>)_____	46
Estabelecendo uma Cadênciaria na Entrada _____	47
Fazendo um Novo Acordo _____	48
Implementando Mudanças _____	48
Ajustando as Políticas _____	49
Procurando mais Melhorias _____	50
Resultados _____	51

CAPÍTULO 5

Uma Cultura de Melhoria Contínua _____	55
Cultura <i>Kaizen</i> _____	56
Kanban Acelera Maturidade e Capacidade Organizacional_____	57
Mudança Sociológica _____	62
Propagação Viral da Colaboração _____	63
Mudança Cultural Talvez Seja o Maior Benefício do Kanban _____	66

PARTE TRÊS ♦ IMPLEMENTANDO KANBAN**CAPÍTULO 6**

Mapeando a Cadeia de Valor	71
Definindo um Ponto Inicial e Final de Controle	72
Tipos de Item de Trabalho	72
Desenhando uma Parede de Cartões	73
Análise da Demanda	77
Anatomia de um Cartão de Item de Trabalho	79
Acompanhamento Eletrônico	80
Atribuindo Limites de Entrada e Saída	82
Lidando com Concorrência	83
Lidando com atividades Desordenadas	85

CAPÍTULO 7

Coordenação com Sistemas Kanban	87
Controle Visual e Puxado	87
Acompanhamento Eletrônico	89
Reuniões <i>Standup</i> Diárias	90
O Após a Reunião	92
Reuniões para Reabastecimento de Fila	92
Reuniões de Planejamento de <i>Release</i>	93
Triagem	94
Revisão e Escalação de Log de Problemas	96
Avatar Amigo (<i>Sticky Buddies</i>)	97
Sincronização através de Localizações Geográficas	97

CAPÍTULO 8

Estabelecendo uma Cadênci a de Entrega	101
Coordenação dos Custos de Entrega	103
Custos de Transação de Entrega	104
Eficiência de Entrega	106
Quanto de eficiência é o bastante?	107
Melhore a Eficiência para Aumentar a Cadênci a de Entrega	108
Fazendo Entregas Sob Demanda ou <i>Ad Hoc</i>	108

CAPÍTULO 9**Estabelecendo uma Cadênciа de Entrada** **113**

Coordinando Custos de Priorização	113
Acordando uma Cadênciа de Priorização	116
Eficiéncia de Priorização	116
Custos de Transação de Priorização	117
Melhore Eficiéncia para Aumentar a Cadênciа de Priorização	118
Fazendo Priorização <i>Ad Hoc</i> ou Por Demanda	119

CAPÍTULO 10**Estabelecedo Limites para o Trabalho-em-Progresso** **123**

Limites para Tarefas	124
Limites para Filas	125
Retenha Gargalos	126
Tamanho da Fila de Entrada	126
Seções Ilimitadas de Trabalho	128
Não Estresse a Sua Organização	129
Não Estabelecer um Limite de <i>WIP</i> é um Erro	130
Alocação de Capacidade	131

CAPÍTULO 11**Estabelecedo Acordo de Nível de Serviço** **133**

Definições Típicas de Classe de Serviço	134
Expedição	135
Data Fixa de Entrega	136
Classe Padrão	138
Classe Intangível	138
Políticas para Classe de Serviço	139
Políticas de Expedição	140
Políticas de Data de Entrega Fixa	140
Políticas de Classe Padrão	141
Classe Intangível	142
Determinando Meta para Entrega de Serviço	142
Estabelecedo uma Classe de Serviço	144
Colocando Classes de Serviço em Uso	145
Aloque Capacidade para Classes de Serviço	146

CAPÍTULO 12	
Métricas e Relatórios de Gerenciamento	149
Monitorando <i>WIP</i>	150
<i>Lead Time</i>	151
Desempenho de Entrega na Data	152
Rendimento	153
Problemas e Itens de Trabalho Bloqueados	154
Eficiência do Fluxo	155
Qualidade Inicial	156
Carga de Falhas	156
CAPÍTULO 13	
Escalando Kanban	159
Requisitos Hierárquicos	160
Dissocie Entrega de Valor de Variabilidade do Item de Trabalho	161
Paredes de Cartão em Duas Camadas	163
Introduzindo Raias	164
Abordagem Alternativa para Variabilidade no Tamanho	165
Incorporando Classes de Serviço	165
Integração de Sistemas	166
Gerenciando Recursos Compartilhados	166
CAPÍTULO 14	
Revisão Operacional	169
Antes da Reunião	169
Defina um Tom de Negócio desde o Início	170
Convide os Participantes a Sair da Plateia e Agregar Valor	171
Agenda Principal	171
Pedra Angular para uma Transição <i>Lean</i>	172
CadênciA Apropriada	173
Demonstrando o Valor dos Gerentes	174
Foco Organizacional Fomenta <i>Kaizen</i>	174
Um Exemplo Anterior	174

CAPÍTULO 15**Empreendendo uma Iniciativa de Mudança Kanban** **179**

Mudança Cultural em vez de uma Iniciativa de Mudança Gerenciada	179
O Primeiro Objetivo para Nossa Sistema Kanban	180
Objetivos Secundários para Nossa Sistema Kanban	181
Conhecer os Objetivos e Articular os Benefícios	185
Passos para Dar Início	186
Kanban Requer um Tipo Diferente de Negociação	188
Fazendo uma Negociação Kanban	190
Limites de <i>WIP</i>	190
Priorização	192
Entrega/ <i>Release</i>	192
<i>Lead Time</i> e Classes de Serviço	194

PARTE QUATRO ❖ FAZENDO MELHORIAS**CAPÍTULO 16****Três Tipos de Oportunidade de Melhoria** **199**

Gargalos, Eliminação de Desperdício, e Redução da Variabilidade	200
Teoria das Restrições	200
Five Focusing Steps	201
<i>Lean</i> , TPS, e Redução de Desperdício	202
Deming e Six Sigma	203
Adequar Kanban para a Cultura da Sua Empresa	204

CAPÍTULO 17**Gargalos e Disponibilidade Não-Instantânea** **207**

Recursos com Capacidade Limitada	209
Ações de Elevação	209
Ações de Exploração/Proteção	210
Ações de Subordinação	213
Recursos Disponíveis Não - Instantaneamente	214
Ações de Exploração/Proteção	217
Ações de Subordinação	218
Ações de Elevação	219

CAPÍTULO 18	
Um Modelo Econômico para <i>Lean</i>	223
Redefinindo “Desperdício”	223
Custos de Transação	224
Custos de Coordenação	226
Como você Sabe se uma Atividade é um Custo?	228
Carga Defeituosa	229
CAPÍTULO 19	
Fontes de Variabilidade	231
Fontes Internas de Variabilidade	233
Tamanho do Item de Trabalho	233
Mesclas de Tipos de Item de Trabalho	234
Mesclas de Classes-de-Serviço	235
Fluxo Irregular	236
<i>Rework</i>	237
Fontes Externas de Variabilidade	237
Ambiguidade de Requisitos	238
Requisições de Expedição	239
Fluxo Irregular	240
Disponibilidade do Ambiente	241
Outros Fatores de Mercado	242
Dificuldades em Agendar Atividades de Coordenação	243
CAPÍTULO 20	
Gerenciamento de Problemas e Políticas de Escalação	247
Escalando Problemas	249
Acompanhando e Reportando Problemas	250
Notas	253
Agradecimentos	257
Índice	261
Sobre o Autor	273
Recursos Adicionais	274

❖ Para Nicola and Natalie ❖

❖ Prefácio ❖

Eu sempre observo o trabalho de David Anderson. Meu primeiro contato com ele foi em outubro de 2003, quando ele me enviou um exemplar do seu livro, *Agile Management for Software Engineering: Applying Theory of Constraints for Business Results*. Como o título indica, esse livro foi altamente influenciado pela Teoria das Restrições (TOC*) de Eli Goldratt. Depois, em março de 2005, eu o visitei na Microsoft; nessa época ele estava fazendo um trabalho impressionante com diagramas de fluxo cumulativo. Mais tarde, em abril de 2007, eu tive a chance de observar o inovador sistema kanban que ele implementou na Corbis.

Certamente, velocidade é mais útil caso se esteja na direção correta; eu tenho confiança de que David está na direção certa. Eu estou particularmente animado com esta última obra sobre sistemas kanban. Eu sempre achei as ideias de produção enxuta mais úteis no desenvolvimento de produtos do que as de TOC. De fato, em outubro de 2003 eu escrevi para David dizendo “Uma das fragilidades de TOC é a sua falta de ênfase na importância do tamanho dos lotes. Se a sua primeira prioridade é encontrar e reduzir a restrição você está muitas vezes resolvendo o problema errado”. Eu ainda acredito que isso seja verdade.

Em nosso encontro de 2005 eu novamente encorajei David a olhar além do foco em gargalos de TOC. Eu expliquei a ele que o dramático sucesso do Sistema de Produção Toyota (TPS) não tinha nada a ver com encontrar e eliminar gargalos. O desempenho dos ganhos da Toyota veio a partir do uso da redução do tamanho do lote e da variabilidade para reduzir o estoque de trabalho-em-progresso. Foi a redução de estoque que proporcionou os benefícios econômicos e

* N. de T.: Sigla de *Theory of Constraints*.

foram esses sistemas de restrição de trabalho-em-progresso como o kanban que tornaram isso possível.

Em nosso encontro de 2005 eu novamente encorajei David a olhar além do foco em gargalos de TOC. Eu expliquei a ele que o dramático sucesso do Sistema de Produção Toyota (TPS*) não tinha nada a ver com encontrar e eliminar gargalos. O desempenho dos ganhos da Toyota veio a partir do uso da redução do tamanho do lote e da variabilidade para reduzir o estoque de trabalho-em-progresso. Foi a redução de estoque que proporcionou os benefícios econômicos e foram esses sistemas de restrição de trabalho-em-progresso como o kanban que tornaram isso possível.

Na época em que visitei a Corbis em 2007 eu vi uma impressionante implementação de um sistema kanban. Eu disse a David que ele tinha progredido muito além da abordagem *kanban* usada pela Toyota. Por que eu disse isso? O Sistema de Produção Toyota é elegantemente aperfeiçoado para lidar com tarefas repetitivas e previsíveis: tarefas com duração e custos de atraso homogêneos. Sob certas condições é correto usar abordagens como a priorização *first-in-first-out (FIFO)*[†]. Também é correto bloquear a entrada de trabalho quando o limite de *WIP*[‡] é atingido. No entanto, essas abordagens não são ótimas quando devemos lidar com atividades não-repetitivas e imprevisíveis; com diferentes custos de atraso; e durações diferentes — exatamente com o que devemos lidar no desenvolvimento de produtos. Precisamos de sistemas mais avançados, e este livro é o primeiro a descrever esses sistemas em detalhes práticos.

Eu gostaria de oferecer aos leitores alguns breves avisos. Primeiro, se você acha que realmente entende como sistemas kanban funcionam provavelmente você está pensando em sistemas kanban usados em produção enxuta. As ideias neste livro vão muito além de tais sistemas simples que usam limites estáticos de WIP, escalonamento FIFO e uma única classe de serviço. Preste muita atenção a essas diferenças.

Segundo, não enxergue essa abordagem apenas como um sistema de controle visual. O modo como quadros kanban deixam o *WIP* visível é surpreendente, mas é apenas um pequeno aspecto dessa abordagem. Se você ler esse livro cuidadosamente você vai encontrar muito mais. Os *insights* reais ficam em aspectos como o de *design*[§] de processos de chegada e partida, a gestão de recursos não-substituíveis e o uso de classes de serviço. Não perca as sutilezas se distraindo com a parte visual.

* N. de T.: Sigla de *Toyota Production System*.

† N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: primeiro a entrar, primeiro a sair.

‡ N. de T.: Sigla de *work-in-progress*.

§ N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: projeto.

Terceiro, não dê menos valor a esses métodos pelo fato deles parecerem fáceis de usar. Essa facilidade de uso é um resultado direto do *insight*[¶] de David sobre o que produz o máximo de benefício com o mínimo de esforço. Ele está bem ciente das necessidades dos praticantes e prestou muita atenção ao que realmente funciona. Métodos simples criam menor ruptura e quase sempre produzem os benefícios mais amplamente sustentados.

Este é um emocionante e importante livro que merece leitura atenta. O que você obterá dele dependerá de quanto seriamente você o lê. Nenhum outro livro lhe dará melhor exposição sobre essas avançadas ideias. Eu espero que você o aprecie tanto quanto eu o apreciei.

Don Reinertsen,
7 de fevereiro de 2010
Redondo Beach, California
Autor de *The Principles of Product Development Flow*

¶ N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: compreensão, visão.

❖ CAPÍTULO 1 ❖

Resolvendo um dilema do Gerente Ágil

Em 2002, eu era um gerente de desenvolvimento preparado para a batalha em uma filial da divisão PCS da Motorola (telefones celulares) situada em Seattle, Washington. Meu departamento havia sido parte de uma empresa *startup* que a Motorola havia adquirido naquele mesmo ano. Nós desenvolvíamos software para servidores de rede para serviços de dados, como *download over-the-air** e gerenciamento de dispositivos *over-the-air*. Essas aplicações para servidor eram parte de sistemas integrados que funcionavam em conjunto com código cliente nos aparelhos de celular, bem como com outros elementos dentro das redes das operadoras de celular e infra-estrutura de retaguarda, como cobrança. Nossos prazos eram definidos pelos gerentes sem levar em conta complexidade de engenharia, riscos ou tamanho do projeto. Nossa base de código havia evoluído desde a empresa *startup* original, onde muitos custos haviam sido cortados. Um desenvolvedor sênior insistiu em referir-se ao nosso produto como “um protótipo”. Nós estávamos com uma necessidade desesperada de produtividade e qualidade maiores para suprir as demandas do negócio.

No meu trabalho, em 2002, e através do meu esforço de ter escrito meu livro anterior¹, dois novos desafios principais estavam na minha

* Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: pelo ar, através do ar.

mente. Primeiro, como eu poderia proteger minha equipe das incessantes demandas de negócio e alcançar o que a comunidade Ágil agora se refere como um “ritmo sustentável”? E segundo, como eu poderia obter sucesso na adoção em escala de uma abordagem Ágil por toda a empresa e superar a inevitável resistência à mudança?

Minha Busca pelo Ritmo Sustentável

Em 2002, a comunidade Ágil se referia à noção de um ritmo sustentável como simplesmente “a semana de 40 horas²”. Os Princípios Por Trás do Manifesto Ágil³ nos diziam que “Processos Ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente”. Dois anos antes, minha equipe na Sprint PCS se acostumou ao me ouvir dizer que “desenvolvimento de software de larga escala é uma maratona, não um *sprint*”. Se os membros da equipe mantivessem o ritmo para o longo curso em um projeto de 18 meses, nós não poderíamos permitir que eles ficasse esgotados depois de um mês ou dois. O projeto deveria ser planejado, orçado, programado e estimado de forma que os membros da equipe pudessem trabalhar um horário razoável a cada dia para evitar que eles ficasse cansados. O desafio pra mim como gerente foi atingir esse objetivo e acomodar todas as demandas do negócio.

Em meu primeiro emprego como gerente em 1991, em uma *startup*[†] de 5 anos de idade que fazia placas de captura de vídeo para PCs e outros computadores menores, o *feedback* do CEO foi de que a liderança me considerava como “muito negativo”. Eu estava sempre respondendo “Não” quando pediam por mais produtos ou funcionalidades quando nossa capacidade de desenvolvimento já estava esticada ao máximo. Em 2002, havia claramente um padrão: Eu passara mais de dez anos dizendo “Não”, lutando contra as incessantes e instáveis demandas de donos de empresas.

Em geral, equipes de engenharia de software e departamentos de IT parecem estar à mercê de outros grupos que negociam, persuadem, intimidam e rejeitam mesmo os planos mais defensáveis e objetivos. Mesmo planos baseados em profunda análise e apoiados por anos de dados históricos estavam vulneráveis. Muitas equipes, que nunca tiveram um método de análise profundo ou qualquer histórico de dados, estavam impotentes nas mãos de outros que as forçavam a se comprometer com entregáveis desconhecidos (e muitas vezes completamente irracionais).

* Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil.
Tradução: nesse contexto de corrida/maratona, significa corrida de velocidade em curta distância.

† Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil.
Tradução: inicialização.

Enquanto isso, a força de trabalho havia aceitado como norma cronogramas loucos e comprometimentos de trabalho absurdos. Aparentemente é suposto que engenheiros de software não possuem vida familiar ou social. Se isso parece abusivo, é porque é! Eu conheço muitas pessoas cujos comprometimentos com o trabalho causaram danos irreparáveis aos relacionamentos com seus filhos e com outros membros da família. É difícil ter compaixão pelo típico *geek* desenvolvedor de software. No estado onde moro, em Washington, nos Estados Unidos, engenheiros de software são a segunda profissão em renda anual, depois dos dentistas. Como os trabalhadores da linha de montagem da Ford na segunda década do século 20 — ganhando cinco vezes o salário médio dos EUA — ninguém se preocupou com a monotonia do trabalho ou com o bem-estar deles porque eram muito bem remunerados. É difícil imaginar o surgimento de sindicatos em áreas de trabalho do conhecimento como desenvolvimento de software, principalmente porque é difícil imaginar alguém buscando as causas de doenças físicas ou psicológicas que esses desenvolvedores apresentam rotineiramente. Empregadores mais ricos têm sido capazes de prover benefícios de saúde adicionais, como massagens e psicoterapia, e prover dias de folga ocasionais de “saúde mental”, ao invés de buscar as causas raiz do problema. Um redator técnico em uma bem conhecida empresa de software uma vez me fez um comentário, “Não há estigma sobre usar antidepressivos, todo mundo está tomando”! Em resposta a esse abuso, engenheiros de software tendem a sujeitar-se às demandas, receber seus extravagantes salários e sofrer as consequências.

Eu queria romper esse modelo. Eu queria encontrar uma abordagem “ganha-ganha” que me permitisse dizer “Sim” e ainda proteger a equipe promovendo um ritmo sustentável. Eu queria devolver à minha equipe — devolver a eles a vida social e familiar — e melhorar as condições que estavam causando problemas de saúde relacionados ao *stress* em desenvolvedores nos seus vinte e poucos anos. Então eu decidi tomar uma posição e tentar fazer alguma coisa em relação esses problemas.

Minha Busca pela Gestão de Mudança Bem Sucedida

A segunda coisa em minha mente era o desafio de liderar a mudança em grandes organizações. Eu havia sido um gerente de desenvolvimento na Sprint PCS e depois na Motorola. Em ambas as empresas, havia uma necessidade real de negócio de desenvolver um modo mais ágil de trabalhar. Mas em ambos os casos eu havia me esforçado para escalar métodos Ágeis além de uma ou duas equipes.

Eu não tinha uma posição de poder suficiente em nenhum dos casos para simplesmente impor a mudança em uma grande quantidade de equipes. Eu estava tentando influenciar a mudança a pedido da liderança sênior, mas sem qualquer poder. Foi pedido

a mim para influenciar colegas a fazerem em suas equipes mudanças similares às quais eu havia implementado com a minha própria equipe. As outras equipes resistiram à adoção das técnicas que estavam claramente produzindo melhores resultados com a minha equipe. Provavelmente havia muitas facetas para essa resistência, mas o tema mais comum era o de que a situação de qualquer das outras equipes era diferente; as técnicas da minha equipe teriam de ser modificadas e adaptadas às necessidades alheias. Na metade de 2002, eu havia concluído que aplicar prescritivamente um processo de desenvolvimento de software em uma equipe não funcionava.

Um processo precisa ser adaptado para cada situação específica. Fazer isso requereria liderança ativa em cada equipe. Isto estava muitas vezes em falta. Mesmo com a liderança certa, eu duvidava que mudanças significativas pudessem acontecer sem um *framework* em uso e sem orientação sobre como adaptar o processo para que se encaixe em diferentes situações. Sem isso para guiar o líder, *coach** ou engenheiro de processos, qualquer adaptação provavelmente seria imposta subjetivamente, baseada em crenças supersticiosas. Era muito provável que isso criaria polêmica e objeções da mesma forma que impor um *template*† de processo inadequado.

Eu tinha abordado parcialmente esse problema no livro que eu estava escrevendo naquele momento, *Agile Management for Software Engineering*. Eu estava perguntando, “Por que o desenvolvimento Ágil produz resultados econômicos melhores do que as abordagens tradicionais?”. Eu procurei usar o *framework* da Teoria das Restrições⁴ para o processo.

Enquanto pesquisava e escrevia o livro, eu percebi que de alguma forma, cada situação era única. Por que o fator limitante ou o gargalo deveria estar no mesmo lugar para toda equipe em todo projeto, todo o tempo? Cada equipe é diferente: diferente conjunto de habilidades, capacidades e experiência. Cada projeto é diferente: diferente orçamento, cronograma, escopo e perfil de risco. E cada organização é diferente: uma diferente cadeia de valor operando em um diferente mercado, como mostra a Figura 1.1. Ocorreu-me que isto poderia fornecer uma pista para a resistência à mudança. Se as propostas de mudanças às práticas de trabalho e comportamentos não tivessem um benefício percebido, as pessoas resistiriam a elas. Se essas mudanças não afetaram o que os membros da equipe percebem como sua restrição ou fator limitante, então eles resistirão. Ou seja, mudanças sugeridas fora de contexto serão rejeitadas pelos trabalhadores que viveram e compreenderam o contexto do projeto.

* Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil.
Tradução: instrutor, mentor, consultor.

† Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil.
Tradução: padrão.

Parecia melhor deixar que um novo processo evoluísse eliminando um gargalo por vez. Esse é a tese central da Teoria das Restrições de Goldratt. Enquanto reconhecia que tinha muito a aprender, eu senti que havia valor no material e continuei com o manuscrito originalmente planejado. Eu sabia muito bem que meu livro não fornecia recomendações sobre como implementar as ideias em escala, pois oferecia pouca ou nenhuma recomendação sobre gestão de mudança.

A abordagem de Goldratt, explicada no capítulo 16, procura identificar um gargalo e, em seguida, encontrar maneiras de aliviá-lo até que ele não restrinja o desempenho. Quando isso acontece, um novo gargalo emerge e o ciclo se repete. É uma abordagem iterativa para melhorar o desempenho sistematicamente, identificando e removendo gargalos.

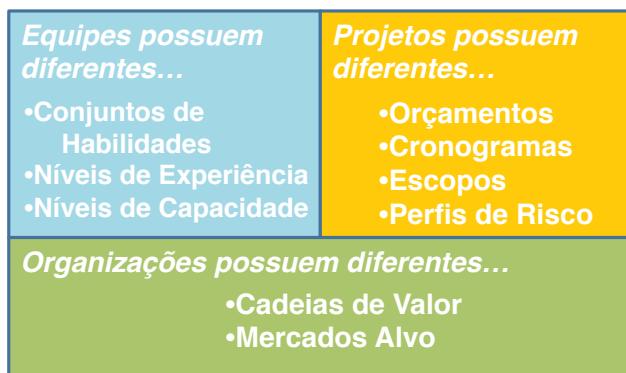


Figura 1.1 Porque metodologias “tamanho único” não funcionam

Eu percebi que eu poderia sintetizar essa técnica com algumas ideias de *Lean*[‡]. Modelando o fluxo de trabalho de um ciclo de vida de desenvolvimento de software como uma cadeia de valor e então criando um sistema de acompanhamento e visualização para acompanhar alterações de estado do trabalho emergente à medida ele “flui” através do sistema, eu poderia ver os gargalos. A capacidade de identificar um gargalo é o primeiro passo para o modelo por baixo da Teoria das Restrições. Goldratt já tinha desenvolvido uma aplicação da teoria para problemas de fluxo, estranhamente chamado de “*Drum-Buffer-Rope*[§]”. Independentemente da estranheza do nome, percebi que uma solução simplificada do *Drum-Buffer-Rope* poderia ser implementada para o desenvolvimento de software.

[‡] Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil.
Tradução: Leve.

[§] Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil.
Tradução: Tambor-Amortecedor-Corda.

Genericamente, *Drum-Buffer-Rope* é um exemplo de uma classe de soluções conhecidas como sistemas puxados. Como veremos no capítulo 2, um sistema kanban é outro exemplo de um sistema puxado. Um efeito colateral interessante de sistemas puxados é que eles limitam trabalhos-em-progresso (*WIP*) para uma quantidade acordada, impedindo assim que os trabalhadores fiquem sobrecarregados. Além disso, apenas os trabalhadores no gargalo permanecem totalmente ocupados; os outros devem ter alguma folga. Percebi que sistemas puxados potencialmente poderiam resolver ambos os meus desafios. Um sistema puxado me permitiria implementar a mudança do processo de forma incremental, com (eu esperava) resistência significativamente reduzida e ele iria facilitar um ritmo sustentável. Decidi estabelecer um sistema puxado *Drum-Buffer-Rope* na primeira oportunidade. Eu queria experimentar a evolução do processo incremental e ver se ele habilitaria ritmo sustentável e reduziria a resistência à mudança.

A oportunidade chegou no Outono de 2004 na Microsoft, e ela é totalmente documentada no estudo de caso no capítulo 4.

Do Drum-Buffer-Rope para o Kanban

Implementar uma solução de *Drum-Buffer-Rope* na Microsoft funcionou bem. Com muito pouca resistência, a produtividade mais que triplicou e os prazos de entrega encolheram em 90 por cento, enquanto que a previsibilidade melhorou em 98 por cento. No outono de 2005, eu informei os resultados em uma conferência em Barcelona⁵ e novamente no inverno de 2006. Meu trabalho chamou a atenção de Donald Reinertsen, que fez uma viagem especial para visitar-me no meu escritório em Redmond, Washington. Ele queria convencer-me que eu tinha todas as peças no lugar para implementar um sistema kanban completo.

Kan-ban é uma palavra japonesa que significa literalmente “cartão de sinal” em Português. Em um ambiente de produção, este cartão é usado como um sinal para informar a uma etapa anterior do processo para produzir mais. Aos trabalhadores em cada etapa do processo não é permitido trabalhar a menos que eles sejam sinalizados com um kanban de uma etapa posterior. Embora eu estivesse ciente desse mecanismo, eu não estava convencido de que era uma técnica útil ou viável para aplicação ao trabalho do conhecimento e engenharia de software, especificamente. Eu entendi que um sistema kanban permitiria um ritmo sustentável. No entanto, não estava ciente de sua reputação como um método para a condução de melhoria de processo incremental. Eu ignorava o que tinha dito Taiichi Ohno, um dos criadores do Sistema de Produção Toyota, “os dois pilares do Sistema de Produção Toyota são *just-in-time* e automação com um toque humano, ou autonomação. A ferramenta usada para operar o sistema é

kanban". Em outras palavras, kanban é fundamental para o processo de *kaizen* ("melhoria contínua") usado na Toyota. É o mecanismo que faz com que ele funcione. Eu cheguei a reconhecê-lo como uma verdade plena através de minhas experiências ao longo de cinco anos que se seguiram.

Felizmente, Don fez um argumento convincente de que eu deveria mudar de implementações de *Drum-Buffer-Rope* para um sistema kanban pela razão altamente esotérica de que um sistema kanban faz uma recuperação mais elegante de uma paralisação no gargalo do que o sistema de *Drum-Buffer-Rope*. Entender essa idiossincrasia, no entanto, não é importante para que você possa ser capaz de ler e compreender este livro.

Revisitando a solução final implementada na Microsoft, percebi que se a tivéssemos concebido como um sistema kanban desde o início, o resultado teria sido idêntico. Foi interessante para mim que duas abordagens diferentes resultariam no mesmo produto. Portanto, se o processo resultante foi o mesmo, não me senti compelido a pensar nele especificamente como uma implementação de *Drum-Buffer-Rope*.

Desenvolvi uma preferência pelo termo "kanban" em relação ao *Drum-Buffer-Rope*. Kanban é usado em produção *Lean* (ou o Sistema de Produção Toyota). Este corpo de conhecimento tem adoção e aceitação muito mais amplas do que a Teoria das Restrições. Kanban, mesmo sendo japonês, é menos metafórico que *Drum-Buffer-Rope*. Kanban era mais fácil de dizer, mais fácil de explicar e se tornou mais fácil de ensinar e implementar, então ele pegou.

Surgimento do Método Kanban

Em Setembro de 2006, saí da Microsoft para assumir o departamento de engenharia de software na Corbis, uma empresa de acervo fotográfico e de direitos de propriedade intelectual, com base no centro de Seattle. Encorajado pelos resultados da Microsoft, eu decidi implementar um sistema puxado kanban na Corbis. Novamente, os resultados foram encorajadores e levaram ao desenvolvimento da maior parte das ideias apresentadas neste livro. É este conjunto ampliado de ideias – visualização de fluxo de trabalho, tipos de item de trabalho, cadências, classes de serviço, relatórios de gestão específicos e revisão de operações — que define o método Kanban.

No restante deste livro descrevo o Kanban (com K maiúsculo) como o método de mudança evolutiva que utiliza um sistema puxado kanban (com k minúsculo), visualização e outras ferramentas para catalisar a introdução de ideias *Lean* no desenvolvimento de tecnologia e operações de TI. O processo é evolutivo e incremental. Kanban permite alcançar o aperfeiçoamento do processo específico ao contexto com

o mínimo de resistência à mudança, mantendo um ritmo sustentável para os trabalhadores envolvidos.

Adoção do Kanban pela Comunidade

Em maio de 2007, Rick Garber e eu apresentamos os primeiros resultados da Corbis na conferência *Lean New Product Development* em Chicago para um público de cerca de 55 pessoas. Mais tarde nesse verão, na conferência *Agile 2007*, em Washington, realizei uma sessão aberta para discutir sistemas kanban; cerca de 25 pessoas participaram. Dois dias mais tarde, um dos participantes, Arlo Belshee, deu uma palestra relâmpago, na qual ele compartilhou sua técnica de *Naked Planning*⁶. Verificou-se que outros também haviam implementado sistemas puxados. Um grupo de discussão do Yahoo! foi formado e cresceu rapidamente para 100 membros. No momento em que escrevo, ele tem mais de 1000 membros. Vários dos participantes da sessão aberta firmaram o compromisso de tentar Kanban em seu local de trabalho, muitas vezes com equipes que haviam se esforçado com Scrum. Os pioneiros mais notáveis foram Karl Scotland, Aaron Sanders e Joe Arnold, todos do Yahoo!, que rapidamente levaram o Kanban para mais de dez equipes em três continentes. Outro participante notável na sessão aberta foi Kenji Hiranabe, que havia desenvolvido soluções kanban no Japão. Logo depois ele escreveu dois artigos sobre o tema para InfoQ^{7,8} que despertaram muito interesse e atenção. No outono de 2007 Sanjiv Augustine, autor de *Managing Agile Projects*⁹ e um dos fundadores da *Agile Project Leadership Network* (APLN), visitou a Corbis em Seattle e descreveu nosso sistema kanban como o “primeiro novo método ágil que observei em cinco anos”.

No ano seguinte, no *Agile 2008*, em Toronto, houve seis apresentações sobre o uso das soluções kanban em diferentes situações. Uma delas, de Joshua Kerievsky, da Industrial Logic, uma empresa de consultoria e treinamento em *Extreme Programming*, mostrou como ele tinha evoluído ideias semelhantes para adaptar e melhorar o *Extreme Programming* para seu contexto de negócios. Nesse ano, a *Agile Alliance* entregou o prêmio Gordon Pask a Arlo Belshee e Kenji Hiranabe por suas contribuições para a comunidade Ágil. Ambos haviam feito uma visível contribuição para o surgimento do Kanban e produzido e comunicado ideias notavelmente semelhantes sobre o assunto.

O Valor do Kanban é Contra-intuitivo

De muitas maneiras, o trabalho do conhecimento é a antítese de uma atividade de produção repetitiva. Desenvolvimento de software certamente não é similar à

manufatura. Os domínios apresentam atributos totalmente diferentes. Manufatura tem pouca variabilidade, enquanto que grande parte do desenvolvimento de software é altamente variável e procura explorar a variabilidade através de novidades no projeto a fim de obter lucro. Software é por natureza “soft” e muitas vezes pode ser alterado de forma fácil e barata, enquanto que a manufatura tende a centralizar-se em coisas “hard” que são difíceis de mudar. É natural ser cético sobre o valor dos sistemas kanban no desenvolvimento de software e outros trabalhos em TI. Grande parte do que aprendemos sobre Kanban ao longo dos últimos anos, como uma comunidade, é contra-intuitivo. Ninguém previu o efeito na cultura corporativa ou a melhora na colaboração multifuncional que ocorreu na Corbis (que é descrita no capítulo 5). Nestas páginas, espero mostrar a você que “Kanban pode!”. Espero convencê-lo de que, empregando as regras simples do Kanban, você pode melhorar a produtividade, a previsibilidade e a satisfação do cliente, bem como reduzir os tempos de entrega. E com tudo isto, a cultura da sua organização irá mudar à medida que o aumento do trabalho colaborativo ajuda a estabelecer melhores e mais funcionais relações de trabalho em toda a organização.

Takeaways

- ❖ Sistemas Kanban são da família de abordagens conhecidas como sistemas *puxados*.
- ❖ A aplicação *Drum-Buffer-Rope* por Eliyahu Goldratt da Teoria das Restrições é uma implementação alternativa de um sistema puxado.
- ❖ A motivação para buscar uma abordagem de sistema puxado foi dupla: encontrar uma maneira sistemática de obter um ritmo sustentável de trabalho e encontrar uma abordagem para introduzir mudanças de processo que encontrariam resistência mínima.
- ❖ Kanban é o mecanismo que sustenta o Sistema de Produção Toyota e sua abordagem *kaizen* para melhoria contínua.
- ❖ O primeiro sistema kanban virtual para engenharia de software foi implementado na Microsoft, no início de 2004.
- ❖ Os resultados das primeiras implementações de Kanban foram encorajadores no que diz respeito a alcançar um ritmo sustentável, minimizando resistência à mudança através de uma abordagem evolutiva e incremental e produzindo benefícios econômicos significativos.
- ❖ O Método Kanban como uma abordagem para mudança começou a crescer em adoção pela comunidade depois da conferência Agile 2007 em Washington, D.C., em agosto de 2007.
- ❖ Ao longo deste texto, “kanban” (“k” minúsculo) refere-se aos cartões sinalizadores e “kanban sistema” (“k” minúsculo) refere-se ao sistema puxado implementado com cartões sinalizadores (virtuais).
- ❖ Kanban (“K” maiúsculo) é usado para se referir à metodologia de melhoria de processo incremental e evolutiva que surgiu na Corbis de 2006 a 2008 e continuou a evoluir na ampla comunidade de desenvolvimento *Lean* de software nos anos seguintes.

❖ CAPÍTULO 2 ❖

O que é o Método Kanban?

Na primavera de 2005, tive a sorte de ter um período de férias em Tóquio, no Japão, no início de Abril, durante a temporada da flor de cerejeira. Para aproveitar este espetáculo, fiz uma segunda visita aos Jardins do Oriente, no Palácio Imperial, no centro de Tóquio. Foi lá que tive uma revelação — kanban não era apenas para manufatura.

No sábado, 9 de abril de 2005, entrei no parque pela entrada norte, atravessando a ponte sobre o fosso próxima à estação de metrô de Takebashi. Muitos cidadãos de Tóquio estavam aproveitando a oportunidade em uma manhã ensolarada de sábado para desfrutar da tranquilidade do parque e da beleza da *sakura* (flor de cerejeira).

A prática de fazer um piquenique sob as cerejeiras, enquanto as flores caem em torno de você, é conhecida como *hanami* (festa da flor). É uma tradição antiga no Japão — uma oportunidade para refletir sobre a beleza, a fragilidade e o quanto curta é a vida. A vida breve da flor de cerejeira é uma metáfora para a nossa vida e nossa existência curta, bonita e frágil no meio a imensidão do universo.

A flor de cerejeira contrastava com os edifícios cinzentos do centro de Tóquio, sua agitação e pressa, multidões pulsantes de pessoas ocupadas e ruído do tráfego. Os jardins eram um oásis no coração da selva de concreto. Assim que eu cruzei a ponte com a minha família, um idoso cavalheiro japonês com uma mochila sobre seu ombro

aproximou-se de nós. Abrindo sua sacola, ele puxou um punhado de cartões plásticos. Ele ofereceu um para cada um de nós, pausando brevemente para decidir se minha filha de três meses presa ao meu peito necessitaria de um cartão. Ele decidiu que sim e entregou-me duas placas. Ele não disse nada e, como meu japonês é limitado, não ofereceu nenhuma conversa. Andamos pelos jardins para procurar um local para desfrutar de nosso piquenique em família.

Duas horas mais tarde, após uma agradável manhã ao sol, nós recolhemos nosso piquenique e fomos em direção à saída, no Portão Leste em Otemachi. Assim que chegamos, entramos em uma fila na frente de um pequeno quiosque. Bem à frente, vi pessoas devolvendo seus cartões plásticos de entrada. Eu procurei no meu bolso e encontrei os cartões que havia recebido. Chegando perto do quiosque, vi dentro uma senhora japonesa perfeitamente uniformizada. Entre nós havia uma tela de vidro com um buraco em semicírculo como um guichê, muito semelhante a um guichê de um cinema ou parque de diversões. Eu coloquei meus cartões plásticos no balcão através do buraco no vidro. A senhora pegou-os com suas mãos com luvas brancas e empilhou-os em um compartimento junto com os outros. Ela inclinou sua cabeça ligeiramente e agradeceu-me com um sorriso. Dinheiro algum foi entregue. Nenhuma explicação foi dada para porque eu estava carregando dois cartões de admissão de plástico branco desde a entrada do parque duas horas antes.

O que estava acontecendo com estes bilhetes de admissão? Por que se preocupar em emitir um bilhete, se nada foi cobrado? Meu primeiro palpite foi que ele devia ser um esquema de segurança. Contando-se todas as placas devolvidas, as autoridades poderiam assegurar-se de que nenhum visitante perdido permanecesse dentro do parque quando ele estivesse fechado ao fim do dia. Após uma rápida reflexão percebi que seria um sistema de segurança muito pobre. Quem poderia dizer que eu havia recebido dois cartões em vez de apenas um? Minha filha de três meses contaria como bagagem ou como um visitante? Parecia haver muita variabilidade no sistema. Oportunidades demais para erros! Se fosse um esquema de segurança, certamente iria falhar e produzir falsos positivos todos os dias. (Como uma breve observação, tal sistema não pode

produzir facilmente falsos negativos, pois ele exigiria a confecção de bilhetes de entrada adicionais. Este é um atributo comum útil dos sistemas kanban.) Dessa maneira, haveria tropas em torno dos arbustos todas as noites a procura de turistas perdidos. Não, deveria ser outra coisa. Percebi, então, que os jardins do Palácio Imperial estavam usando um sistema kanban!



Esta epifania extremamente esclarecedora me permitiu pensar além da fabricação no que diz respeito aos sistemas de kanban. Parecia provável que *tokens** kanban eram úteis para todos os tipos de situações de gestão.

O que é um Sistema Kanban?

Certo número de kanbans (ou cartões) equivalente à capacidade (em acordo) de um sistema é colocado em circulação. Um cartão é anexado a um trabalho. Cada cartão age como um mecanismo de sinalização. Um novo trabalho pode ser iniciado apenas quando um cartão está disponível. Este cartão livre é anexado a um trabalho e o segue à medida que ele flui através do sistema. Quando não há mais cartões livres, nenhum trabalho adicional pode ser iniciado. Qualquer novo trabalho deve esperar em uma fila até que um cartão esteja disponível. Quando algum trabalho for concluído, seu cartão é liberado e reciclado. Com um cartão agora livre, um novo trabalho da fila pode ser iniciado.

Este mecanismo é conhecido como um sistema puxado porque o novo trabalho é puxado para o sistema quando existe capacidade para lidar com ele, em vez de ser empurrado para o sistema com base na demanda. Um sistema puxado não pode ser sobrecarregado se a capacidade, conforme determinado pelo número de cartões sinalizadores em circulação, tiver sido configurada adequadamente.

Nos Jardins do Palácio Imperial, os próprios jardins são o sistema: os visitantes são os trabalhos em progresso, e a capacidade é limitada pelo número de cartões de admissão em circulação. Os visitantes recém chegados ganham cartões somente quando estão disponíveis. Em um dia normal isso nunca é um problema. No entanto, nos dias de maior movimento, como um feriado ou um sábado durante a temporada de flor de cerejeira, o Parque é popular. Quando todos os bilhetes de admissão são entregues, novos visitantes fazem fila na ponte e aguardam que cartões sejam reciclados assim que visitantes deixam o local. O sistema kanban fornece um método simples, barato e fácil de ser implementado para controlar o tamanho da multidão, limitando o número de pessoas dentro do Parque. Isso permite que os administradores do Parque mantenham os jardins em boas condições e evitem danos causados por muitos pedestres e superlotação.

Kanban Aplicado ao Desenvolvimento de Software

No desenvolvimento de software, estamos usando um sistema kanban virtual para limitar o trabalho-em-progresso. Embora “kanban” signifique “cartão sinalizador” e

* N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: sinalizações.

não existem cartões utilizados na maioria das implementações de Kanban no desenvolvimento de software, estes cartões não funcionam realmente como sinais para puxar mais trabalho. Em vez disso, eles representam itens de trabalho. Daí o termo “virtual”, porque não há nenhum cartão sinalizador físico. O sinal para puxar o novo trabalho é inferido da quantidade visual dos trabalhos-em-progresso subtraído de algum indicador do limite (ou capacidade). Alguns profissionais aplicaram kanban físico usando técnicas como clipes autocolantes ou slots físicos em uma placa. Mais frequentemente o sinal é gerado a partir de um sistema de acompanhamento de trabalho em software. O capítulo 6 fornece exemplos da mecânica de sistemas kanban que se aplicam ao trabalho de TI.

Paredes de cartões se tornaram um mecanismo de controle visual popular no desenvolvimento de software Ágil, como mostrado na Figura 2.1. Usando um quadro de aviso de cortiça com cartões de índice fixados em uma placa ou um quadro de comunicações com notas auto-adesivas para acompanhar os trabalhos em progresso (WIP) tornou-se comum. Vale a pena observar, nesta fase inicial, que apesar de alguns comentários da Comunidade em contrário, estes muros de cartão não são inherentemente



Figura 2.1 Um quadro kanban (cortesia da SEP)

sistemas kanban. Eles são sistemas de controle visual apenas. Eles permitem às equipes observarem visualmente os trabalhos em progresso e a se auto-organizarem, atribuindo suas próprias tarefas e movendo o trabalho de um *backlog** para conclusão sem orientação de um gerente de projeto ou linha. No entanto, se não houver nenhum limite explícito para trabalho-em-progresso e uma sinalização para puxar o novo trabalho através do sistema, ele não é um sistema kanban. Isto é mais explicado no capítulo 7.

Por que Usar um Sistema Kanban?

Como deve se tornar evidente nos próximos capítulos, nós usamos um sistema kanban para limitar o trabalho-em-progresso de uma equipe para definir a capacidade e equilibrar a demanda sobre a equipe em relação ao rendimento do trabalho entregue. Fazendo isso, podemos conseguir um ritmo sustentável de desenvolvimento para que todos os indivíduos possam alcançar um equilíbrio entre trabalho e vida pessoal. Como você verá Kanban também elimina rapidamente os problemas que prejudicam o desempenho e desafia uma equipe a se concentrar em resolver esses problemas para manter um fluxo constante de trabalho. Ao fornecer visibilidade para problemas de qualidade e de processos, torna óbvio o impacto de defeitos, gargalos, variabilidade e custos econômicos no fluxo e na vazão. O simples ato de limitar o trabalho-em-progresso com o kanban incentiva maior qualidade e maior desempenho. A combinação de fluxo aperfeiçoado e melhor qualidade ajuda a reduzir os prazos de entrega e a melhorar o desempenho da data de entrega e a previsibilidade. Ao estabelecer uma cadência regular de liberação e entregando consistentemente, Kanban ajuda a construir a confiança com os clientes e confiança ao longo da cadeia de valor com outros departamentos, fornecedores e parceiros.

Ao fazer tudo isso, Kanban contribui para a evolução cultural das organizações. Ao expor problemas, fazendo com que uma organização dê enfoque em resolvê-los e eliminando seus efeitos no futuro, Kanban facilita o surgimento de uma organização altamente colaborativa, de alta confiança, altamente habilitada, e em constante melhoria.

Kanban tem mostrado melhorar a satisfação do cliente através de liberações regulares, confiáveis e de alta qualidade de software de alto valor. Ele também tem mostrado aumentar a produtividade, qualidade e prazos de entrega. Além disso, há evidências de que kanban é um catalisador fundamental para o surgimento de uma organização mais ágil através de mudança cultural evolutiva.

* N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: itens de trabalho em espera.

O restante deste livro é dedicado a ajudá-lo a entender como usar sistemas kanban no desenvolvimento de software e para ensiná-lo a implementar um sistema desse tipo para obter esses benefícios com a sua equipe.

Kanban em Ação

Na edição original deste livro em Inglês, o desenho na Figura 2.2 aparece na capa do livro. Este desenho foi encomendado especialmente para o livro e ilustra o Método Kanban em ação.

No desenho podemos ver uma pequena equipe de trabalhadores do conhecimento numa reunião diária em frente a um quadro branco. O quadro mostra o fluxo de trabalho e o trabalho-em-progresso atual da equipe. É evidente que a atividade de testes está “faminta”. Enquanto isso a atividade de análise está bastante carregada com trabalho incompleto. A discussão entre os membros da equipe realça isto. Enquanto um membro da equipe está ocioso, outro está completamente sobreacarregado e outro está parado esperando pela resolução de um item de bloqueio. O sistema kanban com limite de WIP está certamente estimulando esta conversa. Com limites restritos de trabalho-em-progresso, os testadores ficarão ociosos se a atividade de desenvolvimento não for finalizada em tempo hábil.

Um quarto membro da equipe está falando e mostra uma postura de liderança. O sistema kanban está realçando um problema; membros da equipe estão discutindo o problema; e um quarto membro os está encorajando a “fazer algo em relação a isso.”

O desenho mostra vários aspectos do Método Kanban em ação. O fluxo de trabalho da equipe está visível e um sistema kanban com limite de trabalho-em-progresso; isto serve para realçar problemas e provocar discussões. Combinado com uma ação de liderança, e um entendimento de problemas de processo que ocorrem comumente, a equipe pode discutir os problemas abertamente e sugerir melhorias no processo.

Como você aprenderá na parte 4, a equipe pode utilizar um conhecimento de modelos teóricos para avaliar problemas no processo e sugerir melhoria, isto demonstra uso de método científico. Modelos permitem antecipar os resultados de qualquer mudança que seja implementada. Os resultados podem ser observados durante dias e semanas e comparados com as expectativas.

A parte 4 explica modelos baseados em *Lean*, Teoria das Restrições e o *System of Profound Knowledge*. Quando a equipe tem conhecimento nesses modelos, junto a uma evidência visual, ela negocia mudanças no processo e forma um consenso em relação ao que deve ser feito.

De maneira geral a Figura 2.2 retrata a essência do Método Kanban. Ela ilustra uma cultura *kaizen* - uma organização focada na melhoria contínua, trabalhando colaborativamente para melhorar seu desempenho e capacidade.



Figura 2.2 Essência do método Kanban

Um Modelo para o Método Kanban

A fim de documentar o método Kanban em um livro, acredito que é necessário descrever formalmente o modelo. Como Kanban foi surgindo da prática e evoluindo ao longo dos últimos quatro anos, eu nunca havia pensado em fazer essa definição — até agora. Preocupa-me que pode se tornar estático e pessoas se tornarão dogmáticas em seu pensamento, citando o modelo como a verdadeira fonte para a excelência do processo kanban ou usá-lo como um teste para “Estamos praticando Kanban?”. Espero que quando eu tentar definir Kanban, eu também consiga encorajar você a abrir a sua mente para a ideia de que nosso aprendizado continuará a evoluir, e ao invés de citar esta seção deste livro, você vai procurar uma definição contemporânea na *web* na *Limited WIP Society* (www.limitedwipsociety.org).

Kanban como um Sistema Complexo Adaptável para Lean

O método Kanban introduz um sistema complexo adaptável cujo objetivo é catalisar um resultado Lean dentro de uma organização. Sistemas complexos adaptáveis têm condições iniciais e regras simples que são necessárias para semear comportamento emergente complexo e adaptável. Neste caso, o resultado desejado é uma Cultura

Kaizen - uma organização focada em melhoria contínua que leva tanto a um melhor resultado econômico para o negócio, como a um melhor resultado sociológico para os funcionários. Kanban usa cinco propriedades fundamentais como condições iniciais para estimular um conjunto emergente de comportamentos Lean. Em organizações que desejam como resultado uma cultura de melhoria contínua, eu tenho observado que todas as cinco propriedades principais estão presentes:

1. Visualize o Fluxo de Trabalho
2. Limite Trabalho-em-Progresso
3. Meça e Gerencie o Fluxo
4. Torne as Políticas do Processo Explícitas
5. Use Modelos^{*} para Reconhecer Oportunidades de Melhoria

Com estas cinco propriedades em ação, os atos de liderança como o apresentado na Figura 2.2 catalisam mudanças de processo e levam ao surgimento de novos comportamentos que geram o resultado econômico e sociológico desejados.

Comportamento Emergente com Kanban

Há uma lista crescente de comportamentos emergentes que podemos esperar com implementações Kanban. Alguns ou todos eles têm surgido na maioria das implementações recentes; todos eles surgiram na Corbis durante 2007. Esperamos que esta lista continue a crescer à medida que aprendemos mais sobre os efeitos do Kanban nas organizações.

1. Processo adaptado para cada projeto/cadeia de valor
2. Cadências Independentes (ou Desenvolvimento sem Iterações)
3. Trabalho Agendado por (oportunidade) Custo do Atraso
4. Valor otimizado com Classes de Serviço
5. Risco Gerenciado com Capacidade de Alocação

* Modelos comuns em uso com Kanban incluem a Teoria das Restrições, Pensamento Sistêmico (*System Thinking*), uma compreensão de variabilidade através dos ensinamentos de W. Edwards Deming e o conceito de “muda” (desperdício) do Sistema de Produção Toyota. Os modelos usados com Kanban evoluem continuamente e ideias de outros campos de conhecimento como sociologia, psicologia e gerenciamento de riscos estão surgindo em algumas implementações.

6. Tolerância para um Processo de Experimentação
7. Gerenciamento Quantitativo
8. Propagação Viral (de Kanban) por toda a Organização
9. Equipes pequenas small teams merged for more liquid labor pools

Kanban como um Doador de Permissões

Kanban não é uma metodologia de ciclo de vida para desenvolvimento de software ou uma abordagem de gerenciamento de projetos. Ele requer que algum processo já esteja em vigor de maneira que Kanban possa ser aplicado para alterar incrementalmente o processo base.

Esta abordagem evolutiva, promovendo mudança incremental, tem sido controversa na comunidade de desenvolvimento Ágil de software. É controversa porque ela sugere que as equipes não devem adotar um modelo de processo ou método definido. Uma indústria de serviços e ferramentas se desenvolveu em torno de um pequeno conjunto de práticas definidas em dois métodos populares de desenvolvimento Ágil. Agora com Kanban, indivíduos e equipes podem ser habilitados a desenvolver suas próprias soluções únicas de processo que desviam da necessidade de tais serviços e exigem um novo conjunto de ferramentas. Na verdade, Kanban incentivou uma nova onda de fornecedores de ferramentas rebeldes e ansiosos para deixar para trás as ferramentas de gerenciamento ágil de projetos existentes com algo mais visual e programável que pode ser facilmente adaptado para um fluxo de trabalho específico.

No início do desenvolvimento Ágil de software, os líderes da comunidade muitas vezes não entendiam por que seus métodos funcionavam. Nós falávamos sobre “ecossistemas¹⁰” e aconselhávamos aos implementadores que seguissem todas as práticas ou a solução provavelmente falharia. Nos últimos anos tem havido uma tendência negativa que estende esse pensamento. Algumas empresas têm publicado Modelos de Maturidade Ágil que visam avaliar a adoção de práticas. A comunidade de Scrum tem um teste baseado em prática muitas vezes referido como o “Nokia Test¹¹”. Estas avaliações baseadas em práticas são projetadas para induzir conformidade e negar necessidade de adaptação baseado em contexto. Kanban está dando ao mercado a permissão de ignorar esses esquemas de avaliação baseada em práticas. É encorajarativamente a diversidade.

Em 2007, várias pessoas visitaram meu escritório na Corbis para ver Kanban em ação. A pergunta comum de qualquer visitante associado com a comunidade de desenvolvimento Ágil de software poderia ser parafraseada como, “David, temos andado

pelo seu edifício e vimos sete quadros kanban. Cada um é diferente! Cada equipe está seguindo um processo diferente! Como você consegue lidar com essa complexidade?” Minha resposta era sempre um desdenhoso “É claro! A situação de cada equipe é diferente. Elas evoluem seu processo para encaixar em seu contexto”. Mas eu sabia que esses processos foram derivados dos mesmos princípios porque os membros da equipe compreenderam os princípios básicos, e, portanto, eram capazes de se adaptar ao serem transferidos de uma equipe para outra.

Como mais pessoas têm tentado Kanban, elas perceberam que ele ajuda a resolver os problemas que encontraram com a gestão de mudança em suas organizações. Kanban permitiu às suas equipes, projeto ou organização mostrar melhor agilidade. Reconhecemos que Kanban está dando permissão ao mercado para criar um processo sob medida e aperfeiçoadoo para um contexto específico. Kanban é dar às pessoas per-

missão para pensar por si mesmas. Ele está dando às pessoas permissão para serem diferentes: diferentes da equipe no mesmo andar, no outro andar, no edifício ao lado e

em uma empresa vizinha. Ele está dando às pessoas permissão para se afastar do livro-texto. O melhor de tudo, Kanban está fornecendo as ferramentas que nos permitem explicar (e justificar) porque é melhor ser diferente e porque uma escolha diferente é a escolha certa nesse contexto. Para enfatizar esta escolha, eu projetei uma camiseta para a *Limited WIP Society*, inspirado pelo pôster de Shepard Fairey para a campanha de Obama e exibindo o rosto de Taiichi Ohno, o criador do sistema kanban na Toyota. O slogan “Yes We Kanban” é projetado para enfatizar que você tem permissão. Você tem permissão para tentar Kanban. Você tem permissão para

modificar seu processo. Você tem permissão para ser diferente. Sua situação é única e você merece desenvolver uma definição de processo único, sob medida e aperfeiçoadoo para seu domínio, sua cadeia de valor, os riscos que você gerencia, as competências da sua equipe e as exigências dos seus clientes.



Takeaways

- ❖ Sistemas Kanban podem ser usados em qualquer situação em que há um desejo de limitar a quantidade de coisas dentro de um sistema.
- ❖ Os Jardins do Palácio Imperial em Tóquio usam um sistema kanban para controlar o tamanho da multidão dentro do parque.
- ❖ A quantidade de cartões de sinalização “kanban” em circulação limita o trabalho-em-progresso.
- ❖ Um novo trabalho é puxado para dentro do processo assim que um cartão de sinalização retorna no momento em que uma tarefa ou pedido tem seu trabalho em curso concluído.
- ❖ No trabalho de TI estamos (geralmente) usando um sistema de kanban virtual já que nenhum cartão físico de fato circula para definir o limite de trabalho-em-progresso.
- ❖ Paredes de cartão comuns no desenvolvimento ágil de software não são sistemas kanban.
- ❖ Sistemas kanban criam uma tensão positiva no ambiente de trabalho que força a discussão sobre os problemas.
- ❖ O Método Kanban (ou “Kanban” com K maiúsculo) utiliza um sistema kanban como catalisador de mudança.
- ❖ Kanban requer que as políticas do processo sejam explicitamente definidas.
- ❖ Kanban usa ferramentas de várias áreas do conhecimento para encorajar análise dos problemas e descoberta de soluções.
- ❖ Kanban possibilita a melhoria incremental do processo através da descoberta repetitiva dos problemas que afetam o desempenho do processo.
- ❖ Uma definição contemporânea do Método Kanban pode ser encontrada online no web site da *Limited WIP Society*, www.limitedwipsociety.org.
- ❖ Kanban está agindo como um doador de permissão na profissão de desenvolvimento de software, encorajando as equipes a elaborar soluções de processo específicas ao contexto em vez de seguir dogmaticamente uma definição de processo de ciclo de vida de desenvolvimento de software ou um *template*.

❖ PARTE DOIS ❖

BENEFÍCIOS DO KANBAN

❖ CAPÍTULO 3 ❖

Uma Receita para o Sucesso

Durante toda a década passada fui desafiado a responder a seguinte pergunta: Como um gerente, que medidas devem ser tomadas quando se herda uma equipe já existente, especialmente uma que não funciona de forma ágil, com habilidades diversas, e que talvez seja completamente disfuncional? Normalmente fui colocado em cargos de gestão como um agente de mudança, sendo desafiado a criar uma mudança positiva e progredir rapidamente, dentro de dois ou três meses.

Atuando como gerente em grandes organizações, eu nunca contratei o meu próprio time. Eu sempre fui convidado a liderar uma equipe existente e, com alterações mínimas de pessoal, criar uma revolução no desempenho da organização. Acredito que esta situação é muito mais comum do que aquela onde você começa contratando um time inteiro novo.

Eu gradualmente evoluí uma abordagem para gerenciar mudanças; ela é baseada em experiências, incluindo aprendizado através de falhas. As falhas resultaram das minhas tentativas de impor um processo e um fluxo de trabalho para a equipe. A verdade é que ordens gerenciais tendem a falhar. Quando pedi às equipes para mudar seu comportamento e usar um método ágil como *Feature Driven Development*, encontrei resistência. Sugerir que ninguém deveria ter receio, pois eu iria proporcionar-lhes a formação e a orientação necessárias. Eu tinha

a aquiescência na melhor das hipóteses, e não uma verdadeira, profunda, institucionalização da mudança. Pedir para as pessoas mudarem seu comportamento gera medo e baixa a auto-estima, pois esse pedido dá a entender que as habilidades existentes são claramente desvalorizadas.

Eu desenvolvi o que eu vim a chamar a minha Receita para o Sucesso para abordar estas questões. A Receita para o Sucesso apresenta diretrizes para um novo gerente liderar uma equipe existente. Seguir a receita permite melhora rápida com baixos níveis de resistência da equipe. Eu quero agradecer a influência direta de Donald Reinertsen para as duas primeiras e a última etapa da receita e a influência indireta de Eli Goldratt, cujos escritos sobre a *Theory of Constraints* e o seu *Five Focusing Steps* muito influenciaram as etapas quatro e cinco.

Os seis passos da receita são:

- Foco na qualidade
- Reduzir Trabalho-em-Progresso
- Entregar Frequentemente
- Equilibrar a Demanda e Rendimento
- Priorizar
- Atacar Fontes de Variabilidade para Melhorar a Previsibilidade

Implementando a Receita

A Receita é entregue na sequência de execução para um gerente de função técnica. O foco na qualidade vem em primeiro lugar, porque está sob a influência e o controle exclusivos de um gerente, como um gerente de desenvolvimento ou de teste, ou o supervisor do gerente, com um cargo como Diretor de Engenharia. À medida que a lista continua, é necessário cada vez menos controle e mais colaboração com outros grupos de diferentes níveis organizacionais, até a etapa Priorizar. Priorização é justamente o trabalho do setor de negócios, e não da organização de tecnologia, e por isso não deveria ser da competência do gerente técnico. Infelizmente, é comum a gestão de negócios abdicar dessa responsabilidade e deixar um responsável técnico priorizar o trabalho e depois culpá-lo por fazer más escolhas. Atacar Fontes de Variabilidade para Melhorar a Previsibilidade é a última etapa da lista, porque alguns tipos de variabilidade exigem mudanças comportamentais, a fim de reduzi-las. Pedir às pessoas para mudarem de comportamento é difícil! Assim, é melhor atacar a variabilidade por

último até que a situação melhore a partir da obtenção de algum sucesso com as etapas anteriores. Como veremos no capítulo 4, às vezes é necessário analisar as causas da variabilidade, a fim de permitir que alguns desses passos anteriores aconteçam. O truque é escolher as fontes de variabilidade que requerem pouca mudança comportamental que pode ser facilmente aceita.

Focar na qualidade é mais fácil porque é uma disciplina técnica que pode ser orientada pelo gerente funcional. As outras etapas são mais desafiadoras, por dependerem de acordo e colaboração de outras equipes, exigindo habilidades de articulação, negociação, psicologia, sociologia e inteligência emocional. Construir o consenso em torno da necessidade de Equilibrar a Demanda e o Rendimento é crucial. Resolver problemas de disfunção entre os papéis e as responsabilidades dos membros da equipe exige ainda maior capacidade de diplomacia e negociação. Faz sentido, então, ir atrás do que está diretamente sob seu controle e que você sabe que terá um efeito positivo tanto na sua equipe, como no desempenho do seu negócio.

Desenvolver um maior nível de confiança com outros times pode ajudar a lidar com assuntos mais difíceis. Construir e apresentar código de alta qualidade com poucos defeitos aumenta a confiança. Liberar código de alta qualidade com regularidade constrói ainda mais confiança. O aumento do nível de confiança possibilita que o gerente ganhe mais capital político. Isso permite um movimento para a próxima etapa da receita. Em última análise, a sua equipe vai ganhar respeito suficiente para que você seja capaz de influenciar os donos do produto, sua equipe de marketing, e patrocinadores do negócio para que mudem de comportamento e colaborem para priorizar o trabalho mais valioso a ser desenvolvimento.

Atacar as fontes de variabilidade para melhorar a previsibilidade é difícil. Não deve ser realizada até que a equipe já esteja trabalhando num nível mais maduro e tenha melhorado muito. Os quatro primeiros passos na receita terão um impacto significativo. Eles trarão sucesso para um novo gerente. No entanto, para realmente criar uma cultura de inovação e melhoria contínua, será necessário atacar as fontes de variabilidade no processo. Portanto, o passo final na receita é um crédito extra: É o passo que separa os verdadeiramente grandes líderes técnicos dos gestores meramente competentes.

Foco na Qualidade

O Manifesto Ágil não diz nada sobre qualidade, embora os princípios por detrás do Manifesto¹² falam sobre perfeição, havendo, dessa maneira, um foco implícito em qualidade. Então, se a qualidade não aparece no Manifesto, porque ela é a primeira etapa na minha receita para o sucesso? Simplificando, uma quantidade excessiva de defeitos é o

maior desperdício no desenvolvimento de software. Os números são impressionantes e mostram vários níveis de variação de magnitude. O relatório Capers Jones¹³ em 2000, durante a bolha *dot-com*, mediou a qualidade do software das equipes norte-americanas e o número variou de 6 defeitos por ponto de função para menos de 3 defeitos por 100 pontos de função, um intervalo de 200 para 1. O ponto médio é de aproximadamente 1 defeito por 0.6 a 1.0 pontos de função. Isto implica que é comum que as equipes gastem mais de 90 por cento do seu esforço corrigindo defeitos. Como prova direta disso, no final de 2007, Aaron Sanders, um dos primeiros defensores do Kanban, relatou, na lista de discussão Kanbandev Yahoo! que uma equipe com a qual estava trabalhando estava gastando mais de 90 por cento de sua capacidade disponível na correção de defeitos.

Incentivar a qualidade desde o início terá um grande impacto sobre a produtividade e o rendimento das equipes com altas taxas de defeitos. Uma melhoria de duas a quatro vezes no rendimento é um número razoável. Com equipes realmente ruins, uma melhoria de dez vezes pode ser possível, focando-se na qualidade.

O problema da melhoraria na qualidade de software é bem conhecido.

Tanto o desenvolvimento ágil como as abordagens tradicionais de qualidade têm o seu mérito. Elas devem ser usadas em combinação. Profissionais de teste devem testar. Usar testadores para encontrar defeitos impede que os defeitos escapem para o código em produção. Pedir para que os desenvolvedores escrevam testes unitários e automatize-os para a execução de testes de regressão automatizados também tem efeito surpreendente. Parece haver uma vantagem psicológica quando se pede aos desenvolvedores que escrevam os testes primeiro. O *Test Driven Development* (TDD) parece fornecer a vantagem de que a cobertura do teste seja mais completa. É importante ressaltar que times bem disciplinados que eu gerenciei e que escreveram testes após a codificação funcional, demonstraram liderança de qualidade na indústria. No entanto, parece evidente que, para equipes médias, insistir em escrever os testes primeiro, isto é, antes da codificação funcional, melhora a qualidade.

Inspeções de código melhoram a qualidade. Quer sejam realizadas com programação em pares, *peer review*, *code walkthroughs* ou inspeções *Fagan* completas, inspeções de código funcionam. Elas ajudam a melhorar a qualidade interna e externa do código. Inspeções de código têm um resultado melhor se realizadas muitas vezes e em pequenos lotes. Eu encorajo as equipes a inspecionar o código todo dia, por pelo menos, 30 minutos.

Análise colaborativa e projeto melhoram a qualidade. Quando as equipes são convidadas a trabalhar em conjunto para analisar os problemas e soluções de projeto, a qualidade é superior. Eu encorajo as equipes a realizar análise colaborativa e sessões de modelagem de projeto com toda a equipe. Modelagem de projeto deve ser realizada em pequenos lotes todo dia. Scott Ambler chamou isso de Modelagem Ágil¹⁴.

Uso de padrões de projeto melhora a qualidade. Padrões de projeto capturam soluções conhecidas para problemas conhecidos. Padrões de projeto garantem que mais informação estará disponível no início do ciclo de vida e que os defeitos de projeto são eliminados.

Uso de ferramentas de desenvolvimento modernas melhora a qualidade. Muitas ferramentas modernas incluem funções para executar análise de código estática e dinâmica. Essas funções devem ser utilizadas e ajustadas para cada projeto. Essas ferramentas de análise podem impedir que programadores cometam erros elementares, como a introdução de problemas bem conhecidos, como falhas de segurança.

Ferramentas modernas mais exóticas de desenvolvimento, tais como *Software Product Lines* (ou Fábricas de Software) e Linguagens de Domínio Específico reduzem defeitos. As fábricas de software podem ser utilizadas para encapsular os padrões de projeto, como fragmentos de código, reduzindo o potencial de inserção de defeitos no código. Elas também podem ser utilizadas para automatizar tarefas repetitivas de codificação, novamente, reduzindo o potencial de inserção de defeitos no código. O uso de fábricas de software também reduz a demanda de inspeções de código, visto que o código gerado por essas ferramentas não precisa ser re-inspecionado, ele tem uma qualidade embutida.

Algumas dessas últimas sugestões realmente se enquadram na categoria de redução da variabilidade no processo. O uso de fábricas de software, e talvez até mesmo os padrões de projeto, fazem com que os desenvolvedores mudem de comportamento. A grande sacada vem da utilização de testadores profissionais, da escrita de testes no início, da automação dos testes de regressão, e de inspeções de código. E mais uma coisa...

Reducir a quantidade de projeto-em-progresso aumenta a qualidade do software.

Reducir Trabalho-em-Progresso e Entregar Frequentemente

Em 2004 eu estava trabalhando com duas equipes na Motorola. Ambas as equipes estavam desenvolvendo código para um servidor de aplicações de celular. Uma equipe estava trabalhando no *download over-the-air* (OTA) de ringtones, jogos e outras aplicações e dados do servidor. A outra equipe estava desenvolvendo o dispositivo gerenciador *over-the-air* no servidor. Ambas as equipes estavam usando a metodologia *Feature Driven Development (FDD)*. Ambas as equipes tinham aproximadamente o mesmo tamanho - cerca de oito desenvolvedores, um arquiteto, no máximo cinco testadores, e um gerente de projeto. Trabalhando em conjunto com o pessoal de marketing, as equipes foram responsáveis pela análise e projeto. Além disso, havia equipes que trabalhavam com

projeto centrado na experiência do usuário e equipes responsáveis pela documentação do usuário (redação técnica) que prestavam serviços para ambas as equipes do projeto.

WIP, Lead Time, e Defeitos

A Figura 3.1 apresenta um diagrama de fluxo cumulativo do projeto da equipe de *download OTA*. Um diagrama de fluxo cumulativo é um gráfico de área que representa a quantidade de trabalho em um determinado estado. Os estados indicados neste gráfico são: “inventário”, que significa fila a iniciar; “iniciado”, que significa que os requisitos para a funcionalidade foram explicados para os desenvolvedores; “projetado”, que significa especificamente que um diagrama de sequência UML foi elaborado para a funcionalidade; “codificado”, que significa que os métodos do diagrama de sequência foram implementados, e “completo”, que significa que todos os testes unitários foram executados com sucesso, que o código foi revisado, e que o líder dos desenvolvedores aceitou o código e o promoveu para testes.

A primeira linha do gráfico mostra o número de funcionalidades para o escopo do projeto. O escopo foi entregue em dois lotes para os donos do negócio. A segunda linha mostra a quantidade de funcionalidades iniciadas. A terceira linha mostra a quantidade de funcionalidades projetadas. A quarta mostra a quantidade implementada, e a quinta linha mostra a quantidade de funcionalidades concluídas e prontas para testes.

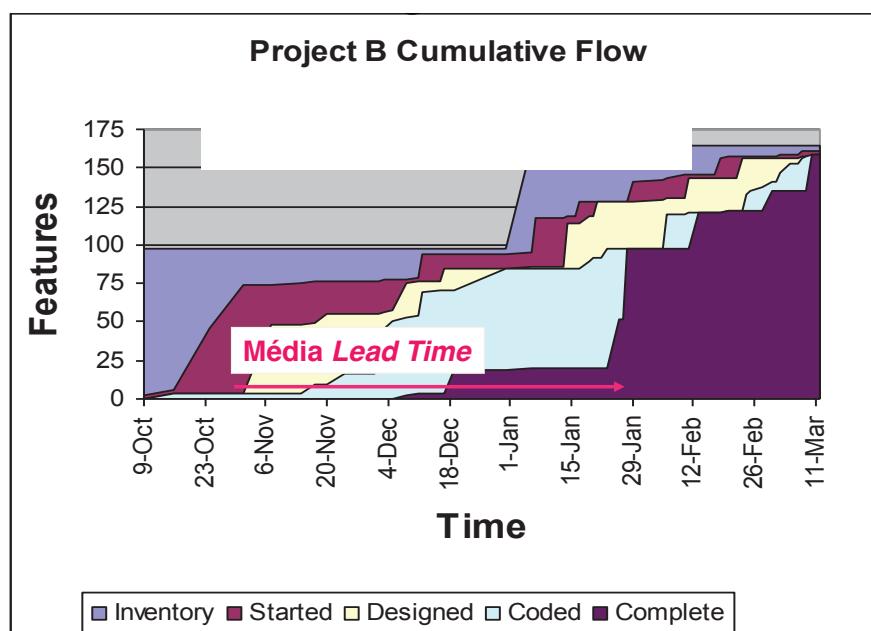


Figura 3.1 Diagrama de fluxo cumulativo da equipe de *Download OTA*, outubro de 2003 a inverno de 2004

A distância vertical entre a segunda e quinta linhas em um determinado dia mostra a quantidade de trabalho-em-progresso, enquanto que a distância horizontal entre a segunda e a quinta linhas mostra o tempo médio entre o inicio e fim de uma funcionalidade (*lead time*). É importante notar que a distância horizontal é um tempo médio, e não um *lead time* específico para uma funcionalidade específica. O diagrama de fluxo cumulativo não controla funcionalidades específicas. A quinquagésima quinta funcionalidade iniciada poderia ser a trigésima funcionalidade concluída. Não existe nenhuma associação entre uma linha sobre o eixo-y e uma funcionalidade específica no *backlog*.

Faltou disciplina à equipe trabalhando no *download* ou talvez a equipe não tenha comprado a ideia de usar o método *FDD*. Eles não trabalharam de forma colaborativa, como exige a *FDD*. Eles distribuíram grandes quantidades de funcionalidades para desenvolvedores individuais. Normalmente, eles tinham dez funcionalidades em progresso por desenvolvedor a qualquer momento. A equipe desenvolvendo o dispositivo gerenciador *over-the-air* no servidor seguiu o método segundo sua definição. Eles trabalharam bem colaborativamente. Eles desenvolveram testes unitários para 100 por cento das funcionalidades. E o mais importante, eles trabalharam em pequenos lotes de funcionalidades por vez, normalmente de cinco a dez funcionalidades em progresso para toda a equipe em qualquer período de tempo. Como referência, uma funcionalidade em *FDD* costuma representar cerca de 1.6 a 2.0 pontos de função de código.

A equipe do servidor de *download* OTA obteve um tempo médio por unidade de três meses a partir do início até a conclusão de uma funcionalidade para entrega à equipe em Seattle, Washington, a fim de testar a integração em Champaign, Illinois, como apresentado na figura 3.1. A equipe desenvolvendo o dispositivo gerenciador OTA teve um tempo médio por unidade de 5-10 dias, como ilustrado na Figura 3.2. A diferença na qualidade inicial, medida pela quantidade de defeitos escapados nos testes de sistema ou nos teste de integração, foi superior a 30 vezes entre as duas equipes. A equipe desenvolvedora do dispositivo gerenciador OTA produziu com uma qualidade inicial superior à da indústria, com dois ou três defeitos para cada 100 funcionalidades, enquanto que a equipe do servidor de *download* OTA obteve um desempenho médio da indústria com cerca de dois defeitos por funcionalidade.

Podemos ver a partir da análise dos gráficos que a quantidade de trabalho-em-progresso está diretamente relacionada ao *lead time*. A Figura 3.2 mostra claramente que o *lead time* médio diminui quando a quantidade de trabalho em progresso diminui. No ponto mais alto, o tempo médio é de doze dias. Mais tarde no projeto, com menos trabalho em progresso , o tempo de espera médio cai para apenas quatro dias.

Há uma relação de causa e efeito entre a quantidade de trabalho em progresso e *lead time* médio, e esta relação é linear. Na indústria de transformação, essa relação

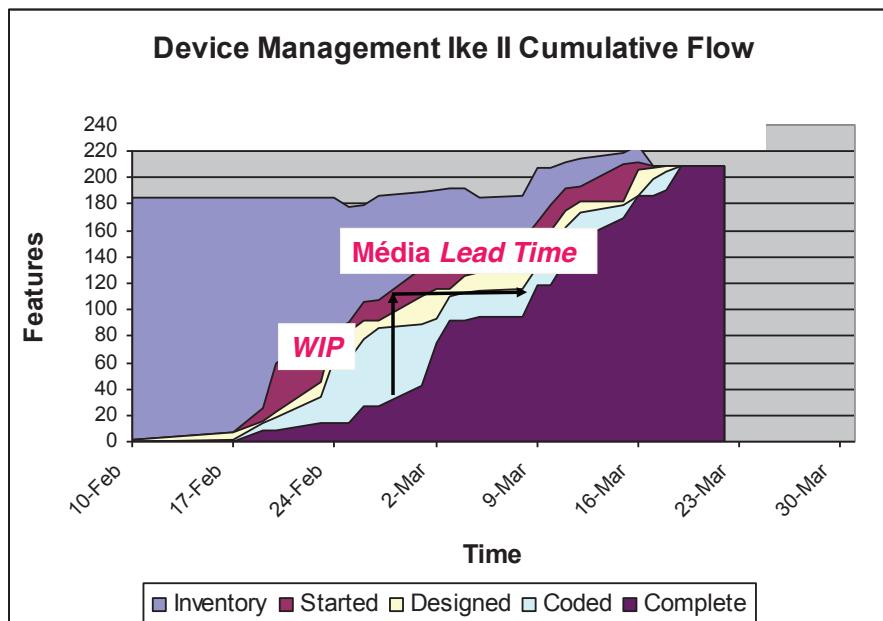


Figura 3.2 Diagrama de fluxo cumulativo da equipe do dispositivo gerenciador OTA inverno de 2004

é conhecida como *Little's Law*. As evidências obtidas nas duas equipes da Motorola sugerem existir uma correlação entre o aumento do *lead time* e um decréscimo na qualidade. *Lead times* mais longos parecem estar associados com uma significativa piora na qualidade. De fato, um *lead time* médio de aproximadamente 6 vezes e meia maior resultou em um aumento de 30 vezes no número de defeitos iniciais. Maiores *lead times* são resultado de maiores quantidades de trabalho em progresso. Assim, o ponto de nivelamento gerencial para a melhoria da qualidade é reduzir a quantidade de trabalho em progresso. Quando descobri isto, passei a gerenciar o trabalho em progresso como meio de controle da qualidade e me convenci da relação entre o *WIP* (*Work In Progress*) e qualidade inicial. No entanto, atualmente não há provas científicas para apoiar este resultado observado empiricamente.

Reducir o trabalho em progresso, ou encurtar o tamanho de uma iteração, impactará significativamente na qualidade inicial. Parece que a relação entre a quantidade de trabalho em progresso e qualidade inicial não é linear, ou seja, defeitos aumentam desproporcionalmente ao aumento da quantidade de *WIP*. Portanto, faz sentido que as iterações de duas semanas sejam melhores do que as iterações de quatro semanas e que iterações de uma semana sejam ainda melhores. Iterações menores conduzirão a um aumento da qualidade.

Segundo a lógica dos elementos apresentados, faz ainda mais sentido simplesmente limitar o *WIP* usando um sistema kanban. Se soubermos que a gestão do *WIP*

aumentará a qualidade, porque não introduzir políticas explícitas para limitá-lo, libertando assim os gerentes para que se concentrem em outras atividades?

Devido à estreita interação entre o trabalho em progresso e qualidade, a etapa 2 da receita deve ser implementada junto, ou logo após a primeira etapa.

Quem é melhor?

Durante a semana de comemoração do Natal no ano de 2003, eu me reuni com a equipe de *download OTA* e comentei com o líder de equipe que havia muito trabalho em progresso, que o *lead time* estava muito longo, e que poucas coisas eram realmente concluídas. Eu compartilhei minha preocupação pela piora na qualidade que resulta disso. Ele considerou o meu conselho e em Janeiro de 2004 introduziu algumas alterações à forma como a equipe trabalhava. O resultado foi uma redução no *WIP* em 2004 e um *lead time* comprovadamente mais curto. Tais mudanças, porém, vieram tarde demais para impedir a equipe de criar um grande número de defeitos.

Embora o diagrama sugira que o projeto foi concluído em meados do mês de Março de 2004, a equipe continuou trabalhando no software até meados de Julho do mesmo ano. Metade da equipe de *download OTA* foi removida de seus projetos originais para ajudar a corrigir os defeitos. Em Julho de 2004, o gerente geral de negócios da unidade declarou o produto completo, apesar de haver frequentes preocupações relacionadas à qualidade. O produto foi entregue à equipe de campo de engenharia. Durante a implantação, aproximadamente 50 por cento dos clientes cancelaram as entregas devido a preocupações com a qualidade. A equipe de campo de engenharia, apesar de manter boas relações pessoais com a equipe de engenharia do produto, tinha por estes um baixo nível de respeito profissional, além de falta de confiança na capacidade da equipe. Sua opinião era de que o produto era de má qualidade e que a equipe foi incapaz de oferecer algo melhor.

Ironicamente, se você entrasse naquele prédio no bairro SODO de Seattle durante aquela época e perguntasse aos desenvolvedores, “Quem é o cara mais inteligente por aqui?”, eles apontariam para alguém da equipe de *download OTA*. Se, ao invés disso, você lhes perguntasse: “Quem tem mais experiência?” Novamente, a mesma resposta. Se você olhasse os currículos, teria visto que a experiência média da equipe de *download OTA* ultrapassava o da equipe de desenvolvimento do dispositivo gerenciador OTA em três anos. No papel, tudo sugeria que a equipe de *download OTA* era melhor. E até hoje, algumas dessas pessoas acreditam que eles eram melhores, apesar de todas as numerosas provas no sentido contrário.

Eu sei pela minha experiência com a gerência e orientação desta equipe que alguns dos membros da equipe do dispositivo gerenciador OTA sofriam de baixa auto-estima profissional e receio de não serem tão talentosos quanto algumas das pessoas realmente

inteligentes que pertenciam à outra equipe. No entanto, a produtividade da equipe do dispositivo gerenciador OTA foi cinco vezes e meia maior, com uma qualidade inicial trinta vezes maior. A adoção do processo correto, uma boa disciplina, gestão firme e boa liderança, tudo isso fez a diferença. O que este exemplo demonstra é que você não precisa das melhores pessoas para produzir resultados de nível mundial. Uma das crenças centrais da comunidade ágil, a que me refiro como “esnobismo artesanal”, sugere que tudo que você precisa para o sucesso no desenvolvimento ágil é um pequeno grupo de pessoas realmente boas. No entanto, neste caso, uma equipe com talentos esparcos foi capaz de produzir resultados de nível mundial.

Entregas frequentes aumentam a confiança

Reducir WIP encurta também o *lead time*. Quando o *lead time* é mais curto, é possível a liberação de código funcional com maior frequência. Entregas com frequência maior constroem a confiança com equipes externas, em especial a equipe de marketing ou patrocinadores comerciais. Confiança é uma coisa difícil de definir. Os sociólogos chamam de capital social. O que eu aprendi é que a confiança é guiada por eventos e que pequenos, porém frequentes gestos ou eventos reforçam mais a confiança do que gestos grandiosos feitos apenas ocasionalmente.

Quando falo isto nas aulas, eu gosto de perguntar às alunas qual a opinião delas depois do primeiro encontro com um homem. Suponha que eles se divertiram bastante, e que depois ele não telefona por duas semanas. Ele, então, aparece em sua porta com um buquê de flores e um pedido de desculpas. Depois, peço-lhes para comparar este cara com outro que arruma tempo para escrever uma mensagem de texto em seu caminho para casa naquela noite para dizer: “Eu tive uma grande noite. Eu realmente quero te ver novamente. Posso te ligar amanhã?”, e depois realmente liga no dia seguinte. Adivinha quem elas preferem? Muitas vezes, pequenos gestos não custam nada, mas constroem mais confiança do que os grandes, caros (ou expansivos) gestos concedidos ocasionalmente.

É assim também com o desenvolvimento de software. Realizar entregas pequenas, frequentes e de alta qualidade cria mais confiança com as equipes parceiras do que entregas maiores, porém com menos frequência.

Entregas pequenas mostram que a equipe de desenvolvimento de software pode cumprir seu trabalho e está comprometida em fornecimentos com qualidade. Elas constroem a confiança com a equipe de marketing ou patrocinadores comerciais. Entregas de código de alta qualidade constroem a confiança com os parceiros de diferentes níveis, como operações, suporte técnico, engenharia de campo e de vendas.

Conhecimento implícito

É muito fácil especular por que pequenos lotes de código melhoram a qualidade. A complexidade dos problemas relacionados a trabalho e conhecimento cresce exponencialmente com a quantidade de trabalho em progresso. Enquanto isso, nossos cérebros humanos lutam para lidar com toda essa complexidade. Em desenvolvimento de software muito da transferência de conhecimento e descoberta de informação é tácita por natureza e é criada durante sessões de trabalho colaborativas, face a face. A informação é verbal e visual, mas em um formato casual, como um desenho em um quadro. Nossas mentes têm uma capacidade limitada para armazenar todo esse conhecimento implícito, que se degrada enquanto tentamos armazená-lo. Falhamos em lembrar detalhes precisos e cometemos enganos. Em equipes que trabalham no mesmo local e que possuem livre acesso entre si, essa perda de memória pode ser corrigida através de reiteradas discussões ou colhendo a memória compartilhada por um grupo de pessoas. Assim, equipes ágeis localizadas em um espaço de trabalho compartilhado são mais propensas a reter o conhecimento implícito por mais tempo. Independentemente de qualquer coisa, o conhecimento implícito se deprecia com o tempo, então *lead times* mais curtos são essenciais. Sabemos que a redução do trabalho em progresso está diretamente relacionada à redução do *lead time*. Assim, podemos inferir que haverá menor depreciação do conhecimento implícito quando temos menos trabalho em progresso, resultando em maior qualidade.

Em resumo, a redução do trabalho em progresso melhora a qualidade e permite entregas mais frequentes. Entregas com maior frequência de código de alta qualidade melhoram a confiança com as equipes externas.

Equilibrando a demanda e o rendimento

Equilibrar a demanda em relação ao rendimento implica em definirmos a frequência na qual aceitaremos novos requisitos durante o desenvolvimento do software, mantendo a velocidade na qual códigos funcionais são entregues. Quando isso ocorre, estamos efetivamente ajustando nosso trabalho em progresso a determinado tamanho. Enquanto o trabalho é entregue, “puxamos” um novo trabalho (ou requisitos) das fontes criadoras de demanda. Assim, qualquer discussão sobre priorização e comprometimento com novas atividades só pode ocorrer após a entrega de algum trabalho atual.

O efeito desta mudança é profundo. O rendimento do seu processo será restrinido por um gargalo. É improvável que você saiba onde o gargalo está. De fato, se você conversar com qualquer pessoa da cadeia de valor, ela provavelmente alegará estar

completamente sobrecarregada. No entanto, uma vez que você consiga equilibrar a demanda e o rendimento e limitar o trabalho em progresso dentro de sua cadeia de valor, a mágica acontecerá. Apenas os recursos relacionados ao gargalo continuarão totalmente carregados. Rapidamente, outras pessoas da cadeia de valor descobrirão ter uma folga na sua capacidade de produção. Enquanto isso, aqueles que trabalham no gargalo estarão ocupados, mas não sobrecarregados de tarefas. Pela primeira vez, talvez em anos, a equipe não estará mais sobrecarregada e muitas pessoas experimentarão algo muito raro nas suas carreiras: a sensação de ter tempo à disposição.

Crie folga

Muito do estresse será retirado da organização e as pessoas serão capazes de se concentrar em fazer o seu trabalho com precisão e qualidade. Elas poderão se orgulhar de seu trabalho e irão desfrutar dessa experiência. As pessoas com algum tempo de sobra começam a usar esse tempo em prol de melhorar suas circunstâncias, podendo, por exemplo, melhorar seu ambiente de trabalho ou fazer algum treinamento. Elas provavelmente começarão a melhorar as suas habilidades, suas ferramentas, e a maneira como interagem com as outras pessoas em outros níveis da organização. Com o passar do tempo e como pequenas melhorias levam a outras melhorias, a equipe estará realizando melhoria contínua. A cultura mudará. A capacidade de folga criada pela ação de limitar o trabalho em progresso e puxar um novo trabalho apenas quando houver capacidade, viabilizará uma melhoria que ninguém antes acreditou ser possível.

É necessário folga para viabilizar a melhoria contínua. É necessário equilibrar demanda e rendimento, e limitar a quantidade de trabalho em progresso para que a folga ocorra.

Intuitivamente, as pessoas acreditam que devem eliminar a folga. Então, depois de limitar o trabalho em progresso, equilibrando demanda e rendimento, a tendência é “equilibrar a produção”, ajustando os recursos a fim de que todos sejam plenamente utilizados de forma eficiente. Embora possa parecer eficiente e satisfazer as práticas de gestão contábil típicas do século XX, isso impedirá a criação de uma cultura de melhoria. É necessário folga para viabilizar a melhoria contínua. Para ter folga, faz-se necessário a presença de uma cadeia de valor desequilibrada com um recurso de gargalo. Aperfeiçoar para utilização não é desejável.

Priorize

Caso as três primeiras etapas da receita tenham sido implementadas, tudo estará funcionando normalmente. Código de alta qualidade estará sendo entregue com frequência. *Lead times* de desenvolvimento serão relativamente curtos, visto que o trabalho em progresso é limitado. Novos trabalhos devem ser puxados para o desenvolvimento apenas quando há capacidade após a conclusão dos trabalhos existentes. Nesse momento, a atenção da gerência pode se voltar para o aperfeiçoamento do valor entregue e não apenas à quantidade de código entregue. Há pouco ganho em se dar atenção a definição de prioridades quando não há previsibilidade na entrega. Por que se esforçar tentando priorizar a entrada, quando não há uma dependência de prioridade na entrega? Até que isso seja resolvido, o tempo de gestão será melhor utilizado ao se concentrar em aumentar tanto a capacidade, como a previsibilidade da entrega. Faz-se necessário priorizar a entrada quando se sabe que é realmente possível entregar o trabalho com a mesma prioridade na qual ele foi solicitado.

Influência

Priorização não deve ser realizada pela organização de engenharia e, portanto, não deve estar sob o controle da gestão de engenharia. Melhorar a priorização exige que o dono do produto, o patrocinador do negócio, ou o departamento de marketing mude seu comportamento. Na melhor das hipóteses, a gerência de engenharia pode procurar influenciar a forma como a priorização é realizada.

A fim de possuir capital político e social para influenciar a mudança, um nível de confiança deve ser estabelecido. Sem uma capacidade de entregar código de alta qualidade com frequência não há como existir confiança e, portanto, nessa situação, há pouca possibilidade de influência na definição de prioridades e de, assim, aperfeiçoar o valor a ser entregue pela equipe de software.

Recentemente, tornou-se popular na comunidade Ágil se falar sobre aperfeiçoamento do valor do negócio e como a taxa de produção de código funcional (chamada de “velocidade” de desenvolvimento de software) não é uma métrica importante. Isso ocorre porque o valor de negócio entregue é a verdadeira medida do sucesso. Embora isso seja verdade, é importante não perder de vista a escada de maturidade da capacidade que uma equipe deve subir. A maioria das organizações é incapaz de medir e informar o valor de negócio entregue. Elas devem primeiramente construir capacidade nas competências básicas antes de tentar maiores desafios.

Construindo Maturidade

Essa é maneira como eu acredito que uma equipe deve amadurecer: em primeiro lugar, aprendendo a construir um código de alta qualidade. Então, reduzindo o trabalho em progresso, encurtando os prazos e entregando frequentemente. Em seguida, equilibrando demanda e rendimento, limitando o trabalho em progresso, e criando folga para liberar banda, o que permitirá melhorias. Então, com um bom funcionamento e aperfeiçoando a capacidade de desenvolvimento de software, melhorar a priorização para aperfeiçoar a entrega de valor. Apenas aguardar pelo aperfeiçoamento do valor do negócio é um desejo. Tome ações para chegar a este nível de maturidade de forma incremental - siga a Receita para o Sucesso.

Ataque as Fontes de Variabilidade para Melhorar Previsibilidade

Os efeitos da variabilidade e como reduzí-los dentro de um processo são tópicos avançados. Reduzir a variabilidade no desenvolvimento de software requer que as pessoas mudem a maneira como elas trabalham para aprender novas técnicas e mudar seu comportamento pessoal. Tudo isso é difícil. Portanto, não é para principiantes ou para as organizações imaturas.

Variabilidade resulta em mais trabalho em progresso e prazos mais longos; isto é explicado detalhadamente no capítulo 19. A variabilidade gera uma maior necessidade de folga nos recursos que não são gargalo, a fim de lidar com o fluxo de trabalho, visto que seus efeitos se manifestam sobre o fluxo de trabalho através da cadeia de valor; uma compreensão completa de por que isso é verdade exige algum conhecimento em controle estatístico de processo e teoria das filas, o que está além do escopo deste livro. Pessoalmente gosto do trabalho de Donald Wheeler e Reinertsen Donald sobre a variabilidade e filas, por isso, se você quiser mais informações sobre esses temas, comece por aí.

Por agora, acredite que a variabilidade no tamanho dos requisitos, e na quantidade de esforço despendida na análise, projeto, codificação, teste, integração e entrega prejudica o rendimento do processo e os custos para executar uma cadeia de valor no desenvolvimento de software.

No entanto, algumas fontes de variabilidade são inadvertidamente projetadas em processos através de escolhas de políticas pobres. O estudo de caso no capítulo 4 destaca alguns exemplos: o replanejamento mensal, o acordo de nível de serviço nas estimativas, e priorização de mudanças em produção. Todos estes três exemplos são controlados por políticas que podem ser alteradas; basta mudar uma política de

processo existente para reduzir drasticamente fontes de variabilidade que afetam a previsibilidade.

Receita para o Sucesso e Kanban

Kanban viabiliza todos os seis passos da Receita para o Sucesso. Kanban viabiliza a Receita para o Sucesso e a Receita para o Sucesso viabiliza a promessa de implementação do Kanban pelos gerentes, além de mostrar porque Kanban é uma técnica tão valiosa.

Takeaways

- ❖ Kanban implementa todos os aspectos da Receita para o Sucesso.
- ❖ A Receita para o Sucesso explica o valor de Kanban.
- ❖ Baixa qualidade pode representar o maior desperdício em desenvolvimento de software.
- ❖ Reduzir o trabalho em progresso aumenta a qualidade.
- ❖ Melhoria na qualidade aumenta a confiança com parceiros de outros níveis, como o de operações.
- ❖ Entregas realizadas com frequência aumentam a confiança com parceiros de maior nível organizacional, como marketing.
- ❖ A demanda e o rendimento podem ser balanceados num sistema puxado.
- ❖ Um sistema puxado expõe os gargalos e cria folgas onde não há gargalos.
- ❖ Uma priorização de boa qualidade maximiza o valor entregue a partir de uma cadeia de valor de desenvolvimento de software em bom funcionamento.
- ❖ Priorização tem pouco valor caso não exista uma boa qualidade ou previsibilidade de entrega.
- ❖ Fazer mudanças para reduzir variabilidade requer folga.
- ❖ Reduzir variabilidade reduz a necessidade de folga.
- ❖ Reduzir a variabilidade viabiliza um balanceamento de recursos (e, potencialmente, uma redução de pessoas).
- ❖ Reduzir a variabilidade reduz requisitos de recursos.
- ❖ Reduzir a variabilidade permite reduzir kanban *tokens*, diminuir *WIP*, e reduzir o *lead time* médio.
- ❖ As folgas viabilizam as oportunidades de melhoria.
- ❖ Melhoria no processo leva a uma maior produtividade e previsibilidade.

❖ CAPÍTULO 4 ❖

Do Pior ao Melhor em Cinco Trimestres

Em outubro de 2004, Dragos Dumitriu era um gerente de programa na Microsoft. Ele havia se tornado recentemente chefe de um departamento que tinha a reputação de ser o pior na divisão de TI da Microsoft.

O cargo de “Gerente de Programa” na Microsoft é mais prontamente interpretado em outros lugares como de gerente de projeto, mas na Microsoft ele tipicamente também inclui alguma responsabilidade pela análise e arquitetura. Atribui-se a um gerente de programa alguma iniciativa, projeto ou produto e ele é responsável por uma *feature* ou um conjunto delas. Um gerente de programa recrutará recursos de áreas funcionais, tais como desenvolvimento e testes, a fim de completar o trabalho. Dragos foi o responsável pela manutenção de software da unidade de negócios XIT. Esta equipe (como ilustrado na figura 4.1) localizada numa filial na Índia que era nível 5 no modelo CMMI, realizou pequenas atualizações e corrigiu alguns defeitos de produção para cerca de 80 aplicações de tecnologia da informação multifuncionais utilizadas pelo pessoal da Microsoft em todo o mundo. Dragos trabalhava no campus corporativo em Redmond, Washington. Nesta época, eu também estava trabalhando lá.



Figura 4.1 A equipe em Hyderabad, India, no final de 2004; Dragos é o quarto da esquerda para a direita

O Problema

Dragos se ofereceu a assumir a equipe que tinha a pior reputação de atendimento ao cliente da Microsoft. Seu trabalho como agente de mudança, determinado a corrigir os longos *lead times* e a baixa expectativa em relação à equipe, foi prejudicado pelo clima político. Vários de seus antecessores na posição ainda eram colegas de trabalho em outros projetos dentro da mesma unidade de negócios, e preocupava-os que a melhora no desempenho, causada por ele, os faria parecer ruins, em comparação.

Os programadores e testadores trabalhando para a filial seguiam o Software Engineering Institute's Personal Software Process / Team Software Process (metodologia PSP / TSP), conforme determinado contratualmente pela Microsoft. Jon De Vaan, que na época se reportava diretamente a Bill Gates, era um grande fã de Watts Humphrey, do Software Engineering Institute. Como chefe da Engineering Excellence da Microsoft, ele estava em posição de determinar quais processos deveriam ser usados dentro do departamento de TI e nas filiais. Isso significava que mudar o método do ciclo de desenvolvimento de software em uso não era uma opção disponível.

Dragos percebeu que nem o PSP / TSP, nem o nível CMMI na filial eram a raiz dos problemas. Na verdade, a equipe produziu basicamente o que lhes foi pedido, e com uma qualidade muito elevada. No entanto, eles tinham um *lead time* de cinco meses para solicitações de mudanças e isso, juntamente com o seu *backlog* de solicitações, foi crescendo incontrolavelmente. A percepção era de um time que foi mal organizado e gerenciado. Como resultado, a gerência sênior não estava disposta a fornecer investimentos adicionais para corrigir o problema.

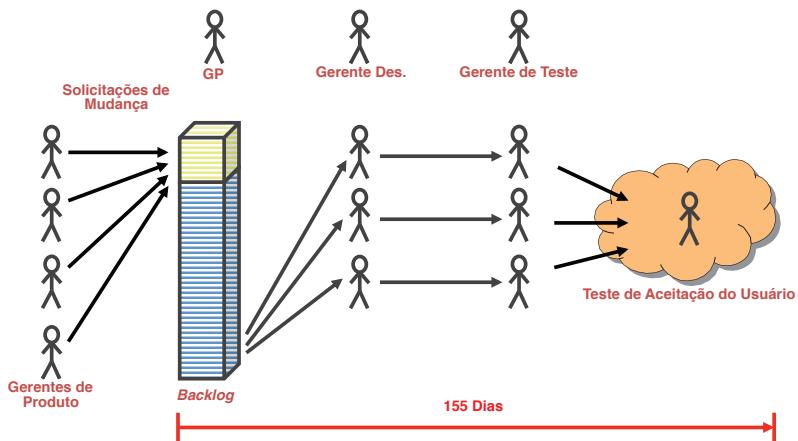


Figura 4.2 Equipe de manutenção XIT: Fluxo inicial apresentando *lead time*

Assim, as restrições para mudanças eram políticas, fiscais e relacionadas às políticas da empresa. Ele pediu meu conselho.

Visualizar o Fluxo de Trabalho

Eu pedi que Dragos esboçasse o fluxo de trabalho e ele elaborou um desenho simples que descrevia o ciclo de vida de uma solicitação de mudança, e então, nós discutimos os problemas. A Figura 4.2 é uma cópia exata do que Dragos desenhou. A figura do GP representa Dragos.

As solicitações chegavam incontrolavelmente. Quatro gerentes de produto representavam e controlavam o orçamento de um conjunto de clientes que eram donos das aplicações mantidas pelo XIT. Eles adicionavam novas requisições, inclusive defeitos escapados na produção (encontrados em campo). Estes defeitos não haviam sido criados pelo time de manutenção, mas pelos times de desenvolvimento das aplicações. Os times de desenvolvimento das aplicações geralmente eram desmontados um mês após a entrega de um sistema novo, e o código fonte era manipulado pelo time de manutenção.

Fatores Influenciando o Desempenho

Quando uma requisição chegava, Dragos devia enviá-la para a Índia para uma estimativa. A política era que as estimativas deviam ser realizadas e então enviadas de volta para os donos do negócio em 48 horas. Isso facilitava o cálculo do retorno do

investimento (ROI) a fim de decidir se a requisição deveria ir em frente. Uma vez por mês, Dragos deveria se reunir com os gerentes de produto e outros *stakeholders* para re-priorizar o *backlog* e criar um plano de projeto para as requisições.

Nessa época eram entregues em torno de sete requisições por mês. O *backlog* possuía 80 ou mais itens e continuava crescendo. Isso significa que 70 ou mais requisições estavam sendo re-priorizadas e re-agendadas todo mês, e que elas demoravam mais do que 4 meses para serem processadas; essa era a causa raiz da insatisfação. As requisições eram pequenas, e a constante re-priorização significava que as pessoas que faziam as requisições estavam constantemente decepcionadas.

As requisições eram acompanhadas numa ferramenta chamada *Product Studio*. Uma versão atualizada dessa ferramenta foi posteriormente publicada como “*Team Foundation Server Work Item Tracking*”. O time de manutenção XIT tinha um tipo de organização que eu frequentemente tinha contato nas minhas aulas e no meu trabalho de consultoria – eles tinham muitos dados, mas não os utilizavam. Dragos começou a explorar os dados e descobriu que uma requisição demandava 11 dias de engenharia. No entanto o *lead time* era de tipicamente 125 a 155 dias. Mais de 90 por cento do *lead time* era gasto em filas, ou em outras formas de desperdício.

As estimativas para novas requisições consumiam muito esforço, então nós decidimos avaliar esse cenário fazendo algumas conjecturas. Apesar das estimativas serem de “*rough order of magnitude* – ordem de magnitude aproximada” (ROM), a expectativa do cliente era receber uma estimativa precisa, dessa maneira, os membros da equipe eram muito cautelosos no levantamento das estimativas. O esforço para realização de cada estimativa era em torno de um dia para cada desenvolvedor e testador. Nós calculamos rapidamente que apenas a estimativa de esforço estava consumindo 33 por cento da capacidade, e que num mês ruim esse valor poderia ser quase 40 por cento. Esta capacidade foi alocada em detrimento das atividades de codificação e teste. Estimativas para novas requisições também atrapalhavam os planos feitos para o mês.

Além das requisições de mudança, a equipe tinha um segundo tipo de trabalho, conhecido como *production text changes* (PTCs) – mudança de texto em produção – os quais eram de natureza gráfica ou textual, ou envolviam modificações de valor em tabelas ou arquivos XML. Essas alterações não necessitavam de um desenvolvedor e eram frequentemente realizadas pelos donos do negócio, gerentes de produto, ou pelo gerente de programa; mas era necessária uma aprovação formal de teste, então elas afetavam os testadores.

PTCs chegavam sem muito aviso, e eram tradicionalmente priorizados em relação a qualquer outro trabalho ou estimativa de esforço. PTCs tendiam a chegar em *batches* esporádicos e também afetavam qualquer plano feito para o mês. (veja Figura 4.3).

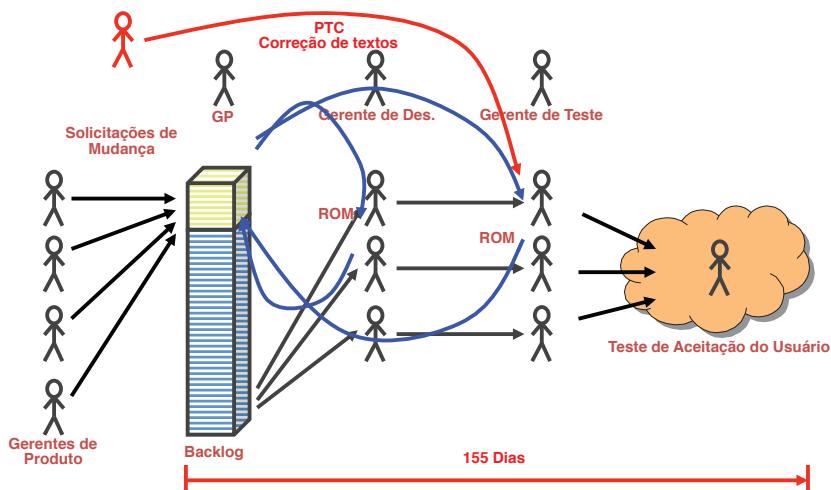


Figura 4.3 Fluxo de trabalho apresentando as estimativas ROM e entrada de PTC

Torne Explícitas as Políticas do Processo

A equipe estava seguindo o processo exigido, o qual incluía muitas políticas de decisão ruins que foram feitas por gerentes em vários níveis. É importante entender um processo como um conjunto de políticas que governam o comportamento e que estão sob controle gerencial. Por exemplo, a política que definia o uso do PSP/TSP foi estabelecida no nível do vice-presidente executivo, um degrau abaixo de Bill Gates, e seria difícil ou impossível mudá-la. No entanto, muitas outras políticas como priorizar estimativas sobre codificação e teste, foram desenvolvidas localmente e estavam sob a autoridade colaborativa dos gerentes imediatos. É possível que as políticas fizessem sentido na época em que foram implementadas; mas as circunstâncias mudaram e não houve tentativa de revisar e atualizar as políticas que governavam as operações da equipe.

Estimativa Era um Desperdício

Após algumas discussões com seus colegas e gerente, Dragos decidiu promulgar duas políticas gerenciais iniciais. Primeiro, a equipe pararia de estimar. Dragos queria recuperar a capacidade desperdiçada pela atividade de estimativa e usá-la para desenvolver e testar software. Eliminar a aleatoriedade no cronograma causada pela realização de estimativas melhoraria a previsibilidade, e Dragos esperava que essa combinação tivesse um grande impacto na satisfação do usuário.

No entanto, remover as estimativas era um problema, pois afetaria os cálculos de ROI, e os clientes poderiam achar que as priorizações seriam mal feitas. Além disso, as

estimativas eram usadas para facilitar a contabilidade de custos entre departamentos e as transferências de orçamento. Estimativas também eram usadas para implementar a política de governança; eram permitidas apenas pequenas requisições através do sistema de manutenção. Grandes requisições que excediam mais de 15 dias de desenvolvimento ou teste deviam ser submetidas para um projeto de iniciativa maior através do *program management office* (PMO) formal. Falaremos sobre esses problemas em breve.

Limitar Trabalho em Progresso (WIP)

A outra mudança que Dragos decidiu fazer foi limitar o trabalho em progresso e puxar trabalho a partir de uma fila de entrada após a finalização do trabalho atual. Ele escolheu limitar o WIP em desenvolvimento para uma requisição por desenvolvedor e usar uma regra similar para os testadores. Ele incluiu uma fila entre desenvolvimento e teste

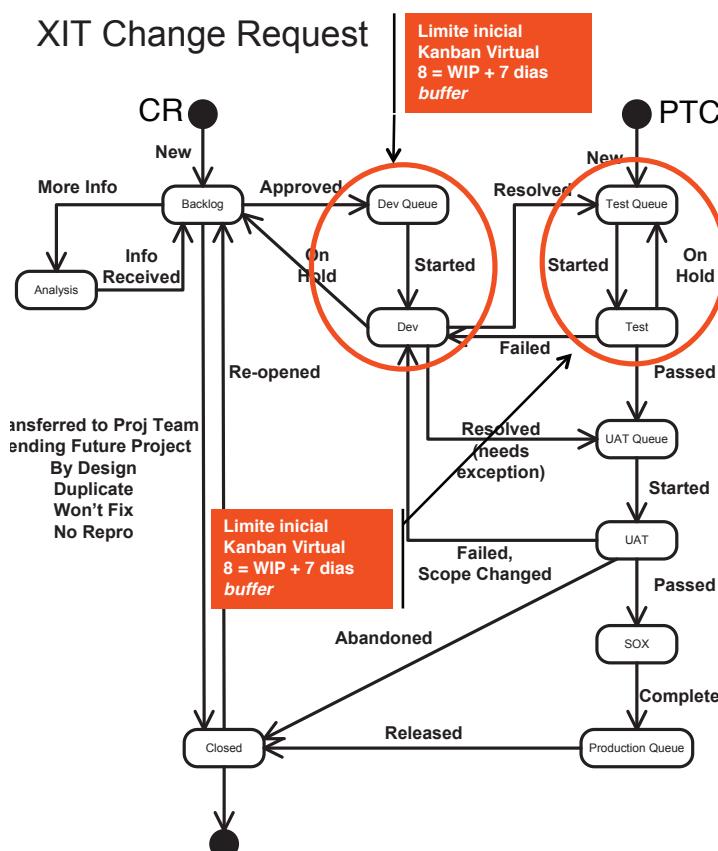


Figura 4.4 Diagrama de estados com o fluxo de trabalho desejado e limites de WIP

Nota: Esta é uma escolha política. Uma requisição de mudança por desenvolvedor em qualquer período de tempo é uma política. Ela pode ser posteriormente modificada. Pensar no processo como um conjunto de políticas é um elemento chave para o Método Kanban.

para o recebimento de PTCs e para suavizar o fluxo de trabalho entre desenvolvimento e teste, como ilustrado na Figura 4.4. Esta abordagem de uso de um *buffer* para suavizar a variabilidade em tamanho e esforço é discutida no capítulo 19.

Estabelecendo uma Cadênci na Entrada

A fim de facilitar a decisão de limitar o *WIP* e institucionalizar um sistema puxado, Dragos precisou pensar sobre a cadênci na interação com os gerentes de produto. Ele imaginou que uma reunião semanal seria viável, já que o tópico da reunião seria simplesmente o reabastecimento da fila de entrada a partir do *backlog*. Numa semana típica haveria três *slots* - espaços disponíveis - na fila. Dessa maneira a discussão seria ao redor da seguinte questão, “Quais três itens do *backlog* você mais deseja que sejam os próximos a serem entregues?”; esta cadênci é modelada na Figura 4.5.

Nota: Cadênci é um conceito no Método Kanban que determina o ritmo de um tipo de evento. Priorização, entregas, retrospectivas, e qualquer outro evento recorrente podem possuir sua própria cadênci.

Ele queria “garantir” um tempo de entrega de 25 dias a partir da aceitação na fila de entrada. Essa duração de 25 dias era consideravelmente maior do que os 11 dias em

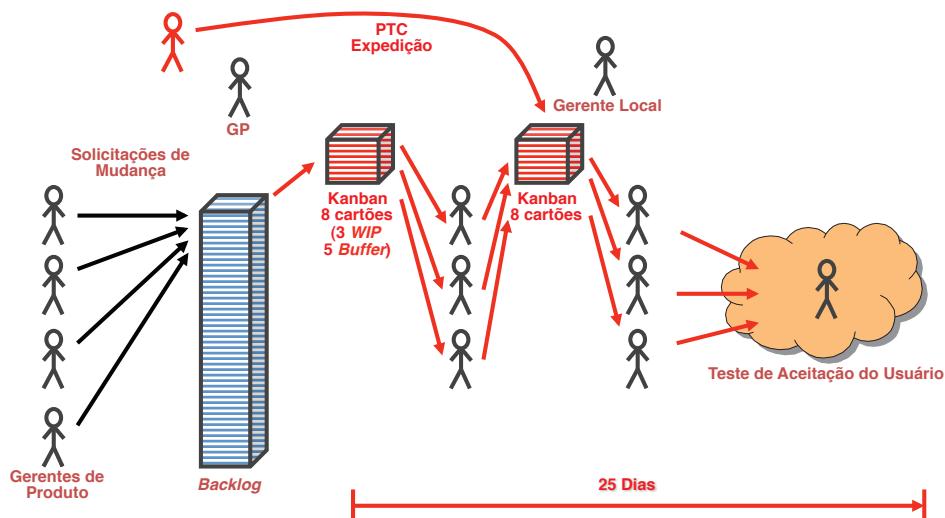


Figura 4.5 Fluxo de trabalho com limites de *WIP* Kanban e filas

média gastos em engenharia para completar o trabalho. Os desvios estatísticos requeriam algo em torno de 30 dias, mas ele antecipou alguns deles; 25 dias soava atrativo, especialmente comparando-se ao *lead time* atual de 140 dias. Ele esperava atingir a meta com regularidade, construindo confiança com os gerentes de produto e seus clientes enquanto isso.

Fazendo um Novo Acordo

Dragos fez uma oferta para os gerentes de produto. Ele pediu para que eles se reunissem uma vez por semana para discutir priorização, pois ele limitaria a quantidade de trabalho em progresso e a equipe pararia de estimar. Em troca, ele garantiria entregas dentro de 25 dias e a reportagem do desempenho da equipe em relação a essa métrica.

Dragos estava pedindo que os clientes desistissem do cálculo do ROI para transferência de orçamento entre departamentos baseando-se nas estimativas por requisição. Em troca, ele estava oferecendo um tempo de entrega menor nunca visto e confiabilidade. Para lidar com os problemas de contabilidade, Dragos pediu que eles considerassem que todas as requisições exigiam, em média, um esforço de 11 dias de engenharia, isto é, eles deviam aceitar que os custos eram essencialmente fixos. Dragos estava pedindo que eles desconsiderassem o paradigma contabilidade de custo no qual a transferência de orçamento entre departamentos era baseada.

A justificativa para isso era que a filial tinha um contrato de 12 meses que era faturado mensalmente. A filial alocava as pessoas com esse contrato e elas eram pagas sem considerar se estavam trabalhando ou não. Os fundos desse orçamento eram fixos e distribuídos proporcionalmente para os quatro gerentes de produto. Dragos garantiu que cada um dos gerentes de produto teria uma alocação de capacidade justa; isso deveria liberá-los do acompanhamento individual das requisições. Se eles aceitassem que os seus dólares lhe compravam capacidade e que ela estava garantida, talvez eles desconsiderassem o acordo de custo baseado em unidade para transferência de orçamento. Algumas regras simples foram criadas para determinar quem deveria selecionar as requisições para reabastecer a fila a fim de que a capacidade fosse alocada de maneira justa. Um esquema simples e leve elaborado por todos foi suficiente para alcançar isso.

Implementando Mudanças

Enquanto os gerentes de produto e muitos colegas gerentes de Dragos na unidade XIT continuavam céticos, o consenso era de que ele deveria ter uma chance, afinal as coisas iam mal e estavam piorando. As coisas não ficariam piores do que já estavam! Alguém precisava tentar algo, e era esperado que Dragos implementasse algumas mudanças.

Então as mudanças foram promulgadas.

Elas começaram a funcionar. Requisições eram processadas e entregues para produção. O tempo de entrega para novos compromissos foi de 25 dias como prometido. A reunião semanal funcionou sem problemas, e a fila era reabastecida a cada semana. A confiança começou a ser construída com os gerentes de produto.

Ajustando as Políticas

Você deve estar se perguntando como a priorização ficou mais fácil se o cálculo do ROI não era mais realizado. Descobrimos que o cálculo não era necessário. Se algo era importante e tinha valor, ele era selecionado do *backlog* para a fila de entrada; se não era importante, ele não era selecionado. Posteriormente Dragos reconheceu que uma nova política era necessária: Qualquer item com mais de seis meses era retirado do *backlog*. Se o item não era importante o suficiente para ser selecionado mesmo após seis meses da sua entrada no *backlog*, poder-se-ia assumir que ele nunca seria importante. Caso ele fosse realmente importante poderia ser submetido novamente.

Onde está a política de governança que previne que itens grandes sejam direcionados para manutenção quando deveriam ser submetidos para um projeto maior? Isso foi resolvido aceitando-se que esses itens poderiam desviar-se para a manutenção, visto que os dados históricos demonstravam que esses itens eram menos de dois por cento do total de requisições. Os desenvolvedores foram instruídos a ficarem em alerta, e caso uma nova requisição apresentasse requerer mais de 15 dias de esforço, eles deveriam alertar o gerente local. O risco e o custo de se fazer isso eram menores do que meio por cento da capacidade disponível; era um ótimo *tradeoff*. Retirando as estimativas, a equipe ganhou mais de 33 por cento da capacidade com um risco menor do que 1 por cento daquela capacidade. Essa nova política delegou aos gerentes a responsabilidade de gerenciar riscos e de manifestarem-se quando fosse necessário!

As primeiras duas mudanças foram estabelecidas para durarem 6 meses. Algumas poucas mudanças foram feitas durante esse período. Como mencionado, a política de expurgar um item do *backlog*

Nota: Este é um tema comum no Método Kanban. A combinação de políticas explícitas, transparência, e visualização fazem com que os membros da equipe tomem suas próprias decisões e gerenciem os riscos. A gerência confia no sistema porque comprehende que o processo é feito de políticas. As políticas são elaboradas para gerenciamento de riscos e para a realização de entregas que atendam às expectativas do cliente. As políticas são explícitas, o trabalho é acompanhado com transparência, e todos os membros da equipe entendem as políticas e como usá-las.

* N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: troca, compensação, compromisso.

foi adicionada; a reunião semanal com os donos do produto também desapareceu. O processo estava sendo executado tão naturalmente que Dragos modificou a ferramenta *Product Studio* a fim de que um e-mail fosse enviado quando um *slot* se tornasse disponível na fila de entrada. Ele poderia dessa maneira alertar os gerentes de produto por e-mail que decidiam entre eles quem colocaria um item na fila. A escolha era feita e uma nova requisição era retirada do *backlog* e incluída na fila duas horas após o *slot* ter se tornado disponível.

Procurando mais Melhorias

Dragos começou a procurar mais melhorias. Ele vinha estudando os dados históricos de produtividade dos testadores da sua equipe e comparando-os com os de outras equipes dentro do XIT mantido pela mesma filial. Ele concluiu que os testadores não estavam sobrecarregados e que tinham muita capacidade disponível. Como consequência os desenvolvedores eram um gargalo significante. Ele decidiu visitar a equipe na Índia. Quando retornou ele instruiu a filial a fazer uma mudança na alocação das pessoas. Ele reduziu a equipe de testes de três para dois e incluiu um novo desenvolvedor (Figura 4.6). Isso resultou num aumento quase linear da produtividade, e o rendimento naquele trimestre subiu de 45 para 56.

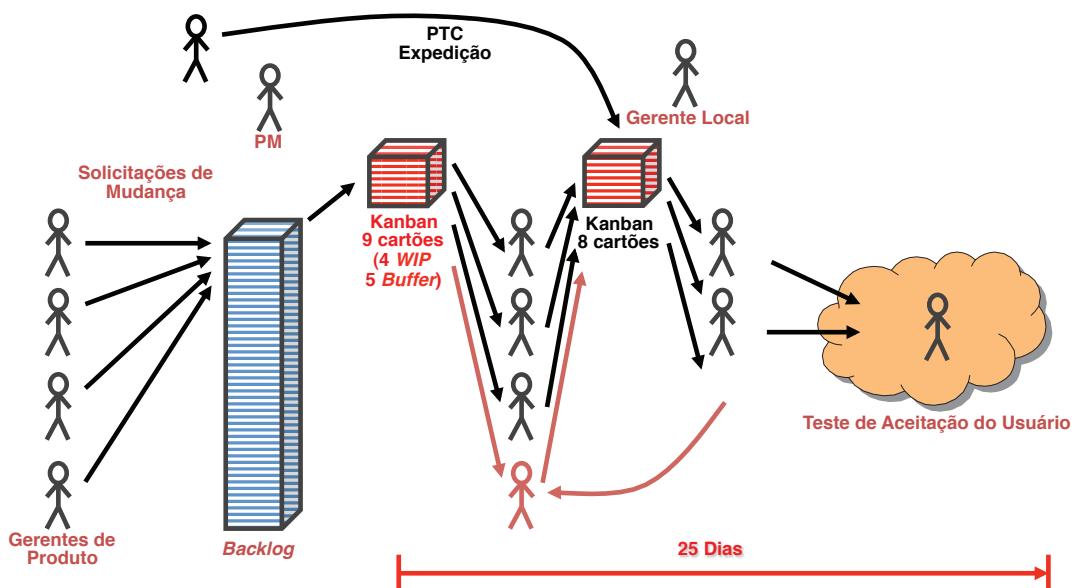


Figura 4.6 Nivelamento de recursos

O ano fiscal da Microsoft estava chegando ao fim. Os gerentes seniores receberam a notícia de uma melhoria significante na produtividade e na consistência das entregas da equipe de manutenção de software XIT – *XIT Sustained Engineering*. Finalmente a gerência confiava em Dragos e nas técnicas que ele estava empregando. Destinou-se mais dinheiro ao departamento para mais duas pessoas: um desenvolvedor e um testador foram contratados em Julho de 2005 (Figura 4.7). Os resultados foram significantes (Figuras 4.8, 4.9).

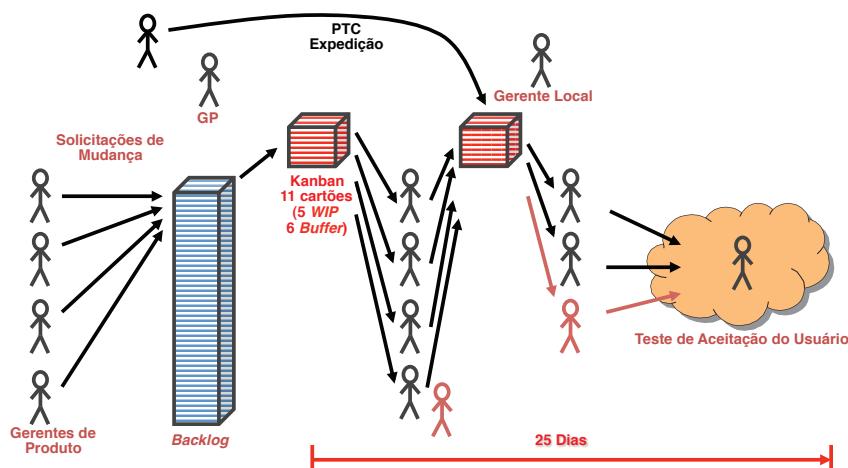


Figura 4.7 Adicionando recursos

Resultados

A capacidade adicional foi suficiente para aumentar o rendimento em relação à demanda. O resultado? O *backlog* foi inteiramente eliminado em 22 de Novembro de 2005. Nessa época a equipe tinha reduzido o *lead time* para uma média de 14 dias contra os 11 dias iniciais de esforço de engenharia. A meta de desempenho para tempo de entrega de 25 dias foi atendida em 98 por cento. O rendimento das requisições aumentou mais de 3 vezes, enquanto o *lead time* diminuiu em mais de 90 por cento, e a confiabilidade aumentou praticamente da mesma maneira. Nenhuma mudança foi realizada no processo de desenvolvimento ou de teste. As pessoas trabalhando em Hyderabad não tinham conhecimento de mudanças significantes. O método PSP/TSP não foi alterado e todos os requisitos de governança corporativa, processo e de contrato com a filial foram atendidos. A equipe ganhou o *Engineering Excellence Award* – Prêmio de Excelência em Engenharia na segunda metade de 2005. Dragos foi recompensando

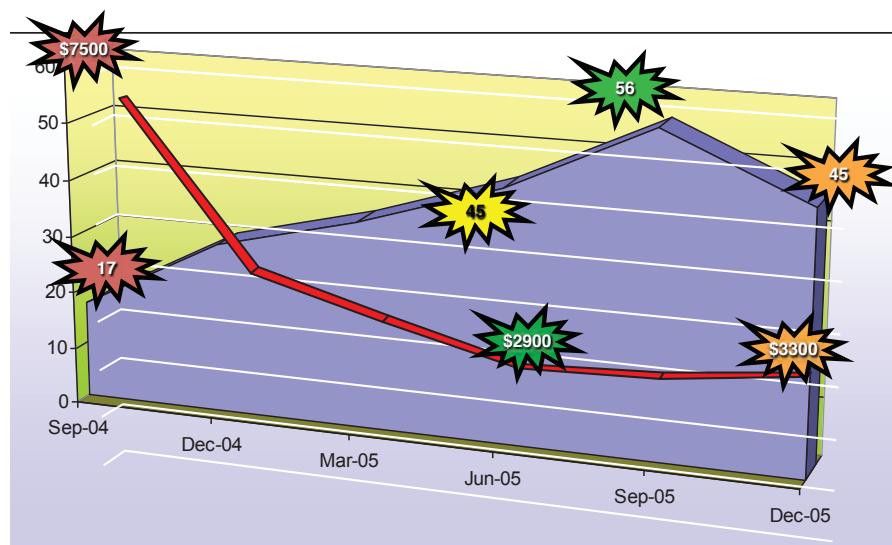


Figura 4.8 Rendimento trimestral sobreposto ao custo unitário

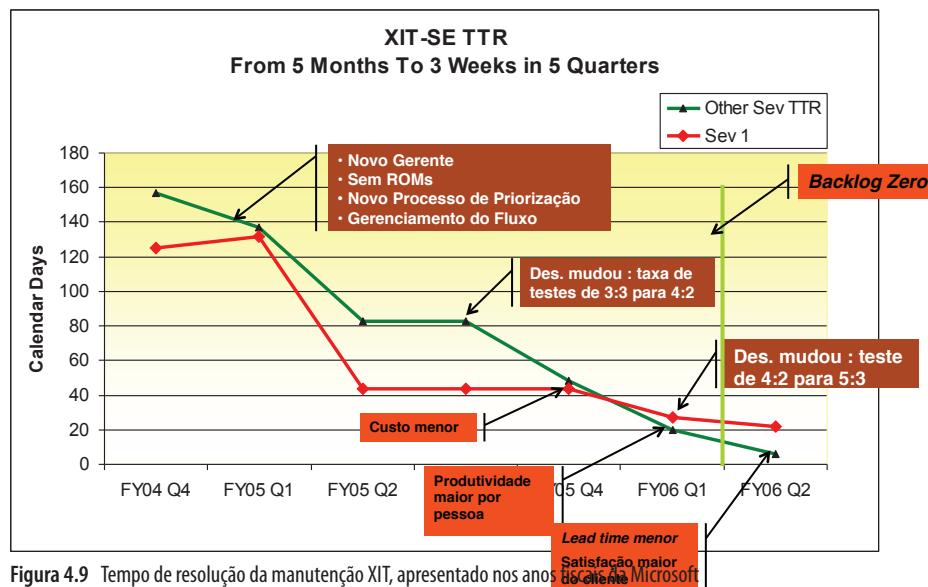


Figura 4.9 Tempo de resolução da manutenção XIT, apresentado nos anos fiscais

com responsabilidades adicionais, e o dia-a-dia do gerenciamento da equipe foi para as mãos do gerente local na Índia, que foi re-alocado para Washington.

Essas melhorias aconteceram em parte devido a incrível habilidade gerencial de Dragos Dumitriu, mas os elementos básicos de mapeamento da cadeia de valor, análise de fluxo, limitação do *WIP*, e implementação de um sistema puxado foram peças chave. Sem o paradigma de fluxo e a abordagem kanban de limitar o *WIP*, os ganhos de desempenho não teriam acontecido. Kanban possibilita mudanças incrementais com políticas de risco simples e baixa resistência à mudança.

O caso XIT mostrou como um sistema puxado com *WIP* limitado foi implementado num projeto distribuído usando recursos *offshore*, e facilitado com o uso de uma ferramenta de acompanhamento eletrônica. Não havia quadro visual e as características mais sofisticadas do Método Kanban descritas nesse livro ainda emergiriam. No entanto, qual gerente poderia ignorar a possibilidade de uma melhoria na produtividade de mais de 200 por cento, com uma redução do *lead time* em 90 por cento, serem alcançadas com uma melhoria significante da previsibilidade e com políticas de risco mínimas e resistência à mudança?

Takeaways

- ❖ O primeiro sistema kanban foi implementado pela equipe de manutenção de software da Microsoft XIT *Sustained Engineering*, que começou em 2004.
- ❖ O primeiro sistema kanban usava uma ferramenta de acompanhamento eletrônica.
- ❖ O primeiro sistema kanban foi implementado com uma equipe *offshore* de nível 5 no modelo CMMI em Hyderabad, Índia.
- ❖ O fluxo de trabalho deve ser esboçado e visualizado.
- ❖ O processo deve ser descrito como um conjunto de políticas explícitas.
- ❖ Kanban proporciona mudanças incrementais.
- ❖ Kanban proporciona mudança com uma política de risco mínima.
- ❖ Kanban proporciona mudança com pouca resistência.
- ❖ Kanban revelará oportunidades de melhoria que não envolvem mudanças complexas nos métodos de engenharia.
- ❖ O primeiro sistema kanban produziu um aumento de 200 por cento na produtividade com uma redução de 90 por cento no *lead time* e uma melhoria similar na previsibilidade.
- ❖ Melhorias significantes são possíveis gerenciando gargalos, eliminando desperdício, e reduzindo a variabilidade que afeta as expectativas e a satisfação dos clientes.
- ❖ Mudanças podem levar tempo para proporcionarem um efeito generalizado. O primeiro estudo de caso levou 15 meses para surtir um bom efeito.

❖ CAPÍTULO 5 ❖

Uma Cultura de Melhoria Contínua

Em Japonês, a palavra kaizen literalmente significa “melhoria contínua”. Uma cultura de trabalho onde todos os trabalhadores estão focados em continuamente melhorar a qualidade, a produtividade e a satisfação do cliente é conhecida como uma “cultura *kaizen*”. Muito poucas empresas realmente conseguiram esta cultura. Empresas como a Toyota, onde a participação dos trabalhadores no seu programa de melhoria é próximo de 100%, e onde cada funcionário tem em média uma sugestão implementada todos os anos como parte de melhoria contínua, são muito raras.

No mundo do desenvolvimento de software, o Software Engineering Institute (SEI) da Carnegie Mellon University definiu o nível mais alto do seu CMMI (*Capability Maturity Model Integration*) como Otimizando. Otimizando implica que a qualidade e o desempenho da organização estão sendo continuamente refinados. Embora não explicitamente declarado, visto que o CMMI fala pouco sobre cultura, é mais provável que um comportamento de otimização contínua aconteça dentro de uma organização com uma cultura *kaizen*.

Cultura *Kaizen*

Para entender porque é tão difícil alcançar uma cultura *kaizen*, primeiro devemos entender como tal cultura seria. Só então podemos discutir por que podemos querer alcançar tal cultura e quais seriam os seus benefícios.

Na cultura *kaizen* a força de trabalho tem poder. Indivíduos sentem-se livres para agir; livres para fazer a coisa certa. Eles espontaneamente se debruçam sobre os problemas, discutem as opções e implementam correções e melhorias. Em uma cultura *kaizen*, a força de trabalho não tem medo. A norma subjacente é para a gestão ser tolerante a falhas se a experimentação e a inovação fizerem parte do processo - assim como a melhoria de desempenho. Em uma cultura *kaizen*, indivíduos são livres (dentro de alguns limites) para se auto-organizarem em torno do trabalho que eles fazem e como eles o fazem. Controles visuais e sinais são evidentes e as pessoas se oferecem para trabalhar em tarefas de trabalho ao invés de serem atribuídas por um superior. Uma cultura *kaizen* envolve um elevado nível de colaboração e uma atmosfera de coleguismo onde todo mundo olha para o desempenho da equipe e os negócios acima de si. Uma cultura *kaizen* centra-se no pensamento em nível de sistemas ao fazer melhorias locais que melhoram o desempenho geral.

Uma cultura *kaizen* tem um elevado nível de capital social. É uma cultura de alta confiança em que indivíduos, independentemente de sua posição na hierarquia de tomada de decisões de negócios, respeitam uns aos outros e a contribuição de cada pessoa. Culturas de alta confiança tendem a ter estruturas mais horizontais do que culturas de confiança inferior. É o grau de capacitação que permite uma estrutura mais horizontal trabalhar de forma eficaz. Assim, alcançar uma cultura *kaizen* pode permitir a eliminação de desperdício de camadas de gerenciamento e reduzir os custos de coordenação, como resultado.

Muitos aspectos de uma cultura *kaizen* estão em oposição às normas culturais e sociais estabelecidas na cultura ocidental moderna. No ocidente, somos levados a ser competitivos. Nossos sistemas de ensino incentivam a concorrência nos estudos e nos esportes atléticos. Mesmo nossos esportes de equipe tendem a incentivar o desenvolvimento de heróis e equipes construídas em torno de um ou dois jogadores excepcionalmente talentosos. A norma social é enfocar primeiramente o indivíduo e confiar em pessoas extraordinárias para alcançar a vitória ou para nos salvar do perigo. Não é de se admirar que tenhamos dificuldade no local de trabalho para incentivar coleguismo e pensamento no nível de sistemas e cooperação.

Kanban Acelera Maturidade e Capacidade Organizacional

O Método Kanban é projetado para minimizar o impacto inicial das mudanças e reduzir a resistência à adoção da mudança. Adotar Kanban deve mudar a cultura da sua empresa e ajudar a torná-la madura. Se a adoção é feita corretamente, a organização irá transformar-se em uma que adota mudanças facilmente e se torna boa na implementação de alterações e melhorias no processo. O SEI refere-se a isso como uma capacidade de Inovação Organizacional e Implantação (OID, sigla no original)¹⁵ dentro do modelo CMMI. Tem sido mostrado¹⁶ que organizações que atingem este nível elevado de capacidade de gestão de mudança podem adotar métodos de desenvolvimento Ágil, tais como Scrum, mais rápido e melhor do que organizações menos maduras.

Quando você implementa Kanban pela primeira vez você está procurando otimizar os processos existentes e alterar a cultura organizacional, ao invés de substituir os processos existentes por outros que podem fornecer melhorias econômicas dramáticas. Esta situação levou a críticas¹⁷ de que Kanban otimiza apenas algo que precisa ser alterado. No entanto, há agora consideráveis evidências empíricas¹⁸ de que Kanban acelera a obtenção dos altos níveis de maturidade organizacional e capacidade em áreas de processo de alta maturidade fundamentais tais como Análise Causal e Resolução (CAR, sigla no original) e Inovação Organizacional e Implantação.

Quando você escolhe usar Kanban como um método para orientar a mudança em sua organização, você está seguindo a opinião de que é melhor aperfeiçoar o que já existe, porque é mais fácil e mais rápido e encontrará menos resistência do que executar uma iniciativa de mudança gerenciada e projetada. A introdução de uma mudança radical é mais difícil do que melhorar incrementalmente o que já existe. Você também deve compreender que os aspectos de jogo colaborativo do Kanban irão contribuir para uma mudança significativa na sua cultura corporativa e sua maturidade. Esta mudança permitirá mais tarde alterações mais significativas, novamente com baixa resistência, do que se estivesse tentando fazer essas alterações imediatamente. A adoção do Kanban é um investimento de longo prazo em capacidade, maturidade e cultura da sua organização. Não se destina como uma solução rápida.

ESTUDO DE CASO: DESENVOLVIMENTO DE APLICAÇÕES NA CORBIS

Quando apresentei um sistema kanban na Corbis, em 2006, eu o fiz por muitos dos benefícios mecânicos que se demonstrou com a Microsoft XIT em 2004 (como descrito no capítulo 4). A aplicação inicial foi a mesma — manutenção de aplicações de TI. Eu não estava antecipando uma mudança cultural significativa ou uma mudança significativa na maturidade organizacional. Eu não esperava o que agora conhecemos como o Método Kanban evoluísse a partir deste trabalho.

Enquanto eu escrevo este livro em 2010, estabeleceu-se agora que Kanban naturalmente encaixa-se com o trabalho de manutenção de TI. Em 2006, ainda não era claro, mas uma forma de sistema kanban parecia se encaixar bem com os problemas funcionais dos trabalhos de manutenção. Eu não entrei na Corbis com a intenção de “praticar Kanban”. Entrei lá com a intenção de melhorar a satisfação do cliente com a equipe de desenvolvimento de software. Foi uma coincidência feliz que o primeiro problema a ser tratado era a falta de previsibilidade com relação às entregas de manutenção de software de TI.

CENÁRIO E CULTURA

Em 2006, Corbis era uma empresa privada, com cerca de 1300 funcionários em todo o mundo. Ela controlava os direitos digitais para muitas obras de arte fascinantes, bem como representava aproximadamente 3000 fotógrafos profissionais, licenciando seu trabalho para uso por editores e anunciantes. Era a segunda maior empresa do mundo de estoque de fotografias. Havia outras linhas de negócios também, o mais notável dos quais foi o negócio de licenciamento de direitos que controlava os direitos em nome dos familiares, propriedades e empresas de gerenciamento, para as imagens e os nomes de personalidades e celebridades. O departamento de TI era composto por cerca de 110 pessoas divididas entre engenharia de software e manutenção de sistemas/operações de rede. A força de trabalho era aumentada em alguns momentos com terceirizados para trabalhar em projetos grandes. No seu auge em 2007, o departamento de engenharia de software empregou 105 pessoas, incluindo um contingente de 35 pessoas em Seattle e outras 30 em um fornecedor em Chennai, Índia. A maioria dos testes era realizada por esta equipe em Chennai. Houve uma abordagem muito tradicional ao gerenciamento de projetos: tudo era planejado em uma árvore de dependências de tarefas e conduzido por um escritório de gerenciamento de programas. Era uma companhia com uma cultura conservadora, no que havia sido uma indústria relativamente conservadora e lenta. Empregava abordagens conservadoras e tradicionais de gerenciamento de projetos e de ciclo de vida de engenharia de software.

O departamento de TI mantém um conjunto diversificado de aproximadamente 30 sistemas. Alguns eram tipicamente sistemas contábeis e sistemas de recursos humanos; outros eram exóticos e às vezes esotéricos, aplicações para a indústria de gestão de direitos digitais. Havia uma ampla gama de tecnologias, plataformas de software e linguagens suportadas. A força de trabalho era incrivelmente fiel; muitas pessoas no departamento de TI haviam estado com a empresa por mais de oito anos, alguns com até 15 anos de serviço. Nada mau para uma empresa que tinha 17 anos. O processo existente era tradicional, o ciclo de vida de desenvolvimento de software (SDLC, sigla no original) estilo cascata, que tinha sido institucionalizado ao longo dos anos com a criação de um departamento de análise de negócios, um departamento de análise de sistemas, um departamento de desenvolvimento e um departamento de teste *offshore*. Dentro destes departamentos havia muitos especialistas, tais como analistas cuja experiência era contabilidade e cuja especialidade era

em finanças. Alguns desenvolvedores também eram especialistas, por exemplo, os programadores de J.D. Edwards, que mantinham o software de contabilidade J.D. Edwards.

Nada disto era ideal; mas era o que era. As coisas eram da forma como eram. Quando entrei para a empresa havia alguma expectativa e ansiedade de que eu iria impor um método Ágil e usar o poder da minha posição para forçar as pessoas a mudar seu comportamento. Embora isto pudesse ter funcionado, teria sido brutal, e o impacto durante a transição teria sido grave. Eu tinha medo de tornar as coisas piores, com medo de que projetos ficasse parados enquanto um novo treinamento era fornecido e o pessoal adaptado às novas formas de trabalho. Eu também estava com medo de perder pessoas-chave, sabendo que a força de trabalho era frágil devido aos níveis excessivos de especialização. Eu escolhi introduzir um sistema kanban, colocar o trabalho de manutenção de sistemas de volta aos trilhos e ver o que aconteceria.

A NECESSIDADE DE UMA FUNÇÃO DE MANUTENÇÃO DE SOFTWARE

A manutenção de software (ou “RRT” para “Rapid Response Team,” como ela era conhecida internamente) tinha sido financiada pelo comitê executivo com um orçamento adicional de dez por cento para o departamento de engenharia de software. Isto equivalia a cinco pessoas adicionais em 2006. Algum tempo antes da minha chegada, as cinco pessoas haviam sido contratadas. Devido à natureza diversa dos sistemas envolvidos e ao elevado grau de especialização existente da equipe, foi decidido que uma equipe dedicada de cinco pessoas para fazer o trabalho de manutenção não seria uma boa solução. Então cinco pessoas adicionais foram colocadas no *pool* geral de recursos: um gerente de projeto, um analista, um desenvolvedor e dois testadores. Isto introduziu uma complicação adicional que era necessária, de uma perspectiva de governança, para mostrar que o adicional de cinco pessoas estava realmente fazendo trabalhos de manutenção e não tinham simplesmente sido sugados para o portfólio do grande projeto. No entanto, em um dia qualquer, essas cinco pessoas poderiam ser qualquer uma das aproximadamente 55 pessoas na equipe.

Uma solução teria sido que todos preenchessem planilhas de tempos complexas, o que teria acrescentado um encargo administrativo, para mostrar que 10 por cento das horas da equipe estavam sendo gastas em atividades de manutenção. Isso teria sido altamente intrusivo, mas é como tipicamente gerentes de nível médio respondem a tal desafio. Outra abordagem era introduzir um sistema kanban.

Uma expectativa havia sido definida de que uma equipe de manutenção permitiria à Corbis fazer versões incrementais para os sistemas de TI a cada duas semanas. Grandes projetos normalmente envolviam grandes atualizações de sistemas e novas versões a cada três meses. Mas com o amadurecimento do negócio e a natureza destes sistemas se tornou mais complexa, esta cadência de grandes liberações trimestrais tinha se tornado intermitente. Além disso, alguns dos sistemas existentes foram efetivamente descontinuados e realmente deveriam ser substituídos completamente.

Substituição de sistemas legados é um grande desafio, e normalmente envolve projetos longos com uma grande equipe até que uma paridade de funcionalidade é alcançada e o antigo sistema possa ser desligado assim que o outro é colocado online (essa abordagem é muito longe de ser ideal, mas é normal).

Assim as liberações de manutenção eram a única área dentro do TI da Corbis onde kanban poderia permitir alguma forma de agilidade nos negócios.

PEQUENOS PROJETOS DE MANUTENÇÃO NÃO ESTAVAM FUNCIONANDO

O sistema existente para entregar versões de manutenção, o sistema que estava quebrado, devia agendar uma série de projetos de curta duração, de duas semanas. Isso poderia ter sido reconhecido como desenvolvimento de software Ágil usando iterações de duas semanas, mas não foi. Quando eu cheguei, a negociação de escopo para um ciclo de liberação de duas semanas estava levando cerca de três semanas. Assim os custos de transação iniciais de uma liberação eram maiores do que o trabalho de valor agregado. Estava levando cerca de seis semanas para que uma versão de duas semanas fosse liberada.

IMPLEMENTANDO A MUDANÇA

Era claro que antes de fazer quaisquer alterações que o *status quo* era inaceitável. O atual sistema era incapaz de oferecer o nível exigido de agilidade nos negócios. Manutenção de sistemas nos deu uma oportunidade ideal para introduzir mudanças. Trabalhos de manutenção não eram em geral de missão crítica. No entanto, eram altamente visíveis, já que as pessoas de negócios tinham acesso direto à priorização, e as escolhas eram altamente táticas e importantes em curto prazo para as metas de negócio. Manutenção de sistemas era algo que todos se preocupavam e queriam trabalhar eficazmente. E finalmente, havia uma razão convincente para fazer alterações. Todo mundo estava descontente com o sistema existente. Os desenvolvedores, testadores e analistas estavam todos irritados com o tempo desperdiçado com negociações de escopo, e as pessoas de negócios estavam extremamente insatisfeitas com os resultados.

Projetamos um sistema kanban com liberações agendadas a cada duas semanas, previstas para as 13h a cada duas quartas-feiras e com reuniões de priorização agendadas com as pessoas de negócios, marcadas às 10h de cada segunda-feira. Assim, a cadência de priorização era semanal e a cadência de liberação era quinzenal. A escolha da cadência foi determinada através de discussões colaborativas com parceiros e baseada nos custos de transação e coordenação das atividades. Algumas outras mudanças foram feitas. Introduzimos uma fila chamada “Pronto para Engenharia” (entrada) com um limite de trabalho-em-progresso de cinco itens e, em seguida, acrescentamos limites durante todo o ciclo de análise, desenvolvimento, construção e teste de sistema. Teste de aceitação, homologação e pronto para produção ficaram ilimitados, já que eles

não foram considerados como sendo de capacidade restrita e estavam, até certo ponto, fora do nosso controle político imediato.

EFEITOS PRIMÁRIOS DAS MUDANÇAS

Os efeitos da introdução de um sistema kanban foram, em um nível, esperados, mas em outro, eles eram muito notáveis. Nós começamos a fazer liberações a cada duas semanas. Após cerca de três iterações, elas foram acontecendo sem incidentes. A qualidade era boa e havia pouca ou nenhuma necessidade de correções emergenciais quando o novo código ia para produção. O *overhead* de agendamento e planejamento de liberações havia caído drasticamente, e as disputas entre as equipes de desenvolvimento e o escritório de gestão de programa tinham desaparecido quase completamente. Portanto, kanban havia cumprido sua promessa básica. Nós estávamos liberando *releases* de alta qualidade muito regularmente, com um mínimo de *overhead** de gerenciamento. Custos de transação e coordenação de um *release* foram reduzidos drasticamente. A equipe estava recebendo mais trabalho e eles estavam entregando esse trabalho ao cliente mais frequentemente.

Os efeitos secundários foram ainda mais notáveis.

EFEITOS IMPREVISTOS DA INTRODUÇÃO DO KANBAN

Para a equipe de desenvolvimento, introduzimos a parede de cartões física usando notas auto-adesivas em um quadro branco em Janeiro de 2007. Começamos a realização de reuniões diárias em torno do quadro por 15 minutos às 9h30min todos os dias. O quadro físico tinha um enorme efeito psicológico comparado a qualquer coisa que obtivemos da ferramenta de registro eletrônico que usamos na Microsoft. Ao participar de pé todos os dias, os membros da equipe eram expostos a uma espécie de fotografia do fluxo de trabalho através do quadro. Itens de trabalho bloqueados eram marcados com bilhetes cor-de-rosa, e a equipe tornou-se muito mais focada na resolução de problemas e manutenção do fluxo. A produtividade aumentou drasticamente.

Com o fluxo de trabalho agora visível no quadro, comecei a prestar atenção ao funcionamento do processo. Como resultado, eu fiz algumas mudanças no quadro. Minha equipe de gerentes veio a entender as alterações que eu estava fazendo e por que, e até março, eles estavam fazendo alterações por eles mesmos. Por sua vez, os membros de suas equipes, os desenvolvedores individuais, testadores e analistas, começaram a ver e a entender como as coisas funcionavam. No início do verão, todos na equipe se sentiam aptos a sugerir mudanças e tínhamos observado a formação espontânea (geralmente multifuncionais) de grupos de indivíduos que discutiam problemas de processo e os desafios, e faziam alterações conforme eles acreditavam ser apropriado. Normalmente, eles informavam à gestão após o fato. O que havia surgido durante aproximadamente seis meses

* N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: sobrecarga.

era uma cultura *kaizen* na equipe de engenharia de software. Os membros da equipe se sentiam com poder. O medo tinha sido removido. Eles se orgulhavam do seu profissionalismo e de suas realizações e eles queriam fazer melhor.

Mudança Sociológica

Desde a experiência da Corbis, tem havido relatos similares da área. Rob Hathaway da Indigo Blue foi o primeiro a replicar verdadeiramente esses resultados com o grupo de TI na IPC Media em Londres. O fato de que outros têm sido capazes de replicar os efeitos sociológicos do Kanban que observamos na Corbis me faz acreditar que existe uma causalidade e que não foi uma coincidência nem um efeito direto do meu envolvimento pessoal.

Eu pensei muito sobre o que provocou essas mudanças sociológicas. Métodos Ágeis têm nos oferecido transparência no trabalho-em-progresso há uma década, e as equipes que seguem o Método Kanban parecem alcançar uma cultura *kaizen* mais rápida e eficazmente do que equipes típicas de desenvolvimento Ágil de software. Muitas vezes, equipes que adicionam Kanban a seus métodos Ágeis existentes encontram uma melhoria significativa no capital social entre membros da equipe. O que poderia ser isso?

A minha conclusão é que Kanban fornece transparência para o trabalho e também para o processo (ou fluxo de trabalho). Ele fornece visibilidade sobre como o trabalho é passado de um grupo para outro. Kanban permite que todos os interessados vejam os efeitos de suas ações ou inatividades. Se um item está bloqueado e alguém é capaz de desbloqueá-lo, Kanban mostra isso. Talvez haja um requisito ambíguo. Normalmente, o especialista no assunto que pode resolver a ambiguidade poderia esperar para receber um e-mail com uma convocação para uma reunião. Após uma chamada, eles organizam uma reunião de acordo com suas agendas, talvez três semanas depois. Com Kanban e a visibilidade que ele proporciona, o especialista no assunto percebe o efeito da falta de ação e prioriza a reunião, talvez reorganizando sua agenda para marcar uma reunião na semana atual, em vez de atrasá-la por duas semanas. Além da visibilidade do fluxo de processo, limites de trabalho-em-progresso também forçam interações desafiadoras a acontecer mais cedo e mais frequentemente. Não é fácil ignorar um item bloqueado e simplesmente trabalhar em outra coisa. Este aspecto “pare a linha” do Kanban parece incentivar o “comportamento de enxame”* através do fluxo de valor.

* N. de T.: *swarming behavior*, no original, designa o comportamento em que indivíduos se reúnem como um enxame, a fim de resolver um problema em conjunto.

Quando as pessoas de diferentes áreas funcionais e diferentes cargos se unem sobre um problema e colaboram para encontrar uma solução, mantendo o fluxo de trabalho e melhorando o desempenho no nível do sistema, aumenta o nível de capital social e de confiança da equipe. Com elevados níveis de confiança gerada através da colaboração melhorada, o medo é eliminado da organização.

Limites de trabalho-em-progresso , juntamente com classes de serviços (explicados no Capítulo 11) também capacitam os indivíduos para tomar decisões de planejamento por conta própria, sem direção ou supervisão da gestão. Delegar poder melhora o nível de capital social, demonstrando que os superiores confiam nos subordinados para tomar decisões de alta qualidade por conta própria. Os gerentes são liberados da supervisão de colaboradores individuais e podem concentrar sua energia mental em outras coisas, como o desempenho de processos, gestão de risco, desenvolvimento da equipe e maior satisfação de clientes e funcionários.

Kanban melhora significativamente o nível de capital social dentro da equipe. Os maiores níveis de confiança e a eliminação do medo incentivam a inovação colaborativa e resolução de problemas. O efeito resultante é o rápido surgimento de uma cultura *kaizen*.

Propagação Viral da Colaboração

Kanban claramente melhorou a atmosfera do departamento de engenharia de software da Corbis, mas foram os resultados além desse grupo os mais notáveis. Vale a pena relatar e analisar como a propagação viral do Kanban melhorou a colaboração na empresa.

ESTUDO DE CASO: DESENVOLVIMENTO DE APLICAÇÕES NA CORBIS, CONTINUAÇÃO

Toda manhã de segunda-feira, às 10h, Diana Kolomiyets, a gerente de projeto responsável por coordenar liberações de manutenção de sistemas de TI, convocava a reunião do comitê de priorização RRT. Os participantes da área de negócios normalmente eram vice-presidentes. Eles comandavam uma unidade de negócios e se reportavam a um vice-presidente sênior ou a um alto executivo da empresa. Dito de outra forma, um vice-presidente se reportava a um membro do comitê executivo. Corbis ainda era pequena o suficiente de modo que fazia sentido para um gerente de tão alto nível participar da reunião semanal. Igualmente, as opções táticas sendo feitas eram suficientemente importantes que precisavam realmente a direção de um vice-presidente para influenciar uma boa escolha.

Normalmente, cada participante recebia um e-mail na sexta-feira anterior à reunião. Nele haveria algo como: “Nós antecipamos que haverá dois *slots* livres na fila na próxima semana. Por favor, examinem seus itens de *backlog* e selecionem os candidatos para discussão na reunião de segunda-feira”.

BARGANHANDO

Nas primeiras semanas do novo processo, alguns dos participantes vinham com uma expectativa de negociação. Eles poderiam dizer, “Sei que há apenas um *slot* livre, mas eu tenho duas pequenas requisições, você pode fazer as duas”? Esta negociação raramente era tolerada. Os outros membros do comitê de priorização assegurariam que todos seguiriam as regras. Eles poderiam responder “Como posso saber que eles são pequenos? Devo acreditar em sua palavra?” ou rebater com “Eu também tenho dois pequenos. Por que motivo eu não devo ter meus favoritos selecionados?” Refiro-me a isso como o “Período de Negociação”, pois indica o estilo de negociação que aconteceu nas reuniões de priorização.

DEMOCRACIA

Após cerca de seis semanas e coincidentemente na mesma época em que a equipe de desenvolvimento introduziu o uso do quadro físico, o comitê de priorização introduziu um sistema de votação democrático. Eles espontaneamente se ofereceram, já que eles se cansaram das disputas. A negociação na reunião estava desperdiçando tempo. Levou algumas iterações para aperfeiçoar o sistema de votação, que se estabeleceu como um sistema onde cada participante tem direito a um voto para cada *slot* livre na fila naquela semana. No início da reunião, cada membro propunha um pequeno número de candidatos para a seleção. Com o passar do tempo, propor solicitações ficou mais sofisticado, algumas pessoas vieram com apresentações do PowerPoint, outros com planilhas que estabeleciam um caso de negócio. Mais tarde, ouvimos que alguns membros estavam fazendo *lobby* com seus colegas, levando-os para o almoço. Acordos estavam sendo feitos, “se eu votar na sua escolha nesta semana, você votará na minha na próxima semana?” Paralelamente ao novo sistema democrático de priorização, o nível de colaboração estava crescendo entre as unidades de negócios no nível de vice-presidente. Embora nós não tenhamos percebido no momento, o nível do capital social na empresa inteira estava crescendo. Quando os líderes de unidades de negócios começam a colaborar, parece que as pessoas fazem o mesmo dentro de suas organizações. Elas seguem o caminho do seu líder. Comportamento colaborativo juntamente com visibilidade e transparência gera mais comportamento colaborativo. Refiro-me a este período como o “Período de Democracia”.

ABAIXO COM A DEMOCRACIA

A democracia foi muito boa, mas depois de mais quatro meses, parecia que ela havia falhado em eleger o melhor candidato. Um esforço considerável foi gasto para implementar uma funcionalidade de *e-commerce* para o mercado do Leste Europeu. O caso de negócio tinha sido estelar, mas sua candidatura foi suspeita desde o início, e alguns tinham questionado a qualidade dos dados. Após várias tentativas, esse recurso foi selecionado e foi devidamente implementado. Foi uma das maiores funcionalidades processadas através do sistema RRT, e muitas pessoas se envolveram e notaram isso. Dois meses após o lançamento, nosso Diretor de Inteligência de Negócio fez alguma mineração de dados sobre as receitas geradas. Era uma fração do que tinha sido prometido no caso de negócio original, e o período de retorno estimado sobre o esforço despendido foi calculado em 19 anos. Devido à transparência que o Kanban nos ofereceu, muitas partes interessadas tornaram-se conscientes disto, e houve discussão sobre como a preciosa capacidade tinha sido desperdiçada com esta escolha quando uma escolha melhor poderia ter sido feita em vez desta. Este foi o fim do Período da Democracia.

COLABORAÇÃO

O que o substituiu foi notável. Tenha em mente que o comitê de priorização consistia em sua maior parte de funcionários do nível de vice-presidente e diretores da empresa. Eles tinham ampla visibilidade sobre aspectos do negócio que muitos de nós não tínhamos consciência. Assim, no início da reunião, eles começavam a perguntar “Diana, qual é o *lead time* atual de entrega?”. Ela poderia responder “No momento nossa média é de 44 dias para produção”. Então eles faziam uma simples pergunta: “Qual é a mais importante iniciativa tática de negócios nesta empresa 44 dias a partir de agora?”. Poderia haver alguma discussão, mas normalmente havia acordo rápido. “Oh, vai ser o lançamento da nossa campanha européia na conferência em Cannes”. “Ótimo! Quais itens na lista de pendências são necessários para apoiar o evento de Cannes?”. Uma busca rápida poderia produzir uma lista de seis itens. “Existem três *slots* livres nesta semana. Vamos escolher três de seis e nós vamos escolher os outros na próxima semana”. Houve muito pouco debate. Não havia nenhuma barganha ou negociação. A reunião terminava depois de cerca de 20 minutos. Eu passei a me referir a isso como o “Período de Colaboração”. Ele representa o mais alto nível de capital social e confiança entre as unidades de negócios que foi alcançado durante o meu tempo como Diretor Sênior de Engenharia de Software na Corbis.

Mudança Cultural Talvez Seja o Maior Benefício do Kanban

Foi interessante ver esta mudança cultural acontecer e ver como afetou a maioria da empresa enquanto os empregados seguiram a liderança dos seus vice-presidentes e começaram a colaborar mais com os seus colegas de outras unidades de negócios. Esta mudança foi profunda o suficiente para que o recém-nomeado CEO, Gary Shenk, me chamassem para o seu escritório para perguntar se eu tinha uma explicação. Ele me disse que ele tinha observado um novo nível de colaboração e espírito de coleguismo nas categorias sênior da empresa e que as unidades de negócios anteriormente antagônicas pareciam estar interagindo muito melhor. Ele sugeriu que o processo da RRT tinha algo a ver com isso e perguntou se eu tinha alguma explicação para ele. Estou certo que eu não era tão articulado como agora, escrevendo este capítulo dois anos mais tarde, mas eu o convenci de que nosso sistema kanban tinha reforçado bastante a colaboração e junto com isso o nível de capital social entre todos os envolvidos.

Os efeitos colaterais culturais que agora reconhecemos ser do Kanban com K maiúsculo foram bastante inesperados e de muitas maneiras contra-intuitivos. Ele perguntou: “Por que não estamos fazendo todos os nossos principais projetos desta forma?” Porque não? Então combinamos de implementar Kanban no portfólio de grandes projetos. O fizemos porque Kanban permitiu uma cultura *kaizen*, e aquela mudança cultural foi tão desejável que o custo de alterar as várias mecânicas de priorização, agendamento, emissão de relatórios e entrega que resultaria da implementação Kanban foi considerado como um preço que valia a pena pagar.

Takeaways

- ❖ *Kaizen* significa “melhoria contínua”.
- ❖ Uma cultura *kaizen* é aquela onde os indivíduos se sentem com poder, agem sem medo, espontaneamente se unem, colaboram e inovam.
- ❖ Uma cultura *kaizen* possui um alto grau de capital social e confiança entre os indivíduos, independente de seu nível na hierarquia corporativa.
- ❖ Kanban fornece transparência tanto no trabalho quanto no processo através do qual o trabalho flui.
- ❖ A transparência do processo permite que todos os interessados vejam os efeitos de suas ações ou falta de ação.
- ❖ Indivíduos estão mais propensos a oferecer seu tempo e colaborar quando eles podem ver os efeitos de suas ações.
- ❖ Limites de *WIP** do Kanban possibilitam o comportamento “pare a linha”.
- ❖ Limites de *WIP* do Kanban incentivam o comportamento de enxame[†] para resolver problemas.
- ❖ Uma maior colaboração através do comportamento de enxame sobre os problemas e interação com interessados externos aumenta o nível de capital social na equipe e a confiança entre seus membros.
- ❖ Limites de *WIP* do Kanban e classes de serviço dão poder aos indivíduos de puxar trabalho e tomar decisões de priorização e agendamento sem supervisão ou direção de um superior.
- ❖ Maiores níveis de autonomia aumentam o capital social entre os trabalhadores e gerentes.
- ❖ Comportamento colaborativo pode se espalhar de forma viral.
- ❖ Indivíduos buscarão a liderança em seus gerentes seniores. Comportamento colaborativo e de coleguismo entre os líderes seniores afetarão o comportamento de toda a força de trabalho.

* N. de T.: Sigla de *work-in-progress*.

† N. de T.: *swarming*, no original

❖ PARTE TRÊS ❖

IMPLEMENTANDO KANBAN

❖ CAPÍTULO 6 ❖

Mapeando a Cadeia de Valor

Kanban é uma abordagem que conduz mudança no processo existente pelo aperfeiçoamento. A essência de se começar com Kanban é mudar o mínimo possível. Você deve resistir a tentação de mudar o fluxo de trabalho, nomes dos cargos, papéis e responsabilidades, e práticas de trabalho específicas. Tudo o que os membros da equipe e outros parceiros, participantes, e *stakeholders* derivaram da sua auto-estima, orgulho profissional, e ego devem permanecer inalterados. O alvo principal da mudança será a quantidade do *WIP* e a interface e interação com as partes do negócio da cadeia de valor. Então você deve trabalhar com a sua equipe para mapear a cadeia de valor como ela existe. Tente não mudá-la ou inventá-la numa maneira idealizada.

Em algumas situações políticas, pode existir um processo oficial que não está sendo seguido. Quando você tenta mapear a cadeia de valor, sua equipe insistirá que você re-documente o processo oficial, e não o processo sendo usado. Você deve resistir a isso e insistir que a equipe mapeie o processo que eles realmente usam. Sem isso será impossível usar uma parede de cartões como uma ferramenta de visualização de processo porque os membros da equipe só podem usar a parede de cartões se ela refletir o que eles realmente fazem.

Definindo um Ponto Inicial e Final de Controle

É necessário decidir onde começar e onde finalizar a visualização do processo, e fazendo isso, definir os pontos de interface com os parceiros envolvidos antes do ponto inicial e após o final. É importante lidar com esse estágio inicial da implementação de Kanban, visto que escolhas erradas podem levar a falhas. Equipes bem sucedidas tendem a adotar a visualização do fluxo de trabalho com cartões e limites de *WIP* dentro da sua esfera política de controle, e negociar uma nova maneira de interagir com os parceiros imediatos fora dessa esfera. Por exemplo, se você controla o departamento de engenharia ou desenvolvimento de software e tem controle ou influência sobre análise, projeto, teste, e codificação; então mapeie esta cadeia de valor e negocie novos estilos de interação com os parceiros de negócio envolvidos antes da cadeia de valor que fornecem requisitos, priorização, e gerenciamento de portfólio, e aqueles envolvidos após a cadeia de valor como operação de sistemas, ou departamento de produção-mantenção. Colocando os limites dessa maneira, você está solicitando à sua equipe apenas a adotar o comportamento de limitar o *WIP*. Você não está solicitando às equipes envolvidas antes e após a cadeia de valor a mudar a maneira como elas trabalham. Você não está solicitando a eles interagir com você de uma maneira diferente – e sim a interagir de uma maneira compatível com o sistema puxado que você quer implementar.

Tipos de Item de Trabalho

Uma vez que você tenha selecionado o ponto inicial no fluxo de trabalho ou cadeia de valor, identifique os tipos de trabalho que chegam nesse ponto e quaisquer outros que existam dentro do fluxo de trabalho que precisarão ser limitados. Por exemplo, defeitos são como um tipo de trabalho que existe dentro do fluxo de trabalho. Você pode identificar outros tipos de trabalho centrados no desenvolvimento, como *refactoring*^{*}, manutenção de sistema, atualização de infra-estrutura e trabalhos relacionados. Para trabalho na entrada, você pode ter tipos como *User Story*[†] ou *Use Case*[‡] ou Requisito Funcional ou *Feature*. Em alguns casos, os tipos da entrada podem ser hierárquicos, como *Epic*[§], uma coleção de *user stories*.

* N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: refatoração.

† N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: Estória de Usuário.

‡ N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: Caso de Uso.

§ N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: Épica.

Tipos típicos de itens de trabalho vistos em equipes que adotam Kanban têm incluído, mas não são limitados, aos seguintes.

- Requisito
- *Feature*
- *User Story*
- *Use Case*
- Solicitação de Mudança
- Defeito em Produção
- Manutenção
- *Refactoring*
- Defeito
- Sugestão de Melhoria
- *Blocking Issue*

É útil relacionar tipos de item de trabalho a sua fonte, como Requisito Regulatório, Solicitação da Área de Vendas, Requisito de Planejamento Estratégico, e assim por diante. Usar uma convenção para a nomenclatura que faz com que a fonte da solicitação seja transparente fornece um contexto adicional e permite que o sistema evolua para servir múltiplos clientes.

Tipos de item de trabalho tendem a ser definidos pela fonte do trabalho, o fluxo do trabalho, ou o tamanho do trabalho. Por exemplo, os PTCs do exemplo da Microsoft no capítulo 4 têm um fluxo de trabalho diferente, apesar de a fonte ser a mesma das Solicitações de Mudança. Não faz sentido ter dois sistemas kanban diferentes para os dois tipos. A mesma equipe faz o trabalho. É fácil visualizar os tipos usando cores diferentes de *tickets* ou linhas diferentes (raias) numa parede de cartões.

Desenhando uma Parede de Cartões

É típico desenhar paredes de cartão para mostrar as atividades que acontecem ao trabalho em vez de funções específicas ou descrições de trabalho. Frequentemente há uma sobreposição forte entre uma função e uma atividade; por exemplo, analistas realizam análise. No entanto, a convenção com o uso de Kanban em projetos de software ao longo dos últimos anos tem sido modelar o trabalho e não os trabalhadores, as funções, ou a interação entre as funções.

Antes de desenhar uma parede de cartão para visualizar o fluxo de trabalho, faz sentido fazer um rascunho ou modelá-la. Figura 4.4 (página 42) mostra um modelo muito formal, usando uma notação de gráfico de estados, do fluxo de trabalho desejado, com filas incluídas para solicitação de mudança e mudanças de texto em produção processadas pela equipe de manutenção do XIT na Microsoft. Você pode encontrar uma abordagem menos formal perfeitamente adequada. Um desenho com atores, similar àquele apresentado no capítulo 4, ou um gráfico de fluxo ou seu equivalente pode ser suficiente.

Uma vez que você entendeu o seu fluxo de trabalho esboçando-o ou modelando-o, comece a definir uma parede de cartões desenhando colunas num quadro que representa as atividades realizadas, na ordem em que elas são realizadas (Figura 6.1). Ao desenhar as colunas iniciais, faz sentido riscá-las com um marcador. No entanto, com o uso as linhas serão apagadas. Durante as primeiras semanas você perceberá que vai querer fazer algumas poucas mudanças no fluxo de trabalho, então faz sentido continuar a usar um marcador que pode ser apagado. No entanto, haverá um ponto que você vai querer algo mais permanente. Rolos de fita de vinil estreitos estão disponíveis em lojas de escritório, especificamente projetados para trabalho de precisão em quadros brancos, como ilustrado na Figura 6.2. Tornou-se comum na Corbis delinear colunas e linhas numa parede de cartões usando essa fita. Esta prática é agora largamente adotada, com equipes usando várias classes e tamanhos de fita para marcar linhas e colunas.

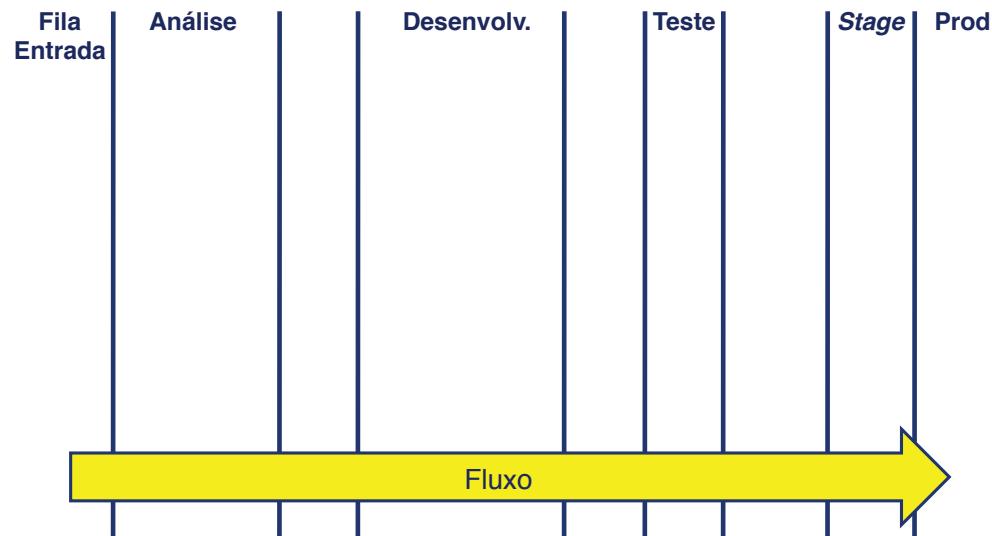


Figura 6.1 Esboço de um fluxo de trabalho numa parede de cartões (da esquerda para a direita)



Figura 6.2 Fita de precisão para quadro branco

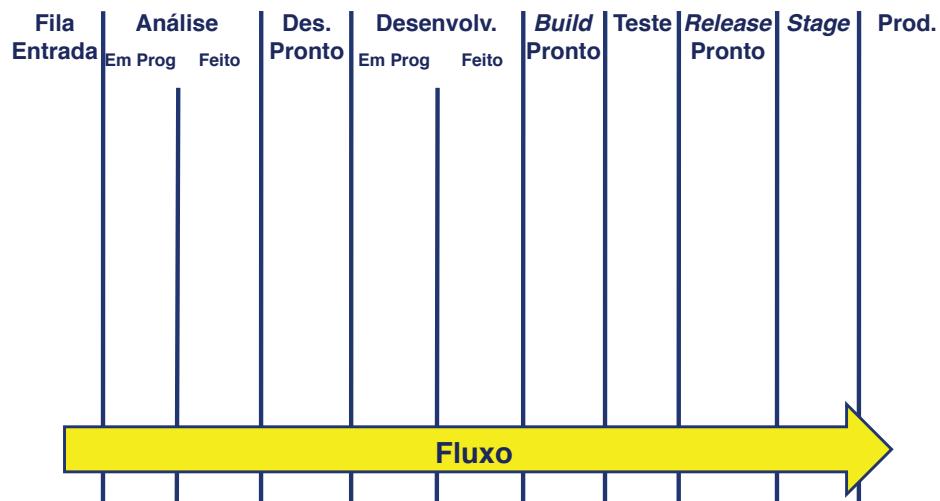


Figura 6.3 Fluxo de trabalho com *buffers* e filas

Note que para os passos da atividade é necessário modelar o que está em progresso e o trabalho finalizado; por convenção isto é feito dividindo-se a coluna.

Depois, adicione uma fila de entrada e qualquer atividade de entrega após a cadeia de valor que você deseja visualizar, como apresentado na Figura 6.3.

Finalmente, adicione qualquer *buffer* ou fila que você acredita ser necessário. Há es-
colas de pensamento diferentes em relação a isto, e ele é realmente um tópico avançado.
Uma discussão completa sobre onde colocar *buffers* e como dimensioná-los está além
do escopo desse livro, então será suficiente nesse momento descrever duas abordagens
populares.



Figura 6.4 Parede de cartões ilustrando o uso de cartões com formato de diamante no topo das colunas de fila ou *buffer* (cortesia da Liquidnet Holdings, Inc.)

A primeira escola de pensamento diz para não tentarmos adivinhar a localização do gargalo ou a fonte de variabilidade que precisará de um *buffer*. Em vez disso, implemente o sistema e espere que o gargalo se revele, então faça mudanças para introduzir um *buffer* (Figura 6.4).

Uma variante desta abordagem sugere que inicialmente devem ser atribuídos limites de *WIP* razoavelmente fracos para que a variabilidade, desperdício, e gargalos não tenham um impacto significante na primeira implementação do sistema puxado. Isto é discutido em mais detalhe no capítulo 10, e depois nos capítulos 17 e 19.

Outra escola de pensamento usa uma abordagem diferente. Ela sugere que em vez de implementar limites de *WIP* frouxos para evitar uma introdução desafiadora do sistema, cada estágio deve ser *buffered*, e as atividades devem ter limites bem apertados. Gargalos e variabilidade se revelarão assim que os *buffers* ficarem cheios. Mudanças pequenas e simples podem então ser tomadas para reduzir o tamanho dos *buffers* e então eventualmente você pode eliminar *buffers* desnecessários.

Durante a escrita deste livro, não houve evidência suficiente que sugerisse qual abordagem é a melhor.

Algumas equipes adotaram uma convenção de mostrar colunas de *buffers* e filas usando um cartão com uma rotação de 45 graus. Isto fornece um indicador de

visualização forte sobre quanto trabalho está fluindo em vez de estar enfileirado em qualquer instante no tempo. Isto permite que a equipe e outros *stakeholders* “vejam”, literalmente, a quantidade do custo econômico (ou desperdício) no sistema.

Análise da Demanda

Para cada tipo de trabalho identificado, você deve fazer um estudo da demanda. Se você tem dados históricos, use-os para fazer um estudo quantitativo. Se você não tem, então uma análise derivada de forma subjetiva será suficiente. No exemplo da Microsoft XIT no capítulo 4, por exemplo, havia dois tipos de trabalho, Solicitações de Mudança e Mudança de Texto em Produção - *Production Text Changes* (PTCs). Indiscutivelmente, as Solicitações de Mudança deveriam ser quebradas em dois tipos, Defeitos em Produção e Solicitações de Mudança (para nova funcionalidade). Se eu estivesse apoiando essa equipe hoje, eu recomendaria que ela acompanhasse quatro tipos de trabalho no total: Solicitações de Mudança, Defeitos em Produção, Mudanças de Texto em Produção, e Defeitos (ou defeitos escapados).

Para cada um destes tipos deveríamos estudar a demanda. As demandas para PTCs vêm em rajadas. Pode não existir PTCs por seis semanas, e numa única semana pode haver uma explosão de talvez dez chegando de uma única vez. PTCs eram pequenos e rápidos de implementar. Isto significava que o seu impacto não era forte. É difícil desenhar um sistema para lidar com uma demanda intermitente como esta. Se PTCs representassem um esforço significante, seria necessária a construção de um espaço considerável no sistema para lidar adequadamente com os PTCs para não impactar fortemente na previsibilidade das Solicitações de Mudança.

Solicitações de Mudança, por outro lado, chegavam num ritmo muito mais constante. Enquanto a sua chegada era de natureza estocástica, a demanda era relativamente constante, de aproximadamente cinco a sete novas requisições por semana. Seria possível colocar num gráfico a taxa de chegada dos PTCs e colocar a demanda para entender a taxa media de chegada e a propagação da variação. O sistema kanban pode então ser projetado e recursos alocados adequadamente para lidar com esta demanda.

Alguns tipos de item de trabalho exibem uma demanda sazonal, como requisitos regulatórios. Novos impostos legislativos afetam sistemas financeiros e de folha de pagamento de maneira sazonal. Em um caso que me deparei, o departamento de TI de uma equipe de corrida automobilística recebia mudanças regulatórias do corpo direutivo do esporte no início de cada temporada de corrida. Eles podiam também receber requisitos regulatórios durante a temporada, mas o volume sobre a pré-temporada era significativamente maior, visto que os principais regulamentos da corrida eram

alterados de um ano para o outro. É importante entender esta demanda para que o projeto do sistema kanban possa ser ajustado para lidar com a demanda para tipos diferentes de trabalho.

Alocando Capacidade de Acordo com a Demanda

Uma vez que você tenha o entendimento da demanda, você pode decidir como alocar capacidade dentro do sistema kanban para lidar com a demanda. O exemplo na Figura 6.5 mostra três raias, uma para cada tipo de trabalho, nomeadas, *change requests*^{*}; *internal maintenance work*[†], como *refactoring* de código; e *production text changes*[‡]. A alocação é 60 por cento para *change requests*, 10 por cento para trabalho de *refactoring* de código, e 30 por cento para *production text changes*. Dada uma análise da demanda que mostra que *production text changes* chegam a rajadas, a alocação apresentada sugere que um espaço significante está sendo reservado para lidar urgentemente com *production text changes* que chegam sem impactar datas de entrega de outros trabalhos. A alocação de capacidade deve estar alinhada ao perfil de risco. Se, por exemplo, é aceitável que o desempenho de entrega na data caia quando *production text changes* chegam, e que *lead times* das requisições de mudança sejam maiores e menos prevíveis, a alocação poderia ser diferente. Talvez 85 por cento para *change requests*, 10 por cento para *maintenance*, e 5 por cento para *production text changes*. Outra escolha

Fila Entrada	Análise Em Prog Feito	Des. Pronto	Desenvolv. Em Prog Feito	Build Pronto	Teste	Release Pronto	...
Solicitação Mudança 60%	⋮		⋮				
Manutenção 10%	⋮		⋮				
Mudança de Texto em Produção 30%	⋮		⋮				

Figura 6.5 Quadro Kanban com raias, indicando capacidade da alocação

* N. de T.: Termo não traduzido. Tradução: solicitações de mudança.

† N. de T.: Termo não traduzido. Tradução: trabalho de manutenção interno.

‡ N. de T.: Termo não traduzido. Tradução: mudanças de texto em produção.

poderia ser deixar uma raia para *production text changes*, mas não alocar capacidade para ela, e adotar uma política de exceder o limite do trabalho-em-progresso quando uma rajada de *production text changes* chegar. Esta política eliminará espaço e produzirá uma saída econômica ideal durante a operação normal. No entanto, quando uma explosão de *production text changes* chegar, outros trabalhos podem ser fortemente impactados no *lead time* e na previsibilidade. Esta foi a escolha feita em muitos exemplos reais do capítulo 4; não havia capacidade sobressalente reservada para lidar com *production text changes*.

Mais tarde, quando discutirmos sobre atribuição de limites para trabalho-em-progresso, usaremos esta informação de alocação para atribuir limites específicos para filas em cada raia.

Anatomia de um Cartão de Item de Trabalho

Cada cartão visual que representa um pedaço discreto de trabalho de valor para o cliente contém muitas informações. O projeto do cartão é importante. A informação nos cartões deve facilitar o sistema puxado e dar poder aos indivíduos para tomarem suas próprias decisões ao puxar. A informação num ticket pode variar por tipo de trabalho ou por classe de serviço (discutido no capítulo 11).

No exemplo da Figura 6.6, o número no topo à esquerda é o número eletrônico de acompanhamento usado para identificar unicamente o item e para ligá-lo a sua

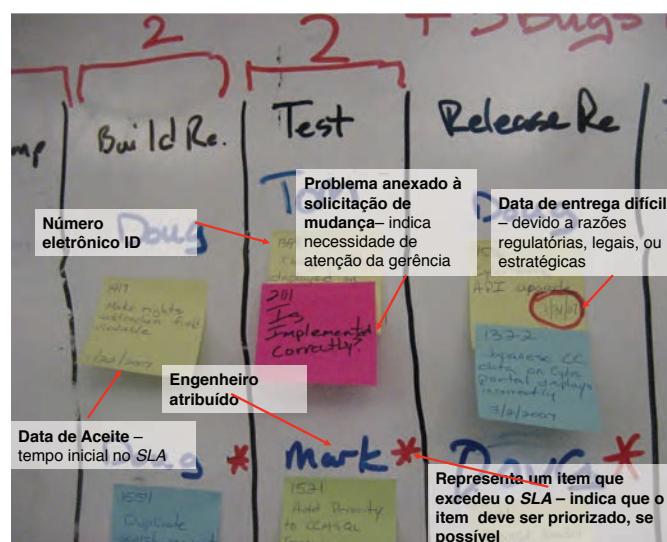


Figura 6.6 Zoom de uma parede de cartões apresentando a anatomia dos cartões de item de trabalho

versão eletrônica no sistema de acompanhamento. O título do item é escrito no meio do cartão. A data que o item entrou no sistema é escrita abaixo no lado esquerdo, que tem um propósito duplo: Facilita o enfileiramento *first-in, first-out (FIFO)* para classe de serviço padrão, e permite que os membros da equipe vejam quantos dias expiraram em relação ao acordo de nível de serviço (descrito no capítulo 11). Para itens de classe de serviço de data de entrega fixa, a data de entrega requerida é escrita abaixo do lado direito do ticket.

No exemplo da Figura 6.6, outras informações são mostradas fora do ticket. Uma estrela indica que o item está atrasado em relação ao *lead time* do nível de acordo de serviço. Eu tenho visto isto sendo feito mais recentemente anexando uma etiqueta na parte de cima à direita do ticket. O nome da pessoa atribuída é também escrito em cima do ticket. Como a pessoa atribuída mudará à medida que o ticket fluir através do quadro, é indesejável escrever o nome no ticket. No entanto, implementações mais recentes apresentam pequenas etiquetas presas aos itens, o uso de imãs (quando o quadro branco é magnético), e etiquetas ou imãs com os avatares dos membros da equipe. Personagens do *South Park* são escolhas populares para os avatares. Qualquer mecanismo que permite aos membros da equipe e a gerência imediata visualizarem rapidamente quem está trabalhando é suficiente.

Como regra geral, o projeto do ticket usado para representar um pedaço individual de trabalho deve possuir informação suficiente para facilitar decisões de gerenciamento de projeto, como qual deve ser o próximo item a ser puxado, sem a intervenção da direção ou gerência. A ideia é delegar poder aos membros da equipe com transparéncia do processo, objetivos do projeto, e informação sobre risco. À medida que você descobre mais sobre classes de serviço e níveis de acordo de serviço, você descobrirá que Kanban facilita um mecanismo poderoso de auto-organização e gerenciamento de risco. Igualmente, delegando poder aos membros da equipe para fazer seu próprio agendamento e priorizar decisões, Kanban mostra respeito aos indivíduos e uma confiança no sistema (ou projeto do processo). Um cartão de item de trabalho bem projetado é um elemento chave para uma cultura de alta confiança e uma organização *Lean*.

Acompanhamento Eletrônico

Acompanhamento eletrônico tem sido uma característica de um sistema kanban em desenvolvimento de software desde que ele foi primeiramente introduzido em 2004. Isto é, no entanto, opcional. Para equipes distribuídas geograficamente, ou para aquelas que possuem políticas que permitem que os membros da equipe trabalhem nas suas casas uma ou mais vezes por semana, o acompanhamento eletrônico é essencial. O

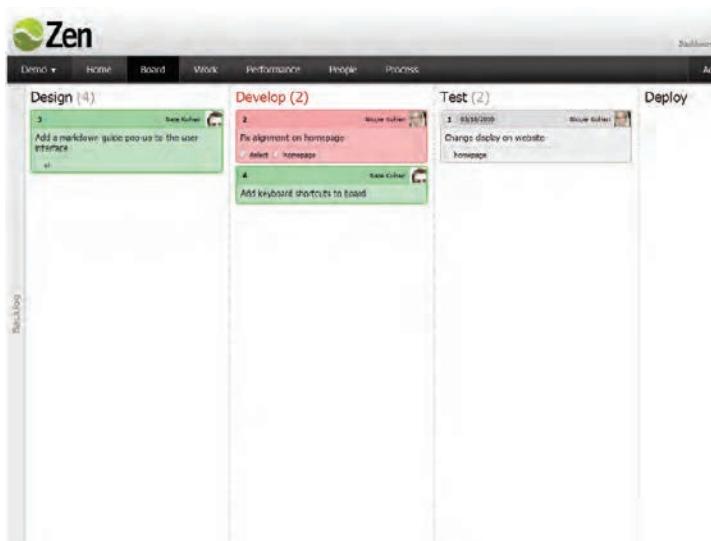


Figura 6.7 Screen shot da ferramenta de acompanhamento eletrônico *Agile Zen*

acompanhamento eletrônico pode ser realizado com sistemas de acompanhamento de *tickets* e itens de trabalho básicos, como Jira, Microsoft *Team Foundation Server*, Fog Bugz, e HP Quality Center. No entanto, um sistema mais poderoso permitirá que você visualize o acompanhamento dos itens de trabalho como se fosse numa parede de cartões.

Durante a escrita deste livro, uma quantidade de ferramentas *web-based* e *application-based* estavam surgindo no mercado para fornecer acompanhamento eletrônico usando quadros visuais que simulam paredes de cartões com colunas, limites de *WIP*, e outros aspectos essenciais de Kanban. Estas incluem, mas não estão limitadas a: Lean Kit Kanban, Agile Zen, Target Process, Silver Catalyst, RadTrack, Kanbanery, VersionOne, Greenhopper para Jira, Flow.io, e alguns projetos adicionais *open-source* que adicionam interfaces Kanban a ferramentas como *Team Foundation Server* e FogBugz. A Figura 6.7 mostra um exemplo da AgileZen.

O acompanhamento eletrônico é necessário para equipes que aspiram níveis mais altos de maturidade organizacional. Se você prevê a necessidade para gerenciamento quantitativo, desempenho do processo organizacional (comparar o desempenho através de sistemas kanban, equipes ou projetos), e/ou análise e resolução de causa raiz (análise de causa raiz baseada em dados estatísticos), você usará uma ferramenta eletrônica desde o início.

Atribuindo Limites de Entrada e Saída

Alinhe o projeto do sistema kanban e a parede de cartões com uma decisão tomada antecipadamente para delimitar o limite de controle do WIP. É provável que os seus parceiros envolvidos antes e após a sua cadeia de valor solicitem visualizar o seu trabalho na parede de cartões. No entanto, é melhor fornecer transparência dentro do seu trabalho primeiramente e esperar que os outros solicitem fazer parte da sua iniciativa Kanban.

No exemplo mostrado na Figura 6.8, a fila de entrada é designada “E.R.” para “Engineering Ready.” Faz sentido atribuir o ponto de entrada nesse momento do ciclo de vida porque o departamento de análise de negócio que atua antes da cadeia de valor reporta através de uma parte diferente da estrutura da organização. Há pouca confiança ou colaboração entre os gerentes através dos dois grupos. A fila de entrada, portanto, foi reabastecida do *backlog* de requisitos gerados pelo departamento de análise de negócio.

Neste exemplo, a entrega da cadeia de valor é a implantação em produção. Uma vez que o software é implantado e entregue para o departamento de operações de redes e sistemas para manutenção e suporte diário, ele é considerado fora do escopo.



Figura 6.8 Apresentando a fila de entrada *Engineering Ready* (ER)

Lidando com Concorrência

Uma concorrência comum ao desenhar uma parede de cartões para um sistema kanban é um processo no qual duas ou mais atividades podem ocorrer concurrentemente, por

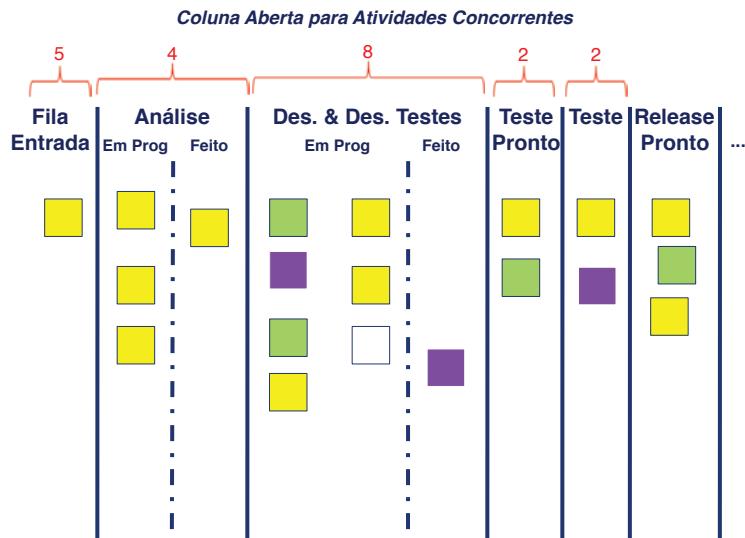


Figura 6.9 Coluna aberta para atividades concorrentes

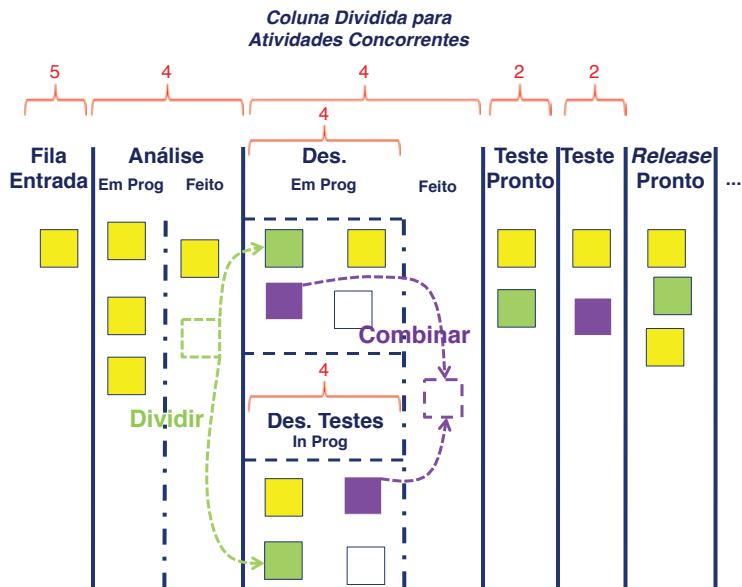


Figura 6.10 Coluna dividida para atividades concorrentes

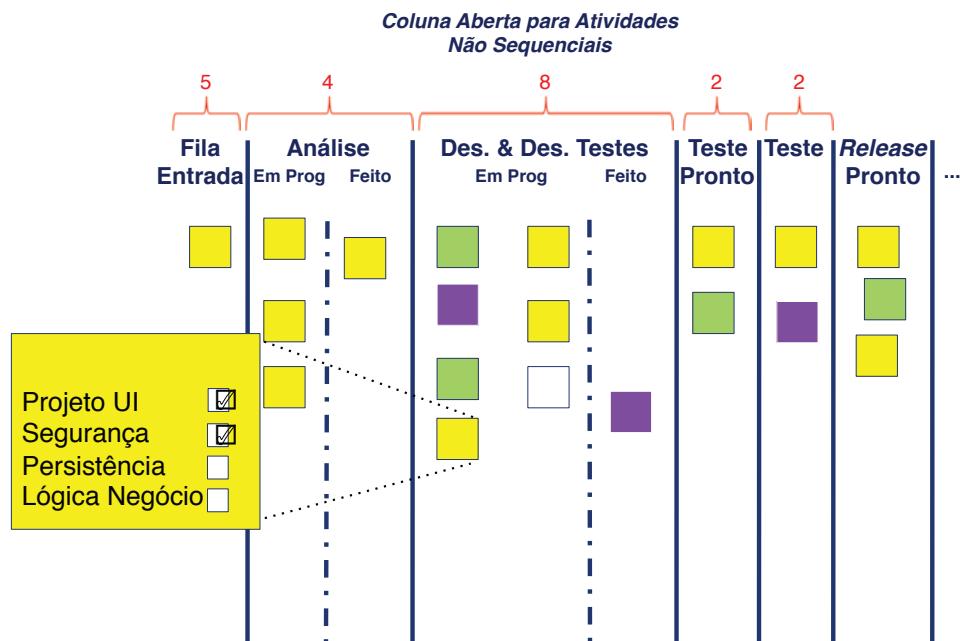


Figura 6.11 Coluna aberta para atividades não sequenciais

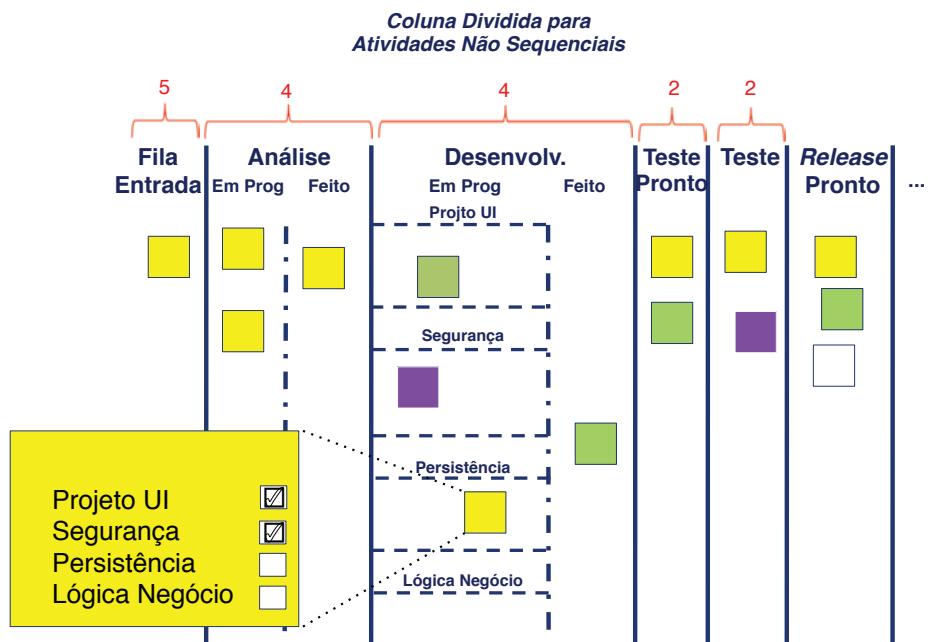


Figura 6.12 Coluna dividida para atividades não sequenciais

exemplo, desenvolvimento de software e teste de software. Há dois padrões básicos para lidar com esta situação. Um é não modelá-la de maneira alguma; apenas deixe uma coluna onde as duas atividades podem ocorrer juntas (Figura 6.9). Isto é simples, mas muitas equipes não optaram por esse padrão. Algumas equipes têm adaptado esse modelo usando cores de *tickets* para mostrar atividades diferentes. A outra opção é dividir o quadro verticalmente em duas (ou mais) partes (Figura 6.10).

Neste exemplo, é necessário algum mecanismo de marcação para ligar itens acima no quadro com os itens abaixo no quadro. Isto pode envolver, por exemplo, uma referência cruzada na parte de cima à direita do ticket ao seu item associado. Num bom sistema de acompanhamento eletrônico é possível ligar itens relacionados como atividades de desenvolvimento e teste.

Lidando com atividades Desordenadas

Particularmente em trabalhos de alta inovação e experimentais, haverá muitas atividades que acontecem num pedaço de trabalho de valor para o cliente, mas estas atividades não precisam acontecer em qualquer ordem em particular. Nestas circunstâncias, é importante perceber que Kanban não deve forçar você a completar as atividades numa ordem determinada. O que é importante ao modelar seu sistema kanban é que ele deve refletir a maneira que o trabalho real é feito.

Há algumas estratégias para o problema de múltiplas atividades desordenadas. A primeira é similar ao lidar com concorrência: Tenha simplesmente uma única coluna como um balde de atividades e não acompanhe explicitamente no quadro qual delas é finalizada.

A segunda, e potencialmente a escolha mais poderosa, é modelar as atividades de maneira similar a atividades concorrentes. Neste projeto, como mostrado na Figura 6.11, os *tickets* devem se mover verticalmente para cima e para baixo nas colunas à medida que eles são puxados para cada uma das atividades específicas. Podem-se visualizar quais atividades foram finalizadas para cada item modificando-se o projeto do ticket para se ter uma caixa pequena para cada atividade. Quando uma atividade é finalizada, a caixa pode ser preenchida para visualmente sinalizar que o item está pronto para ser puxado para outra atividade na mesma coluna. Se todas as caixas são preenchidas, o item está pronto para ser puxado para a próxima coluna no quadro, ou pode ser movido para a coluna “done” (Figura 6.12).

Takeaways

- ❖ Decida os limites externos do sistema kanban. É frequentemente melhor limitar isto para o escopo imediato de controle político. Não force visualização, transparência, e limites de *WIP* em qualquer departamento que não colabora voluntariamente.
- ❖ Modele a parede de cartões alinhada às decisões tomadas limitando *WIP* e visualização do trabalho.
- ❖ Defina tipos de item de trabalho e modele como os seus trabalhos fluem. Alguns tipos podem não precisar de todos os passos no fluxo.
- ❖ Projete o cartão do item de trabalho para que ele tenha informação suficiente para facilitar auto-organização para puxar, e permitir que os membros da equipe tomem decisões de alta qualidade considerando o risco, baseado no tipo do item de trabalho, acordos de nível de serviço, e classes de serviço.
- ❖ Use um sistema de acompanhamento eletrônico se a equipe é distribuída, tem alguma política de trabalho em casa, ou aspira um comportamento de alta maturidade que requer informação quantitativa que um sistema eletrônico pode fornecer.
- ❖ Quando apropriado, discuta métodos para lidar com concorrência de atividade e escolha como modelá-las e visualizá-las.
- ❖ Quando apropriado, discuta métodos para lidar com atividades que não precisam seguir um fluxo ordenado específico e escolha como modelá-las e visualizá-las.

❖ CAPÍTULO 7 ❖

Coordenação com Sistemas Kanban

Controle Visual e Puxado

Quando as pessoas falam sobre Kanban, a forma mais popular de coordenação que vem à mente é a parede de cartões. Tipicamente, os limites de trabalho-em-progresso são desenhados no quadro no topo de cada coluna ou através de uma extensão das colunas. O ato de puxar é sinalizado se o número de cartões na coluna é menor do que o limite indicado. Na Figura 7.1, podemos ver que o limite escrito acima de *Specification** é quatro itens. No entanto, há apenas três cartões naquela coluna. Como $4 - 3 = 1$, isto sinaliza que podemos puxar um item para *Specification* (o departamento de análise de sistemas) da fila de entrada, *Engineering Ready*[†]. Por sua vez, a fila de entrada tem um limite de cinco. Há atualmente apenas dois itens restando naquela fila. Quando puxamos um item para *Analysis*[‡], haverá apenas um item restando ($5 - 1 = 4$). Isto sinaliza que podemos priorizar quatro novos itens para a fila de entrada na próxima reunião de priorização.

Quando uma equipe decide puxar um item, ela pode escolher qual item puxar tendo como base a informação visual disponível, como o tipo do item de trabalho, a classe de serviço, a data da entrega (se

* N. de T.: Tradução: Especificação.

† N. de T.: Tradução: Engenharia Finalizada/Pronta.

‡ N. de T.: Tradução: Análise.

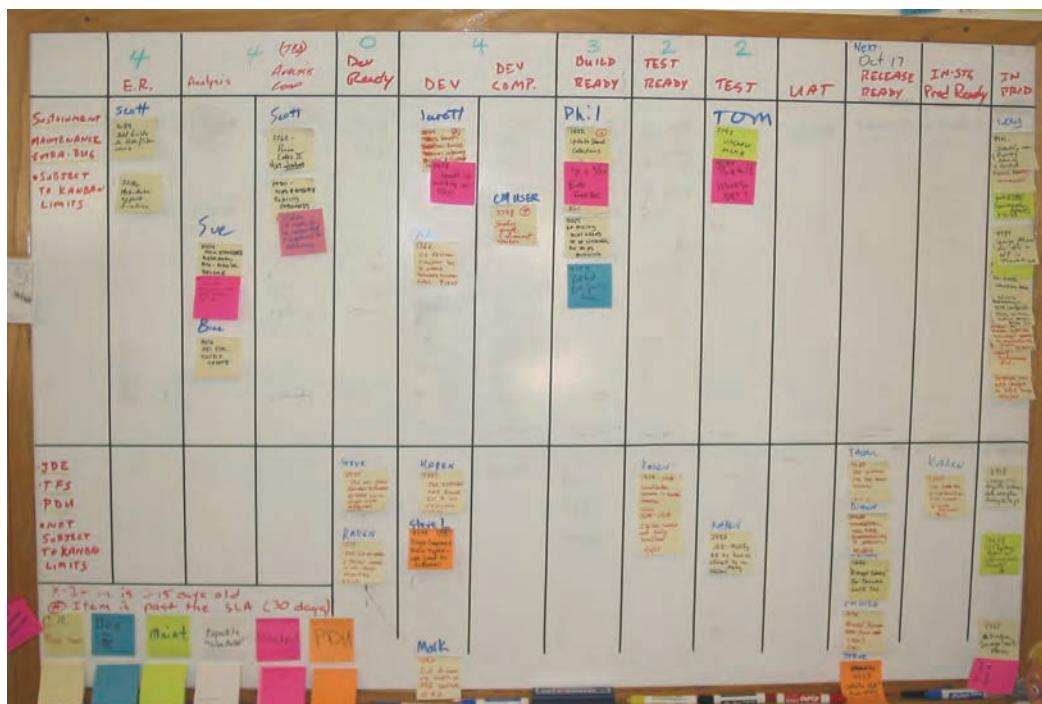


Figura 7.1 Mostrando limites kanban acima na parede de cartões

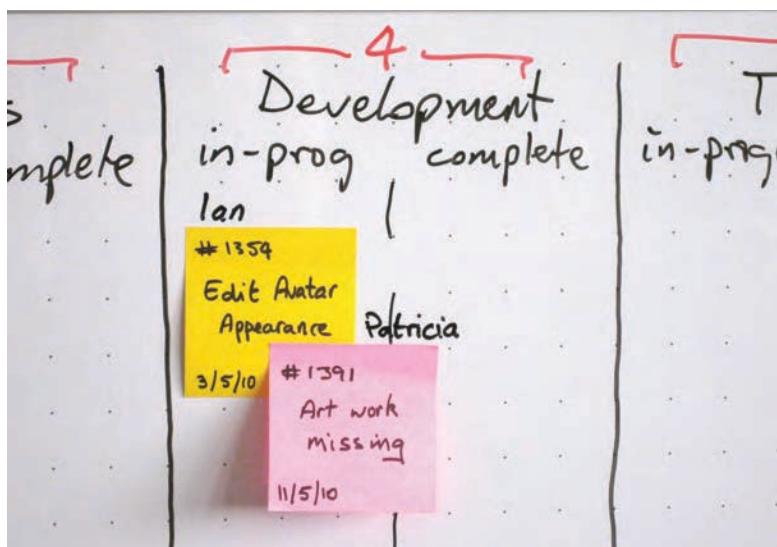


Figura 7.2 Zoom da parede de cartões mostrando a anatomia dos detalhes do cartão

aplicável), e a idade do item de trabalho. Políticas para puxar relacionadas às classes de serviço serão discutidas no capítulo 11.

A Figura 7.2 mostra de perto algumas notas que representam itens de trabalho na nossa parede de cartões. As cores são utilizadas para comunicar uma combinação de tipos de item de trabalho e classes de serviço. O nome do dono ou membro da equipe atribuído é escrito acima do cartão. Algumas equipes gostam de usar notas adesivas menores com nomes, ou pequenos avatares colados ao item de trabalho para mostrar quem está trabalhando nele. Isto permite que todos na equipe vejam quem está trabalhando naquilo.

Na Figura 6.6, o número de acompanhamento eletrônico é mostrado no topo do lado esquerdo do post-it. A data que um item entrou na fila de entrada é apresentada na parte de baixo do lado esquerdo do post-it. A idade do item pode ser deduzida a partir desta data. Se um item é uma classe de serviço que tem uma data de entrega garantida, isso é apresentado na parte de baixo do lado direito do post-it. Se um item está atrasado, isso é representado com um asterisco vermelho no topo do post-it do lado direito. Se algo está bloqueado, um ticket rosa é anexado ao item bloqueado. No exemplo da Figura 7.2, um problema é um item de trabalho de primeira classe, uma vez que ele possui seu próprio número eletrônico de acompanhamento, uma data na qual ele entrou no sistema, e um membro da equipe com atribuição para resolvê-lo apresentado na parte de cima dele.

Este esquema é idiossincrático para a primeira implementação kanban na Corbis. Sua implementação irá certamente diferir. No entanto, você provavelmente vai querer capturar visualmente o membro da equipe atribuído, a data inicial, o número de acompanhamento eletrônico, o tipo do item de trabalho, a classe de serviço, e alguma informação de *status*, como se o item está atrasado ou não. O objetivo é visualmente comunicar informação suficiente para tornar o sistema auto-organizável e auto-*expediting* no nível da equipe. Como um mecanismo de controle visual, o quadro kanban deve permitir que os membros da equipe puxem trabalho sem direção do seu gerente.

Acompanhamento Eletrônico

Como uma alternativa, ou, como um complemento, para a parede de cartões, um sistema eletrônico é frequentemente utilizado para acompanhar trabalho num sistema kanban. Algumas das ferramentas disponíveis para isto são listadas no capítulo 6. Para uma lista mais atualizada, visite a página na web da *Limited WIP Society*, www.limitedwipsociety.org.

Com a minha equipe, nós implementamos nossa própria aplicação, *Digital Whiteboard* (veja Figura 7.3) , a partir do *Team Foundation Server*. No caso de estudo

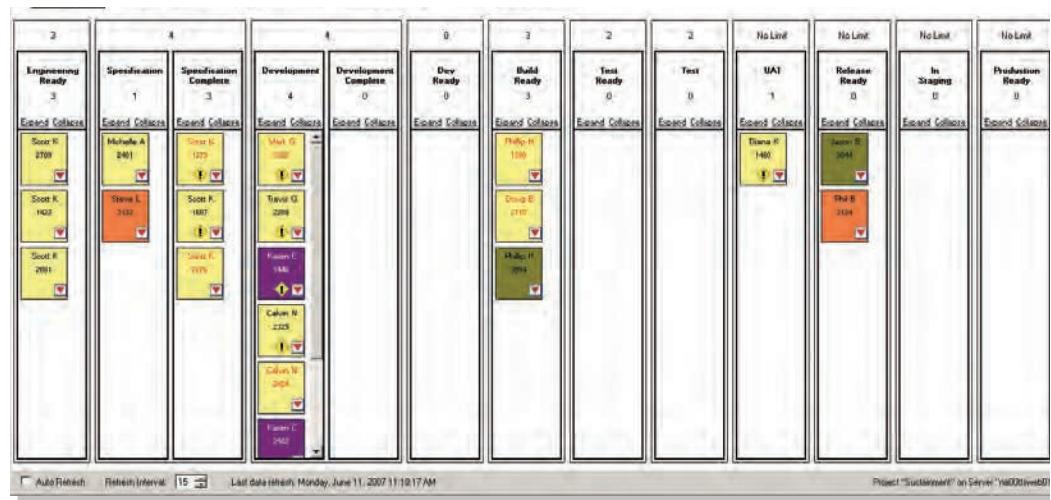


Figura 7.3 Quadro branco digital utilizado na Corbis

do capítulo 4, o acompanhamento eletrônico foi feito utilizando uma ferramenta interna da Microsoft chamada *Product Studio*. Ela foi precursora da *Team Foundation Server*, e desde 2005 a Microsoft tem usado *Team Foundation Server* para o seu próprio acompanhamento interno de projetos de desenvolvimento.

A aplicação mostrada na Figura 7.3 apresenta os limites kanban agrupados acima nas colunas. Ela é capaz de mostrar visualmente quando um limite kanban é excedido. Ela também apresenta a quantidade de itens de *status* para cada item de trabalho, incluindo ícones diferentes que mostram se um item está atrasado ou está bloqueado com um problema.

O acompanhamento eletrônico é importante para sistemas kanban porque ele permite muitas coisas que não são viáveis com uma parede de cartões simples. O acompanhamento eletrônico permite coleta de dados que podem ser usados para gerar métricas e relatórios para o dia a dia do gerenciamento e para retrospectivas, por exemplo, nas revisões operacionais mensais.

Reuniões *Standup* Diárias

Reuniões *standup** são um elemento comum nos processos de desenvolvimento Ágeis. Elas acontecem tipicamente pela manhã antes que o trabalho comece e seguem geral-

* N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: em pé.

mente um formato pré-acordado. Uma reunião *standup* típica acontece para uma única equipe de aproximadamente doze pessoas – usualmente em torno de seis pessoas. O formato geralmente envolve reunir todo o grupo e fazer três perguntas: O que foi feito ontem? O que você fará hoje? Você está bloqueado ou precisa de alguma ajuda? Cada membro da equipe responde estas perguntas e então a equipe é coordenada para fazer o trabalho do dia.

Reuniões *standup* acontecem de maneira diferente com Kanban. A necessidade de fazer as três perguntas é evitada pela parede de cartões. A parede contém toda a informação sobre quem está trabalhando onde. Pessoas que participam regularmente podem ver o que mudou desde ontem e é evidente visualmente se algo está bloqueado ou não. Então *standups* têm um formato diferente com um sistema kanban. O foco é no fluxo de trabalho. O facilitador, tipicamente o gerente do projeto ou um gerente de linha, “andará pelo quadro”. A convenção têm se desenvolvido para se olhar o trabalho de trás para frente – da direita para a esquerda (na direção “do puxar”) – através dos *tickets* no quadro. O facilitador deve solicitar uma atualização do *status* em um ticket ou simplesmente perguntar se há alguma informação adicional que não está no quadro e pode não ser conhecida pela equipe.

Uma ênfase particular será dada para itens que estão bloqueados (têm um ticket rosa anexado) ou atrasados devido a defeitos (têm uma série de *tickets* azuis anexados). Perguntas também podem ser feitas sobre itens que parecem estar parados e não têm se movimentado por alguns dias. Algumas equipes planejaram maneiras de visualizar isto. Uma equipe de corrida italiana e fabricante de carros esportivos, por exemplo, marca um ponto ao lado do ticket para cada dia que ele permanece numa posição única. Isto permite que a equipe pergunta se um item pode ser marcado como bloqueado se ele não está fluindo ativamente. Isto melhora a capacidade de gerenciamento de problemas da organização (descrita mais detalhadamente no capítulo 20). A equipe discutirá brevemente quem está trabalhando no problema e quando ele será resolvido. Haverá também uma chamada para outros problemas bloqueadores que não estão no quadro e para as pessoas que precisam de ajuda para falar. Equipes mais avançadas e maduras sabem que não precisarão navegar sobre cada cartão na parede. Elas tenderão a focar apenas nos *tickets* que estão bloqueados ou que têm defeito. Este mecanismo permite que a reunião *standup* seja escalada para envolver um número muito maior de pessoas; Daniel Vacanti liderou uma *standup* bem sucedida com mais de 50 pessoas num projeto na Corbis em 2007 onde, apesar do grande tamanho da equipe, a reunião era finalizada em 10 minutos toda manhã.

O Após a Reunião

O Após a Reunião consiste na conversa de pequenos grupos de 2 ou 3 pessoas. Este comportamento surgiu espontaneamente, visto que os membros da equipe queriam discutir algo que estavam pensando: talvez um problema bloqueador, talvez um problema de projeto técnico ou de arquitetura, mas mais frequentemente, um problema relacionado ao processo. O Após a Reunião é um elemento vital da cultura de transformação que surge com Kanban. Após a Reunião gera ideias de melhoria e resulta em adaptação do processo e inovação.

Em grandes projetos, algumas Após a Reunião têm o formato de reuniões *standup* ao estilo de Scrum. Equipes de no máximo seis pessoas trabalhando juntas numa *feature*, *story*, ou requisito podem ser reunir brevemente para coordenar seus esforços do dia. Há uma diferença interessante entre esse comportamento de processo de Kanban e Scrum. Com Scrum, as equipes primeiramente se reúnem e então delegam ao Scrum de Scrums a coordenação de um programa ou projeto grande. Com Kanban o comportamento é inverso – a reunião no nível de programa acontece primeiramente.

Reuniões para Reabastecimento de Fila

Reuniões para reabastecimento de fila servem ao propósito de priorização em Kanban. Esta priorização é dita ser diferida ou adiada até o último momento razoável, devido à natureza do mecanismo de reabastecimento da fila e a cadência das reuniões. Reuniões para reabastecimento de fila são realizadas com um grupo de representantes do negócio ou donos do produto (para usar o vocabulário do desenvolvimento Ágil). É recomendado que estas reuniões aconteçam em intervalos regulares. Fornecer uma cadência constante para reabastecimento de fila reduz o custo de coordenação de se fazer a reunião e fornece certeza e confiança para o relacionamento entre o negócio e a equipe de desenvolvimento de software.

O propósito desta reunião é preencher a fila do sistema kanban para uma única cadeia de valor, sistema, ou projeto. *Stakeholders* que têm interesse nas entregas de uma equipe e que possui itens a espera no *backlog* devem participar desta reunião. Os participantes do negócio devem ser os mais seniores da organização. Pessoas mais seniores podem tomar mais decisões e frequentemente têm acesso a uma larga informação sobre o contexto. Isto melhora a qualidade da tomada da decisão e aperfeiçoa o processo de seleção para reabastecimento de fila.

Idealmente, uma reunião de priorização vai envolver muitos donos do negócio ou pessoas do negócio de grupos potencialmente concorrentes da empresa. A tensão criada por esta situação se torna uma influência positiva na tomada de decisão e estimula

um ambiente saudável e colaborativo com a equipe de desenvolvimento de software. Se apenas um dono do negócio participa, há um potencial para que a interação seja controversa.

Outros *stakeholders* interessados devem estar presentes na reunião, idealmente incluindo: qualquer um responsável pela entrega, por exemplo, um gerente de projeto; pelo menos um gerente funcional técnico, como um gerente de desenvolvimento ou teste, ou um gerente funcional técnico mais sênior; algumas pessoas que possam avaliar risco técnico, por exemplo, um técnico – ou arquiteto de dados; um profissional de usabilidade; um especialista em sistemas e operação; e um analista de negócio. Com a minha equipe em 2007, um gerente de desenvolvimento, o gerente da equipe de análise, e ocasionalmente o arquiteto corporativo ou arquiteto de dados participavam da reunião. Os gerentes de desenvolvimento se revezavam, participando da reunião de maneira rotativa.

A cadência das reuniões de priorização afetará o tamanho da fila no sistema kanban e, portanto, o *lead time* geral através do sistema. Para maximizar a agilidade da equipe, é recomendado que as reuniões sejam tão frequentes quanto seja possível; um intervalo semanal é o comumente recomendado.

Algumas equipes têm evoluído para uma priorização orientada à demanda em vez de reuniões regulares. Isto é recomendado apenas para organizações maduras nas quais todos os *stakeholders* da reunião podem estar disponíveis sob demanda. No caso de estudo da Microsoft do capítulo 3, o gerente de projeto criou gatilhos no banco de dados que o alertavam quando havia um espaço livre na fila de entrada. Ele então instigaria uma discussão de priorização com quatro donos do negócio via email, alertando-os que o espaço estava disponível para priorização. Uma discussão eletrônica garantiria que um novo item seria selecionado do *backlog*. Este processo tipicamente durava duas horas. Ter esse sistema por demanda em vez de uma reunião semanal poderia reduzir o tamanho da fila, o que levaria a uma melhoria subsequente no *lead time* através do sistema.

Reuniões de Planejamento de *Release*

Reuniões de planejamento de *release* acontecem especificamente para planejar o *release* ao final da cadeia de valor. Se os *releases* estão ocorrendo regularmente, com uma cadência, digamos, quinzenal, então faz sentido agendar a atividade de planejamento de *release* para que ela aconteça regularmente. Isto reduz o custo de coordenação para conduzir a reunião e garante que todos que precisam participar terão tempo disponível.

A pessoa responsável por coordenar a entrega, usualmente o gerente de projeto, tipicamente lidera as reuniões de planejamento de *release*. Quaisquer outras partes

interessadas devem ser convidadas: Isto usualmente inclui especialistas em gerência de configuração; operação de sistemas e especialistas de rede; desenvolvedores; testadores; analistas de negócio; e para todos esses colaboradores individuais, seus supervisores ou gerentes imediatos. Especialistas estão presentes devido ao seu conhecimento técnico e capacidade de avaliação de riscos. Gerentes estão presentes para que decisões possam ser tomadas.

Uma organização madura terá um *checklist* ou *framework* para um *release* que facilita o planejamento. Alguns dos pontos a serem considerados são os seguintes.

- Quais itens no sistema estão (ou estarão) prontos para o *release*?
- O que é necessário para realmente fazer o *release* de cada item para produção?
- Qual teste *post-release* será necessário para validar a integridade dos sistemas em produção?
- Quais os riscos envolvidos?
- Como esses riscos estão sendo mitigados?
- Quais planos de contingência são necessários?
- Quem precisa ser envolvido no *release* e estar presente durante “o puxar” para produção (ou outro mecanismo de entrega)?
- Quanto tempo dura o *release*?
- Quais outras logísticas serão envolvidas?

A saída deve ser um *template* completo representando um plano de *release*. Tenho visto com equipes particularmente sofisticadas o *release* planejado como uma orquestração de procedimentos a serem executados em uma sequencia determinada.

Numa reunião grande, pode ser impossível completar um planejamento de *release*, e pode ser necessário algum trabalho de acompanhamento após a reunião por parte do gerente de projeto.

Triagem

Triagem é um termo emprestado da profissão médica; ele se refere à prática de avaliar e classificar pacientes de emergência em categorias para prioridade de atenção. O sistema foi usado primeiramente em unidades médicas em campos de batalha, onde os pacientes eram separados em três categorias: fora do alcance de ajuda e provavelmente

morrerá em breve, provavelmente sobreviverá apenas se um tratamento médico imediato for realizado, e provavelmente sobreviverá sem tratamento médico imediato. Atualmente salas de emergência usam um sistema similar para priorizar pacientes à medida que eles chegam para tratamento.

A triagem foi adotada em desenvolvimento de software para classificar defeitos (*bugs*^{*}) durante a fase de estabilização de um projeto de software tradicional. Triagem é utilizada para classificar *bugs* que serão corrigidos, e sua prioridade, versus *bugs* que não serão corrigidos e será permitido que escapem para produção quando o produto for released. Uma triagem típica de defeitos envolve um líder de testes, um supervisor de testes ou gerente, um desenvolvedor líder, um desenvolvedor supervisor ou gerente, e um dono do negócio.

Com Kanban faz sentido fazer a triagem de defeitos. No entanto, a aplicação mais útil da triagem é no *backlog* de itens esperando para entrar no sistema.

Triagem de *backlog* deve ser realizada em intervalos relativamente pouco frequentes (Nota: alguns métodos Ágeis de desenvolvimento de software referem-se a isso como “preparação de *backlog*”). Mensalmente, trimestralmente, e duas vezes ao ano são intervalos que são populares com as equipes. Os participantes de uma triagem de *backlog* serão tipicamente os mesmos donos do negócio ou representantes do negócio que participam da reunião de reabastecimento de fila, juntamente ao gerente do projeto. As pessoas técnicas tipicamente não participarão em quantidade tão grande. Talvez um gerente médio de função técnica esteja presente.

O propósito de uma triagem de *backlog* é repassar cada item no *backlog* e decidir se ele deve permanecer no *backlog* ou se deve ser excluído. Ela não é uma classificação de pilha ou fornece qualquer priorização além de uma escolha simples mantenha-ou-exclua.

Algumas equipes têm evitado a necessidade de triagem através de automação e políticas. O caso de estudo da equipe Microsoft XIT do capítulo 4 excluiria qualquer item com mais de seis meses num intervalo mensal regular. O motivo era que se um item não havia sido selecionado para a fila de entrada em seis meses, era improvável que ele tivesse um valor significante e, portanto, improvável que ele fosse selecionado em algum momento. Se isto acontecesse, era também improvável que ele fosse solicitado novamente e, portanto, nada era perdido ao excluí-lo do *backlog*.

O propósito de se fazer uma triagem no *backlog* é reduzir o seu tamanho. O benefício de um *backlog* menor é que ele facilita discussões de priorização. Se há 200 itens no

* N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: defeitos.

backlog, será gasto um tempo significativamente menor em selecionar os ganhadores do que se existir 2000 itens no *backlog*.

Uma boa regra é se o *backlog* excede três meses de trabalho, isto é, três meses de rendimento de entrega, e todos os itens no *backlog* não podem entrar no sistema dentro deste tempo, pode ser uma boa ideia podar o *backlog*. O tamanho apropriado do *backlog* vai variar de acordo com mercados e domínios diferentes. Domínios com uma alta volatilidade precisarão talvez de um tamanho de *backlog* equivalente a um mês de itens em trabalho. Domínios com uma volatilidade muito baixa podem ser capazes de lidar com um *backlog* de até um ano de itens em trabalho.

Portanto, há uma relação entre o tamanho do *backlog*, a volatilidade do domínio no qual o sistema kanban está operando, e a velocidade de entrega, ou rendimento, da equipe. Se uma equipe entrega 20 *user stories* por mês e o domínio tem alguma, mas não muita, volatilidade, então o *backlog* de três meses é desejável, o *backlog* deve ter aproximadamente 60 itens.

Revisão e Escalação de Log de Problemas

Quando itens de trabalho num sistema Kanban estão impedidos, eles serão marcados como tal e um item de trabalho de problema será criado. O problema permanecerá aberto até que o impedimento seja removido e o item de trabalho original possa progredir através do sistema. Revisar problemas abertos, no entanto, torna-se vital para melhorar o fluxo através do sistema.

Revisão de *log* de problemas deve acontecer frequentemente e regularmente. Novamente, uma cadência regular reduz custos de coordenação e garante que *stakeholders* relevantes terão tempo para participar. Organizações muito maduras podem ser capazes de dispensar reuniões regulares e realizar reuniões sob demanda. Isto será apropriado se há uma quantidade relativamente pequena de problemas em aberto e se o custo de coordenação maior em reuniões sob demanda é realmente menor do que o custo de se realizar uma reunião agendada regularmente.

Revisões de *log* de problemas devem envolver o gerente de projeto e os membros da equipe que possuem itens bloqueados. As principais perguntas a serem respondidas são “Quem está atribuído e trabalhando no problema?” e “Para quando a resolução é esperada?” Problemas que não estão progredindo e estão eles mesmos bloqueados e antigos devem ser escalados para a gerência mais sênior.

Pode não ser necessário que a gerência sênior esteja presente na revisão de *log* de problemas, mas é importante se ter caminhos e políticas de escalação claramente definidos.

Gerenciamento e escalação de problemas são tipicamente mal conduzidos, até mesmo em organizações de desenvolvimento Ágil. Resolver problemas rapidamente, particularmente problemas que são externos a equipe de desenvolvimento, como disponibilidade do ambiente, requisitos ambíguos, ou falta de equipamento para teste, melhora o fluxo e aumenta muito a produtividade da equipe e o valor entregue. Gerenciamento e escalação de problemas são disciplinas importantes que fornecem um grande retorno. Melhorá-las deve ser uma prioridade até mesmo para equipes imaturas. Isto é discutido no capítulo 20.

Avatar Amigo (*Sticky Buddies*)

O conceito de um “avatar amigo” foi introduzido na Corbis para resolver um problema de coordenação. Havia uma política que permitia a realização de *home office* pelo menos uma vez por semana, particularmente para funcionários que moravam longe da cidade. A política remonta a uma mudança de escritório de Bellevue para Seattle, Washington, alguns anos antes. Pessoas em *home office* eram capazes de acessar o sistema de acompanhamento eletrônico, controle de versão, ambiente de *build*, e assim por diante via VPN. Então eles eram capazes de visualizar o trabalho atribuído para eles, trabalhar neles, completá-los e testá-los. Eles eram capazes de atualizar o *status* eletrônico do trabalho, marcá-lo como finalizado e de torná-los disponíveis para serem puxados para o fim da cadeia de valor. No entanto, eles não estavam fisicamente presentes no escritório e aptos a moverem a sua etiqueta na parede de cartões.

A solução para isto foi cada pessoa em *home office* fazer um acordo com alguma pessoa par que estaria presente no escritório para agir como seu delegado. Quando alguém em *home office* finalizava um item e mudava seu *status* eletrônico, ele deveria contatar seu avatar amigo por mensagem instantânea, email, ou telefone, e solicitá-lo a atualizar o quadro físico.

“Avatares amigos” também facilitavam o desenvolvimento distribuído através de diferentes localizações geográficas. Isto foi particularmente importante na Corbis, visto que a equipe de teste estava em Chennai, na Índia, e havia outros desenvolvedores especialistas em sistemas financeiros no Sul da Califórnia.

Sincronização através de Localizações Geográficas

Sincronizar equipes que usam sistema kanban através de localizações geográficas múltiplas é um assunto que vem sempre a tona como uma questão elementar para

aqueles que consideram implementar um sistema kanban. Frequentemente o questionador assume que implementações anteriores de Kanban foram realizadas numa única localização geográfica e que eu (e outros defensores de Kanban) não considerei os desafios de coordenação através de equipes geograficamente distribuídas.

Na verdade, o oposto é verdadeiro. A primeira equipe da Microsoft, do capítulo 4, estava localizada em Hyderabad, Índia, com o gerenciamento e donos do negócio em Redmond, Washington. Corbis, descrita no capítulo 5, também tinha pessoas na Índia e outras localidades fora de Seattle, como Los Angeles e Nova York, como também pessoas em *home office*.

A chave para coordenar através de múltiplas localizações é usar um sistema eletrônico; não é suficiente ter apenas uma parede de cartões.

Além do acompanhamento eletrônico, será necessário manter a parede física de cartões sincronizada, pelo menos diariamente. É importante atribuir a alguém essa responsabilidade em cada localização. Uma equipe que trabalhamos em 2008 era distribuída entre Nova York e Los Angeles. Eles mantinham (quase sempre) paredes de cartão idênticas em cada localização e faziam com que um membro da equipe fosse responsável por mantê-las sincronizadas diariamente.

Algumas equipes coordenam reuniões *standup* pelo telefone ou usando um sistema de vídeo-conferência. Antes de qualquer reunião *standup*, vídeo-conferência, ou chamada telefônica, a pessoa localmente responsável deveria gastar um tempo para garantir que o quadro físico estava sincronizado com o sistema eletrônico.

Takeaways

- ❖ A melhor prática é utilizar uma parede física de cartões e um sistema de acompanhamento eletrônico.
- ❖ É possível utilizar Kanban através de localizações geográficas múltiplas, desde que um sistema eletrônico esteja sendo utilizado.
- ❖ Sistemas eletrônicos que simulam a funcionalidade de uma parede física de cartões estão disponíveis em uma variedade de fornecedores.
- ❖ Realizar reuniões regulares reduz o custo de coordenação para estas reuniões e melhora a participação.
- ❖ Priorização e planejamento de *release* devem ser realizados independentemente e devem ter uma cadência independente.
- ❖ Reuniões *standup* diárias devem ser utilizadas para discutir problemas, impedimentos, e fluxo. Tipicamente, elas não seguem os padrões estabelecidos de outros métodos de desenvolvimento Ágil.
- ❖ *Standups* diários são uma parte essencial para incentivar uma cultura de melhoria contínua. Como o *standup* envolve toda a equipe brevemente todo dia, ele fornece uma oportunidade para todos os *stakeholders* sugerirem e discutirem oportunidades de melhoria. O período imediatamente posterior ao *standup* frequentemente se desenvolve em uma discussão informal sobre melhorias no processo.
- ❖ Preparar o *backlog* com uma triagem regular para reduzir o seu tamanho melhora a efetividade e eficiência das reuniões de priorização.
- ❖ Gerenciamento, escalação, e resolução de problemas são disciplinas centrais para melhoria do desempenho de uma equipe e elas devem ser endereçadas o quanto antes no desenvolvimento da equipe.
- ❖ Caminhos e políticas de escalação devem ser claramente definidos.

❖ CAPÍTULO 8 ❖

Estabelecendo uma Cadência de Entrega

A parte 3 (capítulos 6 até o 15) descreve os mecanismos para a implementação de um sistema kanban, concluindo com o capítulo 15, o qual descreve como conduzir uma iniciativa de mudança Kanban. A permissão para implementar a iniciativa requer uma negociação diferente daquela tipicamente realizada entre uma organização de desenvolvimento de software e seus parceiros e *stakeholders*. Parte desse novo tipo de negociação envolve acordos e compromissos de entregas regulares de software funcional.

O termo “cadência de entrega” no título deste capítulo implica no estabelecimento de um padrão de entrega de software funcional em um intervalo regular. Por exemplo, se o acordo é realizar uma entrega a cada duas semanas teremos uma cadência quinzenal ou de 26 entregas por ano. O acordo pode também considerar o dia da entrega. Por exemplo, toda segunda quarta-feira do mês, como foi o caso dos *releases* de manutenção das aplicações de TI da Corbis.

Na comunidade de desenvolvimento de software Ágil geralmente estabelece-se que uma cadência regular é importante. Métodos de desenvolvimento Ágil alcançam a regularidade com iterações *time-boxed*, que duram tipicamente de uma a quatro semanas. O argumento para o uso de *time-boxing* é baseado na noção de que uma “pulsação” regular do projeto é importante. Há um entendimento de que, para isso, é necessário o uso de iterações *time-boxed*. No início de

uma iteração, um escopo, ou *backlog*, é acordado e um compromisso feito. O trabalho começa! Atividades de análise, planejamento de testes, projeto, desenvolvimento, teste e *refactoring* são realizadas. Se tudo der certo, todo o escopo acordado é finalizado. A iteração acaba com a entrega de software funcional e a reunião de retrospectiva para discutir melhorias futuras e ajustes no processo. O ciclo então começa novamente. Tudo isso acontece numa cadência regular que foi acordada no início – semanal, quinzenal, mensal, ou qualquer outra.

Kanban dispensa o uso de iterações *time-boxed* desacoplando as atividades de priorização, desenvolvimento e entrega. É permitido o ajuste da cadência de cada uma dessas atividades naturalmente. Kanban não dispensa o uso do conceito de cadência regular. Equipes Kanban entregam software regularmente, dando preferência a uma escala menor de tempo. Kanban faz entregas de acordo com os Princípios Por Trás do Manifesto Ágil¹⁹, no entanto, Kanban evita qualquer disfunção introduzida artificialmente pelo uso de *time-box* ao forçar que as atividades sejam realizadas num período fixo de tempo.

Nos últimos dez anos as equipes que usam métodos Ágeis têm aprendido que menos *WIP* é melhor. Aprenderam também que transferências menores de *batch* são melhores do que transferências maiores. Como uma resposta a esse aprendizado, a partir da segunda metade da última década, essas equipes começaram a adotar iterações de duração menor. Equipes SCRUM típicas saíram de quatro para duas semanas e equipes *Extreme Programming* de duas para uma semana. Um dos problemas introduzidos por essa abordagem é a dificuldade de analisar o trabalho em unidades pequenas o suficiente de maneira que caibam na janela de tempo disponível. O mercado respondeu a isso desenvolvendo técnicas mais sofisticadas de análise e escrita de *user stories*. O objetivo foi reduzir o tamanho das estórias tornando-as mais granulares, reduzindo a variabilidade no tamanho a fim de que elas coubessem em iterações menores. Embora essa abordagem soe boa na teoria, ela é difícil de ser alcançada. Ela se encaixa na categoria do sexto elemento da Receita de Sucesso: Atacar fontes de variabilidade para melhorar a previsibilidade. Como descrito no capítulo 3, reduzir variabilidade é algo difícil de conseguir porque sempre requer que as pessoas mudem seu comportamento e aprendam novas habilidades.

As equipes têm então lutado para escrever *user stories* consistentemente pequenas o suficiente para que caibam em iterações *time-boxed* pequenas; isso leva a muitas disfunções. A primeira é a reversão da tendência de iterações menores para iterações maiores. A alternativa é escrever estórias com foco em elementos da arquitetura ou alguma decomposição técnica dos requisitos. Isso resulta, por exemplo, em uma estória para a interface do usuário, uma estória para a camada de persistência, e assim por diante. Uma segunda alternativa é quebrar as estórias em fases por três iterações, na

primeira iteração faz-se a análise e talvez o planejamento dos testes, a segunda envolve o desenvolvimento de código e a terceira os testes de sistema e a correção de defeitos. Qualquer uma dessas disfunções é possível. As duas últimas burlam a noção de iterações *time-boxed* e disfarçam o fato de que o trabalho ainda está em progresso quando ele é sinalizado como finalizado.

Kanban dissocia o tempo de criação de uma *user story* da taxa de entrega. Enquanto algum trabalho é finalizado e está pronto para ser entregue, outro trabalho pode estar em progresso. Tendo dissociado *lead time* para desenvolvimento da cadência de entrega, faz sentido perguntar com qual frequência a priorização (e talvez o planejamento e a estimativa) deve acontecer. É improvável que discussões sobre planejamento, estimativa e priorização devam acontecer no mesmo ritmo, visto que são atividades completamente diferentes que frequentemente precisam da atenção de diferentes grupos de pessoas. O esforço de coordenação relacionado a entregas é certamente diferente do esforço de coordenação para priorização de novas atividades; Kanban permite a dissociação dessas atividades.

Kanban também dissocia a cadência da priorização, tanto do *lead time*, como da cadência da entrega. Este capítulo aborda os elementos envolvidos na obtenção de uma cadência de entrega adequada e quando e se faz sentido ter entregas *ad hoc* ou por demanda em vez de entregas regulares. Seguindo a mesma linha de raciocínio, o capítulo 9 aborda como estabelecer uma cadência de priorização e quando ou se faz sentido ter reuniões de priorização por demanda ou *ad hoc* em vez de reuniões de priorização regulares. E o capítulo 11 descreve como estabelecer expectativas em relação ao *lead time* e como comunicar o conteúdo de um *release*.

Coordenação dos Custos de Entrega

Coordenar cada entrega de software envolve custos. Faz-se necessário envolver as pessoas para se discutir a implantação (ou *release*), empacotamento, marketing, comunicação, documentação, treinamento para o usuário final, treinamento para o revendedor, *help-desk* e para o suporte técnico, documentação de instalação, procedimentos de instalação, equipe de plantão, assistência no ambiente de implantação, e assim por diante. Planejar o *release* de um pedaço de software funcional pode ser inacreditavelmente complexo, dependendo da natureza do domínio de negócios e do tipo de software. Atualizar um *web site* pode ser trivial comparado a atualização de um *firmware* de equipamentos militares espalhados pelo mundo, ou de satélites em órbita, ou de um avião caça, ou dos nós de uma rede telefônica.

Em 2002, quando estávamos nos Estados Unidos planejando o *release* da atualização do PCS Vision para a operadora de celular Sprint PCS, dezenas de milhares de pessoas precisaram ser treinadas. Nas lojas ao redor de todo o país 17.000 pessoas tiveram que ser treinadas nas *features* da nova rede e nos 15 ou mais aparelhos de celular oferecidos. Um número similar de pessoas teve que ser treinada para dar suporte às inevitáveis ligações para o suporte que poderiam advir quando o consumidor adquirisse seus novos aparelhos. Apenas o planejamento do treinamento de 30.000 pessoas é um custo enorme de tempo e dinheiro.

É importante entender então, os custos associados à coordenação de entregas. Por exemplo, se os desenvolvedores de software têm que participar de reuniões para coordenação de entregas, isso está distraindo-os da construção do software para a entrega? Em seguida há uma lista de algumas questões a serem consideradas:

- Quantas reuniões?
- Quantas pessoas?
- Quanto tempo será gasto?
- Qual custo de oportunidade é incorrido quando as pessoas são distraídas das suas atividades regulares?

Custos de Transação de Entrega

É fácil entender os custos físicos de transação para se fazer uma entrega; primeiramente há o pagamento. O cliente pagará ao fornecedor usando algum instrumento monetário, um cartão de crédito, por exemplo. Para a comodidade de se ter um pagamento via cartão de crédito, as bandeiras líderes de mercado, como Visa ou MasterCard cobram uma taxa de transação de tipicamente dois a quatro por cento do valor da transação.

Além dos custos de transação financeira entre cliente e fornecedor, há também as taxas de entrega. Entrega envolve gasto em dinheiro, mas também em tempo e mão de obra, e pode haver também os custos de instalação. Digamos, por exemplo, que você compre uma máquina de lavar da Sears e agende a entrega para um determinado dia. Nos bastidores, agendar a entrega e coordenar o motorista para que o modelo correto da máquina de lavar seja entregue na casa correta no dia determinado é uma coordenação de custos de entrega. Retirar a máquina do estoque, dirigir até a sua casa e entregá-la para você são custos de transação. Talvez a mesma pessoa, ou outra pessoa, um encanador, instale-a para você. O encanador gasta tempo para dirigir até a sua casa

e ainda mais tempo para realizar a instalação. Todo esse tempo e esforço de entrega e instalação faz parte do custo de transação de se comprar uma máquina de lavar.

Economicamente, o varejo absorve o custo da transação do cartão de crédito. Os outros custos de transação para entrega e instalação são frequentemente repassados para o consumidor. Nem todos os custos de transação são “vistos” ou “sentidos” por todos os participantes na cadeia de valor, mas eles afetam o desempenho do sistema econômico como um todo. O efeito real de todos esses custos é o acréscimo no valor final pago pelo consumidor sem o aumento do valor entregue.

É verdade que a máquina de lavar sem ser entregue ou instalada tem pouco valor, mas o seu valor agregado está na sua capacidade de lavar roupas. A entrega e a instalação são atividades sem valor agregado que devem ser contadas como custo de transação.

Em desenvolvimento de software, os custos de transação de entrega também podem ser físicos por natureza. Algumas empresas, como a Microsoft, ainda “*release to manufacture*” RTM e fazem impressão de mídia física, como DVDs, e os empacota e os envia para distribuidores, varejistas e outros parceiros. Com software embarcado pode ser necessário fabricar chips, ou, pelo menos, “injetar” código de software dentro do *firmware* usando tecnologia como EE-PROM. Os chips, se necessário, poderão então ser colocados dentro do hardware que controlarão.

Em outros casos, pode ser necessário se fazer uma implantação eletrônica. Por exemplo, celulares agora permitem o que é chamado de gerenciamento de dispositivo *over-the-air* para atualização de *firmware* e configurações do dispositivo. Muitos satélites e sondas espaciais permitem atualização over the air. Essa facilidade na implantação faz com que as missões espaciais sejam muito mais ágeis do que eram no passado. As missões espaciais podem ser alteradas pelo *upload* do software. Defeitos podem ser corrigidos no local. Alguns defeitos vergonhosos do Telescópio Hubble, como a capacidade de foco, foram parcialmente corrigidos com alterações no software; isto alterou a economia da implantação.

Muitas pessoas lendo este livro podem estar envolvidas em desenvolvimento *web* ou desenvolvimento de aplicações internas. Implantação pode significar apenas copiar arquivos de discos para outras máquinas. Apesar disso parecer trivial, frequentemente não é. Frequentemente faz-se necessário planejar um procedimento elaborado para desativar bancos de dados, servidores de aplicação, e outros sistemas, e então atualizá-los e ativá-los novamente. Uma das maiores dificuldades é a migração de um esquema de banco de dados de uma geração do banco para outra, pois bancos de dados podem ser muito grandes. O processo de colocar os dados em série num arquivo e analisá-los, descompactá-los e, talvez melhorá-los ou aumentá-los com outros dados, e então compactá-los e colocá-los num novo esquema pode levar horas – talvez até dias.

Em alguns ambientes, desenvolvimento de software pode levar horas ou dias. Isto acontece não porque o software é de baixa qualidade ou tem uma arquitetura defeituosa; isso simplesmente reflete a natureza do domínio no qual o software é utilizado. Todas as atividades envolvidas em entregar software, sendo ele uma aplicação empacotada, *firmware* embarcado ou uma aplicação a ser executada num servidor interno, devem contabilizar planejamento, agendamento e recursos para então serem realmente realizadas.

Eficiência de Entrega

A equação que calcula a eficiência da entrega pode ser avaliada por dois caminhos. O caminho mais simples é olhar apenas para o trabalho e custos envolvidos. O método mais complexo considera o valor entregue.

Primeiro, o modelo apenas de custos. Devemos considerar os custos totais incorridos entre *releases*. Esse valor é muitas vezes conhecido – *burn rate* da organização. Se os *releases* são mensais e o fluxo de caixa negativo é de \$1.3 milhões por mês, os custos são de pelo menos \$1.3 milhões por *release*. Podemos também incorrer em custos de fabricação, custos de impressão e custos de publicidade, isto é, despesas não reembolsáveis de coordenação do *release*. Tudo isso é relativamente fácil de contar. Vamos imaginar que o valor é \$200.000, então o custo total é de \$1.5 milhões.

Sabemos que os custos não reembolsáveis de entrega são de \$200.000, mas quanto dos \$1.3 milhões foram gastos planejando, coordenando, e realmente realizando a entrega? Se tivermos dados adequados de controle de tempo disponíveis, poderemos calcular este valor. Mesmo que não tenhamos, poderemos dar um bom palpite. Quantas reuniões? Quantas pessoas? Quantas horas por reunião? Adicione o número de homens-hora para as atividades relacionadas à implantação, multiplique pela carga horária. Se isso somar \$300.000, teremos um custo de transação e coordenação de \$500.000 por entrega.

$$\text{Eficiência de Entrega\%} = 100\% \times (\text{Custo Total} - (\text{Custo de Coordenação} + \text{Custo de Transação})) / \text{Custo Total do Release de Software}$$

No nosso exemplo, nosso percentual de eficiência é:

$$100\% \times (1 - (\$500,000 / (\$500,000 + \$500,000))) = 50\%$$

Para ser mais eficiente temos que (a) aumentar o tempo entre entregas, ou (b) reduzir os custos de coordenação e transação. A escolha (a) é a escolha típica das empresas ocidentais do século XX. É a escolha que valoriza a “economia de escala”: Faça coisas em *batches* maiores para amortizar os custos durante os longos períodos de

tempo. Escolha (b) é a escolha típica das empresas japonesas do século XX e empresas buscando o *Lean Thinking*. Escolha (b) foca na redução do desperdício reduzindo custos de coordenação e transação a fim de tornar o tamanho do *batch* eficiente - nesse caso, tornar o tempo entre *releases* eficiente.

Quanto de eficiência é o bastante?

Essa é realmente uma questão aberta. Cada empresa tem visões diferentes sobre o valor adequado para eficiência, e muito dependerá do valor a ser entregue.

Estabelecendo uma Cadênciade Entrega

Se entendermos o valor envolvido num *release* podemos fazer uma escolha melhor em relação à frequencia de entrega. Caso a nossa entrega mensal de software gera uma receita de \$2 milhões versus um custo de \$1.5 milhões, saberemos que temos uma margem de \$500.000 dessa atividade. Podemos reescrever nossa equação de eficiência como:

$$\text{Eficiência de Entrega\%} = \\ 100\% \times (1 - ((\text{Custos de Transação} + \text{Custo de Coordenação}) / (\text{Margem} + \text{Custos de Transação} + \text{Custos de Coordenação})))$$

Para o nosso exemplo, isto produziria um percentual de eficiência de:

$$100\% \times (1 - ($500,000 / ($500,000 + $500,000))) = 50\%$$

Agora isso fica ainda mais complicado porque calcular o valor real de uma entrega pode ser quase impossível. Podemos não ter pedidos de preço fixo. Podemos especular sobre a aceitação do mercado, o preço e a margem que podemos alcançar. Podemos entregar itens de valor intangível, como uma revisão da identidade da nossa marca e materiais de marketing, ou usabilidade e correção de defeitos para o nosso produto ou *web site*.

Calcular se devemos ou não postergar e entregar com uma frequência menor para sermos mais eficientes, é igualmente difícil. Aumentar nosso *time to market* pode causar um efeito adverso em nossa participação no mercado, preço e margem. O conceito de eficiência de entrega não é uma ciência exata. O mais importante é que você, a equipe, e a organização estejam cientes dos custos de se fazer uma entrega – em tempo e dinheiro – e que sejam capazes de fazer alguma avaliação racional sobre uma frequencia de entrega aceitável.

Se uma equipe de quinze pessoas precisa de dez pessoas por três dias para fazer uma entrega de código, é aceitável fazer uma entrega a cada dez dias úteis (ou duas semanas)? A resposta será provavelmente, não. Talvez uma vez por mês, ou a cada vinte dias

úteis, seja uma escolha melhor. Por outro lado, o mercado pode exigir agilidade e *time to market*, e os riscos podem ser mitigados por *releases* mais frequentes, e nesse caso o custo vale a pena. Você precisa usar o bom senso e decidir por si mesmo.

Melhore a Eficiência para Aumentar a Cadência de Entrega

Continuando o exemplo, determinamos a necessidade de dez pessoas para entregar o código. A partir disso inferimos que um *release* mensal é aceitável. No entanto, muitas pessoas acreditam que com a melhoria na qualidade do código, melhoria na gerência de configuração, melhores ferramentas para lidar com a migração de dados e ensaios regulares dos procedimentos de instalação, seria possível diminuir o tempo de três dias para oito horas. Subitamente um *release* a cada duas semanas parece viável. Será que semanalmente seria possível? O que você faria?

Meu conselho é que inicialmente a escolha seja conservadora. Concorde com um *release* mensal. Deixe a organização provar que ela pode alcançar esse nível de consistência. Depois de alguns meses, reflita sobre a qualidade do código e fomente um programa para melhoria da gerência de configuração. Caso exista recursos disponíveis envolva-os na criação de ferramentas para melhorar a migração de dados para outros esquemas durante um *release*. E finalmente, encoraje a equipe a exercitar o *release* num ambiente de testes. Talvez você precise comprar, instalar e licenciar um ambiente como esse; tudo isso vai levar tempo.

Desafie a equipe e os gerentes funcionais imediatos que controlam e fazem os *releases* a reduzir os custos de transação e de coordenação. A partir do momento que esses custos diminuam, revise o progresso nas reuniões de revisão operacionais e estabeleça contato com outros *stakeholders*. Quando você se sentir seguro a se comprometer com uma cadência de *release* mais frequente, como uma quinzenal, faça!

Reducir custos de coordenação e transação está no coração do *Lean*. Isso é eliminação de desperdício na sua forma mais potente, permitindo que *batches* menores tornem-se eficientes e permitindo agilidade no negócio. Reduzir custos de coordenação e transação é mudar o jogo. No entanto, não foque simplesmente em reduzi-los. Reduza-os com um objetivo em mente – fazer entregas mais frequentes de software funcional e consequentemente entregar valor aos seus clientes mais regularmente.

Fazendo Entregas Sob Demanda ou *Ad Hoc*

Entregas regulares trazem vantagens. Comprometer-se com datas específicas de entrega, por exemplo, toda segunda quarta-feira, permite que as pessoas envolvidas se

programem; isso fornece segurança. Isso também reduz os custos de coordenação já que não há mais sobrecarga para decidir quando a entrega deve ser realizada e quem deve participar – tudo isso foi estabelecido uma vez e continuará consistente a partir de então.

Entregas regulares também ajudam a construir confiança. Falta de previsibilidade destrói confiança. Notam-se muito mais as falhas em se fazer entregas em datas pré-acordadas do que falhas específicas no conteúdo da entrega.

Tendo-se usado um forte argumento para uma cadência regular de entrega, faz sentido fazer entregas sob demanda ou *ad hoc* em alguma circunstância? Quais seriam essas circunstâncias?

Primeiramente, entregas sob demanda ou *ad hoc* fazem sentido quando os custos de coordenação para as atividades do *release* são pequenos. Quando os custos de coordenação são baixos, não há benefícios de agendar atividades de coordenação regulares. Em segundo lugar, isso faz sentido quando os custos de transação são baixos, talvez porque a implantação do código seja largamente automatizada e a qualidade seja garantida com antecedência, antes da implantação. E finalmente, isso faz sentido em ambientes onde as implantações são tão frequentes que não há necessidade real de se desenvolver um padrão: Novo software está sendo entregue tão frequentemente que parece contínuo para a maioria dos observadores e *stakeholders* externos – seus cérebros não foram programados para antecipar uma data de entrega. E quando não há expectativas, não podem existir desapontamentos.

Esse tipo de implantação quase contínua de código parece ser útil e necessária em algumas indústrias. Os exemplos que emergiram dos pioneiros no uso de Kanban têm sido principalmente na indústria de mídia, por exemplo, a IPC Media em Londres, onde eles usam múltiplos sistemas Kanban para planejar o desenvolvimento para propriedades de mídia online como o mousebreaker.com, um jogo online muito viciante.

As duas primeiras circunstâncias de custos baixos de coordenação e transação tendem a indicar organizações de alta maturidade. Isso também tem sido observado pelos pioneiros no uso de Kanban. O departamento XIT da Microsoft fez uma parceria com um fornecedor CMMI nível de maturidade 5 na Índia e com a Microsoft IT em Redmond, Washington, que é aproximadamente CMMI nível 3 de maturidade. Organizações com alta maturidade tendem a ter níveis estabelecidos de confiança com parceiros da cadeia de valor e *stakeholders* externos, incluindo gerência sênior, dessa maneira elas não precisam de cadência regular de entrega para construir confiança.

Então, em geral, escolha uma cadência regular exceto em circunstâncias nas quais, confiança e altos níveis de capacidade e maturidade já existam, e em domínios onde uma implantação quase contínua é desejável.

Há uma circunstância final na qual é aceitável se fazer entregas sob demanda. Quando há uma requisição urgente que está sendo tratada como um caso especial e de expedição. Esse conceito de classe de serviço de Expedição é explicado na seção *Lead Time* do capítulo 11. Devemos escolher expedir por diversas razões, a primeira e mais óbvia delas está no advento de um defeito de produção crítico. Em circunstâncias na quais nada mais importa a não ser corrigir o problema, um *release* fora do ciclo deve ser planejado.

Há outras circunstâncias onde *releases* fora do ciclo fazem sentido. Talvez a equipe de vendas tenha feito um pedido para um grande cliente que deseja uma versão customizada de um software e devido a restrições orçamentárias e do ciclo fiscal ele precisa da entrega para antes do final do mês (e do trimestre). O pedido vem do grupo de operações e a engenharia de software deve parar tudo para cumprir esse pedido do cliente porque isso trará muita receita.

Sob essas circunstâncias faz sentido planejar um *release* especial e fora do ciclo. Esse *release* deve ser tratado como excepcional, e a cadênciā regular deve ser restabelecida assim que possível logo após o *release* excepcional; embora seja necessário usar o bom senso. Por exemplo, se um *release* regular é agendado para quarta-feira e o *release* excepcional é requisitado para a sexta-feira da mesma semana, faz sentido atrasar o *release* de quarta-feira para sexta-feira. Caso você escolha fazer isso, é importante comunicar apropriadamente e suficientemente cedo de modo que as expectativas sejam redefinidas. Você não quer perder a confiança com seus parceiros da cadeia de valor como um efeito colateral de tentar ser flexível e útil.

Takeaways

- ❖ “Cadência de entrega” significa um acordo de entregas de software funcional a intervalos regulares.
- ❖ Kanban desacopla cadência de entrega tanto do *lead time* de desenvolvimento como da cadência de priorização.
- ❖ Iterações curtas e *time-boxed* têm levado a disfunções para algumas equipes tentando adotar métodos de desenvolvimento ágil.
- ❖ Entrega ou *release* de software envolve coordenação de muitas pessoas de várias funções; toda essa coordenação tem um custo mensurável.
- ❖ Entrega ou *release* de software traz consigo um conjunto de custos de transação em tempo e dinheiro; esses custos podem ser determinados e acompanhados.
- ❖ Eficiência de entrega pode ser calculada comparando-se a soma dos custos de transação e coordenação para se fazer uma entrega e o custo total para se criar o software para a entrega.
- ❖ Cadência de entrega pode ser estabelecida comparando-se o custo de produção do *release* e o valor recebido do *release*.
- ❖ Eficiência e cadência podem ser aumentadas focando-se na redução dos custos de transação e coordenação.
- ❖ Entregas regulares constroem confiança.
- ❖ Fixar uma expectativa de cadência regular e fazer entregas consistentes a essa expectativa pode ser viciante.
- ❖ Agendar entregas regulares reduz custos de coordenação.
- ❖ Entregas sob demanda ou *ad hoc* podem fazer sentido para organizações de alta maturidade com altos níveis de confiança e baixos custos de transação e coordenação de entrega estabelecida.
- ❖ Requisições legítimas para expedir entregas também podem ser causa de *releases* fora do ciclo. *Releases* regulares devem ser restabelecidos o mais rápido possível depois de um *release* excepcional.

❖ CAPÍTULO 9 ❖

Estabelecendo uma Cadência de Entrada

Este capítulo discute os elementos envolvidos no estabelecimento de uma cadência de priorização adequada e quando, ou se, faz sentido ter uma priorização sob demanda ou *ad hoc* em vez de uma reunião de priorização agendada regularmente.

Coordenando Custos de Priorização

Quando estávamos introduzindo Kanban na Corbis, em 2006, escolhemos começar com a manutenção de engenharia que lidava com requisições de atualização pequenas e correção de defeitos para a suíte completa das aplicações de IT, incluindo funções tanto financeiras e de recursos humanos, como também de sistemas de domínio mais específico do sistema de gerenciamento de ativos e o *web site* de *e-commerce*. Esses sistemas serviam pelo menos seis unidades de negócio, incluindo vendas, marketing, operações de venda, finanças, e ao departamento que dava suporte à cadeia de fornecedores que vendia fotografia digital, *meta-data tagging*, catalogação, e execução – essencialmente a cadeia de fornecedores do negócio.

Seis departamentos competiam pelos recursos compartilhados disponíveis para fazer essas pequenas mudanças e atualizações. Quando o sistema kanban foi introduzido, um caso de negócio foi feito para

uma unidade de engenharia de apoio que poderia fornecer *releases* estratégicos e frequentes. Essa unidade de apoio possibilitaria agilidade limitada no negócio através de *releases* de pequenas funções de forma incremental enquanto outros novos projetos de aplicações de TI foram construídos utilizando um mecanismo de governança departamental de gerenciamento de programa (*Program Management Office – PMO*) tradicional. Cada projeto do *portfolio* foi justificado e autorizado com base no seu caso de negócios independente. A unidade de apoio foi aprovada pelo comitê executivo, e um financiamento adicional de dez por cento foi autorizado para a unidade de engenharia de software, o que gerou cinco contratações adicionais para o departamento. Essa nova capacidade foi chamada de Equipe de Resposta Rápida (ERR). O nome foi completamente impróprio – inicialmente a equipe não era rápida, não dava respostas rápidas, e na verdade nem havia uma equipe.

Não era viável criar um departamento de manutenção especializado com essas cinco pessoas novas. Corbis tinha uma diversidade considerável de sistemas de TI e muitos requeriam habilidades específicas. A unidade de análise que desenvolvia e elaborava requisitos de sistema dependia particularmente fortemente de especialistas. As cinco pessoas adicionais foram distribuídas pela unidade de engenharia de software nas funções de gerenciamento de projeto, análise de sistema, desenvolvimento, teste, gerência de configuração, e *build*. Dessa maneira não havia uma equipe, o E em ERR não tinha sentido. O desafio da gerência foi mostrar que os dez por cento adicionais para recursos estavam sendo gastos em trabalhos de manutenção e simplesmente não estavam sendo consumidos pelo trabalho em grandes projetos do portfólio.

Decidiu-se dedicar um gerente de projeto na unidade de apoio de engenharia. Apesar dessa gerente não trabalhar em tempo integral na unidade de apoio, ela era um ponto focal único para comunicação e coordenação e contava como metade de uma das cinco pessoas alocadas na iniciativa. Um engenheiro de *build* da equipe de gerência de configuração também foi designado especificamente para a iniciativa. Seus deveres eram manter os sistemas de pré-produção requeridos para teste e staging e construir código e colocá-lo no ambiente de teste quando requisitado.

Para manter a integridade do ambiente de teste, que era usado por múltiplos projetos ao mesmo tempo, Corbis tinha uma política onde apenas engenheiros de *build* tinham permissão de promover o código do ambiente de desenvolvimento para o ambiente de teste. Essa política mudou posteriormente, mas em Setembro de 2006 a realidade é que era necessário um engenheiro de *build* para promover o código para teste.

Antes da introdução de Kanban, o esforço de coordenação requerido para se chegar num acordo em relação ao escopo de um *release* de manutenção era proibitivo. O

gerente de projeto, e frequentemente seu chefe, e o grupo de gerência de projeto, deveriam convocar uma reunião com todas as partes relevantes, incluindo analistas de negócio, representantes do negócio, analistas de sistema, gerentes de desenvolvimento, líderes de teste, e engenheiro de *build*, e algumas vezes o gerente da gerência de configuração, como também o pessoal de operações de sistema e *help-desk*. Essas reuniões poderiam durar muitas horas e eram frequentemente inconclusivas. Membros da equipe deveriam fazer estimativas, e outra reunião era agendada. A última reunião era frequentemente repleta de debates sobre prioridades, e novamente, era frequentemente inconclusiva. Em Setembro de 2006 gastava-se duas semanas com muitas reuniões que duravam muitas horas para se chegar num acordo em relação ao escopo de um *release* que supostamente deveria ter apenas duas semanas de construção e implantação. Devido à duração de duas semanas de uma iteração, apenas requisições muito pequenas poderiam ser acomodadas e muitas requisições potencialmente valiosas eram ignoradas. Essas requisições deveriam ser redirecionadas para um projeto maior e dessa maneira poderiam levar meses ou anos antes de serem implementadas. O sistema não era rápido e não dava respostas rápidas antes da introdução do sistema kanban, então o RR de ERR também não tinha sentido.

Kanban libertou esta equipe de todas essas disfunções e ela, por iniciativa própria, ganhou o nome de Equipe de Resposta Rápida.

Quando o sistema kanban foi introduzido, os donos do negócio foram educados no fluxo de trabalho, na fila de entrada e no mecanismo puxado. Eles aprenderam que seriam questionados simplesmente para preencher os espaços vazios na fila e que não seria necessário priorizar o *backlog* de requisições. Se havia dois espaços livres na fila, a pergunta deveria ser, “Quais são os dois próximos novos itens?” Assumindo que tínhamos dados do *lead time* médio e do desempenho do atendimento aos prazos versus o *lead time* alvo e o acordo de nível de serviço, a pergunta mais elaborada poderia ser, “Quais dois itens poderiam ser entregues em 30 dias a partir de hoje?” O desafio seria então que os seis donos de negócio que competiam entre si de alguma maneira chegassem num acordo de quais seriam os dois itens dentre às muitas possibilidades de escolha.

Contudo, a pergunta é simples, e era esperado responder-se a uma pergunta tão simples em uma hora facilmente. Havia um consenso que uma hora era razoável, consequentemente os donos do negócio eram questionados se eles abririam mão de uma hora de sua semana para participar de uma reunião semanal de priorização para preencher a fila de entrada para a unidade de engenharia de apoio.

Acordando uma Cadência de Priorização

A reunião foi agendada para toda segunda-feira às 10 a.m. Geralmente havia a participação de gerentes de negócio de hierarquia superior que não costumavam participar das reuniões de escopo de *release*, e usualmente o vice-presidente de cada grupo funcional. Adicionalmente, o gerente de projeto, o Diretor Sênior da Engenharia de Software, o Diretor Sênior dos Serviços de TI (cujas responsabilidades incluíam a gerência de projeto), pelo menos um gerente de desenvolvimento, o gerente de teste, o gerente do time de análise, e ocasionalmente alguns contribuidores individuais poderiam participar.

Entrar em acordo para uma cadência regular fornecia a todos, previsibilidade. Eles estavam prontos a abrir mão de uma hora nas manhãs das segundas-feiras, e geralmente a participação na reunião era boa.

Cadência de priorização semanal é uma boa periodicidade. Ela fornece interações frequentes com os donos do negócio, e ela constrói confiança através da interação dos envolvidos. No jogo colaborativo e cooperativo de desenvolvimento de software, ela possibilita que os jogadores movam as peças uma vez por semana. Reuniões semanais são possíveis devido à simplicidade da pergunta a ser respondida e a garantia de que a reunião pode ser finalizada em uma hora. Quando pessoas de negócio gastam tempo longe da administração dos seus negócios, elas precisam ver que o seu tempo está sendo bem gasto.

Muitos dos aspectos de kanban contribuem para que a reunião de priorização semanal seja gratificante: É uma experiência colaborativa; há transparência no trabalho e no fluxo de trabalho; o progresso pode ser reportado toda semana; e todos sentem que estão contribuindo para algo valioso. Para muitos dos vice-presidentes na Corbis, parecia que o processo ERR estava fazendo a diferença. Eles ganharam um novo nível de respeito no departamento de TI e eles aprenderam a colaborar com os seus pares em outros departamentos de uma maneira que anteriormente não era comum na Corbis.

Eficiência de Priorização

Reuniões de coordenação de periodicidade semanal pode não ser a resposta certa para a sua organização. Você pode achar que os seus desafios de coordenação são mais difíceis ou mais simples do que àqueles da Corbis. Algumas equipes sentam-se próximas, então não há necessidade de uma reunião; a coordenação de priorização pode ser uma rápida discussão pelas mesas. Por outro lado, algumas equipes podem ter pessoas em fusos horários diferentes e em muitos continentes diferentes, então reuniões semanais não serão facilmente agendadas. Talvez as perguntas a serem respondidas não sejam simples

como no exemplo da Corbis e a reunião pode durar muito tempo. É difícil imaginar uma situação com mais de 6 grupos competindo pelo mesmo recurso compartilhado, mas é possível. Quanto mais grupos envolvidos, mais tempo a reunião pode demorar. Quanto mais longa a reunião, menos frequentemente você vai conseguir fazê-la.

Como um conselho geral, priorizações mais frequentes são desejáveis. Elas permitem que a fila de entrada seja menor, e, como resultado, há menos desperdício no sistema. O WIP é baixo e dessa maneira o *lead time* é mais curto. Priorizações mais frequentes permitem que todas as partes trabalhem junto mais frequentemente. A experiência de trabalho colaborativo constrói confiança e melhora a cultura. Empenhe-se para encontrar o esquema de coordenação menor e mais eficiente possível e faça reuniões de priorização o mais frequentemente possível.

Custos de Transação de Priorização

Para facilitar uma reunião eficiente toda segunda-feira, Diana Kolomyets, a gerente de projeto, deveria enviar um e-mail na quinta-feira ou sexta-feira para informar aos participantes o número estimado de espaços vazios na fila na segunda-feira pela manhã. Ela solicitava que eles olhassem o backlog de requisições e escolhessem os candidatos para seleção na segunda-feira. Essa “tarefa de casa” frequentemente solicitava-os a preparar algum argumento para dar suporte à seleção bem sucedida do seu item favorito. Nós começamos a ver documentos de suporte nas reuniões da segunda-feira. Algumas pessoas preparavam um caso de negócio, ou alguma apresentação para dar suporte à sua escolha; outros começaram a fazer *lobby* entre si. Era comum que alguns participantes da reunião levassem outra pessoa da reunião para um almoço na sexta-feira, especificamente para ganhar apoio para sua escolha na segunda-feira pela manhã. A “troca de votos” foi introduzida e um membro poderia concordar em dar apoio a outro membro da equipe numa semana e em troca receberia apoio para o seu próprio item numa semana futura. A natureza das regras do jogo, onde múltiplas organizações competem por um recurso ERR compartilhado, introduziu um novo nível de colaboração.

Ocasionalmente, a equipe de negócio poderia ficar preocupada que uma requisição fosse muito cara para implementar comparando-se com o seu valor, então eles poderiam solicitar à equipe de análise que fizesse uma estimativa. Posteriormente, regras relacionadas a classes de serviço foram introduzidas como forma de guiar se as estimativas valeram à pena ou não; isso é totalmente explicado no capítulo 11.

Todas essas atividades, incluindo estimativas, preparação de plano de negócio, e seleção de candidatos do *backlog*, são trabalhos de preparação para priorização. Em termos econômicos esses são custos de transação de priorização; é desejável que esses

custos sejam mantidos baixos. Se os custos de transação tornam-se onerosos, não importando o quanto baixos são os custos de coordenação, a equipe não vai querer se reunir regularmente. Evitando, o máximo possível, estimativas detalhadas, os custos de transação são minimizados, fazendo que reuniões de priorização sejam mais frequentes.

Melhore Eficiência para Aumentar a Cadênciade Priorização

No geral, a equipe de gerenciamento deve estar consciente de todos os custos de transação e coordenação incorridos por todos – não apenas a equipe de desenvolvimento – os envolvidos na priorização e seleção de novos itens na fila para desenvolvimento e entrega.

Muitas organizações Ágeis usam uma forma de priorização chamada *Planning Poker* que utiliza a técnica “*wisdom of the crowds* – sabedoria da multidão”, na qual todos os membros da equipe votam usando um cartão representando um número de tamanho. É realizada uma média dos votos, ou algumas vezes chega-se a um consenso discutindo-se os desvios nos votos e votando-se novamente até que todos na equipe concordem numa estimativa. Os cartões de *poker* frequentemente usam uma escala de numeração não linear, como a Série de Fibonacci, para incentivar o dimensionamento relativo.

Alguns argumentam que essa técnica de planejamento, que é também uma forma de jogo colaborativo e cooperativo, é muito eficiente, visto que ela permite que uma estimativa bastante precisa seja estabelecida rapidamente. Há evidências que comprovam isso, mas há igualmente evidências que sugerem que pensamento em grupo também é possível. Já ouvi relatos de equipes, como um *startup* em São Francisco, que recorrentemente subestimavam, apesar do uso de um jogo transparente e colaborativo como o *Planning Poker*. Já ouvi de gerentes seniores de um *web site* de agendamento de viagens muito conhecido, que suas equipes consistentemente superestimavam, apesar de utilizarem o *Planning Poker*. Acreditando ou não que esses jogos de planejamento sejam efetivos, vale muito a pena considerá-los, dado os argumentos de que eles são efetivos.

É verdade que jogos de planejamento envolvendo toda a equipe podem gerar muito rapidamente uma estimativa para um item individual, como uma *user story*; no entanto, o exercício envolve toda a equipe; há um custo de coordenação significante para isso. Isso funcionará efetivamente em pequenas equipes focadas em produtos simples; no entanto, se extrapolamos a técnica para uma organização como a Corbis, onde estávamos mantendo 27 sistemas de TI usando 55 pessoas, muitas das quais especialistas no seu campo, domínio, sistema, ou tecnologia, seria necessário ter quase todas as 55 pessoas na reunião para se fazer uma boa estimativa e alcançar uma “*wisdom from the*

crowd – sabedoria a partir da multidão”. Os custos de transação para planejamento e estimativas seriam baixos, mas os de coordenação seriam altos.

No geral, devido a esse efeito de custo de coordenação, esses métodos de planejamento Ágeis são eficientes apenas para pequenas equipes focadas em sistemas simples e linhas de produto.

Escolhendo-se eliminar as estimativas para a maioria das classes de serviço, os custos de transação e de coordenação serão reduzidos. Essa redução facilita reuniões de priorização mais frequentes porque as reuniões permanecem eficientes. Isso permitiu que equipes kanban fizessem priorização *ad hoc* ou por demanda.

Fazendo Priorização Ad Hoc ou Por Demanda

Como descrito no capítulo 4, em 2004, Dragos Dumitriu introduziu um sistema kanban com sua equipe de Engenharia de Apoio XIT na Microsoft. Os parceiros de negócio principais foram quatro gerentes de produto que representavam várias unidades de negócio. Eles focaram e priorizaram requisições de mudança para os 80 ou mais sistemas de TI apoiados pela XIT.

Quando Dragos e eu projetamos o sistema kanban para introdução com a XIT, nós projetamos uma fila de entrada grande o bastante em face de pelo menos uma semana de rendimento. Apesar do fato que todos os quatro representantes de negócio e Dragos estavam situados em Redmond, Washington, no campus da Microsoft, a reunião de priorização acontecia por telefone. O campus da Microsoft é imenso; a quantidade de prédios pode atingir o número de centenas, embora haja apenas algo em torno de 40 prédios no total. A área coberta tem vários quilômetros quadrados e o transporte entre as seções do campus é feito por mini ônibus ou Toyota Prius. Muitos preferem realizar chamadas de conferência para reuniões de priorização do que fazê-las face a face. Isso tem um impacto negativo no nível de confiança e no capital social, mas facilita eficiência.

Então Dragos estabeleceu uma ligação telefônica semanal para priorizar novas requisições do *backlog*. Os quatro gerentes de produto representavam as unidades de negócio que forneciam financiamento para os fundos da equipe de Dragos via transferência de fundos entre companhias. Baseado nesse financiamento era possível determinar aproximadamente quantas vezes alguém deveria escolher um item do *backlog*. O gerente de produto que fornecia seis décimos para o fundo poderia escolher três de cada cinco oportunidades. Os outros poderiam selecionar itens de uma maneira similar baseando-se nos seus níveis de financiamento. O gerente de produto que fornecia o menor financiamento poderia escolher um item em cada 11. Podemos descrever isso como um método de seleção ponderado *round-robin*.

Então as regras do jogo de priorização colaborativo e cooperativo XIT eram simples. Cada semana os gerentes de produto deveriam preencher os espaços abertos da fila de entrada – tipicamente três espaços. Cada um deles poderia escolher baseando-se na sua posição na fila *round-robin*. O *lead time* alvo para o acordo de nível de serviço era de 25 dias. Então se eles tinham uma chance de escolher uma requisição de mudança para desenvolvimento, eles deveriam perguntar a si mesmos “Quais dos itens no meu *backlog* eu gostaria que fossem entregues em 25 dias a partir de hoje?” A ordem na qual eles fariam a escolha era muito clara e simples baseada no seu nível de financiamento para o departamento.

Devido à natureza simples das regras, a reunião era finalizada muito rapidamente. Tornou-se claro que uma ligação telefônica de coordenação não era realmente necessária. Dragos tinha o banco de dados Microsoft *Product Studio* (um precursor do *Visual Studio Team System*, *Team Foundation Server*) que fornecia um email a partir de um gatilho que indicava quando um espaço tornava-se livre. Ele então enviava o e-mail para os quatro gerentes de produto. Eles acordavam rapidamente sobre quem deveria escolher um item e a pessoa selecionava algum deles. Tipicamente, um espaço vazio na fila era preenchido em duas horas.

Os custos de coordenação excepcionalmente baixos, unidos aos custos de transação baixos relacionados à decisão de não se estimar requisições de mudança, e junto à relativa alta maturidade da equipe envolvida, permitiu que a Microsoft XIT dispensasse o agendamento regular das reuniões de priorização.

É importante notar que a Microsoft em Redmond é aproximadamente o equivalente a uma organização CMMI-ML3 e que o fornecedor sendo utilizado para o desenvolvimento e teste XIT era uma equipe CMMI-ML5 situada em Hyderabad, na Índia. Então essa equipe tinha a vantagem de custos de coordenação baixos, custos de transação baixos, e particularmente níveis altos de maturidade organizacional. O efeito desses três fatos significou que reuniões de priorização por demanda tornaram a equipe mais efetiva.

Como regra geral, você deve escolher priorização *ad hoc* ou por demanda quando você tem uma organização de alto nível de maturidade, custos baixos de transação, e custos baixos de coordenação de priorização. Caso contrário, é melhor usar reuniões de priorização com agendamento regular e coordenar a seleção da fila de entrada com uma cadência regular.

Takeaways

- ❖ “Cadência de priorização” significa um acordo sobre um intervalo regular entre reuniões para priorizar novo trabalho para desenvolvimento na fila de entrada.
- ❖ Kanban remove disfunções potenciais relacionadas à coordenação do planejamento de iteração dos métodos Ágeis dissociando a cadência de priorização do *lead time* de desenvolvimento e entrega.
- ❖ Priorização de novas requisições de trabalho como *user stories* envolve a coordenação de muitas pessoas de várias funções; toda essa coordenação tem um custo mensurável.
- ❖ Estimativas para facilitar decisões de priorização representam custos de transação em tempo e dinheiro associados com a priorização; esses custos podem ser determinados e acompanhados.
- ❖ Políticas relacionadas ao método de priorização e as entradas para tomada de decisão representam as regras do jogo colaborativo e cooperativo de priorização no Kanban aplicado ao desenvolvimento de software.
- ❖ Jogos de planejamento utilizados nas metodologias Ágeis não são escalados com facilidade e podem representar um custo de coordenação significante para equipes maiores com um foco mais amplo do que uma simples linha de produto.
- ❖ Cadência de priorização pode ser estabelecida encorajando-se os envolvidos na decisão de priorização a fazer reuniões regulares baseado nos custos de transação e coordenação envolvidos.
- ❖ Eficiência e cadência de priorização podem ser incrementadas focando-se em reduzir seus custos de transação e coordenação.
- ❖ Reuniões de priorização frequentes constroem confiança.
- ❖ Agendar reuniões de priorização regulares reduz os custos de coordenação e é particularmente útil em organizações de baixa maturidade.
- ❖ Priorização *ad hoc* ou por demanda pode fazer sentido para organizações de alta maturidade com níveis estabelecidos de confiança e com custos baixos de transação e coordenação associados às políticas de priorização para tomada de decisão.

❖ CAPÍTULO 10 ❖

Estabelecendo Limites para o Trabalho- em-Progresso

Como discutido no capítulo 2, uma das cinco propriedades fundamentais no Método Kanban é o trabalho-em-progresso (*WIP* – *Work-In-Progress*) ser limitado. Então é verdade dizer que uma das decisões mais importantes que você fará quando introduzir Kanban é escolher os limites para o trabalho-em-progresso através do fluxo de trabalho.

O capítulo 15 aconselha que o trabalho-em-progresso deve ser acordado deve ser acordado por consenso com os *stakeholders* em todos os níveis e gerência sênior. É verdade que limites podem ser declarados unilateralmente; no entanto, há poder em conseguir consenso e obter um comprometimento dos *stakeholders* externos em relação à política de limitar o *WIP*. Quando sua equipe e o processo são colocados sob *stress* você pode recorrer ao acordo colaborativo no qual houve consenso. Você pode redirecionar a discussão para uma redefinição do processo em vez de desviar-se ou estender o sistema, ou usá-lo indevidamente diferentemente de como foi projetado e implementado. Construir consenso é uma maneira de manter a disciplina do limite do *WIP* e evitar substituir ou abandonar um limite.

Limites para Tarefas

Na Microsoft com a equipe XIT, Dragos Dumitriu decidiu que os desenvolvedores e testadores deveriam trabalhar num único item de cada vez; não deveriam existir múltiplas tarefas. Isso foi declarado unilateralmente, mas felizmente essa escolha não se tornou problemática com outros *stakeholders*. Isso estava alinhado com as práticas de trabalho correntes e o método Personal Software Process (PSP) em uso pela equipe. A organização era madura o bastante para manter a disciplina e seguir o processo que havia sido acordado. Você deve lembrar que no início do estudo de caso, no outono de 2004, havia três desenvolvedores e três testadores na equipe. Então o limite do *WIP* para cada, desenvolvimento e teste, era três.

Em 2006 na Corbis, com a iniciativa da engenharia de apoio, nós tomamos uma decisão similar; onde analistas, desenvolvedores e testadores, deveriam trabalhar em apenas um item de valor para o cliente de cada vez. Com novos projetos grandes, nós começamos a tomar decisões diferentes; havia mais trabalho colaborativo nesses projetos. Era comum que equipes de duas ou três pessoas trabalhassem num único item. Como esses itens poderiam ser bloqueados ou atrasarem, nós especulamos se faria sentido permitir a troca de algumas tarefas e algum paralelismo adicional; consequentemente o limite do *WIP* foi estabelecido de maneira que duas ou três pessoas trabalhassem no mesmo item, mas que permitisse também algum acúmulo. Por exemplo, se tínhamos dez pessoas e antecipadamente duas pessoas por item, o limite do *WIP* poderia ser cinco adicionando um pouco mais para suavizar o impacto de um item bloqueado. Talvez oito (cinco mais três) seria o limite correto nessas circunstâncias.

Havia alguma pesquisa e observação empírica que sugeriam que dois itens em progresso para cada trabalhador era o número ideal. Esse resultado é frequentemente citado para justificar multitarefa. No entanto, eu acredito que essa pesquisa tende a refletir a realidade de trabalho nas organizações observadas; há muitos impedimentos e razões para o trabalho atrasar. A pesquisa não relata a maturidade das organizações estudadas, nem correlaciona os dados com quaisquer questões externas (variações causadas por atribuição, discutidas no capítulo 19). Portanto, o resultado pode ser uma consequência dos ambientes estudados e não um número ideal de fato. Não obstante, você pode enfrentar resistência à noção de que um item por pessoa, par, ou equipe pequena, é a escolha correta. O argumento pode ser que tal política é muito restritiva. Nesse caso, estabelecer um limite de *WIP* de dois itens por pessoa, par ou equipe é razoável. Pode haver inclusive casos onde um limite de três itens por pessoa, par, ou equipe é aceitável.

Não há fórmula mágica para se fazer essa escolha. É importante lembrar que o número pode ser ajustado empiricamente. Você pode escolher um número e então

observar se ele está funcionando bem. Caso não esteja, ajuste-o para cima ou para baixo.

Limites para Filas

Quando um trabalho é finalizado e está à espera para ser puxado para o próximo estágio no seu fluxo de trabalho, diz-se que ele está “enfileirado”. Quão grande essas filas podem ser? O menor possível. O limite de *WIP* para uma fila aparece frequentemente entre colchetes com a etapa de trabalho anterior. Por exemplo, as filas Desenvolvimento e Desenvolvimento Completo são colocadas juntas entre colchetes (Figura 10.1). Caso uma política muito firme para o *WIP* das tarefas foi estabelecida, como um item por pessoa, par, ou equipe pequena, será necessário ter alguma fila que absorva a variação e mantenha o fluxo. Se, em operação, o seu sistema kanban sofrer do comportamento “pare-avante” que faz com que os funcionários fiquem parados devido à variabilidade da quantidade de tempo necessário para finalizar as tarefas, você pode precisar aumentar o tamanho das filas. No entanto, caso você tenha escolhido dois itens por pessoa, par ou equipe, por exemplo, você já tem um retentor para variabilidade, dessa maneira

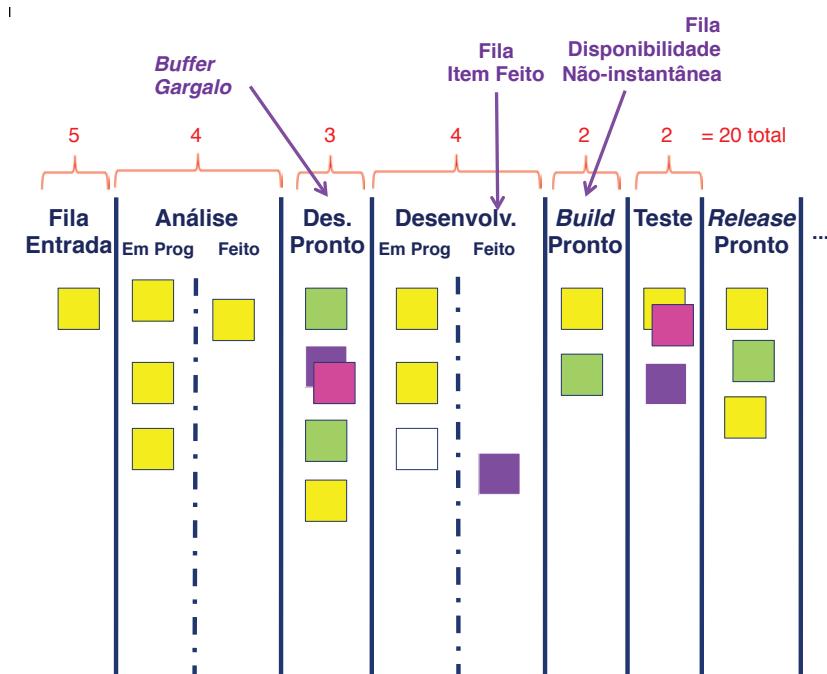


Figura 10.1 Parede de cartões mostrando tipos diferentes de filas e um *buffer*

o tamanho da sua fila pode ser efetivamente zero. Simplesmente coloque juntas entre colchetes a coluna da tarefa e a fila completa.

Retenha Gargalos

O gargalo no seu fluxo de trabalho pode requerer uma retenção à sua frente. Este é um mecanismo típico de exploração de gargalo, como explicado no capítulo 16. O tamanho da retenção é importante. Novamente você deve querer que ele seja o menor possível. Retenções e filas adicionam *WIP* ao seu sistema e o seu efeito é aumentar o *lead time*. No entanto, retenções e filas suavizam o fluxo e melhoram a previsibilidade do *lead time*. Suavizando o fluxo, há aumento do rendimento, então mais trabalho é entregue através do sistema kanban. Retenções também garantem que as pessoas se mantêm trabalhando e proporcionam uma melhor utilização do sistema. Deve existir balanceamento e a retenção ajuda a mantê-lo. Em muitas circunstâncias você está à procura de agilidade no negócio através de *lead times* menores e maior qualidade, em parte, através de menos trabalho-em-progresso. No entanto, não sacrifique previsibilidade a fim de alcançar agilidade ou qualidade. Se os tamanhos da sua fila e retenção são muito pequenos e o seu sistema sofre do comportamento “pare-avance” devido à variabilidade, seu *lead time* será imprevisível, com uma ampla variabilidade. A chave para escolher um limite de *WIP* para uma retenção é que ele deve ser grande o suficiente para garantir um fluxo suave no sistema e evitar tempo ocioso no gargalo. Mais detalhes no tamanho da retenção e como projetar retenções para restrição de capacidade e gargalos de não disponibilidade imediata serão discutidos no capítulo 16.

Tamanho da Fila de Entrada

O tamanho da fila de entrada pode ser diretamente determinado a partir da cadência de priorização e do rendimento, ou taxa de produção do sistema. Por exemplo, se uma equipe está produzindo a uma taxa media de cinco trabalhos finalizados por semana (com uma variação típica de quatro a sete itens por semana), e a cadência de reabastecimento da fila é semanal, o tamanho da fila pode ser atribuído para sete; novamente isso pode ser ajustado empiricamente. Se você executa o seu sistema por muitos meses e a fila nunca é totalmente esgotada antes que a reunião de priorização ocorra, ele está provavelmente muito grande, então a reduza em um e observe os resultados. Repita isso até que você tenha uma reunião de priorização na qual você solicitará aos representantes do negócio que preencham a fila inteira.

Se, por outro lado, você tem uma reunião de priorização na segunda-feira e a fila é esgotada na quinta-feira à tarde – e como resultado algumas pessoas da equipe estão ociosas – sua fila é muito pequena. Aumente o tamanho da fila em um e observe por mais algumas poucas semanas.

Tamanhos de fila e retenção devem ser ajustados empiricamente conforme necessário. Portanto, não se afilia em tomar uma decisão para estabelecer um limite de WIP. Não atrasse o início do uso do seu sistema kanban porque você não consegue chegar a números perfeitos para limite do WIP. Escolha alguma coisa! Escolha dar progresso com informações imperfeitas e então observe e ajuste; Kanban é um processo empírico.

Qual tamanho deve ter a fila de entrada se você está usando priorização por demanda? Você deve recordar do capítulo 4 que a equipe XIT tinha uma fila de entrada de cinco itens. Ela foi projetada para ser grande o bastante para absorver o rendimento de uma semana. Ela foi baseada no pressuposto de que a reunião de priorização estava acontecendo semanalmente. No entanto, os gerentes de produto decidiram rapidamente que a reunião era desnecessária e que era aceitável se fazer decisões dirigidas a evento quando um espaço na fila tornava-se disponível. Uma vez que isso aconteceu, eu deveria ter avisado a Dragos para reduzir a entrada de cinco para apenas um. É um reflexo da minha inexperiência naquele momento eu não ter feito isso. O sistema mudou. Os pressupostos sob os quais ele foi projetado mudaram. A política para o tamanho da fila de entrada foi baseada naqueles pressupostos e deveria ter sido revisitada. As melhorias no *lead time* teriam sido mais expressivas caso tivéssemos feito isso.

Quando a XIT mudou para uma priorização sob demanda eles levavam tipicamente duas horas para preencher um espaço vazio na fila. Teria sido aceitável assumir que o maior tempo para reiniciar a fila seria de quatro horas. No entanto, os desenvolvedores não estavam fisicamente próximos aos gerentes de produto. As pessoas que tomavam as decisões de priorização estavam em Redmond, Washington, e os desenvolvedores em Hyderabad. Cada um deles estava trabalhando (oficialmente) oito horas por dia em fusos horários opostos. Então havia ocasiões nas quais os indianos pegavam um trabalho pela manhã, o finalizavam, e precisavam que a fila fosse reabastecida, mas os gerentes de produto estavam obviamente, dormindo. Devido a esse problema de disponibilidade não instantânea, nós provavelmente deveríamos ter permitido 16 horas para reabastecimento de um único item na fila em circunstâncias extremas. Lembre-se que os desenvolvedores eram o gargalo no fluxo de trabalho. Para maximizar o rendimento, nós nunca queríamos os desenvolvedores ociosos. Então nós precisamos ser conservadores; 16 horas é conservador quando se gasta apenas duas horas em média para a decisão de reabastecimento da fila de entrada. Então qual seria o rendimento num período médio de 16 horas? No auge do desempenho, a equipe alcançaria 56 itens

num único trimestre; isso é menos do que cinco por semana. Então num período de 16 horas era improvável que eles finalizassem até mesmo um único item da fila, então um tamanho de fila de um era aceitável. Não existir fila seria aceitável. Haveria ainda alguma chance da equipe sofrer de tempo ocioso quando eles finalizassem um item durante a janela de 16 horas quando os gerentes de produto poderiam estar indisponíveis para reabastecer a fila.

Seções Ilimitadas de Trabalho

Na solução de sistema puxado para problemas de fluxo da Teoria das Restrições, conhecida como Tambor-Retenção-Corda (*Drum-Buffer-Rope*), todas as etapas de trabalho a partir do gargalo têm *WIP* ilimitado. Este esquema é baseado no pressuposto de que elas têm mais capacidade para aumento do rendimento do que o gargalo e têm folga na capacidade que resulta em tempo ocioso; como resultado não há necessidade de limitar o *WIP*. Isto é ilustrado na Figura 10.2(a) a qual é baseada na metáfora usada em *The Goal* de Goldrat e apresenta uma patrulha de escoteiros caminhando em fila indiana. Uma corda é amarrada entre o escoteiro líder e o escoteiro que se movimenta mais lentamente (no. 4 na fila), que é o gargalo do rendimento (a taxa na qual a patrulha cobre o terreno). Apenas uma corda é necessária, visto que os escoteiros que estão atrás do escoteiro mais lento nunca ficarão para trás, visto que eles podem andar mais rápido do que o quarto da fila que restringe o passo de toda a patrulha.

Com um sistema kanban, a maioria de todas as situações no fluxo de trabalho tem limites de *WIP*. Isso traz uma vantagem potencial, pois impedimentos causados por

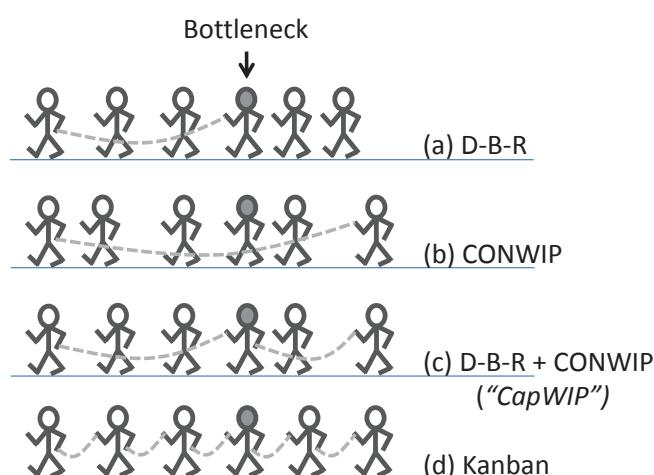


Figura 10.2 Desenhos ilustrando quatro sistemas puxados diferentes com limites de *WIP*

uma variabilidade inesperada, ou não antecipada, podem fazer com que um passo anterior torne-se um gargalo temporário. O limite de *WIP* local com o sistema kanban fará com que a fila pare rapidamente, mantendo o sistema livre de gargalos e não sobrecarregado. Quando o impedimento é removido, o sistema reiniciará graciosamente. O estilo kanban de limitar o *WIP* é ilustrado na figura 10.2(d), que mostra como os escoteiros poderiam ser amarrados uns aos outros se utilizassem o estilo kanban. Neste caso, cada escoteiro é amarrado ao próximo da fila. São necessárias cinco cordas para controlar o passo da patrulha inteira de seis escoteiros.

Em alguns casos pode ser aceitável que um sistema kanban não tenha limites nos passos finais do processo. No exemplo da XIT na Microsoft, assumiu-se que a disponibilidade do usuário base para realizar os testes de aceitação era efetivamente infinita e essencialmente instantânea, portanto não havia necessidade de limitar o *WIP* no teste de aceitação do usuário. Na Corbis, a fila *release-pronto* foi eliminada. Isto foi baseado no pressuposto de que o *batch* do trabalho *release-pronto* nunca se tornaria excessivo, dado à cadência de *release* ser quinzenal. Se, por outro lado, fosse possível que o material para *release-pronto* se tornasse excessivo isso poderia aumentar a complexidade do *release*, o que poderia resultar em custos antieconômicos de coordenação e transação do *release* sendo então necessário limitar o *WIP* na fila *release-pronto*. No entanto, esse nunca foi o caso da Corbis, dessa maneira o estado *release-pronto* permaneceu sem restrições.

Não Estresse a Sua Organização

Escolher estreitar demasiadamente os limites de *WIP* inicialmente pode gerar estresse excessivo na sua organização. Organizações de baixa maturidade com poucas capacidades terão mais impedimentos. Portanto, organizações de baixa maturidade com poucas capacidades podem acreditar que introduzir um sistema kanban exige muito esforço caso atribuam-se limites de *WIP* muito baixos. Se há muitos impedimentos, representados por vários *tickets* de cor rosa espalhados pelo quadro, apertar demasiadamente os limites de *WIP* significará que tudo vai parar e as pessoas ficarão ociosas. Apesar de em tempo ocioso tendermos a focar a atenção e acelerar os esforços para resolução de problemas e remoção de impedimentos, isso pode ser muito doloroso para uma organização de baixa maturidade. Gerentes seniores podem ficar irritados observando tantas pessoas ociosas ainda recebendo contracheques.

Ao introduzir mudança, você deve estar ciente da curva de efeito J. Idealmente, cada mudança produz um pequeno J onde qualquer impacto no desempenho é superficial,

então o sistema rapidamente se recupera e apresenta melhoria. Caso você aperte muito os limites de *WIP*, você sofrerá da curva de efeito J que é muito profunda e muito longa, a qual pode causar efeitos indesejáveis: Kanban está expondo todos os problemas da organização, mas ele pode acabar sendo culpado por fazer tudo ficar pior, e será visto como sendo parte do problema mais do que a solução; então tenha cuidado. Você pode ser mais agressivo com as suas políticas de limite de *WIP* com organizações de maior capacidade e mais maduras que sofrem com pequenos problemas inesperados (variações atribuídas à causa). Para organizações mais caóticas, você deve introduzir limites mais soltos inicialmente com um *WIP* maior e uma intenção de reduzi-lo posteriormente.

Não Estabelecer um Limite de *WIP* é um Erro

Embora eu advirta você a não ser agressivo quando estabelecer os limites de *WIP* iniciais, eu me tornei convicto de que não estabelecer limite para *WIP* é um erro.

Alguns precursores de Kanban, como o Yahoo!, escolheram não estabelecer limites porque eles supuseram que suas equipes eram muito caóticas para lidar com o esforço que isso pode introduzir. A esperança era que essas organizações amadureceriam através do controle visual dos elementos de Kanban e que os limites de *WIP* poderiam ser introduzidos posteriormente. Isto se provou problemático, no entanto, e muitas equipes abandonaram Kanban sem visualizar muitas melhorias, enquanto outras foram dissolvidas em reorganizações corporativas ou cancelamento de projetos, negando-nos mais dados. Na Corbis, muitas equipes em projetos maiores usavam Kanban com limites muito soltos de *WIP* para funcionalidades de alto nível; os resultados eram bastante distintos.

Tenho convicção de que a tensão criada pela imposição de limite de *WIP* através da cadeia de valor é uma tensão positiva. Esta tensão positiva força a discussão sobre os problemas da organização e disfunções. As disfunções geram impedimentos para o fluxo e resultam em produtividade, *lead time* e qualidade aquém do esperado. A discussão e colaboração invocadas pela tensão positiva do *WIP* limitado são saudáveis. É um mecanismo que possibilita a emergência de uma cultura de melhoria contínua. Sem limites de *WIP*, o progresso na melhoria de processo é lento. Equipes que impuseram limites de *WIP* desde o início relataram crescimento acelerado na capacidade e maturidade organizacional e têm entregado resultados de negócio superiores com frequência; entregas previsíveis de software de alta qualidade. Em comparação, equipes que deferiram a introdução de limites de *WIP* têm geralmente lutado muito e apresentado melhorias limitadas.

Alocação de Capacidade

Uma vez que estabelecemos os limites de *WIP* para o fluxo através do sistema, podemos considerar a alocação de capacidade por tipo de item de trabalho ou classe de serviço.

Figura 10.3 mostra o projeto da parede de cartões do capítulo 6 com limites de *WIP* através das colunas totalizando 20 cartões. A capacidade é alocada através dos tipos de item de trabalho, nomeados, 60 por cento para requisições de mudança, 10 por cento para itens de manutenção, e 30 por cento para mudanças de textos em produção. Isso equivale a um limite de *WIP* na raia de 12 para requisições de mudança, 2 para itens de manutenção, e 6 para mudanças de textos em produção.

Alocação de capacidade nos permite garantir o serviço para cada tipo de trabalho recebido pelo sistema kanban. A alocação deve geralmente ser feita em resposta à demanda observada para cada tipo de trabalho. Portanto, é importante realizar análise de demanda para facilitar uma alocação razoável de limites de *WIP* nas raias para cada tipo de trabalho.

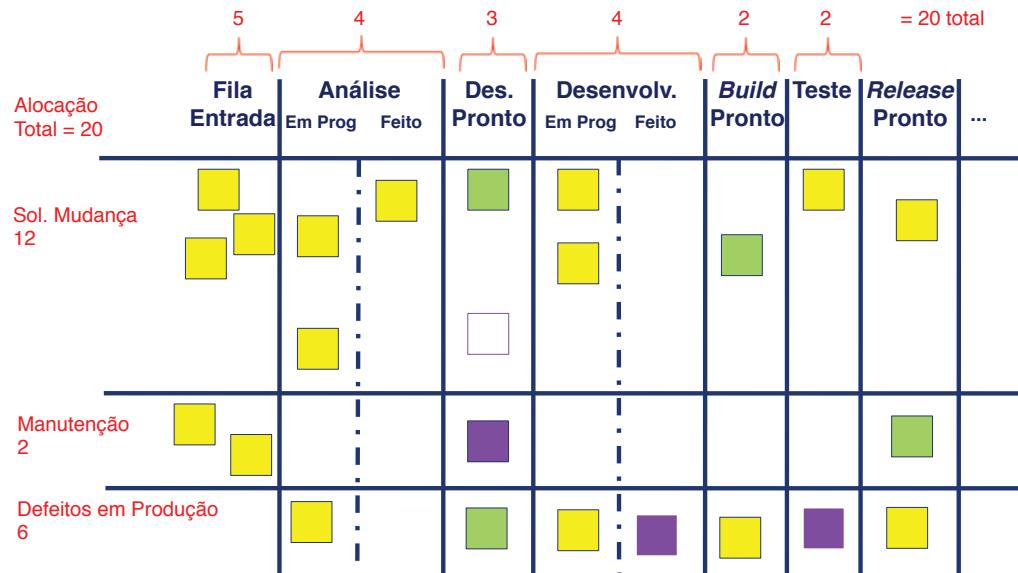


Figura 10.3 Parede de cartões mostrando raias para cada tipo de trabalho com limites de *WIP* explícitos para cada raias

Takeaways

- ❖ Limites de *WIP* devem ser acordados a partir de consenso com os *stakeholders* de todos os níveis e a gerência sênior.
- ❖ Decisões unilaterais para limites de *WIP* são possíveis, mas podem ser difíceis de defender posteriormente, quando o sistema é colocado sob estresse.
- ❖ Limites de *WIP* para tarefas devem ser estabelecidos para um número médio de itens por pessoa, pares de desenvolvedores, ou equipes pequenas e colaborativas.
- ❖ Tipicamente, o limite deve estar num intervalo de um a três itens por pessoa, par, ou equipe.
- ❖ Limites de fila devem ser mantidos pequenos, tipicamente apenas grandes o bastante para absorver a variação natural (causa randômica) no tamanho dos itens e duração das tarefas.
- ❖ Os gargalos devem ser retidos.
- ❖ O tamanho da retenção deve ser o menor possível, mas grande o suficiente para garantir um ótimo desempenho no gargalo e manutenção do fluxo do sistema.
- ❖ Todos os limites de *WIP* podem ser ajustados empiricamente.
- ❖ Kanban é um processo empírico.
- ❖ Não se deve desperdiçar tempo tentando determinar um limite de *WIP* perfeito; simplesmente escolha um número que é próximo o bastante, e progride; ajuste empiricamente, se necessário.
- ❖ Etapas de trabalho posteriores sem limite são possíveis.
- ❖ Deve-se ter cuidado ao se estabelecer etapas sem limite no fluxo para que não sejam introduzidos gargalos ou custos de transação e coordenação excessivos quando entregas são feitas (transferência de *batch* para etapas posteriores)
- ❖ Uma vez que os limites de *WIP* foram estabelecidos, a capacidade pode ser alocada através dos itens de trabalho.
- ❖ Usa-se frequentemente raias para cada tipo de trabalho e um limite de *WIP* para cada raia.
- ❖ Alocação de capacidade requer análise da demanda comparativa através dos diferentes tipos de trabalho recebidos pelo sistema kanban.

❖ CAPÍTULO 11 ❖

Estabelecendo Acordo de Nível de Serviço

Nós temos familiaridade com o conceito de diferenciar classes de serviço. Todos que fizeram um *check-in* para um vôo num aeroporto sabem que clientes que pagam mais pela passagem, ou que gozam de prêmios de um programa de fidelidade, são autorizados a usar uma via rápida para driblar a fila. Algumas vezes esses privilégios são estendidos para as filas de segurança do aeroporto e incluem o uso de um salão especial e tratamento preferencial no momento do embarque. Os clientes que pagam mais ou aqueles que são fiéis à companhia aérea gozam de uma melhor classe de serviço.

Somos familiarizados com o conceito em desenvolvimento de software e sistemas de TI também, mas evidentemente com resolução de defeitos, e particularmente com defeitos em produção. Nós avaliamos defeitos por severidade (impacto) e prioridade (urgência). Defeitos de severidade e prioridade altas são resolvidos imediatamente. Eles recebem uma classe de serviço diferente, mais alta do que outros trabalhos. Para corrigir um defeito de produção de severidade alta, colocamos outros trabalhos de lado, envolvemos quantas pessoas sejam necessárias, e frequentemente elaboram-se planos especiais para uma correção de emergência, *patch*, ou *release* para mitigar o problema.

Este conceito pode ser aplicado mais genericamente, o que oferece mais vantagens para a agilidade no negócio e gerenciamento de risco. Algumas requisições são mais necessárias do que outras, enquanto

algumas são mais valiosas do que outras. Oferecendo tratamento diferente para tipos de trabalho com classes de serviço diferentes, podemos fornecer ao cliente mais flexibilidade otimizando o resultado econômico.

Classes de serviço nos fornecem um caminho conveniente de classificação do trabalho a fim de fornecer níveis aceitáveis de satisfação do cliente a um custo economicamente excelente. Identificando rapidamente a classe de serviço de um item, somos poupadados da necessidade de realizar estimativas ou análise detalhadas. Políticas associadas com uma classe de serviço afetam a maneira como os itens são puxados no sistema. Classe de serviço determina prioridade dentro do sistema. Classes de serviço permitem uma auto-organização, uma abordagem de valor – e risco – otimizada para priorização e re-planejamento.

Definições Típicas de Classe de Serviço

Classes de serviço são tipicamente definidas baseando-se no impacto do negócio. Diferentes notas coloridas, cartões de índice, ou *tickets* são atribuídos para cada classe e claramente identificam a classe de serviço de uma dada requisição, como na figura 11.1; caso contrário, são desenhadas raias separadas na parede de cartões para representar os membros das classes de serviço.

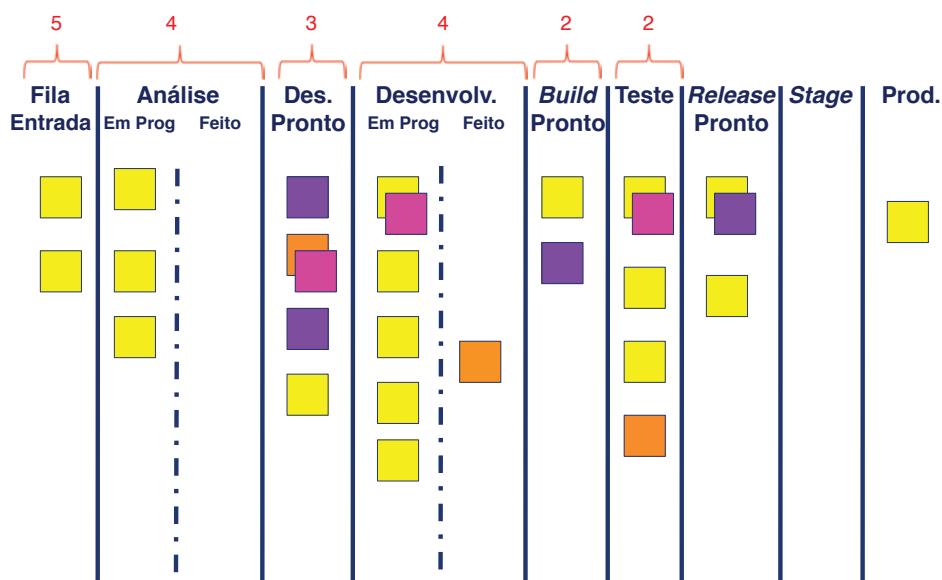


Figura 11.1 Parede de cartões mostrando *tickets* coloridos para demonstrar classes de serviço

Cada classe de serviço traz seu conjunto de políticas próprio que afeta a maneira como os itens são priorizados quando eles são puxados no sistema kanban. A classe de serviço também traz uma promessa explícita para o cliente. Em seguida há um breve exemplo de um conjunto de definições de classes de serviço. Apesar desse conjunto não ser uma cópia precisa daqueles utilizados em qualquer implementação específica de Kanban, ele representa uma forte generalização de classes de serviço observadas em campo.

Neste conjunto exemplo são oferecidas quatro classes de serviço. Como um guia geral, você deve oferecer no máximo seis classes. Será muito complicado administrar e operar muitas classes. O número de classes de serviço deve ser pequeno o bastante de maneira que todos os envolvidos – membros da equipe e *stakeholders* externos – possam lembrar todas elas, e deve ser o suficiente para oferecer flexibilidade na resposta à demanda do cliente.

Expedição

A classe de serviço de Expedição (ou “Bala de Prata”) é bem compreendida na indústria de manufatura. Um cenário típico pode ser uma equipe de vendas tentando alcançar a meta de venda trimestral associada a um cliente que tem um orçamento que deve ser gasto até o final do ano fiscal. O cliente vem postergando uma decisão de compra; ele finalmente faz a escolha, mas o ano fiscal está acabando. O pedido é realizado, mas ele deve ser entregue antes do prazo acordado. O fabricante acorda o preço e a quantidade e aceita o pedido. O pedido deve ser preenchido, entregue, e enviado antes do último dia do trimestre. Um pedido como esse tipicamente chega à fábrica via o escritório do vice-presidente regional de vendas, como uma requisição de entrega de expedição, dado o curto espaço de tempo e o valor do pedido.



A habilidade de expedir dá ao vendedor a habilidade de dizer “Sim!” em circunstâncias difíceis para atender às necessidades do cliente. No entanto, expedir pedidos, afeta de maneira negativa a cadeia de fornecedores e os sistemas de distribuição da manufatura. Expedir é conhecida na engenharia e operação industrial por aumentar os níveis de estoque e aumentar os *lead times* de outros pedidos que não sejam de expedição. O negócio escolhe gerar valor numa venda específica ao custo de atrasar outros pedidos e incorrer em custos adicionais de transporte para altos níveis de estoque. Caso a companhia seja bem gerenciada, o valor gerado a partir da expedição excederá os custos

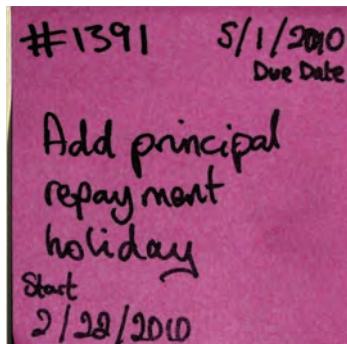
incorridos a partir de *lead times* maiores (e potencial perda de negócio como resultado) e o custo de transporte para um nível maior de estoque.

Companhias de manufatura frequentemente criam políticas para limitar o número de requisições de expedição que chegam à fábrica. Uma política comum é conceder um número fixo de “balas de prata” para o vice-presidente regional de vendas num período determinado de tempo. Portanto, o termo “bala de prata” tornou-se sinônimo de expedir na manufatura ou distribuição.

Infelizmente, como esclarecimento, o termo “bala de prata” já estava em uso pela engenharia de software. Fred Brooks a definiu como uma mudança simples (na tecnologia ou no processo) que geraria melhoria de uma ordem de magnitude (dez vezes) na produtividade do programador. Portanto, eu recomendo que você use o termo Expedição para esta classe de serviço. Nas companhias que fazem manufatura, ou nas quais o gerente sênior é familiar com manufatura, no entanto, observei que eles preferem o termo “bala de prata”. Isso é bom desde que as pessoas de tecnologia percebam a diferença do uso.

Data Fixa de Entrega

Em meados de Fevereiro de 2007, um desenvolvedor entrou no meu escritório para perguntar se eu estava ciente de um problema com uma plataforma de serviço que usávamos para processamento de cartão de crédito; eu não estava ciente, então ele explicou. Para atender à demanda de novas funcionalidades em 2006, eles tinham substituído completamente a plataforma por um novo sistema que tinha uma interface de programação de aplicação (API) completamente nova. Eles tinham informado a todos os seus clientes e deram a eles 15 meses para que o sistema antigo fosse substituído até 31 de Março de 2007. Colocando de outra maneira, se nós não atualizássemos nossos sistemas para usar a nova plataforma, nós deixaríamos de comercializar na Internet em 1 de Abril de 2007. Isto seria significativamente inconveniente para um negócio onde muito da receita era gerada por meio de vendas na *web*, sem mencionar o constrangimento para o dono da firma. Nós tínhamos apenas seis semanas para fazer as mudanças necessárias e distribuir o novo código em produção. O ticket para esse trabalho entrou no sistema kanban marcado com uma data fixa de entrega. A informação adicional no ticket visava chamar atenção para o custo e o impacto no atraso da entrega, como também para permitir que a equipe expedisse o item por ela mesma e garantisse a entrega no prazo.



Não foi a primeira vez que nós vimos uma requisição desse tipo. Tivemos uma requisição anterior a essa relacionada à integração de sistemas de TI de uma empresa adquirida. A data fixa anexada foi derivada do caso de negócio para a aquisição, o que mostrou significante economia nos custos a partir de 1 de Fevereiro do mesmo ano.

Um conceito, ou padrão, parecia estar emergindo. Algumas requisições relacionadas a grandes obrigações contratuais, algumas relacionadas a requisitos de regulamentação (usualmente do governo federal), e outras relacionadas a iniciativas estratégicas, como a aquisição de outros negócios. Requisições dessa natureza traziam um custo de atraso significante, direta ou indiretamente, que tendiam a se encaixar em uma das duas categorias: haveria uma data quando uma pena ou multa – um custo direto, específico e não reembolsável – poderia ser cobrada, imposta pela autoridade regulatória ou pelos termos estabelecidos no contrato. Alternativamente, haveria um requisito para parar alguma atividade, como a venda de um tipo de item em particular ou uma operação numa jurisdição específica, até que os requisitos fossem atendidos. Na segunda alternativa, o custo indireto é o custo da perda de uma oportunidade – a perda potencial de receita durante o período de atraso. Os dois tipos estão graficamente representados na Figura 11.2.

Negócios com calendário sazonal, como escolas e faculdades, tendem a ter restrições fortes no calendário. Se você trabalha com o setor de educação, você tem clientes para os quais você deve entregar software em momentos fixos do ano: Falhas para entregar no período esperado causam perdas nas vendas. Devemos considerar que tudo que envolve física ou culturalmente uma “janela de mercado” tem uma função de custo do atraso de primeiro grau (ou próxima disso), e deve ser tratado com uma data de entrega fixa, se esta data futura está dentro de uma janela de tempo razoável a partir

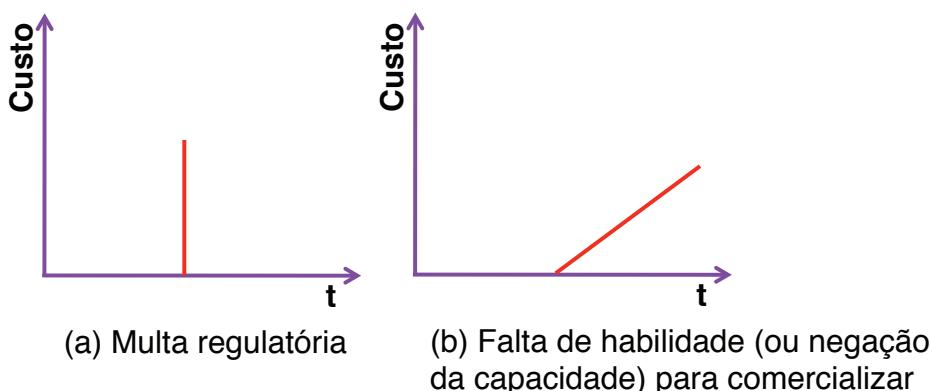


Figura 11.2 Duas funções de custo de perfil de atraso para classe de serviço de Data Fixa

da data atual.. ou culturalmente uma “janela de mercado” deve ser considerado como uma função de custo do atraso de primeiro grau, e deve ser tratado com uma data de entrega fixa.

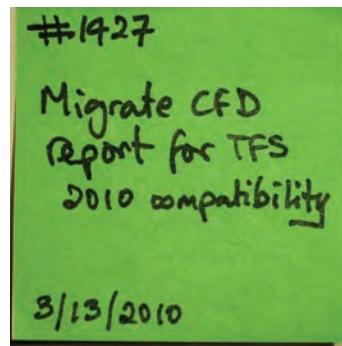
Classe Padrão

A maioria dos itens necessários com urgência deve ser tratada como itens de classe padrão. As políticas e o acordo de nível de serviço para um item de classe padrão podem variar de acordo com o tipo de item de trabalho. Um projeto de sistema kanban comum separa os tipos de trabalho por tamanho, como pequeno, médio e grande. Pode ser oferecido um acordo de serviço diferente para cada tamanho de itens de classe padrão. Por exemplo, itens pequenos são processados tipicamente dentro de quatro dias, itens de tamanho médio em um mês e itens grandes em três meses. Itens de classe padrão tendem a ter um custo de atraso tangível que pode ser calculado (embora não necessariamente em unidade monetária). O custo de atraso tende a ser imediato: se temos esta função hoje, os benefícios virão amanhã.



Classe Intangível

Faz sentido oferecer uma quarta classe de serviço menor. Eu me esforcei para encontrar um nome adequado para esta classe e estabeleci o nome “intangível”. Eu não estou completamente satisfeito com ele, dessa maneira ele pode ser alterado numa próxima edição desse livro. Itens de classe intangível podem ser importantes e valiosos, mas não há um custo de atraso tangível associado a eles num futuro próximo. Isto é, não há custo de atraso dentro do prazo de entrega do item. Requisições que atendem a esse padrão são as entregas que possuem uma data fixa de entrega em potencial, mas é uma data que está num futuro distante, como uma substituição de plataforma.



Por exemplo, em 2005 a Microsoft lançou o SQL Server 2005, a última versão do seu servidor de banco de dados RDBMS. A edição de 2005 substituiu a edição de 2000, a qual se tornou “fim de vida”. Como *player* dominante do Mercado, era requerido

que a Microsoft desse suporte aos seus produtos por dez anos após eles terem sido lançados. Como resultado disso, o suporte ao SQL Server 2000 deveria continuar até 2010. Isto deu aos clientes um período de cinco anos para substituir o código que era incompatível com a edição mais nova, 2005 (ou 2010) da plataforma. Então substituir o código em 2005 ou 2006 – *stored procedures*, código persistente – não tem uma prioridade de tempo crítico. Não há custo de atraso incorrido nestes anos. No entanto, com o passar do tempo e se o código não é alterado, o custo aumenta. Torna-se cada vez mais difícil trabalhar com outros produtos, visto que eles requerem novas versões do SQL Server 2005 como pré-requisito. Ocorre mais e mais pressão para se fazer a troca para a nova plataforma. Em 2009 o problema é urgente, visto que a Microsoft parará de dar suporte ao antigo produto, e falhas para atualização farão com que o negócio seja executado em máquinas antigas com sistemas operacionais e infra-estruturas não suportadas. Se o risco é inaceitável, o código deve ser atualizado. Substituição de plataforma é um problema comum que equipes de engenharia de software lutam continuamente. Há um desejo de começar logo o trabalho e completá-lo no tempo adequado, mas a capacidade para fazer o trabalho de atualização é deslocada por outro trabalho mais urgente ou crítico. Em outras palavras, substituição de plataforma, embora tenha um custo de atraso imediato baixo, é deslocada por outros trabalhos com um custo de atraso maior e mais imediato.

Faz sentido oferecer uma classe de serviço que permitirá que esse tipo de trabalho seja empreendido o quanto antes. Pode-se alocar capacidade para o trabalho a fim de garantir que ele seja finalizado; no entanto, pode não haver garantia de tempo. Mais do que isso, é este custo de atraso baixo do trabalho que sempre o colocará de lado quando mais requisições de urgência aparecerem. Para existir uma folga para processar uma requisição de expedição, deve existir trabalho de custo de atraso baixo que pode ser deslocado enquanto a requisição de expedição é processada. Os itens de classe intangível fornecem essa folga.

Políticas para Classe de Serviço

Uma técnica de visualização deve ser empregada para prontamente se identificar uma classe de serviço. Como mencionado anteriormente, cores diferentes de *tickets* ou raias diferentes na parede de cartões são as mais comuns. Algumas equipes têm usado decorações como adesivos de estrelas colados ao ticket do item de trabalho. Uma raia para requisições de expedição é também uma escolha comum. Use um critério seu para como visualizar suas classes de serviço. Este capítulo usa cores diferentes para mostrar classe de serviço. O objetivo é garantir que qualquer membro da equipe, em qualquer

dia, possa usar políticas simples de priorização associadas a uma classe de serviço para tomar decisões de priorização de qualidade sem intervenção ou supervisão da gerência.

A seguir há um exemplo de políticas de priorização para as quatro classes de serviço definidas previamente. Naturalmente, para toda implementação, as definições de classe de serviço serão únicas e suas políticas de uso serão diferentes destes exemplos. Estas políticas são baseadas em evidência empírica e elas refletem com bastante precisão políticas que equipes reais têm utilizado.

Políticas de Expedição

- Requisições de expedição usam cartões brancos.
- Apenas uma requisição de expedição é permitida em qualquer momento. Em outras palavras, uma classe de serviço de Expedição tem um limite de *WIP* igual a 1.
- Um recurso qualificado deve puxar requisições de Expedição imediatamente. Outro trabalho será colocado *on hold* para processar a requisição de expedição.
- Em qualquer ponto do fluxo de trabalho, o limite *WIP* pode ser excedido para acomodar uma requisição de expedição. A capacidade não é mantida em reserva para expedição.
- Se necessário, um *release* especial (fora do ciclo) será planejado para colocar uma requisição de expedição em produção o mais cedo possível.

Políticas de Data de Entrega Fixa

- Itens com data de entrega fixa usam cartões roxos.
- A data de entrega exigida é apresentada no canto direito abaixo do cartão.
- Itens com data de entrega fixa são analisados e devem ser realizadas estimativas de tamanho e esforço para avaliar o tempo de fluxo. Se o item é grande pode ser necessário quebrá-lo em itens menores; cada item menor será avaliado independentemente a fim de verificar se ele se qualifica como um item de data de entrega fixa.
- Itens com data de entrega fixa são mantidos no *backlog* até eles serem selecionados para a fila de entrada, próximo ao ponto ideal no tempo no qual ele deve ser entregue no prazo, dado a estimativa do tempo de fluxo.

- Itens com data de entrega fixa são puxados preferencialmente em relação a outros itens de menor risco. Neste exemplo, eles são puxados antes dos itens de classe padrão ou intangível.
- Itens com data de entrega fixa devem aceitar o limite *WIP*.
- Itens com data fixa de entrega enfileiram para *release* quando eles estão finalizados e prontos para *release*. Eles são released regularmente um pouco antes da data de entrega exigida.
- Se um item com data de entrega fixa fica para trás, e entregar o *release* na data desejada é um risco, sua classe de serviço deve ser promovida para uma requisição de expedição.

Políticas de Classe Padrão

- Itens de classe padrão usam cartões amarelos.
- Itens de classe padrão são priorizados para a fila de entrada baseado num mecanismo acordado previamente, como uma votação democrática, e são selecionados tipicamente baseando-se no seu custo de atraso ou valor de negócio.
- Itens de classe padrão usam o mecanismo de fila *first in, first out (FIFO)* a partir do momento que são puxados pelo sistema. Tipicamente, quando uma opção é dada, um membro da equipe puxa o item de classe padrão mais antigo se não há um item de expedição ou com data fixa para ser escolhido preferencialmente.
- Itens de classe padrão enfileiram para *release* quando eles estão finalizados e prontos para *release*. Eles são released no próximo *release* agendado.
- Não é realizada estimativa para determinar o nível de esforço ou tempo de fluxo.
- O tamanho dos itens de classe padrão deve ser analisado. Itens grandes devem ser quebrados em itens menores. Cada item deve ser enfileirado e fluir separadamente.
- Itens de classe padrão são geralmente entregues dentro de x dias da seleção com um desempenho de entrega de m por cento.

Um acordo de nível de serviço típico para uma classe padrão deve oferecer um *lead time* de 30 dias com um desempenho de 80% no desempenho de entrega no prazo. Em outras palavras, quatro de cinco requisições devem ser entregues em 30 dias.

Classe Intangível

- Itens de classe intangível usam cartões verdes.
- Itens de classe intangível são priorizados na fila de entrada baseado num mecanismo acordado previamente, como uma votação democrática, e são selecionados tipicamente baseando-se em algum impacto em longo prazo ou custo de atraso.
- Itens de classe intangível são puxados pelo sistema de maneira *ad hoc*. Membros da equipe podem escolher puxar um item de classe intangível desconsiderando sua data de entrada, enquanto um item de maior classe não está disponível.
- Itens de classe intangível enfileiram para *release* quando são finalizados e prontos para *release*. Eles são released no próximo *release* agendado ou ficam em espera para serem agrupados a outros itens.
- Não é realizada estimativa para determinar o nível de esforço ou tempo de fluxo.
- O tamanho dos itens de classe intangível deve ser analisado. Itens grandes devem ser quebrados em itens menores. Cada item deve ser enfileirado e fluir separadamente.
- Tipicamente, um item de classe intangível é colocado de lado para se processar uma requisição de expedição.
- Pode não ser necessário oferecer um acordo de nível de serviço para itens de classe intangível. Se for necessário, deve ser um acordo significativamente mais frouxo do que aquele oferecido para itens de classe padrão, por exemplo, 60 dias com 50 por cento de desempenho para entrega no prazo.

Determinando Meta para Entrega de Serviço

No conjunto de classes exemplo acima, a classe de serviço Padrão usou uma meta para o *lead time*, por exemplo, de 28 dias (4 semanas). O conceito de oferta de uma meta para o *lead time*, associado à métrica de desempenho de entrega no prazo é uma alternativa para tratar cada item individualmente e se ter uma estimativa e compromisso para a data de entrega de cada item. O acordo de nível de serviço nos permite evitar atividades custosas, como estimativas; atividades que reduzam a confiança, como assumir compromissos; e propagar risco agregando um conjunto grande de requisições e comprometendo-se apenas com o desempenho global sob a forma de desempenho

percentual de entrega no prazo. Ao evitar compromissos que não seremos capazes de cumprir, evitamos o perigo de perder a confiança de nossos clientes. No entanto, é importante comunicar que a meta do *lead time* para a classe de serviço Padrão é apenas isso, uma meta!

Dados históricos ajudam a determinar a meta do *lead time*. Caso você não tenha dados históricos, dê um palpite razoável. Caso você tenha, então o meio mais científico para determinar a meta do *lead time* é processar o *lead time* (a partir da seleção do item até a entrega) através de um controle-processo estatístico ou uma ferramenta kanban de acompanhamento que dê suporte a controle de processo estatístico (como a Sylvan Catalyst) e usar o limite de controle máximo (o limite *plus-3 six sigma*) para o seu *lead time*. Isso garante uma meta que você pode alcançar sob circunstâncias mais normais e perdê-la apenas quando há um problema genuíno de causa atribuível. (veja capítulo 19 para uma explanação mais detalhada).

Se, por outro lado, o ultimo parágrafo não significou nada para você, então uma explicação mais leiga é que você precisa que o *lead time* seja atingível a maior parte do tempo, mas que também seja agressivo o bastante para manter a equipe focada. É provável que os seus itens de trabalho variem em tamanho, complexidade, risco, e *expertise* requeridos; dessa maneira, o *lead time* varia significativamente. Tudo bem. Se você realiza uma análise espectral de alguns dados históricos e verifica que talvez 70 por cento dos itens sejam entregues em 28 dias, e os 30 por cento restantes espalham-se por 100 dias, então talvez seja razoável sugerir uma meta para data de entrega de 28 dias.

Eu aprendi que o uso de classes de serviço é uma técnica poderosa. Com a minha equipe de 2007, aproximadamente 30 por cento de todas as requisições atrasavam comparando à meta do *lead time*. Nós relatamos isso como a métrica de Desempenho do Prazo. Ela nunca foi acima de 70 por cento. No entanto, apesar desse mau desempenho em relação à data de entrega, nós tínhamos poucas reclamações. As razões para isso tornaram-se evidentes: Todos os itens importantes – aqueles com risco ou valor alto – eram sempre entregues na data, e havia confiança que os últimos seriam entregues dentro de duas ou quatro semanas adicionais, visto que as entregas aconteciam com uma regularidade confiável.

As classes de serviço de Expedição e Data de Entrega Fixa estavam garantindo que itens importantes fossem sempre entregues no prazo. Enquanto isso, os outros itens da classe Padrão atrasavam em apenas um ou dois *releases* (14 ou 28 dias, respectivamente). Os clientes confiavam na cadência de *release*; a confiança foi conquistada pela ação. Nós entregávamos consistentemente um *release* toda segunda quarta-feira. Com o custo de atraso insignificante associado a muitos itens de classe Padrão (e Intangível, visto que eles não foram delineados separadamente), o negócio focou no que tinha

que ser entregue e no planejamento de itens futuros em vez de se preocupar muito em relação a datas precisas de entrega para trabalho-em-progresso.

Este resultado foi significante porque Kanban com classes de serviço claramente mudou a psicologia do cliente e alterou significativamente a natureza do relacionamento e as expectativas. Os clientes foram orientados a uma relação de longo prazo e ao desempenho do sistema, e não a entrega de um item ou itens específicos. Isto deu liberdade à equipe de desenvolvimento para focar nas coisas certas e não desperdiçar tempo com questões que surgiam a partir de um baixo nível de confiança entre a equipe e o cliente.

Estabelecendo uma Classe de Serviço

A classe de serviço para um item deve ser estabelecida quando o item é selecionado para a fila de entrada. Se um item é uma requisição de expedição, isto deve ser evidente – isso vem com uma requisição para o item ser processado o mais cedo possível. A justificativa baseia-se no caso de negócio que mostra alguma oportunidade imediata ou identifica um custo significante que será incorrido se a requisição não for realizada, talvez o custo já esteja sendo incorrido. Isto é típico com defeitos de produção de severidade alta.

Se um item tem uma data de entrega fixa isto também deve ser evidente por sua natureza. Talvez a requisição esteja relacionada a um novo requisito regulatório previsto por uma autoridade regulatória independente, ou a uma natureza sazonal do negócio. Se um item é da classe de data fixa, a data é tipicamente conhecida, e encontra-se dentro de um período de tempo razoável – talvez duas vezes mais longo do que a meta de *lead time* da classe de serviço padrão – e o item deve ter sido estimado de modo a ser introduzido no ponto ideal para garantir entrega no prazo.

Uma escolha difícil é se o item é da classe padrão ou intangível. Minha observação é que itens de classe padrão seguem uma função de custo de oportunidade que faz efeito imediatamente. Por exemplo, se tivéssemos essa nova *feature* hoje, poderíamos estar fazendo dinheiro a partir dela amanhã. Então entregar antes é desejável, mas atrasar não acarreta a mesma pena, como algo de natureza de data fixa ou uma requisição de expedição.

Itens intangíveis tendem a ser associados com itens importantes e de valor que têm um (oportunidade) custo de atraso que não traz efeito num futuro próximo. Tipicamente, a função de custo faz uma curva de inflexão para cima em trimestres, ou anos, no futuro. Isto está bem além de um horizonte de planejamento imediato, isto é, duas ou três vezes um *lead time* típico. Se o nosso *lead time* atual é geralmente 28

dias, então nosso horizonte de planejamento é talvez de três meses. Itens que incorrerão numa perda de oportunidade ou num custo tangível dentro de três meses devem ser tratados como classe padrão, enquanto itens no qual o custo ou benefício não é visualizado dentro de trimestres ou anos no futuro devem ser tratados como itens de classe intangível.

Colocando Classes de Serviço em Uso

Classes de serviço devem ser definidas para cada sistema kanban. As políticas associadas com cada classe de serviço devem ser explicadas para todos os membros da equipe. Todos que participam da *standup meeting** pela manhã devem gostar e entender as classes de serviço em uso. Para isso se tornar efetivo, o número de classes de serviço deve ser mantido relativamente pequeno – quatro a seis é uma boa diretriz. E novamente, como queremos que cada membro da equipe lembre as classes de serviço, o que elas significam, e como usá-las, o número de políticas para cada classe de serviço deve ser mantido pequeno e as políticas devem ser simples. As definições devem ser

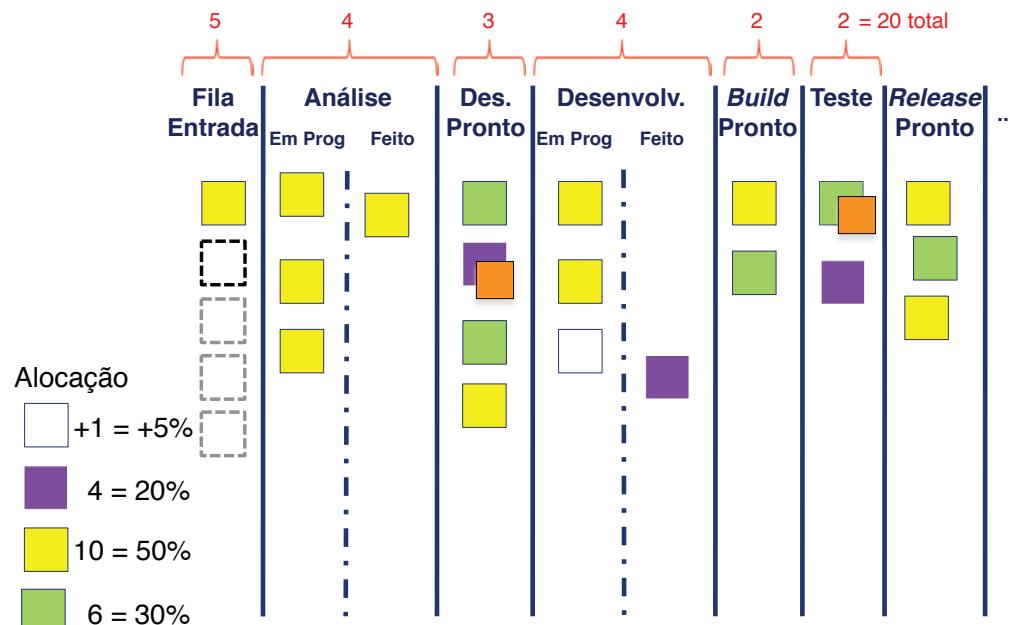


Figura 11.3 Parede de cartões mostrando capacidade de alocação através das classes de serviço

* N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: reunião em pé.

precisas e sem ambiguidade. Novamente, uma boa diretriz deve ser não mais de seis políticas por classe de serviço.

Armados com um entendimento das classes de serviço e das políticas associadas a elas, a equipe deve ter poder para auto-organizar o fluxo de trabalho. Itens de trabalho devem fluir através do sistema de uma maneira a otimizar o valor do negócio e o serviço ao cliente, tendo como resultado *releases* de software que maximizam a satisfação do cliente.

Aloque Capacidade para Classes de Serviço

Figura 11.3 mostra um sistema kanban com um limite de *WIP* total de 20. Classes de serviço são designadas por quatro cores de *tickets*. Os *tickets* brancos de expedição não contam para o limite de *WIP*, mas são limitados a apenas um item de cada vez. Portanto, eles têm um impacto de cinco por cento na capacidade total quando presentes, e aumentam o trabalho-em-progresso para 21 itens. Neste exemplo, os *tickets* roxos para data de entrega fixa representam 20 por cento do total. Isto significa que podem existir apenas quatro *tickets* roxos no quadro a qualquer momento, mas eles podem estar presentes em qualquer coluna. Itens de classe padrão amarelos representam 50 por cento do total de alocação, com um total de 10 itens. Os 30 por cento restantes são alocados para os itens verdes de classe intangível.

Agora que alocamos capacidade para as diferentes classes de serviço, a atividade de preenchimento da fila de entrada torna-se complicada pela capacidade disponível para cada classe. Como apresentado, há capacidade para um item de data de entrega fixa e três para itens de classe intangível; isto gera muitas perguntas. E se não tivermos demanda atual para um item de data fixa? O que devemos fazer? Devemos preencher o espaço vazio com um item de classe padrão? Caso positivo, o *status* desse item deve ser de data fixa ou deve ser tratado como um item de classe padrão? Se fizermos isso, não estaremos ajustando a política de alocação de capacidade?

Todas essas questões legítimas e problemas legítimos destacados vêm à tona diariamente quando usamos um sistema kanban. Não há respostas certas ou erradas para estas questões. As respostas precisam ser derivadas no contexto e são específicas para cada situação.

O que podemos deduzir da alocação escolhida é que o domínio tem um número significante de itens de natureza de data fixa e que há também uma capacidade para itens intangíveis. Isto deve implicar que há algumas iniciativas maiores com datas de entrega de maior prazo – como uma substituição de plataforma – a caminho. Isso também pode indicar riscos significantes no domínio. Talvez haja uma natureza sazonal

para a demanda o que aumenta o número de requisições de expedição ou itens de data fixa. Para ser capaz de responder de maneira adequada a esta demanda sazonal sem causar aumento da insatisfação do cliente, escolhemos alocar mais capacidade para itens intangíveis em vez de itens de classe padrão, dessa forma estamos construindo mais folga no sistema.

Escolher o que fazer quando a fila de entrada tem um espaço vazio para um item de data fixa quando não há itens de data fixa disponíveis depende dos riscos presentes no domínio. Se há uma demanda significante para itens de data fixa e o custo associado a estes itens é alto (portanto o risco é alto), devemos escolher deixar o espaço vazio. Isso pode fazer sentido para efetivamente reservar capacidade para um futuro item de data fixa. Se posteriormente existir um item de data fixa, podemos escolher retirar o item de classe padrão ou exceder o limite de WIP temporariamente. Todas estas escolhas terão efeitos diferentes no lead time, no desempenho de entrega no prazo, propagação de variabilidade no lead time, satisfação do cliente, e gerenciamento de risco. Você precisa tomar essas decisões por você mesmo; levará algum tempo para desenvolver uma experiência e discernimento adequados para permitir que você faça as melhores escolhas para a sua equipe, projeto, ou organização.

Alocação de capacidade é apenas outra estratégia do sistema kanban. Se você acredita que sua alocação não está alinhada com a demanda, então a ajuste – mude as políticas e ajuste os limites de WIP de acordo com as mudanças.

Takeaways

- ❖ Classes de serviço fornecem um método simples para otimizar a satisfação do cliente.
- ❖ Itens de trabalho devem ser atribuídos para uma classe de serviço de acordo com o impacto no negócio.
- ❖ Classes de serviço devem ser claras, apresentadas visualmente com o uso, por exemplo, de cartões de cores diferentes ou raias diferentes na parede de cartões.
- ❖ Deve ser definido um conjunto de políticas de gerenciamento para cada classe de serviço. Apenas classes de serviço relacionadas a itens de maior risco devem envolver atividades que desperdiçam tempo como estimativa.
- ❖ Os membros da equipe devem ser treinados para que entendam as classes de serviço e as políticas associadas a elas.
- ❖ Algumas classes de serviço devem incluir uma meta de *lead time*.
- ❖ O desempenho de entrega no prazo (percentual) deve ser monitorado para metas de *lead time*.
- ❖ Classes de serviço possibilitam uma auto-organização, delegam poder aos membros da equipe, e liberam tempo da gerência para focar no processo em vez do trabalho.
- ❖ Classes de serviço mudam a psicologia do cliente.
- ❖ Se as classes de serviço são utilizadas apropriadamente e combinadas com uma cadênci a de entrega regular, poucas reclamações serão recebidas, até mesmo se uma porção significante dos itens não alcançarem sua meta de *lead time*.
- ❖ A capacidade do sistema Kanban deve ser alocada para cada classe de serviço.
- ❖ O percentual de capacidade alocada para cada classe de serviço deve estar alinhado com a demanda.

❖ CAPÍTULO 12 ❖

Métricas e Relatórios de Gerenciamento

Embora a ideia é que Kanban seja minimamente invasivo e altere o menos possível a cadeia de valor, papéis, e responsabilidades; ele altera a maneira como a equipe interage com os seus parceiros – os *stakeholders* externos. Por causa disso, Kanban precisa reportar métricas um pouco diferentes das que você deve estar acostumado numa abordagem tradicional ou num gerenciamento de projeto Ágil.

O fluxo contínuo do sistema Kanban significa que estamos menos interessados em reportar se um projeto está “*on-time*”* ou se um plano específico está sendo seguido. O que é importante mostrar: que o sistema Kanban é previsível e está funcionando como projetado, que a organização tem agilidade no negócio, que há um foco no fluxo, e que existe um desenvolvimento claro de melhoria contínua.

Para previsibilidade queremos mostrar o quanto nosso desempenho é bom com as classes de serviço. Itens de trabalho são tratados apropriadamente, e, se a classe de serviço tem uma meta de *lead time*, o quanto estamos conseguindo atendê-la? O que é desempenho de entrega no prazo?

Para cada um dos nossos indicadores, queremos acompanhar a evolução ao longo do tempo, de maneira que possamos ver a propagação da variação. Se quisermos demonstrar melhoria continua

* N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: dentro do prazo.

precisamos que a tendência melhore com o passar do tempo. Se quisermos demonstrar melhoria na previsibilidade, precisamos que a propagação da variação diminua e que o desempenho de entrega no prazo aumente.

Monitorando WIP

Antes de chegarmos aos indicadores de desempenho, no entanto, acredito que a métrica mais fundamental deve mostrar que o sistema kanban está funcionando apropriadamente. Para fazer isto, precisamos de um diagrama de fluxo acumulativo que mostra as quantidades de trabalho-em-progresso em cada estágio do sistema. Se o sistema kanban está fluindo corretamente, as faixas do gráfico devem ser regulares e os tamanhos devem ser estáveis.

O exemplo do gráfico na Figura 12.1 mostra quão bem a equipe está mantendo os

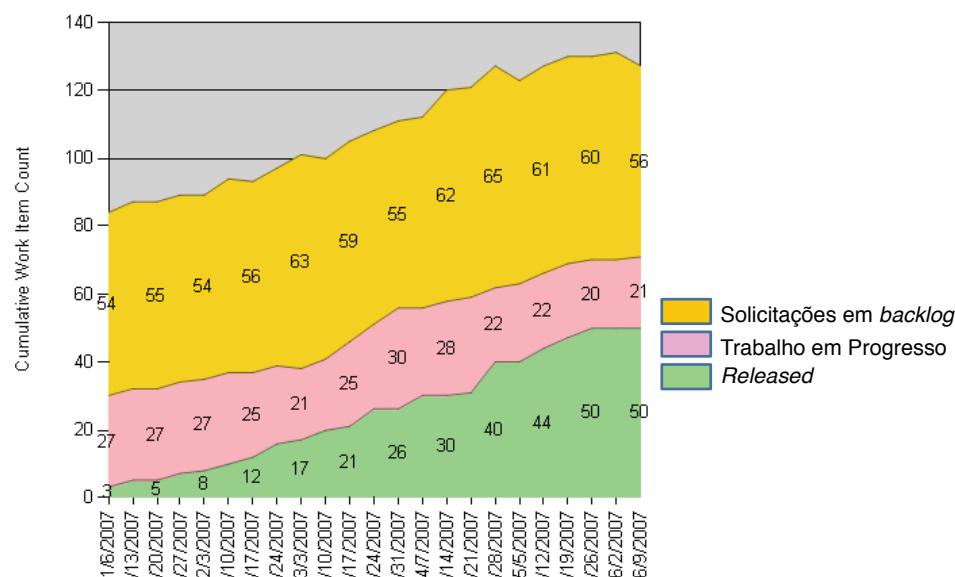


Figura 12.1 Exemplo de um diagrama de fluxo cumulativo de um Sistema Kanban

limites de *WIP*. Podemos ver que o *WIP* (a faixa de cor clara no meio) está crescendo no meio do período de tempo. No começo, o limite de *WIP* é 27; no final do período, devido a um ajuste pessoal, o limite de *WIP* é 21. Também podemos ver a média do *lead time* escaneando o gráfico horizontalmente.

Lead Time

A próxima métrica de interesse indica o quanto previsível é nossa organização na entrega, considerando as definições de classe de serviço. A métrica fundamental para isto é o *lead time*. Se um item foi expedido, quão rápido ele foi do pedido até a produção? Se ele era de uma classe padrão, nós o entregamos dentro da meta de *lead time*? Descobri que a melhor maneira para mostrar estes dados é com um espectro de

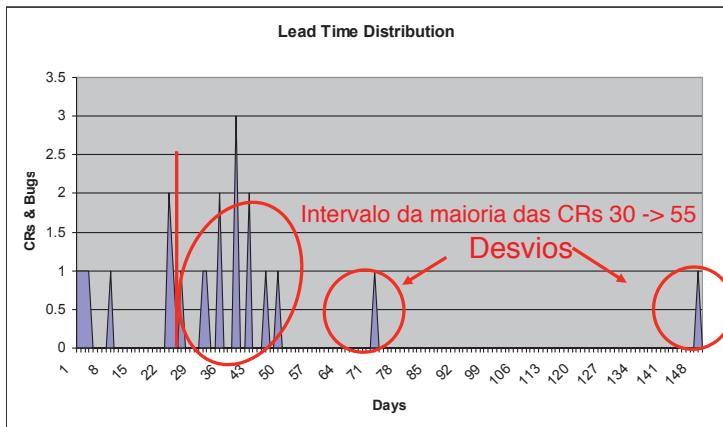


Figura 12.2 Exemplo de uma análise espectral do *Lead Time*

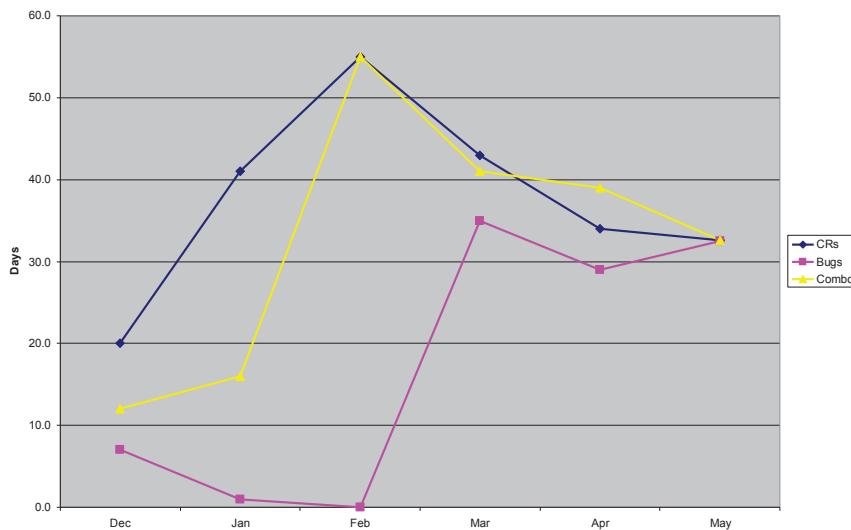


Figura 12.3 Exemplo de uma Tendência Média do *Lead Time*

Lead Time and Due Date Percentages		Lead Time (Average # of Days)			Due Date Performance (%)	
Interval	Target	May 2007	Dec 2006 to May 2007	May 2007	Dec 2006 to May 2007	
Lead Time, Engineering Ready to Release (CRs & Bug Fixes)	30	32.5	31.1	52	50	
Lead Time, Engineering Ready to Release (CRs Only)	30	32.6	40.4	50	30	
Lead Time, Engineering Ready to Release (Bugs Only)	30	32.5	19.6	55	75	

Figura 12.4 Exemplo de um relatório mostrando a média do *lead time* e o desempenho da data na entrega

análise do *lead time*, apresentando num gráfico a meta de *lead time* versus o acordo de nível de serviço (SLA) para uma classe de serviço (veja Figura 12.2).

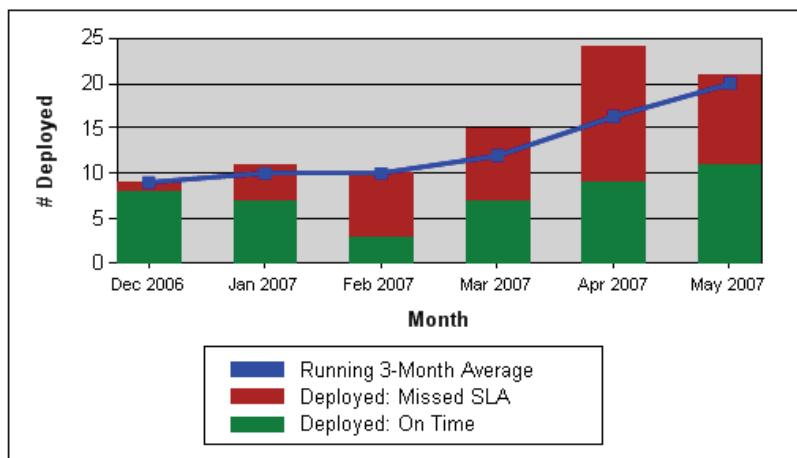
Reportar a taxa de *lead time* (media) tem utilidade como um relatório do desempenho geral do sistema (veja Figura 12.3), mas não é muito útil como um indicador de previsibilidade ou como um meio para informar oportunidades de melhoria.

O espectro de análise é muito mais útil em nos informar sobre itens que falharam em atingir a meta de tempo e outros desvios estatísticos. No exemplo mostrado na Figura 12.4, faria sentido investigar a causa raiz do grupo de itens que falharam em atingir a meta. Se a causa raiz puder ser tratada, o Desempenho de Entrega na Data (percentual de itens entregues como esperado) deve melhorar.

Desempenho de Entrega na Data

Acredito que seja útil reportar Desempenho de Entrega na Data para os meses mais recentes e para o ano da data. Você pode também reportar desempenho ano a ano (ou de 12 meses atrás) para comparação; portanto, é útil se ter 13 meses de dados.

Você pode incluir os itens da classe de serviço de Data de Entrega Fixa na métrica de Desempenho de Entrega na Data. Neste caso, você estará respondendo à seguinte pergunta, “O item foi entregue na data?” No entanto, embora você tenha o *lead time* registrado, ele não é tão útil quanto comparar o *lead time* estimado e o atual. Estimado versus o atual demonstra o quanto previsível a equipe é e quão bem ela está trabalhando com itens de serviço de Data Fixa de Entrega. Lembre-se que itens de Data Fixa são analisados e estimados. O desempenho da data de entrega com itens de data fixa é um fator determinante da qualidade da estimativa inicial. Naturalmente, a métrica mais importante é se o item foi entregue antes da data limite. A precisão da estimativa é um indicador da eficiência de execução do sistema. Se as estimativas são conhecidas por serem imprecisas, a equipe tenderá a iniciar itens de data fixa em primeiro lugar

Throughput And Production Rate:**Figura 12.5** Exemplo de um gráfico de barra mostrando o rendimento

para garantir a entrega; isto não é o ideal. O desempenho geral em termos de valor e rendimento pode ser melhorado através da melhoria nas estimativas.

Rendimento

Rendimento deve ser reportado como o número de itens – ou alguma indicação do seu valor – que foram entregues num período de tempo determinado, como um mês. Rendimento deve ser reportado como uma tendência ao longo do tempo, como mostrado na Figura 12.5. O objetivo é melhorá-lo continuamente. Rendimento é muito similar à métrica Ágil “velocidade”. Ele indica quantas *user stories*, ou *story points*, foram finalizados num período de tempo determinado. Se você não está usando técnicas de requisitos Ágeis, mas utilizando outras coisas, como itens de especificação funcional, requisição de mudança, casos de uso, etc., então reporte esses números.

Em primeira instância é importante ser capaz de reportar o número bruto. À medida que a sua equipe amadurecer e tornar-se mais sofisticada, você será capaz de reportar o tamanho relativo, como o número total de *story points*, pontos de função, ou alguma outra medida de qualidade. Se a sua organização é muito sofisticada, você poderá reportar o valor monetário do trabalho entregue. No período da escrita desse livro eu sabia de apenas uma equipe, na BBC de Londres, que era capaz de reportar o valor em dólares do trabalho entregue.

Dado de rendimento é usado em Kanban com um propósito inteiramente diferente do que velocidade num ambiente típico de desenvolvimento Ágil. Rendimento não é

How many issues and blocked work items do we have?

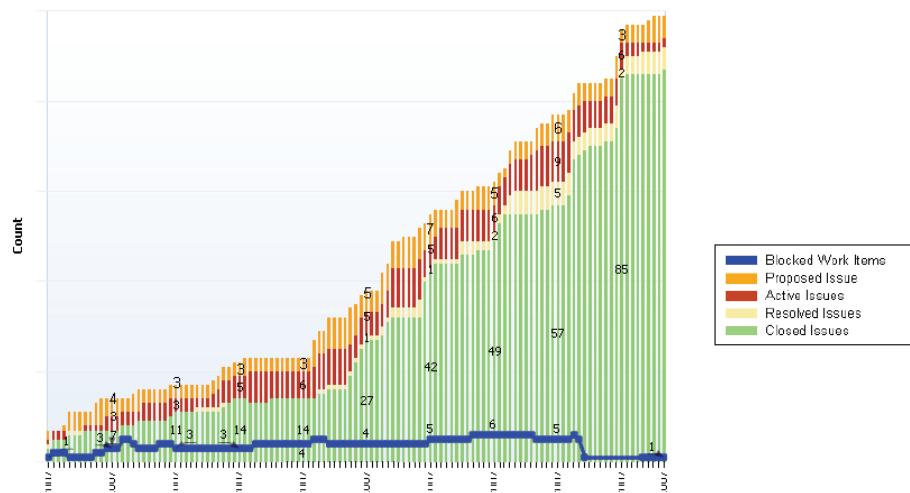


Figura 12.6 Exemplo de um gráfico de Problemas e Itens de Trabalho Bloqueados

usado para prever a quantidade de entregas em um intervalo de tempo ou para qualquer compromisso específico de entrega. Rendimento é usado como um indicador de quanto bem o sistema (a equipe e a organização) está executando e para demonstrar melhoria contínua. Compromissos são feitos em Kanban para o *lead time* e as metas de entrega na data. Rendimento pode ser usado em grandes projetos para indicar o tempo aproximado para finalização com um *buffering* apropriado para variação.

Problemas e Itens de Trabalho Bloqueados

O gráfico *The Issues and Blocked Work Items* mostra um diagrama de fluxo acumulado de impedimentos reportados sobreposto a um gráfico da quantidade de trabalho-em-progresso que se tornaram bloqueados, como mostrados na Figura 12.6. Este gráfico nos dá uma indicação de quanto boa é a organização em identificar, reportar, e gerenciar problemas bloqueados e os seus impactos. Se o Desempenho de Entrega na Data é ruim, haverá uma evidência correspondente no gráfico mostrando que alguns impedimentos foram descobertos e não resolvidos rápido o suficiente. Este gráfico pode ser usado dia a dia para alertar a gerência sênior dos impedimentos e seus impactos. Ele pode ser usado também como um relatório de longo prazo para indicar a capacidade da organização em resolver impedimentos e manter as coisas fluindo – uma medida da capacidade em gerenciamento e resolução de problemas.

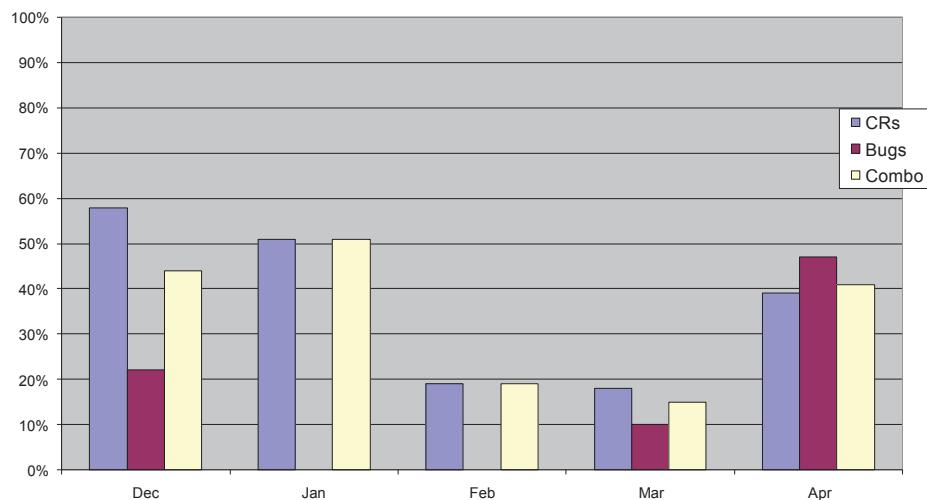


Figura 12.7 Exemplo de *lead time*: taxa de tempo atribuída

Eficiência do Fluxo

Um bom indicador *Lean* de desperdício no sistema é medir o *lead time* em relação ao “tempo de toque”. Na manufatura, tempo de toque se refere ao tempo que um trabalhador gasta realmente tocando um trabalho. Em desenvolvimento de software, isto é muito difícil de medir. No entanto, a maioria dos sistemas de acompanhamento consegue acompanhar o tempo atribuído (a um indivíduo) em relação ao tempo gasto bloqueando e enfileirando. Portanto, embora reportar a relação do *lead time* a um tempo atribuído não nos traga uma indicação precisa do desperdício no sistema, ela nos traz uma relação conservadora que mostra o potencial que existe para melhoria, como na Figura 12.7.

Não fique alarmado caso essa relação inicial seja algo em torno de 10:1. Eu participei de muitas conferências e vi participantes de indústrias diversas, como projeto de aeronaves e projeto de equipamentos médicos reportarem relações similares. Parece que com trabalho de conhecimento nós somos terrivelmente ineficientes e incapazes de uma agilidade necessária para tornar uma ideia ou requisição em um produto de trabalho eficientemente.

A métrica de eficiência do fluxo não é muito útil dia a dia, mas, novamente, pode ser outro indicador de melhoria contínua.

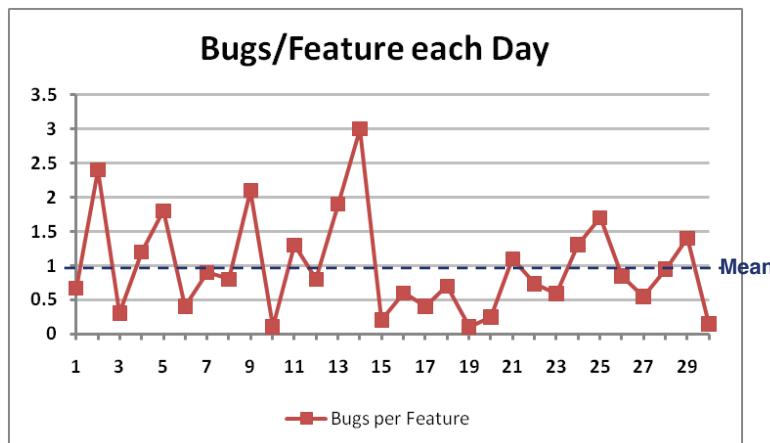


Figura 12.8 Gráfico mostrando defeitos por feature

Qualidade Inicial

Defeitos representam custo de oportunidade e afetam o *lead time* e o rendimento do sistema Kanban. Faz sentido reportar o número de defeitos escapados como um percentual em relação ao *WIP* total e rendimento. No decorrer do tempo, queremos ver a taxa de defeito próxima a zero, como mostrado na Figura 12.8.

Carga de Falhas

Carga de falhas acompanha quantos itens de trabalho nós processamos devido à baixa qualidade de itens anteriores – quantos itens de trabalho são defeitos de produção ou novas *features* que foram requisitadas através do serviço ao cliente da organização devido à usabilidade fraca ou falhas em antecipar necessidades do usuário apropriadamente. Idealmente, a carga de falhas deve cair ao longo do tempo. Isto é um bom indicador de que estamos melhorando como um todo na organização e pensando no nível de sistema.

Takeaways

- ❖ Acompanhe *WIP* com um diagrama de fluxo acumulado para monitorar dia a dia os limites *WIP*.
- ❖ Acompanhe *lead time* para cada item processado e reporte a média e o espectro de análise para cada classe de serviço.
- ❖ *Lead time* é um indicador da agilidade do negócio.
- ❖ Acompanhe *lead time* estimado versus atual para itens de classe de serviço de Data de Entrega Fixa.
- ❖ Reporte Desempenho de Entrega na Data como um indicador de previsibilidade.
- ❖ Impedimentos bloqueiam o fluxo e impactam o *lead time* e o desempenho de entrega na data; reporte problemas bloqueados e o número de itens de trabalho bloqueados em um diagrama de fluxo acumulado sobrepondo um gráfico de itens bloqueados. Use-o para indicar a capacidade de reportar e resolver problemas rapidamente.
- ❖ Eficiência de fluxo é a relação do *lead time* a um tempo atribuído de engenharia. Ele indica o quanto eficiente é a organização em processar trabalho novo, e é um indicador secundário da agilidade do negócio. Ele indica também o espaço disponível para melhoria sem a necessidade de mudar métodos de engenharia.
- ❖ Relatórios de Qualidade Inicial reportam o número de defeitos descobertos pelos testadores dentro do sistema e indica quanta capacidade está sendo gasta devido à baixa qualidade inicial.
- ❖ Carga de Falha reporta o percentual de trabalho que é gerado devido a alguma falha do sistema e mostra a capacidade que poderia estar sendo usada para *features* novas e de valor agregado.

❖ CAPÍTULO 13 ❖

Escalando Kanban

Até agora os exemplos e histórias de implementações Kanban apresentados focaram na manutenção de software – pequenas mudanças de sistema feitas com *releases* para produção rápidos e frequentes. Há muitos sistemas que estão em manutenção, e uma porção significante dos leitores envolvidos em desenvolvimento de software achará os conselhos e guias úteis. Igualmente, há muito mais pessoas de TI envolvidas em suporte e operação onde sistemas de ticketing para pedidos pequenos de trabalho são também comuns; uma abordagem Kanban será igualmente útil para eles. No entanto, há outros cujo desenvolvimento de projetos de tamanho significante é a norma. Se você está lendo isto se perguntando por que e como eu uso Kanban em grandes projetos e através de um portfólio de projetos, espero que o capítulo 5 tenha convencido você que Kanban possibilita mudanças culturais significantes e positivas. Os benefícios observados com Kanban são suficientemente desejáveis nos desafiando a fazer a seguinte pergunta, “Como devemos fazer Kanban em grandes projetos?”

Grandes projetos apresentam desafios consideráveis. Muitos requisitos a serem entregues juntos. Serão necessários alguns meses até que uma entrega seja realizada. O tamanho da equipe é grande. Muitos trabalhos estarão acontecendo em paralelo. Pedaços significantes de trabalho podem precisar ser integrados. Por exemplo, documentação e pacotes de projeto podem precisar ser integrados com o *build* final do software antes que um *release* seja realizado.

Como lidar com estes desafios?

A resposta é olhar para os primeiros princípios. O primeiro princípio de Kanban é limitar o trabalho-em-progresso e puxar trabalho usando um sistema visual de sinalização. Além disso, olharmos para os princípios *Lean*, princípios Ágeis, e o fluxo de trabalho e o processo que já estão em uso como nosso ponto inicial. Então queremos limitar o *WIP*, usar controles visuais e de sinalização, e puxar trabalho apenas quando há capacidade para se fazer isso; mas também queremos pequenas transferências de *batch*, para priorizar por valor, gerenciar risco, realizar progresso com informações imperfeitas, construir uma cultura de alta confiança, e responder rapidamente e graciosamente a mudanças que chegam durante o projeto.

Com um projeto grande, como numa iniciativa de manutenção, você precisará acordar uma cadência de priorização para preenchimento da fila de entrada. A regra geral é uma cadência maior com reuniões mais frequentes. Olhe para os princípios novamente. Quais são os custos de transação e coordenação de sentar-se com a equipe de marketing ou donos do negócio e entrar em acordo nos próximos itens da fila para desenvolvimento? Do outro lado da cadeia de valor, você terá muitas integrações ou pontos de sincronização convergindo para um *release* do que apenas um único ponto de *release*; novamente uma maior frequencia é melhor. Faça a pergunta, “Quem está envolvido em reuniões com pessoas do negócio para demonstrar trabalho recente e então integrá-lo a fim de que ele se torne um ‘*release ready*’?”

Depois, você deverá entrar em acordo em relação aos limites de *WIP*; os princípios para se fazer isso não mudam. Classes de serviço continuarão a fazer sentido e a ajudá-lo a lidar graciosamente com mudanças durante o projeto.

Requisitos Hierárquicos

Você precisará definir também tipos de item de trabalho para o seu projeto. Muitos projetos grandes exibem requisitos hierárquicos. Não é incomum que muitos deles tenham até três níveis de hierarquia. Podem existir também diferentes tipos de requisitos, como requisitos do cliente que vêm dos donos do negócio e requisitos do produto, provenientes da equipe técnica, de qualidade ou de arquitetura. Os requisitos devem ser quebrados em requisitos funcionais e não funcionais ou requisitos de qualidade do serviço. Mesmo com desenvolvimento de software Ágil, o cliente pode especificar requisitos em termos de estórias de tamanho grande que são quebradas em *user stories* e são talvez quebradas para o nível menor de tarefas ou pequenas unidades referenciadas

* N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: release pronto.

como “grãos de areia”. Também vi grandes estórias quebradas em estórias arquiteturais que por sua vez são quebradas em *user stories*. *Feature-driven development* também possui três camadas de requisitos – *features*, conjunto de *features* (ou atividades), e áreas de assunto.

Para equipes adaptando Kanban para projetos maiores fez sentido atribuir tipos de item de trabalho diferentes para diferentes níveis da hierarquia. Por exemplo, as estórias grandes são um tipo de item de trabalho, e *user stories* menores são outro tipo. Em um projeto mais tradicional, requisitos do cliente são de um tipo, enquanto requisitos do produto são de outro, e itens de especificação funcional pequenos são o terceiro item.

Tipicamente, as equipes têm escolhido acompanhar os dois níveis superiores de um quadro Kanban. Eu pessoalmente não vi uma equipe ou projeto no qual se tentou acompanhar três níveis com Kanban. Algumas ferramentas eletrônicas dão suporte a requisitos hierárquicos que permitem ao usuário exibir ou omitir níveis diferentes, apresentando apenas dois níveis de cada vez.

Usualmente há um terceiro nível mais baixo, como tarefas, em um projeto Ágil; as tarefas não são acompanhadas no mural de cartões do projeto ou dentro de um sistema kanban no nível da equipe. Desenvolvedores individuais podem escolher acompanhar tarefas, ou talvez equipes pequenas e *cross-functional* possam escolher acompanhar tarefas localmente, mas fora do quadro do projeto e fora da visão dos gerentes e parceiros da cadeia de valor; isto não é motivado pela necessidade de esconder informação. Simplesmente o nível mais baixo de atividade não é interessante da perspectiva da cadeia de valor e nível de desempenho. O nível mais baixo é frequentemente focado no esforço e atividades em vez do valor para o cliente e funcionalidade.

Enquanto escrevia esse livro, uma variação do Kanban para uso pessoal estava surgindo, liderado por Jim Benson e outros. Kanban Pessoal é usado em casa e no escritório a nível individual, ou com pequenos grupos de duas ou três pessoas que estão colaborandoativamente no mesmo conjunto de itens de trabalho. É impossível saber no tempo da escrita desse livro se Kanban Pessoal será assimilado para dentro do corpo do conhecimento ou emergirá como uma disciplina de valor próprio.

Dissocie Entrega de Valor de Variabilidade do Item de Trabalho

A ideia que surgiu na maioria das equipes Kanban acompanhando os dois níveis mais altos de requisitos é que o nível mais alto de requisitos, os requisitos menos granulares, geralmente descreviam alguma unidade atômica de valor para o mercado ou cliente. Essas *user stories* grandes ou requisitos do cliente eram frequentemente escritas em um nível no qual fazia sentido um *release* para o mercado. O produto já estava em

manutenção e essas requisições deviam ser processadas e released individualmente. Algumas vezes a comunidade Kanban refere-se a este nível de requisitos como “*minimal marketable feature*” (ou MMF). Há confusão em relação a esse conceito, visto que MMF foi definido por Denne e Cleland-Liang no livro *Software by Numbers*, e sua definição não é totalmente aderente a essa. Eu prefiro uma definição de *minimum marketable release* (MMR) que descreve um conjunto de *features* que são coerentes como um conjunto para o cliente e úteis o bastante para justificar os custos de entrega.

Não faz sentido tratar MMR como um item único que flui através do nosso sistema kanban. Um MMR é feito de muitos itens de trabalho. MMR faz sentido a partir de uma perspectiva do custo de transação do *release*, e não a partir da perspectiva do fluxo. Em alguns casos pode fazer sentido o *release* de uma pequena *feature* diferenciada de valor alto. Por outro lado, como muitos descobriram, “o primeiro MMF é sempre grande” porque o primeiro *release* de um novo sistema deve incluir todas as capacidades essenciais para entrar no mercado e toda a infra-estrutura para suportá-lo. Podem existir dois ou três pedidos de diferentes magnitudes em relação ao tamanho dos MMFs (ou MMRs). Um tipo de item de trabalho com instâncias que variam no tamanho acima de mil vezes será problemático.

Sistemas kanban não prezam por uma variação tão grande no tamanho. Isso requer *buffers* grandes e que o limite do *WIP* seja excedido para amenizar o fluxo; sem os *buffers* os sistemas sofreriam grandes variações no *lead time*. *Buffers* grandes e mais *WIP* significa *lead times* maiores e perda da agilidade no negócio. A alternativa é pior! Se não usarmos *buffer* para a variabilidade no tamanho, haverá grandes flutuações no *lead time*. Como resultado, é impossível oferecer uma meta de *lead time* abaixo do acordo de nível de serviço que possa ser alcançada com consistência. O resultado será uma previsibilidade ruim e perda de confiança no sistema. Projetar um sistema kanban em torno do conceito de MMF é como levar a uma perda na agilidade do negócio, e/ou uma perda da previsibilidade, uma perda da confiança entre TI e o negócio, e uma insatisfação geral com a abordagem Kanban.

No entanto, usar *Minimal Marketable Release* (MMR) como um gatilho para uma entrega, em conjunto com tipos de item de trabalho menores e mais granulares, é minimizar custos e maximizar a satisfação com o *release*.

Equipes podem se adaptar a esse desafio focando em técnicas de análise que produzem um nível mais baixo de requisitos, como *user stories* ou uma especificação funcional. Isso vai gerar uma variação de tamanho menor e mais granular. Um tamanho ideal seria algo num intervalo de meio dia a quatro dias de esforço de engenharia.

Num projeto maior estabelecemos que cada item de trabalho grande, chamado “Requisito”, e acompanhado com *tickets* verdes, se quebraria numa média de 21 “*Features*” menores, acompanhadas com *tickets* amarelos. Embora as *features* fossem escritas de uma maneira orientada ao usuário – e valor, elas eram analisadas para serem

menores e similares no tamanho. Se ele fosse um projeto Ágil, esses níveis teriam sido *Epic*, acompanhados em verde, e *User Story*, acompanhados em amarelo.

Os itens menores, mais granulares, possibilitam fluxo e previsibilidade do rendimento e *lead time*, enquanto que itens menos granulares numa camada mais alta do quadro nos permite controlar o número de requisitos *releasable*, *marketable* em progresso a qualquer tempo.

Adotando uma abordagem de duas camadas, nós dissociamos a entrega de valor da variabilidade no tamanho e esforço necessários para entregar esse valor.

Faz sentido atribuir limites de *WIP* nos dois níveis. Com muitos projetos verificamos que fazia sentido atribuir uma equipe pequena e cross-funcional para cada requisito de alto nível. Estas equipes puxariam itens menores e mais granulares para o seu requisito de alto nível fluindo-os através do quadro sem intervenções até que o requisito fosse finalizado e estivesse pronto para integração ou entrega; então a equipe puxaria outro requisito menos granular. Haveria também oportunidade de redistribuir membros da equipe, adicionando ou liberando membros da equipe dependendo do tamanho do próximo item puxado.

Paredes de Cartão em Duas Camadas

As primeiras equipes usando Kanban em projetos grandes adotaram um estilo de duas camadas para a parede de cartão, como apresentado na Figura 13.1.

Nessa foto, os requisitos menos granulares são apresentados com *tickets* verdes. Eles fluem da esquerda para a direita através de um conjunto de estados, nomeados, *backlog*, proposto (análise), ativo (projeto e desenvolvimento), resolvido (testando) e fechado.

Os requisitos ativos são apresentados no topo na parte central da foto. Eles, por sua vez, são quebrados em várias *features* menores, apresentadas com *tickets* amarelos. As *features* fluem através do próprio conjunto de estados, nomeados, proposto (análise), ativo (projeto e código), resolvido (testando) e pronto. Os estados que as *features* fluem são similares aos estados dos requisitos de alto nível, mas eles não precisam ser similares; você pode modelar isso da maneira que achar mais adequada; meu conselho é modelar o que você faz agora; evite mudar o processo.

Os *tickets* amarelos são rastreados de volta aos seus pais identificando-os com os números IDs dos seus pais.

Num exemplo como este, é possível limitar o *WIP* nos dois níveis hierárquicos, mas os *tickets* amarelos são todos agrupados em um *pool*. Não tenho evidência de campo suficiente para saber se essa é ou não uma boa estratégia. O que eu sei é que ela não de adequou a essa equipe.



Figura 13.1 Fotografia de um quadro de duas camadas

Introduzindo Raias

Fazer a correspondência dos *tickets* amarelos mais granulares com os seus pais menos granulares é importante. Parece fazer sentido limitar o *WIP* no nível mais baixo dentro de uma equipe *cross-functional*. Para facilitar essa abordagem, algumas equipes inovaram o sistema de parede de cartão e introduziram raias horizontais.

Na figura 13.2, o requisitos de alto nível, apresentados com *tickets* verdes, fluem através do mesmo conjunto de estados – nomeados, *backlog*, proposto, ativo, resolvido e fechado. No entanto, a parte central foi redesenhada comparando a Figura 13.1. Os requisitos menos granulares em verde estão verticalmente empilhados à esquerda. Para cada um destes *tickets* verdes prolonga-se uma raias dividida no mesmo conjunto de estados para as *features* amarelas mais granulares. O número de raias é agora o limite de *WIP* para os requisitos *customer-marketable* menos granulares, enquanto que os limites para as *features* mais granulares podem agora ser atribuídos para cada raias se as equipes individuais escolherem fazer assim. A coluna imediatamente à direita da pilha vertical de requisitos verdes contém os nomes dos membros permanentes da equipe. Os *tickets* laranja menores anexados aos *tickets* amarelos, efetivamente contêm os nomes dos recursos especialistas flutuantes, como *user-experience designers* e arquitetos de banco de dados.

A raias variável da parede de cartão significa que agora estamos gerenciando o *WIP* de *customer-marketable* verticalmente, enquanto estamos gerenciando o *WIP* das

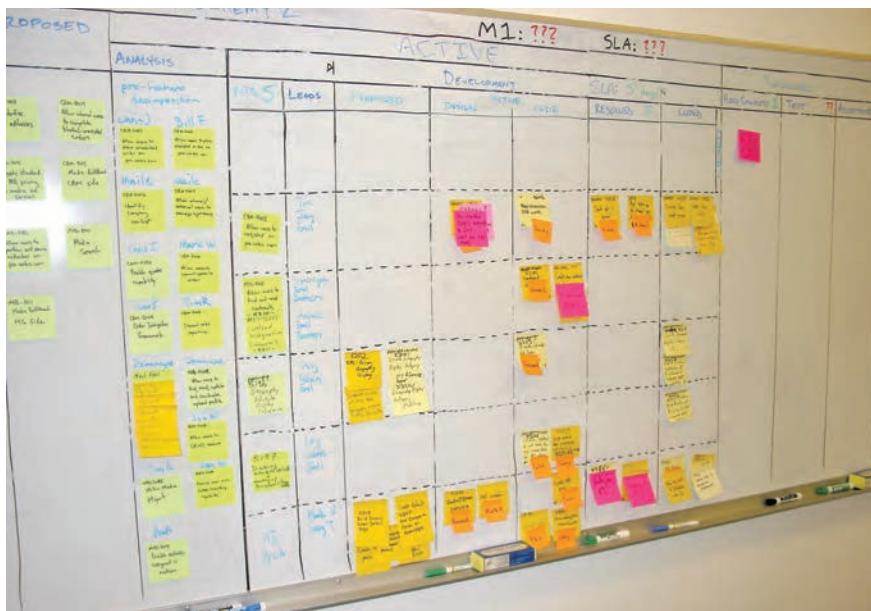


Figura 13.2 Fotografia de um quadro de duas camadas com raias

features de baixa variabilidade horizontalmente. Este formato provou ser muito popular e tornou-se típico.

Abordagem Alternativa para Variabilidade no Tamanho

Outra abordagem para lidar com variabilidade no tamanho é criar tipos de trabalho diferentes para tamanhos diferentes de itens. Raias podem então ser estabelecidas para itens de cada tamanho/tipo. Limites de *WIP* devem ser estabelecidos para cada coluna em cada raia, i.e., cada caixa na parede de cartão. Devido à baixa variabilidade através de cada raia (como os itens são similares no tamanho), cada raia deve fluir relativamente suave. Esta é uma maneira de lidar com variabilidade sem recorrer a um sistema de duas camadas.

Incorporando Classes de Serviço

Os dois métodos mais óbvios para visualmente diferenciar cartões na parede são usar cores e raias. No entanto, em projetos grandes, cada ticket tipicamente tem três atributos que precisamos comunicar: tipo do item de trabalho, nível na hierarquia, e classe de serviço. É importante notar que, no exemplo (Figura 13.2), a escolha de se ter diferentes

tipos de item de trabalho na hierarquia e então usar tanto cores como raias para designar o nível na hierarquia significa que a hierarquia está sendo sobre carregada com dois métodos de visualização.

Se você precisa comunicar classe de serviço além do tipo e nível de hierarquia do requisito, pode fazer sentido usar cores para as classes de serviço. Se os tipos são usados para propósitos diferentes do nível de hierarquia, por exemplo, para mostrar erros ou defeitos, ou valor agregado versus carga de falha, você pode escolher outra abordagem, talvez introduzindo um ícone ou um adesivo anexado ao cartão para identificar o tipo; ou você pode preferir usar cores para os tipos e um ícone ou adesivo para classes de serviço, e.g., uma estrela prata para uma requisição de expedição.

O que pode ser mais fácil, como vimos evoluir na Corbis, é o uso de cores para diversos propósitos, como nível na hierarquia, tipo e classe de serviço. Esta abordagem, na qual as cores não designam claramente qualquer atributo, é aceitável para os usuários de sistema Kanban e muito eficiente em termos de opções disponíveis para visualização.

Integração de Sistemas

Em alguns projetos grandes, você pode ter equipes múltiplas trabalhando em componentes diferentes de um sistema que precisa ser integrado posteriormente. Alguns destes componentes podem envolver hardware ou *firmware*, e pode não ser passível a técnicas modernas de integração continua. Quando você possui esses tipos de componentes que precisam ser integrados, você precisa determinar um ponto de integração baseado numa atividade de planejamento de alto nível. Este ponto pode então ser tratado como uma data fixa para entrega destes componentes dependentes. Isto permite que cada equipe vá em frente independentemente com o seu sistema kanban, mas também permite coordenar a entrega de itens dependentes quando eles são necessários. Entrega em atraso de um item dependente causa grande atraso para o projeto como um todo. Este alto custo de atraso justifica tratá-lo como um item de classe de serviço de data fixa.

Gerenciando Recursos Compartilhados

É comum em grandes projetos e através de *portfolios* de projetos o compartilhamento de recursos especialistas; por exemplo, arquitetos de software, arquitetos e administradores de banco de dados, *user-experience testing*, *user-experience design*, e auditor de segurança de software. Há três métodos estabelecidos para lidar com estes recursos compartilhados em Kanban.

No primeiro método alguns dos itens de trabalho têm *tickets* laranja menores anexados a eles. Estes *tickets* menores mostram o nome do recurso compartilhado requerido, como Sandy, a arquiteta de dados da empresa. No nível menor de intrusão, esta ação simples de visualizar o trabalho de recursos compartilhados é frequentemente suficiente para coordenar a carga de trabalho individual. Se muitos *tickets* começarem a aparecer com o mesmo nome, isso pode trazer à tona questões sobre como esta pessoa está gerenciando o trabalho em múltiplas coisas ao mesmo tempo. Isto deve ser suficiente para facilitar uma discussão na mudança da política – todo o trabalho precisa ser visto por aquela pessoa? – ou escalá-la para um próximo nível.

O próximo nível é reconhecer que recursos compartilhados não estão instantaneamente disponíveis, e visualizar isto marcando os itens que precisam de atenção de um recurso compartilhado (um ticket laranja), como bloqueado até uma pessoa estar efetivamente trabalhando nele. Isto tem o efeito de invocar o responsável do gerenciamento e resolução de problemas para resolver a disponibilidade do recurso compartilhado. Isto também tem o efeito de mostrar à gerência se a disponibilidade do recurso é um problema e um gargalo potencial.

O nível mais alto de gerenciamento de um recurso compartilhado é dar ao recurso seu próprio sistema kanban. Por exemplo, a arquiteta de dados da empresa, o *user experience*, o auditor de segurança de software, e assim por diante, podem ter seu próprio sistema kanban. Cada equipe ou recurso analisaria independentemente sua demanda e atribuiria tipos de item de trabalho baseados na fonte da requisição, e classe de serviço baseados na prioridade e resposta requisitada. A demanda seria analisada e políticas para alocação da capacidade atribuídas.

Neste nível, o que emergiu é uma arquitetura orientada a serviços para desenvolvimento do software. Cada grupo dentro da empresa oferece seu próprio conjunto de serviços que são exibidos como níveis de acordo de serviço para diferentes classes de serviço e tipos de item de trabalho. Clientes destes recursos compartilhados submetem requisições de trabalho para o seu *backlog*; estas requisições são enfileiradas e selecionadas para processamento, como descrito neste livro. Se requisições de um determinado cliente não são processadas de maneira suficientemente rápida, isto pode abrir uma discussão sobre se o sistema está projetado corretamente ou não, e se as políticas em relação à capacidade de alocação e classes de serviço precisam ou não ser alteradas. Isto pode inclusive fornecer evidências para realinhamento ou aumento de pessoal.

Takeaways

- ❖ Projetos grandes devem seguir os princípios centrais de Kanban.
- ❖ Limites de *WIP*, cadência de priorização, cadência de entrega, e classes de serviço são técnicas válidas para projetos maiores.
- ❖ Projetos maiores tendem a ter requisitos hierárquicos; estes níveis de hierarquia devem ser modelados com tipos de item de trabalho.
- ❖ Tipicamente, as equipes acompanham os dois itens mais altos de requisitos da hierarquia na parede de cartão e limitam o *WIP* para um ou os dois níveis.
- ❖ O nível mais alto de requisitos tipicamente modela requisitos *customer-marketable* que se tornam unidades atômicas que podem ser released individualmente.
- ❖ O segundo nível de requisitos é tipicamente escrito numa linguagem orientada ao cliente/usuário e é analisado de maneira que os requisitos sejam mais granulares e tenham tamanhos similares.
- ❖ O segundo nível de requisitos mais granulares facilita o fluxo reduzindo variabilidade no sistema puxado kanban.
- ❖ Será necessária uma parede de cartão em duas camadas pra visualização dos dois níveis de requisitos a serem acompanhados.
- ❖ A utilização de raias tornou-se uma técnica popular para mostrar hierarquia e facilitar a limitação do *WIP*.
- ❖ O limite de *WIP* dos requisitos menos granulares é limitado pelo número de raias.
- ❖ O limite de *WIP* dos requisitos mais granulares pode ser limitado para cada raia, se desejado.
- ❖ Tipicamente, pequenas equipes *cross-functional* são atribuídas para cada raia.
- ❖ Demanda de recurso compartilhado pode ser visualizada com adesivos menores anexados aos itens de trabalho.
- ❖ A disponibilidade não instantânea de recursos compartilhados pode ser destacada com *tickets* de bloqueio (rosa, por exemplo) anexados ao ticket original do item de trabalho.
- ❖ Recursos compartilhados devem desenvolver seus sistemas kanban próprios.
- ❖ Uma rede de sistemas kanban para recursos compartilhados através de um *portfolio* de projetos pode ser vista como uma arquitetura de desenvolvimento de software orientada a serviços.

❖ CAPÍTULO 14 ❖

Revisão Operacional

Antes da Reunião

São 7:30 a.m. da segunda sexta-feira de Março de 2007. Cheguei cedo no trabalho porque nesta manhã acontecerá nossa quarta revisão operacional mensal do departamento. Estou com Rick Garber, o gerente do nosso grupo de engenharia de processo de software. Rick tem a responsabilidade de coordenar a agenda e a reunião de revisão operacional. Ele está ocupado imprimindo a apresentação que contém aproximadamente 70 *slides* PowerPoint para a apresentação de hoje. Depois que a impressão estava pronta seguimos para o Harbor Club no centro da cidade de Seattle com uma caixa com 100 apresentações. A revisão operacional estava agendada para começar às 8:30 a.m., mas o *buffet* do café da manhã foi servido às 8:00. A reunião foi planejada para 80 pessoas. O convite incluía todos da minha organização e todos os meus colegas da organização de Erik Arnold. No entanto, com algumas pessoas na Índia, ou em outras partes do EUA, e sempre algumas que não podem comparecer por motivos pessoais, nós alcançamos aproximadamente 80 pessoas.

O convite também incluía meu chefe, o CIO da Corbis, e outros gerentes seniores, e nossos parceiros na cadeia de valor. O grupo externo que compareceu em maior número foi a equipe Operações de Rede e Sistemas liderada pelo meu colega Peter Tutak. Eles eram responsáveis por recuperar falhas de sistemas em produção, então eles

eram os mais impactados pelas nossas falhas. Eles também sentiam o maior impacto quando fazíamos novos *releases* para produção. Então, evidentemente, eles tinham muito a ganhar participando ativamente.

O grupo começou a chegar a tempo para o café da manhã. A sala era no último andar de uma torre de Seattle e nos proporcionava uma vista linda da cidade, o porto, os piers, a Baía Elliott. A sala estava cheia de mesas redondas, com seis a oito pessoas em cada uma delas. Nós tínhamos uma tela de projeção e um púlpito em uma das extremidades. Rick gerenciou a agenda com precisão. Cada apresentador tinha em torno de oito minutos para os seus quatro ou cinco *slides*. Havia um *buffer* de tempo para permitir a variabilidade devido aos questionamentos e discussões. Eu comecei minha apresentação prontamente com algumas declarações. Eu pedi a todos que voltassem ao final do mês de Janeiro e ao que eles estavam fazendo naquele período. Lembrei a todos que estávamos ali para revisar o desempenho da organização para o mês de Fevereiro. Rick pegou uma bela imagem da empresa nos arquivos para simbolizar um tema para o mês e para ajudar a sacudir a memória, lembrando a todos uma atividade chave do mês.

Defina um Tom de Negócio desde o Início

Passei a vez para Rick, que resumiu os itens de ação gerenciais do último mês e forneceu uma atualização do *status*. Depois introduzimos nossa analista financeira, que apresentou um resumo do desempenho da companhia no mês – o motivo para o adiamento até a segunda sexta-feira do mês subsequente era para que tivéssemos dados financeiros após o fechamento dos livros do mês anterior. Ela resumiu os detalhes do orçamento para o meu centro de custo e para o de Erick. Verificamos o planejado versus o realizado para as áreas com maior orçamento, como também as metas de *headcount*^{*}. Discutimos posições em aberto e incentivamos os membros das equipes a submeterem candidatos para essas posições. Saindo dessa primeira parte, todos os participantes sabiam o quanto bem a companhia estava e o quanto bem o grupo de engenharia de software estava gerenciando o orçamento e, portanto, quanto espaço disponível nós tínhamos para comprar itens como monitores de tela plana e novos computadores. O propósito de lidar com números financeiros é lembrar a todos da equipe que estamos administrando um negócio; não estamos apenas mostrando zeros e uns como uma brincadeira com um grupo de amigos.

* N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: número de funcionários.

Convide os Participantes a Sair da Plateia e Agregar Valor

O próximo orador é um convidado – um vice-presidente de outra parte da companhia. Eu tive a brilhante ideia de que se nós queríamos que nossos parceiros da cadeia de valor tivessem interesse em nós, devíamos mostrar interesse neles e convidá-los a apresentar. Oferecemos 15 minutos ao nosso convidado, que os usou. Então ouvimos uma apresentação da operação de vendas, a parte do negócio que preenchia os pedidos do cliente e garantia a entrega do produto. Embora alguns negócios da Corbis fossem realizados pela web e preenchidos eletronicamente, nem tudo que a firma oferecia era entregue como um *download*; um departamento inteiro preenchia pedidos mais complexos para profissionais de agências de publicidade e firmas de mídia. Meu colega, Erick Arnold, teve a brilhante ideia de solicitar ao convidado o patrocínio do café da manhã para manter nossos custos sob controle; isso funcionou também. Ao longo dos meses seguintes nossa equipe aprendeu muitos aspectos do negócio e os líderes seniores da companhia aprenderam o que fazíamos e como fazíamos, e como era difícil lidar com os nossos problemas. Nove meses depois, executivos falavam abertamente sobre como a equipe de TI era bem gerenciada e como a sua unidade de negócio deveria seguir o nosso exemplo.

Agenda Principal

Quando nosso convidado finalizou sua apresentação, nós partimos para a parte principal da nossa reunião. Cada gerente tinha oito minutos para uma apresentação do desempenho do seu departamento. Fizemos isso com algumas atualizações de projetos específicos do nosso escritório de gerenciamento de programa. Cada um dos gerentes imediatos levantou-se e gastou cinco minutos apresentando rapidamente suas métricas. Geralmente eles seguiram o formato apresentado no capítulo 12: eles apresentavam informação sobre taxa de defeito, *lead time*, rendimento, eficiência do valor agregado, e, ocasionalmente, um relatório específico que entrava em algum aspecto do nosso processo para o qual eles precisavam de mais informação. Então eles fizeram perguntas, comentários, e sugestões por alguns minutos.

Esta quarta reunião mensal de Revisão Operacional, em Março de 2007 foi particularmente interessante. A primeira Revisão Operacional foi em Dezembro; todos vieram, uma participação de quase 100 por cento. Muitas curiosidades e depois muitos comentários como, “Eu nunca vi transparência como essa na minha carreira,” e “Isto foi interessante.” O *feedback* mais útil foi, “Na próxima vez podemos ter um *buffet* quente em vez de frio?” Então incluímos um café da manhã quente. No segundo mês as pessoas disseram, “Sim, outro mês bom. Algo interessante! Obrigada pelo café da manhã

quente!” No terceiro mês alguns dos desenvolvedores perguntaram, “Porque eu preciso acordar tão cedo?” e “Isso é um bom uso do meu tempo?”

No quarto mês aconteceu de revermos um problema significante. A companhia tinha adquirido um negócio na Austrália. Foi solicitado ao departamento de TI desligar todos os sistemas de TI da subsidiária australiana e migrar todos os 50 usuários para os sistemas da Corbis. A solicitação tinha uma data arbitrária, mas era urgente. Esta data baseou-se numa “economia de escala” – estilo de redução de custos que tinha justificado parte do preço da aquisição, então havia um custo de atraso envolvido. A solicitação chegou com um único item na nossa fila de manutenção. Ela era grande o bastante para justificar 10 *tickets*, mas nós a tratamos como sendo apenas um. O efeito de um item mal dimensionado como esse entrando num sistema kanban é bem conhecido na engenharia industrial. Ele obstrui o sistema e estende o *lead time* para tudo o que vem atrás dele; e isso aconteceu conosco. O *lead time* pulou, de uma média de 30 para 55 dias. A teoria das filas também nos diz que reduzir o *backlog* quando ele está completamente preenchido leva um tempo grande. Nós descobrimos que gastaríamos cinco meses para recuperar a meta de *lead time*.

Além disso, nós tínhamos um *release* que requisitava uma correção de emergência.

De repente, a sala estava iluminada com perguntas, comentários, e debate. Depois de três meses de tédio e bons dados, nós tínhamos uma história para contar. As pessoas estavam surpresas que nós (a gerência) estávamos dispostos a falar abertamente sobre o problema e o que fazer sobre ele, e que a Revisão Operacional não era apenas apresentar o quanto éramos bons, apresentar apenas os dados bons. Ninguém questionou novamente porque fazíamos a reunião mensalmente.

A reunião terminou com Rick resumindo os itens de ação gerencial das discussões ocorridas pela manhã e agradecendo a todos por terem comparecido. Era 10:30 e hora de voltar ao escritório.

Pedra Angular para uma Transição *Lean*

Há muitas coisas importantes a serem compreendidas sobre Revisão Operacional. Primeiramente, acredito que Revisão Operacional é a pedra angular para uma transição *Lean* e a implementação do método Kanban. Ela é uma retrospectiva objetiva, orientada aos dados do desempenho da organização. Ela está acima e além de qualquer projeto e ela define uma expectativa de gerenciamento quantitativa objetiva e orientada a dados mais do que um gerenciamento qualitativo subjetivo que é mais utilizado em um projeto Ágil com retrospectiva de iteração. Revisão operacional fornece um *feedback* contínuo que permite crescimento da maturidade organizacional e melhoria

contínua do nível da organização. Acredito verdadeiramente que ela é essencial para a transição com sucesso para uma escala de negócio *Lean* (ou Ágil).

CadênciA Apropriada

Também acredito que a Revisão Operacional deve ser mensal. Uma frequência maior pode ser onerosa para coleta de dados, e o tempo envolvido para a reunião significa que há um desejo de não fazê-la frequentemente. Fazer com que a reunião dure duas horas é um desafio. Se ela não fosse uma reunião orientada a dados, repleta de gráficos e relatórios, isso não seria possível. Uma reunião subjetiva de escala não pode ser finalizada em duas horas. Uma retrospectiva de projeto típica leva mais de duas horas, então imagine tentar realizar uma retrospectiva de toda uma organização e completá-la em duas horas usando um estilo de análise delta. Parte do segredo de manter a duração curta da reunião é realizá-la como base em dados objetivos; manter a agenda apertada e gerenciá-la durante a reunião.

Pode haver uma tendência de realizar revisões operacionais menos frequentemente; trimestralmente é mais comum. Minha experiência com revisões operacionais trimestrais são do meu tempo na divisão PCS da Motorola. Minha observação em relação a essas reuniões é que elas eram sessões de reportagem e revisão da gerência superior, e não sessões organizacionais projetadas para orientar melhoria contínua e maturidade organizacional. Uma frequência trimestral é pouco para verdadeiramente conduzir um programa de melhoria. Os dados têm frequentemente quatro meses no momento da revisão trimestral. Um trimestre é um grande espaço de tempo para ser revisado numa única reunião, dessa maneira a revisão tende a ser superficial. Relatórios e métricas tendem a serem indicadores de resultado e focados na reportagem de desempenho em relação à meta dos líderes seniores.

Reuniões trimestrais tornam-se atraentes porque elas parecem ser mais eficientes – apenas uma reunião de duas horas a cada três meses do que mensalmente. Elas também custam menos numa base anual – apenas quatro reuniões em vez de 12. Depois que eu deixei a Corbis no início de 2008, meu antigo chefe reduziu a cadênciA da revisão operacional para trimestral para reduzir custos. Depois de três trimestres, e com a saída desse chefe, a nova liderança questionou o valor das reuniões e decidiu cancelá-las. Dentro de poucos meses o desempenho da organização desvalorizou consideravelmente, e o nível de maturidade da organização retraiu de uma equivalência ao Nível 4 do Modelo CMMI para o Nível 2 do Modelo CMMI, de Gerenciado Quantitativamente para meramente Gerenciado.

Podemos tirar várias coisas disso. A perda de um *feedback* contínuo reduziu as oportunidades para reflexão e adaptação que poderiam levar a melhorias. Eliminar uma reunião focada numa revisão objetiva de desempenho da organização dá a entender que a liderança não se preocupa com o desempenho. O resultado foi um retrocesso significante na maturidade da organização e no desempenho em termos de previsibilidade, qualidade, *lead time*, e rendimento.

Demonstrando o Valor dos Gerentes

A revisão operacional também mostra às pessoas o que os gerentes fazem e como o gerenciamento pode agregar valor às suas vidas. Ela também ajuda a treinar a força de trabalho a pensar como gerentes, e a entender em que momento realizar intervenções e quando recuar e deixar a equipe se auto-organizar e resolver seus próprios problemas. Revisões operacionais ajudam a desenvolver o respeito entre o conhecimento individual dos trabalhadores e seus gerentes e entre os diferentes níveis de gerenciamento. Desenvolver o respeito constrói confiança, estimula a colaboração, e desenvolve o capital social da organização.

Foco Organizacional Fomenta *Kaizen*

Enquanto retrospectivas de projetos individuais são sempre úteis, uma revisão operacional de toda a organização fomenta a institucionalização de mudanças, melhorias, e processos. Ela estimula a propagação viral de melhorias através de uma organização criando uma pequena rivalidade entre projetos e equipes que motiva a todos melhorar o seu desempenho. Equipes querem demonstrar como elas podem ajudar a organização com uma melhor previsibilidade, mais rendimento, *lead times* menores, custos menores, e maior qualidade.

Um Exemplo Anterior

Eu não inventei revisões operacionais. Elas são muito comuns em muitas empresas grandes. No entanto, eu aprendi como conduzi-las de uma maneira objetiva para toda a unidade de negócio quando eu trabalhava na Sprint PCS em 2001. Meu chefe, o Vice Presidente e Gerente Geral da sprintpcs.com, as instituiu por vários motivos. Ele queria desenvolver a maturidade da sua organização – uma unidade de negócio de 350 pessoas responsável pelo *web site*, por todo o *e-commerce*, e o atendimento online aos clientes do negócio de telefonia celular da Sprint. Nós fazíamos revisão operacional

na sprintpcs.com toda terceira sexta-feira do mês às 2 p.m. Ela durava duas horas e agrupava aproximadamente 70 pessoas seniores e gerentes da unidade de negócios, mais o diretor – ou convidados do nível da gerência sênior dos nossos parceiros da cadeia de valor. Líderes seniores, incluindo o Chefe do Escritório de Marketing e o VP do Planejamento Estratégico também eram participantes regulares. O formato era muito similar ao da Corbis. Ela era inteiramente orientada a dados objetivos. Cada gerente apresentava seus próprios dados. A reunião começava primeiramente com os dados financeiros. A agenda era planejada e gerenciada firmemente. Depois da reunião, todos iam para casa mais cedo na sexta-feira. A reunião acontecia fora da empresa, num campus local de uma faculdade. Enquanto a sprintpcs.com lutou com técnicas de desenvolvimento de software Ágil, a revisão operacional era um elemento chave para o desenvolvimento da maturidade organizacional e melhoria da governança da organização. Ela mostrou às pessoas que os gerentes estavam fazendo a diferença e sabiam como gerenciar, e ela deu às pessoas e aos gerentes da linha de frente uma chance de mostrar aos líderes seniores como elas podiam ajudar e onde eles precisavam de intervenções para verdadeiramente fazer a diferença.

Dado duas experiências dentro de um período de quatro anos através da última década, tornei-me convicto de que revisão operacional é uma parte crítica para uma transição com sucesso para *Lean* ou Agilidade e um componente vital para o desenvolvimento da maturidade organizacional.

Takeaways

- ❖ Revisões operacionais devem ser para toda a organização.
- ❖ Revisões operacionais devem ter o foco em dados objetivos.
- ❖ Cada departamento deve reportar seus dados.
- ❖ As apresentações devem ser mantidas curtas e devem tipicamente relatar métricas e indicadores similares àqueles discutidos no capítulo 12.
- ❖ Lidar com informação financeira ratifica que a engenharia de software é parte de um negócio maior, e que uma boa governança é importante.
- ❖ Uma cadência mensal para a revisão operacional é adequada. Uma frequência maior é onerosa em relação ao tempo gasto e para a coleta e preparação dos dados. Uma frequencia menor tende a reduzir o valor e minar a natureza da reunião.
- ❖ As reuniões devem ser mantidas curtas; tipicamente duas horas.
- ❖ Revisões operacionais devem ser utilizadas para fornecer um *feedback* contínuo e orientar melhoria continua na empresa ou no nível da unidade de negócio.
- ❖ Revisões operacionais mostram aos indivíduos como o gerenciamento pode agregar valor às suas vidas e o que efetivamente os gerentes fazem.
- ❖ Revisões operacionais efetivas constroem confiança mútua entre os gerentes e os trabalhadores.
- ❖ *Stakeholders* externos que participam das revisões operacionais têm a oportunidade de ver como a engenharia de software e o grupo de TI funcionam e entender seus problemas e desafios; isto fomenta confiança e colaboração.
- ❖ Revisões operacionais devem analisar dados ruins e problemas da mesma maneira que devem gozar o sucesso e exaltar as virtudes das equipes com bons resultados.
- ❖ Realizar as reuniões fora do escritório ajuda a manter o foco dos participantes.
- ❖ Fornecer alimentação estimula o comparecimento.
- ❖ O envolvimento de líderes seniores comunica que a organização considera seriamente o desempenho e melhoria contínua da organização

- ❖ Atribuir um interesse sério ao desempenho, melhoria continua, e gerenciamento quantitativo é vital para o desenvolvimento de uma cultura *kaizen* entre todos os trabalhadores.
- ❖ Revisões operacionais têm mostrado que lidam diretamente com os níveis de crescimento da maturidade organizacional.
- ❖ Sugestões de melhoria devem ser capturadas como itens de ação gerenciais e devem ter seu progresso revisado no início das reuniões subsequentes.
- ❖ Gerentes devem ter responsabilidade e devem demonstrar que estão fazendo o acompanhamento das sugestões.

❖ CAPÍTULO 15 ❖

Empreendendo uma Iniciativa de Mudança Kanban

Dar início ao uso de Kanban não é uma iniciativa comum de processo que você pode ter empreendido no passado. É importante estabelecer uma base para um sucesso de longo prazo. Para fazer isso, é necessário entender os objetivos por detrás do uso da abordagem Kanban para mudança. Intitulei esse livro, “Mudança Evolucionária de Sucesso para o Seu Negócio de Tecnologia.” Fiz isso para destacar o ponto que o principal motivo para adotar Kanban é gerenciamento de mudança; tudo mais é secundário.

Mudança Cultural em vez de uma Iniciativa de Mudança Gerenciada

O capítulo 5 descreve como Kanban aperfeiçoa o processo existente através de uma série de mudanças incrementais e evolucionárias. Esse processo de aperfeiçoar o que já existe leva a uma melhoria na maturidade organizacional e eventualmente possibilita que sejam introduzidas mudanças maiores e mais estratégicas. Devido a isto, é improvável que você direcionará uma adoção de Kanban através de uma iniciativa de transição planejada e um programa de treinamento prescrito. Esta é uma mudança significativa em relação a como uma transição Ágil típica é planejada e gerenciada. De fato, a abordagem de gerenciamento de mudança para a introdução de métodos Ágeis é bastante

parecida com as iniciativas de gerenciamento de mudança anteriores, como aquelas baseadas no CMMI ou na introdução de métodos como o *Rational Unified Process*. A iniciativa de mudança tende a ser uma iniciativa com o estilo grande planejamento antecipado. Há um tipo específico de mudança gerenciada na qual o processo atual é primeiramente definido e avaliado, seguindo-se de uma seleção de um método Ágil a partir de um livro. Após essa etapa há um empenho no planejamento de treinamento e formação para se fazer a transição da equipe ou organização do que eles fazem no momento atual para o novo processo Ágil definido. Uma vez que isto é finalizado e o novo processo está em execução, outra avaliação é conduzida para demonstrar a adoção dos novos métodos. Esta não é a abordagem com Kanban. Com Kanban não há uma iniciativa planejada, não há avaliações, e não há uma declaração no final que diga, “Agora nós somos Ágeis!” Idealmente não há final. Ao invés disso, a liderança conduz um processo contínuo encorajando mudanças incrementais. Como resultado, há uma transformação gradual através de uma cultura Kaizen.

É verdade que algum treinamento será necessário. No entanto, mais do que desperdiçar muito tempo em educação, num primeiro momento, é mais importante obter um consenso em relação à introdução de Kanban e começar a usá-lo. Este capítulo procura explicar os fundamentos para uma transição bem sucedida para Kanban e fornece a você um guia com 12 passos simples para dar início à transição.

Embora nosso principal objetivo com Kanban seja introduzir mudança com resistência mínina, devem existir outros objetivos; mudar pelo objetivo de mudança é inútil. Estes outros objetivos devem refletir necessidades genuínas de negócio como uma maior qualidade ou entrega previsível. Os objetivos listados aqui devem ser considerados exemplos. Os objetivos específicos da sua organização podem ser diferentes. O passo 1 no seu processo deve ser acordar os objetivos da sua organização para a introdução de Kanban.

O Primeiro Objetivo para Nosso Sistema Kanban

Estamos fazendo Kanban porque acreditamos que ele proporciona um caminho melhor para introduzir mudança. Portanto mudança com resistência mínina deve ser nosso primeiro objetivo.

OBJETIVO 1 APERFEIÇOAR O PROCESSO ATUAL

Os processos atuais serão aperfeiçoados através da introdução de visualização e limite do trabalho-em-progresso para catalisar mudanças. Como os papéis e responsabilidades não mudam, a resistência dos funcionários deverá ser mínima.

Objetivos Secundários para Nosso Sistema Kanban

Sabemos que Kanban nos permite alcançar todos os seis elementos da Receita para o Sucesso (do capítulo 3). No entanto, podemos reformular levemente os objetivos e expandir alguns dos pontos da receita para refletir aquele ponto que pode nos ajudar a cumprir mais de um objetivo.

OBJETIVO 2 ENTREGAR COM ALTA QUALIDADE

Kanban nos ajuda a focar na qualidade limitando o trabalho-em-progresso e permitindo-nos definir políticas sobre o que é aceitável antes de um item de trabalho ser puxado para o próximo passo no processo. Estas políticas podem incluir critérios de qualidade. Se, por exemplo, atribuímos uma política rigorosa de que *user stories* não podem ser puxadas até que todos os testes tenham sido executados com sucesso e os defeitos resolvidos, estamos efetivamente “parando a linha” até que a *user story* esteja em condição boa o bastante para continuar. Com uma equipe nova em Kanban, não devemos implementar uma regra tão rigorosa, mas devem existir algumas políticas relacionadas à qualidade que direcionam a equipe a desenvolver código com uma quantidade menor de defeitos.

OBJETIVO 3 MELHORAR A PREVISIBILIDADE DO LEAD TIME

Sabemos que a quantidade de *WIP* tem relacionamento direto com o *lead time* e que há também uma correlação entre *lead time* e um crescimento não linear na taxa de defeitos; portanto faz sentido mantermos o *WIP* pequeno; seria mais fácil simplesmente acordar uma quantidade fixa. Isto pode tornar *lead times* seguros e nos ajudar a manter a taxa de defeitos baixa.

OBJETIVO 4 MELHORAR A SATISFAÇÃO DOS FUNCIONÁRIOS

Embora a satisfação dos funcionários seja valorizada na maioria das empresas, raramente é uma prioridade. Investidores e gerentes seniores têm a visão de que recursos são fungíveis e facilmente substituídos. Isto reflete uma tendência de gerenciamento centrada nos custos ou abordagem de investimento. Ela não considera o grande impacto no desempenho com uma força de trabalho motivada e experiente. Retenção de pessoal é importante. Com o envelhecimento da população de desenvolvedores de software, eles preocupam-se mais com o resto das suas vidas. Muitos lamentam desperdiçar vinte anos trancados num escritório forçados a trabalhar num código que falha em atender às expectativas do mercado e torna-se obsoleto logo após o *release*.

Balancear trabalho/vida não é apenas balancear o número de horas que uma pessoa gasta no trabalho e o número de horas que ela está disponível para sua família, amigos, *hobbies*, paixões, e outras atividades; é também fornecer confiança. Digamos, por exemplo, que um membro da equipe com uma paixão por arte quer assistir a uma aula de pintura numa escola média local. Ela começa às

6:30 p.m e acontece toda quarta-feira por dez semanas. Sua equipe pode assegurar para essa pessoa que ele ou ela está livre para deixar o escritório em tempo de comparecer à aula toda semana?

Fornecer um bom balanceamento para trabalho/vida tornará sua empresa um empregador mais atraente no seu mercado local. Isto ajudará a motivar os funcionários e dará aos membros da sua equipe a energia para manter níveis altos de desempenho por meses ou anos. É uma ilusão acreditar que você conseguirá um alto desempenho a partir de trabalhadores com conhecimento quando você os sobrecarrega com trabalho. Isto pode ser verdade taticamente por poucos dias, mas não é sustentável por mais de uma ou duas semanas. É um bom negócio fornecer um bom balanceamento trabalho/vida nunca sobrecarregando suas equipes.

OBJETIVO 5 PROPORCIONAR FOLGA PARA PERMITIR MELHORIA

Embora o terceiro elemento da Receita para o Sucesso – Balancear Demanda e Rendimento – possa ser utilizado para evitar sobrecarregar os membros da equipe e para permitir que eles tenham um balanceamento confiável entre trabalho/vida, ele tem um efeito secundário. Ele cria uma folga na cadeia de valor. Deve existir um gargalo na sua organização. Toda cadeia de valor tem um. O rendimento dos passos posteriores ao gargalo é limitado pelo rendimento do gargalo, não importando o quanto rápidos sejam os passos anteriores ao gargalo. Portanto, quando você balanceia a demanda da entrada e o rendimento, você cria tempo ocioso em todos os pontos da sua cadeia de valor exceto no gargalo.

A maioria dos gerentes repudia a ideia de tempo ocioso. Eles têm sido geralmente treinados para gerenciar a utilização dos recursos (ou eficiência, como é frequentemente chamado), e inerentemente acreditam que mudanças podem ser feitas para reduzir custos se há tempo ocioso. Isto pode ser verdade, mas é importante apreciar o valor da folga.

Folga pode ser usada para melhorar a resposta de requisições urgentes e fornecer banda para possibilitar melhoria no processo. Sem folga, os membros da equipe não têm tempo para aprender novas técnicas, ou melhorar suas ferramentas, ou suas habilidades e capacidades. Sem folga não há liquidez no sistema para responder a requisições urgentes ou mudanças tardias. Sem folga não há agilidade tática no negócio.

OBJETIVO 6 SIMPLIFICAR PRIORIZAÇÃO

Uma vez que a equipe seja capaz de focar na qualidade, limitar o WIP, entregar frequentemente e balancear demanda e rendimento, eles terão uma capacidade de desenvolvimento de software confiável: uma máquina de fazer software! Uma “fábrica de software”. Uma vez que essa capacidade está em vigor, convém ao negócio fazer um uso aperfeiçoado dela. Fazer isso requer um método de priorização que maximiza o valor de negócios e minimiza risco e custo. Idealmente um esquema

de priorização que aperfeiçoa o desempenho do negócio (ou departamento de tecnologia) é mais desejável.

Os campos de engenharia de software e gerenciamento de projeto têm desenvolvido esquemas de priorização desde que os projetos de software começaram há talvez 50 anos atrás; a maioria dos esquemas é simples. Por exemplo, Alto, Médio e Baixo fornecem três classificações simples. Nenhuma delas tem um significado direto para o negócio. Alguns esquemas mais elaborados começaram a ser usados com a chegada de métodos de desenvolvimento de software Ágil como MoSCoW ("Must have", "Should have", "Could have", "Won't have"). Outros métodos como *Feature Driven Development* usam uma versão modificada e simplificada da técnica *Kano Analysis* muito popular em companhias japonesas. No entanto, outros defendem uma ordem estritamente numérica (1, 2, 3, 4...) por valor de negócio ou risco técnico. O desafio do último esquema é que ele frequentemente cria um conflito entre itens de risco alto que devem ser priorizados primeiramente e itens de maior valor que também devem ser priorizados primeiramente.

Todos estes esquemas sofrem de um problema fundamental. Para responder à mudança do mercado e outros eventos, é necessário repriorizar. Imagine, por exemplo, que você tem um *backlog* de 400 requisitos priorizados por ordem numérica – 1 a 400 – e que você está entregando incrementalmente em iterações mensais usando um método de desenvolvimento Ágil. Todo mês você terá que repriorizar o *backlog* restante de até 400 itens.

Na minha experiência, solicitar que os donos do negócio priorizem itens é um desafio. O motivo disto é simples: Há muita incerteza no mercado e no ambiente de negócio. É difícil prever o valor futuro de um item em relação a outro, quando algum item será necessário, e se um item terá mais valor se priorizado primeiramente. Solicitar um dono de negócio para priorizar um *backlog* em um sistema de tecnologia é fazer um grande número de perguntas difíceis para as quais as respostas são incertas. Quando as pessoas não têm certeza, elas tendem a ter uma reação ruim; elas podem se mover mais lentamente, elas podem recusar cooperação, elas podem tornar-se desconfortáveis e disfuncionais. Elas podem reagir simplesmente mudando constantemente de opinião, fazendo planos de projeto aleatórios, e desperdiçando muito tempo da equipe reagindo às mudanças.

É necessário um esquema que posterga comprometimentos o máximo possível e que faz apenas uma simples pergunta que é fácil de responder. Kanban fornece isto solicitando aos donos do negócio que preencham os espaços vazios da fila enquanto os fornece um *lead time* confiável e uma métrica de desempenho de entrega na data.

Nós já temos seis objetivos valiosos para o nosso sistema Kanban, e para muitos negócios, eles devem ser o bastante. No entanto, eu e outros precursores de Kanban descobrimos que outros dois objetivos ainda mais valiosos são possíveis e desejáveis.

OBJETIVO 7 FORNECER TRANSPARÊNCIA NO PROJETO E OPERAÇÃO DO SISTEMA

Quando eu comecei a usar sistemas Kanban, eu acreditava em transparência para o trabalho-em-progresso, taxa de entrega (rendimento), e qualidade porque eu entendia que isso contribuiria para a confiança dos clientes e gerentes seniores. Eu estava fornecendo transparência em relação ao local onde a requisição estava no sistema, quando ela seria finalizada, e qual qualidade estava associada a ela. Eu estava fornecendo também transparência em relação ao desempenho da equipe. Fiz isso para fornecer aos clientes confiança de que estávamos trabalhando nas suas requisições e uma ideia de quando elas deveriam estar finalizadas. Além disso, eu queria educar a gerência sênior nas nossas técnicas e desempenho e construir uma confiança em mim como gerente e na minha equipe como um grupo de profissionais engenheiros de software bem formados.

Há um efeito secundário de toda essa transparência que eu não havia previsto. Enquanto tudo fica bem quanto à transparência para requisições de trabalho e desempenho, transparência no processo e em como ele funciona tem um efeito mágico. Ela faz com que todos os envolvidos vejam os efeitos das suas ações ou falta de ações. Como resultado, as pessoas são mais sensatas. Elas mudarão o seu comportamento para melhorar o desempenho do sistema como um todo. Elas irão colaborar em mudanças necessárias no nível de política, pessoal, recursos humanos, e assim por diante.

GOAL 8 PROJETAR UM PROCESSO PARA VIABILIZAR O SURGIMENTO DE UMA ORGANIZAÇÃO DE “ALTA MATURIDADE”

Para a maioria dos líderes de negócio seniores com quem eu falo, esse objetivo final representa seus desejos e expectativas para os seus negócios e organizações de desenvolvimento de tecnologia. Eles procuram por previsibilidade acima de tudo, juntamente à agilidade no negócio e boa governança.

Líderes de negócio querem ser capazes de fazer promessas aos seus colegas numa mesa de comitê executivo, para os seus conselhos administrativos, para os seus *stakeholders*, para os seus clientes, e para o mercado em geral, e eles querem ser capazes de manter as suas promessas. Sucesso no nível de um executivo sênior depende muito de confiança, e confiança requer confiabilidade. Acima de tudo isso, líderes seniores querem riscos gerenciados apropriadamente para que eles possam entregar resultados previsíveis.

Além disso, eles percebem que o mundo de hoje anda mais rápido e que mudança ocorre rapidamente: Novas tecnologias surgem; a globalização muda o mercado de trabalho e o mercado de consumidores, causando grandes flutuações na demanda (por produto) e fornecimento (de trabalho); condições da economia mudam; concorrentes mudam suas estratégias e ofertas de mercado; o mercado experimenta mudança com o envelhecimento da população à medida que essa se torna mais rica e mais classe média. Então líderes de negócio querem que seus negócios sejam ágeis. Eles querem responder à mudança rapidamente e tirar vantagem de oportunidades.

Alinhado a tudo isso eles querem uma boa governança. Eles querem apresentar que os fundos de investimento estão sendo gastos sabiamente. Eles querem os custos sob controle e querem que o risco de investimento no *portfolio* seja distribuído de maneira ideal.

Para fazer tudo isso, eles gostariam de ter mais transparência dentro das suas organizações de desenvolvimento de tecnologia. Eles gostariam de ter acesso a verdadeiros *status* de projetos e gostariam de ser capazes de ajudar quando necessário. Eles querem uma organização gerenciada de forma mais madura que reporta fatos com dados, métricas, e indicadores, e não anedotas e avaliações subjetivas.

Todos esses desejos equiparam-se a uma organização operando no que o SEI define como Nível de Maturidade 4 na sua escala de cinco níveis de capacidade e maturidade do CMMI. Os níveis 4 e 5 são conhecidos nessa escala como sendo os níveis de maturidade mais altos. Pouquíssimas organizações têm alcançado esse nível de maturidade independentemente da solicitação de uma avaliação formal *Standard CMMI Appraisal Method for Process Improvement* (SCAMPI). Não é de se admirar que a maioria dos líderes seniores de grandes empresas de tecnologia é frustrada com o desempenho dos seus grupos de engenharia de software, visto que a maturidade organizacional real não atende o nível desejado.

Conhecer os Objetivos e Articular os Benefícios

Então agora temos um conjunto de objetivos para o nosso sistema Kanban. Precisamos conhecer esses objetivos e sermos capazes de articulá-los, porque antes de começarmos Kanban, precisamos ter o aval dos *stakeholders* na nossa cadeia de valor. Kanban mudará a maneira que interagimos com outros grupos no negócio. Se esses *stakeholders* aceitam as mudanças, seremos capazes de articular os benefícios.

A seguir há um guia passo a passo prescritivo para a inicialização de um sistema Kanban para uma única cadeia de valor na sua organização. Este guia tem sido desenvolvido baseado em experiências reais e validado por muitos iniciantes na adoção de Kanban, aqueles que seguiram esses passos (quase todos) e obtiveram sucesso e aqueles que reconheceram que a sua falha parcial poderia ter sido evitada se esse guia estivesse disponível no momento.

Este guia é fornecido, em parte, para chamar a atenção para a diferença entre Kanban e métodos de desenvolvimento Ágil anteriores. Kanban requer um engajamento colaborativo com a cadeia de valor mais ampla e um gerenciamento de risco médio (e talvez sênior) desde o início. Uma adoção unilateral de Kanban sem primeiramente

construir um consenso entre os gerentes externos e a equipe imediata terá um sucesso limitado e trará benefícios limitados ao negócio.

Tem sido dito para mim que esse conjunto de passos pode parecer assustador, e que algumas pessoas comentaram que tendo lido isso, elas teriam desistido completamente de tentar Kanban. Espero que o âmbito mais amplo desse livro tenha explicado como se engajar em cada um desses passos e tenha fornecido a você conselhos úteis aprendidos com a experiência em campo.

Passos para Dar Início

1. Acorde um conjunto de objetivos para a introdução de Kanban.
2. Mapeie a cadeia de valor (a sequência de todas as ações que a organização de desenvolvimento realiza para atender a requisição dos clientes/*stakeholders*). (Veja capítulo 6)
3. Defina algum ponto onde você quer controlar a entrada. Defina o que vem antes desse ponto e quais são os *stakeholders* envolvidos (explicado no capítulo 6). Por exemplo, você deseja controlar a chegada de requisitos para a equipe de pré-produção de projeto? Os *stakeholders* envolvidos antes desse ponto devem ser os gerentes de produto.
4. Defina algum ponto de saída a partir do qual você não quer ter controle. Defina o que vem depois dele e quais são os *stakeholders* envolvidos (explicado no capítulo 6). Por exemplo, talvez você não precise controlar o cumprimento da entrega do produto.
5. Defina um conjunto de tipos de item de trabalho baseados nos tipos de requisição de trabalho que vêm dos *stakeholders* antes do ponto de entrada (explicado no capítulo 6). Você tem tipos de itens que são sensíveis ao tempo e outros não? Caso positivo, então você pode precisar de algumas classes de serviço (explicado no capítulo 11).
6. Analise a demanda para cada tipo de item de trabalho. Observe a taxa de chegada e variação dos itens. A variação é sazonal ou orientada a eventos? Quais riscos estão associados com este tipo de demanda? O sistema deve ser projetado para lidar com uma demanda média ou de pico? Qual é a tolerância para entregas em atraso ou incerteza para este tipo de trabalho? Crie um perfil de risco para demanda. (explicado no capítulo 6)

7. Reúna-se com os *stakeholders* envolvidos antes do ponto de entrada e depois do ponto de saída do sistema – deve ser uma reunião grande, ou muitas reuniões pequenas. (Explicado com maior profundidade mais à frente neste capítulo.)
 - a. Discuta políticas relacionadas à capacidade da parte da cadeia de valor que você quer controlar e acorde um limite de *WIP* (explicado no capítulo 10).
 - b. Discuta e acorde um mecanismo de coordenação da entrada, como uma reunião de priorização regular, com os parceiros envolvidos antes do ponto de entrada (explicado no capítulo 9).
 - c. Discuta e acorde um mecanismo de coordenação de entrega/*release*, como *release* regular, com os parceiros envolvidos após o ponto de saída (explicado no capítulo 8).
 - d. Você pode precisar introduzir o conceito de classes de serviço diferentes para as requisições de trabalho (explicado no capítulo 11).
 - e. Acorde uma meta de *lead time* para cada classe de serviço dos itens de trabalho. Isso é conhecido como *service-level agreement* (SLA), e foi explicado no capítulo 11.
8. Crie um quadro/parede de cartões para acompanhar a cadeia de valor que você está controlando (explicado nos capítulos 6 e 7).
9. Opcionalmente, crie um sistema eletrônico para acompanhar e reportar o mesmo (explicado nos capítulos 6 e 7).
10. Acorde com a equipe realizar uma *standup meeting* diária na frente do quadro; convide os *stakeholders* envolvidos antes do ponto de entrada e depois do ponto de saída, mas não obrigue o envolvimento deles (explicado no capítulo 7).
11. Acorde em ter revisões operacionais regulares para análise retrospectiva do processo; convide os *stakeholders* envolvidos antes do ponto de entrada e depois do ponto de saída, mas não obrigue o envolvimento deles (explicado no capítulo 14).
12. Eduque a equipe no quadro, limites de *WIP* e sistema puxado. Nada mais no mundo delas deve ter mudado. As atribuições do emprego são as mesmas. As atividades são as mesmas. As entregas são as mesmas. Os artefatos são os mesmos. O processo deles não foi alterado, além do seu pedido para que eles aceitem limites de *WIP* e puxar trabalho baseados na política de classe de serviço em vez de recebê-los de maneira empurrada.

Kanban Requer um Tipo Diferente de Negociação

Kanban requer que a equipe de desenvolvimento de software faça uma negociação diferente com os seus parceiros de negócio. Para entender isto, devemos primeiramente entender as alternativas típicas que estão em uso.

Gerenciamento de projeto tradicional faz uma promessa baseada na restrição tripla escopo, cronograma e orçamento. Após alguns elementos de estimativa e planejamento, é elaborado um orçamento para o provimento de recursos, e um escopo de requisitos e cronograma são acordados.

Gerenciamento ágil de projeto, no entanto, não faz um compromisso tão rígido e arrojado. Pode existir uma data de entrega acordada para alguns meses no futuro, mas um escopo preciso nunca é estabelecido. Alguma definição de alto nível do escopo pode ser estabelecida, mas detalhamentos nunca são realizados. Um orçamento (ou *burn rate*) pode ser acordado para fornecer uma quantidade fixa de recursos. A equipe de desenvolvimento Ágil prossegue de uma maneira iterativa, entregando incrementos de funcionalidades em iterações *time-boxed* (ou *sprints*) curtas. Tipicamente, elas duram de uma a quatro semanas. No início de cada uma dessas iterações, algumas estimativas e planejamentos são realizados e um compromisso é feito. O escopo é priorizado frequentemente e é entendido que se a equipe não pôde manter o compromisso, o escopo será reduzido, e a data de entrega será mantida. No nível de iteração (ou *time-box*), o desenvolvimento Ágil é bem similar ao gerenciamento de projeto tradicional. A única diferença chave é o entendimento explícito que o escopo será reduzido se necessário; enquanto que um gerente de projeto tradicional poderia escolher aumentar o cronograma, adicionar recursos, reduzir escopo, ou uma combinação dos três.

Kanban requer um tipo diferente de negociação. Kanban não procura fazer uma promessa e um compromisso baseado em algo incerto. Uma implementação típica Kanban envolve um consenso de que existirá uma entrega regular de software em funcionamento de alta qualidade – talvez quinzenalmente. É oferecida transparência completa aos *stakeholders* externos em relação ao trabalho e ao processo e visibilidade diária do progresso, caso eles queiram. São oferecidas também oportunidades frequentes de seleção de novos itens mais importantes para desenvolvimento. A frequência deste processo de seleção deve ser maior do que a taxa de entrega – tipicamente, uma vez por semana, embora algumas equipes tenham alcançado uma frequência de seleção por demanda ou taxas frequentes, como diariamente ou duas vezes por semana.

A equipe se compromete a fazer um ótimo trabalho e entregar a maior quantidade de software em funcionamento possível; e a fazer esforço contínuo para aumentar a qualidade, frequência e *lead time* da entrega. Além de oferecer uma flexibilidade inacreditável ao negócio de seleção de itens para processamento em quantidades muito

pequenas, a equipe pode também oferecer uma flexibilidade na priorização e importância fornecendo classes de serviço de trabalho diferentes. Este conceito é explicado no capítulo 11.

Kanban não se compromete com certa quantidade de trabalho entregue numa certa data. Ele oferece compromisso considerando o nível de acordo de serviço para cada classe de serviço junto a um compromisso de entregas regulares confiáveis, transparência, flexibilidade na priorização e processamento, melhoria contínua na qualidade, rendimento, frequência de entrega, e *lead time*. Kanban oferece um compromisso no nível de serviço, balanceando risco através da agregação de uma grande quantidade de itens. Um sistema Kanban projetado apropriadamente oferece um compromisso em relação a pontos que o cliente realmente valoriza. Em troca, a equipe solicita um compromisso de longo prazo para os clientes e parceiros da cadeia de valor: um compromisso de ter um relacionamento de negócio contínuo no qual a equipe de desenvolvimento de software luta constantemente para melhorar o nível de serviço através da melhoria da qualidade, rendimento, frequência de entrega, e *lead time* para entrega. Como o cliente reconhece um relacionamento continuo e de longo prazo, se ele estiver disposto a medir o serviço mais do que pressionar a equipe a ser preciso em relação a qualquer item, o sistema pode funcionar.

A abordagem tradicional para se formar um compromisso em relação ao escopo, cronograma, e orçamento é um indicativo de uma transição única. Isso implica que não há um relacionamento continuo; isso implica num baixo nível de confiança.

A abordagem Kanban é baseada na noção de que a equipe estará junta e engajada num relacionamento de longo período de tempo com o fornecedor. A abordagem Kanban implica repetição de negócio. Ela implica num compromisso para um relacionamento, e não meramente para um pedaço de trabalho. Kanban implica num nível de confiança alto desejado entre a equipe de software e os parceiros da cadeia de valor. Ela implica que todos acreditam que estão formando uma parceria de longo prazo e que eles desejam que essa parceria seja altamente eficaz.

Um compromisso Kanban é convidar a todos da cadeia de valor a cuidar do desempenho do sistema – a cuidar da qualidade e da quantidade de software sendo entregue, da frequência da entrega, e do *lead time* da entrega. Kanban convida os parceiros da cadeia de valor a se comprometer com o conceito de agilidade de negócio verdadeiro e entrar em consenso num trabalho colaborativo para que ele aconteça. Isto diferencia significativamente Kanban das abordagens Ágeis anteriores de desenvolvimento de software.

Estabelecendo uma negociação Kanban com os *stakeholders* envolvidos antes do ponto de entrada e depois do ponto de saída do sistema, você estabelecerá um

compromisso fundamental para o nível de desempenho do sistema. Você estabelecerá a fundação para uma cultura de melhoria continua.

Fazendo uma Negociação Kanban

Uma parte crítica para uma implementação Kanban bem sucedida é a negociação inicial desse tipo diferente de barganha. O que acontece durante essas negociações iniciais é o estabelecimento das regras do jogo colaborativo que será jogado a partir de então. É vital que os parceiros da cadeia de valor sejam envolvidos com o estabelecimento dessas regras, pois será necessário que eles participem do jogo para que ele seja justo e para que as saídas reflitam os objetivos e as intenções.

O passo 7 do processo de 12 passos para introdução de Kanban sugere que nos reunamos com os *stakeholders* envolvidos antes do ponto de entrada no sistema, como marketing e pessoas do negócio que fornecem os requisitos, e os parceiros envolvidos após o ponto de saída no sistema, como operação de sistema e equipe de desenvolvimento ou organização de vendas e entrega. Precisamos acordar com eles as políticas relacionadas ao *WIP*, priorização, entregas, classes de serviço, e *lead time*. O conjunto de políticas acordado com esses parceiros definirá as regras do nosso jogo colaborativo de desenvolvimento de software. É difícil tratar cada um dos cinco elementos isoladamente, visto que eles são essencialmente inter-relacionados. Então embora entendemos que deve haver um conjunto de políticas para cada um dos cinco elementos, as negociações serão circulares por natureza à medida que os participantes iteram sobre as opções. Por exemplo, se uma meta de *lead time* proposta é inaceitável, pode ser possível introduzir uma classe de serviço diferente que oferece um *lead time* menor para certos tipos de requisição de trabalho. Os cinco elementos – *WIP*, priorização, entrega, classes de serviço e *lead time* – fornecem alavancas que podem ser puxadas para afetar o desempenho do sistema. A habilidade é saber como puxar essas alavancas e como negociar opções para chegar a um acordo que funcionará efetivamente.

Limites de *WIP*

Eu conheci um gerente de desenvolvimento em Denmark que me contou que os seus desenvolvedores trabalhavam em sete atividades e meia, em média, simultaneamente; isto é claramente indesejável. Eu me pergunto se alguém acredita realmente que este nível de multitarefa é apropriado. Se eu fosse ele, eu usaria esse fato como ponto inicial para as minhas negociações. Eu abalaria a conversa declarando que, em média, os membros da equipe estão trabalhando em sete atividades e meia em paralelo. Eu

assinalaria o que isso faz com o *lead time* e previsibilidade e convidaria os meus colegas – e outros *stakeholders* – a sugerir qual seria um número melhor. Alguns deles iriam sugerir que apenas um item por pessoa é a melhor ideia. E poderia ser, mas essa é uma escolha muito agressiva. O que aconteceria se algo estivesse bloqueado? Não seria bom ter uma alternativa para onde mudar? Então talvez outra pessoa sugerisse que duas atividades em paralelo é a resposta correta. Alguns podem concordar que três; é provável que o intervalo de opções fique entre uma e três atividades. Se a equipe tem dez desenvolvedores e você chega a um consenso de no máximo duas atividades no processo por pessoa então você terá um acordo de limite de *WIP* de 20 para a equipe de desenvolvimento.

Há outras alternativas. Talvez você queira trabalhar com pares de programadores; então duas atividades por par com dez desenvolvedores significará um limite de *WIP* de dez. Alternativamente, você pode usar um método altamente colaborativo como *Feature Driven Development* ou *Feature Crews*, nos quais equipes pequenas de no máximo cinco ou seis pessoas trabalham numa única *Minimum Marketable Feature*, *User story*, ou em *batches* de *features* (como em *FDD*), conhecido como *Chief Programmer Work Package* (*CPWP*). Uma equipe *FDD* pode acordar limitar *CPWPs* em três para a equipe de dez desenvolvedores. (Um *CPWP* é tipicamente otimizado para eficiência do desenvolvimento baseado na análise da arquitetura do domínio e contém de 5 a 15 funções de alta granularidade.)

Então tivemos uma conversa sobre limites de *WIP* com os nossos *stakeholders*. Fizemos isso discutindo qual a expectativa razoável para multitarefa e relacionando-a a confiabilidade de entrega e às expectativas de *lead lime*. Acordar limites de *WIP* com os nossos parceiros é um elemento vital. Embora possamos declarar unilateralmente os limites de *WIP*, envolver outros *stakeholders* e chegar a um consenso estabelece um compromisso com as regras do nosso jogo colaborativo; em algum ponto no futuro esse compromisso será inestimável. Haverá um dia que nossos parceiros nos solicitarão trabalho adicional. Eles farão isso porque alguma coisa é importante e valiosa. Seus motivos serão genuínos. Quando eles fizerem isso, seremos capazes de responder pedindo a eles para reconhecerem que temos um limite de *WIP* acordado. É provável que nosso sistema esteja cheio e aceitar outro item, embora importante, quebrará esse limite; então nossa resposta deve ser:

“Sim, adoraríamos aceitar esse novo trabalho, visto que percebemos que ele é muito importante para você. Igualmente, você sabe e entende que temos um acordo em relação ao limite de *WIP*. Você fez parte dessa decisão, e você entende porque a fizemos. Queremos ser capazes de processar requisições confiáveis e em tempo hábil. Para atender a sua requisição, precisaremos colocar algo de lado. Qual dos itens atuais em progresso você prefere que seja retirado para dar início ao novo item?”

Se não tivéssemos incluído nossos parceiros na decisão de limite de *WIP*, seríamos incapazes de ter essa discussão. Eles simplesmente continuariam a nos pressionar. Nosso sistema puxado com *WIP* limitado seria quebrado e nossa organização escorregaria por um caminho íngreme de volta a um sistema empurrado.

Se nós temos um jogo colaborativo verdadeiro e bem sucedido de desenvolvimento de software, as regras para o jogo devem ser acordadas através de um consenso entre todos os *stakeholders*.

Priorização

Nós também queremos entrar em consenso num mecanismo para re-preenchimento da fila. Tipicamente, estamos à procura de um acordo para termos uma reunião regular de re-preenchimento e um mecanismo que diga qual novo item será selecionado. Devemos ter essa conversa perguntando, “Se fôssemos fazer uma pergunta muito simples para você, como, “Quais duas coisas você precisa para fazer uma entrega daqui a 42 dias?” com qual freqüência você seria capaz de se reunir conosco para termos essa discussão? “Desejamos que a reunião não dure mais do que 30 minutos.” Como você está oferecendo realizar uma reunião extremamente focada, e você está fazendo uma pergunta muito direta e sugerindo que o compromisso de tempo é mínimo, você verá que os parceiros envolvidos antes do ponto de entrada do sistema estarão dispostos a ser muito colaborativos. Não é incomum conseguir um acordo para uma reunião semanal. Uma freqüência maior é comum em domínios que se movem com maior velocidade como mídia, onde os ciclos de *release* podem ser mais frequentes.

Entrega/*Release*

Agora devemos acordar algo similar com nossos parceiros envolvidos no final da cadeia de valor. Uma cadênciа de entrega que faz sentido é muito específica para o domínio ou situação. Se for um software para a *web*, nós temos que implantá-lo num servidor. Implantação envolve cópia de arquivos e talvez atualização de um esquema de banco de dados e então migração dos dados de uma versão do esquema para outra. Esta migração de dados provavelmente tem o seu código próprio e levará seu próprio tempo para executar. Para calcular o tempo total de implantação, você precisará considerar em quantos servidores ele será instalado, quantos arquivos serão copiados, quanto tempo levará para desligar os sistemas e reiniciá-los, quanto tempo será gasto na migração dos dados, e assim por diante. Algumas implantações podem durar minutos, outras horas-ou até dias. Em outros domínios, podemos precisar fabricar mídia física como DVDs,

empacotá-los em caixas, e distribuí-los através de canais físicos como distribuidores, concessionárias, varejistas, ou clientes corporativos. Podem existir outros elementos envolvidos, como a impressão de manuais ou treinamento para equipe de vendas e suporte. Pode ser necessário elaborar um treinamento para essas pessoas.

Por exemplo, em 2002, eu fiz parte do primeiro *release* de uma atualização por estágios da rede de telefonia móvel da Sprint PCS. Esta primeira atualização para a tecnologia 3G foi chamada 1xRTT. O lançamento envolveu o *release* de 15 novos aparelhos com uma meta de 16 *features* novas que utilizavam a capacidade de alta velocidade da nova rede. A Sprint tinha uma rede espalhada através dos Estados Unidos e empregava 17000 pessoas. Eles tinham uma quantidade similar de pessoas em *call centers* que cuidavam das chamadas dos usuários. O canal de vendas a varejo e o atendimento a clientes tinham sido treinados para dar suporte no lançamento do novo aparelho de celular. Por brincadeira, sugeri que a melhor maneira de se fazer isso seria desligar tudo por dois dias, levar todos para Kansas City por uma noite, alugar o estádio Kansas City Chief, onde deveríamos apresentar um PowerPoint de duas horas nas duas telas gigantes do estádio. Essa teria sido a maneira mais eficiente, mas era totalmente inaceitável por muitos motivos. Nossos clientes dificilmente aceitariam um suporte 48 horas fora do ar enquanto treinávamos nossos operadores na próxima geração de tecnologia. E perder dois dias de lucro nas vendas do canal de varejo não teria ajudado nossas metas de lucro anual.

Um programa de treinamento foi elaborado, e uma educação no formato treine os treinadores foi realizada. Um programa para treinamento do pessoal de varejo regional foi elaborado, como também um similar para os *call centers*. Formadores foram enviados a campo por seis semanas para treinamento de grupos pequenos de pessoas à medida que as mudanças aconteciam. O custo de realização do treinamento foi alto. O comprometimento de tempo – seis semanas – foi significante, e a meia vida do treinamento na memória dos funcionários era também de seis semanas. Se perdêssemos a janela de lançamento do novo serviço, o treinamento teria que ser repetido e a implantação seria postergada em no mínimo seis semanas.

Se o seu domínio é como uma rede telefônica, você sabe que a cadência de *release* não será frequente. Quando os custos de transação para se fazer um *release* inclui seis semanas de treinamento, fazer *releases* com uma frequência maior do que anual é proibitivo.

A saída desejável é uma cadência de *release* com a maior frequência que faça sentido. Então comece perguntando, “Se entregamos a você código de alta qualidade com uma quantidade mínima de defeitos, e ele vem com uma nota adequada, transparência em relação a sua complexidade, e confiança na entrega, com qual frequência você poderia distribuí-lo em produção?” Isto provocará uma discussão em relação às definições que

você usou, e alguma reafirmação será requerida. No entanto, você deve pressionar por um resultado que maximize a agilidade no negócio sem estressar demasiadamente qualquer parte do sistema.

Lead Time e Classes de Serviço

Quando conversamos sobre *lead time*, ajuda termos algum dado histórico de desempenho anterior. Idealmente, queremos ter um *lead time* e dados de tempo das atividades de engenharia. No exemplo da Microsoft do capítulo 14, sabemos que o *lead time* foi em torno de 125 dias para defeitos de severidade 1 e 155 dias para outras severidades. A primeira coisa que deve chamar a sua atenção nisso é que há duas classes de serviço. Defeitos de severidade 1 têm recebido historicamente alguma forma de tratamento especial. Nunca houve alguma formalidade em relação a isso, mas o resultado disso é que defeitos de severidade 1 foram processados mais rapidamente.

Saber isso nos ajuda a oferecer duas classes de serviço diferentes. Podemos sugerir aos *stakeholders* externos que iremos adotar duas classes de serviço e que teremos *lead times* distintos para cada uma delas.

Igualmente, nós também sabemos a partir dos dados históricos que o esforço médio da engenharia era de 11 dias e que o valor maior foi de 15 dias. Então decidimos sugerir um *lead time* de 25 dias a partir da seleção na fila de entrada. Não há mais ciência envolvida do que essa. Agora imagine o efeito psicológico disso. O negócio tinha um desempenho médio de quatro a cinco meses e acabamos de sugerir 25 dias. A diferença é que oferecemos um *lead time* de 25 dias sem incluir nenhum enfileiramento inicial e os 155 dias era um *lead time* que envolvia enfileiramento. Não obstante isso soe como uma melhoria fantástica. Não é uma surpresa que a empresa concorde.

Existem outras alternativas. Você pode pegar os dados históricos do esforço de engenharia e colocá-lo num gráfico de controle de processo estatístico. Isso dará a você um limite máximo de controle (ou 3-sigma). Você pode então querer aumentar o número de limite máximo por segurança para absorver variações externas. No entanto, se você fizer isso, você deve ser transparente com os seus parceiros e mostrá-los como você está calculando os números.

Outra alternativa seria perguntar qual o nível de resposta que o seu negócio realmente precisa. Isso pode ser feito no contexto de um conjunto de classes de serviço. Por exemplo, se o negócio responde, “Precisamos entregar em três dias.” Você deve responder, “Tudo precisa ser entregue em três dias?” A resposta certamente será, “Não” Isto poderá dar a você a oportunidade de solicitar uma definição de tipos de requisição que precisam ser entregues em três dias. Você pode então criar uma classe de serviço

para este tipo de trabalho. Então repita o processo para o restante do trabalho. A saída deve ser uma estratificação das requisições de trabalho em muitas faixas para as quais uma classe de serviço pode ser criada. É provável que cada uma dessas faixas contenha trabalho que possui o mesmo formato de função para custo de atraso. Os detalhes envolvidos na criação de classes de serviço e o conceito de função de custo de atraso são totalmente explicados no capítulo 11.

A meta de *lead time* que você acordar para cada classe de serviço deve ser apresentada como uma meta e não como um compromisso. Você se comprometerá a fazer o seu melhor para alcançar a meta e a reportar o desempenho de entrega na data em relação ao *lead time* do *service-level agreement* (SLA) para cada classe de serviço. Em algumas situações, não haverá confiança suficiente que permita um acordo que o *lead time* do SLA é uma meta em vez de um compromisso. Se você precisa acordar que o *lead time* no SLA representa um compromisso, você deve aumentar a meta com uma margem de segurança. Isto irá deixar claro que um nível baixo de confiança resulta em custo econômico direto.

O critério de saída para a discussão com o seu parceiro em relação a isso é: Você ter um consenso em relação aos limites de *WIP* ao longo da cadeia de valor; você ter um acordo para a coordenação de priorização e o método a ser usado; você ter um acordo similar para a coordenação de entrega e o método para isso; e você ter uma definição de um conjunto de acordos de nível de serviço que incluem uma meta de *lead time* para cada classe de serviço.

Takeaways

- ❖ Há pelo menos oito objetivos possíveis para a introdução de Kanban na sua organização.
- ❖ Melhore o desempenho através de melhorias no processo introduzidas com resistência mínima.
- ❖ Faça entregas com alta qualidade.
- ❖ Tenha um *lead time* previsível controlando a quantidade de trabalho-em-progresso.
- ❖ Proporcione uma vida melhor aos membros da sua equipe através de um melhor balanceamento trabalho/vida.
- ❖ Forneça folga no sistema balanceando demanda e rendimento.
- ❖ Forneça um mecanismo de priorização simples que posterga comprometimento e mantém as opções abertas.
- ❖ Forneça um esquema transparente para visualizar oportunidades de melhoria, possibilitando, desse modo, uma mudança para uma cultura mais colaborativa que estimula a melhoria contínua.
- ❖ Lute por um processo que permita resultados previsíveis, agilidade no negócio, boa governança, e o desenvolvimento que o Software Engineering Institute chama de organização de alta maturidade.
- ❖ É importante definir os seus objetivos e ser capaz de articular os benefícios da introdução Kanban para conseguir um acordo consensual com outros *stakeholders*.
- ❖ Siga o guia de 12 passos para introduzir o processo Kanban.
- ❖ Kanban requer um tipo diferente de barganha com os *stakeholders* externos e donos do negócio. É uma barganha baseada na suposição de um relacionamento de longo prazo e num comprometimento em relação ao nível de desempenho do sistema.
- ❖ Incluir *stakeholders* externos para se chegar a um acordo nos elementos básicos do sistema Kanban faz com que eles se tornem colaboradores.
- ❖ Políticas básicas para limite de *WIP*, metas de *lead time*, classes de serviço, priorização, e entrega, representam as regras do jogo colaborativo de desenvolvimento de software.
- ❖ Envolver *stakeholders* externos como colaboradores para acordar as regras do jogo permitirá um comportamento colaborativo mais tarde, quando o sistema será colocado sobre *stress*.

❖ PARTE QUATRO ❖

FAZENDO MELHORIAS

❖ CAPÍTULO 16 ❖

Três Tipos de Oportunidade de Melhoria

Capítulos 6 ao 15 descrevem como construir e operar um sistema Kanban e a adotar a abordagem Kanban para gerenciamento de mudança e melhoria. O restante do livro descreve como identificar oportunidades para melhoria, o que fazer com elas, e como escolher entre elas.

O capítulo 2 identifica as cinco principais propriedades que você deve encontrar numa organização usando Kanban. A quinta propriedade descreve como os modelos são usados para identificar, avaliar, e orientar oportunidades de melhoria. Há muitos modelos possíveis. Este capítulo foca em três modelos e suas variantes: A Teoria das Restrições e o seu *Five Focusing Steps*; um subconjunto de ideias do Pensamento *Lean* que identificam atividades desnecessárias, como custos econômicos; e algumas variações que focam no entendimento e redução da variabilidade. Outros modelos são possíveis. A comunidade já experimenta modelos como Teoria da Opção Real e Gerenciamento de Risco. O que está descrito aqui são exemplos. Eles são um início. Eu gostaria de motivá-lo a adotá-los, visto que sei que eles funcionam, e gostaria de incentivar você a expandir seu pensamento e olhar para a grande variedade de modelos que capacitam e delegam à equipe a geração de melhorias.

Gargalos, Eliminação de Desperdício, e Redução da Variabilidade

Cada um destes modelos para melhoria tem sido completamente explorado e tem desenvolvido seu próprio corpo de conhecimento. Cada um tem a sua própria escola de pensamentos em melhoria contínua. Com Kanban, escolhi sintetizar os três e fornecer uma visão geral de como identificar oportunidades de melhoria usando cada modelo. Cada uma das três escolas de pensamento em melhoria contínua descrita abaixo tem o seu próprio grupo de pensadores líderes, suas próprias conferências, seu próprio critério de conhecimento e experiência, e seu próprio grupo de seguidores. Sua empresa deve se inscrever em uma ou mais destas escolas. Ser capaz de mostrar como as técnicas Kanban podem fornecer oportunidades para melhoria usando a técnica favorita da sua organização será uma vantagem. Saber que você tem um conjunto vasto de paradigmas de melhoria e ferramentas para escolher fornecem uma flexibilidade maior para a realização de mudanças.

Aqueles amplamente familiarizados com metodologias de melhoria contínua podem pular o resto deste capítulo e irem para o capítulo 17. Aqueles que desejam uma visão geral dos métodos disponíveis, e alguma referência na literatura e história, podem achar o restante deste capítulo valioso.

Teoria das Restrições

A Teoria das Restrições foi desenvolvida por Eli Goldratt e publicada pela primeira vez no seu livro sobre empresas, *The Goal*, em 1984. Nos últimos 25 anos, *The Goal* passou por muitas revisões e o framework teórico conhecido como *Five Focusing Steps* tem se tornado mais óbvio em edições mais recentes.

O *Five Focusing Steps* é a base para melhoria continua na Teoria das Restrições. Ele é conhecido como POOGI (*Process Of OnGoing Improvement*). A Teoria das Restrições (ou TOC – *Theory of Constraints*) é cheia de acrônimos. Estranhamente, *Five Focusing Steps* é uma exceção; ele não é abreviado para “FFS”.

Na década de 90, a Teoria das Restrições evoluiu um método para análise da causa raiz e gerenciamento de mudança conhecido como *Thinking Processes* (ou TP). O motivo para este desenvolvimento foi a descoberta junto à comunidade de TOC que as suas restrições para o alcance de melhoria com os clientes eram gerenciamento de mudança e resistência a mudanças.

Parecia que o *Five Focusing Steps* funcionava bem apenas para problemas de fluxo e que muitos desafios no trabalho não se encaixavam perfeitamente no paradigma de fluxo. Então TP evoluiu. A qualificação profissional e o programa de treinamento

para os consultores TOC mudaram do uso de *Five Focusing Steps* e suas aplicações, como *Drum-Buffer-Rope*, para TP. Portanto, quando muitas pessoas da comunidade se referem a TOC, estão na verdade se referindo a TP e não aos *Five Focusing Steps*. Observei ao participar de conferências TOC que o uso do *Five Focusing Steps* dentro da comunidade TOC transformou-se em algo como uma arte esquecida.

Até onde eu tenho visto a comunidade TOC tende a aceitar paradigmas como eles são estabelecidos em vez de desafiá-los. Portanto, a solução TOC para gerenciamento de projeto, Corrente Crítica, evoluiu apoiada no paradigma de restrição tripla do gerenciamento de projeto (escopo, orçamento e cronograma) e o modelo gráfico de dependência para agendar as atividades no projeto. Ninguém desafiou esse modelo. Ninguém até eu publicar meu primeiro livro, *Agile Management for Software Engineering*, que desafiou o paradigma de gerenciamento de projeto e sugeriu que é melhor modelar projetos como cadeias de valor e fluxo de problemas e aplicar o *Five Focusing Steps*. Fazendo isso, seria então possível usar todo o corpo de conhecimento *Lean*, baseado em fluxo, e sintetizá-lo com foco *Five Focusing Steps* nos gargalos. A síntese de TOC com *Lean* possibilitaria melhorias no projeto e no desempenho da organização e a base para o surgimento de Kanban.

Tenho concordado que qualquer processo ou fluxo de trabalho que envolve divisão de trabalho pode ser definido como uma cadeia de valor; e que qualquer cadeia de valor pode ser observada como um fluxo. *Lean* e *Toyota Production System* foram construídos essencialmente ao redor dessa suposição. Se qualquer cadeia de valor tem fluxo, então o *Five Focusing Steps* pode ser aplicado. Portanto, o *Five Focusing Steps* é perfeitamente um satisfatório POOGI, e TP não é necessário a menos que você esteja usando-o como uma ferramenta para gerenciamento de mudança. Eu pessoalmente não desenvolvi uma afinidade com TP. Minha ferramenta de gerenciamento de mudança preferida é Kanban, como mostra esse texto.

Five Focusing Steps

O *Five Focusing Steps* é uma fórmula simples para um processo em melhoria contínua. Ele declara:

1. Identifique a restrição.
2. Decida como explorar a restrição.
3. Subordine tudo o mais no sistema à decisão tomada no Passo 2.
4. Eleve a restrição.
5. Evite inércia; identifique a próxima restrição e retorne ao Passo 2.

O Passo 1 nos solicita a identificar de um gargalo na nossa cadeia de valor.

O Passo 2 nos solicita a identificar o rendimento potencial do gargalo e compará-lo com o que está realmente acontecendo. Como você perceberá, o gargalo estará raramente ou nunca funcionando com toda a sua capacidade. Então pergunte, “Como conseguir atingir toda a capacidade do nosso gargalo? O que precisaremos mudar para isso acontecer?” Esta é a “decisão” do Passo 2.

O Passo 3 nos solicita a fazer qualquer que seja a mudança necessária para implementar as ideias do passo 2. Isto deve envolver fazer mudanças adicionais em outros lugares da cadeia de valor para obter toda a capacidade do gargalo. Esta ação de maximizar a capacidade do gargalo é conhecida como “explorar o gargalo”.

O Passo 4 sugere que se o gargalo está operando com toda a sua capacidade e ainda não produz rendimento suficiente, sua capacidade precisa ser reforçada para aumentar o rendimento. O passo 4 nos solicita a implementar uma melhoria para reforçar a capacidade e aumentar o rendimento de maneira suficiente, aliviando o gargalo e fazendo com que a restrição no sistema se move para outro local na cadeia de valor.

O Passo 5 requer que demos tempo para as mudanças se estabilizarem e então identificar um novo gargalo na cadeia de valor e repetir o processo. O resultado é um sistema de melhoria continua no qual o rendimento sempre aumenta.

Se os *Five Focusing Steps* são institucionalizados apropriadamente, toda a organização alcançará uma cultura de melhoria continua.

O capítulo 17 explica como identificar e gerenciar gargalos usando o *Five Focusing Steps*.

Lean, TPS, e Redução de Desperdício

Lean surgiu no início da década de 90 após o texto seminal, *The Machine That Changed the World*, de Womack, Jones, e Daniels, descrito a partir de uma observação empírica e externa de como o *Toyota Production System (TPS)* funcionava. As primeiras literaturas sobre *Lean* tinham algumas falhas. Elas falharam em identificar o gerenciamento da variabilidade que é inerente ao TPS e isso foi aprendido e adaptado a partir do *System of Profound Knowledge* de Deming. *Lean* também foi vítima de má interpretação e uma simplificação exacerbada. Muitos consultores *Lean* avançaram sobre o conceito de Redução de Desperdício (ou eliminação) e ensinaram *Lean* como um exercício puro e simples de eliminação de desperdício. Neste anti-padrão de *Lean*, todas as atividades de trabalho foram classificadas como valor-agregado e sem-valor-agregado. As atividades desnecessárias, sem-valor-agregado eram então sub-classificadas em desperdício necessário e desnecessário. As atividades desnecessárias eram eliminadas e as necessárias eram reduzidas. Embora esse seja um uso válido das ferramentas *Lean*

para melhoria, ele tende a não considerar as saídas, para a redução de custos, e o valor, visto que não abrange as ideias *Lean* de Valor, Cadeia de Valor e Fluxo.

Kanban viabiliza todos os aspectos do pensamento *Lean* e fornece as ferramentas para melhorar o resultado do valor através do foco no gerenciamento do fluxo, como também na eliminação do desperdício.

O capítulo 18 explica como identificar atividades com desperdício e o que fazer com elas.

Deming e Six Sigma

W. Edwards Deming é geralmente lembrado como um dos três pais do movimento de Garantia da Qualidade no século vinte. No entanto, sua contribuição foi consideravelmente maior. Ele evoluiu o uso do *Statistical Process Control* (SPC) e o desenvolveu para uma técnica de gerenciamento que ele nomeou de *System of Profound Knowledge*. Seu sistema foi elaborado para impedir os gerentes de tomarem decisões de baixa qualidade e substituí-las por decisões objetivas, melhores e contra-intuitivas. Deming é ocasionalmente mencionado como talvez o mais importante cientista gerencial do século vinte, e, em minha opinião, isto é muito merecido. Suas contribuições se estenderam do SPC para Garantia da Qualidade e Ciência Gerencial.

Deming teve uma influência significante na filosofia de gerenciamento japonesa na metade do século vinte, e seu trabalho relacionado ao SPC e *System of Profound Knowledge* é um pilar chave do TPS.

Enquanto algumas equipes altamente maduras Kanban, por exemplo, no banco de investimento BNP Paribas em Londres, adotaram SPC, SPC está além do escopo desse livro e será abordado num texto futuro sobre técnicas Kanban avançadas.

No entanto, os princípios para o entendimento da variação em sistemas e tarefas de trabalho que sustentam SPC são muito úteis. O predecessor de Deming, Walter Shewhart, classificou variabilidade no desempenho de tarefas em duas categorias, causa de oportunidade e causa de atribuição. Mais tarde Deming as renomeou para causa comum e causa especial, e na segunda edição do *The New Economics*, ele admitiu que isto ocorreu por “motivos amplamente pedagógicos”. Não há inovação específica em mudar os termos. Entender variação e como ela impacta no desempenho, e desenvolver a capacidade de classificá-la em uma das duas categorias são habilidades gerenciais necessárias. Aprender as ações gerenciais necessárias que devem ser tomadas baseadas no tipo de variação é essencial para um programa de melhoria contínua. *Lean* e Teoria das Restrições dependem fortemente de um entendimento sobre variação para

viabilizar melhoria, mesmo quando essas melhorias são expressas como gerenciamento de gargalo ou redução de desperdício.

O capítulo 19 explica como identificar variações comuns e especiais e sugere ideias para ações gerenciais apropriadas. O capítulo 20 elabora esse tema; ele descreve como construir uma capacidade de gerenciamento de problemas que responde a variações de causa atribuível com o objetivo de eliminar esses problemas o mais rápido possível para manter o fluxo e maximizar a entrega de valor. Nota: Sem conhecimento e foco no gerenciamento de variabilidade, focalizar o fluxo será ineficaz. *Lean* sem as ideias de Deming é *Lean* sem um entendimento de variação, e, por implicação, é *Lean* sem um foco na manutenção do fluxo. Dado que a literatura inicial de *Lean* não incluiu um entendimento de variação e nenhuma referência ao *System of Profound Knowledge* de Deming, é fácil entender a causa raiz do anti-padrão de ensino de *Lean* como apenas um processo de redução de desperdício.

Enquanto as ideias de Deming eram incorporadas no TPS no Japão no nível de chão de fábrica, onde SPC e o *System of Profound Knowledge* eram empregados para identificar oportunidades de melhoria locais, outro corpo de conhecimento desenvolveu-se nos EUA baseado nas ideias de Deming. *Six Sigma* começou na Motorola, mas se tornou realmente conhecida quando foi adotada pela GE sob a liderança de Jack Welch.

Six Sigma emprega SPC para identificar causas de variação comum e especial e usa um processo similar àquele descrito por Deming para eliminar causa de variação especial e sua causa raiz e evita que eles sejam recorrentes; e adicionalmente, para reduzir causa de variação comum e tornar o processo, fluxo e o sistema mais previsíveis.

Ao contrário de TPS, relacionado a iniciativas de chão de fábrica executadas por trabalhadores com poder de implementar pequenos eventos *kaizen* às centenas e milhares, *Six Sigma* desenvolveu um método de comando-e-controle de menor confiabilidade que tende a envolver pouquíssimas oportunidades de melhoria, geralmente implementadas num nível mais estratégico, e a executá-las como projetos específicos próprios. O líder de projeto carrega o título de Black Belt e tem geralmente anos de treinamento na metodologia para ganhar esse *status*. Como Kanban incorpora as ideias de Deming e fornece instrumentos e transparência para visualização de variabilidade e seu efeito, Kanban pode ser usado para viabilizar um programa de melhoria no estilo *kaizen* ou programa de melhoria no estilo *Six Sigma*.

Adequar Kanban para a Cultura da Sua Empresa

Se a sua empresa é uma companhia *Six Sigma*, Kanban pode ajudá-lo a executar iniciativas *Six Sigma* no desenvolvimento de software, sistemas, produto, ou organização de TI. Se a sua empresa é uma companhia *Lean*, Kanban naturalmente se adéqua. Ele

pode viabilizar uma iniciativa *Lean* completa para o seu desenvolvimento de software, sistemas, produto, ou organização de TI. Se a sua empresa usa a Teoria das Restrições, Kanban pode viabilizar um programa de gerenciamento de restrições completo (remoção de gargalos) para o seu desenvolvimento de software, sistemas, produto, ou organização de TI. No entanto, talvez seja necessário você reformular a implementação do sistema puxado como uma implementação *Drum-Buffer-Rope* em vez de se referir a isso como sendo um sistema kanban. Como Kanban se desenvolveu a partir de uma implementação *Drum-Buffer-Rope*, eu sei que isso funcionará. No entanto, discussões específicas sobre como modelar uma cadeia de valor e atribuir limites de *WIP* para o *Buffer* e *Rope* estão além do escopo desse texto.

Takeaways

- ❖ Kanban requer que modelos sejam usados para identificar oportunidades de melhoria.
- ❖ Kanban dá suporte a pelo menos três tipos de métodos de melhoria continua: Gerenciamento de Restrição (remoção de gargalo), Redução de Desperdício e Gerenciamento de Variabilidade (conhecido como SPC e o *System of Profound Knowledge*).
- ❖ Kanban viabiliza a identificação de gargalos e uma implementação completa do *Five Focusing Steps* da Teoria das Restrições.
- ❖ Kanban viabiliza a visualização de atividades de desperdício, e pode ser usado para viabilizar uma iniciativa dentro do desenvolvimento de software, sistemas, produto, ou organização de TI.
- ❖ Kanban fornece os instrumentos para o uso da *Theory of Profound Knowledge* e *Statistical Process Control* de Deming. Ele pode ser usado para orientar uma iniciativa *kaizen* ou uma iniciativa *Six Sigma*.

❖ CAPÍTULO 17 ❖

Gargalos e Disponibilidade Não-Instantânea

Washington SR-520 é uma auto-estrada que liga Seattle às periferias do nordeste de Kirkland e Redmond. Ela é a artéria principal para os moradores da periferia suburbana que trabalham no centro da cidade e para os funcionários da Microsoft e outras empresas de alta tecnologia localizadas nestes subúrbios, como AT&T, Honeywell, e Nintendo, que moram na cidade e transitam na direção oposta todos os dias da semana. Durante oito horas todos os dias, a rodovia tem um gargalo de tráfego sério nas duas direções. Se você ficar na ponte que atravessa a auto-estrada na Rua N.E. 76 no subúrbio pequeno de Medina (logo após a rua de propriedade de Bill Gates, às margens do *Lake Washington*), ao final da tarde e olhar para o lado leste, você verá a ligação com o tráfego do lado oeste da cidade rastejando lentamente até a colina Bellevue antes de fundir-se às duas faixas para cruzar a ponte flutuante para Seattle. A velocidade do tráfego até a colina é de aproximadamente 10 milhas por hora e o fluxo é irregular, com os veículos constantemente diminuindo a velocidade e parando. Se você atravessa a rua e olha para o lado leste em direção aos arranha-céus da cidade de Seattle, o *Space Needle*, e o Olímpicos montanhas à distância, você verá o tráfego movendo-se suavemente até você há quase 50 milhas por hora. Que mágica está acontecendo bem debaixo dos seus pés para que as mudanças de velocidade do tráfego aconteçam de maneira tão dramática que o fluxo se transforme de irregular para suave?

Um pouco antes da ponte, a estrada diminui de três para duas faixas antes de cruzar o lago. A faixa mais à direita da rodovia é uma faixa *high-occupancy vehicle (HOV)* que requer que o veículo tenha dois ou mais passageiros. Ela é frequentada por muitos ônibus públicos e carros particulares. A ação desses veículos afunilando-se no outro tráfego é suficiente para causar ruptura e desaceleração. Nas muitas milhas que precedem a ponte, muitas outras rodovias fundem-se à auto-estrada, aumentando o volume do tráfego que já é pesado em horário de pico. O efeito disso é fluxo irregular e velocidades muito lentas.

Em segurança de tráfego os planejadores preocupam-se com a distância entre os carros. Idealmente, é necessária uma distância suficiente para que os carros possam reagir às mudanças e parar com segurança caso seja necessário. Esta distância está relacionada à velocidade e ao tempo de reação. A recomendação legal de “distância” é de dois segundos. Na linguagem *Lean* este é o tempo ideal entre veículos. Portanto, se há duas faixas e dois segundos entre veículos, o rendimento máximo da estrada é de 30 veículos em cada faixa por minuto ou 60 veículos por minuto; isto é verdade independentemente da velocidade dos veículos. Essas regras são quebradas em velocidades muito baixas ou velocidades excessivamente altas – para aqueles que ultrapassam 50 milhas por hora na SR-520. Por razões práticas o rendimento (referido como a capacidade de criar confusão na gestão do tráfego) é de 3600 veículos por hora.

No entanto, caso você fique em pé na ponte e conte o número de carros que passam abaixo dela numa tarde típica às 5 p.m., você observará que menos de 10 carros por minuto atravessam a ponte flutuante em direção a Seattle. Apesar da demanda alta, a rodovia opera em menos de um quinto do seu rendimento potencial. Por quê?

A ponte flutuante acima do *Lake Washington* é um gargalo. Nós entendemos intuitivamente este conceito. A largura de um gargalo controla o fluxo de um líquido para dentro e para fora de uma garrafa. Podemos tirar o líquido rapidamente da garrafa com um gargalo grande, mas haverá um risco grande de derramá-lo. Com um gargalo menor, o fluxo é menor, mas pode ser mais preciso. Gargalos restringem o potencial do rendimento, neste exemplo, de 60 carros por minuto, ou 3600 por hora, para menos de 10 carros por minuto, ou 600 carros por hora.

De maneira geral, um gargalo pode estar em qualquer lugar do processo onde um *backlog* de itens fica a espera para ser processado. No exemplo da SR-520, esse *backlog* é a fila de veículos ocasionalmente parada até o *Overlake*, sete milhas a leste. Em desenvolvimento de software, pode existir qualquer *backlog* de trabalho não iniciado ou trabalho-em-progresso ; requisitos esperando por análise; trabalho analisado esperando por projeto, desenvolvimento, e teste; trabalho testado esperando por implantação; e assim por diante.

Como discutido, a SR-520 entrega apenas 20 por cento do seu potencial em horário de pico quando ela é mais necessária. Para uma explicação completa sobre isso, precisamos entender como explorar completamente o potencial do gargalo e o efeito que a variabilidade tem sobre o potencial. Estes conceitos são explicados aqui no capítulo 17 e mais a frente no capítulo 19.

Recursos com Capacidade Limitada

A Rua N.E. 76 ponte da SR-520 é um gargalo de capacidade limitada. Sua capacidade é de 60 carros por minuto em duas faixas. Antes disso a estrada tem três faixas, fazendo com que o tráfego afunile para cruzar o lago no ponto mais antigo da ponte que foi projetado 50 anos atrás com apenas duas faixas. Naquele tempo, havia muita capacidade e a ponte não era um gargalo. Os subúrbios ao leste eram pequenas vilas, e o deslocamento pela água era raro – e naqueles dias, apenas em direção à cidade e não o inverso como é comum atualmente.

Ações de Elevação

O gargalo com capacidade limitada da SR-520 é similar a um *user-experience designer* numa equipe de software onde é responsável por projetar todas as telas e caixas de diálogo nas quais o usuário interage. Ela trabalha a todo vapor, mas seu rendimento é insuficiente para alcançar a demanda do projeto. A reação natural da maioria dos gerentes nessa situação é contratar outra pessoa para ajudar. Na Teoria das Restrições de Eli Goldratt, isso é conhecido como “elevando a restrição” – adicionar capacidade a fim de que o gargalo seja removido.

No nosso exemplo da SR-520 seria algo equivalente a substituir a ponte flutuante que atravessa o *Lake Washington* por uma nova fonte com três faixas de tráfego nos dois sentidos. Para manter tudo igual, deveria ser uma ponte composta por uma faixa HOV e uma faixa para bicicleta, como também duas faixas abertas para todo o tráfego; esta é, de fato, a ação a qual o Washington State Department of Transportation está se dedicando. A ponte custará muitas centenas de milhões de dólares e levará uma década para ser construída. Durante a escrita deste livro a construção não foi iniciada.

Elevar a capacidade limitada com recursos deve ser a última opção. Aumentar a capacidade do gargalo custa tempo e dinheiro. Se tivermos que contratar outro *user-experience designer*, por exemplo, será necessário levantar orçamento para pagar esta nova pessoa, como também orçamento para bancar o processo de contratação, o qual pode incluir taxas que deverão ser pagas aos agentes para indicações. O progresso do

projeto atual diminuirá enquanto revisão de currículos e entrevistas com os candidatos são realizadas. Nossa recurso mais precioso, nosso *user-experience designer* que restringe o gargalo, será solicitado a ler currículos, selecionar candidatos, e entrevistá-los, gastando tempo do projeto. Como resultado, sua capacidade de finalizar *designs* será reduzida, como também seu potencial de rendimento para todo o projeto. Isto é em parte a afirmação da “Lei” de Fred Brook que adicionar pessoas a um projeto atrasado apenas faz com que ele atrasse ainda mais. Embora a observação de Brook tenha sido anedótica, e que agora possamos fazer uma explanação mais científica deste fenômeno, a indústria de software entendeu, pelo menos nos últimos 35 anos, que a ideia de contratar mais pessoas atrasa o projeto.

Ações de Exploração/Proteção

Em vez de escolher imediatamente elevar a capacidade limitada e gastar mais dinheiro e tempo enquanto diminui o ritmo do projeto, é melhor procurar maneiras de explorar completamente a capacidade do recurso do gargalo. Por exemplo, é observado que a SR-520 tem um rendimento de apenas 20 por cento do seu potencial em horário de pico. Quais ações poderiam ser tomadas para melhorar esse rendimento? Vamos sonhar por um momento. Se o rendimento da estrada no horário de *rush* estivesse alcançando seu potencial de 3600 veículos por hora, seria necessário substituir a ponte existente por uma nova? O tempo de viagem poderia ser curto o suficiente que os contribuintes do Estado de Washington (como o autor) prefeririam gastar seus dólares de impostos em outras coisas mais importantes e urgentes? Tal como mais livros em escolas locais? Talvez!

Como poderíamos então explorar o verdadeiro potencial da rodovia? Na realidade a origem dos problemas está nos condutores dos veículos. O tempo de reação deles e as ações que eles tomam são altamente variáveis. À medida que os carros afunilam-se a partir da faixa *HOV*, os veículos da faixa central precisam desacelerar e dar espaço para um carro entrando na faixa. Alguns motoristas reagem mais lentamente do que outros; outros freiam mais vigorosamente do que outros, e efeito disso é o tráfego se comportar de maneira imprevisível. Alguns motoristas, distraídos pela flutuação na faixa a frente e a velocidade reduzida em relação à faixa da esquerda, decidem mudar para a faixa da esquerda. O mesmo efeito acontece novamente. Todos os veículos diminuem de velocidade, mas a velocidade não afeta realmente o rendimento. A distância entre os veículos é o mais importante. O que se deseja é um fluxo mais suave do tráfego com uma distância de dois segundos entre os veículos*. No entanto, com o elemento

* California footnote needed

humano os veículos não aceleram ou desaceleram suavemente, causando um efeito sanfona na distância entre os veículos. O tempo de reação dos indivíduos em pressionar o acelerador e os pedais de freio, e o tempo de reação das peças e transmissão para as engrenagens dos veículos significa uma diminuição da distância com o aumento do tráfego. A variabilidade no sistema tem um alto impacto no rendimento.

Corrigir este problema para a SR-520 nos leva a ilha da fantasia em termos de controle de veículos, embora algumas indústrias alemãs tenham experimentos com esse tipo de sistema. Sistemas que usam radar ou lasers para avaliar a distância entre veículos e manter o tráfego movendo-se de maneira suave podem remover a variabilidade que ocorre na SR-520. Tais sistemas têm a habilidade de desacelerar suavemente cadeias inteiras de veículos, mantendo a distância entre eles. Como resultado, o rendimento do tráfego permanece alto. No entanto, eliminar a variabilidade de veículos privados dirigidos pelos seus ocupantes tem alguns limites. Se você quer um transporte de baixa variabilidade, você teria que acorrentar os carros uns aos outros e colocá-los em trilhos. Esse é o motivo fundamental pelo qual o transporte ferroviário de massas é mais efetivo do que automóveis que transportam um grande número de pessoas rapidamente.

A boa notícia é que em nosso escritório, nossos recursos de capacidade limitada são afetados por variabilidade que podemos tratar. Falamos muito nesse livro sobre atividades de coordenação e custos de transação associados a atividades que agregam valor. Se tivermos um *user-experience designer* com capacidade restrita, podemos procurar manter essa pessoa ocupada trabalhando em atividades que agregam valor minimizando a quantidade de atividades sem valor agregado (desperdício) solicitadas a ela.

Por exemplo, em 2003 eu tinha uma equipe de testes com capacidade limitada. A fim de maximizar a exploração de toda a sua capacidade, procurei por recursos com mais folga e encontrei analistas de negócio e um gerente de projeto. A equipe de teste foi liberada de atividades burocráticas, como preenchimento de folhas de ponto. Eles também foram liberados do planejamento de projetos futuros. Permitimos que os analistas desenvolvessem planos de teste para iterações e projetos futuros enquanto os testadores estavam ocupados em realizar testes das atividades em progresso.

Uma abordagem melhor que eu não considerei teria sido elaborar um perfil de risco para os requisitos que deveriam ser testados apenas pelos profissionais da equipe de teste. Requisitos que não atendessem os critérios poderiam então ser testados por profissionais de outras áreas que exerceriam um papel amador como testadores; por exemplo, os analistas de negócio. Esta técnica de “bifurcação” usando um perfil de risco é uma boa maneira de otimizar a utilização de um gargalo enquanto gerencia-se risco no projeto.

Uma solução em longo prazo pode ser investir pesadamente em automação de teste. A palavra chave nessa última frase é “investir”. Se você se pega dizendo “investir”, você

está falando numa ação de elevação. Adicionar recursos não é a única maneira de elevar a capacidade. Automação é uma estratégia boa e natural de elevação. A comunidade de desenvolvimento de software Ágil tem feito muito na última década para encorajar automação de testes. Como regra geral, considere automação como uma estratégia de elevação. No entanto, um efeito colateral maravilhoso da automação é que ela reduz a variabilidade: tarefas e atividades são repetidas com precisão digital. Então automação reduz variabilidade no processo e pode ajudar a melhorar a exploração da capacidade de outro gargalo.

O próximo caminho para garantir exploração máxima da capacidade restringida da sua *user-experience designer* é garantir que ela está sempre fazendo progresso no trabalho atual. Se a *user-experience designer* reporta que ela está bloqueada por algum motivo externo, o gerente de projeto e, se necessário, toda a equipe pode se debruçar sobre o problema para resolvê-lo. Uma capacidade organizacional forte para identificar, escalar e resolver problemas é essencial para a exploração de gargalos com capacidade restrita.

Se há muitos problemas bloqueando o trabalho atual, então os problemas impedindo o recurso de capacidade restrita, neste caso, nossa *user-experience designer*, devem ter prioridade máxima. Para um gerenciamento de problemas de alto desempenho e efetivo faz-se necessário conhecer a localização do recurso de capacidade restrita e dar prioridade a ele quando solicitado.

A transparência do sistema Kanban ajudará a identificar a localização do recurso de capacidade restrita (gargalo) e o impacto de quaisquer problemas impedindo o fluxo a partir daquele ponto no sistema. Com todos do projeto conscientes do impacto em nível de sistema de um impedimento no gargalo, a equipe se debruçará com prazer sobre o problema para resolvê-lo. A gerência sênior e os *stakeholders* externos, com interesse que o *release* saia na data, irão também investir uma maior parte do tempo deles mais espontaneamente quando entenderem o valor deste tempo e o impacto que uma resolução rápida do problema proporcionará.

Portanto, desenvolver uma capacidade organizacional de acompanhar e reportar de forma transparente em projetos usando um sistema kanban é crítico para melhorar o desempenho. Transparência leva a visibilidade dos gargalos e impedimentos e, consequentemente, a uma melhor exploração da capacidade disponível para realizar trabalho de valor através de uma equipe focada em manter o fluxo.

Outra técnica que é comumente utilizada para garantir exploração máxima de um recurso de capacidade restrita é garantir que o recurso nunca está ocioso. Um recurso de capacidade restrita deixado sem trabalho devido a problemas inesperados no fluxo de atividades anterior a ele, como, por exemplo, ausência de um analista de requisitos por alguns dias no trabalho devido a problemas médicos familiares, pode ser um

desperdício terrível. De repente a restrição é movida. Ou talvez uma grande parte dos requisitos seja alterada pelo negócio ao se fazer uma mudança estratégica. Enquanto a equipe espera por novos requisitos a serem desenvolvidos, o *user-experience* está ocioso. O que acontece se as atividades anteriores do fluxo são variáveis por natureza? Isto é comum com solicitação e desenvolvimento de requisitos. Portanto, a taxa de chegada de trabalho a ser realizado deve ser regular. Podem existir muitos motivos que façam o recurso de capacidade restrita se tornar ocioso devido a uma lacuna temporária no trabalho. A maneira mais comum de evitar esse tempo ocioso é proteger o recurso do gargalo com um *buffer* de trabalho. O objetivo do *buffer* é absorver a variabilidade na taxa de chegada de novos trabalhos na fila, neste exemplo, para a nossa *user-experience designer*. *Buffering* adiciona um *WIP* total ao nosso sistema. De uma perspectiva *Lean*, adicionar um *buffer* de trabalho adiciona desperdício, e aumenta o *lead time*. No entanto, a vantagem no rendimento fornecida pela garantia de um fluxo constante de trabalho através do nosso recurso de capacidade restrita é normalmente um negócio melhor. Você terá mais trabalho pronto apesar de um *lead time* ligeiramente mais longo e de um total de trabalho-em-progresso ligeiramente maior.

Usar *buffers* para garantir que um recurso de gargalo está protegido de tempo ocioso é frequentemente referido como “proteger o gargalo” ou uma ação de proteção. Antes de considerar elevar um gargalo você deve procurar maximizar a exploração e protegê-lo para garantir que a capacidade disponível é utilizada o máximo possível.

Nosso exemplo de gerenciamento de tráfego na SR-520, onde o rendimento real foi menor do que 20 por cento do potencial é muito comum com trabalhos de conhecimento como análise de requisitos e desenvolvimento de software. É frequentemente possível enxergar melhorias em mais de quatro vezes na taxa de entrega explorando o gargalo.

No exemplo da Microsoft no capítulo 4, foi alcançada uma melhoria de duas vezes e meia através de uma melhor exploração e proteção que removia a variabilidade do sistema sem gastar qualquer dinheiro ou elevando o gargalo.

Ações de Subordinação

Uma vez que você tenha decidido explorar e proteger um recurso de capacidade limitada, você pode precisar tomar uma ação – subordinar outras coisas no sistema – para fazer o seu esquema de exploração funcionar efetivamente.

Deixe-me revisitar nosso sistema de tráfego de fantasia num mundo futuro. Aqui, decidimos não construir uma nova ponte flutuante através do *Lake Washington*; em vez disso decidimos encaixar todos os veículos viajando na SR-520 em horário de pico

num sistema de gerenciamento de velocidade que usa radar e comunicação *wireless* para regular a velocidade do tráfego em um trecho de sete milhas da rodovia. Esse novo sistema agirá como um controle de cruzeiro e substituirá o uso manual do acelerador e pedais de freio. Os cidadãos serão incentivados a se ajustarem ao sistema com benefícios fiscais. Uma vez que uma quantidade suficiente de carros tenha o sistema, ele pode ser ligado e os carros que não o tiverem terão que encontrar uma rota alternativa ou atravessar a rodovia fora dos horários de pico. O resultado será um tráfego suavizado e um aumento da exploração da capacidade do gargalo. Meu palpite é que um sistema assim, se efetivo, poderia recuperar 50 por cento da capacidade perdida. Ou colocando de outra maneira, ele poderia aumentar o rendimento através da SR-520 em horários de pico em aproximadamente duas vezes e meia.

Então o que fizemos neste exemplo? Nós subordinamos o direito do motorista de empregar e controlar sua própria velocidade em busca de um objetivo maior de uma viagem mais rápida facilitada por um rendimento maior através da ponte. Esta é a essência de uma ação de subordinação. Alguma coisa terá que mudar para melhorar a exploração do gargalo.

Para os convededores da Teoria das Restrições é frequentemente não intuitivo notar que mudanças requeridas para melhorar o desempenho do gargalo não são usualmente realizadas no gargalo. Enquanto eu revisava o texto do meu primeiro livro²⁰ um membro muito conhecido da comunidade de desenvolvimento de software Ágil sugeriu que usar Teoria das Restrições como uma abordagem de melhoria levaria todos da equipe a querer ser parte de um recurso de gargalo porque eles teriam toda a atenção da gerência. Este é um erro fácil de ser cometido. De maneira não intuitiva a maioria do gerenciamento de gargalos acontece fora do gargalo. Muitas das mudanças focam em reduzir a carga de falhas para o gargalo para maximizar o seu rendimento. Como regra geral, espere maximizar a exploração da capacidade do gargalo, e, portanto maximizar o rendimento, e como resultado, minimizar o tempo de entrega no seu projeto tomando ações em toda a cadeia de valor, e muito provavelmente não no próprio gargalo.

Recursos Disponíveis Não - Instantaneamente

Recursos disponíveis não-instantaneamente não são, estritamente falando, gargalos. No entanto, geralmente eles se parecem com gargalos, e as ações que precisamos tomar para compensá-los são similares em natureza àquelas para o gargalo. Qualquer pessoa que já tenha dirigido um carro e parado num sinal de tráfego entende o conceito de disponibilidade não-instantânea. Enquanto estamos parados num sinal vermelho, o carro não pode fluir pela rodovia. A ausência de fluxo não é causada por limitação da

capacidade na rodovia, mas por uma política que permite aos carros viajando na outra rodovia o direito de atravessar a rodovia na qual o nosso carro está viajando.

Um exemplo melhor, e voltando ao nosso tema sobre transporte no Estado de Washington deste capítulo, seria o sistema de balsa que opera através do *Puget Sound*, ligando o Kitsap e Olympic Península com o continente ao redor da cidade de Seattle. Há três barcas, duas que saem de Seattle e atravessam para Bremerton e Bainbridge, e a minha favorita, a SR-104 que atravessa de Edmonds no lado leste para Kingston no lado oeste. No mapa, a rota da balsa é apresentada como sendo parte da rodovia SR-104. Ela é frequentemente marcada como “pedágio”, em vez de explicitamente dizer “você tem que pegar um barco a partir daqui ;-)”. As pessoas que a utilizam pensam na balsa como uma rodovia disponível não-instantaneamente.

Quando você chega à balsa, você paga um valor em dinheiro e é solicitado a ficar numa área de espera. O tempo de espera típico é em torno de 30 minutos, visto que a balsa gasta mais ou menos 30 minutos para atravessar o *Puget Sound*, e há um período de 10 a 15 minutos para descarregar os veículos, e outro período semelhante para carregar os novos veículos antes da partida. Usualmente a companhia opera dois barcos, então há uma partida a cada 50 minutos aproximadamente. Nos horários de pico eles podem operar três barcos na rota, diminuindo o tempo de espera entre partidas para aproximadamente 35 minutos.

Na maior parte do tempo, a balsa fica praticamente cheia, mas o sistema não tem capacidade limitada. O fato dos carros ficarem numa área de espera – um *buffer* – e então serem carregados na balsa para a partida (*batch* transferido) não indica um recurso de capacidade restrita. As barcas partem uma ou duas vezes a cada hora, com uma capacidade de aproximadamente 200 carros a cada partida.

Em horários de pico, como nas tardes de sexta-feira, a capacidade da barca torna-se restrita. Quando isto acontece, a taxa de chegada de carros que desejam atravessar excede a capacidade de transportá-los. A capacidade é cerca de 300 carros por hora. A quantidade de carros começa a se acumular, enfileirando-se para fora da área de espera, antes do pedágio. Durante as demandas de pico, você pode observar veículos frequentemente acumulando-se por duas milhas através de Edmond ou Kingston. Pouco pode ser feito; os carros precisam apenas esperar. Não é fácil elevar a restrição trazendo outra barca. A agenda e o cronograma das partidas das barcas são projetados para fornecer um nível razoável de serviço num tempo razoável. Ter um excesso de capacidade seria excessivamente caro para os contribuintes do estado que subsidiam o serviço.

Voltando para desenvolvimento de software e trabalho de conhecimento, disponibilidade não-instantânea tende a ser um problema com recursos compartilhados ou pessoas que são solicitadas a realizar uma grande quantidade de tarefas distintas em

paralelo – *multi-tasking*. Como sabemos, não existe realmente algo como *multi-tasking* no escritório; o que fazemos frequentemente é um chaveamento de tarefas. Se formos solicitados a trabalhar em três coisas simultaneamente, trabalhamos em uma por um período de tempo, então chaveamos para a segunda, e então para a terceira. Se alguém está esperando que a primeira tarefa seja finalizada enquanto estamos na segunda ou terceira, então pareceremos ter uma disponibilidade não-instantânea na perspectiva daquela pessoa (e da primeira tarefa).

Um exemplo de disponibilidade não-instantânea que observei ocorreu com um engenheiro de *build*. A companhia tinha uma política que apenas o gerente de configuração da equipe pessoalmente poderia “*buildar*” o código e empurrá-lo para o ambiente de teste. Esta política estava relacionada a uma estratégia de gerenciamento de risco baseada na experiência que desenvolvedores eram menos cuidadosos e poderiam “*buildar*” código que quebraria o ambiente de teste. O ambiente de teste era frequentemente compartilhado entre muitos projetos, portanto o impacto de um *build* ruim era significante. O departamento de tecnologia não tinha uma boa capacidade de coordenação no nível de programa, e a probabilidade da equipe de um projeto estar trabalhando numa área de sistemas integrados de TI que poderia afetar outro projeto era muito grande. A função de coordenação técnica para saber o que estava acontecendo no nível de código entre e pelos projetos foi dada ao departamento de gerência de configuração. Estes profissionais eram conhecidos como engenheiros de *build*. O engenheiro de *build* era responsável por saber o impacto de um conjunto de mudanças em um determinado *build* de software e por evitar a quebra do ambiente de teste a fim de que o fluxo de todos os projetos não fosse afetado por uma queda do ambiente de teste.

Geralmente, um projeto tinha um engenheiro de *build* atribuído a partir do *pool* compartilhado da equipe de engenheiros de configuração. No entanto, a demanda de um único projeto para *builds* de código dentro do ambiente de teste não era suficiente para manter um engenheiro de *build* ocupado por um dia inteiro. De fato, não era suficiente para mantê-lo ocupado por mais do que uma ou duas horas por dia. Portanto, os engenheiros de *build* eram solicitados a fazer várias atividades em paralelo. Eles eram alocados em vários projetos, como também em outros deveres.

Pegue o exemplo de Doug Burros na Corbis; ele foi alocado em duas atividades: ele tinha a responsabilidade de montar novos ambientes e também de manter os existentes. Ele era o engenheiro de gerência de configuração com responsabilidade integral de manter a configuração atual. Isto incluía aplicar *patches* e atualizações de sistemas operacionais e servidores de banco de dados, *patches* e atualizações de *middleware*, configuração de sistema e topologia da rede, entre outras coisas. Ele separava em torno de uma hora por dia para realizar a tarefa de *build*; tipicamente pela manhã, a partir de

10:00 até 11:00. Caso os desenvolvedores solicitassem um *build* de teste às 3:00 p.m., eles esperariam até o dia seguinte. O engenheiro de *build* não estava disponível instantaneamente. O trabalho poderia ser bloqueado e, como a engenharia de suporte era operada usando um sistema kanban, o trabalho poderia voltar toda a cadeia de valor, causando tempo ocioso para muitos outros membros da equipe.

As ações tomadas em resposta ao problema de disponibilidade não-instantânea são extraordinariamente parecidas àquelas para recurso com capacidade limitada.

Ações de Exploração/Proteção

A primeira coisa foi reconhecer que Doug era um recurso disponível não-instantaneamente e observar o impacto disso. O trabalho estava acumulando quando ele não estava disponível porque os limites do kanban eram firmemente definidos. Como Doug era uma fonte de variabilidade no fluxo, o curso correto da ação foi colocar um *buffer* de trabalho antes de Doug. O truque era fazer este *buffer* grande o suficiente para permitir que o fluxo continuasse sem fazê-lo tão grande que Doug se tornaria um recurso de capacidade restrita. Tive uma conversa com ele sobre a natureza da atividade de *build*. Ficou claro que ele poderia fazer o *build* de sete itens dentro de sua hora por dia de disponibilidade. Então criamos um *buffer* com um limite kanban de sete. Introduzimos isso na cadeia de valor e na parede de cartões incluindo uma nova coluna chamada *Build Ready*. Nós aumentamos então o potencial total do *WIP* no sistema para 20 por cento, mas isso funcionou. Embora os *builds* não estivessem disponíveis instantaneamente, era possível manter o fluxo das atividades antes dele durante o dia. O resultado foi um impulso significante no rendimento e um *lead time* menor, apesar do aumento do *WIP*. Outra opção, na qual novamente nós não pensamos naquele momento, poderia ter sido solicitar a Doug trabalhar duas vezes por dia, meia hora cada, em vez de uma hora seguida por dia. Essas duas vezes de meia hora cada uma teriam sido divididas no dia, uma pela manhã, e outra à tarde. Isso poderia ter facilitado o fluxo. O efeito poderia ter sido redução da pressão para fazer o *buffer* da disponibilidade não-instantânea. Talvez o tamanho do *buffer* pudesse ter sido reduzido para dois ou três. Isso teria um impacto de apenas 10 por cento no total do *WIP* e poderia ter resultado em um *lead time* menor através do sistema.

Como regra geral, quando encontrar problemas de disponibilidade não-instantânea, pense em como melhorar a disponibilidade. A última opção é tornar um problema de disponibilidade não-instantânea num recurso disponível instantaneamente.

Ações de Subordinação

Como discutido anteriormente, ações de subordinação geralmente envolvem mudanças nas políticas através da cadeia de valor a fim de maximizar a exploração do gargalo. Quais as opções disponíveis como ações de subordinação para o nosso engenheiro de *build* disponível não-instantaneamente?

A primeira coisa foi examinar a política de solicitar a Doug realizar três funções diferentes. Qual a melhor escolha? Eu conversei isso com a gerente de Doug. Parecia que na equipe dela, os engenheiros gostavam e precisavam de diversidade no trabalho para se manter interessados. Também, ao solicitar aos membros da equipe a realização do *build* do sistema, manutenção no sistema, e *build* de engenharia, manter-se-ia uma habilidade generalista através de todos os membros da equipe e a manutenção do *pool* de recursos flexível. Isto forneceria ao gerente mais opções e evitaria o potencial de gargalos de recurso com capacidade restrita devido a um alto grau de especialização. A generalização também apelava aos membros da equipe de uma perspectiva da carreira e currículo. Eles não queriam se tornar tão restritivamente qualificados. Então solicitar à equipe trabalhar em apenas uma área, como engenharia de *build*, não era desejado.

Outra opção poderia ter sido abandonar a ideia de múltiplas tarefas e dedicar o esforço de Doug na equipe de suporte. Isto teria dado a ele muito tempo ocioso. Ele poderia estar sentando a espera de trabalho, como um bombeiro no quartel esperando uma chamada de um incêndio. Manter Doug em constante espera certamente curaria os problemas de fluxo, mas seria uma escolha razoável?

Os orçamentos eram apertados e adicionar pessoas à equipe de gerenciamento de configuração para fazer as atividades de Doug, manipular os *builds* dos sistemas e manutenção, seria muito caro e talvez impossível. Eu precisaria solicitar ao meu chefe um orçamento para ter outra pessoa porque eu queria ter alguém ocioso a maior parte do tempo. Isto seria uma boa negociação para gerenciamento de riscos?

Para decidir isto precisamos olhar para o custo do atraso da equipe de suporte e compará-lo ao custo de ter outro membro na equipe e ao custo de alternativas para a manutenção do fluxo. A realidade era que poucos itens na fila da equipe de suporte tinham um custo de atraso significante. Então a ideia de que manteríamos alguém ocioso, esperando por trabalho, para aperfeiçoar o fluxo não era uma alternativa viável. Claramente, a ação de exploração ao adicionar um *buffer* de trabalho para manter o fluxo era a alternativa melhor e mais barata.

No entanto, a conversa sobre o que fazer em relação a falta de disponibilidade instantânea de Doug criou um debate na equipe sobre a política de se ter apenas engenheiros de *build* realizando esse trabalho. Discutimos a opção de acabar com a política

e permitir que desenvolvedores fizessem o *build* de código e o jogassem no ambiente de teste. Essa ideia foi rejeitada porque a organização não tinha um método alternativo viável para coordenar os riscos técnicos através dos projetos. Uma opção de fornecer um ambiente de teste dedicado para cada projeto foi rejeitada por motivos de custo e não era uma alternativa prática num período de tempo curto ou médio. Todos continuaram a enxergar o valor da função do engenheiro de *build* e da equipe de gerência de configuração.

Ações de Elevação

No entanto, fazer o *buffer* e adicionar *WIP* para resolver o problema era como um Band-Aid sobre uma ferida (um paliativo). Era como uma alternativa de contorno. E você pode enxergá-la dessa maneira. Foi uma correção tática – uma correção tática efetiva, mas, todavia uma correção tática – que gerou um custo. Como o sistema Kanban foi exposto a um gargalo de disponibilidade não-instantânea e permitiu a equipe ter um grande debate sobre a causa e as possíveis correções, uma discussão sobre ter um humano para fazer o *build* do código como resposta foi inevitável. Seria possível automatizar o processo de *build*? A resposta foi “Sim”, embora o investimento fosse pesado. Seria necessário um desenvolvimento considerável da capacidade em gerência de configuração e uma coordenação entre projetos. Adicionalmente, alguns especialistas em automação deveriam ser contratados por um período de tempo para criar o sistema e fazê-lo funcionar.

Gastamos em torno de seis meses e tivemos dois contratados por oito semanas. O financiamento total final ficou em torno de \$60.000. No entanto, o resultado final foi que Doug não foi mais requisitado a fazer *builds*, e os *builds* eram instantaneamente disponibilizados assim que os desenvolvedores precisassem deles. Neste ponto foi possível eliminar o *buffer* e reduzir o *WIP* do sistema. Isto resultou numa pequena redução do *lead time*.

A automação foi a última rota para elevação do gargalo de disponibilidade não-instantânea. Adicionar capacidade, isto é, contratar outro engenheiro, não era uma boa escolha.

Outro caminho que envolve automação também foi perseguido – virtualização de ambientes. Virtualização já foi lugar comum na nossa indústria; no entanto, naquele momento nosso ambiente de teste ainda era físico. Virtualização não era uma capacidade organizacional. Ao se gastar tempo para que isto ocorresse, o ambiente de teste poderia ser facilmente configurado e restaurado. Isto reduziria o impacto se um *build* quebrasse o ambiente; em gerenciamento de risco isto é uma estratégia de mitigação.

Isto também possibilitaria ambientes dedicados – reduzindo ou eliminado o risco de um *build* quebrar a configuração do projeto de outra pessoa.

Então o *buffer* foi utilizado como uma estratégia de curto prazo e de exploração tática, enquanto a automação foi perseguida como uma estratégia de elevação de longo prazo.

E o nosso exemplo da balsa Edmonds-to-Kingston? Como ela poderia ser elevada? Bem, o Estado de Washington está atualmente considerando duas opções. Uma seria substituir as atuais balsas envelhecidas da frota por uma nova frota de barcos maiores e mais eficientes. No entanto, Washington tem muita experiência com pontes flutuantes. Há duas que atravessam o *Lake Washington*, incluindo uma na SR-520, que é aparentemente a ponte mais longa do mundo desse tipo, e outra que atravessa o *Hood Canal* na SR-104. O que está sendo considerado agora é a possibilidade de construção em tempo recorde de uma nova ponte flutuante através do *Puget Sound* como parte da SR-104 e a substituição completa da balsa. A ponte planejada não apenas resolveria o problema de capacidade limitada em horários de pico, como resolveria problemas de disponibilidade não-instantânea que prejudicam todos os serviços de balsa que são uma opção para fluxo de tráfego. Tal ponte abriria o crescimento da economia de Kitsap e Olympic Península. Talvez daqui a 50 anos alguém esteja escrevendo um livro discutindo como a ponte flutuante da SR-104 que atravessa o *Puget Sound* é um gargalo e um recurso de capacidade limitada durante os horários de pico?

Takeaways

- ❖ Gargalos restringem e limitam o fluxo de trabalho.
- ❖ Gargalos aparecem de duas maneiras: capacidade limitada – incapacidade de realizar mais trabalho; e disponibilidade não-instantânea – capacidade limitada devido à disponibilidade limitada (mas usualmente previsível).
- ❖ Gerenciamos gargalos usando o *Five Focusing Steps* da Teoria das Restrições.
- ❖ Aumentar a capacidade de um gargalo é conhecido como elevação.
- ❖ As ações tomadas para elevar um recurso de capacidade limitada serão tipicamente diferentes das ações tomadas para elevar um recurso de disponibilidade não-instantânea.
- ❖ Elevação pode envolver inclusão de recursos, ou automação, ou mudanças em políticas que fazem com que o recurso previamente não disponível instantaneamente torne-se disponível.
- ❖ Ações de elevação tipicamente são custosas e levam tempo para serem implementadas. Ações de elevação são frequentemente consideradas investimentos estratégicos em melhoria de processo.
- ❖ Frequentemente os gargalos no processo estão atuando bem abaixo da sua capacidade potencial - abaixo da capacidade restrita teórica.
- ❖ O rendimento de um gargalo pode ser melhorado acima do limite da capacidade restrita teórica utilizando ações de exploração e proteção.
- ❖ Tipicamente, proteção envolve adicionar um *buffer* de WIP em frente ao gargalo. Isto é verdadeiro para recursos de capacidade limitada ou recursos disponíveis não-instantaneamente.
- ❖ Ações de exploração tipicamente envolvem mudanças em políticas que controlam o trabalho realizado pelo recurso do gargalo.
- ❖ Classes de serviço podem ser usadas como ações de exploração.
- ❖ Ações de subordinação são tomadas em qualquer lugar da cadeia de valor a fim de possibilitar a exploração desejada ou ações de proteção. Ações de subordinação são tipicamente mudanças de políticas.
- ❖ Ações de exploração, proteção e subordinação são frequentemente fáceis e baratas de se implementar, visto que elas envolvem primariamente mudanças nas políticas. Portanto, maximizar o rendimento de um gargalo através

de uma exploração completa pode ser vista como uma melhoria tática de processo.

- ❖ Apesar da natureza tática de exploração de um gargalo, os ganhos são frequentemente muito bons. Melhorias no rendimento de uma vez e meia a cinco vezes com uma consequente queda do *lead time*, podem ser alcançadas com pequeno ou nenhum custo durante um período curto de meses.
- ❖ Exploração sempre deve ser perseguida primeiramente, antes de se tentar elevação.
- ❖ Não é incomum que um conjunto de ações de exploração e subordinação táticas seja implementado enquanto um plano para uma mudança estratégica para elevar uma restrição seja implementado durante um longo período de tempo.

❖ CAPÍTULO 18 ❖

Um Modelo Econômico para *Lean*

Desperdício (ou *muda* em Japonês) é a metáfora usada em *Lean* (e no *Toyota Production System*) para atividades que não agregam valor ao produto final. A metáfora do desperdício tem se provado problemática com trabalhadores do conhecimento. Frequentemente é difícil aceitar como desperdício tarefas ou atividades que são custosas ou dispendiosas, mas são necessárias ou essenciais para completar um trabalho com valor agregado; por exemplo, reuniões *stand-up* diárias são necessárias para coordenar a maioria das equipes. Essas reuniões não agregam valor direto ao produto final, então, tecnicamente, elas são “desperdício”, mas isso tem sido difícil de aceitar pela maioria dos desenvolvedores Ágeis praticantes. Em vez ter pessoas confusas com argumentos sobre o que é desperdício ou não, eu concluí que seria melhor encontrar um paradigma alternativo e uma linguagem alternativa menos confusa ou com um apelo emocional.

Redefinindo “Desperdício”

Seguindo autores líderes como Donald G. Reinertsen, adotei o uso do linguajar econômico e me refiro a essas atividades de “desperdício” como custos. Classifico custos em três categorias abstratas principais: Custos de Transação; Custos de Coordenação; e Carga Defeituosa. A figura 18.1 ilustra isto.



Figura 18.1 O modelo Econômico de Custo para o desenvolvimento de software *Lean*

A figura 18.1 mostra que no passar do tempo há um número de atividades de valor agregado para uma iteração ou projeto. Ao redor destas atividades estão os custos de transação e coordenação. A capacidade para atividades de valor agregado pode ser deslocada para trabalhos que são considerados carga defeituosa, isto é, trabalho que é ou re-trabalho ou demanda colocada no sistema devido a uma implementação anterior ruim.

Cada um destes custos é discutido em detalhes nas seções seguintes. Eu os descreverei usando um exemplo simples do mundo real – a atividade de pintar a cerca na minha casa em Seattle com um conservante de madeira colorido.

Custos de Transação

A cerca tem vinte e uma partes. O valor do cliente é a entrega de uma parte da cerca pintada com o conservante de madeira. O valor completo da entrega é quando as vinte e uma partes estão pintadas em ambos os lados.

Antes de começar o trabalho, eu tinha primeiramente que procurar todos os materiais. Isto envolveu uma ida a *Home Depot*. Havia também um trabalho de preparação da cerca: reparar, lixar, e aparar as plantas e arbustos a fim de permitir acesso ao trabalho de pintura. Nenhuma destas atividades poderia ser descrita como de valor agregado. O cliente não se preocupa se eu precisei ir a *Home Depot*. O cliente não se preocupa se esta atividade demanda tempo. De fato, ela é irritante, visto que atrasa o início e o final do projeto. Todas estas atividades atrasarão a entrega de valor ao cliente.

Então o projeto tem algumas atividades de configuração que são essenciais antes de se iniciar o trabalho de valor agregado.

Podem existir outras. Pode existir um planejamento. Pode existir também uma atividade de estimativa e um nivelamento de expectativas. O cliente pode ter cotado um preço para o trabalho e a data de entrega. (Neste caso, o cliente do projeto era a minha esposa.)

Quando realmente começou o trabalho de pintura, concluí que quarenta e duas partes da cerca era muita coisa para uma sessão simples. A velocidade da pintura foi de aproximadamente quatro partes por hora. Então o trabalho foi dividido em seis sessões. Se isso fosse desenvolvimento de software, chamaríamos isto de iterações ou *sprints*. Se fosse manufatura chamaríamos de *batches*. Quando comecei uma sessão de pintura simples, eu também tinha algumas atividades de configuração. A primeira era mudar de roupa. Então eu deveria arrumar os materiais: deveria mover a tintura, escovas, e outras ferramentas da garagem para o lugar adequado para aquele dia de pintura; só então eu poderia começar a pintar.

Resumindo, projetos e iterações têm atividades de configuração.

Após algumas horas de pintura, eu queria fazer uma parada. Talvez fosse a hora do almoço, mas eu não poderia simplesmente deixar tudo e começar a almoçar. Eu devia primeiramente me assegurar de que a lata da coloração foi fechada, e então deixar as escovas protegidas, limpando-as ou jogando-as num jarro de água prevenindo que elas não endureçam enquanto faço uma parada. Depois precisaria me limpar. Eu lavo as minhas mãos e tiro meu macacão de trabalho; só então posso ir almoçar.

Quando todo o projeto foi finalizado, uma parte do conservante de madeira sobrou, e toda lata cheia pode retornar a *Home Depot* para reembolso. Então foi necessária outra ida para a *Home Depot*.

Parece que tanto projetos como iterações têm um conjunto de atividades de limpeza.

Em termos econômicos, estas atividades de configuração e limpeza são referidas como custos de transação. Toda atividade de valor agregado tem custos de transação associados. Estas atividades de custo de transação são coisas que o cliente não vê e a maioria não dá valor, e para as quais eles são no máximo ambivalentes. O cliente pode ser forçado a pagar os custos destas atividades, mas preferiria não fazê-lo. Quão frequentemente você chama um encanador para consertar sua máquina de lavar roupas ou máquina de lavar pratos e é cobrado por um valor de \$90 de honorários? Este é um custo de transação. Você preferiria um valor menor? Escolheria um encanador que não cobrasse esta taxa? Os custos de transação não agregam valor. Eles podem ser necessários, mas em termos *Lean*, eles são desperdício.

Então os dois primeiros tipos de desperdício são custos de transação, especificamente: a configuração, ou *front-end*, e a limpeza, ou custos de transação de *back-end*.

Se você considerar isto para as atividades de desenvolvimento de software, você notará que todos os projetos têm um número de atividades de configuração, como

planejamento de projeto, planejamento de recursos e recrutamento, precificação, estimativa, planejamento de risco, planejamento da comunicação, aquisição de instalações, e assim por diante. A maioria dos projetos também tem custos de limpeza e outros custos de transação de *back-end*, como entrega para o cliente, desmontagem de ambientes, retrospectivas, auditorias, treinamento para usuário, e assim por diante.

Iterações também têm custos de transação, incluindo o planejamento da iteração e a seleção do *backlog* (ou requisitos do escopo), talvez estimativa, orçamento, alocação de recursos, e *setup** do ambiente. No back end, elas terão custos de transação que incluem integração, entrega, retrospectiva e configuração do ambiente.

Custos de Coordenação

É necessário coordenação tão logo tenhamos duas ou mais pessoas tentando alcançar um objetivo em comum, juntas. Inventamos sistemas de linguagem e comunicação para a coordenação entre humanos. Quando combinamos em nos encontrar com amigos, beber, jantar, e assistir um cinema na noite de sexta-feira, nós incorremos em custos de coordenação. Todos os emails, mensagens de texto, e chamadas telefônicas que são necessárias para organizar a noite social são custos de coordenação.

Então custos de coordenação em projetos é qualquer atividade que envolve comunicação e agendamento. Quando pessoas na equipes de projeto reclamam que elas não podem fazer trabalho de valor agregado – como análise, desenvolvimento, ou teste – porque elas estão respondendo a emails, elas estão realizando um conjunto de atividades de coordenação: cada email lido e respondido é uma atividade de coordenação. Quando elas reclamam que elas não conseguem realizar trabalho de valor agregado – como análise, programação, ou teste – porque elas estão sempre em reuniões, estas reuniões também são custos de coordenação.

De maneira geral, qualquer forma de reunião é uma atividade de coordenação, incluindo as favoritas da comunidade Ágil, como reuniões *standup* diárias, a menos que a reunião seja projetada para produzir uma entrega de valor ao cliente. Se três desenvolvedores se juntam em frente a um quadro branco e modelam um projeto para o código que eles vão implementar, isto não é uma atividade de coordenação, ela é uma atividade de valor agregado. Porque? Ela é de valor agregado porque ela produz informação que está diretamente relacionada a uma função de valor para o usuário.

Se enxergarmos desenvolvimento de sistemas e de software como um processo de chegada de informações, no qual nosso ponto inicial não tem informação, e ter a

* N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: configuração.

informação completa significa trabalhar em funcionalidades que atendem às necessidades do cliente, então qualquer informação que chega entre o ponto inicial e o ponto final – que nos move para frente no trabalho em funcionalidades que atendem às necessidades do cliente – é de valor agregado.

Se os membros da equipe se reúnem para criar informação de valor agregado, como projeto, um teste, uma parte da análise, ou parte do código, então esta reunião não é um custo de coordenação, é uma atividade de valor agregado.

No entanto, se os membros da equipe se reúnem para discutir *status*, ou atribuição de atividades, ou cronograma que ajudam a coordenar suas ações e o fluxo de entrega, essa reunião É um custo de coordenação e deve ser considerada desperdício. Como tal, você deve procurar reduzir ou eliminar reuniões de coordenação.

Portanto, um *standup* de 5 minutos é melhor do que um *standup* de 15 minutos se ela alcança a mesma quantidade de coordenação. Um *standup* de 15 minutos é melhor do que um *standup* de 30 minutos se ele atinge a mesma quantidade de coordenação.

Você pode pensar em reduzir atividades de coordenação encontrando maneiras melhores de coordenar pessoas.

Uma maneira é dar poder a equipe para se auto-organizar. Gerenciamento do tipo comandar-e-controlar, no qual as pessoas se reúnem para atribuir tarefas aos indivíduos é um desperdício. É melhor deixar a equipe auto-organizar suas tarefas. Auto-organização geralmente reduz os custos de coordenação em um projeto. No entanto para que funcione ela requer informação. Técnicas dentro do Kanban – como o uso de acompanhamento visual da cadeia de valor e visualização do trabalho usando uma parede de cartões e ferramentas eletrônicas e relatórios – fornecem informação de coordenação que possibilita auto-organização e reduz os custos de coordenação no projeto. O uso de classes de serviço e visualização delas com cartões coloridos ou raias na parede de cartões, junto com um conjunto de políticas para classes de serviço, possibilitem auto-organização de cronograma e priorização automática. Isto é algumas vezes referenciado como “auto-expedição” (um termo que associo a Eli Goldrat na referência a gerenciamento de *buffer*).

De maneira geral, quanto mais informação pode ser transparente para os trabalhadores do conhecimento da equipe, mais a auto-organização será possível e menos atividades de coordenação serão necessárias. Deixe a transparência no trabalho, fluxo do processo, e políticas de gerenciamento de risco deslocar as atividades de coordenação. Reduza desperdício através de um uso abrangente de transparência.

Como você Sabe se uma Atividade é um Custo?

Eu descobri que muitas pessoas têm problema em identificar atividades de desperdício. Tenho visto, por exemplo, defensores Ágeis concordarem que reuniões *standup* diárias têm valor agregado. Eu não concordo com essa visão. Eu não posso saber se o cliente se preocupa ou não se a equipe faz reuniões *standup* ou não. O que o cliente quer é funcionalidade que atenda os seus objetivos, entregue na data com alta qualidade. Se a equipe precisa ou não fazer reuniões *standup* diárias para atender a entrega não “vai nem vem” na sua perspectiva.

Então como identificar atividades de custos de transação ou coordenação que geram desperdício?

Acredito que fazer a seguinte pergunta a você mesmo, “Se esta atividade agrega valor verdadeiramente, eu a faria mais?”.

Quando você pergunta aos defensores do Scrum que veementemente concordam que reuniões *standup* diárias agregam valor, se eles fariam o *standup* duas vezes ao dia ou se eles aumentariam de 15 minutos para 30 minutos, eles com certeza irão responder, “Não!”.

“Bem, se reuniões *standup* verdadeiramente agregam valor,” Eu respondo, “então fazê-las com uma frequência maior é uma coisa boa!”

Este é realmente um teste ácido que demonstra a diferença entre uma atividade que verdadeiramente agraga valor e um custo de transação ou coordenação. Desenvolver mais requisitos do cliente claramente agraga valor. Você faria mais se pudesse e o cliente alegremente pagaria mais por isso. Planejamento claramente não agraga valor. O cliente não pagaria por mais planejamento se ele pudesse evitar.

Então pergunte a você mesmo, “Eu faria mais disso?” Desafie outras pessoas com a mesma pergunta sobre as atividades que eles empreenderiam. Se a resposta é, “Não”, então considere o que você deve fazer para minimizar tempo e energia gastos na atividade, ou como você poderia tornar a atividade mais efetiva, e, portanto, reduzir a duração, frequência, ou quantidade da atividade.

Algumas vezes pode ser difícil determinar se uma atividade é um custo de transação ou coordenação. Algumas atividades frequentemente parecem com os dois. Eu vejo essa confusão a todo o momento ensinando Kanban. Como eu faço com os participantes das aulas, eu incitaria você a não desperdiçar muita energia tentando determinar a diferença. O que é importante é você identificar uma atividade que não agraga valor – e, portanto, é um desperdício – e você saber que você quer reduzir ou eliminar essa atividade como parte de um programa de melhoria contínua.

Carga Defeituosa

Carga defeituosa é uma demanda gerada pelo cliente que deve ser evitada através de entregas de qualidade antecipadamente. Por exemplo, várias chamadas de *help desk* geram custos para o negócio. Se o produto ou serviço de software ou tecnologia forem de alta qualidade, mais usáveis, mais intuitivos, mais adequados ao propósito, então haverá menos chamadas. Isso pode possibilitar ao negócio redução do número de pessoas no *call center* e redução de custos.

Várias chamadas tendem a gerar muitos *tickets* de defeito em produção. Na seleção das funções do escopo para um projeto ou iteração, o negócio deve escolher entre novas ideias e defeitos em produção. Defeitos em produção não são apenas defeitos no software; eles incluem problemas de usabilidade e outros problemas não-funcionais como baixo desempenho, falta de resposta sobre carga ou certas condições na rede, e assim por diante. A resolução para um defeito de produção em relação a um requisito não-funcional pode parecer uma nova funcionalidade – um design para uma nova tela talvez – mas na verdade não é. É uma carga defeituosa. Aquele design da nova tela veio à tona devido a um defeito de usabilidade de um *release* anterior.

Carga defeituosa não cria novo valor; ela faz com que o valor de um *release* anterior não seja alcançado. Embora alguns deles possam estar relacionados a variabilidade do mercado ou imprevisibilidade, será descoberto que alguns dos déficits vieram de problemas de *releases* anteriores. Talvez um erro no produto evite o uso de alguma funcionalidade. Por causa disto, clientes potenciais se desligam do produto e retardam a compra ou escolhem um produto concorrente.

Então a figura é confusa. Carga defeituosa ainda agraga valor. Mas o que é importante é que ela agraga valor que já deveria estar lá. Reduzir carga de defeitos reduz a oportunidade do custo de atraso. Reduzir custos significa mais lucros mais cedo. Reduzir carga defeituosa significa mais capacidade disponível para ser gasta em novas funcionalidades. Reduzir carga defeituosa permite ao negócio perseguir mais nichos de Mercado, oferecendo mais produtos. Reduzir carga defeituosa permite mais opções. Reduzir carga defeituosa pode permitir uma redução no tamanho da equipe, e, portanto, uma redução direta dos custos.

Takeaways

- ❖ Desperdício pode ser classificado em três categorias abstratas principais: custos de transação, custos de coordenação e carga defeituosa.
- ❖ O conceito de desperdício é uma metáfora.
- ❖ A metáfora do desperdício não funciona bem para todos, visto que desperdício é frequentemente necessário, embora não agregue valor especificamente. Como resultado, eu o substituí por um modelo econômico de custos.
- ❖ Para determinar se uma atividade é verdadeiramente um desperdício pergunte, “Eu faria mais dela se pudesse?” Se a resposta for não, então a atividade é alguma forma de desperdício.
- ❖ Custos de transação são de dois tipos: atividade de configuração e limpeza, ou atividade de entrega.
- ❖ Custos de coordenação são atividades que são realizadas para atribuir tarefas às pessoas, agendar eventos, ou coordenar o trabalho de duas ou mais pessoas em direção a uma saída comum.
- ❖ Carga defeituosa é um trabalho de valor não agregado que é gerado devido a alguma falha anterior, como um defeito no software, ou por um projeto ou implementação ruim que levou a uma falta de adoção pelo cliente, uma falha em notar a função de retorno, ou uma chegada significante de chamadas no *help desk* ou requisições de serviço.
- ❖ Carga defeituosa usa capacidade que poderia ser utilizada para *features* novas, inovadoras, de valor agregado para o cliente e que geram retorno.
- ❖ Transformar ideias em trabalho e entregar código ao cliente mais rapidamente maximiza o potencial do valor e minimiza o desperdício.

❖ CAPÍTULO 19 ❖

Fontes de Variabilidade

Variabilidade tem sido estudada desde o início de 1920 no processo industrial. O pioneiro no assunto foi Walter Shewhart. Suas técnicas tornaram-se a fundação para o movimento de garantia da qualidade e são elementos fundamentais dos métodos para qualidade e melhoria contínua *Toyota Production System* e *Six Sigma*. As técnicas de Shewhart foram adotadas, desenvolvidas, e avançadas por W. Edwards Deming, Joseph Juran, e David Chambers. O trabalho dessas pessoas inspirou Watts Humphrey e os membros fundadores do *Software Engineering Institute* na *Carnegie Mellon University*, que abraçaram a crença de que o estudo da variabilidade e sua redução sistemática poderiam trazer grandes benefícios para a profissão de engenharia de software.

Há uma grande quantidade de material publicado por Shewhart, Deming e Juran no estudo da variabilidade e seu uso como uma técnica de gerenciamento e base para um programa de melhorias. Adicionalmente, muito tem sido publicado sobre o método de avaliação quantitativo conhecido como *Statistical Process Control* (SPC) que surgiu como a ferramenta principal para o estudo da variação e as ações sobre ela. Durante a escrita desse livro, o uso de SPC está surgindo com equipes que adotam Kanban. No entanto, o uso de SPC é considerado um tópico avançado e de alta maturidade que será visto

mais a frente neste livro. Aqui falaremos sobre variação em termos mais gerais na sua forma mais simples.

Shewhart classificou variabilidade e variação no processo de desempenho em duas categorias: interna e externa.

Fontes internas de variação são variações que estão sob o controle do sistema em operação. Com Kanban para engenharia de software e operações de TI, pensamos no sistema como um processo que é definido por um conjunto de políticas que governam a operação do sistema. Elas podem ser afetadas diretamente por mudanças realizadas por indivíduos, a equipe, ou gerência. Mudanças nas políticas mudam a operação do sistema e o seu desempenho. Portanto, uma mudança na definição do processo, representa uma mudança que afeta as fontes internas de variação. Ironicamente, Shewhart nomeou estas variações geradas internamente de “variação de causa aleatória – *chance-cause variations*”. “Aleatória” implica que a variação é aleatória e a aleatoriedade é uma consequência direta do projeto do sistema. Isso não implica que a aleatoriedade é distribuída uniformemente ou segue uma distribuição padrão. Mudanças no processo do projeto a partir de mudanças nas políticas internas afetarão qualquer média da variação, propagação, e forma da distribuição.

Para usar um exemplo geral, um rebatedor no jogo de baseball tem uma taxa de acertos (conhecida como média de críquete) que indica a frequência com a qual ele acerta arremessos que alcançam a primeira base, ou mais. Rebatedores diferentes exibem taxas diferentes, com um intervalo típico de aproximadamente 0.100 a 0.350. Em um determinado dia, um rebatedor pode não alcançar sua taxa de arremesso típica. Isto é determinado por alguns fatores como seleção do arremesso, o quanto bem os jogadores acertam a bola, e arremessos específicos.

Se mudarmos as regras do baseball para permitir, digamos quatro golpes antes que um rebatedor estivesse fora, então mudaríamos as probabilidades em favor do rebatedor em relação ao arremesso. Como resultado a média do rebatedor aumentaria. Como resultado dessa regra, alguns jogadores mais talentosos poderiam muito bem alcançar taxas de acerto mais altas como 0.500. Este é um exemplo de alteração no sistema que modifica a variação de causa aleatória dentro do sistema.

Se interpretarmos isso para um exemplo específico em desenvolvimento de software, uma variação interna de causa aleatória seria o número de defeitos criados por linha de código, por requisito, por tarefa, ou por unidade de tempo. O valor médio, a propagação, e a distribuição da taxa de defeito podem ser afetados mudando-se as ferramentas e o processo, como insistir em testes unitários, integração contínua, e revisão em pares.

A definição de processo em uso pela sua equipe, expressa como as políticas representam as regras do jogo colaborativo de desenvolvimento de software. As regras do jogo determinam as fontes e a quantidade de variação interna. A ironia está na noção

de que variações de “causa aleatória” estão na verdade diretamente sob o controle da equipe e gerência através da sua habilidade de modificar políticas, mudar o processo, e afetar fontes de variação interna.

Fontes externas de variação são eventos que acontecem que estão fora do controle da equipe ou gerência imediata. Elas são aleatoriedades que vêm de outras equipes, fornecedores, clientes, são “atos de Deus” aleatórios, como eles são conhecidos na indústria de segurança; por exemplo, a interrupção de duas semanas de um *server farm* causada por uma inundação devido a um temporal. Fontes externas de variação requerem uma abordagem diferente de gerenciamento. Elas não podem ser afetadas diretamente por políticas, mas um processo pode ser colocado em uso para lidar efetivamente com variações externas. O corpo de conhecimento que se relaciona diretamente com esse campo é o gerenciamento de problemas e risco.

Shewhart nomeou variações externas como “variações causadas por atribuição – *assignable-cause variations*”. Por “atribuição”, ele coloca que alguém (ou um grupo de pessoas) pode facilmente apontar para a fonte do problema e descrevê-la consistente – como um, “Havia uma tempestade. Choveu muito forte e nosso *server farm* foi inundado.”. Variações causadas por atribuição não podem ser controladas pela equipe ou gerência local, mas podem ser previstas, e planos podem ser feitos e processos projetados para lidar com elas adequadamente.

Fontes Internas de Variabilidade

O processo de desenvolvimento de software e gerenciamento de projeto em uso, acoplados a uma maturidade organizacional e capacidade dos indivíduos da equipe determina a quantidade de fontes internas de variabilidade e o grau desta variabilidade.

Para evitar confusão, Kanban não deve ser pensado como um processo de ciclo de vida de desenvolvimento de software ou um processo de gerência de projeto. Kanban é uma técnica de gerenciamento de mudança que requer alterações no processo existente: mudanças como adicionar limites de trabalho-em-progresso no processo.

Tamanho do Item de Trabalho

O método de análise usado para quebrar os requisitos e especificá-los para desenvolvimento tem o seu próprio grau de variabilidade. Uma dimensão para isto é o tamanho dos itens de trabalho. A literatura prévia que descreve o método *Extreme Programming* explica *user stories* como uma descrição narrativa de uma *feature* como implementada e usada por um usuário final, escrita num cartão. A única restrição era o tamanho do

cartão. O esforço requerido para criar uma *user story* foi descrito como qualquer coisa entre metade de um dia a 5 semanas de trabalho. Em alguns anos, um *template* para escrita de *user stories* surgiu de uma comunidade em Londres.

As a <user>, I want a <feature>, in order to <deliver some value>

O uso deste *template* padronizou enormemente a escrita de *user stories*. Um dos criadores dessa abordagem, Tim McKinnon, relatou para mim em 2008 que agora ele tinha dados para apresentar que a média do esforço para uma *user story* era 1.2 dias e a propagação da variação era de aproximadamente metade de um dia a quatro dias.

Este é um exemplo específico de redução da variação de causa aleatória no método *Extreme Programming* solicitando à equipe a padronizar a escrita de *user stories* usando um *template*. Fazendo isso, Tim mudou as regras do jogo. As regras originais solicitavam aos membros da equipe a escrever *stories* em cartões numa forma narrativa, e as novas regras solicitavam a eles continuar a escrita em cartões, mas seguindo um formato de sentença específico. Estas mudanças estavam claramente sob a influência e controle dos gerentes locais. Elas são internas ao sistema. O tamanho de uma *user story* é controlado por variação de causa aleatória.

Mesclas de Tipos de Item de Trabalho

Quando todo o trabalho é tratado da mesma maneira, e nomeado por um único tipo, há uma probabilidade de uma variação maior no tamanho, esforço, risco, ou outros fatores. Quebrando o trabalho em tipos específicos, é possível tratar tipos diferentes de formas diferentes e fornecer uma previsibilidade maior.

Por exemplo, a comunidade *Extreme Programming* desenvolveu definições de tipo para tamanhos de *stories* diferentes. Elas adquiriram nomes como “élicas” ou “grão de areia”. Uma *story* épica deve ser uma *story* grande que precisará de muitas pessoas e muitas semanas para ser desenvolvida, enquanto que um grão de areia deve ser uma *story* pequena que pode ser completada por um único desenvolvedor ou um par de desenvolvedores em poucas horas. Adotando esta nomenclatura – Épica, *Story*, e Grão de Areia – temos agora três tipos. Para cada tipo individual, a propagação da variação será menor do que seria se todo o trabalho fosse tratado como uma *story* única.

Dentro de um departamento de desenvolvimento de software típico devem existir diferentes tipos de trabalho. Deve existir trabalho de valor para o cliente com um nome como “*feature*”, “*story*” ou “*use case*”. Como descrito, eles devem ser estratificados em elementos de tamanho, ou em algum subtipo de domínio ou perfil de risco. Devem existir trabalhos de remoção de defeitos como “*production bugs*” ou “(*internal*) *bug*”.

Deve existir trabalho de manutenção descrito como “*refactoring*”, “*re-architecting*”, ou simplesmente “*upgrading*”. Softwares que operam sistemas, bancos de dados, plataformas, linguagens, APIs, e arquiteturas de serviço mudam no decorrer do tempo e o código base precisa ser atualizado para endereçar as mudanças.

Uma estratégia adicional para melhorar a previsibilidade é alocar capacidade total de *WIP* para tipos específicos. Por exemplo, com a minha equipe de manutenção na Corbis, eram permitidos apenas dois itens *IT Maintenance* a qualquer momento. Isto limitou a capacidade a ser utilizada nas atualizações de API e banco de dados. Esta estratégia é particularmente útil quando os tipos são divididos por tamanho e esforço requeridos, como épicos, *story* e grão de areia. Alocando capacidade específica para cada tipo, a resposta do sistema é mantida e a previsibilidade é maior.

Considere uma equipe com um quadro Kanban no qual há um limite de duas *stories* épicas, oito *stories* regulares, e quatro *stories* grão de areia. Duas épicas estão em progresso. Um *slot* abre na fila para uma *story* regular, mas não há nenhuma no *backlog* pronta para ser iniciada. A equipe pode escolher começar uma *story* épica ou grão de areia, ou se ater ao tipo de alocação e incorrer em algum tempo ocioso.

Se eles começam uma *story* épica, e alguns dias depois uma *story* regular aparece no *backlog*, eles serão incapazes de iniciar uma *story* regular por algum tempo. Isto aumentará o *lead time* para as *stories* regulares.

Começar um grão de areia menor é uma escolha melhor, visto que ela deve acabar antes de outra *story* regular estar pronta para ser iniciada. Neste caso, não há impacto, mas há um benefício de rendimento adicional. No entanto, se eles não tiverem sorte e falharem em completar o item menor antes de uma *story* estar pronta para iniciar, então o *lead time* das *stories* regulares será afetado adversamente, embora de maneira mais leve do que no cenário épico.

Previsibilidade e gerenciamento de riscos devem ser trunfos típicos para oportunidade de aumento do rendimento, visto que os donos do negócio e gerentes seniores valorizam mais previsibilidade do que rendimento. Previsibilidade constrói e mantém confiança, um dos principais valores Ágeis, melhor do que entregar mais com menos confiança.

Mesclas de Classes-de-Serviço

Se considerarmos as classes de serviço descritas no capítulo 11, podemos antecipar como a variabilidade pode ser afetada mesclando os itens. Se uma organização sofre com uma grande quantidade de requisições de expedição, isto tornará tudo aleatório,

aumentando a média de *lead time* e a propagação de variação, reduzindo a previsibilidade de todo o sistema.

Requisições de expedição são essencialmente variações externas e serão descritas na próxima seção.

Se a demanda por outras classes de serviço é relativamente estável, o desempenho do *lead time* para cada classe de serviço deve ser relativamente confiável. A média e a propagação da variação devem ser mensuráveis e devem permanecer constantes de alguma maneira. Isto fornece previsibilidade. Você pode alcançar isto se o *backlog* é suficientemente grande e cheio de uma grande quantidade mesclada de cada classe. Aloque um limite de *WIP* para cada classe de serviço. Isto possibilitará que a média e a propagação da variabilidade se estabeleçam e o sistema será previsível.

Se a demanda é variável – por exemplo, se há apenas poucos itens de data de entrega fixa e eles tendem a ser sazonais, alguma ação deve ser tomada para dar forma e controlar a demanda: devem ser instituídas mudanças na alocação de limites de *WIP* através dos tipos, sazonalmente, para antecipar mudanças na demanda, ou, alternativamente, mudanças nas políticas para puxar associadas a um conjunto de classes de serviço em cascata devem ser feitas sazonalmente para lidar com flutuações na demanda.

Considere uma equipe com um limite de *WIP* de 20, alocados como 4 itens de data fixa, 10 itens de classe padrão, e 6 itens de classe intangível. Você pode ter uma política onde estes limites devem ser estritamente seguidos, ou, você pode afrouxar a regra e permitir que um item padrão ou intangível preencha um *slot* de um item de data fixa quando há demanda sazonal insuficiente para itens de data fixa. Estas políticas podem ser trocadas em épocas diferentes do ano para melhorar o resultado econômico global e garantir que o sistema permaneça relativamente previsível.

Fluxo Irregular

Fluxo irregular de trabalho pode ser causado por fontes de variação externa e interna. Cada item único puxado pelo sistema kanban será diferente: diferentes em natureza em algum grau, diferentes no tamanho, complexidade, perfil de risco e esforço requeridos. A aleatoriedade natural disto causará altos e baixos no fluxo. Um sistema kanban naturalmente lida com isto à medida que os limites de *WIP* são reforçados. No entanto, a variabilidade maior de outras fontes, como tamanho do item de trabalho, padrões de demanda, mescla de tipos, mescla de classes de serviço, e fontes externas, requerem um *buffering* adequado para absorver os altos e baixos no fluxo através do sistema. *Buffers* adicionais podem ser necessários – e os limites de *WIP* deverão ser maiores – quando há mais variabilidade no sistema. Limites de *WIP* maiores resultarão em *lead times*

maiores, mas o fluxo mais suave deve reduzir a variabilidade. Portanto, aumentar o limite de *WIP* para suavizar o fluxo aumentará o *lead time* médio e reduzirá o intervalo da variabilidade do *lead time*. Este é geralmente o resultado mais desejado dos gerentes, proprietários, e usualmente clientes, valorizar previsibilidade em relação a uma chance aleatória de um *lead time* menor ou rendimento maior.

Rework

Rework^{*}, devido a defeitos internos sendo corrigidos antes do *release* ou defeitos em produção deslocando novos trabalhos de valor agregado ao cliente, afetam a variabilidade. Se uma taxa de defeito é conhecida, medida regularmente, e razoavelmente constante, o sistema pode ser projetado para lidar com isso graciosamente. Tal sistema será economicamente ineficiente, mas ele deve ser confiável. O que causa uma falta de previsibilidade é quando a taxa de defeito não é prevista corretamente. *Rework* devido a defeitos não planejados aumenta o *lead time*, tende a aumentar a propagação da variação, e reduz enormemente o rendimento. Parece ser muito difícil planejar uma taxa de defeito específica, por exemplo, oito defeitos por *user story*, muito menos saber ou ser capaz de prever seu tamanho ou complexidade. A melhor estratégia para redução da variabilidade devido a defeitos é perseguir incansavelmente alta qualidade com uma quantidade muito baixa de defeitos.

Fazer mudanças no processo de ciclo de vida do desenvolvimento de software pode afetar muito as taxas de defeito. Uso de revisão em pares, programação em pares, testes unitários, *framework* de testes automáticos, integração contínua (ou muito frequente), *batch* de tamanho pequeno, arquiteturas definidas corretamente, e projeto de código bem elaborado, fracamente acoplado e de alta coesão reduzirá enormemente os defeitos. Mudanças que afetam diretamente as taxas de defeitos e indiretamente melhoram a previsibilidade do sistema estão sob o controle direto da equipe e gerência local.

Fontes Externas de Variabilidade

Fontes externas de variabilidade vêm de lugares que não são diretamente controlados pelo processo de desenvolvimento de software ou método de gerência de projeto. Algumas destas virão de partes do negócio ou cadeia de valor, como fornecedores ou clientes. Outras fontes externas incluem elementos do mundo físico que não podem ser

* N. de T.: Termo não traduzido devido à sua utilização mais ampla no idioma original (Inglês) pela comunidade de TI no Brasil. Tradução: retrabalho.

facilmente antecipados, previstos, ou controlados – por exemplo, parte do equipamento falhar ou condições de tempo adversas.

Ambiguidade de Requisitos

Requisitos mal escritos, planos de negócio mal definidos, e falta de planejamento estratégico, visão, ou outra informação de definição do contexto pode significar que um membro da equipe é incapaz de tomar uma decisão e, portanto, incapaz de completar uma parte do trabalho. Um item de trabalho se torna bloqueado devido a sua inabilidade de tomar uma decisão; é necessária nova informação para esclarecer a situação para que então o membro da equipe possa tomar uma boa decisão, permitindo que o trabalho-em-progresso flua em direção a sua conclusão.

Para reduzir o impacto desses bloqueios, a equipe e o gerente direto precisam implementar um processo de gerenciamento e resolução de problemas efetivo, como descrito no capítulo 20.

À medida que a equipe e a organização amadurecem, pode ser possível discutir a causa raiz da falha e eliminá-la. Problemas de bloqueio devido à ambiguidade de requisitos podem ser resolvidos influenciando diretamente o processo de análise usado para desenvolver os requisitos, e melhorando o nível de capacidade e habilidade de quem os define. Medições como estas requerem tipicamente a colaboração de outros departamentos e gerentes e uma vontade por parte da empresa de melhorar.

Isto foi alcançado em 2007 na Corbis através de um processo gradual. Primeiro, o sistema kanban foi implementado, incluindo um quadro visual, um sistema de acompanhamento eletrônico, e a transparência que veio com isso. A empresa se tornou cada vez mais envolvida e interessada na atividade de desenvolvimento de software e em monitorar o desempenho do processo. Foi gerado um relatório apresentando o número de problemas abertos, o número de itens de trabalho bloqueados, e o tempo médio de resolução. (Veja Figura 12.6, relatório de Problemas e Itens de Trabalho Bloqueados, na página 144.)

Quando um requisito fazia todo o seu caminho até o teste de aceitação antes de ser rejeitado pela empresa como algo que ela não precisava, a equipe reagia criando uma lixeira no quadro e colocando o ticket nela, como apresentado na Figura 19.1. A gerência então solicitava um conjunto pequeno



Figura 19.1 Quadro com uma lixeira

Rejected and Canceled Work Items									
Report generated 4/27/2007 4:14:05 PM by CONTINUUM DavidA; Last Warehouse update: 4/27/2007 3:31 PM									
List Bugs, CRs and PDUs either Rejected or Canceled									
ID	Work Item Type	Title	Dept	GTM Related	Business Priority	Submitted Date	Approved Date	Closed Date	Reason
2458	Bug	A iacinia leo justo vitae massa			1-Expedited	4/5/2007		4/5/2007	Overtaken by Events
2470	CR	Quisque vitae lacus sodales urna	Creative Services	Not Related to GTM		4/5/2007		4/5/2007	Overtaken by Events
1463	CR	Donec posuere malesuada sodales	Media Services	Not Related to GTM	2-High	11/26/2006		4/12/2007	Released
1443	CR	Pellentesque a. Duis et felis	Customer Experience	Not Related to GTM	2-High	11/26/2006		4/12/2007	Released
2703	CR	Semper turpis facilisis quis	HR	Not Related to GTM		4/26/2007		4/26/2007	Canceled

Figura 19.2 Relatório de Trabalho Rejeitado e Cancelado mostrando os itens abandonados do mês anterior

de relatórios que apresentava os trabalhos que entravam no sistema, mas falhavam em fazer todo o caminho por ele (Figura 19.2).

A combinação de transparência, relatórios, e sensibilização em relação ao impacto e o custo de requisitos ruins resultaram na mudança voluntária do comportamento da empresa. O relatório de desperdício que apresentava os efeitos de requisitos ruins mostrou inicialmente cinco dos dez itens mensais. No quinto mês ele estava vazio. A empresa compreendeu que tomando mais cuidado poderia evitar desperdício de capacidade. Eles colaboravam voluntariamente para melhorar o resultado do sistema. O efeito disso foi a eliminação da causa raiz de variações de causa atribuída a partir de requisitos mal escritos ou informação de contexto mal definida.

Embora a equipe de desenvolvimento de software tenha tomado ações para fornecer uma transparência maior e sensibilização, estas ações não afetaram diretamente o processo de desenvolvimento de requisitos. O processo de gerenciamento e resolução de problemas meramente mitigou o impacto dos problemas bloqueadores levantando sensibilização e reduzindo o tempo de resolução. O resultado foi um impacto menor na média do *lead time* e na propagação da variação. Os efeitos de transparência e reportagem eventualmente resultaram numa mudança externa do processo que eliminou a causa raiz do problema.

Esta é uma evidência de que ações podem ser tomadas localmente e afetarão indiretamente variações de causa atribuída.

Requisições de Expedição

Requisições de expedição acontecem devido a eventos externos, como um pedido inesperado do cliente, ou devido a alguma quebra no processo interno da empresa, por exemplo, uma falta de comunicação que resulta numa descoberta tardia de algum requisito importante. Requisições de expedição são, por definição, variações de causa atribuída, visto que o motivo para a requisição é sempre conhecido e, portanto sempre “atribuível”.

Expedição é conhecida na indústria de engenharia por ser ruim. Ela afeta a previsibilidade de outras requisições. Ela aumenta o *lead time* médio e a propagação da variabilidade e reduz o rendimento. Evidência coletada na Corbis durante 2007 demonstrou que esse resultado da engenharia industrial é verdadeiro para o processo de desenvolvimento de software: Expedir é indesejável mesmo se está sendo feito para gerar valor.

A necessidade para expedir pode ser reduzida. Aumentar a capacidade de folga através de melhorias no rendimento, automação, ou aumento de recursos melhorará a habilidade de resposta. *Lead times* menores, transparência maior, e melhoria na maturidade organizacional reduzirão a necessidade de expedir. Boas equipes que adotam a abordagem Kanban têm exibido uma demanda muito pequena para requisições de expedição. De fato, durante o ano de 2007 na Corbis houve apenas cinco requisições desse tipo no total.

Podemos esperar que tanto quanto requisitos ruins, como transparência do processo e informação de boa qualidade relacionada a rendimento, *lead time*, e desempenho de entrega na data influenciarão o comportamento no início da cadeia de valor. Espera-se que a demanda seja moldada efetivamente e entendida cedo o bastante de maneira que possa ser manipulada como uma classe regular em vez de uma requisição de expedição.

Um método para provocar esta mudança é concordar em limitar o número de requisições de expedição que serão processadas a qualquer momento. Na Corbis esse limite era um. Negando habilidade ao negócio de expedir qualquer coisa que eles queiram, você força as pessoas do início da cadeia de valor, como pessoas de marketing e vendas, a explorar oportunidades mais cedo e avaliá-las efetivamente. Se as pessoas de vendas são pagas por comissão e medidas pela receita gerada, falha para expedir alguma coisa irá afetá-las. Se a falha é devido ao limite de *WIP* para requisições de expedição ter sido alcançado, então elas tentarão fortemente coletar no futuro informação suficiente para colocar uma requisição em tempo de atender uma classe de serviço regular.

Fluxo Irregular

Fluxo irregular de trabalho pode ser causado por variações de causa aleatória ou atribuível mencionadas acima. Todas as variações de causa atribuível que afetam o fluxo resultam em trabalho bloqueado. Problemas como ambiguidade de requisitos, e ambiente – ou recursos especialistas de disponibilidade compartilhada são motivos comuns para trabalho bloqueado por causa atribuível.

Itens de trabalho bloqueados requerem uma forte disciplina e capacidade com o gerenciamento e resolução de problemas, como descrito no capítulo 20. Há duas abordagens para lidar com problemas de itens de trabalho bloqueados.

A primeira abordagem facilitará o fluxo, mas à custa do *lead time* e possivelmente qualidade. Você pode melhorar o fluxo tendo um limite geral maior de *WIP* – alcançado

através de *buffering* explícito ou usando uma política com menos restrição no *WIP*, por exemplo, 3 atividades por pessoa, em média, em vez de 1.2 atividades por pessoa. O limite de *WIP* maior significa que enquanto alguma coisa está bloqueada, os membros da equipe podem trabalhar em outros itens. Eu recomendo essa abordagem para organizações imaturas. Os efeitos devem ser simples e não dramáticas. Os *lead times* serão maiores, mas isso não deve ser um problema em muitos domínios. A propagação da variação deverá ser maior, e então o lead time será menos previsível; no entanto, eles deverão ser mais previsíveis através do uso do sistema kanban do que eles eram antes. A maior desvantagem em usar limites de *WIP* maiores é que há uma tensão menor (ou nenhuma) para provocar discussão e implementação de melhorias. Não há consequentemente nenhuma pressão para melhoria: O efeito catalítico de kanban é perdido.

A segunda abordagem é perseguir gerenciamento e resolução de problemas imiplacavelmente e, à medida que a equipe amadurecer, se mover em direção a análise e eliminação da causa raiz com melhorias específicas projetadas para prevenir variações de causa atribuível no futuro. Nesta abordagem você deixa os limites de *WIP*, tamanhos de *buffer*, e políticas de trabalho razoavelmente apertadas, e você faz com que o trabalho pare quando as atividades ficam bloqueadas. O tempo ocioso destas pessoas atribuídas a trabalhos bloqueados levanta a conscientização do problema bloqueado. Isto pode causar um comportamento colaborativo de resolução do problema, o qual encorajará os membros ociosos da equipe a pensar sobre a causa raiz e possíveis mudanças no processo que reduzirão ou eliminarão a possibilidade de recorrência. Mantendo os limites de *WIP* apertados e perseguindo o gerenciamento e resolução de problemas como uma capacidade tem criado uma cultura de melhoria contínua. Vi isso pela primeira vez na Corbis em 2007, mas houve muitos outros relatórios que surgiram em 2009 em empresas como *Software Engineering Professionals* em Indianapolis, IPC Media, e BBC *WorldWide*, ambas em Londres. Não há evidência suficiente que possa sugerir que Kanban gera uma cultura que é focada na melhoria contínua. Os elementos de processo consistentes nos exemplos parecem ser uma vontade de reforçar políticas de *WIP* apertado, de marcar trabalho bloqueado, de permitir que a linha pare, de incorrer em tempo ocioso, e de perseguir o gerenciamento e resolução de problemas como uma disciplina organizacional. O que resulta disto é um foco na análise e eliminação de causa raiz e a introdução gradual de melhorias que reduzem variações de causa atribuível e dão início a uma cultura maior de melhoria contínua.

Disponibilidade do Ambiente

Disponibilidade do ambiente é um problema típico de causa atribuível que pode ter um efeito significativo no fluxo, rendimento, e previsibilidade. Falhas no ambiente

frequentemente causam uma falha em todo o fluxo de trabalho. Um sistema kanban trará visibilidade para o problema e seu impacto. O tempo ocioso incorrido pela fortificação do limite de WIP encoraja colaboração para resolver a falha. Quando as pessoas do início da cadeia de valor, como desenvolvedores e testadores, ajudam pessoas de manutenção de sistemas a recuperar um ambiente, este comportamento é referido frequentemente como um movimento em massa (*swarming*). Movimento em massa implica no conceito de que a equipe se move em conjunto para trabalhar num único problema até que ele seja resolvido. A natureza de Kanban encoraja equipes a focarem em lead time, rendimento, e fluxo através da cadeia de valor. Alinhando todos os grupos no início e final da cadeia de valor com os mesmos objetivos, há um incentivo para que o comportamento em massa surja. Todos ganham quando pessoas ociosas se voluntariam a colaborar para resolver um problema que as afeta mesmo se ele não está na sua área de trabalho imediata ou área de responsabilidade.

Outros Fatores de Mercado

Em Outubro de 2008, seguindo o colapso de Lehman Brothers e uma série de eventos traumáticos no setor financeiro, bancos e firmas de investimento de centros financeiros líderes como Londres e Nova York começaram a cancelar ou significativamente modificar projetos de TI em desenvolvimento. O motivo era que o mundo deles virou de cabeça para baixo. Eles estavam lutando pela sua sobrevivência. De repente eles precisaram entender melhor sua liquidez – e a do Mercado. Não era mais importante entregar o mais recente produto de *commodity* exótico. O mercado não poderia cuidar menos dos investimentos. No outono de 2008, empresas financeiras estavam interessadas apenas na solvência ou insolvência, dependendo de quanta sorte elas tinham.

Este é um exemplo grave – mas muito real – que mostra como *portfolios* de projetos e requisitos para projetos em progresso podem mudar dramaticamente. Reagir a esse tipo de mudanças tende a distrair as equipes e resultar em significantes quedas no rendimento, aumentos dramáticos no lead time, queda (frequente) da qualidade, e perda de previsibilidade à medida que a equipe recupera-se da aleatoriedade que a flutuação no mercado causou aos trabalhos internos do projeto.

Tais eventos claramente são variações de causa atribuível. Eles precisam ser acomodados usando estratégias e táticas de gerenciamento de risco. Há um corpo de conhecimento considerável em variação de causa atribuível, ou risco dirigido a eventos. Construir uma capacidade de gerenciamento de riscos forte como parte de um objetivo geral de melhoria da maturidade organizacional melhorará a previsibilidade da unidade de engenharia de software se ela está usando Kanban ou não. No entanto, sistemas

kanban exibem alta previsibilidade quando o risco é bem gerenciado. Isto constrói confiança no sistema.

Sistemas Kanban têm um número de outros elementos que ajudam no gerenciamento de riscos. O limite de *WIP* reduz risco, visto que apenas uma pequena fração do trabalho está em progresso em qualquer momento. Alocação de limites de *WIP* pelos tipos de itens de trabalho e classes de serviço ajuda a gerenciar riscos e absorver variações de causa atribuível. Outras estratégias estão surgindo, e é provável que um livro subsequente também surja, detalhando métodos avançados para melhorar Kanban e táticas de gerenciamento de riscos

Tenho apresentado material em gerenciamento de riscos com sistemas kanban que surgiram a partir do uso de Kanban em conferências durante o ano de 2009, que está disponível online²⁵.

Dificuldades em Agendar Atividades de Coordenação

Outra fonte comum de variação de causa atribuível que causa bloqueio no trabalho e fluxo irregular é o desafio de coordenar equipes externas, *stakeholders*, e recursos. Uma reação frequente aos desafios de coordenação é agendar reuniões com uma cadência regular. Em algumas instâncias isso é muito eficiente. No entanto, não é sempre possível.

O fluxo pode ser interrompido por uma restrição governamental ou regulatória que requer uma auditoria ou um *signoff*. As pessoas necessárias para realizar esta função podem não estar disponíveis instantaneamente ou pode ser difícil fazer um agendamento.

Em primeira instância, variação de causa atribuível desta natureza deve ser endereçada conscientizando e dirigindo a atenção a ela com visibilidade e transparéncia. Marcando itens como bloqueados e levantando visibilidade na fonte do bloqueio, a gerência, equipe, e *stakeholders* da cadeia de valor ficarão conscientes do impacto dos desafios de coordenação.

Esta consciência deve liderar algumas mudanças comportamentais que melhoraram a situação.

Uma tática pode ser examinar as regras governamentais e regulatórias e decidir se tudo precisa ser avaliado, aprovado, auditado, ou examinado. Assumindo que algum perfil de risco permite que o trabalho seja bifurcado em duas categorias, as que precisam e as que não precisam que uma reunião aconteça, tipos de item de trabalho ou classes de serviço podem ser usados para separar o trabalho. Você pode então usar alocação de limites de *WIP* para os tipos ou classes para garantir a suavidade do fluxo.

Takeaways

- ❖ O estudo de variação no processo industrial iniciou na década de 1920 com Walter Shewhart e evoluiu através do trabalho de W. Edwards Deming, Joseph Duran, e David Chambers da metade do século em diante.
- ❖ Um estudo de variação e seu método de análise estatística estão no coração do *Toyota Production System* (e, portanto, *Lean*) e do método *Six Sigma* para melhoria de processo.
- ❖ Os trabalhos de W. Edwards Deming e Joseph Duran foram a maior inspiração para o trabalho do *Software Engineering Institute* na *Carnegie Mellon University* e do *Capability Maturity Model* (agora *Capability Maturity Model Integration*, ou *CMMI*).
- ❖ Shewhart dividiu fontes de variação em duas categorias: aquelas internas ao processo ou sistema e aquelas externas ao processo ou sistema.
- ❖ Variações internas são referidas como variações de causa aleatória.
- ❖ Variações externas são referidas como variações de causa atribuível.
- ❖ Podem existir várias fontes de variação de causa aleatória na cadeia de valor do ciclo de vida de desenvolvimento de software. Exemplos típicos incluem tamanho do item de trabalho, tipo, classe de serviço, fluxo irregular, e *rework*.
- ❖ Há possivelmente uma lista infinita de fontes de variação de causa atribuível. Exemplos típicos incluem ambiguidade de requisitos, requisições de expedição, disponibilidade do ambiente, fluxo irregular, fatores relacionados às pessoas, e desafios em agendar atividades de coordenação ou *overhead*.
- ❖ Variação de causa aleatória pode ser controlada através do uso de políticas que definem o ciclo de vida de desenvolvimento de software e processo de gerência de projeto em uso.
- ❖ Variações de causa atribuível podem ser gerenciadas através do uso da capacidade de gerenciamento e resolução de problemas, como também da capacidade de gerenciamento de riscos, e elas podem ser reduzidas ou eliminadas através da capacidade de análise e eliminação de causa raiz.
- ❖ Sistemas Kanban produzem melhores resultados econômicos quando acoplados a uma capacidade sólida e dirigida a eventos de gerenciamento de riscos.

- ❖ Kanban também oferece caminhos adicionais para gerenciar riscos como uma alocação de *WIP* através das classes de serviço e tipos de itens de trabalho e o uso de um perfil de risco para categorizar trabalhos em tipos ou classes e processá-los de acordo com a categorização.
- ❖ Mais trabalhos em estratégias avançadas de gerenciamento de riscos e táticas com Kanban estão em progresso , e será um tópico de um futuro livro.

❖ CAPÍTULO 20 ❖

Gerenciamento de Problemas e Políticas de Escalação

Quando um trabalho no nosso sistema kanban torna-se bloqueado por qualquer motivo, tornou-se convenção indicar isso na parede de cartões anexando uma nota adesiva rosa ao cartão, indicando o motivo do bloqueio. Em sistemas eletrônicos, podem existir outras maneiras para indicar que um item de trabalho está bloqueado, como mostrá-lo com uma borda vermelha. Preferencialmente, as *features* do sistema eletrônico devem permitir que o motivo para o bloqueio seja acompanhado separadamente, ou o problema que está bloqueando o trabalho deve ser acompanhado como um item de trabalho de primeira classe ligado ao item de valor para o usuário que está dependendo da resolução do problema.

Durante a escrita deste livro notei que algumas pessoas que tentam usar Kanban pela primeira vez referem-se a estes itens bloqueados como gargalos. Isto está errado. Um item bloqueado deve estar entupindo o *pipe* e restringindo o fluxo, mas não é um gargalo, como o descrito no capítulo 17. Ele também não é um recurso de capacidade limitada ou um recurso de disponibilidade não instantânea. Da mesma maneira que uma rolha em um gargalo não é um gargalo. Se você deseja restaurar o fluxo no gargalo você simplesmente retira a rolha.

É perigoso pensar em itens de trabalho bloqueados como gargalos porque isso leva a um tipo de pensamento errado para resolver o problema. Itens de trabalho bloqueados devem ser tratados como

causa especial de variações em vez de gargalos. O que é similar é o resultado esperado. Em ambos os casos – um gargalo ou item de trabalho bloqueado – queremos resolver o problema para melhorar o fluxo.

Itens de trabalho bloqueados requerem que uma organização desenvolva uma capacidade para gerenciamento e resolução de problemas para restaurar o fluxo o mais rápido possível, como também uma capacidade de análise e resolução de causa raiz para prevenir recorrência. A segunda capacidade foi discutida no capítulo 19 como remoção de variações de causa especial. A primeira é discutida aqui.

Gerenciando Problemas

Não é suficiente simplesmente marcar e acompanhar um item de trabalho bloqueado. Muitas ferramentas de desenvolvimento de software Ágeis iniciais permitiam apenas esta capacidade. Enquanto é útil ter informação sobre um item, uma *story*, uma *feature*, ou *use case* bloqueado, minha observação de equipes ao redor do mundo tem sido que ter conhecimento que alguma coisa está bloqueada não leva ao desenvolvimento de uma capacidade forte de torná-la não bloqueada.

É essencial acompanhar o motivo do bloqueio e tratá-lo como um item de primeira classe, embora seja um item de trabalho de carga defeituosa. Um tipo de item de trabalho especial, Problema, é colocado a parte para esse propósito. Problemas são acompanhados usando-se *tickets* na cor rosa (Figura 20.1). Eles devem ter um número de acompanhamento, e um membro da equipe – usualmente o gerente de projeto – deve ser atribuído para garantir a resolução.

Quando um membro da equipe trabalhando num item de valor para o cliente é incapaz de proceder, ele ou ela devem marcar o item como bloqueado, anexar um ticket rosa descrevendo o motivo do bloqueio, e criar um item de trabalho Problema na ferramenta de acompanhamento eletrônica. O problema deve ser ligado ao seu item de trabalho original (veja Figura 20.1). Alguns exemplos são: ambiguidade nos requisitos, e uma pessoa com conhecimento não está disponível para instantaneamente resolver a ambiguidade; uma configuração de ambiente é requerida e o engenheiro para realizar esta tarefa está atualmente indisponível; ou é requerido um especialista para trabalhar num item e a pessoa está indisponível devido a férias, doença, ou outro tempo fora do trabalho.

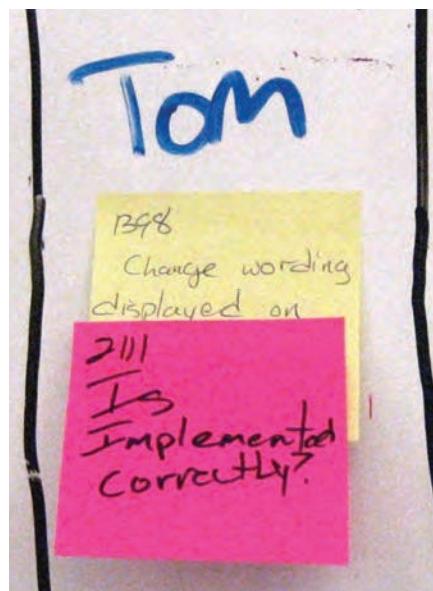


Figura 20.1 Item de impedimento (rosa) anexado ao item de trabalho Solicitação de Mudança diretamente afetado

Como discutido no capítulo 7, manter o fluxo deve ser o principal foco de discussão de uma reunião *standup* diária. Portanto, a reunião deve focar em discutir bloqueios e progresso direcionado a resolução de problemas. A reunião deve focar fortemente nos *tickets* rosa. Questões devem ser levantadas sobre quem está trabalhando em resolver o problema e o *status* do progresso da resolução. O problema precisa ser escalado? Se sim, para quem?

Membros ociosos da equipe devem ser encorajados a acompanhar os problemas e geralmente mergulhar nos problemas e ajudar da maneira que eles possam a resolvê-los e restaurar o fluxo do sistema. Uma equipe com uma capacidade forte de auto-organização tende a fazer isso naturalmente. Membros da equipe serão voluntários na ajuda para a resolução dos problemas. Onde esta capacidade de organização ainda vai emergir, o gerente de projeto pode precisar atribuir membros da equipe para trabalhar na resolução de problemas.

Problemas devem ser acompanhados como todos os itens de trabalho. O acompanhamento inclui uma data inicial e uma data final e um *link* para todos os itens de trabalho de valor para o cliente afetados. Note que um único item pode estar bloqueando mais de um item de valor para o cliente. Está é outra boa razão para acompanhar problemas independentemente dos itens de trabalho e para ter Problema como um item de trabalho distinto. Quando escolher uma ferramenta eletrônica para acompanhar seu sistema kanban, esteja certo de escolher uma que dá suporte a acompanhamento de problemas como um item de primeira classe ou uma ferramenta que é suficientemente customizável de maneira que você possa criar um item de trabalho chamado Problema e designar que ele será apresentado usando cartões rosa (ou vermelho).

Escalando Problemas

Quando uma equipe é incapaz de resolver um problema por ela própria, ou uma parte externa é requerida para resolver um problema e está indisponível ou sem dar resposta, o problema deve ser escalado para um gerente mais sênior ou outro departamento.

É importante para a organização desenvolver uma forte capacidade de escalação de problema. Sem isso, manter e restaurar o fluxo após um bloqueio pode ser problemático.

A base para uma boa capacidade de escalação é um política ou processo de escalação documentada, O capítulo 15 discute o poder das políticas organizacionais desenvolvidas de maneira colaborativa. Políticas de escalação devem ser desenvolvidas de maneira colaborativa e os departamentos envolvidos na cadeia de valor devem entrar em consenso. As políticas de escalação devem ser largamente conhecidas e compreendidas, e

um documento (ou *web site*) descrevendo-as deve estar prontamente disponível para todos os membros da equipe. Não deve existir ambiguidade sobre como e quando escalar um problema. Tomando-se um tempo para definir caminhos de escalação e escrever as políticas que os circundam, a equipe sabe para onde enviar os problemas para resolução. Isto economiza tempo da equipe imaginar para quem um problema deve ser escalado e atribui expectativas para aqueles indivíduos mais seniores de que eles fazem parte do sistema. Gerentes seniores precisam ter responsabilidade em resolver problemas. Isto ajudará a manter o fluxo e por fim a minimizar o custo de atraso (ou melhorar o *payoff* a partir de maior velocidade de entrega).

Acompanhando e Reportando Problemas

Como declarado anteriormente, problemas devem ser acompanhados como itens de trabalho de primeira classe com o seu próprio item de trabalho. Há uma evolução para a convenção do uso de cartões rosa ou vermelhos, ou notas adesivas para a visualização de problemas (Figura 20.2). Uma data inicial, uma data final, um membro da equipe atribuído, uma descrição do problema, e *links* para os itens de valor bloqueados para o cliente são os requisitos mínimos para um sistema de acompanhamento de problemas (Figura 20.3). Também pode ser útil acompanhar alguns históricos de esforço gasto

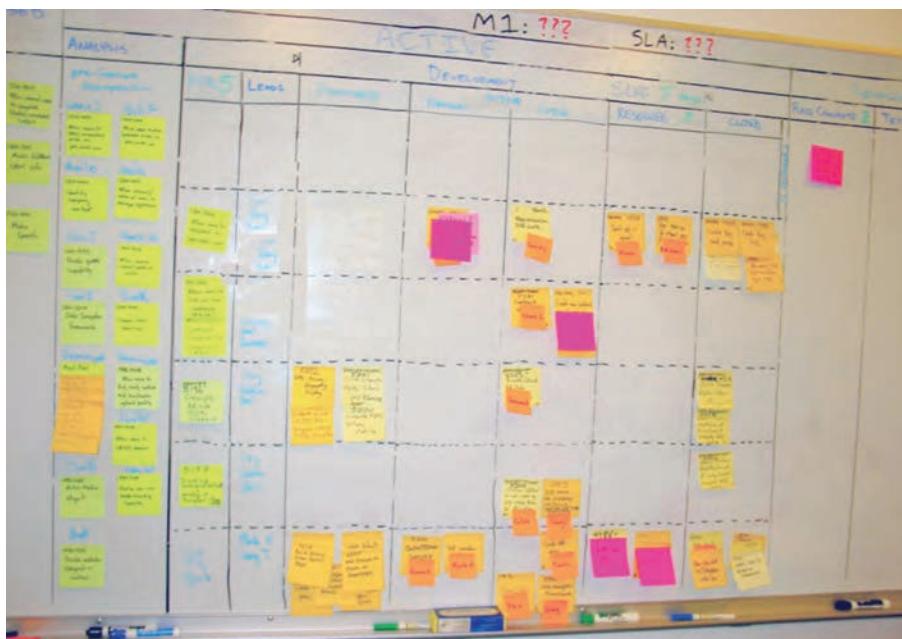


Figura 20.2 Quadro mostrando muitos impedimentos afetando múltiplas features

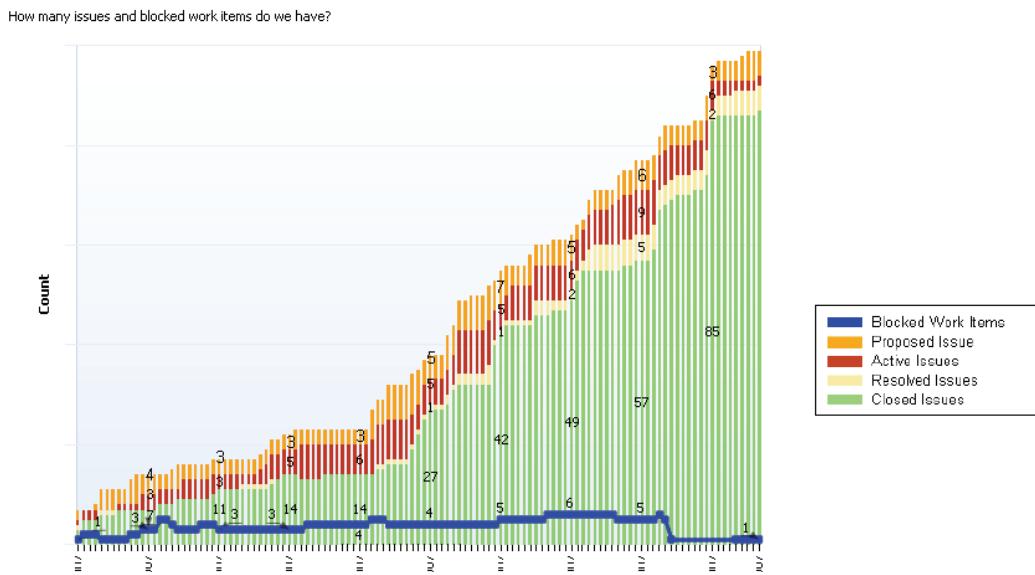


Figura 20.3 Gráfico de problemas CFD sobreposto com itens de trabalho Bloqueados

para resolução, um histórico de indivíduos atribuídos, uma indicação do caminho de escalação, um tempo estimado para resolução, uma avaliação de impacto, e sugestões de correções de causa raiz para futuras prevenções.

Mesmo que *tickets* rosa na parede de cartões forneçam uma visualização forte de quantos itens estão atualmente bloqueados, também é útil acompanhar e reportar problemas de outras maneiras. Um diagrama de fluxo cumulativo de problemas e itens de trabalho bloqueados fornece um indicador visual forte da capacidade da organização no gerenciamento e resolução de problemas. A tendência de itens de trabalho bloqueados no decorrer do tempo indica se a capacidade de análise e resolução de causa raiz – oportunidades de melhoria para eliminar variações de causa atribuível – está desenvolvendo. Um relatório tabular de problemas atuais, indivíduos atribuídos, *status*, data de resolução prevista, itens de trabalho afetados, e o potencial impacto pode também se provar útil no dia a dia do gerenciamento de grandes projetos.

Estes relatórios devem ser apresentados em cada revisão operacional e deve ser reservado um tempo para se discutir a maturidade da capacidade organizacional no gerenciamento e resolução de problemas e na análise e resolução de causa raiz. A organização deve estar consciente do impacto de carga defeituosa de problemas bloqueando trabalho. Isto possibilitará decisões objetivas sobre oportunidades de melhoria e benefícios prováveis de investimentos em correções de causa raiz para prevenir variações de causa especial.

Takeaways

- ❖ Sistemas Kanban devem ter um tipo de item de trabalho, Problema, usado para acompanhar problemas que causam bloqueio de trabalho de valor para o cliente.
- ❖ Tornou-se convenção a utilização de notas adesivas rosa (ou vermelha) na parede de cartões para a visualização de problemas de bloqueio.
- ❖ *Tickets* rosa de problemas são anexados aos itens que estão bloqueados.
- ❖ Para manter o fluxo é essencial uma forte capacidade para gerenciamento e resolução de problemas.
- ❖ Itens de trabalho bloqueados e problemas não são gargalos. Eles devem ser gerenciados como variações de causa especial em vez de recursos de capacidade limitada ou recursos de disponibilidade não instantânea.
- ❖ Gerenciamento de problemas deve ser um foco forte nas reuniões *standup* diárias.
- ❖ Para gerenciamento de problemas é essencial uma forte capacidade de escalação de problemas.
- ❖ Políticas de escalação devem ser claramente definidas e documentadas e todos os membros da equipe devem ter consciência delas.
- ❖ Políticas de escalação funcionam melhor quando elas são acordadas colaborativamente por todos os departamentos envolvidos na cadeia de valor.
- ❖ Problemas podem ser acompanhados eletronicamente.
- ❖ Relatórios baseados em dados eletrônicos facilitarão o dia a dia do gerenciamento e resolução em grandes projetos.
- ❖ Use um diagrama de Problemas e Itens de Trabalho Bloqueados para visualizar o desenvolvimento da capacidade para gerenciamento e resolução de problema e análise e resolução de causa raiz.

❖ Notas ❖

1. Anderson, David J. *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results*. Upper Saddle River, NJ: Prentice Hall, 2003.
2. Beck, Kent. *Extreme Programming Explained: Embrace Change*. Boston: Addison Wesley, 2000.
3. Beck, Kent et al., “The Principles Behind the Agile Manifesto.” www.agilemanifesto.org/principles.html.
4. Goldratt, Eliyahu M. *What is this thing called The Theory of Constraints and How should it be implemented?* Great Barrington, MA: North River Press, 1999.
5. Anderson, David J., and Dragos Dumitriu, “From Worst to Best in 9 Months: Implementing a Drum-Buffer-Rope Solution in Microsoft’s IT Department,” Proceedings of the TOCICO World Conference, Barcelona, November 2005.
6. Belshee, Arlo. “Naked Planning, Promiscuous Pairing and Other Unmentionables.” 2008 Agile Conference, podcast. http://agiletoolkit.libsyn.com/index.php?post_id=400364.
7. Hiranabe, Kenji. “Visualizing Agile Projects Using Kanban Boards.” *InfoQ*, August 27, 2007; www.infoq.com/articles/agile-kanban-boards.

8. Hiranabe, Kenji, "Kanban Applied to Software Development: From Agile to *Lean*," *InfoQ*, January 14, 2008. www.infoq.com/articles/hiranabe-lean-agile-kanban.
9. Augustine, Sanjiv. *Managing Agile Projects*. Upper Saddle River, NJ: Prentice Hall, 2005.
10. Highsmith, Jim. *Agile Software Development Ecosystems*. Boston: Addison Wesley, 2002.
11. The Nokia Test is attributed in origin to Bas Vodde, described here by Jeff Sutherland, who has adopted and updated it. <http://jeffsutherland.com/scrum/2008/08/nokia-test-where-did-it-come-from.html>.
12. Beck et al., "The Principles Behind the Manifesto." www.agilemanifesto.org/principles.html.
13. Jones, Capers. *Software Assessment Benchmarks and Best Practices*. Boston: Addison Wesley, 2000.
14. Ambler, Scott. *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*. Hoboken, N.J.: Wiley, 2002.
15. Chrissis, Mary Beth, Mike Konrad, and Sandy Shrum. *CMMI: Guidelines for Process Integration and Product Improvement*, 2d ed. Boston: Addison Wesley, 2006.
16. Sutherland, Jeff, Carsten Ruseng Jakobsen, and Kent Johnson. "Scrum and CMMI Level 5: A Magic Potion for Code Warriors." Proceedings of the Agile Conference, Agile Alliance/IEEE, 2007.
Jakobsen, Carsten Ruseng and Jeff Sutherland. "Mature Scrum at Systematic." *Methods & Tools*, Fall 2009. www.methodsandtools.com/archive/archive.php?id=95.
17. Larman, Craig and Bas Vodde. *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum*. Boston: Addison Wesley, 2008.
18. Willeke, Eric, with David J. Anderson and Eric Landes (editors) *Proceedings of the Lean & Kanban 2009 Conference*. Bloomington, IN: Wordclay, 2009.
19. Beck et al, Principles Behind the Agile Manifesto, 2001, www.agilemanifesto.org/principles.html

20. Anderson, David J. "New Approaches to Risk Management." Agile 2009, Chicago, Illinois; www.agilemanagement.net/Articles/Papers/Agile2009-NewApproachesto.html.

❖ Agradecimentos ❖

Cada livro publicado representa um esforço de gerenciamento de projeto e de coordenação significantes e envolve toda uma equipe de pessoas, da qual o autor é realmente apenas uma pequena parte. Este livro não teria acontecido sem os esforços de valor inestimável de Janice Linden-Reed e Vicki Rowland. Eu gostaria de agradecê-las pela incrível paciência e perseverança em ter este manuscrito pronto para publicação dentro de um prazo apertado (com um custo de atraso alto).

Eu gostaria de agradecer a Donald Reinertsen por me estimular a usar Kanban e por me fornecer um fórum para falar sobre ele publicamente. Eu também gostaria de agradecê-lo por suas palavras gentis no Prefácio e seus esforços em construir uma comunidade de vida longa e próspera através da formação do Lean Software & Systems Consortium.

Eu gostaria de agradecer a Karl Scotland, Joe Arnold, e Aaron Sanders, e também a Eric Willeke, Chris Shinkle, Olav Maassen, Chris Matts, e Rob Hathaway. O entusiasmo inicial deles e a adoção de Kanban levaram diretamente à formação de uma comunidade agora próspera e à disseminação viral do método ao redor do mundo. Sem o suporte destas pessoas não haveria demanda para este manuscrito, e Kanban seria um método obscuro usado por algumas companhias no Pacífico Nordeste dos Estados Unidos, em vez de uma nova abordagem excitante usada por equipes em cada continente - de startups de 5 pessoas no Cambodia a companhias de segurança de 300 anos em Netherlands, em grandes companhias de petróleo no Brasil e fornecedores terceirizados na Argentina, como também companhias de mídia em Londres, Los Angeles, e Nova York, e muito mais pelo mundo. A adoção de Kanban é um fenômeno, e não teria acontecido

sem o encontro afortunado de mentes que ocorreu em Agosto de 2007 em Washington, DC, na conferência Agile 2007.

Este livro não seria uma ferramenta tão útil e uma experiência de leitura tão prazerosa sem os comentários inteligentes e feedback construtivo de um grande número de revisores do manuscrito. Eu gostaria de dar atenção especial às contribuições de Daniel Vacanti, Greg Brougham, Christina Skaskiw, Chris Matts, Bruce Mount, Norbert Winklareth, e, novamente, Janice Linden-Reed. Cada um deles produziu uma revisão estratégica e inteligente em uma ou mais versões iniciais do manuscrito que levaram a uma significante reestruturação do conteúdo. O resultado é um livro melhor e mais fácil de ler e entender, e isso será uma ferramenta mais útil à comunidade em longo prazo.

Além disso, há muitos outros membros da comunidade que contribuíram com feedback e edições que foram todas consideradas cuidadosamente à medida que o manuscrito evoluía durante 2009 e 2010. Obrigada: Jim Benson, Matthias Bohlen, Joshua Kerievsky, Chris Simmons, Dennis Stevens, Arne Roock, Matthias Skarin, Bill Barnett, Olav Maassen, Steve Freeman, Derick Bailey, John Heintz, Lilian Nijboer, Si Alhir, Siddharta Govindaraj, Russell Healy, Benjamin Mitchell, David Joyce, Tim Uttormark, Allan Kelly, Eric Willeke, Alan Shalloway, Alisson Vale, Maxwell Keeler, Guilherme Amorim, Reni Elisabeth Pihl Friis, Nis Holst, Karl Scotland, e Robert Hathaway.

Eu gostaria de agradecer a minha incansável gerente de escritório, Mikiko Fujisaki, que mantêm as rodas girando do David J. Anderson & Associates, Incorporated, e sem a qual eu nunca teria encontrado tempo para escrever esse livro.

Meu velho amigo e colega Pujan Roka kindly se ofereceu para desenhar a ilustração da capa. Pujan é um artista talentoso de histórias em quadrinhos, e também tem livros publicados, até o momento dois deles sobre gerenciamento. Saiba mais sobre ele e suas publicações em <http://www.pujanroka.com/>.

A comunidade tem sido generosa na adoção e entusiasmo sobre Kanban, o que resultou em gentis propostas de tradução do livro para idiomas locais. Eu gostaria de agradecer a Jan Piccard de Muller, Andrea Pinto, Eduardo Bobsin, Arne Roock, Masa Maeda, e Hiroki Kondo, que já estão trabalhando fortemente na tradução para os idiomas Francês, Português, Espanhol e Japonês. Tenho certeza que os seus esforços ajudarão a espalhar a adoção de Kanban ao redor do mundo e expandir a comunidade e o entusiasmo em relação ao método nas suas respectivas regiões.

Eu gostaria de agradecer a Nicole Kohari, Chris Hefley, David Joyce, Thomas Blomseth, Jeff Patton, e Steve Reid, todos eles contribuíram com imagens para o livro.

E finalmente eu gostaria de agradecer ao meu bom amigo Dragos Dumitriu, que está agora na Avanade, e à minha equipe na Corbis: Darren Davis, Larry Cohen, Mark Grotte, Dominica Degrandis, Troy Magennis, Stuart Corcoran, e também a Rick

Garber, Corey Ladas, e Diana Kolomiyets. Sem eles, Kanban nunca teria acontecido. Seus esforços em implementá-lo e usá-lo criaram exemplos e histórias a partir das quais todos nós aprendemos e subsequentemente adotamos e adaptamos soluções para situações novas e mais desafiadoras. Sem eles não haveria livro, comunidade, e um crescente número de clientes satisfeitos que estão desfrutando de um software de alta qualidade desenvolvido de maneira regular e rápida, quando e onde eles precisam, com agilidade, em resposta à natural demanda da suas indústrias e usuários.

Nossa jornada Kanban continua e espero que com este livro você seja persuadido a fazer parte dela.

David J. Anderson
Na trilha de evangelização Kanban,
em algum lugar na Europa, Abril 2010

❖ Sobre o Autor ❖

David J. Anderson lidera uma firma de consultoria em gerenciamento focada em melhorar o desempenho de companhias de tecnologia. Ele está na área de desenvolvimento de software há aproximadamente 30 anos e gerenciou equipes em projetos de desenvolvimento de software ágil na Sprint, Motorola, Microsoft, e Corbis. David tem o crédito da primeira implementação de um processo kanban para desenvolvimento de software em 2005. David foi um fundador do movimento Ágil através do seu envolvimento na criação do *Feature Driven Development*. Ele também foi um fundador da Agile Project Leadership Network (APLN), um fundador signatário da Declaration of Interdependence, e um membro fundador do Lean Software and Systems Consortium. Ele modera muitas comunidades online para desenvolvimento lean/ágil. Ele é autor do *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results*. Mais recentemente, David está focado em criar uma sinergia do modelo CMMI para maturidade organizacional com métodos Ágeis e Lean através de projetos com a Microsoft e o SEI. Ele é um co-autor da Nota Técnica do SEI “CMMI and Agile: Why not Embrace Both!” Ele mora em Sequim, Washington, EUA.

Recursos Adicionais

David J. Anderson and Associates

<http://djandersonassociates.com>

The Limited WIP Society

www.limitedwipsociety.org

Kanban Development Yahoo! Group

<http://finance.groups.yahoo.com/group/kanbandev/>

Kanban101

www.kanban101.com

Kanban está se tornando uma maneira popular de visualizar e limitar trabalho-em-progresso em desenvolvimento de software e no trabalho em tecnologia da informação. Equipes ao redor do mundo estão incluindo Kanban aos seus processos existentes para catalisar uma mudança cultural e promover melhor agilidade no negócio.

Este livro responde algumas perguntas:

O que é Kanban?

Porque eu gostaria de usar Kanban?

Como faço para implementar Kanban?

Como reconheço oportunidades de melhoria e o que eu deveria fazer em relação a elas?

David J. Anderson foi pioneiro na técnica Kanban com a Microsoft em 2004 e desde então tem refinado a abordagem. Kanban fornece os insights de David nessa nova e evolucionária abordagem para gerenciamento de mudança. O Método Kanban melhorará a maturidade e a agilidade da sua organização com uma resistência mínima a mudança.

David J. Anderson é um consultor de gestão independente com muitos anos de experiência gerenciando, e liderando equipes de software em algumas das maiores empresas do mundo. Ele é fundador do **Lean Software & Systems Consortium** e co-autor da Nota Técnica "**CMMI and Agile: Why not embrace both!**" do Software Engineering Institute.



"Este livro inaugura a segunda geração dos Metódos Ágeis. Eu prevejo que ele se tornará um clássico instantaneamente."

—Alan Shalloway,
CEO e Consultor Sênior
Net Objectives

"Kanban mudou o meu negócio. Lendo este livro, eu agora entendo porque!"

—Alisson Vale
Fundador, Phidelis

Com a popularização dos métodos ágeis temos muitas perguntas sobre como mudar grandes organizações do caos para um lugar melhor. A abordagem Kanban descrita neste livro revolucionou meu trabalho nesses ambientes, na melhor forma Kaizen."

—Rodrigo Yoshima,
Fundador, Aspercom



BLUE HOLE PRESS
Sequim, Washington
www.blueholepress.com