Métricas Ágeis

Obtenha melhores resultados em sua equipe

Casa do Código

RAPHAEL DONAIRE ALBINO

Table of Contents

Sumário

ISBN

Impresso e PDF: 978-85-5519-276-0

EPUB: 978-85-5519-277-7

MOBI: 978-85-5519-278-4

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

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

Agradecimentos

Vivian e equipe da Casa do Código, obrigado pelo apoio no desenvolvimento do livro. Sem vocês, o sonho de publicar uma obra não seria uma realidade.

Agradeço à minha esposa, Jeniffer, por todo o apoio e incentivo nos momentos mais difíceis desta jornada. Te amo de todas as maneiras.

Mãe Tânia e pai João, vocês são minha maior inspiração no ato de transmitir conhecimento para o próximo. Amo vocês.

Aos meus irmãos, Daniel e João, obrigado pelo apoio e carinho.

Aos amigos da turma de BSI Unesp 2005, até hoje você são capazes de tornarem quaisquer discussões em aprendizado. Todos moram no meu coração.

Fica aqui o meu agradecimento aos amigos que trabalham como consultores na Ptec. Vocês são fonte de conhecimento. Lucas Colucci, você é parte deste livro, obrigado pela atenção nas revisões.

Hugo Baraúna, Guilherme Fré, Wesley Zapellini, Eric Camalionte e Tiago Porangaba, obrigado pelos pitacos quando havia tempo para as revisões.

Deixo o meu agradecimento ao Rodrigo Yoshima, Leonardo Campos, Celso Martins, Alisson Vale e a todos que, de alguma forma, trouxeram conteúdo, discussões e *insights* para que o livro fizesse sentido.

Finalizo agradecendo ao Hugo, Marcelo, George, Eliel, Valim e Ozaki. Obrigado pela chance de trabalhar em uma empresa incrível.

Dedico este livro a Clarice. Você foi o maior presente que ganhei ao longo da jornada. Titio te ama.

Sobre o autor

Raphael Donaire Albino é doutorando e mestre em Administração de Empresas pela FEA-USP, formado em Sistemas de Informação, pela Unesp de Bauru, e com MBA em Gerenciamento de projetos, pela FGV. Profissional com mais de 10 anos de experiência na área de desenvolvimento de *software*, atua com programação, concepção de novos produtos, análise de requisitos e gestão de projetos e equipes.

Atualmente, é consultor na Plataformatec (uma empresa ágil da cabeça aos pés) e atua como docente em cursos de pós-graduação em Gerência de projetos e Mídias Digitais. Adora escutar, ler, falar e escrever sobre tecnologia e gestão.

Prefácio

Desde 2001, quando o Manifesto Ágil foi publicado, até os dias de hoje, tenho acompanhado o crescimento da comunidade Agile no Brasil. Entre 2001 e 2007, o que chamo de "primeira onda", o Agile era um assunto limitado às catacumbas de um punhado de programadores XP, que viam no Kent Beck um modelo a ser seguido. Nessa gênese da agilidade, o fenômeno era local e bastante fechado.

Passado um tempo, em 2007, aconteceu o primeiro treinamento *Certified Scrum Master* aqui nas terras tupiniquins, com a presença de inúmeros líderes que até hoje influenciam o mercado. Eu estava lá e, após o treinamento, fizemos uma mesa redonda discutindo "como tomar o mercado das mãos dos tradicionalistas".

Depois desse marco, o ágil saiu de um ritual tribal de alguns desenvolvedores renegados para o grande mercado, e partir daí, bancos, telecoms e diversas organizações de software e serviços de TI começaram a tentar rodar seus *sprints* e a se adequar ao "*mindset* ágil". Essa foi a "segunda onda" Agile no país: de um assunto de programadores na primeira onda, o Agile se transformou em um assunto de gestão, em que o Scrum era seu maior expoente.

Quando as conversas migraram de código para gestão, agilistas martelavam *ad nauseum* as diferenças entre um processo prescritivo (geralmente atacando métodos tradicionais como *RUP*, *PMBOK* e *CMMi*) para um processo empírico (como o *XP* ou *Scrum*). Isso foi até positivo, pois educou o mercado um pouco sobre complexidade. Porém, o efeito colateral é que a falta de profundidade nesses estudos levou muitos agilistas a prematuramente afirmar que "desenvolvimento de software é complexo", e por isso a imprevisibilidade inerente desse tipo de sistema impedia qualquer tipo de planejamento, medição e projeção futura, já que no "complexo tudo pode acontecer".

Consequentemente, se desenvolveu no Agile uma alergia ou intolerância a números, estatística e qualquer tipo de avaliação quantitativa de seus ambientes. "Pessoas mais que processos", vociferavam.

Nem preciso dizer que tal narrativa sem planejamento, sem números, sem previsibilidade, sem hierarquia não colou para a maioria dos CIOs, executivos de TI, gerentes de projeto e líderes de equipes no país. Experimentar "o complexo", a inovação e ambientes sem restrições envolve risco e o perfil típico do gestor brasileiro é conservador, fruto de uma sociedade de baixa confiança (tente explicar o que é um cartório para um dinamarquês, americano ou suíço, e espere as risadas). O fato é que a agilidade precisa se reinventar aqui no Brasil para formar uma nova narrativa, e assim cumprir a missão de *tomar o mercado*.

A segunda onda continua e implementações ágeis vão continuar a pipocar aqui e ali de norte a sul do país. Entretanto, a partir de 2012, algumas coisas mudaram. Sob influência do Alisson Vale e da comunidade Kanban, um pequeno movimento tem emergido no Brasil e no mundo.

Esse grupo de pensadores tentam conciliar as complexidades dos ambientes de TI com métricas objetivas, visando principalmente melhoria contínua, garantia da qualidade, aumento da previsibilidade e também análise de risco. O ferramental desses pensadores se resume basicamente a entender seu ambiente de trabalho como um sistema passível de medição e alavancagem (*Systems Thinking*). Se essa é a terceira onda da agilidade no Brasil, só o tempo vai nos dizer. O que acreditamos é que deve existir um casamento entre *soft skills* e *hard skills* de gestão.

Fico muito feliz que este livro tenha sido escrito pelo Raphael Albino, um dos meus alunos de Kanban, pois nossas discussões sempre me renderam bons aprendizados. Creio que este texto serve como apoio a toda comunidade de desenvolvimento de software, não só de agilistas.

Taiichi Ohno, o criador do Sistema Toyota de Produção, e talvez um dos maiores pensadores de gestão do mundo, dizia: *"Não é possível ter melhoria sem padrões".* Este livro vai ensiná-lo a entender esses padrões e principalmente lhe oferecerá um ferramental que vai ajudá-lo a entender melhor seus problemas começando na próxima segunda-feira.

É muito empolgante ver que muito dos assuntos deste livro estão em constante evolução e as novidades aparecem a cada semana. Temos muito ainda o que aprender sobre a matemática e a estatística de um fluxo de trabalho. Creio que este livro será a literatura base para quem quiser entrar nessa "terceira onda".

Rodrigo Yoshima

Introdução

Toda vez que penso em indicadores e métricas, recordo-me de uma frase de H. James Harrington que diz: *"Metrificar é o primeiro passo para o controle e eventualmente para a melhoria. Se você não consegue medir algo, você não consegue entendê-lo. Se você não consegue capturá-lo, você não consegue controlá-lo. Se você não consegue controlá-lo, você não consegue melhorá-lo".*

Antes que você pense que estou fazendo qualquer apologia à cultura do comando e controle, gostaria de compartilhar que, para mim, controle é a capacidade que uma equipe tem de manusear e desenvolver ferramentas que promovam um ambiente de autogestão.

Desde que eu comecei a trabalhar com desenvolvimento de software, tenho lidado com dois pontos de vista quanto às métricas:

1. No primeiro, as métricas são aplicadas como ferramentas que buscam simplificar a equipe em números, e a única razão para coletá-las visa exigir respostas das pessoas e criar conflitos perigosos. Exemplos deste tipo de métricas são número de testes unitários escritos por desenvolvedor, velocidade individual etc.

2. No segundo, as métricas são utilizadas com o intuito de promover ações de melhoria contínua e, a partir das respectivas visualizações, trazem visibilidade sobre a saúde do processo ao time. Além disso, sua análise promove um ambiente em que os cenários dos prazos de entrega são projetados a partir de uma base consciente (interpretação) e consistente (histórico real da equipe).

Dado que estamos em um meio onde queremos entregar melhores produtos de software para os clientes e usuários, incluir em nosso repertório formas de monitorar a eficiência do processo de construção de software se torna algo relevante. Mas, afinal, por que você deve aprender sobre métricas e visualizações do processo de desenvolvimento software?

- Se você trabalha como Agile Coach ou desenvolvendo software em um ambiente ágil e tem interesse em discutir métricas que podem ajudar a analisar a saúde do seu processo de construção. Especialmente se você tem interesse em fugir de análises subjetivas.

- Se você é um Gerente de Produtos, ou *Product Owner*, e precisa administrar as expectativas dos *stakeholders* interessados no produto que está em construção ou operação, e está em busca de formas para projetar cenários para as entregas que estão sendo construídas a partir de dados históricos e não de achismos.

- Se você é um CTO ou CIO e busca propor aos times informações úteis para analisar o processo de desenvolvimento e atendimento de demandas, de forma concreta, quantitativa e analítica, buscando promover melhorias a partir de dados.

- Se você acaba de entrar em um departamento no qual o produto é um software e não tem a mínima ideia de quais métricas, variáveis ou visualizações levantar para compreender esse admirável mundo novo.

O livro está estruturado de uma forma que você possa: compreender o universo que suporta um processo ágil (capítulo 1); analisar os desafios de quem busca gerenciar o trabalho em progresso (capítulo 2); visualizar e interpretar o tempo que as demandas têm levado no seu processo (capítulo 3); acompanhar a cadência de entrega (capítulo 4); avaliar a saúde do seu processo (capítulo 5); e, por fim, acompanhar e projetar cenários para a entrega de um escopo (capítulo 6). No último capítulo, compartilharei algumas dicas e considerações finais sobre a temática métricas de processo.

Espero que você encare esta obra como um guia de aprendizado e boas práticas que lhe traga uma maior visibilidade do seu trabalho e aumente a previsibilidade das suas entregas. Sinta-se à vontade para escolher o capítulo que quiser e fizer sentido para o seu dia a dia.

Boa leitura.

CAPÍTULO 1

Análise do processo de desenvolvimento de uma equipe

1.1 Introdução

Você já parou para pensar quão conectado está o processo de desenvolvimento de software com os métodos científicos? Quando se pensa na produção de um artefato científico, seguimos os passos de:

1. Observar o fenômeno.
2. Formular uma hipótese para explicar o fenômeno.
3. Usar a hipótese para fazer uma predição.
4. Testar a predição através de experimentos ou novas observações, e modificar a hipótese com base nos resultados do teste.

Ao fazer um paralelo com o desenvolvimento de software, é possível pensar que o fenômeno nada mais é do que um problema de negócio que existe no mundo exterior. As hipóteses formuladas são a ideação das funcionalidades que estarão presentes no produto que resolverão o problema de negócio. As predições nada mais são do que as suposições quanto a utilidade do produto para os usuários finais e, a partir dos resultados obtidos quanto ao uso do produto e dos *feedbacks* das reais necessidades dos usuários, um novo direcionamento é criado para a evolução do software.

A partir do momento em que você enxerga o desenvolvimento de software como um método que tem por objetivo resolver um problema a partir de uma solução que prioriza as necessidades reais e com um processo de criação otimizado, você traz as bases da ciência para a forma como você cria software.

Neste capítulo, teremos a oportunidade de dar um passo para trás e olharmos para o problema que estamos buscando resolver com o software que será criado. Além disso, discutiremos a importância de criar e olhar para o processo de desenvolvimento de uma equipe antes de começar a medi-lo e visualizá-lo. Ao final, espero que você consiga ter respostas para perguntas como:

- Que problema de negócio estou resolvendo?
- Como priorizar as funcionalidades a partir de referências de negócio?

- Por que é importante gerenciar o fluxo de trabalho da equipe?
- Como gerenciar o fluxo de trabalho de forma mais eficiente?

1.2 O entendimento da solução a ser construída

Antes de discutirmos a relevância de analisar e gerenciar o fluxo de desenvolvimento de uma equipe, gostaria de destacar o quão importante é criarmos produtos de software que gerem resultados para os nossos clientes e usuários.

Uma das grandes dificuldades que as equipes têm ao construírem uma solução é a de compreenderem como ela se conecta com os objetivos de negócio de uma organização. Como reflexo de tal desalinhamento, temos situações nas quais são criadas soluções em que grande parte das funcionalidades são raramente ou nunca utilizadas.

Ao olharmos para o mercado nacional de Tecnologia da Informação, que em 2015 movimentou 12,3 bilhões de dólares em software e 14,3 bilhões de dólares do mercado de serviços (ABES, 2016), devemos estar conscientes da importância de construirmos produtos que tragam resultados dado o tamanho do investimento que está sendo feito.

Para que você consiga avaliar onde o produto de software se encaixará no modelo de negócio da empresa que receberá a solução, recomendo que antes de começar a desenvolvê-lo, você e a sua equipe criem um *Business Model Canvas* (BMC).

O BMC tem como objetivo disponibilizar uma ferramenta capaz de permitir a qualquer pessoa criar ou modificar seu modelo de negócio, de maneira que essa pessoa utilize uma linguagem comum que possibilite a troca de experiência e ideias com outras pessoas envolvidas no mesmo processo. Ele funciona com um mapa ou guia para a implantação de uma estratégia organizacional, processos ou sistemas, pois um modelo de negócio tem como finalidade descrever a lógica de como uma organização cria, entrega e captura valor (OSTERWALDER; PIGNEUR, 2013).

Os nove blocos da ferramenta são:

- **Clientes:** define os diferentes grupos de pessoas ou organizações que a empresa em questão pretende atender ou atingir.
- **Proposições de valor:** descreve o conjunto de produtos e serviços que criam valor para um segmento específico de clientes.
- **Canais:** descreve como uma empresa atinge e se comunica com seu segmento de clientes para entregar a proposição de valor pretendida.

- **Relacionamento com clientes:** descreve os tipos de relacionamentos que uma empresa estabelece com um segmento específico de clientes.
- **Fontes de receitas:** representa o lucro que uma empresa gera a partir de cada segmento de clientes atendidos, identificando o valor real que cada cliente está disposto a pagar pelo bem ou serviço.
- **Recursos-chave:** descreve os ativos mais importantes necessários para que o modelo de negócio funcione.
- **Atividades-chave:** descreve as atividades mais importantes que a empresa deve executar para fazer o modelo de negócio funcionar.
- **Parcerias-chave:** descreve a rede de relacionamento de fornecedores e parceiros necessários ao desempenho do modelo de negócio.
- **Estrutura de custos:** descreve todos os custos envolvidos na operação do modelo de negócio.

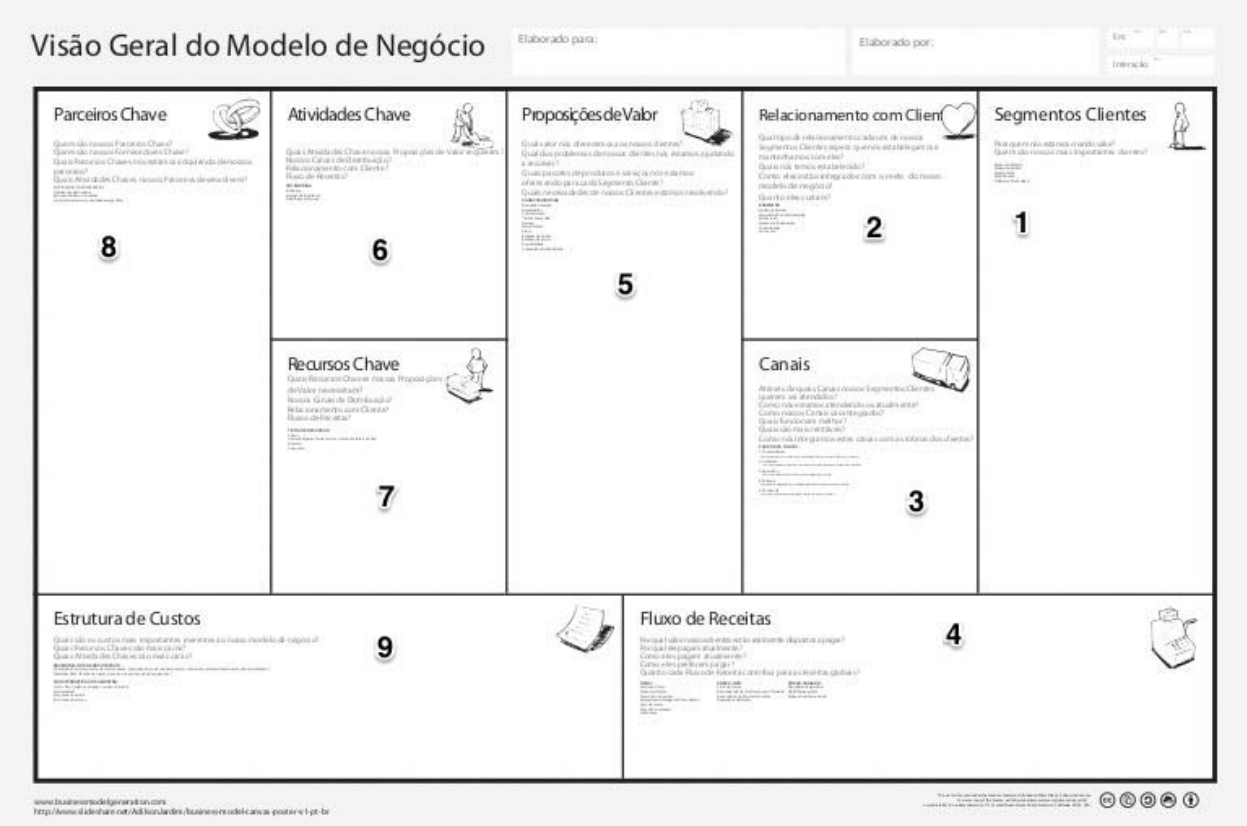


Figura 1.1: Representação do Business Model Canvas

Segundo Osterwalder e Pigneur (2013), os blocos numerados de 1 a 5 posicionados no lado direito do Canvas representam o lado emocional e de valor para uma organização; enquanto que o lado esquerdo, englobando os blocos numerados de 6 a 9, representam a parte lógica e eficiente do processo.

A partir do BMC criado e atualizado, você poderá discutir com os *stakeholders* quais são os resultados que a empresa espera alcançar com o produto que será criado, como ele se conectará aos novos ou atuais clientes, como auxiliará na geração de receita, como será distribuído, qual tipo de valor entregará aos diferentes segmentos de clientes, quais recursos serão necessários para que ele ganhe vida e como ele se enquadrará na estrutura de custos da organização.

Ao aplicar tal tipo de técnica antes da construção do software, você entenderá o modelo de negócio da empresa e, principalmente, terá a oportunidade de alinhar os objetivos do produto com os objetivos de negócio da empresa junto aos *stakeholders*. Isso dará clareza para todos os interessados e envolvidos na criação da solução.

O QUE SÃO *STAKEHOLDERS*?

São indivíduos ou estruturas ativamente envolvidas com um projeto ou produto, ou cujos interesses podem ser afetados pelos resultados de determinada ação (PROJECT MANAGEMENT INSTITUTE, 2013). A palavra vem do inglês e possui uma estrutura em que *stake* significa interesse, participação, risco, enquanto *holder* representa aquele que possui.

Dentro do cenário de software, é comum categorizar clientes, usuários, equipe de desenvolvimento, gestores, diretores, áreas de negócio, como *stakeholders*.

A partir do BMC criado e atualizado, você poderá discutir com os *stakeholders* quais são os resultados que a empresa espera alcançar com o produto que será criado, como ele se conectará aos novos ou atuais clientes, como auxiliará na geração de receita, como será distribuído, qual tipo de valor entregará aos diferentes segmentos de clientes, quais recursos serão necessários para que ele ganhe vida e como ele se enquadrará na estrutura de custos da organização. Outra discussão importante de se realizar antes de começar a construir um software diz respeito a definição dos critérios que servirão como norte para a priorização das funcionalidades que comporão a solução.

Neste sentido, é válido reforçar que entregar soluções técnicas complexas e sofisticadas não representa que você esteja melhorando ou atingindo resultados de negócio. Em outras palavras, você pode por vezes estar entregando produtos ou melhorias técnicas que não farão o negócio melhor.

Para que você priorize funcionalidades que trarão valor, lembre-se de criar métricas de negócio. Ao longo do livro, teremos a oportunidade de discutir métricas de processo e como elas se relacionam com as métricas de negócio.

O QUE É UMA BOA MÉTRICA DE NEGÓCIO?

Uma boa métrica de negócio deve ser comparável. Uma taxa que diz que 10% das pessoas interessadas em seu produto se tornaram de fato clientes não lhe dirá muita coisa se você não tiver o que comparar com ela. Como estava a métrica na semana passada? No mês passado? No ano anterior? As conversões estão crescendo? Responder questões como estas permitirão que você possa fazer melhor uso da métrica.

Outra característica importante sobre métricas de negócio é que elas devem ser compreensíveis, isto é, elas não podem ser complicadas e todos devem compreender o que elas representam.

A métrica de negócio deve também ser uma relação ou uma taxa. Números absolutos não devem ser usados. A quantidade de usuários pode ser pouco útil, porém, a porcentagem de usuários ativos diários já poderá trazer tendências de crescimento. Razões e taxas são comparativas, o que ajuda a tomar melhores decisões.

Por fim, lembre-se: métricas de negócio boas mudam como você se comporta. Se a métrica muda e você não sabe a motivação ou o que fazer com ela, então você tem uma métrica ruim.

As métricas de negócio podem ser classificadas em quantitativas e qualitativas.

- **Qualitativas:** geralmente são originadas de conversas com clientes atuais ou possíveis futuros clientes. Dificilmente é possível transformar tais métricas em números, porém, elas geram informações importantes sobre percepção, interesse e motivações.
- **Quantitativa:** são os números. Elas não lhe darão respostas, mas vão ajudá-lo a fazer melhores perguntas.

No contexto de negócios, você perceberá que descobre coisas qualitativamente e prova-as quantitativamente. Uma pergunta que você deve estar se fazendo agora é: que tipo de métrica devo utilizar para garantir que a solução que será criada está melhorando meu modelo de negócio?

A seguir, listarei algumas métricas que você poderá usar quando tiver o trabalho de analisar os resultados dos produtos de software que serão criados. Porém, sugiro leituras

complementares como *Lean Analytics* e *Product Management* (uma boa dica de leitura é o livro **Gestão de produtos de software: como aumentar as chances de sucesso do seu software**, do Joaquim Torres, lançado pela Casa do Código).

- **Receita recorrente anual:** total de receita recorrente gerada pelo produto (por exemplo, total de receita mensal gerada pela assinatura do produto).

- **Receita anual total:** total de receita gerada pela organização, tanto via produto como via serviço. Geralmente os investidores avaliam as empresas a partir da sua capacidade de geração de receita recorrente, pois prestação de serviço é algo periódico e menos escalável do que a venda de um produto.

- **Receita recorrente anual por cliente:** total de receita recorrente gerada por cada cliente da carteira. Se o valor dessa métrica está subindo, isso representa que você está conseguindo gerar vendas casadas aos clientes, o que acaba sendo um indicador positivo de saúde do negócio.

- **Receita recorrente mensal:** total de receita gerada pelos clientes que assinam o produto mês a mês.

- **Lucro bruto:** é o valor que você obtém ao subtrair, da receita total, os custos variáveis do seu negócio. Como o próprio nome diz, os custos variáveis são gastos que mudam de acordo com o volume de vendas ou de produção. Quando elas aumentam, eles seguem a mesma tendência. Se caem, os custos acompanham a queda.

- **Custo de aquisição do cliente:** deve ser o custo total de aquisição de clientes. Ao calcular o CAC como um custo "misturado" (incluindo os clientes adquiridos organicamente), em vez de isolar clientes adquiridos através de marketing "pago", você deixa de saber o quão bem suas campanhas pagas estão funcionando e se elas são lucrativas. É por isso que os investidores consideram que o CAC pago é mais importante do que o CAC misturado na avaliação da viabilidade de um negócio, pois ele informa se uma empresa pode escalar seu orçamento de aquisição de usuários de forma lucrativa.

- **Evasão:** indica a média de cancelamentos de assinaturas dos clientes de uma empresa, assim como as perdas financeiras que elas representam, em determinado período. Pode ser calculado através de uma taxa do total de clientes cancelados dividido pelo número total de clientes ativos do último mês, ou pela receita recorrente mensal gerada pelos clientes cancelados dividido pelo total de receita recorrente mensal.

- **Número de usuários ativos:** pode ser considerada uma métrica que não ajuda na tomada de decisão, dado que você pode ter muitos usuários ativos que não geram

receita. Porém, tal métrica pode ser útil como forma de entender o engajamento da solução no mercado.

As métricas discutidas até aqui ajudarão você e a sua equipe na tomada de melhores decisões quando estiverem discutindo qual funcionalidade trará maior resultado para o negócio.

Nos próximos capítulos do livro, você terá a oportunidade de aprender outras técnicas de priorização e espero que, a partir do que vimos até aqui, você passe a:

1. Compreender as motivações por trás da solução que será criada;
2. Conectar o produto criado (software) com os resultados de negócio (como novos clientes, novos mercados);
3. Definir métricas de negócio que suportarão a priorização da solução que você estiver criando.

1.3 O fluxo de desenvolvimento de uma equipe

Se você trabalha em um ambiente que não conta com um processo para se construir software, é bem provável que você lide com desafios na qualidade do produto final, na organização e otimização dos recursos, e no planejamento do que será entregue. Costumo dizer que é melhor se ter um processo com suas limitações e que pode ser evoluído do que trabalhar em um local caótico.

Quando discutimos formas de criar ou evoluir produtos de software, comumente as categorizamos em dois tipos de abordagem: tradicional e ágil. Ao longo dos anos, a abordagem tradicional tem perdido sua popularidade, enquanto a adoção da abordagem ágil tem crescido, geralmente sustentada por *frameworks* e boas práticas como o XP (*eXtreme Programming*), Scrum, Kanban, Lean, FDD (*Feature Driven Development*), entre outras.

QUAIS AS DIFERENÇAS ENTRE A ABORDAGEM ÁGIL E A TRADICIONAL?

Basicamente, o modelo tradicional separa o desenvolvimento de software em uma sequência pré-definida de fases, incluindo planejamento, *design*, construção, validações, entrega e suporte.

Figura 1.2: Modelo tradicional de desenvolvimento

Por outro lado, o desenvolvimento ágil de software segue uma linha de entregas cíclicas e incrementais, a partir de uma colaboração entre as pessoas envolvidas no processo de criação da solução.

Figura 1.3: Modelo ágil de desenvolvimento

Enquanto no modelo tradicional existe uma separação entre as etapas necessárias para a construção de um software, no modelo ágil tal separação não existe, dado que há uma multidisciplinaridade das pessoas envolvidas na criação da solução.

O modelo tradicional de desenvolvimento de software tem como principal desvantagem a dificuldade de lidar com mudanças. Por exemplo, dado que uma funcionalidade foi mal definida na etapa de planejamento, o usuário só descobrirá o erro após uma série de outras funcionalidades ter sido criada. Em suma, o ciclo de *feedback* em uma abordagem tradicional é mais lento quando comparado com abordagens ágeis.

Quando estamos lidando com abordagens ágeis, geralmente temos o contexto de um prazo esperado, recursos e orçamento limitados. Entretanto, o escopo será ajustado a partir do valor que se espera entregar aos usuários. Já em uma abordagem tradicional, temos um escopo de entrega fixo, e o grande desafio será administrar o prazo de entrega e o orçamento que será necessário para produzir a solução.

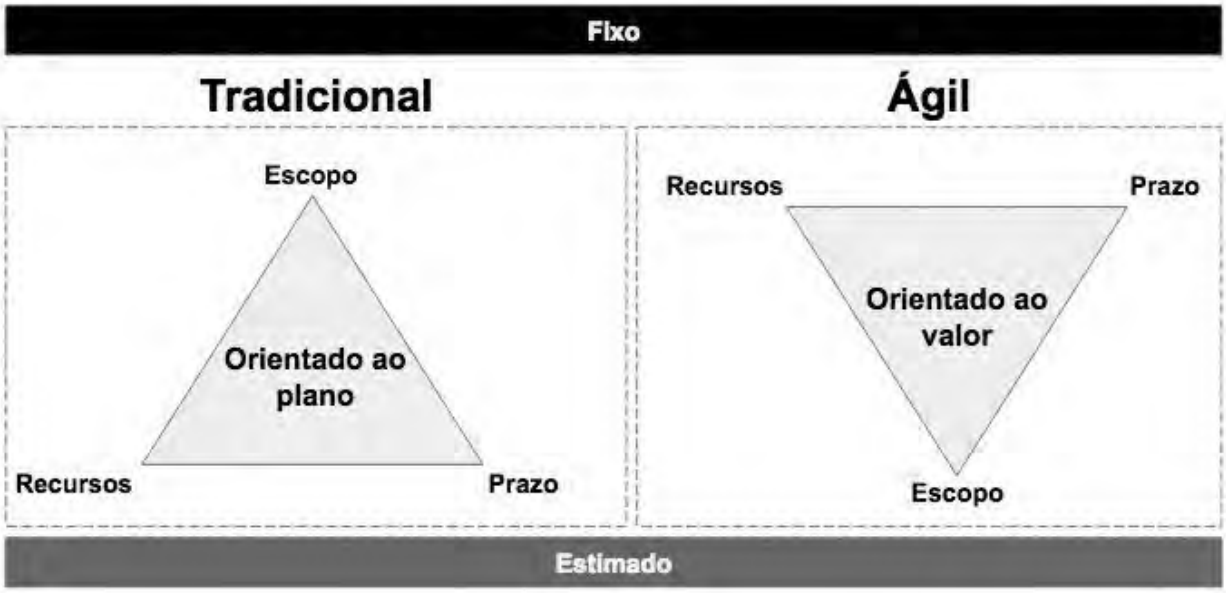


Figura 1.4: Diferença entre os modelos tradicional e ágil

Uma das grandes diferenças entre as abordagens de desenvolvimento de software ágil e tradicional diz respeito a qualidade e testes. Ao passo que, no modelo tradicional, a fase de teste vem depois da fase de construção, na abordagem ágil, os testes são feitos concomitantemente com o desenvolvimento. Dessa maneira, a rapidez em que são feitas as trocas de informação acaba reduzindo o número de falhas no produto que está sendo criado.

O modelo tradicional cria uma mentalidade de projeto e estabelece seu foco estritamente na entrega do software. Já o ágil introduz uma mentalidade de produto que se concentra em garantir que a solução criada satisfaça as necessidades dos clientes finais.

Uma dúvida que pode estar pairando sua cabeça neste momento é: mas afinal, será que o modelo tradicional não serve para nenhum caso? A resposta é: depende. Se você está lidando com situações nas quais há uma clareza nas funcionalidades que precisarão ser construídas e que não haverá qualquer tipo de mudança no que se espera com o produto final, utilizar um modelo tradicional pode ser mais eficiente. Agora, dada a natureza de incerteza que envolve o desenvolvimento de software e sabendo que a única certeza é que existirão mudanças, é aconselhável se pensar em seguir um processo ágil.

Ambas as abordagens têm suas fraquezas e forças. A chave para decidir qual é a certa dependerá do contexto em que você estiver inserido. Ocorrerão mudanças frequentes? Se sim, escolha uma abordagem ágil. Você sabe exatamente o que precisa ser feito? Então, siga uma abordagem mais tradicional. Independente da maneira, o que realmente importa é que você consiga construir software de qualidade e que gere valor aos seus usuários e clientes.

Todo o restante do livro será baseado na abordagem ágil de se construir software, e vale um reforço do uso da palavra *ágil* em vez de *Ágil*. Ao utilizar a palavra Ágil, há naturalmente uma associação com métodos ou *frameworks* como Kanban, Scrum etc.

Você perceberá no dia a dia que o ferramental usado será um suporte para a criação da sua forma de construir software, e que o grande desafio está em trazer **agilidade** (em minúsculo) para a cultura da sua empresa e para todos que estão envolvidos no processo de forma direta ou indireta.

Quando falamos sobre formas de gerenciar o fluxo de uma equipe de desenvolvimento de software que está buscando uma abordagem ágil de desenvolvimento, grandes são as chances de nos depararmos com o Scrum e o Kanban.

Kanban

Kanban é uma abordagem enxuta para o desenvolvimento de software ágil. Na realidade, o Kanban possui uma série de significados. Literalmente, é uma palavra japonesa que significa "cartão visual". Na Toyota, Kanban é o termo usado para o sistema de sinalização visual e física que liga todo o processo produtivo de produção enxuta.

Em 2010, David Anderson criou o que ele chamou de "sistema Kanban para desenvolvimento de software", que combina o pensamento *lean* e a teoria das restrições. Assim, enquanto Kanban no desenvolvimento de software é relativamente novo, no processo produtivo o termo já é utilizado há mais de meio século.

Segundo Anderson (2011), o sistema Kanban trabalha com três premissas que são:

- **Comece com o que você já faz hoje:** mapeie o seu fluxo de trabalho atual, crie formas de ter visibilidade sobre o que está sendo trabalhado e busque evoluir o processo.

- **Busque mudanças incrementais:** melhore sempre e sem ser radical. Grandes mudanças ocasionam desafios de comunicação, dificuldades no aspecto técnico e, geralmente, afetam a capacidade produtiva de uma equipe.

- **Respeite o processo atual, seus papéis, responsabilidades e cargos:** não mude drasticamente a forma como sua organização está estruturada. Faça melhorias evolutivas e não seja disruptivo.

Além disso, o sistema Kanban está sustentado em seis propriedades:

1. **Visualize o fluxo de trabalho:** divida o trabalho em partes, escreva cada demanda em um cartão e tenha visão de todo o trabalho que está em andamento. Além disso, nomeie colunas para ilustrar onde cada demanda está no fluxo de trabalho.

2. **Limite o trabalho em progresso:** atribua limites explícitos para a quantidade de itens que podem estar em andamento em cada etapa do fluxo de trabalho.

3. **Meça e gerencie o fluxo de trabalho:** tenha visibilidade de quanto tempo as demandas que passam pelo processo têm levado para serem concluídas e remova os impedimentos que por ventura estejam bloqueando o fluxo de trabalho.

4. **Torne as políticas explícitas:** defina, divulgue e socialize o processo, assim todos terão uma clareza de como ele funciona e de como o trabalho realmente é feito. Exemplos de políticas são os critérios para as transições entre as etapas do fluxo de trabalho e o limite de trabalho em progresso.

5. **Desenvolva loops de feedback:** reduza o tempo do ciclo de *feedback*, pois quanto mais tardia a entrega, mais demorada será a geração de valor daquilo que foi trabalhado.

6. **Melhore de forma colaborativa:** exerça a filosofia de melhoria contínua baseada no *kaizen*, promovendo pequenas alterações no processo que causem menor resistência à mudança.

A seguir, listo alguns benefícios que você pode ter caso opte por utilizar o Kanban na sua equipe:

- Os gargalos do processo ficarão visíveis em tempo real, o que levará as pessoas a colaborarem para otimizar toda a cadeia de valor em vez de olhar apenas para a sua parte.

- Provê um caminho mais ameno quando a organização está em um processo de transição de uma abordagem tradicional para uma abordagem de desenvolvimento de software ágil.

- Fornece uma maneira de criar software de forma ágil, sem necessariamente ter de lidar com iterações de tempo fixo, como por exemplo, as *sprints* do Scrum.

- Útil para situações nas quais há uma alta taxa de incerteza e alta variabilidade sobre o trabalho que precisará ser executado.

- Tende a se espalhar naturalmente por toda a organização e para outros departamentos, como por exemplo, recursos humanos, operações e vendas, aumentando assim a visibilidade de tudo o que está acontecendo.

Tenha muito cuidado caso você escute alguém dizendo: "No Kanban não se usa iteração"; "No Kanban não se estima"; "O Kanban é melhor do que qualquer outro *framework* ágil"; "O Kanban substitui qualquer outro *framework* ágil". Todas as afirmações anteriores são mitos.

Trabalhar com iterações no Kanban é algo opcional, faça tal controle de tempo caso o seu contexto exija. O mesmo se aplica para as estimativas. Dado que a sua equipe passou a medir, utilize os dados do passado como referência para desenhar cenários futuros. Por fim, o Kanban é apenas uma ferramenta para a gestão do fluxo de trabalho e, por isso, dificilmente ela substituirá qualquer outro tipo de ferramental que alicerce a forma como é construído software na organização.

Scrum

Já o Scrum é um *framework* para gestão ágil, originalmente aplicado em projetos de desenvolvimento de software, criado em 1996 por Ken Schwaber e Jeff Sutherland. Com ele, o produto que está sendo desenvolvido progride a partir de uma série de iterações chamadas *sprints*, que podem ser organizadas semanalmente, quinzenalmente ou mensalmente.

O Scrum utiliza um esqueleto iterativo e incremental que é sustentado por três papéis principais (SCHWABER, 2004):

- **Product Owner (Dono do Produto):** representa os interesses do cliente e, em alguns casos, é o próprio cliente.
- **Scrum Master (Mestre Scrum):** responsável pela execução de todas as regras do Scrum.
- **Team (Time):** responsável por desenvolver o software.

Segundo Schwaber (2004), o Scrum não é um processo previsível, ele não define o que fazer em todas as circunstâncias. Ele é utilizado em contextos que mudam frequentemente, possuem requisitos altamente mutáveis, e é recomendado para outras áreas que não o desenvolvimento de software e principalmente para projetos de pesquisa e inovação.

O desenvolvimento de um projeto Scrum tem início quando uma visão do projeto que deverá ser desenvolvido é criada. Essa visão deve conter uma lista das características que o cliente espera que o projeto atenda ao seu final, tendo como referência o seu problema atual.

Posteriormente, o *Product Backlog* é criado contendo a lista de todos os requisitos que foram identificados. Os requisitos devem ser priorizados e divididos em entregáveis (*releases*).

Schwaber (2004) explica que cada *sprint* se inicia com uma reunião de planejamento chamada *Sprint Planning Meeting*, na qual o *Product Owner* e o *Team* decidem em conjunto o que deverá ser desenvolvido naquela *sprint* gerando assim o *Sprint Product Backlog*.

Ao longo da *sprint*, reuniões são feitas diariamente para se acompanhar o progresso do trabalho (*daily meeting*) e outras reuniões podem ser agendadas se necessário. Ao final da *sprint*, uma reunião de revisão (*Sprint Review Meeting*) é realizada para que seja apresentado o resultado alcançado. Neste instante, são validadas as funcionalidades e adaptações são realizadas caso sejam necessárias.

Após a revisão do trabalho, a equipe realiza a última cerimônia do fluxo Scrum, chamada *Sprint Retrospective Meeting*, que tem como objetivo avaliar o processo de trabalho, analisando os acontecimentos da *sprint* e planejando melhorias que serão adotadas para garantir uma melhor fluidez do trabalho da equipe.

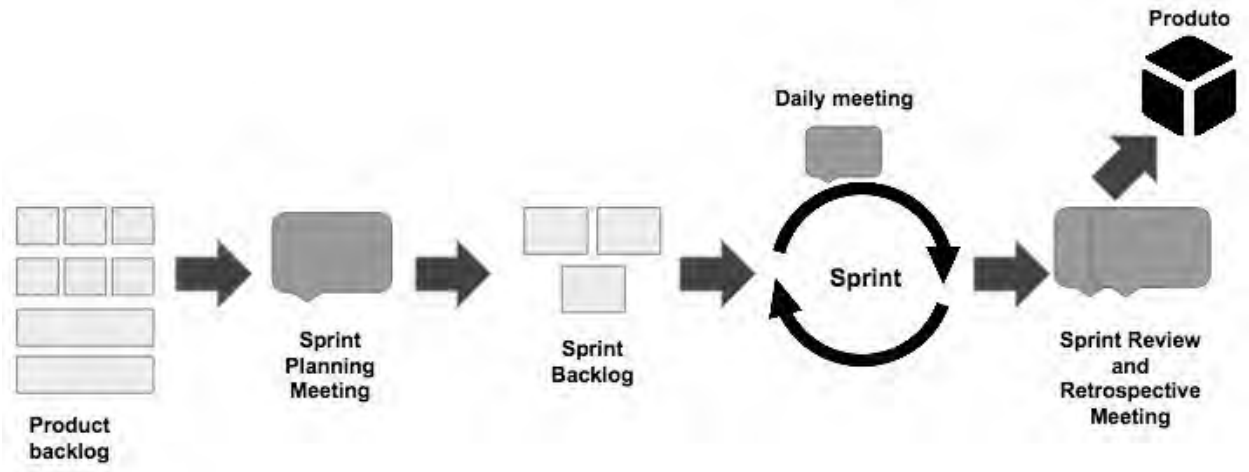


Figura 1.5: Fluxo Scrum

Um produto com valor agregado deve ser entregue ao cliente e esse processo se repete até que se tenha uma versão minimamente utilizável do software.

Toda equipe que adota o Scrum deve estar sustentada por três pilares. O primeiro deles é a transparência, que tem como objetivo dar clareza e visibilidade sobre o trabalho que está sendo desenvolvido para todas as partes envolvidas.

O segundo pilar é a inspeção, que visa reforçar a ideia de que todo trabalho deve ser inspecionado com a frequência necessária para que haja a garantia da qualidade no produto final. O terceiro e último pilar é a adaptação, que visa instigar a filosofia de melhoria contínua em todo o trabalho que é conduzido pela equipe com o objetivo de se atingir melhores resultados.

O Scrum é uma ótima forma de estruturar uma equipe que tem como desafio gerar entregas constantes, a partir de um conjunto de requisitos. Caso você esteja pensando em adotar o Scrum ou alguma das cerimônias compartilhadas, saiba que você poderá coletar benefícios como:

- Criar uma organização estruturada em equipes pequenas, multifuncionais e auto-organizadas.
- Ter clareza do trabalho necessário a partir de uma lista de entregas pequenas, concretas e priorizadas.
- Dividir o ciclo de trabalho da equipe em períodos de tempo fixo, garantido uma maior frequência de entrega dentro de um espaço de tempo, o que pode trazer como resultado uma previsibilidade na cadência de entrega.
- Otimizar o planejamento do produto que está sendo construído a partir da colaboração com os usuários e clientes, baseado no aprendizado obtido nas entregas passadas.
- Promover melhorias no processo a partir das retrospectivas realizadas ao final de cada *sprint*.

Tenha muito cuidado para não cair na tentação de achar que existe *framework* ou modelo de gestão perfeito, e que você será ágil da noite para o dia. Você perceberá que o caminho para se trazer um processo ágil de construção de software é árduo, exige abertura, maturidade e gerará mudanças em todas as áreas de uma organização.

1.4 Práticas de engenharia de software

Gostaria de reforçar e citar algumas práticas de engenharia de software que devem suportar o seu processo. O motivo? Não quero que você caia na falácia de que, para ser ágil, basta adotar Scrum ou Kanban, e para que você não lide depois de um tempo com um cenário em que o progresso é lento em decorrência do excesso de débitos técnicos e da baixa qualidade interna do seu software.

Várias das práticas que compartilharei a seguir são baseadas no *Extreme programming* (XP) que é uma disciplina de desenvolvimento de software fundamentada em valores como simplicidade, comunicação, *feedback*, coragem e respeito.

Além disso, o XPScrum possui princípios como:

- Equipe unida;
- Planejamento contínuo;
- Entregas curtas;
- Testes a partir da visão do cliente;
- Propriedade coletiva do código;
- Padronização de código;
- Ritmo de entrega sustentável;
- Metáforas para compartilhar conceitos gerais sobre o software criado;
- Integração contínua;
- Desenvolvimento orientado a testes;
- Melhoria da qualidade do código existente;
- Simplicidade no software escrito e programação em par.

Programação em par

A primeira prática que recomendo é a programação em par. A técnica tem por objetivo colocar duas pessoas trabalhando com o desenvolvimento em uma mesma base de trabalho, sendo que há um piloto, responsável pela escrita do código, e o observador, que revisará o que está sendo desenvolvido. A ideia é que a dupla mude frequentemente de papel.

A programação em par é uma boa prática, pois ela melhora a qualidade do código a partir da troca de experiência, conhecimento e de informações entre as pessoas. Além disso, a técnica ajuda a aumentar o foco de desenvolvimento da equipe.

Cockburn et al. (2001) mostraram a partir de um estudo de campo que a programação em par traz os seguintes benefícios:

- Muitos erros são identificados ao longo do desenvolvimento em vez de etapas futuras (por exemplo, estágio de teste);
- A identificação de erro é estatisticamente menor;
- O código é menor e tem um design melhor;
- A equipe resolve problemas rapidamente;
- A equipe aprende significativamente mais sobre o problema e sobre as práticas de construção de software;
- Projetos de software acabam com várias pessoas conhecendo mais cada peça da solução que foi desenvolvida;
- A equipe aprende a trabalhar junto e a falar mais frequentemente, dando melhor dinâmica e fluidez nas informações.

Surpreendentemente pessoas que são contra a programação em par nunca tentaram utilizá-la e rapidamente aprendem a gostar dela quando tentam.

Vale ressaltar que desenvolver em par a todo momento é algo exaustivo, portanto, fomente pausas ao longo do dia. Outro ponto importante diz respeito ao revezamento entre as duplas com certa frequência, para que os ganhos do uso da técnica se perpetuem por toda a equipe.

TDD (Test-Driven Development)

A segunda prática que recomendo é o *TDD* (*Test-Driven Development*), que nada mais é do que uma técnica na qual você deve primeiro escrever um teste falho antes de escrever o código funcional. Ao escrever os testes antes, você garante que boa parte do seu sistema seja coberto por um teste que garanta o seu funcionamento.

Cabe citar que aplicar o TDD é difícil e leva um tempo para que a equipe passe a adotá-lo. A melhor maneira de aprender a aplicar a técnica é a partir da prática, por isso, faça com que pessoas que já a aplicam colaborem com os novatos, assim, você potencializará a apropriação da prática.

O TDD traz benefícios como maior clareza do código que já foi produzido (gerando um ativo de conhecimento) e qualidade externa do produto (os testes automatizados darão segurança de que uma nova funcionalidade incluída no sistema não impactará as funcionalidades já existentes).

Design incremental

Uma terceira prática que gostaria de recomendar é o design incremental da base de código. Isso significa dizer que a forma como o código está estruturado deve ser simples desde a sua concepção, e deve ser melhorado continuamente.

Se você estiver criando um software complexo, pensar no design é importante a fim de garantir uma melhor comunicação da estrutura da aplicação para quem for trabalhar em seu desenvolvimento, e para garantir a robustez e qualidade futura. Porém, você perceberá que buscar prever todos os detalhes do design de forma antecipada não funcionará, dado que detalhes sobre o problema que está sendo resolvido são incertos por natureza.

Algumas decisões de estrutura do software poderão ser tomadas após a disponibilização de uma primeira versão do produto para os clientes e usuários.

Estimule que a equipe na qual você atua tenha tempo para reescrever e melhorar códigos antigos, com o objetivo de aumentar a vida útil da aplicação a partir de um design claro e simples.

Colaboração no código

Outra prática importante de se trazer para as equipes é a colaboração no código. Técnicas como revisão compartilhada do código produzido auxiliam a equipe no desenvolvimento do senso de responsabilidade coletiva sobre o que está sendo criado.

Promova um ambiente no qual a equipe possa ter tempo para analisar e dar *feedback* sobre o código que está sendo produzido, pois isso aumentará a base de conhecimento de todos os envolvidos.

Processo de publicação automatizado

Para que você e a sua equipe possam entregar código de forma consistente, é importante que haja um processo de publicação (*deploy*) automatizado a fim de que se evite, a cada nova tentativa de entrega da sua aplicação, a produção de novos erros. O grande segredo para que isso aconteça é estruturar o ambiente de forma que ele seja replicável entre os contextos de desenvolvimento, homologação e produção.

Padronização do código

Uma última sugestão de prática diz respeito a padronização do código que é construído. Pode parecer algo burocrático, porém técnicas como documentação de código, definição de estilos e padrões de design auxiliarão a equipe a não desenvolver aplicações inconsistentes e difíceis de serem lidas, entendidas e mantidas.

Considerações finais

As práticas citadas anteriormente podem ser aprimoradas e até mesmo evoluídas. Trazer à tona a discussão sobre a importância de se pensar na engenharia de software como forma

de sustentar um processo tem como objetivo garantir que você não pense que entregar software de forma ágil diz respeito apenas a seguir o *framework* X ou Y.

Antes de propor qualquer tipo de mudança na sua organização, recomendo que você prepare o ambiente com um bom aparato de práticas e técnicas de desenvolvimento. Assim, o seu processo estará dentro de um terreno fértil para entregar software de qualidade.

Caso você esteja em uma situação em que já existe um legado de software e está em busca de incorporar as práticas de engenharia e um processo ágil, comece pelo simples. Avalie que tipo de equipe ou aplicação está mais preparada para desbravar o novo mundo, comece a mudança e não desanime na primeira dificuldade que encontrar.

O desafio é grande, porém os benefícios são inúmeros, como por exemplo: qualidade do produto; cadência (frequência) de entrega; foco no que traz maior valor ao usuário; engajamento entre os *stakeholders*; transparência e visibilidade do status do que está sendo construído; redução dos riscos inerentes a natureza do desenvolvimento de software; controle do custo (dado que o orçamento e prazo são fixos, mas o escopo flexível); antecipação na geração de receita no caso de produtos.

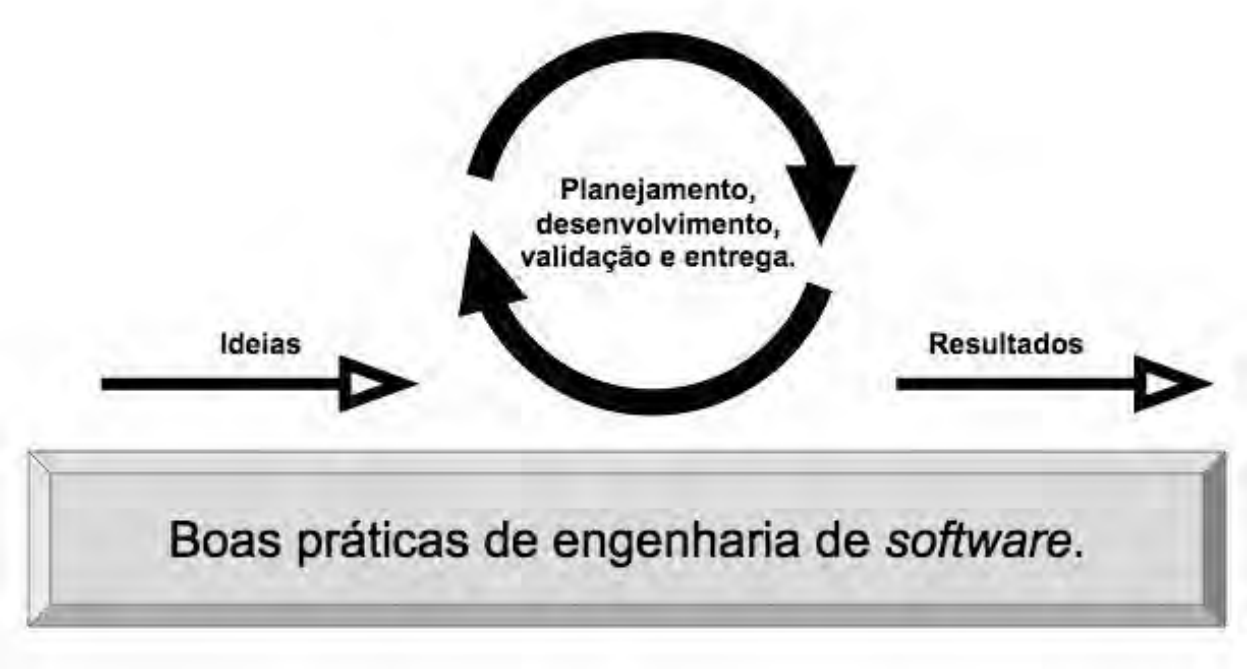


Figura 1.6: Relação entre boas práticas de engenharia e o processo de desenvolvimento

Agora que tivemos a oportunidade de discutir formas de estruturar um processo e práticas de engenharia que garantirão o alicerce para que você e sua equipe possam entregar

melhores soluções de forma ágil, gostaria de passar algumas recomendações para que você construa o seu processo.

1.5 Crie seu processo de desenvolvimento de forma madura

Após entendermos a importância em se definir o que precisa ser feito e de como relacionar boas práticas de gestão de processo com a engenharia de software, a fim de gerarmos entregas mais ágeis, você pode estar se perguntando: como estruturar o fluxo de trabalho da minha equipe?

A primeira recomendação que gostaria de deixar é: parta de uma ideia simples. Quem nunca interagiu com um fluxo de trabalho de uma equipe de desenvolvimento que era composto pelas etapas "A fazer", "Em desenvolvimento" e "Concluído"?

Não há problema algum em desenhar o seu processo dessa maneira e pode até ser esclarecedor para equipes que estejam debutando em olhar para o seu fluxo de trabalho.

O grande desafio ao ter um processo simples é que ele não evidencia possíveis gargalos e restringe a capacidade da equipe em administrar as filas que por ventura surjam ao longo do fluxo de desenvolvimento. Em outras palavras, é como se deixássemos de compreender as restrições existentes no sistema.

Durante a década de 1980, o físico israelense Eliyahu Goldratt criou uma teoria chamada *Teoria das Restrições*, também conhecida como TOC (*Theory of Constraints*). A TOC é baseada em três pressupostos: uma organização possui uma meta a ser atingida; uma organização é mais que a soma de suas partes; e o desempenho de uma organização é limitado por poucas variáveis, ditas restrições do sistema.

Existem cinco etapas decorrentes desses pressupostos:

1. Identificar as restrições do sistema;
2. Decidir como explorá-las;
3. Subordinar tudo o mais à decisão anterior;
4. Elevar as restrições do sistema;
5. Voltar à primeira etapa sem permitir que a inércia cause uma nova restrição.

Trazendo o conceito colocado por Goldratt para o processo de uma equipe que desenvolve software é como se passássemos a olhar o fluxo de trabalho a partir de uma visão sistêmica, isto é, enxergando-o como um fluxo contínuo em vez de segmentá-lo em diversas unidades

independentes (como desenvolvimento, testes e publicação são etapas de um mesmo processo, e não unidades independentes). Isso garante que todo o sistema esteja alinhado com uma única meta (no caso de uma equipe, de entregar software de forma ágil) e permite que os gargalos possam ser trabalhados para atingi-la mais facilmente.

Portanto, mesmo que o seu processo seja simples, garanta que será possível identificar as restrições (gargalos). Uma técnica que poderá ajudá-lo nesse exercício é a criação de etapas de espera ao longo do fluxo.

Imagine que a equipe *XPTO* possui as etapas de "A fazer", "Em desenvolvimento", "Em testes", "Em aprovação" e "Finalizado". Uma estratégia que poderia ser adotada é a de criar etapas sinalizando o aguardo de uma demanda, como ilustrado na figura a seguir.

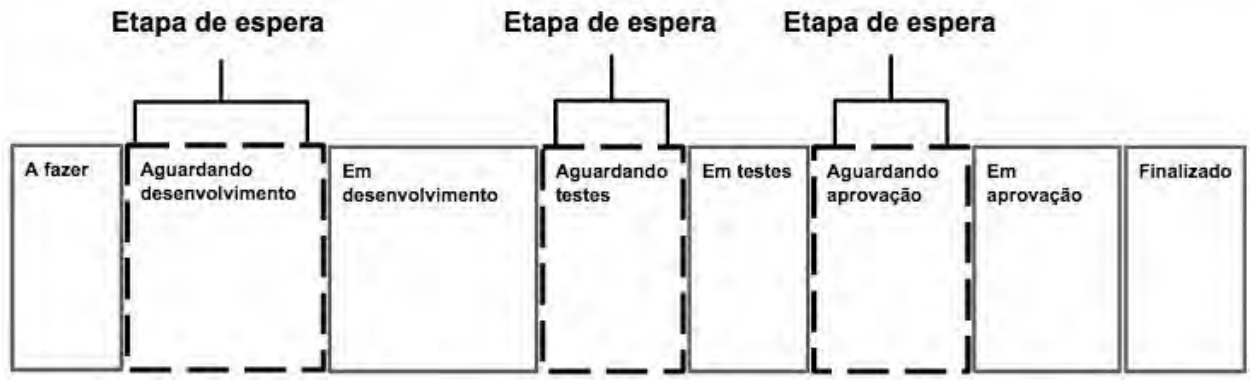


Figura 1.7: Criando etapas de espera no processo

Ao criar uma primeira versão do processo, você passará a tangibilizar o fluxo de trabalho. Tal ação pode ser feita a partir de um quadro que expõe as etapas do processo a partir de colunas em uma parede, ou até mesmo em uma plataforma digital para gestão de fluxo. O importante é que agora você está com o processo mapeado e pode começar a identificar os motivos pelos quais são criados gargalos nele.

A segunda dica ao construir o seu processo é: evite criar um ambiente orientado a tarefas e favoreça um ambiente orientado a missão.

Para Zapellini (2016), ambientes orientados a tarefa são aqueles em que os integrantes do time têm apenas a visibilidade do trabalho que estão executando naquele momento, sem enxergar as implicações disso no fluxo como um todo. Quando isso acontece, normalmente são criadas filas de trabalho individuais e um gestor que centraliza a tomada de decisão, definindo para qual fila será atribuída determinada demanda.

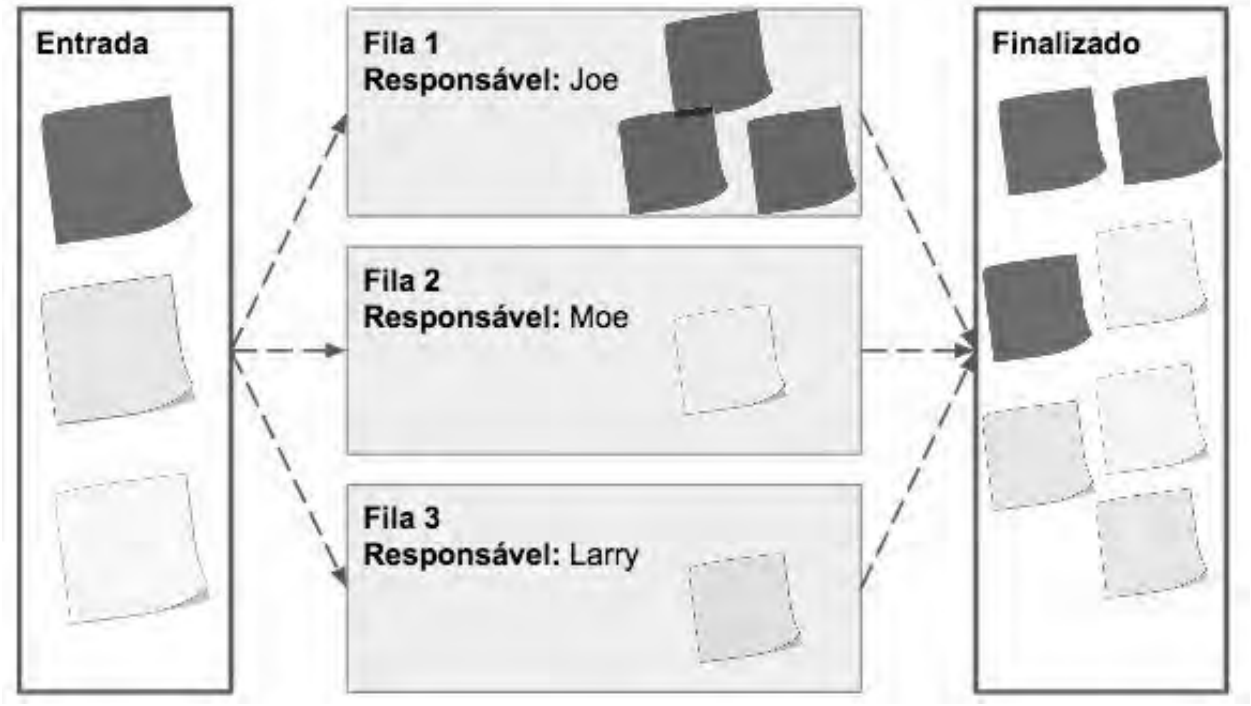


Figura 1.8: Fluxo de trabalho orientado a tarefa

Esta configuração tende a trazer desvantagens, como:

- **Dificuldade para atribuir e priorizar o trabalho:** fica difícil decidir qual a fila mais adequada para receber um item de trabalho, gerando um custo de coordenação desnecessário para o responsável por direcionar as demandas.
- **Cultura de heróis e vilões:** quem entrega bastante acaba recebendo mais demandas para atender, o que nem sempre é saudável no longo prazo; e quem não entrega tanto assim, por tem um perfil diferente, sofre com a pressão dos pares e pode ter sua imagem desgastada injustamente.
- **Especialização exagerada:** itens de determinado tipo são encaminhados para quem domina melhor aquele assunto, o que gera uma centralização do conhecimento e cria uma dependência do processo para determinados profissionais.
- **Desmotivação das pessoas:** as pessoas não entendem a motivação por trás do trabalho que está sendo desenvolvido e por isso não geram propósito.

- **Desperdício:** dado que com filas individuais não se consegue atribuir o trabalho às filas de modo satisfatório e que os integrantes do time são fortemente especializados, algumas filas ficarão sobrecarregadas enquanto outras estarão com baixa utilização.

Ao favorecer ambientes orientados à missão, você muda a pergunta de "O que devemos fazer agora?" para "O que precisamos alcançar?". Assim, o esforço dos integrantes do time fica voltado para resolver os problemas que realmente importam em um determinado momento em vez de limitar-se a atender demandas recebidas de alguém.

Ao criar um ambiente orientado à missão, você estará criando uma fila única de entrada de demandas no seu processo de desenvolvimento, o que fará com que você melhore a utilização dos recursos disponíveis (fazendo com que elas possam exercer mais do que um papel ao longo do processo), diminua o tempo de entrega das demandas que passarão pelo fluxo, e colete um *feedback* mais rápido daquilo que está sendo construído.

Figura 1.9: Fluxo de trabalho orientado a missão

A terceira dica é: faça com que a equipe tenha o compromisso de seguir e evoluir o processo. Crie um ambiente no qual as pessoas possam:

- **Reduzir o tamanho de um item:** se uma demanda que entrará para o fluxo de trabalho da equipe for complexa demais, promova discussões para quebrá-la em entregas mais simples.

- **Monitorar as filas:** faça com que a equipe identifique os gargalos e fortaleça o espírito de ajuda fomentando nas pessoas o questionamento de onde elas podem ser úteis naquele momento.

- **Criar políticas explícitas:** garanta que todos saibam quais são os critérios usados nas transições dos estágios do processo e quais são as condições para que uma demanda comece e seja entregue.

- **Reduzir o abuso:** a partir do momento que você deixa explícito como as coisas funcionam ou por que são como são, você abre uma janela para argumentações baseadas na razão e em fatos, deixando de culpabilizar pessoas ou trazer as discussões para o nível das ideias.

A última dica sobre a criação de um processo maduro diz respeito a tangibilização dos resultados produzidos. Dado que o desenvolvimento de software está em uma perspectiva do conhecimento, é atraente levar as discussões dos problemas para uma seara subjetiva.

Para que você consiga tomar melhores decisões baseadas nos resultados produzidos pelo seu processo, você precisa criar evidências, e é aqui que as métricas se tornam um grande aliado. Conforme os *stakeholders* e a equipe passam a consumi-las com frequência, eles começam a monitorar melhor o processo de desenvolvimento, passam a sugerir melhorias baseadas em dados, passam a discutir cenários quanto aos prazos de entrega, passam a ter maior visibilidade do progresso e começam a construir um cenário que busca trazer maior previsibilidade quanto ao produto que será entregue. Desta maneira, a confiança aumenta e as quebras de expectativas se tornam mais raras.

1.6 Recapitulando

Criar um ambiente onde há clareza sobre as razões nas quais o software está sendo criado, de quais são os resultados esperados com ele e onde haja entregas frequentes e de qualidade é desafiador e altamente convidativo. Neste capítulo, compartilhei algumas dicas para que você crie produtos que tenham seu valor mensurado a partir de um fluxo de trabalho eficiente.

Um aprendizado que espero que você tenha levado é de que não existe formula mágica para transformar o seu processo de construção de software em ágil e que, para que ele seja consistente, é necessária a combinação de boas práticas de processo e engenharia.

Os próximos capítulos deste livro lhe auxiliarão a enxergar as métricas de processo como uma forma de evolui-lo, e não como algo burocrático ou que terá como objetivo medir desempenho dos indivíduos de uma equipe.

1.7 Referências

ABES. *Mercado Brasileiro de Software:* panorama e tendências, 2016. São Paulo: ABES, 2016. Disponível em:

http://central.abessoftware.com.br/Content/UploadedFiles/Arquivos/Dados%202011/ABES-Publicacao-Mercado-2016.pdf.

ANDERSON, D. J. *Kanban:* Mudança evolucionária de Sucesso para seu Negócio de Tecnologia. Blue Hole Press, 2011.

COCKBURN, A.; WILLIAMS, L.; SUCCI, G.; MARCHESI, M. *The costs and benefits of pair programming.* Extreme Programming Examined. Addison-Wesley, 2001.

OSTERWALDER, A.; PIGNEUR, Y. *Business Model Generation.* Hoboken: Wiley, 2013.

PROJECT MANAGEMENT INSTITUTE. *A guide to the project management body of knowledge.* Project Management Institute, 2013.

SCHWABER, K. *Agile Project Management with Scrum.* Microsoft Press, 2004.

ZAPELLINI, W. *5 Strategies to improve software development workflow.* Ago. 2016. Disponível em: http://blog.plataformatec.com.br/2016/08/5-strategies-to-improve-software-development-workflow/

CAPÍTULO 2

A importância de analisar o trabalho em progresso

2.1 Introdução

É muito comum nos depararmos com times de desenvolvimento de software que possuem aquele sentimento de estar trabalhando demais, mas que não conseguem demonstrar todo esse esforço em entregas mensuráveis.

Ser parte de um ambiente no qual as pessoas não têm visibilidade da quantidade de demandas em progresso, ou de onde há um paralelismo excessivo do trabalho, gera algumas consequências como: o não cumprimento de prazos; uma constante alternância de tarefas e contextos; e uma gestão ineficiente do processo.

Em linhas gerais, podemos pensar que o excesso de trabalho em progresso é um grande ofensor quando estamos em busca de entregas de software constantes e de qualidade.

Várias organizações ágeis costumam entoar o mantra "paremos de começar e comecemos a terminar". No entanto, a pergunta que fica é: como? Uma estratégia que tem se mostrado eficiente é de controlar a quantidade de trabalho que ocupa o fluxo de desenvolvimento.

No fundo, os times que visam prevenir gargalos ao longo do processo de construção de software têm buscado o equilíbrio entre o que precisa ser feito e a capacidade de entrega disponível.

Neste capítulo, teremos a oportunidade de discutir a importância da mensuração e da análise do trabalho em progresso (do inglês *work in progress*, ou WIP), bem como a relação entre o WIP e qualidade.

Se você já faz a gestão do WIP, não deixe de conferir o estudo de caso descrito a seguir. Nele, compartilharei uma série de dicas para que você possa aumentar a eficiência do trabalho em progresso do seu time. Ao final, espero que você entenda como o WIP pode lhe ajudar a identificar gargalos e, principalmente, a melhorar a forma como você constrói software.

2.2 O que é o WIP (Work In Progress)?

Antes de discutirmos formas de analisarmos o trabalho em progresso (a partir daqui passarei a utilizar a sigla em inglês *WIP*), gostaria de compartilhar algumas definições importantes. Pensando em nível de processo, podemos dizer que WIP é qualquer trabalho que não tenha sido concluído, mas que já tenha incorrido em um custo de capital para a organização.

O WIP refere-se a todos os materiais e produtos parcialmente acabados que estão em diferentes estágios do fluxo de desenvolvimento. Para o contexto do trabalhador do conhecimento, o WIP geralmente se traduz em uma demanda importante que foi começada, mas que ainda não está fornecendo valor para o cliente. Essencialmente, é um investimento que ainda não gera retorno e se deprecia em valor ao longo do tempo.

Mas, afinal, o que poderíamos considerar como sendo WIP dentro do desenvolvimento ágil de software?

Primeiramente, pensando em um contexto ágil, uma unidade de trabalho deve ser algo que gerará de forma direta ou indireta valor para o cliente ou usuário. Um item de trabalho pode ser uma história de usuário, um épico, uma funcionalidade, um *bug*, uma melhoria etc.

Em segundo lugar, para definirmos "em andamento", devemos considerar as fronteiras do nosso processo. Para isso, vamos usar a metáfora de um sistema de filas simples, ilustrado na figura a seguir.

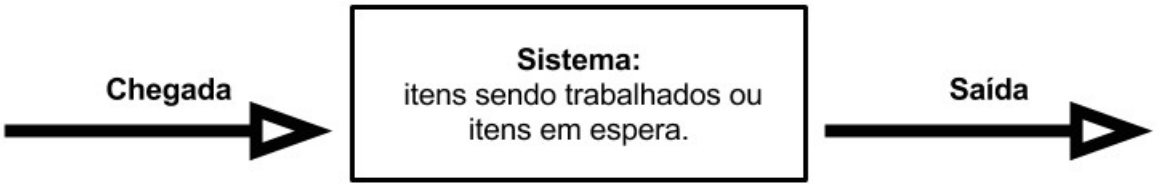


Figura 2.1: Exemplo de sistema de fila simples

Em um sistema de fila, existe o conceito de trabalho que chega e do trabalho que sai de um processo. Ao fazer uma determinação sobre algo ser considerado "em andamento" ou não, o primeiro aspecto do sistema que precisa ser considerado é: o que significa algo ter chegado?

Ou seja, o time precisa definir a partir de que momento uma ideia se transforma efetivamente em um item de trabalho. Antes desse momento, a ideia é apenas algo candidato a ser trabalhado, no entanto, após esse momento, o item passa a ser contado como WIP.

Para um item de trabalho deixar de contar como WIP, é necessário determinar um ponto de saída do processo que pode ser definido como a entrega de uma correção a um usuário final, ou a aprovação de uma funcionalidade por parte do gestor de produto.

Fazendo um paralelo com processos ágeis, o Scrum determina, na cerimônia de *Sprint Planning*, o ponto de chegada de um item de trabalho para o sistema de desenvolvimento do time. Ao término da cerimônia, os itens que foram incluídos no escopo de trabalho (*Sprint Backlog*) passarão a ser o montante de WIP que, idealmente, será finalizado no

término da iteração (*sprint*). O ponto de saída de um sistema Scrum nada mais é do que o trabalho aprovado na cerimônia de *Review*.

Dentro de uma abordagem Kanban, o ponto de partida do processo dependerá da forma como o fluxo de desenvolvimento do time está organizado. A partir do momento em que o time assume o compromisso de trabalhar com determinado item, ele passa a fazer parte do total de WIP. Já o ponto de saída pode ser definido como a entrega a um usuário final, ou a entrega a uma outra equipe ou processo.

Resumindo, independente do processo que o seu time estiver usando, só será possível identificar a quantidade de trabalho em progresso a partir do momento em que estiverem claras as etapas de entrada e saída do processo.

A definição desses dois limites do sistema é o ponto de partida na concepção de um processo que está em busca da previsibilidade de entrega e visibilidade do trabalho.

2.3 Os benefícios em limitar o trabalho em progresso

Medir o WIP é um dos trabalhos mais importantes no monitoramento da saúde do processo de desenvolvimento de software, pois, como veremos nos próximos capítulos, ele é um dos preditores do desempenho geral do fluxo de desenvolvimento. Gerenciando o WIP, garantimos um controle na cadência das entregas e impactamos o tempo que uma demanda passará pelo fluxo de desenvolvimento.

Antes de compartilhar algumas dicas de como melhorar e analisar o que passa pelo processo de desenvolvimento do seu time, gostaria de discutir a importância do monitoramento e limitação do WIP a partir de um exemplo.

Imagine que o time XPTO está em sua terceira semana de projeto e o seu fluxo de trabalho está na seguinte situação:

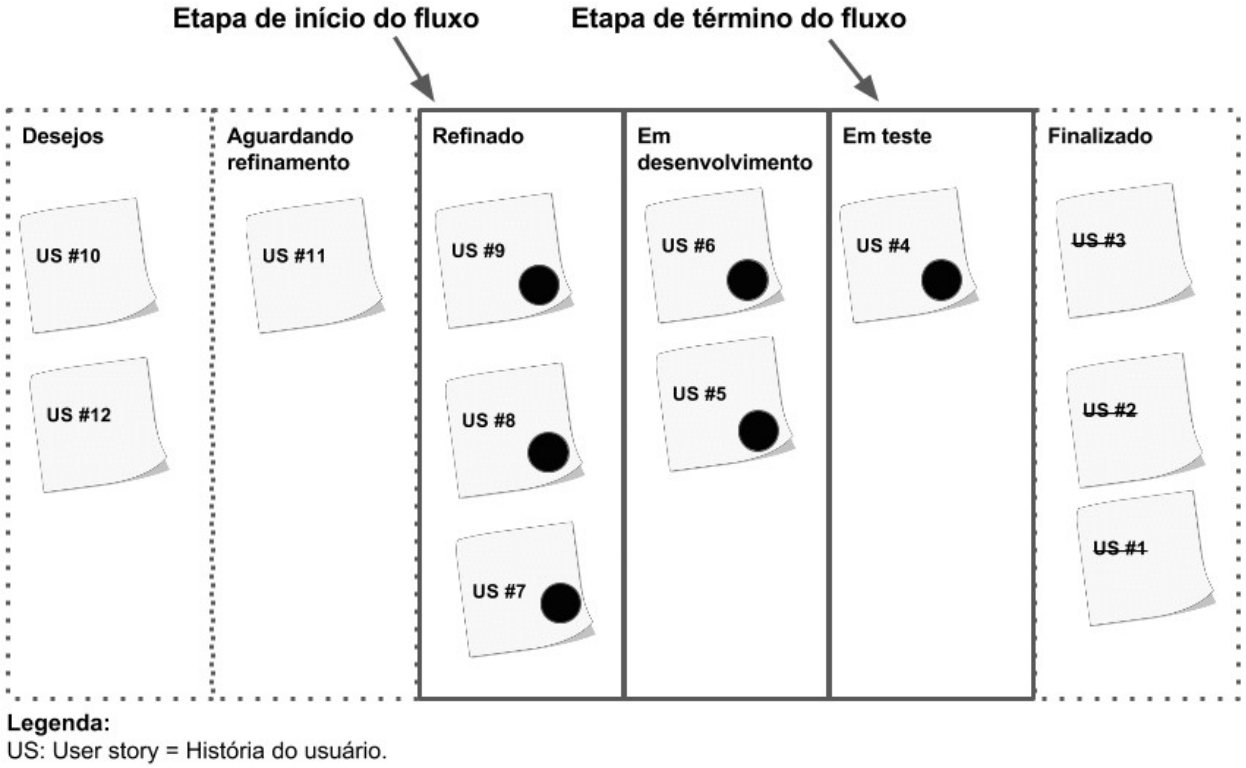


Figura 2.2: Exemplo de fluxo de trabalho do time XPTO

As etapas "Desejos" e "Aguardando refinamento" não fazem parte do início do processo, pois o custo de adicionar itens e transacioná-los entre essas fases é baixo. As etapas "Refinado" (etapa de entrada no processo), "Em desenvolvimento" e "Em teste" (etapa de saída do processo) representam o trabalho em progresso que deve ser contabilizado e monitorado.

A partir do exemplo, quantos itens o time XPTO possui em WIP no momento da análise? Somando os itens que se encontram em cada uma das etapas que consideram o trabalho em progresso, temos que o time possui seis histórias de usuário em andamento.

Uma semana se passou e o fluxo de trabalho do time XPTO encontra-se da seguinte maneira:

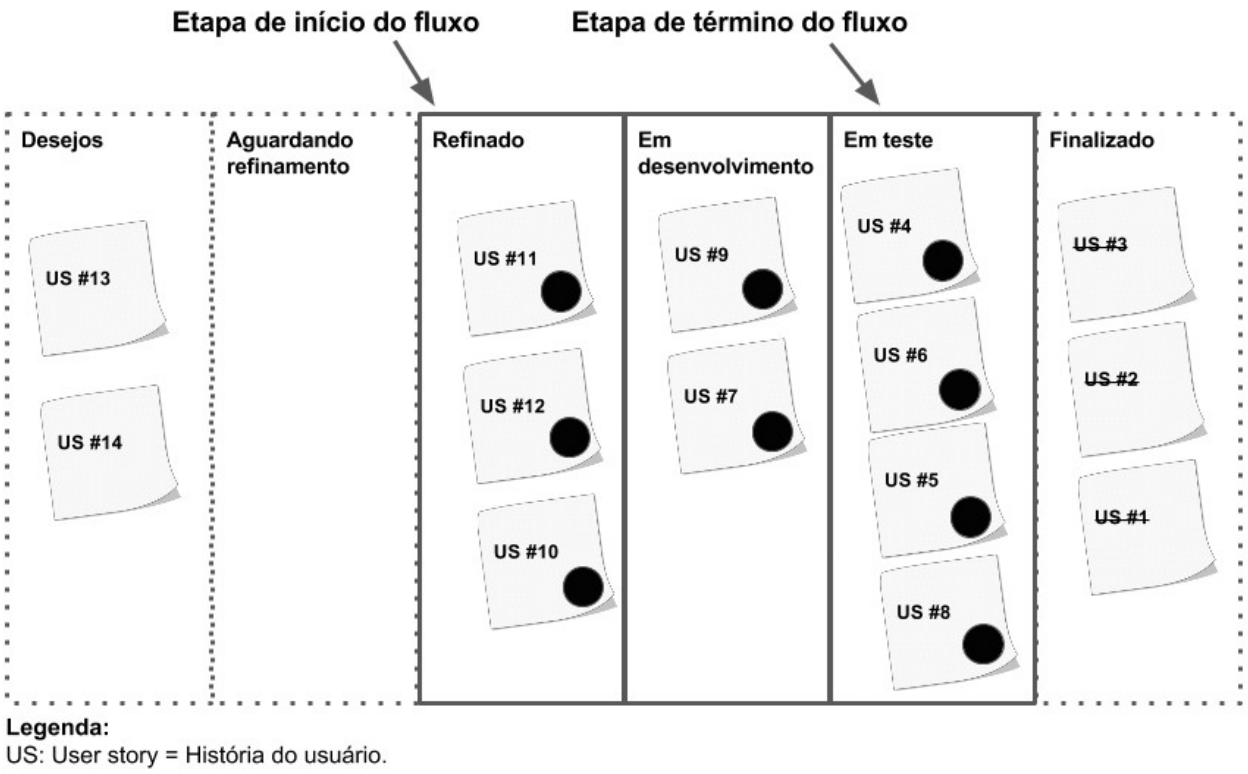


Figura 2.3: Fluxo de trabalho do time XPTO após uma semana

Podemos observar que o total de WIP do time aumentou de seis para nove histórias em um período de uma semana. Além disso, o número de demandas na fase de testes cresceu de 1 para 4. A partir dessa breve análise, quais seriam sugestões que você poderia recomendar ao time XPTO?

Se você, ao acompanhar sistematicamente o fluxo de trabalho do time (uma boa dica é fazer esse trabalho diariamente), perceber que as demandas não estão se movendo continuamente ao longo do fluxo, é bem provável que um gargalo esteja se formando em alguma etapa. No exemplo do time XPTO, percebemos que a etapa "Em testes" passou a acumular novos itens e não gerou nenhuma saída (passada uma semana, a US #4 não saiu do processo).

Alguns membros do time poderiam estar com a sensação de trabalho realizado, afinal, as etapas de "Refinado" e "Em desenvolvimento" sofreram mudanças (novos itens entraram e outros seguiram para as etapas seguintes). O aumento no WIP total do fluxo de trabalho do time nos diz que tal sensação esconde possíveis gargalos.

Algumas dicas que poderíamos compartilhar com o time são:

1. Foque os esforços para diminuir o estoque da etapa de gargalo.

2. Não aumente o WIP até que a etapa do gargalo esteja em um nível saudável, afinal, ela determinará a capacidade de vazão do processo. No caso do time XPTO, podemos perceber que a baixa vazão da etapa "Em teste" impactou o fluxo como um todo.

Outra dica que poderia ser compartilhada com o time XPTO é a de definir um limite de WIP para o fluxo de desenvolvimento. Mas por quê?

1. Podemos ter uma maior previsibilidade de qual é a capacidade de trabalho do time em qualquer espaço de tempo.

2. Reduzimos a quantidade de trabalho "quase pronto", forçando o time a se concentrar em um conjunto menor de tarefas e incentivando a cultura de "pronto".

3. Gargalos e bloqueios ficarão visíveis e serão destacados antes de uma situação se tornar difícil de ser gerenciada. Uma vez que os bloqueios são removidos, o trabalho de toda a equipe começa a fluir novamente.

4. Quanto mais trabalho colocamos dentro do fluxo de desenvolvimento do time, menor será a qualidade (ANDERSON, 2011).

Dado que dentro do desenvolvimento ágil de software temos como principal objetivo entregar valor para o cliente, limitar o WIP pode resultar a não inclusão de novas demandas no fluxo de desenvolvimento, o que levará a termos membros ociosos na equipe.

Antes de você pensar que eu esteja maluco em não querer que todas as pessoas do time fiquem 100% do tempo ocupadas, gostaria de lembrá-lo que WIP é como um inventário em um armazém: quanto mais você o carrega, mais distante do mercado você ficará e mais difícil é para você responder às mudanças.

Estimule os membros do time a enxergarem que eles não estão parados, mas sim que ganharam tempo para olhar o fluxo de desenvolvimento como um todo e que estão aptos a alocar o esforço disponível em algum lugar que esteja precisando de apoio.

Portanto, monitorar e controlar o WIP exigirão que todos do time compreendam os efeitos de suas ações para que haja cadência nas entregas, sem a geração de gargalos e estoques desnecessários ao longo do processo de desenvolvimento.

Como o Kanban e Scrum lidam com o limite de WIP?

Segundo Anderson (2011), uma das propriedades básicas do método Kanban é a de limitar o WIP. O autor defende a necessidade de criar limites explícitos para a quantidade máxima de itens que estarão em progresso para cada uma das etapas do fluxo de desenvolvimento.

Fazendo um comparativo com o Scrum, em que temos os conceitos de *sprint* (um ciclo de trabalho que pode ser de 2, 3 ou 4 semanas) e *Sprint Backlog* (lista de demandas que o time se compromete a fazer no intervalo de uma *sprint*), percebemos que não existe uma regra dizendo que o time não possa trabalhar com todas as demandas do *Sprint Backlog* ao mesmo tempo. Porém, dado que a *sprint* possui um escopo fixo, há um limite implícito.

Podemos concluir que o Scrum limita o WIP indiretamente, enquanto o Kanban limita o WIP diretamente. Além disso, podemos dizer que o Scrum limita o WIP por unidade de tempo, enquanto o Kanban limita o WIP por etapa do fluxo de trabalho.

Mesmo não sendo uma regra do *framework*, vejo que a maioria das equipes Scrum aprende que não é uma boa ideia ter muitas demandas em aberto e a desenvolver uma cultura que preze a conclusão de uma demanda antes de iniciar uma nova.

No fundo, tais equipes estão trazendo para o seu processo a filosofia de limitar o trabalho em progresso, seja por etapa ou pelo fluxo de trabalho como um todo.

2.4 Como analisar e melhorar o WIP a partir de um caso real

Nesta seção, teremos a oportunidade de analisar a importância do monitoramento do WIP e discutir técnicas para se aprimorar o trabalho em progresso de um time de desenvolvimento de software.

Os tópicos que acompanharão o caso terão dicas para que você melhore a qualidade daquilo que passa pelo processo de desenvolvimento do time do qual você faz parte.

Estudo de caso — Criação das versões digital e móvel de um jornal impresso

O caso se passou em uma equipe que trabalhou no desenvolvimento de um portal de notícias para um jornal de grande circulação regional. O projeto foi realizado em um momento que a empresa detentora do periódico investia na digitalização do seu conteúdo através do desenvolvimento de um portal de notícias e sua versão para dispositivos móveis, bem como a criação de um aplicativo em formato de banca digital para *tablets*.

A empresa passava por um momento em que era preciso repensar sua estratégia, pois o mercado de jornal impresso estava perdendo espaço para os conteúdos advindos da internet. Outro fator que impulsionou a digitalização foi que os jornais concorrentes começavam a lançar suas versões digitais.

O time teria quatro meses para lançar o projeto do portal e sua versão móvel, afinal, a diretoria gostaria de aproveitar a data comemorativa de aniversário do jornal para divulgar a novidade. Todo o trabalho de produção do conteúdo seria conduzido pelo time de redatores que estariam dedicados ao projeto.

O sistema responsável pela gestão do conteúdo do portal já estava pronto para uso e não fazia parte do escopo de desenvolvimento da construção do portal. No entanto, algumas integrações se fariam necessárias para que o portal tivesse o seu conteúdo atualizado de forma dinâmica.

A equipe que trabalharia no desenvolvimento era composta por: 3 desenvolvedores; 1 Agile Coach; 1 testador; 1 designer de interface e 1 gestor de produto.

O time decidiu trabalhar com o método Scrum a partir de iterações (*sprints*) quinzenais, e incorporou para o seu processo de desenvolvimento as cerimônias de:

- **Sprint Planning**: onde eram priorizadas e discutidas as funcionalidades que entrariam no fluxo de desenvolvimento;
- **Sprint Retrospective**: onde eram discutidas melhorias de processo;
- **Sprint Review**: onde eram apresentadas as entregas da quinzena para o gestor de produto;
- **Reunião diária**: onde o time se mantinha atualizado da situação do trabalho em progresso e de possíveis bloqueios existentes que precisariam ser tratados em um momento futuro;
- **Refinamento**: onde eram avaliadas as incertezas, complexidades, dependências e critérios de aceite das funcionalidades que seriam trazidas para o fluxo de desenvolvimento.

Diferente de outras aplicações de Scrum que você possa ter visto, este time não estimava o esforço do trabalho que seria desenvolvido através do método de *story points*. O que motivou o time a não utilizá-lo foi o baixo valor que os membros viam em estimar as demandas que passavam pelo fluxo de desenvolvimento a partir de uma unidade de medida qualitativa (pontos).

Além disso, como veremos a seguir, foram empregadas algumas técnicas que ajudaram o time a reduzir as incertezas e complexidades das funcionalidades que seriam trabalhadas. Portanto, fazia pouco sentido saber se uma funcionalidade tinha como estimativa de esforço 1 ou 13 pontos.

STORY POINTS

> Segundo Cohn (2006), *story point* é uma medida arbitrária usada por times na medição do esforço necessário para desenvolver determinado requisito, funcionalidade, história do usuário, *bug* etc. Usualmente, os times que utilizam tal medida se baseiam na sequência de Fibonacci (1, 2, 3, 5, 8, 13 etc.) para atribuir o esforço de trabalho necessário para implementar uma demanda.
>
> É possível definir que *story point* nada mais é do que um número que nos diz quão difícil uma demanda é. Cabe ressaltar que a definição de difícil pode estar relacionada a complexidade, incertezas e esforço. Como *story point* é uma métrica relativa, não é possível correlacioná-la com horas de trabalho (métrica absoluta).
>
> Por fim, para cada time, o tamanho de um *story point* terá um significado diferente dependendo da base de referência escolhida. Portanto, tenha muito cuidado ao comparar equipes que usam tal técnica para estimar esforço, pois você estará comparando abacaxi com banana.

O fluxo de trabalho do time estava dividido nas seguintes etapas:

- **Em refinamento**: etapa em que as demandas eram discutidas, o design da interface era desenhado e os critérios de aceite eram definidos.
- **Pronta para desenvolvimento**: etapa na qual as demandas que estavam refinadas e priorizadas entravam para o fluxo de desenvolvimento do time. A definição das demandas era feita na cerimônia de *Sprint Planning*.
- **Em codificação**: etapa na qual os desenvolvedores criavam e revisavam o código que havia sido produzido, a fim de garantir qualidade naquilo que estava sendo construído.
- **Aguardando testes**: etapa onde os desenvolvedores liberavam as funcionalidades criadas para a validação do testador que estava no time.
- **Em testes**: etapa em que eram testadas as funcionalidades no ambiente de homologação.
- **Em aprovação**: etapa onde o gestor de produto validava se as funcionalidades atingiam o objetivo de negócio que havia sido definido. Tal ação ocorria quinzenalmente na cerimônia de *Sprint Review*.
- **Finalizado**: etapa de publicação das funcionalidades aprovadas para o ambiente de produção.

O time definiu que a métrica de WIP seria contabilizada a partir dos estágios "Pronta para desenvolvimento" e "Em aprovação".

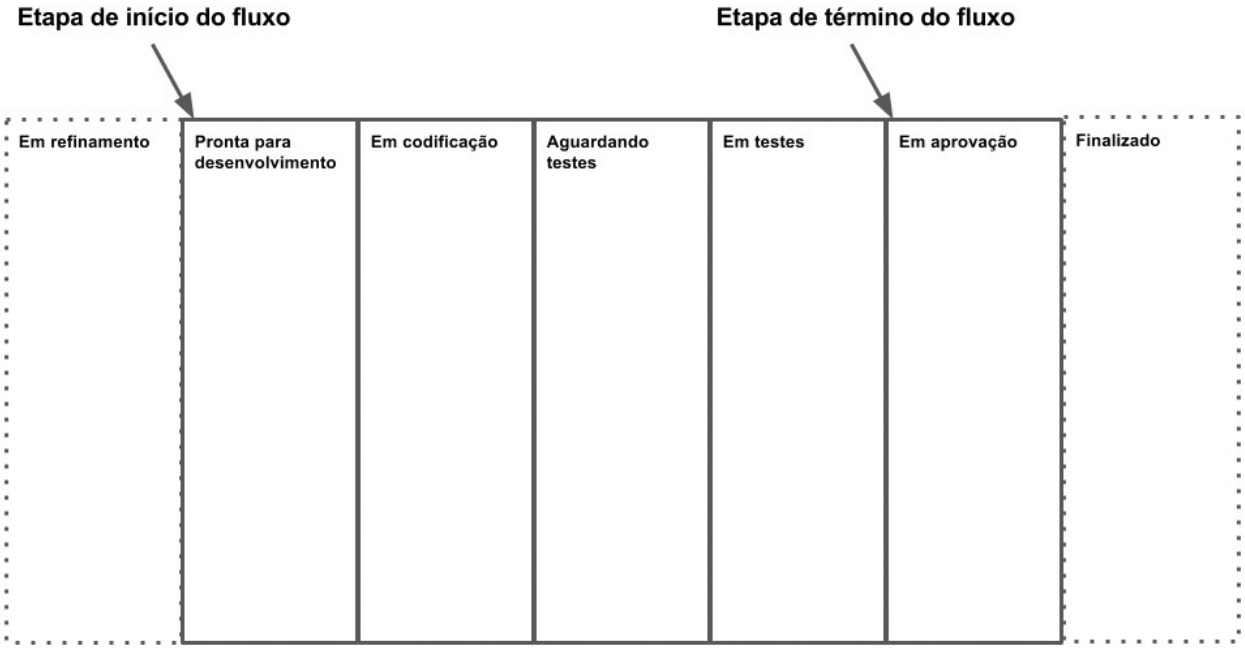


Figura 2.4: Estudo de caso: fluxo de trabalho do time

Após uma sessão de *story mapping*, foram identificadas e definidas as funcionalidades que seriam parte das versões digital e móvel do portal que deveriam ser publicadas no dia de aniversário do jornal. O time sabia que o desafio era grande, dado a quantidade de trabalho a ser feito (aproximadamente 30 histórias de usuário, ou seja, seriam necessárias entregas de quatro histórias por *sprint* até a data do lançamento).

Além disso, a dependência de uma integração com um terceiro, o desenvolvimento de dois produtos (versões móvel e digital) e o prazo fixo eram ameaças que o time teria de lidar durante o desenvolvimento do projeto.

USER STORY MAPPPING

Jeff Patton define que o Story Mapping é uma técnica colaborativa que auxilia na priorização e no planejamento de releases de produtos interativos.

Mais detalhes sobre assunto podem ser lidos em Patton (2008).

Primeiro mês — WIP como suporte na identificação da baixa qualidade das demandas que entravam no fluxo

Nas duas primeiras *sprints*, monitorar o WIP foi fundamental para evidenciar ao time que existiam problemas no fluxo de desenvolvimento, pois o número de trabalho em progresso não diminuiu, pior, só aumentou.

Vejamos a situação do quadro de trabalho do time para entendermos o que aconteceu:

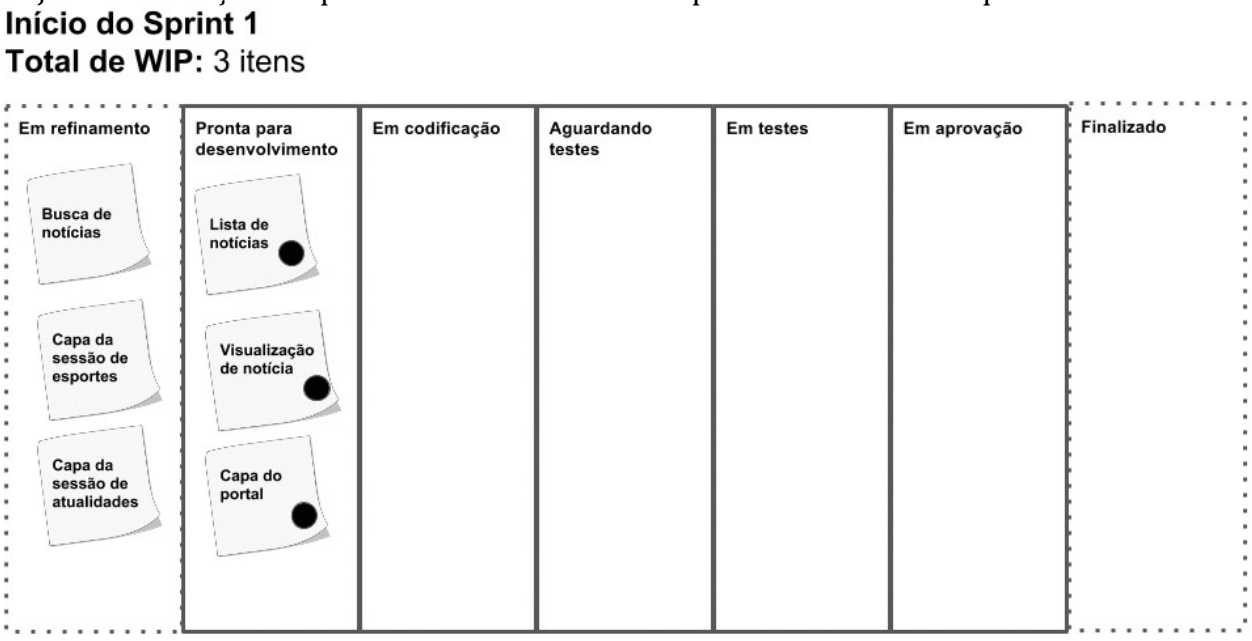


Figura 2.5: Estudo de caso: início da primeira sprint

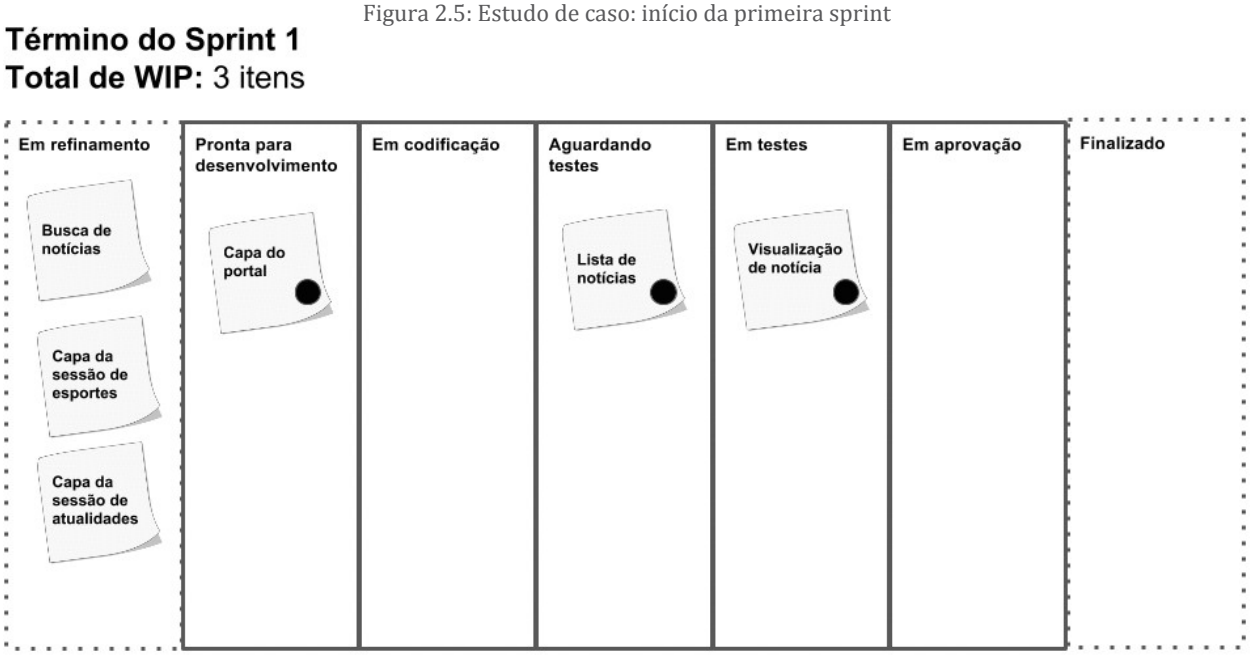


Figura 2.6: Estudo de caso: fim da primeira sprint

Na primeira *sprint*, o time se comprometeu com o gestor de produto a entregar três funcionalidades. No entanto, ao final do período nenhuma delas foi entregue.

Na cerimônia de retrospectiva, todos consideraram tal comportamento (nenhuma entrega) normal, afinal, o time estava se familiarizando com o escopo do projeto, algumas automatizações estavam sendo finalizadas e os ambientes de desenvolvimento, homologação e produção estavam sofrendo ajustes.

No planejamento da segunda *sprint*, o gestor de produto, pressionado pela diretoria a dar visibilidade sobre o progresso do projeto, convenceu o time a incluir mais duas funcionalidades que, se entregues nos quinze dias seguintes, acalmariam a ansiedade das pessoas que estavam acompanhando o projeto.

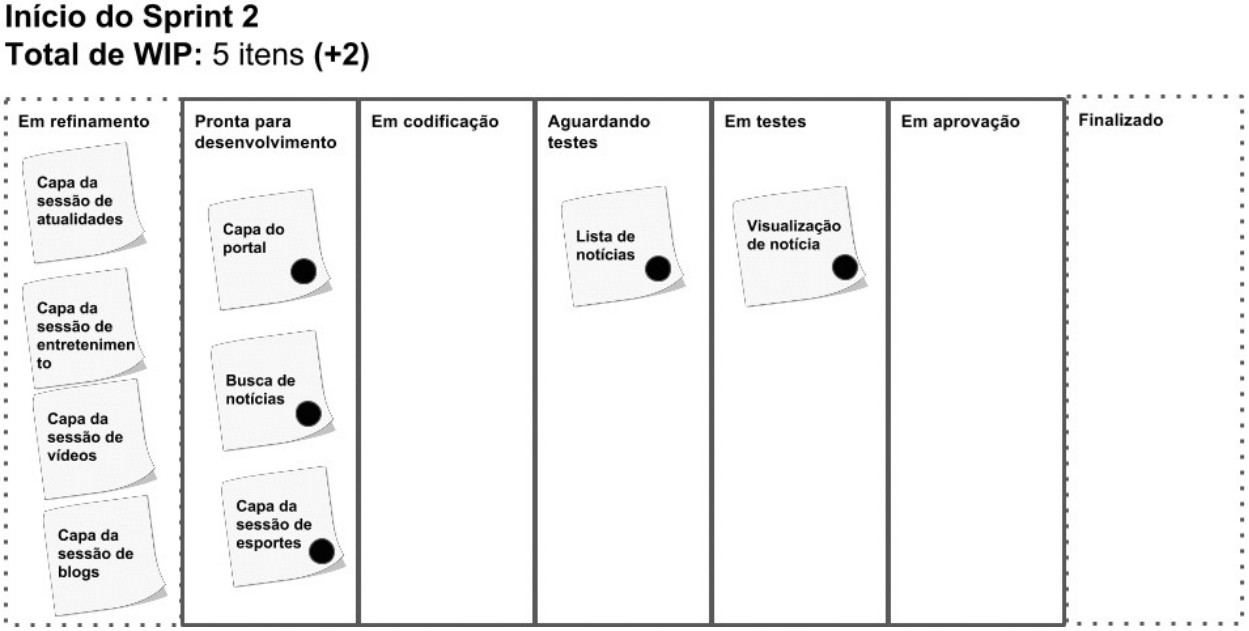


Figura 2.7: Estudo de caso: início da segunda sprint

Chegado o fim da segunda *sprint*, o time se viu em uma situação onde:

1. O tamanho do WIP não diminuíra, agora eram cinco funcionalidades que não haviam sido finalizadas;

2. Faltava apenas três meses para o prazo de entrega do projeto;

3. A equipe se sentia frustada, pois não conseguia gerar entregas concretas;

4. O time de redatores começava a ficar impaciente para ver o conteúdo que estava sendo produzido refletido na interface do portal;

5. O gestor de produto estava recebendo uma forte pressão pela falta de resultado produzido pelo time.

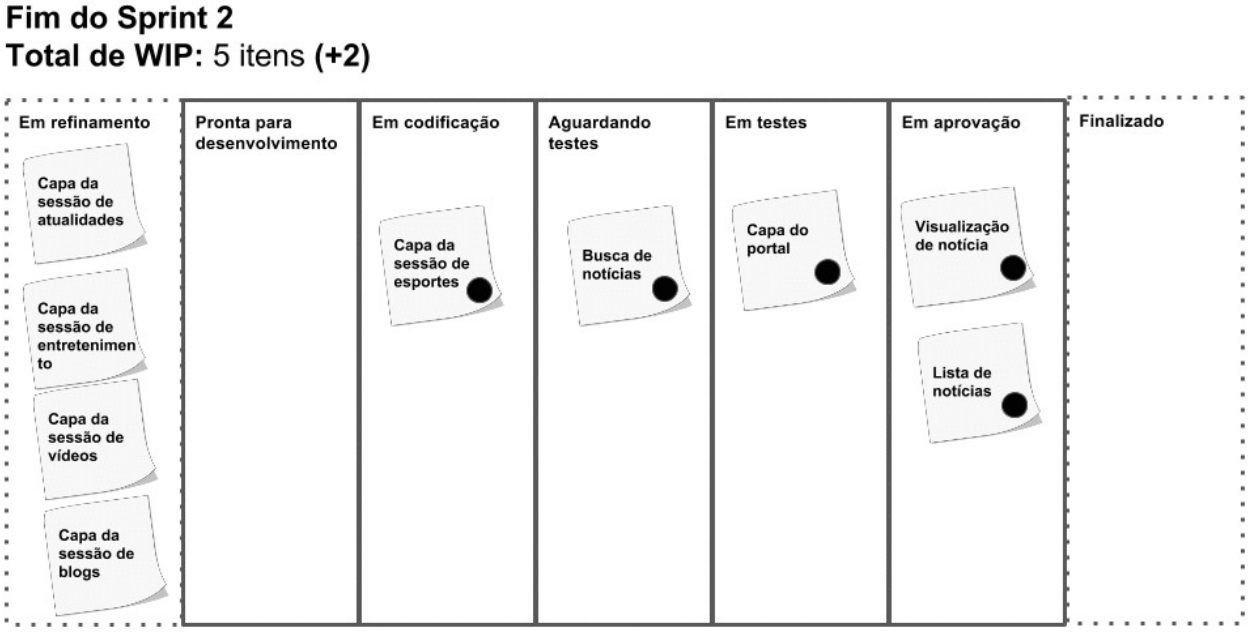


Figura 2.8: Estudo de caso: fim da segunda sprint

A partir de uma retrospectiva que teve como objetivo discutir justamente os motivos do baixo volume de entrega e, principalmente, definir ações para se reverter o cenário, o time constatou que a cerimônia de refinamento não estava sendo realizada.

Outro ponto levantado foi de que as funcionalidades que entravam para o fluxo de desenvolvimento estavam:

- Mal definidas;
- Não possuíam uma avaliação de complexidade;
- Geravam dúvidas sobre as regras de negócio que deveriam ser implementadas; e
- Estavam gerando um alto número de retrabalho.

Um exemplo que o time usou para constatar os problemas foi o caso no qual os componentes visuais da capa do portal e a lógica de disposição do conteúdo só foram definidos quando estavam na etapa de teste.

Para que você não sofra dos problemas vividos pelo time do estudo de caso, gostaria de compartilhar contigo três estratégias para melhorar a qualidade de uma demanda antes que ela se transforme em um trabalho em progresso.

A importância de se ter um momento para refinar as demandas

A cerimônia de refinamento (do inglês *Refining*, ou também conhecida como *Grooming*) consiste em discutir de forma detalhada as demandas que serão trabalhadas em um futuro próximo pelo time. O principal objetivo desta cerimônia é fazer com que os membros do time compreendam a necessidade e o que deve ser feito para cada demanda que entrará no fluxo de desenvolvimento.

O refinamento traz como principais benefícios um melhor alinhamento de expectativas entre os membros do time com relação à solução e aos fatores que impactam a entrega das demandas.

Um ponto que vale a pena compartilhar é que talvez o time no qual você faça parte não necessite de uma cerimônia na agenda para discutir sobre as demandas. O mais importante é que esteja presente a cultura de preparar as demandas antes que elas comecem a ser trabalhadas, para que o fluxo de desenvolvimento não perca eficiência e nem velocidade de entrega.

Crie uma definição de pronto para que a demanda entre no fluxo de desenvolvimento

Dentro de times ágeis, você já deve ter ouvido falar do conceito de "*ready for development*", "*ready to dev*" ou "*definition of ready*". Particularmente, gosto da definição de Pichler (2010) sobre o que caracteriza uma demanda pronta para ser trabalhada. O autor diz que qualquer demanda, para ser considerada pronta para ser desenvolvida, deve ser clara, confiável e testável.

Uma demanda clara é aquela em que há um entendimento do seu significado por todos do time. Descrever de forma colaborativa os objetivos das demandas e os respectivos critérios de aceite são formas de auxiliar na clareza.

Para que uma demanda seja testável, é necessário que exista um meio efetivo de determinar se a funcionalidade que será construída por ela atenderá o que é esperado. Para isso, é fundamental a descrição de critérios de aceite que permitam a verificação e validação do que será construído.

Por fim, uma demanda é confiável quando ela pode ser completada dentro de um espaço de tempo (no caso do Scrum, dentro de uma *sprint*). Isso implica que as demandas devem ser pequenas o suficiente e não devem ser demasiadamente complexas.

Se você busca que o seu time não inicie o desenvolvimento de uma demanda mal definida, crie uma política que descreva quais são as características necessárias para que algo entre no fluxo de desenvolvimento.

Você pode estar se perguntando: legal, como eu poderia avaliar se uma demanda está pronta ou não? Compartilho a seguir algumas questões e critérios que costumo usar junto dos times no processo de avaliação.

Questões orientadoras

Qual o problema a ser resolvido?

Qual o resultado esperado?

Existe alguma dependência de negócio ou técnica para esta demanda?

No caso de times que trabalham com interface, a demanda depende de alguma definição visual?

Qual a relevância/prioridade do problema para a área de negócio?

Quais são os critérios de aceite que garantirão a verificação e a validação da demanda?

Existe algum requisito não funcional (exemplo: performance, design) que deve ser atendido?

Critérios para definir se uma demanda está pronta para ser trabalhada

Demanda definida e escrita.

Critérios de aceite definidos e compreendidos por todos do time.

Dependências identificadas.

Informações de interface definidas (quando necessário).

Critérios de desempenho definidos (quando necessário).

Existência de um alinhamento entre as partes que construirão e avaliarão a demanda.

Garantir que as demandas estejam claras e prontas para serem desenvolvidas fará com que o time seja eficiente ao tratá-las ao longo do fluxo de desenvolvimento.

COMO O CONCEITO INVEST PODE AJUDAR NA CLAREZA DAS DEMANDAS?

Mike Cohn, em seu livro **User Stories applied**, define que uma história de usuário com qualidade deve seguir os critérios agrupados no acrônimo INVEST. O autor defende que, se a história falhar em algum dos critérios, o time deverá repensá-la ou até mesmo reescrevê-la. Uma boa história de usuário deve ser:

Independent **(Independente):** histórias devem ser independentes uma das outras sempre que possível, para que seja fácil priorizá-las.

Negotiable **(Negociável):** histórias não são contratos para implementar funcionalidades. Boas histórias captam a essência e não os detalhes.

Valuable **(Tenha valor):** se uma história não tem valor perceptível, ela não deve ser feita.

Estimable **(Estimável):** toda história tem de ser estimável ou dimensionável, de modo que possa ser convenientemente priorizada.

Small **(Pequena):** boas histórias são pequenas, pois são mais fáceis de se estimar e planejar.

Testable **(Testável):** os testes demonstram que uma história alcançou as expectativas dos usuários/clientes.

Os critérios apresentados anteriormente podem ser úteis no caso do seu time estar se familiarizando com a criação e definição das demandas que farão parte do fluxo de desenvolvimento.

Analisar a complexidade e a incerteza das demandas

Se o time no qual você trabalha lida com demandas que possuem uma alta variabilidade no tempo que elas passam pelo fluxo de desenvolvimento — isto é, existem funcionalidades que são entregues rápido demais e outras que levam muito tempo para serem concluídas —, é provável que você esteja em um ambiente com baixa previsibilidade de entrega e um cenário perfeito para a geração de estoque de WIP.

Como última dica, gostaria de compartilhar uma forma para que os times avaliem quaisquer demandas a partir das perspectivas de incerteza e complexidade (técnica e de

negócios), com o objetivo de padronizar os respectivos esforços que serão necessários para a implementação delas.

Essa tem sido uma abordagem que venho propondo como substituição ao *planning poker* e a estimativa através de *story points*. A intenção é que o processo de "estimar" o esforço de uma demanda seja rápido e, principalmente, leve em consideração aspectos mais concretos, como, por exemplo, incertezas e complexidades.

Antes de descrevê-la, gostaria de agradecer ao Diego Poblete (*@dipoblete*) por ter me apresentado sua primeira versão.

O QUE É O PLANNING POKER?

Segundo Cohn (2006), o *planning poker* é uma forma de estimativa em conjunto, que pode ser feita como um jogo. Todos os membros do time participam democraticamente, com o objetivo de chegar a um consenso de estimativa, em *story points*, para cada item que será trabalhado no fluxo de desenvolvimento.

Para iniciar uma sessão de *planning poker*, o gestor do produto ou o cliente lê a demanda para as pessoas do time que farão a estimativa. Cada membro do time terá em mãos um conjunto de cartas com os valores como 1, 2, 3, 5, 8, 13 etc. Os valores representam o número de *story points* que o time utiliza como unidade para estimar.

Após a apresentação, os membros discutirão a demanda, questionando o cliente ou gestor de produtos quando necessário. Após uma demanda ser totalmente analisada, cada membro, individualmente, selecionará um cartão que representará a sua estimativa. Na sequência, todos os cartões são exibidos ao mesmo tempo.

Se todos os membros atribuírem o mesmo valor para a demanda, aquela se tornará a estimativa. Caso contrário, os membros do time discutirão as respectivas escolhas. Aqueles que deram o valor mais alto e o mais baixo deverão compartilhar os motivos.

Depois da discussão, cada membro selecionará novamente um cartão, e todos abrirão novamente suas estimativas para análise. O processo de *planning poker* se repete até que um consenso seja atingido, ou que os membros decidam que a demanda que está sendo estimada requer maiores informações para ser avaliada.

A matriz de complexidade está estruturada em dois eixos que combinam as análises de complexidade e incerteza de cada demanda que será trabalhada. O eixo X da matriz representa o quanto de incerteza técnica e de negócio a demanda que está sendo avaliada possui. Já o eixo Y analisa a complexidade e o esforço para se entregar a demanda.

A categoria pequena, no eixo X, representa se a demanda está bem definida e sem incertezas. No eixo Y, ela representa que a demanda exige pouco esforço e tem baixa complexidade na sua implementação.

A categoria média, no eixo X, representa que a demanda tem poucas incertezas sobre sua implementação ou definição de negócio, e estas incertezas o time sabe ou já resolveu antes. No eixo Y, tal categoria representa que a demanda exigirá um esforço considerável para ser completada, mas que o time possui o domínio para fazê-lo (exemplo: funcionalidades que exijam integração com sistemas satélites).

Por fim, a categoria grande, no eixo X, representa que a demanda possui incertezas relevantes sobre a implementação ou sobre o negócio, o que traz insegurança para os membros do time por se tratar de algo onde há um baixo entendimento de contexto. No eixo Y, a atribuição grande representa que uma demanda é muito complexa, ou que precisará de muito esforço para ser concluída.

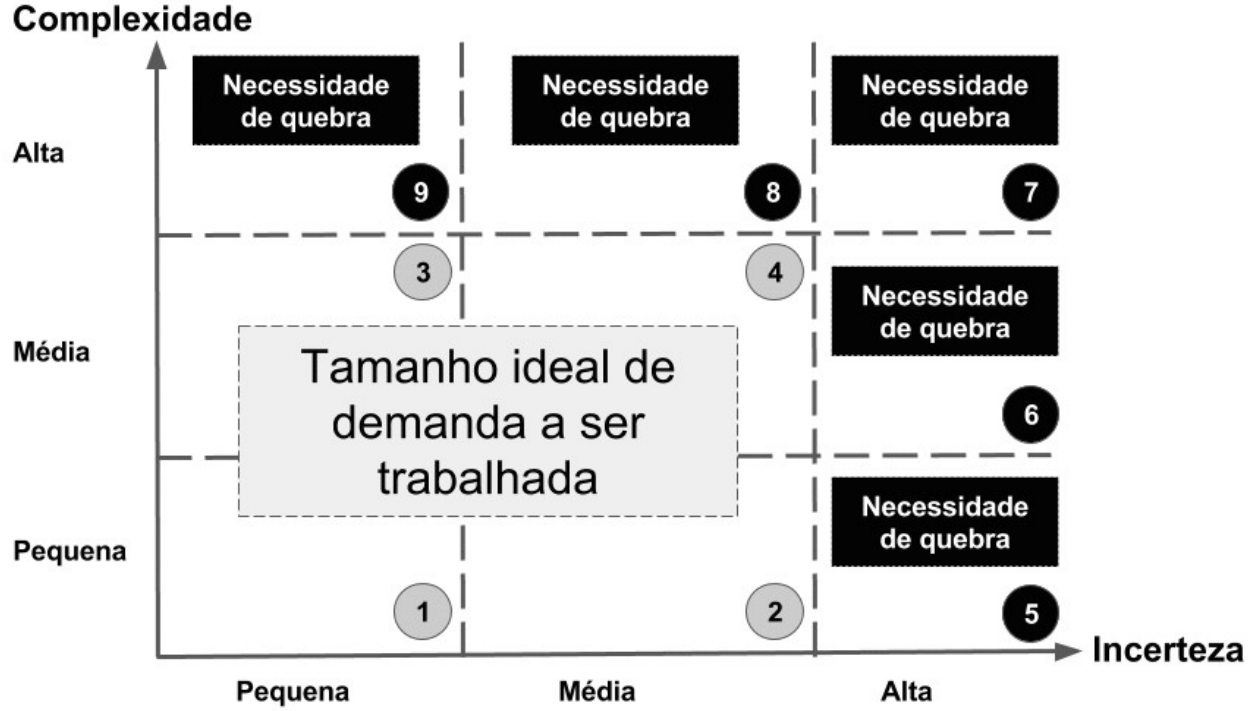


Figura 2.9: Matriz de complexidade e incerteza

As demandas classificadas nos quadrantes 9, 8, 7, 6 e 5 devem ser discutidas e dificilmente estarão prontas para entrarem no fluxo de desenvolvimento do time. O motivo? Carregam um alto grau de incerteza que naturalmente acarretará retrabalho, atraso na entrega e outros efeitos não desejados quando estamos em busca de entregas constantes e previsíveis.

Listo a seguir algumas dicas para que seu time evite trazer para o fluxo de desenvolvimento demandas com alto índice de incerteza:

1. Faça um bom trabalho de refinamento e discuta o quão pronta a demanda está para entrar no fluxo de desenvolvimento (leia as dicas anteriores).

2. Caso a demanda carregue muita incerteza, crie estratégias de investigação para que o time ganhe conhecimento técnico ou de negócio. Em determinados momentos, é melhor fazer com que as pessoas tenham um tempo para avaliar e estudar em vez de incluir uma demanda para ser desenvolvida a qualquer custo.

3. Caso uma demanda tenha um excesso de incerteza de negócio, busque especialistas para sanarem as dúvidas. Mesmo por vezes não conhecendo do linguajar técnico, tais pessoas são extremamente úteis para descrever o mundo no qual a solução será útil, e tal intercâmbio é saudável para que o time evolua.

4. Exercite a quebra de grandes demandas em pequenas, mesmo que você tenha de lidar com questionamentos de que pequenas entregas não geram valor ao negócio. Quando quebramos a incerteza, estamos reduzindo risco, o que, para mim, representa valor, pois eliminamos desperdício, provamos conceitos abstratos mais rápido e validamos hipóteses complexas fracionadamente.

As demandas classificadas nos quadrantes 1, 2, 3 e 4 geralmente estarão prontas para entrar no fluxo de desenvolvimento, pois terão tamanhos saudáveis de esforço/complexidade e estarão com baixa ou nenhuma incerteza.

Segundo mês — WIP como forma de identificar gargalos

Nas duas *sprints* seguintes, o monitoramento do WIP foi importante para que o time do estudo de caso pudesse analisar e melhorar a capacidade de demandas suportada pelo fluxo de desenvolvimento.

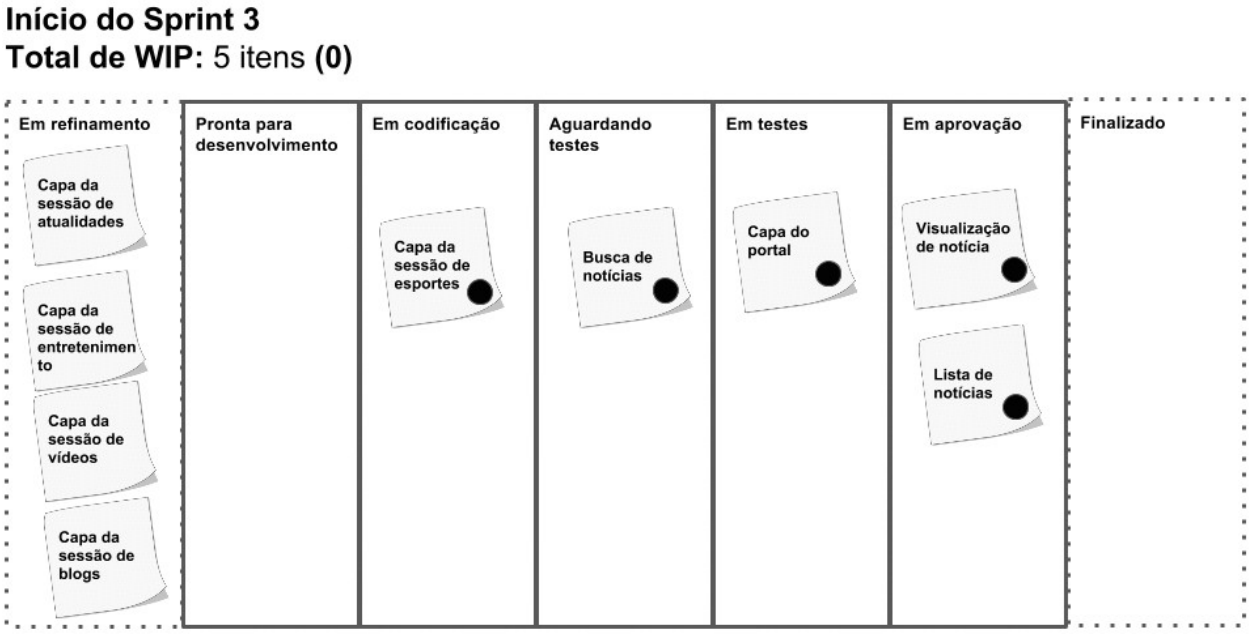


Figura 2.10: Estudo de caso: início da terceira sprint

A terceira *sprint* começou com uma decisão tomada em conjunto: o time não incluiria nenhum novo trabalho no fluxo de desenvolvimento.

O objetivo das próximas duas semanas era conseguir remover o trabalho que estava estocado em progresso, a fim de diminuir o estresse que estava sendo gerado pela expectativa e ansiedade das partes interessadas no resultado do projeto.

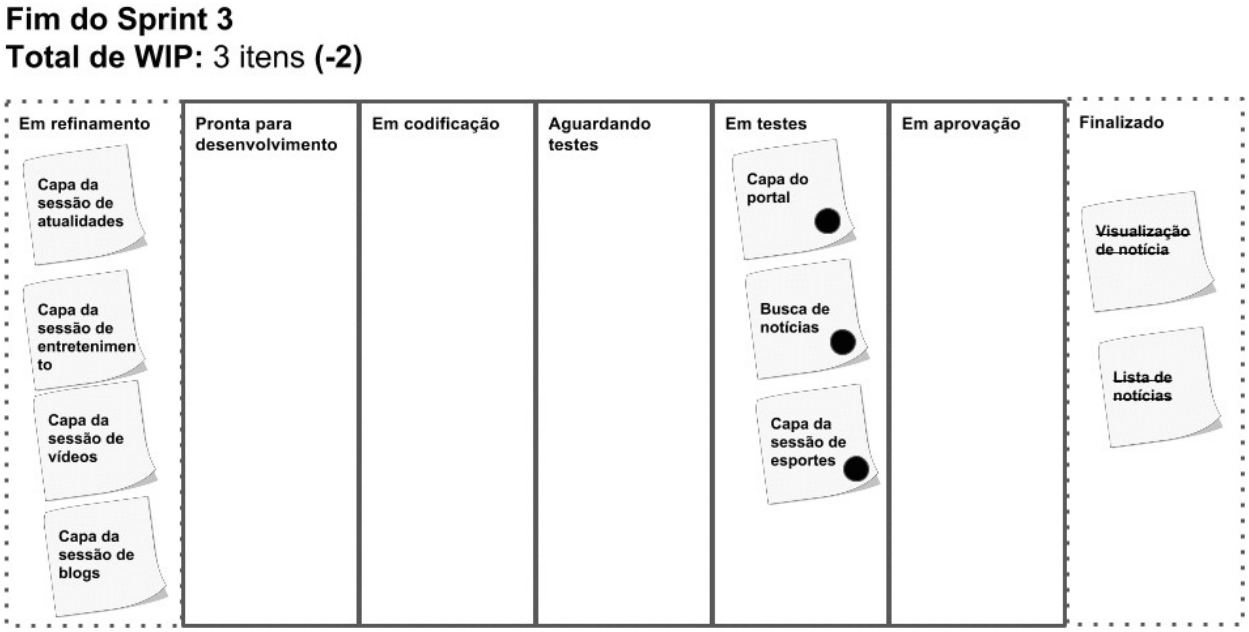


Figura 2.11: Estudo de caso: fim da terceira sprint

Ao final da *sprint*, o time conseguiu entregar duas funcionalidades em produção. Tal marco foi comemorado por todos já que trouxe a esperança de que, a partir daquele momento, o time passaria a ter uma cadência de entrega. O ponto negativo da euforia gerada foi de que o time negligenciou o motivo do estoque de WIP não ter sido limpado por completo.

Como existia um conjunto de funcionalidades que já haviam sido discutidas, refinadas e estavam prontas para serem desenvolvidas, o gestor de produto e o time de desenvolvimento decidiram, em comum acordo, trazê-las para a *sprint* seguinte.

Um dos argumentos usados pelo time para aumentar o WIP foi de que as funcionalidades que haviam ficado estocadas na *sprint* 3 estavam na fase de testes e não demorariam muito para seguirem adiante (etapa de aprovação). E assim começou a quarta *sprint*.

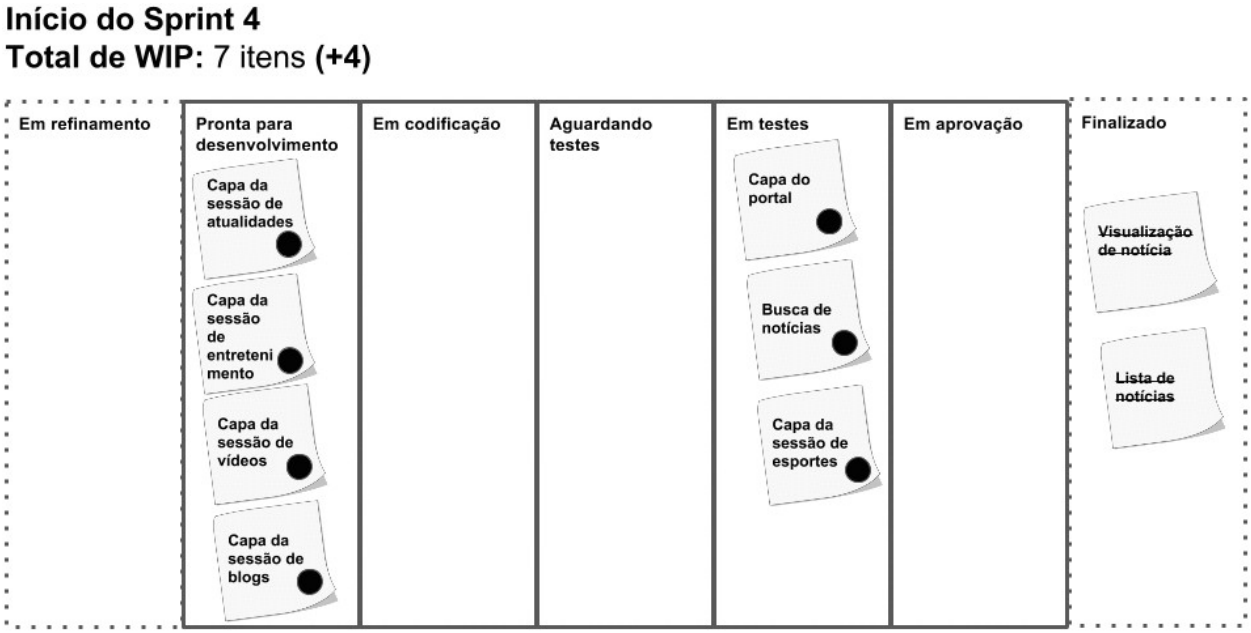


Figura 2.12: Estudo de caso: início da quarta sprint

Os dias foram se passando e o time de programadores trabalhava a todo vapor no desenvolvimento das funcionalidades. Eles estavam bem empolgados, afinal, haviam criado uma arquitetura que poderia ser reutilizada e gerava um aumento na produtividade do que precisava ser construído.

O testador convivia com alguma indisponibilidade no ambiente de homologação, pois volta e meia este ficava fora do ar para atualizações. Além disso, problemas na integração com o sistema de gestão de conteúdo faziam com que os testes atrasassem.

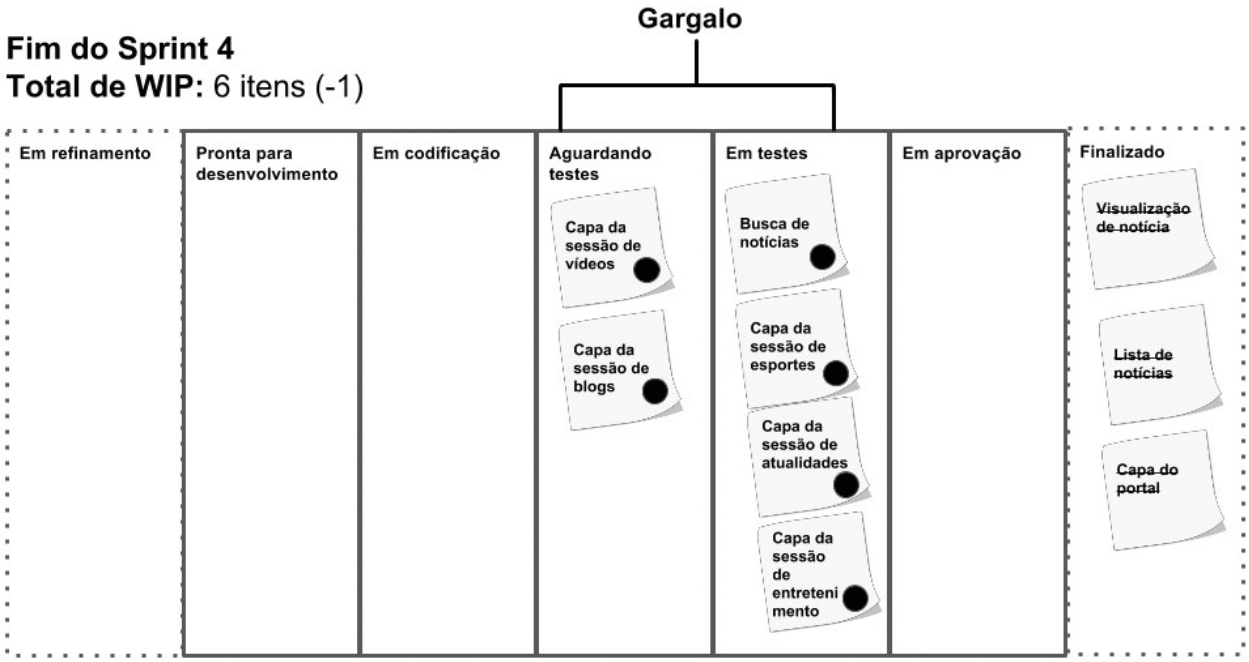


Figura 2.13: Estudo de caso: fim da quarta sprint

O final da *sprint* chegou e apenas uma nova funcionalidade foi entregue. Novamente o sentimento de insegurança quanto ao atingimento do prazo de entrega pairava nos arredores do time, e mais pressão era depositada nos ombros do gestor de produto.

Após uma reflexão sobre o fluxo como um todo, o time chegou à conclusão de que um gargalo estava se formando nas etapas que envolviam o trabalho de verificação e validação das funcionalidades. A vazão dos testes limitava a capacidade de entrega do time.

A partir do exemplo do estudo de caso, podemos observar que, na terceira *sprint*, um gargalo se formava na etapa "Em teste". E se o time tivesse o hábito de monitorar o WIP, tal problema teria sido evidenciado antes.

Identifique rapidamente os gargalos no fluxo de desenvolvimento

Caso o seu time perceba a formação de um gargalo, foque todos os esforços para removê-lo, estabilizando o sistema de trabalho novamente. Uma estratégia que pode ser utilizada e que o time do estudo de caso aplicou foi de analisar o fluxo de desenvolvimento como um todo e lê-lo da direita para esquerda.

O objetivo dessa abordagem é tentar identificar qualquer tipo de bloqueio que possa estar atrapalhando que uma demanda siga no fluxo. No estudo de caso, o time percebeu que a

instabilidade do sistema e a lentidão nas integrações com o sistema de gestão de conteúdo retardavam a capacidade de o testador trabalhar.

A lição que gostaria de compartilhar contigo é: monitore o WIP para evitar gargalos no sistema. Finalizo lembrando que gargalos no fluxo de desenvolvimento trarão:

- Falsa sensação de progresso (provavelmente os programadores do time do estudo de caso achavam que estavam sendo muito produtivos, já que as funcionalidades estavam codificadas);
- Ineficiência no sistema (o testador estava lidando com uma sobrecarga de demandas);
- Demora no tempo de entrega de uma demanda;
- Redução no número de entregas.

Terceiro mês — Estabelecer limites de WIP

Você já deve ter ouvido falar que o método Kanban criado por Anderson (2011) tem como um dos fundamentos a ideia de limitação do WIP ou do trabalho que será processado pelo fluxo de trabalho de um time. No último tópico do estudo de caso, gostaria de compartilhar os benefícios de se limitar o WIP e algumas dicas de como fazer.

Antes de seguir, gostaria de adiantar que não existe uma fórmula mágica para se definir o quanto um time pode processar em seu sistema de trabalho como um todo ou em partes dele. Até mesmo a literatura no assunto é limitada, no entanto, gostaria de deixar como dica a leitura do post **The Kanban Story: Kanban Board**, em que Brodzinski (2009) cita o empirismo (mais conhecido como "teste e adeque") como melhor forma de se limitar o WIP.

Voltando ao estudo de caso, o time percebeu novamente que precisava estabilizar o processo, removendo o gargalo que havia sido evidenciado pela etapa de teste.

Após ótimas discussões de como tratar o problema no processo, ficou decidido que a *sprint* 5 teria como objetivo limpar o estoque de WIP, para que os resultados pudessem ser apresentados para os envolvidos no projeto.

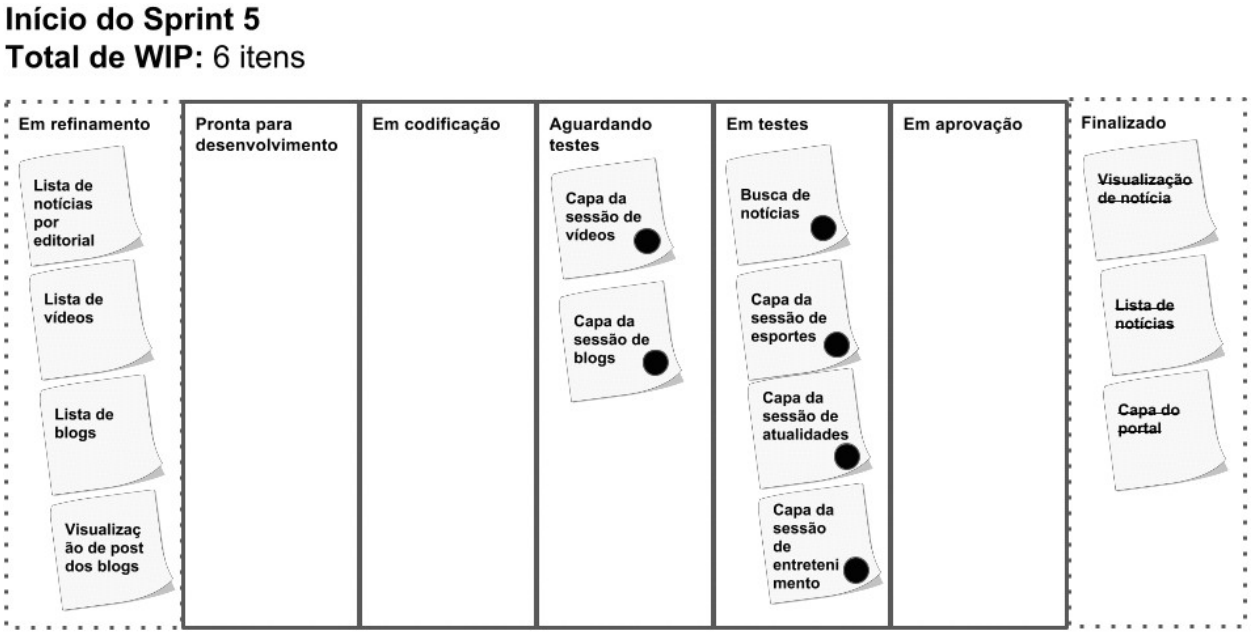


Figura 2.14: Estudo de caso: início da quinta sprint

Passado uma semana do início da *sprint*, o time de programadores conseguiu, em conjunto com o testador, estabilizar o ambiente de homologação e as integrações passaram a funcionar conforme o esperado. Como resultado desse esforço de coordenação do processo, a *sprint* 5 acabou com seis entregas em produção e com um sentimento de alívio por parte de todos.

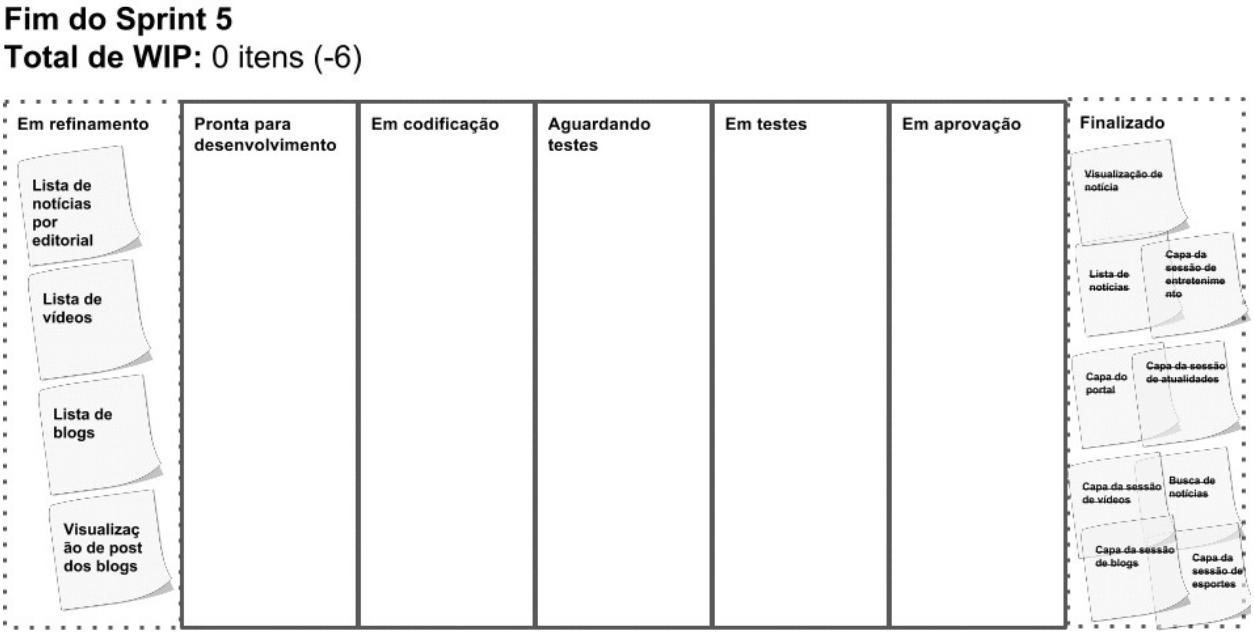


Figura 2.15: Estudo de caso: fim da quinta sprint

Antes do planejamento da *sprint* 6, o Agile Coach do time decidiu sugerir ao time que fosse avaliada a real capacidade de geração de entregas no período de duas semanas. Ele havia lido algo sobre limite de WIP e gostaria que o time se compromete a entregar o que seria o escopo de entrega da *sprint* futura.

Feito um estudo da capacidade do processo como um todo, o time chegou à conclusão de que o testador (ponto de gargalo) conseguia ter uma vazão semanal de duas funcionalidades. Sendo assim, o time determinou que assumiria quatro funcionalidades para serem construídas naquela *sprint*.

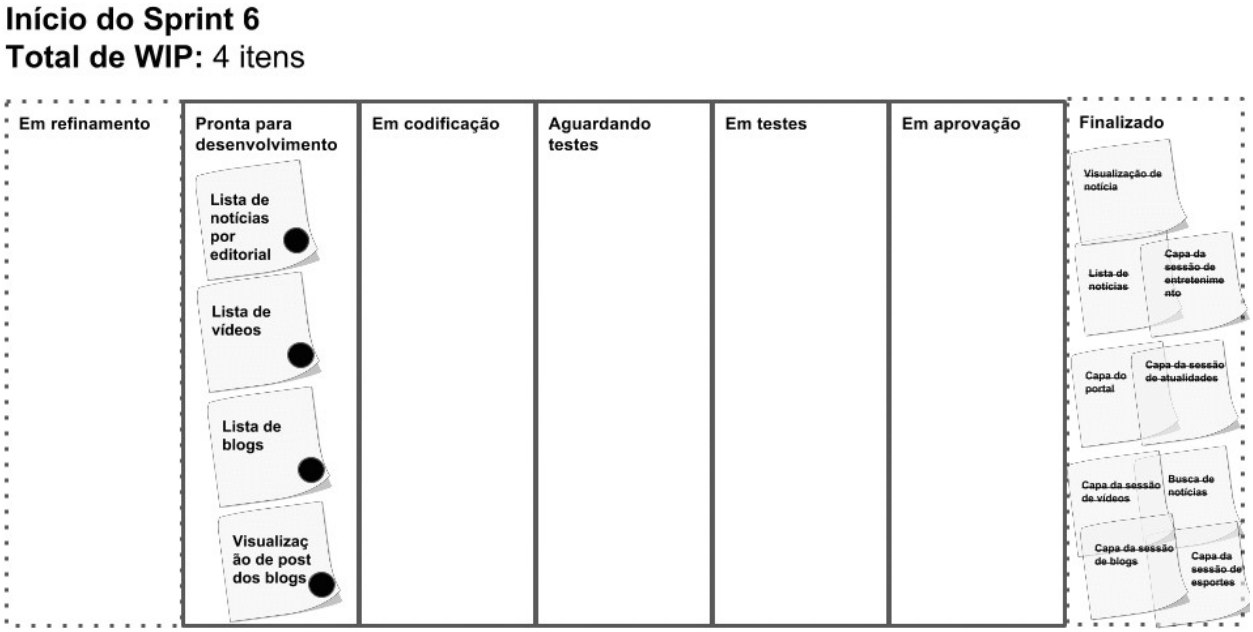


Figura 2.16: Estudo de caso: início da sexta sprint

Passada a primeira semana da *sprint* 6, o time havia percebido, na prática, que o desempenho do sistema (fluxo de desenvolvimento) está diretamente relacionado com a vazão do ponto de gargalo (no caso, a etapa de teste). O resultado mostrava que as duas funcionalidades que haviam sido pensadas como limite do sistema haviam sido entregues.

Pela primeira vez desde o início do projeto, o time conseguiu chegar ao final de uma *sprint* sem trabalho em progresso. Neste momento, todos sabiam que o processo de desenvolvimento havia chegado ao seu ponto ótimo de maturidade e que a limitação do WIP alinhada com a capacidade real do time trazia para todos os envolvidos o tão esperado sentimento de previsibilidade.

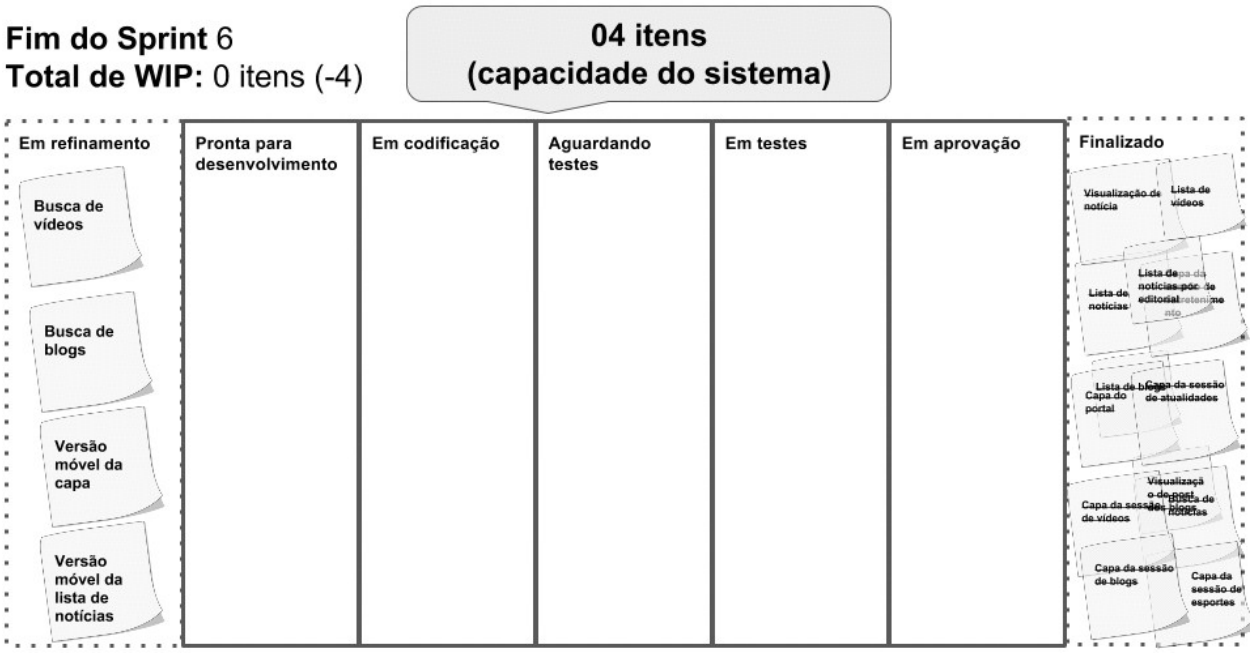


Figura 2.17: Estudo de caso: fim da sexta sprint

Ainda sobre a gestão de WIP, gostaria de compartilhar algumas considerações que poderão ser úteis para o time que você estiver trabalhando.

Limitar o WIP significa determinar o total de trabalho máximo que pode ser alocado em cada etapa do fluxo de desenvolvimento ou no sistema como um todo. Explicite com o seu time a política o quanto antes.

Independente da abordagem, sugiro que você defina os limites de WIP do processo de seu time após o monitoramento da média do número de itens que passam por cada etapa do processo a partir da amostra de algumas semanas ou *sprints*.

Caso utilize a abordagem de limitar o WIP de cada uma das etapas do processo, escolha algumas, pois alterar radicalmente muitas variáveis ao mesmo tempo não permitirá que você compreenda o efeito da ação no desempenho do sistema. Portanto, comece estipulando limites para as etapas mais críticas.

Tenha consciência de que colocar um limite mais restritivo no início gerará um desconforto no time, afinal, exigirá um envolvimento de todos para que as demandas que passarão pelo fluxo saiam o mais rápido possível dele.

Gostaria de ressaltar que o limite WIP de uma etapa do processo não será necessariamente o total de pessoas que estarão disponíveis para trabalhar nela.

Um exemplo seria se determinássemos que o limite da etapa "Em codificação" do estudo de caso fosse 3, dado que existiam 3 programadores disponíveis.

Tal abordagem seria arriscada, pois, em algum momento, alguém poderia parar o que estava sendo feito para ajudar em outra demanda mais importante, o que faria com aquela demanda já iniciada ficasse parada, ocupando, literalmente, espaço do estoque de uma etapa, sem de fato ter esforço sendo despendido nela.

Outro questionamento que você pode se fazer é: por que definir um limite de WIP? Aqui vão algumas constatações:

1. Você evitará a formação de gargalos que reduzirão a performance do time.
2. Ajuda a transparecer bloqueios no sistema para os *stakeholders*.
3. Quanto maior o número de demandas em andamento em determinada parte do processo, maior será o tempo em que uma demanda passará pelo fluxo (discutiremos sobre a métrica de *lead time* no próximo capítulo).
4. Reduzindo o trabalho em progresso, estaremos aumentando a qualidade do que se passa pelo sistema. Observo que o aumento de WIP está relacionado com fatores que invariavelmente decrescem a qualidade, aumentando o número de falhas.

Limitar o WIP gerará dor no sistema como um todo, dado que chegar ao número ideal não é um processo simples. Em contrapartida, tal estratégia é um passo para promover colaboração, pois, se as pessoas não se ajudarem, o sistema como um todo travará.

Só para não matá-lo de curiosidade, gostaria de lhe dizer que o time do estudo de caso rodou ainda mais 2 *sprints*, e o resultado final do projeto foi um sucesso. Os redatores conseguiram cadastrar todo o conteúdo que estava previsto para a versão que foi lançada, e o portal e sua versão móvel foram liberados para os usuários na data esperada.

2.5 Recapitulando

Gerenciar o trabalho em progresso é um importante aliado para times que procuram desenvolver um processo de desenvolvimento de software orientado à previsibilidade. Tivemos a oportunidade de discutir a relação do WIP com a capacidade de entrega de um time e com a qualidade do que é processado no seu fluxo de desenvolvimento. Além disso, vimos ao longo do estudo de caso apresentado algumas técnicas para reduzir ou aprimorar o que será trabalho em progresso.

Um aprendizado que espero que você tenha levado deste capítulo é de que WIP representa esforço e energia que ainda não foram validados, pois estão envoltos em uma caixa

chamada processo. Quanto mais tempo você passar carregando WIP, menos *feedback* você estará recebendo, mais lento será o processo de validação das hipóteses por detrás das iniciativas que originaram o trabalho e maior será o risco de você estar perdendo uma oportunidade de mercado.

Por fim, lembre-se: gerir o WIP faz parte da cultura de melhoria contínua.

PESSOAS QUE SUGIRO VOCÊ ACOMPANHAR SOBRE O ASSUNTO

Internacionais

David Anderson - http://www.djaa.com/

Pawel Brodzinski - http://brodzinski.com/

Nacionais

Rodrigo Yoshima - http://blog.aspercom.com.br/

Blog Plataformatec - http://blog.plataformatec.com.br/

Alisson Vale - http://softwarezen.me/

Celson Martins - http://celsoavmartins.blogspot.com.br/

2.6 Referências

ANDERSON, D. J. *Kanban:* Mudança evolucionária de Sucesso para seu Negócio de Tecnologia. Blue Hole Press, 2011.

BRODZINSKI, P. *The Kanban Story:* Kanban Board. Nov. 2009. Disponível: http://brodzinski.com/2009/11/kanban-story-kanban-board.html.

COHN, M. *Agile Estimating and Planning.* Prentice Hall, 2006.

PATTON, J. *The New User Story Backlog is a Map.* Out. 2008. Disponível em: http://jpattonassociates.com/the-new-backlog/.

PICHLER, R. *Agile Product Management with Scrum:* Creating Products that Customers Love. Addison-Wesley, 2010.

CAPÍTULO 3

Identificar o tempo para a entrega de uma demanda

3.1 Introdução

Agora que descobrimos a importância de medirmos o trabalho em progresso (WIP), precisamos começar a olhar quanto tempo as demandas que passam pelo nosso fluxo de desenvolvimento têm levado para serem finalizadas.

Conviver em um ambiente no qual clientes, usuários e equipe não possuem visibilidade sobre o tempo que é necessário para que determinado trabalho seja concluído gera desconfiança, dúvida, ruído na comunicação e, principalmente, a sensação de que tudo poderia ser feito mais rápido.

Se você lida no dia a dia com demandas que levam muito tempo e outras que acabam rápido demais, saiba que tal variabilidade é um grande ofensor quando se está em busca de um processo previsível, e dificilmente você conseguirá prever entregas futuras.

Para ajudá-lo no divertido desafio de analisar fatores que influenciem a previsibilidade quanto ao prazo de entrega de um projeto, produto, *bug*, funcionalidade ou história de usuário de software, teremos a oportunidade de discutir neste capítulo a importância da mensuração e da análise do *lead time*.

Se você já sabe o que é o *lead time* ou já o monitora, não deixe de conferir as diferenças entre *lead time* e *cycle time*, ou novas formas de olhar para tal métrica. Veja também o estudo de caso no qual relato a rotina de uma equipe que trabalha na operação e evolução de um produto de software.

Lendo o conteúdo que será apresentado, você terá a oportunidade de avaliar a relação entre tempo e tamanho das demandas que estão passando pelo fluxo de desenvolvimento, aprimorar a saúde do processo da sua equipe e negociar de forma mais clara prazos de entrega.

Ao final, espero que você passe a utilizar o *lead time* como mais uma métrica útil na proposição de melhorias no processo de desenvolvimento de software baseadas em dados.

3.2 O que é o lead time?

Se quisermos responder aos nossos clientes quando um item de trabalho estará pronto, antes de iniciarmos o desenvolvimento, precisaremos ter como referência *lead times* de demandas que passaram pelo processo de desenvolvimento.

Para a indústria, *lead time* representa a latência entre a iniciação e a execução de um processo. No setor de bens industriais, a redução de tal métrica representa um aspecto-chave da manufatura enxuta (*lean manufacturing*).

No contexto de desenvolvimento de software, podemos considerar o *lead time* como sendo o número de dias entre o início e o fim do processo de entrega de um item de trabalho (por exemplo, história do usuário, *bugs* etc.).

Conforme podemos visualizar na figura a seguir, o *lead time* nada mais é do que o tempo de atravessamento de um item a partir dos limites de entrada e saída definidos no processo de desenvolvimento.

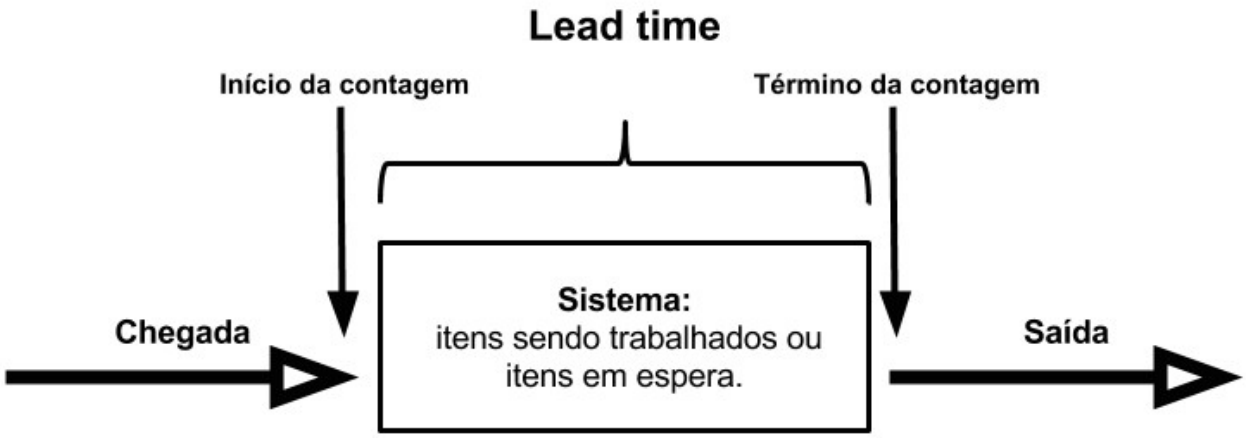


Figura 3.1: Cálculo do lead time através dos limites do sistema

A mensuração do *lead time* no desenvolvimento de software passou a ganhar notoriedade quando David Anderson, criador do método Kanban, trouxe à tona a importância de coletar a métrica. No entanto, medi-la independe do método de trabalho que está sendo adotado.

Por exemplo, já participei de projetos que utilizavam Scrum em que todas as histórias de usuário que passavam pelo fluxo de desenvolvimento tinham o seu *lead time* aferido.

Medir o *lead time* é útil para:

1. Compreender quanto tempo a equipe tem levado para desenvolver um item de trabalho.

2. Analisar a saúde do processo de desenvolvimento dado que altas dispersões representam algum tipo de gargalo ou aumento no tempo de passagem em alguma etapa do fluxo de desenvolvimento (exemplo: nas últimas duas semanas, o *lead*

time das histórias de desenvolvimento cresceram, pois o ambiente de homologação estava com problemas e os testes eram mais complexos).

3. Identificar casos extremos (do inglês, *outliers*) e aprender com o ocorrido (exemplo: determinado *bug* levou muito mais tempo do que o normal para ser corrigido em decorrência da ausência de clareza sobre o que era de fato o problema a ser resolvido).

4. Analisar se a equipe tem entregue os itens de trabalho dentro de um padrão de dias ou semanas (exemplo: a maioria das histórias de usuário da equipe estão dentro de um período de 2 semanas).

5. Para compreender os efeitos que as incertezas e as complexidades não mapeadas podem causar, na forma de variabilidade, no tempo necessário para a conclusão dos itens de trabalho de uma equipe de desenvolvimento de software.

Além do aspecto temporal, o *lead time* pode ser um ótimo preditor de custo. De um modo geral, quanto mais tempo algo leva para ser concluído, maior o seu custo.

O custo de um projeto, funcionalidade ou até mesmo de uma história de usuário pode ser um dos fatores determinantes para que um patrocinador ou gestor de produto invista no desenvolvimento de algo. Gostando ou não, precisaremos do *lead time* para descobrir o custo de desenvolvimento.

3.3 Diferentes formas de medir o lead time

Uma pergunta que as pessoas que trabalham com desenvolvimento de software costumam me fazer é: *lead time* é medido em dias úteis ou dias corridos? Em partes, tal questionamento surge pois algumas ferramentas de gestão de projetos ágil do mercado permitem que os usuários configurem a forma de se gerar a métrica.

Independente do ferramental, gostaria de discorrer sobre as duas abordagens para que você decida o que fará mais sentido no seu contexto. Se você está em um ambiente de desenvolvimento no qual há um prazo de entrega ou de um projeto de software que ainda não virou um produto, o *lead time* tenderá a ser uma métrica de controle do processo por parte da equipe, portanto, medi-lo em dias úteis pode fazer mais sentido. Mas por quê?

1. A métrica levará em consideração apenas o tempo que de fato está disponível para a equipe finalizar a entrega prevista.

2. Se a equipe precisar projetar o prazo de entrega de uma funcionalidade futura, poderá utilizar o *lead time* passado e afirmar, em dias úteis, quando aquela

demanda poderá ser finalizada. Por exemplo, se a equipe tem um padrão de levar 3 dias úteis para entregar um item e alguém demanda algo na sexta-feira, provavelmente tal item será finalizado na quarta-feira e não na segunda, pois, a equipe não trabalha aos finais de semana.

3. Sabendo que a equipe trabalhará dentro de um prazo de entrega definido, as decisões que serão tomadas a partir do *lead time* precisarão ser pontuais e estarão relacionadas ao fluxo de desenvolvimento da equipe, afinal, o produto de software ainda não está em uso. Por exemplo, a partir da análise do *lead time*, a equipe pode identificar que uma otimização na etapa de aprovação das histórias de usuário reduzirá o *lead time* das histórias que passam pelo fluxo de desenvolvimento.

Caso você esteja em um contexto em que o software já é um produto e precisa de evoluções e correções de falhas, e que o *feedback* do usuário e o tempo de resposta ao mercado são aspectos extremamente sensíveis, medir o *lead time* considerando dias corridos será importante. Isso porque:

1. A métrica levará em consideração os atrasos e o tempo gasto em espera gerado pelos finais de semana, feriados e dias não úteis. Podemos dizer que a métrica refletirá de fato a dor do cliente no caso de um *bug* não resolvido no sábado, ou a perda da oportunidade de disponibilizar, em um feriado, uma nova funcionalidade frente à concorrência.

2. Outro motivador para que você meça o *lead time* em dias corridos é de que ele representa o tempo necessário para se coletar *feedback* do cliente ou usuário, algo tão importante para quem trabalha no universo do conhecimento. Valor por si só é, em última instância, determinado pelo cliente, o que quer dizer que a equipe buscará ter certeza de estar recebendo *feedback* do valor gerado o mais rápido possível. A última coisa que você e sua equipe buscam é criar algo que o cliente não precisa. Diminuindo o *lead time*, reduzimos o ciclo de *feedback* com nossos clientes.

3. Dada a consideração de todo tipo de espera, as projeções de entrega de um item (como falha, funcionalidade etc.) a partir do histórico de *lead time*, serão mais previsíveis. Por exemplo, se um *bug* tem por padrão demorar 3 dias para ser corrigido e foi descoberta uma falha grave na quarta-feira, a equipe terá maior certeza de que, sendo iniciada na quinta-feira, até sábado o problema estará sanado e que ações específicas podem ser tomadas para tratar o caso, como, por exemplo: pessoas trabalhando um tempo extra durante a semana; redução nas etapas de aprovação de um *deploy*; priorização máxima nos testes etc.

4. As decisões que serão tomadas a partir do *lead time* levarão em consideração aspectos extras no fluxo de desenvolvimento da equipe e estarão relacionadas intimamente ao negócio. Por exemplo, quantos clientes meu produto perdeu por conta de determinado *bug* não ter sido resolvido mais rápido? Quantos clientes não

aderiram ao produto por não termos colocado determinada funcionalidade em produção antes?

Sintetizando o que foi comentado até aqui, gostaria de compartilhar uma tabela para que você discuta com sua equipe qual das estratégias de mensuração de *lead time* é a melhor, afinal, lembre-se, contexto é importante para qualquer métrica.

Lead time dias úteis	**Lead time dias corridos**
Produto de software em desenvolvimento.	Produto de software em produção.
Melhorias centradas no processo de desenvolvimento da equipe.	Melhorias em todo o fluxo, inclusive em esperas externas (ex: finais de semana).
Ainda não existe *feedback* do usuário.	*Feedback* do usuário é importante.
Atraso influenciando o prazo de entrega.	Atraso influenciando a perda de clientes atuais, a perda de oportunidades de mercado por conta da concorrência e a perda da credibilidade do produto.

Antes de começar a analisar o *lead time*, tenha em mente o que a métrica está dizendo para que a análise tenha embasamento. Sempre que se deparar com uma distribuição de *lead time*, pergunte: o que este número representa?

Bem, espero tê-lo convencido a começar a medir o *lead time*. No próximo tópico, teremos a oportunidade de entender por que algumas pessoas acham que *lead time* e *cycle time* são sinônimos.

3.4 Qual a diferença entre lead time e cycle time?

Independente de quais serão as etapas que definirão os limites para a mensuração do *lead time*, gostaria de lhe mostrar por que *lead time* e *cycle time* são medidas distintas. Mas, espere, você já deve ter lido ou escutado em algum lugar que *cycle time* nada mais é do que o *lead time* do processo de desenvolvimento como um todo, certo?

Uma série de autores utilizam o termo *cycle time* para descrever algo que foge da definição discutida pelo núcleo de conhecimento das áreas de operações e produções, de onde puxamos o conceito de tempo de atravessamento (*lead time*).

Segundo Ohno (1988), *cycle time* é medido pela quantidade de itens por unidade de tempo (exemplos: clientes/minuto, peças/hora, roupas lavadas/segundo etc.). Trazendo para o dia a dia de desenvolvimento de software, é como se uma equipe nos dissesse que o seu *cycle time* médio é de 1 história de usuário a cada 4 horas.

Vejamos através das figuras a seguir um exemplo que ajudará a desmistificar de vez a ideia de que *lead time* é igual a *cycle time*, a partir das definições vistas anteriormente.

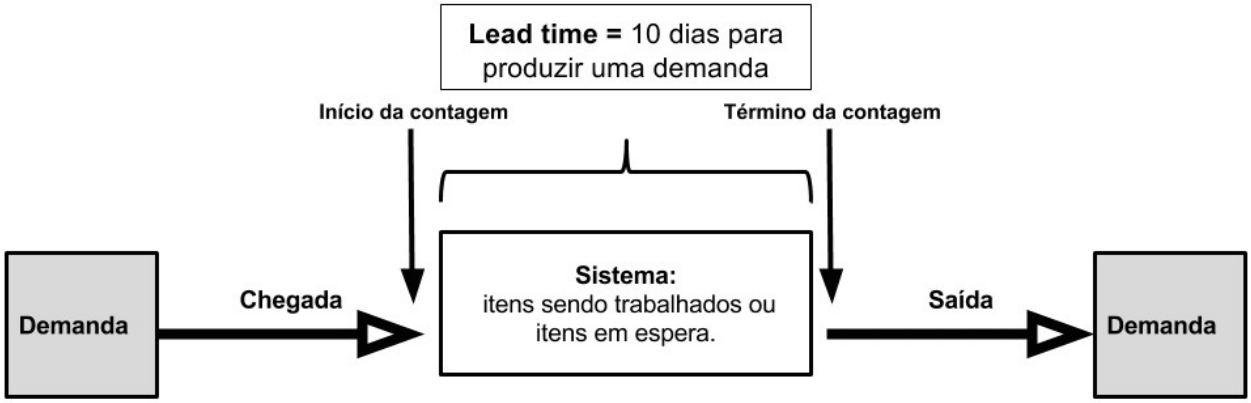


Figura 3.2: Exemplo de lead time calculado

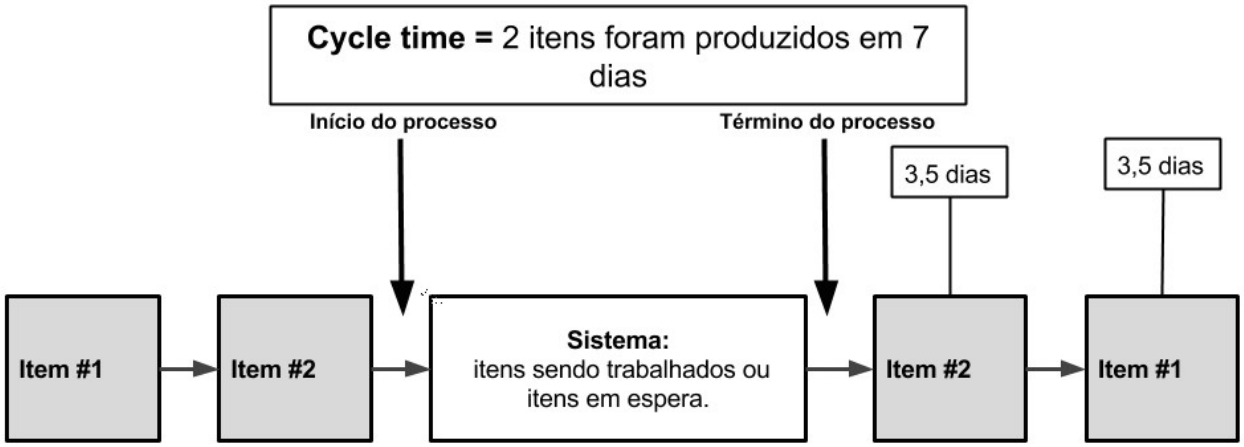


Figura 3.3: Exemplo de cycle time calculado

A partir dessas imagens, imagine que a equipe que esteja rodando o processo chegue às seguintes conclusões:

- Levamos 10 dias para terminar a funcionalidade de *login* através dos perfis de redes sociais.

- Na última semana, ou seja, nos últimos 7 dias, foram desenvolvidas as funcionalidades de inclusão e atualização dos usuários do sistema.

Podemos dizer que o *lead time* da funcionalidade de *login* foi de 10 dias e que *cycle time* ou número de itens entregues nos últimos 7 dias foi de duas funcionalidades.

Conforme veremos no próximo capítulo, *cycle time* pode ter o mesmo significado que *throughput* (ou vazão do processo). Além disso, para que seja possível a aplicação da lei de *Little* no processo de desenvolvimento de software, precisamos usar uma versão baseada na saída de itens e não na entrada, como é o caso da fórmula original.

Pessoalmente, aboli do meu vocabulário o uso de *cycle time* no dia a dia de desenvolvimento de software, pois o termo carrega dúvidas sobre o seu significado.

Espero que, com essas definições, você passe a me acompanhar nesse movimento de usar apenas o termo *lead time* quando quiser falar sobre o tempo que um item de trabalho levou para sair do seu processo.

Deixo como dica de leitura um ótimo artigo produzido por Thomas - mais detalhes podem ser vistos em Thomas (2015). Nele é descrita a história de onde vem esse mal-entendido sobre o uso do *cycle time* como sendo sinônimo de *lead time*.

Nas duas próximas seções, discutiremos algumas derivações que você poderá fazer a partir do momento em que você passou a coletar o *lead time* da sua equipe e que serão úteis para uma análise da saúde do processo.

3.5 Uma proposta para quebrar a análise do lead time

Conforme temos aprendido ao longo deste capítulo, analisar o tempo que determinado item passa pelo fluxo de desenvolvimento é fundamental para a redução da variabilidade do processo, a fim de se conquistar a tão esperada previsibilidade. Buscando segmentar a visualização da métrica de *lead time*, gostaria de apresentá-lo ao *lead time breakdown*.

Resumidamente, este gráfico permite que o time avalie, compare e analise em detalhes a latência entre o início e o término de cada uma das etapas do processo de desenvolvimento para cada item de trabalho que passa pelo fluxo (exemplo: *bugs*, histórias do usuário etc.).

Esse tipo de visualização permite analisar o processo com mais detalhes se comparado com a análise do *CFD* (veremos mais detalhes no capítulo *Visualizando o fluxo de desenvolvimento de um time ágil*). Geralmente, o *lead time breakdown* é utilizado para responder questões como:

- O que está acontecendo com os itens que estão em progresso?
- Existe algum tipo de impedimento atrapalhando a fluidez da equipe?
- A equipe está lidando com algum gargalo no processo (exemplo: sobrecarga na etapa de testes)?

A forma de criar a visualização é bem simples. No eixo vertical, são expostos os dias de trabalho. Já no eixo horizontal, são exibidos os itens de trabalho em progresso ou já entregues pela equipe (exemplo: o montante de histórias do usuário monitorados pelo quadro Kanban). Para representar cada uma das etapas do processo, é usado o gráfico de barra acumulado.

A seguir, veremos um exemplo para que você possa utilizar o *lead time breakdown* com a sua equipe. Imagine que a equipe *XPTO* tenha um fluxo de trabalho com as seguintes etapas que estão ilustradas na figura seguinte.

Figura 3.4: Fluxo de desenvolvimento da equipe XPTO

O fluxo da equipe *XTPO* possui algumas etapas com o objetivo de medir gargalos criados pelas etapas de teste (Aguardando testes), ou pelo responsável pelas aprovações das entregas dos itens trabalhados (Aguardando homologação).

Analisando as demandas que passaram pelo fluxo de desenvolvimento da equipe no período de uma semana, extraímos a seguinte representação do gráfico *lead time breakdown*.

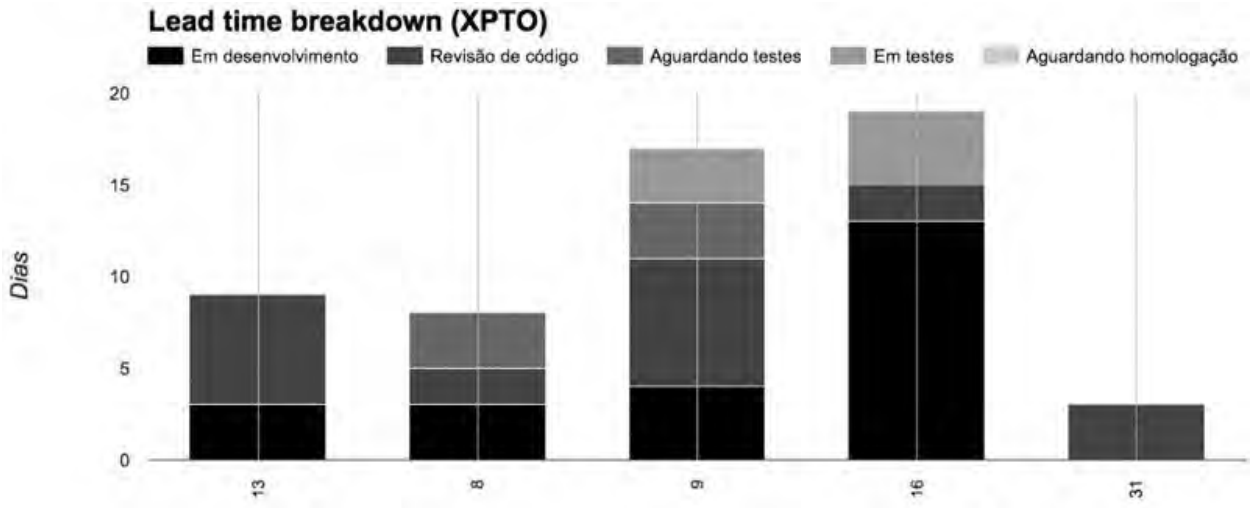


Figura 3.5: Exemplo do gráfico de lead time breakdown — Visão geral

Olhando pela perspectiva das demandas que estão mais próximas de serem finalizadas, isto é, aquelas que estão mais próximas do último estágio do processo, que é aguardando homologação, podemos identificar que duas histórias estão há mais de 3 dias para serem revisadas.

Uma ação que a equipe *XPTO* poderia tomar ao analisar o gráfico seria a de centralizar os esforços para finalizar tais itens. Além disso, a equipe poderia se questionar: por que tais demandas estão levando tanto tempo em uma etapa de espera (aguardando homologação)? Após a identificação dos prováveis bloqueios, a equipe poderia atuar na remoção deles.

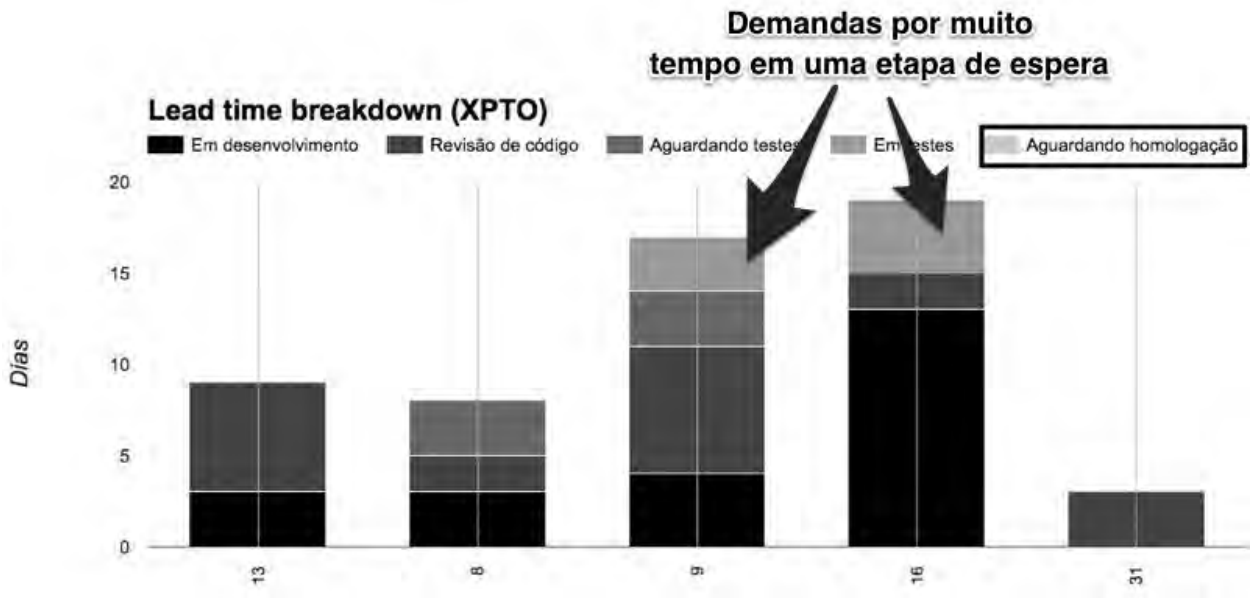


Figura 3.6: Exemplo do gráfico de lead time breakdown — Etapa de homologação

Outro ponto interessante de se observar no exemplo da equipe *XPTO* diz respeito à alta variabilidade no *lead time* da etapa de codificação (Em desenvolvimento). A equipe provavelmente trouxe para o fluxo de desenvolvimento histórias de usuário com níveis de complexidade distintos, ou seja, algumas foram muito simples (*lead time* baixo) e outros foram mais complexos (*lead time* alto).

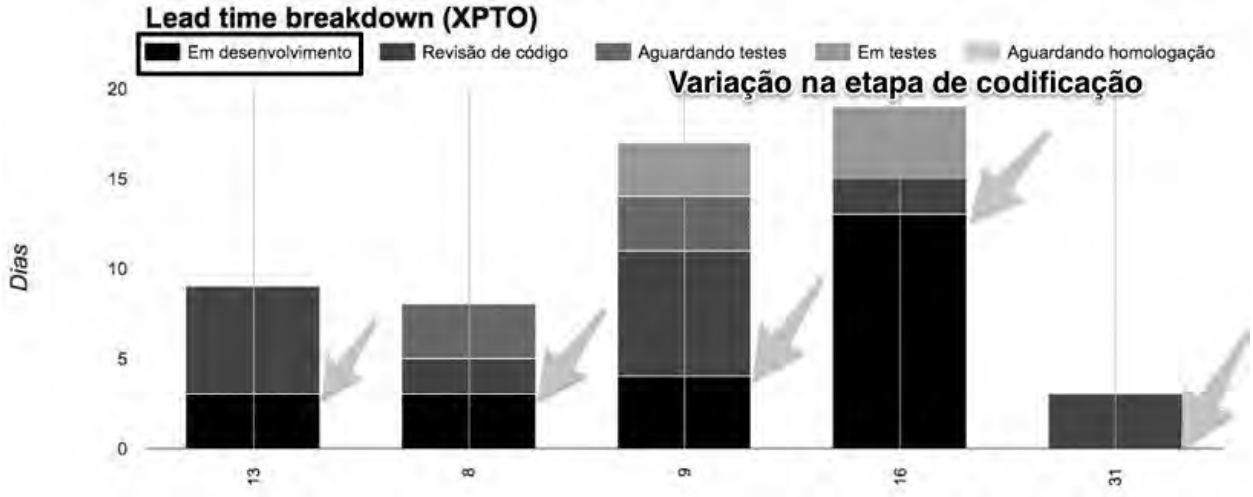


Figura 3.7: Exemplo do gráfico de lead time breakdown — Etapa de desenvolvimento

Conforme a equipe e os envolvidos no projeto passam a ter contato com esse tipo de visualização, situações como as apontadas no exemplo passam a ser evidenciadas mais cedo e soluções são discutidas para contornar possíveis problemas.

Caso você adote o *lead time breakdown* em sua equipe, traga a análise da visualização para cerimônias como a reunião diária ou para momentos de reflexão sobre melhorias no processo. Você perceberá que a equipe passará a olhar o fluxo de desenvolvimento de uma maneira objetiva.

3.6 Como analisar a eficiência do fluxo através do lead time

Fora a visualização, quando quebramos a métrica de *lead time* por etapa, ganhamos a possibilidade de analisarmos a eficiência do fluxo de desenvolvimento.

Para aqueles que não conhecem, a eficiência de um fluxo nada mais é do que o somatório do total de trabalho empregado em um item dividido pelo seu *lead time* total. No fundo, é como se quiséssemos analisar a relação entre quanto do trabalho empregado de fato em uma demanda foi refletido no tempo que foi necessário para entregá-la, desconsiderando tempos de bloqueios e filas.

$$eficienciadofluxo = \frac{somatório(diasdetrabalho)}{leadtimetotal}$$

Figura 3.8: Fórmula para o cálculo da eficiência do fluxo

Bloqueios por alguma dependência externa (exemplo: equipe, terceiros etc.) ou tempos de espera na fila são exemplos de quando um item não está de fato sendo trabalhado. Em ambos os casos, o *lead time* do item continua sendo contabilizado, porém, ninguém está ativamente trabalhando nele.

Mensurar tal informação pode ser trabalhosa, afinal, exige um acompanhamento diário para saber distinguir, dentro do *lead time* total, o que foi de fato espera e o que foi trabalho empregado. Porém, caso você precise mostrar para a equipe ou para *stakeholders* externos que o fluxo de desenvolvimento precisa de melhorias e, principalmente, mostrar, em números, evoluções que ocorreram no processo a partir de ações que foram aplicadas, tal informação será de extrema valia.

Para facilitar o que foi explicado até agora, gostaria de lhe propor a análise de um exemplo. Imagine que nos últimos quinze dias as demandas entregues pela equipe *XPTO* possuam o seguinte comportamento de *lead time*.

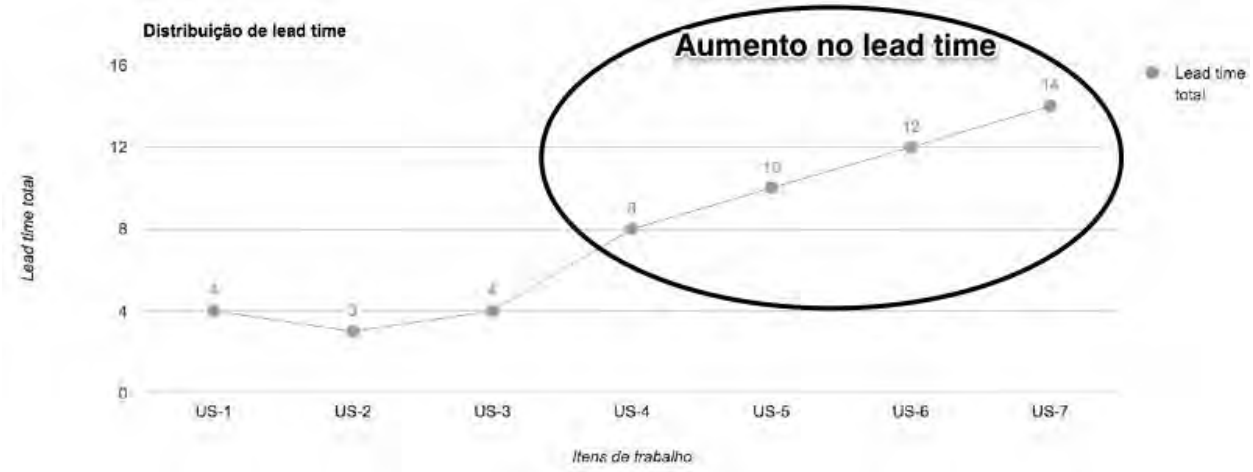


Figura 3.9: Gráfico de lead time da equipe XPTO nos últimos 15 dias

Os itens de trabalho representados nesse gráfico estão organizados em um formato no qual aqueles que foram iniciados primeiro são apresentados à esquerda.

Ao observar os dados podemos dizer que os últimos itens que entraram no fluxo de desenvolvimento da equipe *XPTO* tiveram um maior *lead time*. Alguém poderia já imaginar que tal elevação tenha sido gerada por conta de a equipe ter trabalhando no desenvolvimento de itens de trabalho mais complexos ou incertos. Mas será que é isso mesmo?

Vejamos na tabela a seguir qual foi a relação entre o *lead time* total, os dias em que de fato houve trabalho empregado, os dias em que houve espera ou algum tipo de bloqueio e, por fim, a métrica de eficiência do fluxo.

Itens de trabalho	Lead time total	Dias de trabalho	Dias em espera/bloqueados	Eficiência do fluxo
US-1	4	1	3	25.00%
US-2	3	1	2	33.33%
US-3	4	1	3	25.00%
US-4	8	2	6	25.00%
US-5	10	2	8	20.00%
US-6	12	2	10	16.67%
US-7	14	2	12	14.29%

Figura 3.10: Tabela com a eficiência do fluxo nos últimos 15 dias

Os dados anteriores nos mostram uma queda na eficiência do fluxo. É possível afirmar que a média da eficiência do fluxo nos itens US-5, US-6 e US-7 foi de aproximadamente 17% ante uma performance anterior aproximada de 27%. É como se disséssemos que o fluxo de desenvolvimento da equipe *XPTO* tenha perdido 10% da sua eficiência quando analisadas as últimas demandas entregues.

Pense em tal queda de eficiência por um segundo. Estamos dizendo que se o próximo item de trabalho da equipe *XPTO* tiver um *lead time* de 10 dias, aproximadamente, 2 dias serão de trabalho e os outros 8 serão de algum tipo de estado inativo. Onde será que você, se estivesse na equipe *XPTO*, proporia algum tipo de melhoria no processo?

Provavelmente seria muito difícil diminuir os dias de trabalho, porém, existiriam milhares de oportunidades para reduzir os 8 dias de inatividade. Qualquer diminuição no tempo de espera ou de bloqueio teria um efeito de melhoria em 80% do *lead time* da equipe.

Outra informação que conseguimos tirar da tabela da equipe *XPTO* é de que a quantidade de dias trabalhados nos itens não variou muito (entre 1 e 2 dias), porém, não podemos dizer o mesmo para o *lead time* total.

Portanto, uma conclusão a que podemos chegar é a de que o aumento do *lead time* total das últimas demandas da equipe *XPTO* não estava relacionado a algum tipo de aumento na complexidade ou incerteza nos itens de trabalho, mas, sim, que havia outros ofensores no fluxo (como aumento no tempo de espera para *deploy*; aumento no tempo de espera para aprovação; ausências na equipe; problemas de ambiente etc.).

E a eficiência do seu fluxo, como está? Antes de criar qualquer conclusão a partir do *lead time*, olhe para a eficiência, você perceberá que o seu processo tem muito espaço para melhoria.

Conforme vimos nas seções anteriores, quebrar a visualização do *lead time* e olhar para a eficiência do fluxo são ótimas ferramentas para analisar a saúde do processo de desenvolvimento de uma equipe que desenvolve ou mantém um software.

No próximo tópico, teremos a oportunidade de analisar um caso onde o *lead time* foi útil para identificar uma série de problemas internos e externos à equipe.

3.7 O uso do lead time a partir de um caso real

A seguir, teremos a oportunidade de analisar a importância do acompanhamento periódico do *lead time* e discutir alguns problemas no processo que podem ser identificados a partir de tal monitoramento.

Espero que, ao final deste estudo de caso, você possa aprender e aprimorar no seu dia a dia as técnicas de análise que serão compartilhadas.

Estudo de caso — Manutenção de um sistema de automação de força de vendas web por uma equipe de TI interno

O caso se passou em uma equipe que trabalhava na manutenção e evolução de um sistema web responsável pela automação comercial de uma distribuidora de chocolates. A aplicação estava em produção havia 1 ano e contava com um total de 100 representantes de vendas utilizando a solução. Por mês, a plataforma transacionava mais de 1 milhão de reais em vendas.

Sabendo da criticidade da aplicação para o faturamento mensal da distribuidora, o diretor do departamento de TI (Tecnologia da Informação) criou uma área responsável por corrigir possíveis *bugs* na plataforma e evoluir as funcionalidades já existentes a partir dos *feedbacks* dos usuários. A equipe que operava o sistema foi a mesma que trabalhou na construção da primeira versão que foi lançada, e era composta por 2 desenvolvedores e 1 analista de negócios.

Dada a característica de se trabalhar com múltiplas demandas (exemplo: as mesmas pessoas corrigiriam as falhas e também criariam novas funcionalidades), a equipe optou por utilizar o método Kanban como ferramental para a gestão dos trabalhos que surgiriam no dia a dia.

Semanalmente a equipe se reunia para definir as prioridades e para avaliar como andava a saúde do processo de desenvolvimento. Quando convinha, eram realizadas sessões de retrospectiva com o intuito de promover melhorias.

Tanto as novas funcionalidades como as falhas eram implementadas em um ambiente de desenvolvimento. Depois, passavam por um processo de validação em um ambiente de homologação, para enfim serem liberadas aos usuários no ambiente de produção.

O fluxo de trabalho da equipe estava dividido nas seguintes etapas:

- **Para ser feito**: etapa na qual uma demanda recebida e analisada passava a fazer parte do fluxo de desenvolvimento da equipe.
- **Em desenvolvimento**: etapa em que uma demanda era implementada em um ambiente específico para os desenvolvedores.
- **Aguardando homologação**: etapa onde os desenvolvedores disponibilizavam determinada demanda implementada para a validação do analista de negócios no ambiente de homologação.
- **Em homologação**: etapa na qual eram testadas e homologadas as demandas pelo analista de negócios no ambiente de homologação.
- **Aguardando publicação**: etapa onde a demanda aguardava para ser publicada em produção.
- **Em produção**: etapa em que os usuários eram notificados de que determinada demanda havia sido disponibilizada no sistema.

A equipe decidiu monitorar o *lead time* incluindo dias corridos, afinal, a equipe de vendas operava o sistema durante os 7 dias da semana e existia, entre a equipe técnica, uma escala para suporte a possíveis problemas nos finais de semana.

Os tópicos que seguirão a exposição do caso mostrarão alguns problemas que foram identificados a partir da análise e do monitoramento do *lead time*. Você perceberá que os gráficos usados na análise combinam, no eixo horizontal, as demandas que passaram pelo fluxo de desenvolvimento e, no eixo vertical, o *lead time* de cada demanda.

Como informação complementar, baseado nos 5 últimos valores de *lead times* foi traçada uma média móvel. Tal medida visa apresentar uma tendência de aumento ou queda nos valores de *lead time*.

Problema #1: indefinição nas demandas

Nas primeiras duas semanas que o *lead time* começou a ser monitorado, a equipe percebeu que as três primeiras demandas entregues (sinalizadas na figura a seguir pelas setas em preto) tiveram tempos de entregas muito distintos.

Figura 3.11: Estudo de caso: indefinição nos requisitos

Em um primeiro momento, a conclusão da equipe foi de que o *lead time* variou dada a diferença na natureza das demandas, pois o gráfico misturava entregas de funcionalidades e *bugs*.

Mesmo considerando a diferença do tipo de demanda, se analisarmos apenas as funcionalidades, temos uma diferença entre os 9 dias necessários para a entrega da segunda funcionalidade, quando comparados com os 3 dias que foram despendidos para a entrega da primeira funcionalidade.

Realizada uma reflexão para identificar o motivo de tal variação, a equipe percebeu que a ***Feature#2*** entrou para a etapa de implementação sem que estivesse devidamente alinhada e compreendida por todos, o que acabou gerando dúvidas, discussões e vários ajustes ao longo do seu processo de construção.

Para ilustrar o que aconteceu, o analista de negócios da equipe lembrou de que a funcionalidade foi e voltou para a equipe de desenvolvimento ao menos 3 vezes na etapa de homologação. A partir do aprendizado gerado na primeira análise do *lead time*, a equipe definiu que as funcionalidades e *bugs* deveriam respeitar uma estrutura básica antes que fossem implementadas.

Para os futuros *bugs*, as informações propostas foram:

- **Cenário atual:** descrevia a não conformidade encontrada através de evidências, como por exemplo: imagens que apresentavam os erros encontrados no sistema, descrição explicando passo a passo como reproduzir o problema etc.
- **Resultados esperados:** elencava os resultados que o solicitante do *bug* esperava com a entrega da correção.

No caso das funcionalidades, as informações listadas para que uma demanda fosse desenvolvida foram:

- **Descrição:** uma breve descrição daquilo a que se refere a demanda.
- **Critérios de aceite:** a descrição de todos os critérios de aceite referentes à demanda que seriam utilizados para o desenvolvimento e validação.
- **Informações complementares:** caso necessário, seriam informados detalhes sobre a interface que deveria ser usada, ou se existia a dependência com algum tipo de aplicação externa.

O primeiro problema apontado no estudo de caso nos mostra que identificar oscilações no *lead time* nas primeiras semanas de um projeto, ou até mesmo em uma realidade de operação de um produto de software, pode nos dar sinais de que existem melhorias que precisam ser discutidas no fluxo de desenvolvimento.

Assim como discutimos no capítulo de WIP, se você busca que a sua equipe não inicie o desenvolvimento de uma demanda mal definida, crie maneiras de caracterizá-la minimamente.

Se você trabalha com Scrum, é como se a equipe passasse a garantir qualidade na forma como são descritas e transmitidas as informações das histórias de usuário ou itens de trabalho que são discutidos em uma cerimônia como o *planning*. No caso de uma abordagem do método Kanban, é como se a equipe estivesse deixando claro quais são as regras para que haja um compromisso com a entrega de uma demanda que passará pelo fluxo de desenvolvimento.

Se você deseja que a sua equipe não sofra com variações no *lead time* por conta de indefinições, antes que a equipe assuma o desenvolvimento de qualquer trabalho, proponha que ela responda às seguintes questões:

- Qual o problema que precisamos resolver?
- Qual é o resultado esperado por quem demandou (como cliente, usuário etc.)?
- Existe alguma dependência de negócio ou técnica para esta demanda (exemplo: a equipe precisa solicitar algum tipo de correção, pois há uma integração com uma aplicação externa)?
- Existe algum tipo de informação de interface importante para que a demanda seja desenvolvida?
- Os critérios para que haja o aceite da demanda estão claros?

Problema #2: entregas gerando bugs

Após a entrega da **Feature#2**, a equipe responsável pela evolução do sistema do estudo de caso passou a receber certa pressão dos usuários da aplicação e do diretor da área de tecnologia para entregar em duas semanas as próximas demandas priorizadas no fluxo.

A justificativa era de que as funcionalidades **Feature#3** e **Feature#4** permitiriam a comercialização de chocolates para um conjunto de países da Europa. A previsão era de que, com o software em funcionamento, o volume de vendas da empresa aumentaria em 15%.

Como as demandas foram paralelizadas entre os dois desenvolvedores da equipe, no início da segunda semana as funcionalidades encontravam-se entregues em produção. O clima era de dever cumprido e a área de vendas passou a captar pedidos antes do prazo estipulado.

Mas, infelizmente, nem tudo são flores. Após o início da operação das funcionalidades, a equipe de desenvolvimento passou por três duras semanas trabalhando na correção de falhas geradas no sistema.

O gráfico a seguir nos mostra que, das 6 demandas que passaram pelo fluxo de desenvolvimento no período, cinco delas diziam respeito a **falhas**. Ou seja, aproximadamente 85% do trabalho da equipe foi despendido na correção de erros gerados pela publicação das novas funcionalidades.

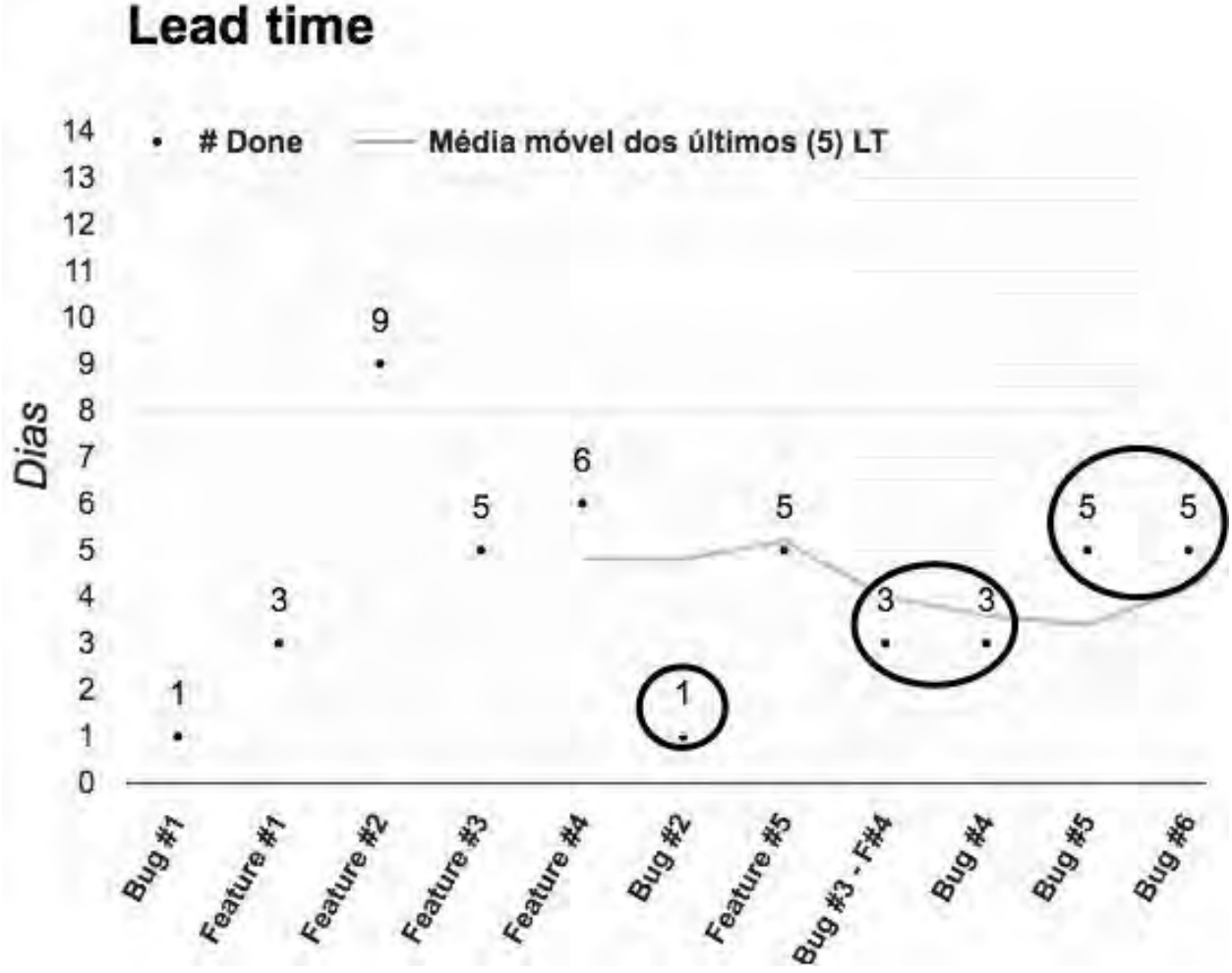


Figura 3.12: Estudo de caso: entregas gerando falhas

Ao analisar o aspecto econômico do que estava acontecendo, o diretor de TI chegou à conclusão de que as correções das falhas tiveram um custo menor do que o de não liberar as funcionalidades em produção no período esperado. Isto é, o resultado das vendas supriu o gasto com as pessoas envolvidas na gestão das falhas que foram tratadas.

Haverá situações nas quais a inclusão de funcionalidades com falha serão maléficas para a saúde econômica de um produto de software. Imagine você em um cenário no qual a publicação de uma funcionalidade, que deveria ser uma novidade ou até mesmo uma equiparação quanto ao que é ofertado pela concorrência, comece a gerar um volume de erros tão grande que faça com que a base de clientes comece a diminuir sistematicamente, dada a insatisfação criada.

Ou seja, moral da história: entregas mais rápidas (*lead time* baixo) nem sempre são de qualidade.

Fuja de ambientes que estipulem metas quanto ao prazo de entrega de uma demanda. Em vez de tal abordagem, proponha que sejam feitas análises e melhorias no processo como forma de otimizar o fluxo de desenvolvimento, pois, por conseguinte, haverá uma melhora no *lead time*.

Problema #3: demandas sem uma análise de complexidade e incerteza

Passada algumas semanas, a equipe do estudo de caso percebeu que duas funcionalidades que haviam sido concluídas levaram quase o dobro de tempo quando comparadas com as funcionalidades entregues anteriormente.

Analisando o gráfico a seguir, percebemos que a média móvel cresceu em decorrência do *lead time* das funcionalidades **Feature#6** e **Feature#7** (sinalizadas pelas setas). Até então, percebíamos uma baixa variabilidade no *lead time* dado um comportamento de baixa oscilação na média móvel.

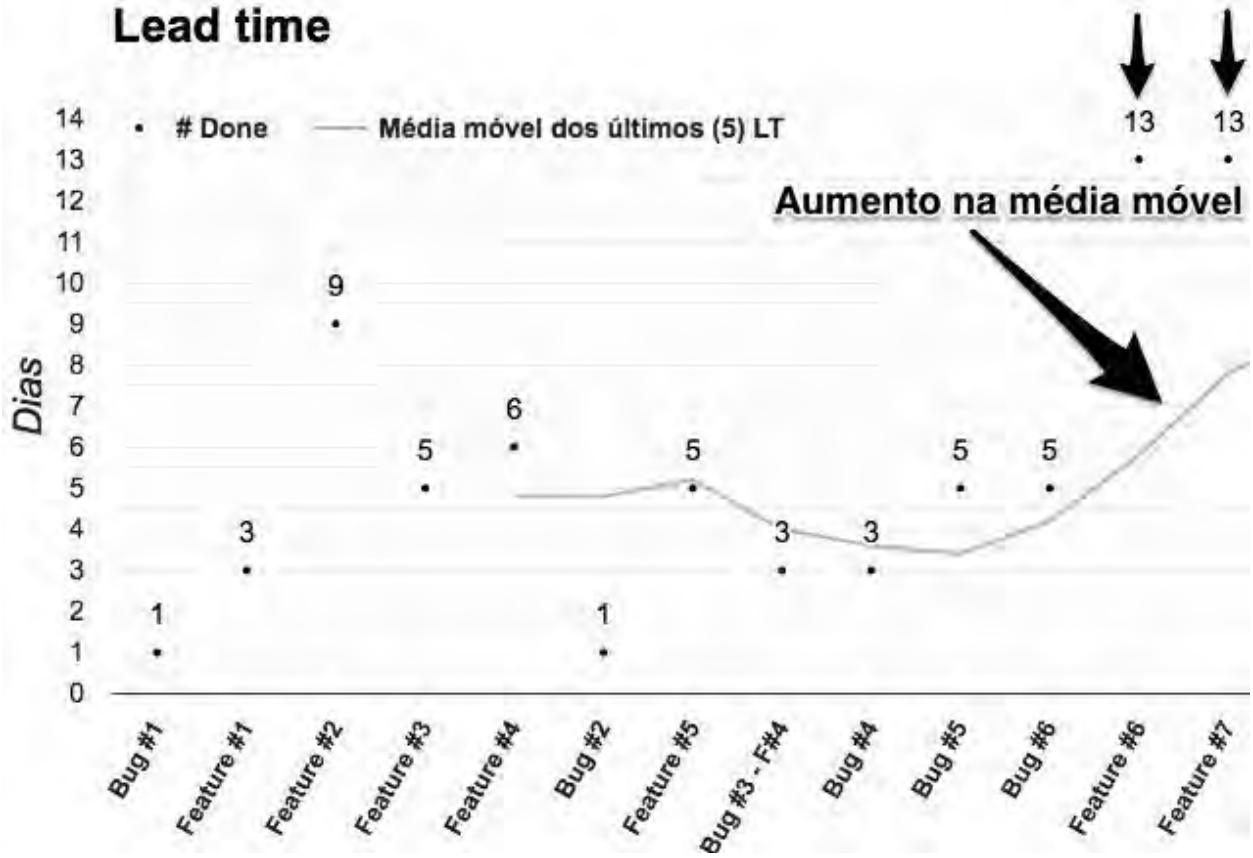


Figura 3.13: Estudo de caso: falta de padronização no tamanho das demandas

Sabendo que a equipe havia incorporado em sua rotina formas de preparar as demandas antes que elas percorressem o fluxo de desenvolvimento, um ponto de alerta foi gerado: mas, afinal, por que tais funcionalidades levaram tanto tempo?

Um dos desenvolvedores da equipe chegou a dizer que a **Feature#6** absorveu no seu desenvolvimento mais escopo do que havia sido pensado inicialmente. Costumo dizer que tal demanda materializou a síndrome do "Já Que".

Quantas vezes não nos deparamos com clientes ou gestores de produtos que nos dizem: "Já Que" estamos fazendo o relatório de vendas, por que não fazemos uma geração de cobrança automática a partir das informações apresentadas?

Mudanças no escopo de uma demanda refletem de forma direta no aumento repentino do *lead time*. Outros dois aspectos que podem prejudicar a saúde do *lead time* são a incerteza (técnica ou de negócios) e a complexidade na construção de uma demanda.

Se ao avaliar uma demanda a equipe percebe que ela possui alta incerteza ou alta complexidade, tenha muito cuidado e considere a possibilidade de quebrá-la em fatias menores.

No estudo de caso, a equipe percebeu que a **Feature#7** carregava muitas dúvidas de negócio, pois envolvia aspectos relacionados às áreas de logística e de faturamento. Como o especialista de negócios só se envolveu na etapa de homologação da demanda, muitas informações importantes surgiram nas etapas finais do fluxo de desenvolvimento.

Se você deseja que a sua equipe não sofra com variações no *lead time* por conta da falta de uma análise quanto à complexidade e à incerteza do que passará pelo fluxo de desenvolvimento, gostaria de lhe sugerir o uso da matriz apresentada no capítulo sobre WIP.

Caso a demanda carregue muita incerteza técnica, estimule a equipe a investigar formas de resolver tecnologicamente o problema, antes de assumir o compromisso com a área de negócios. Tal situação é muito comum quando há a adoção de uma nova tecnologia e, caso você esteja lidando com esse contexto, minha sugestão é estimular o desenvolvimento de demandas que coloquem à prova a novidade que está sendo proposta.

No caso de uma demanda carregar incertezas quanto ao aspecto de negócio, minha sugestão é a busca exaustiva de especialistas. A equipe do estudo de caso sofreu na prática a ausência de alguém que pudesse orientar a formulação da solução que estava sendo criada.

Costumo dizer que, quanto antes aproximarmos as pessoas de negócio ou usuários da solução técnica, maiores serão as chances de resolvermos de fato o problema daqueles que conviverão com a entrega gerada pela equipe.

Por fim, mas nem por isso menos importante, pratique a quebra de demandas complexas em pequenas, mesmo que a equipe tenha de conviver com o questionamento de que pequenas demandas não geram valor para o negócio.

Quando partimos a incerteza, estamos reduzindo risco, o que, para mim, representa valor, pois eliminamos desperdício, provamos conceitos abstratos mais rápido e validamos hipóteses complexas fracionadamente.

Outros problemas

A métrica *lead time* foi útil também na análise de dois outros problemas enfrentados pela equipe:

1. A saída de um dos desenvolvedores da equipe;
2. O crescimento do número de demandas em aberto (WIP).

Sobre a saída de um membro, o *lead time* cresceu, pois os desenvolvedores que estavam trabalhando até então na evolução do sistema possuíam um vasto conhecimento da base de código da aplicação.

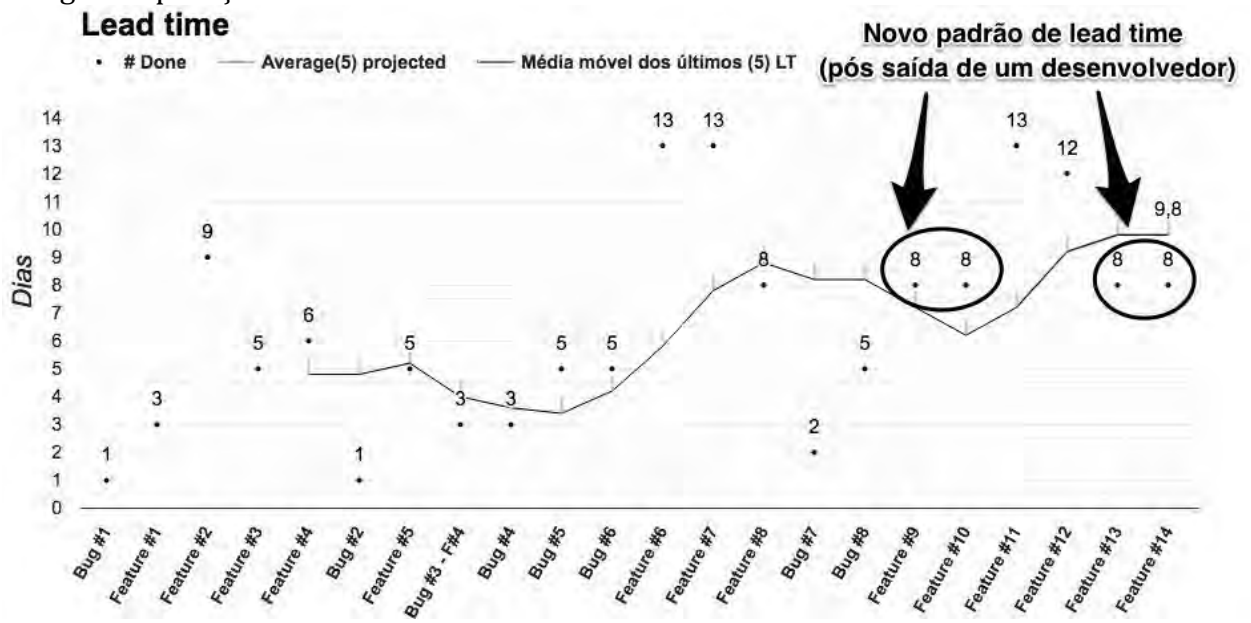


Figura 3.14: Estudo de caso: novo padrão a partir da saída de um desenvolvedor

Foi necessário um alinhamento com os geradores de demandas (usuários e áreas de negócio) a fim de dar visibilidade sobre o novo padrão de *lead time*, dada a nova composição da equipe.

Caso você sofra algum tipo de mudança na equipe, saiba que será o momento de gerar uma nova referência quanto ao *lead time* por conta do aprendizado que será necessário para que os novos integrantes possam performar melhor.

Uma técnica que tem sido útil para aumentar o processo de aprendizado de novos membros é o *pair programming*. Com o passar dos anos, tenho observado algumas vantagens ao se adotar tal prática, como:

- Aumento da atenção no processo de codificação, há uma queda no número de distrações e interrupções desnecessárias;
- As pessoas ficam mais atentas aos padrões de qualidade de código definidos;
- A necessidade de refatoração diminui;

- Há um aumento na qualidade dos testes que são realizados;
- Os desenvolvedores trocam conhecimento, o que gera soluções otimizadas e criativas.

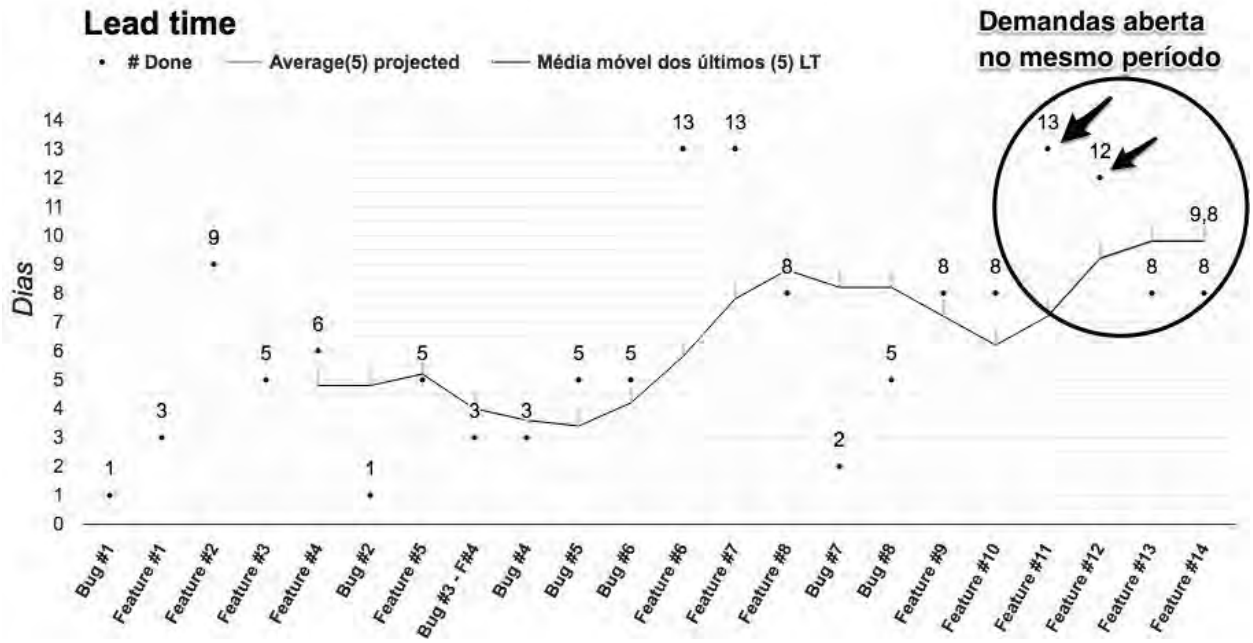
PAIR PROGRAMMING

Segundo a organização responsável por disseminar o Extreme Programming (http://www.extremeprogramming.org/rules/pair.html), *pair programmming* é o mesmo que dizer que todo código que será desenvolvido e publicado em produção será criado por duas pessoas que trabalham juntas em um único computador.

O *pair programming* aumenta a qualidade do software, sem afetar o tempo de entrega. É contra-intuitivo, porém, duas pessoas que trabalham em um único computador produzirão tanto se comparado a um contexto onde elas estivessem trabalhando sozinhas.

O *lead time* é um ótimo indicador de que, às vezes, a melhor estratégia é finalizar as demandas que estão em aberto antes de abrir novas frentes de trabalho.

Voltando ao estudo de caso, as funcionalidades sinalizadas pelas setas no gráfico a seguir sofreram com problemas relacionados à liberação e indisponibilidade de ambientes. Além disso, elas demoraram para serem homologadas, o que acabou refletindo no aumento do



lead time.

Figura 3.15: Estudo de caso: problemas externos e WIP

Em vez de focar na resolução dos problemas externos (ambientes) ou das esperas (homologação), a equipe decidiu começar o desenvolvimento de novas funcionalidades. No fundo, a equipe se sentiu no dever de começar a trabalhar em algo novo, pois se deparou com uma circunstância em que os problemas que estavam acontecendo fugiam da sua "alçada".

O esforço de gerenciar quatro demandas em paralelo (circuladas no gráfico anterior) fez com que a equipe perdesse o foco em entregar o que era mais importante, afinal, tinha de lidar com contextos distintos ao mesmo tempo (paralelismo). Ora era necessário discutir problemas de ambiente, ora era necessário pensar na funcionalidade que havia sido iniciada.

Portanto, caso você se depare com a possibilidade de abrir uma nova frente de trabalho por conta de um bloqueio: atenção! Por vezes, será melhor direcionar os esforços para não aumentar o *lead time* de algo que já está em aberto e que, se foi iniciado antes, é mais importante do que correr o risco de não terminar nada.

Ao evidenciar para as pessoas que estão acompanhamento a entrega de uma demanda quais são os fatores que oneram de forma direta o *lead time*, você está predispondo a criação de um ambiente que busca a cultura da melhoria contínua.

A partir da análise que fizemos do estudo de caso e do que aprendemos até aqui, você já se encontra apto a provar com números que há espaço para uma otimização do seu fluxo de desenvolvimento de software.

Na última seção deste capítulo, gostaria de compartilhar algumas técnicas avançadas para se analisar o *lead time* através de gráficos e estatística.

3.8 Técnicas para analisar o lead time de forma avançada

A seguir, compartilharei algumas formas não tão triviais de olhar para o *lead time*, para que você aprimore sua análise.

Dificilmente você terá acesso aos gráficos e as informações expostas a seguir a partir de softwares de gestão de projetos ou processo disponíveis no mercado. Por isso, sugiro que todo esse trabalho de análise mais avançada seja feito utilizando planilhas eletrônicas ou, até mesmo, através de ferramentas estatísticas como o R (se quiser conhecer, veja em https://www.r-project.org/).

Estatística descritiva e a distribuição de lead time

Ao iniciar o trabalho de análise do *lead time* de uma equipe, geralmente, começo de um conjunto de variáveis oriundas da estatística descritiva e que são comuns em nosso linguajar. São elas: média, moda, mediana e percentil. Antes de explorarmos tais estatísticas, gostaria de conceituá-las a fim de que não haja nenhum tipo de dúvida sobre os termos.

Para extrairmos a **média** de um conjunto de dados numéricos, precisamos somar os valores de todos os dados e dividi-los pela quantidade de dados. Já a **moda** é o valor mais frequente de um conjunto de dados. Você já parou para pensar o porquê de uma tendência de vestuário ou tecnológica ser chamada da "moda" da vez?

A **mediana** nada mais é do que uma medida de localização do centro da distribuição dos dados, definida da seguinte maneira: ordenados os elementos do conjunto de dados, a mediana é o valor que o divide ao meio, isto é, 50% dos elementos do conjunto de dados são menores ou iguais à mediana e os outros 50% são maiores ou iguais a ela.

Ainda sobre medidas de localização, definimos que os **percentis** são medidas que dividem a amostra ordenada (por ordem crescente dos dados) em 100 partes, cada uma com uma percentagem de dados aproximadamente igual. Quando dizemos que o percentil 75 possui o valor 7, estamos afirmando que 75% do conjunto de dados possui um valor inferior ou igual a 7, ou que 25% do conjunto possui um valor superior a 7.

Dado que estamos na mesma página quanto as definições sobre as variáveis, você deve estar se perguntando: por que raios tais informações serão úteis para a análise da minha equipe?

Basicamente, tais informações suportarão qualquer tipo de previsão quanto aos prazos de entrega para demandas futuras baseados no histórico. Ainda no que diz respeito ao histórico, minha sugestão é de que sempre seja feita a análise de um tempo mais recente.

Mudanças que aconteceram no fluxo de trabalho da equipe podem ter melhorado o *lead time*. E caso você esteja considerando momentos onde o *lead time* foi alto, as estatísticas serão influenciadas por tal comportamento que, no entanto, dado o novo contexto, já não tem mais sentido. Com isso, você removerá das medidas qualquer tipo de ruído que aconteceu no passado e que pode estar prejudicando uma visão mais realista de qual tem sido o tempo de entrega de uma demanda.

Para ilustrar o uso das estatísticas descritivas, nada melhor do que vermos a aplicação de uma análise em um exemplo. Imagine que o gestor do produto da equipe *XPTO* acaba de priorizar uma demanda importante para o negócio e chega com a seguinte pergunta para a equipe: quando poderemos entregá-la?

Antes que a equipe caia na tentação de compartilhar um prazo de entrega, tenha em mãos informações que sustentarão a previsão.

O primeiro passo é compreender os valores das estatísticas descritivas dos *lead times* passados. Extraindo tais dados do time *XPTO*, chegamos aos seguintes valores:

- **Média:** 13 dias.
- **Moda:** 8 dias.
- **Mediana:** 10 dias.
- **Percentil 75:** 18 dias.
- **Percentil 95:** 30 dias.

Analisando esses dados, percebemos que a moda não será uma boa medida para fazer a projeção, pois seu valor é menor do que a mediana (responsável por dividir o conjunto de dados ao meio).

Se usássemos 8 dias como sendo uma estimativa de entrega, estaríamos indo contra 50% da base histórica da equipe *XPTO*, o que geraria uma estimativa extremamente otimista. Sendo assim, você deve estar pensando que o caminho seria utilizar a média, correto?

Pois bem, muitas vezes, a média também não será uma boa referência, dado que seu valor pode ser distorcido por valores discrepantes. Ao longo dos anos, venho utilizando a mediana como medida de referência, pois ela é menos sensível a alguns casos extremos.

Voltando ao exemplo da equipe *XPTO*, ao analisarmos o histograma de *lead time*, percebemos que ele está enviesado para a direita, já que a média é maior que a mediana. Tal comportamento acontece, pois existem alguns valores grandes ao final da cauda da distribuição. Observamos no histograma ocorrências de demandas que levaram 24, 27, 30 e 33 dias respectivamente para serem finalizadas, o que por consequência, elevam a média.

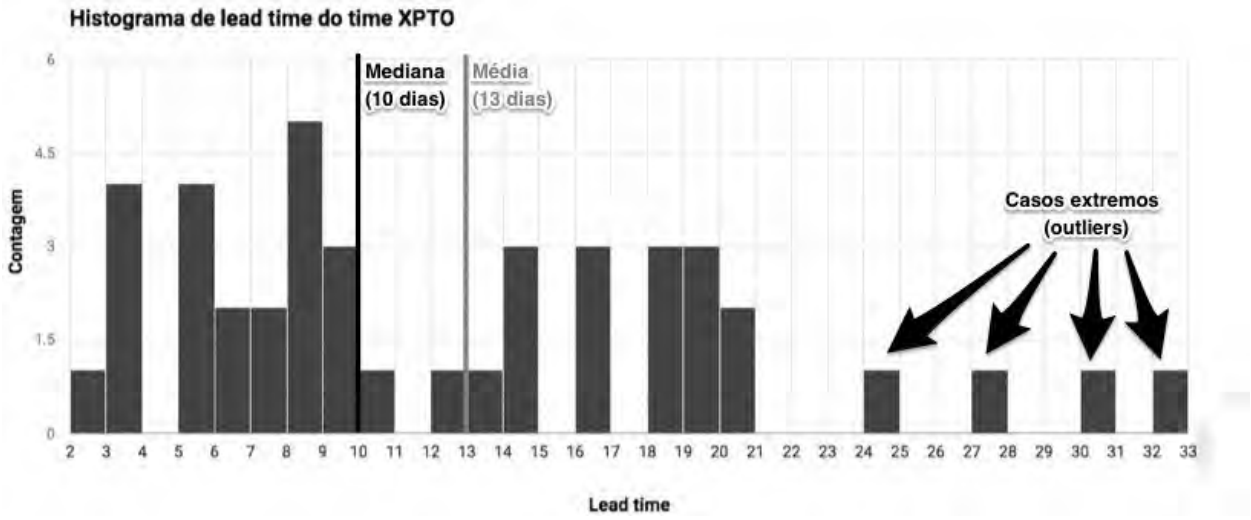


Figura 3.16: Histograma de lead time da equipe XPTO

Do que vimos até aqui, chegamos à conclusão de que, entre as variáveis levantadas, a mediana pode ser uma das estimativas a serem dadas ao gerente de produto da equipe *XPTO*. No entanto, por conta da complexidade que é trabalhar em um produto de software, devemos evitar uma abordagem determinística em um mundo que é receptivo por natureza a uma realidade probabilística.

Pensando nisso, comunicar estimativas de entrega a partir de intervalos passa a fazer muito sentido. Indo nessa linha, os valores representados pelos percentis 75 e 95 da distribuição de *lead time* podem ser importantes aliados na construção de cenários mais prováveis e pessimistas.

Voltando aos dados da equipe *XPTO*, temos que 75% das funcionalidades desenvolvidas até então levaram até 18 dias para serem concluídas, e que 95% das funcionalidades desenvolvidas precisaram de até 30 dias para serem finalizadas.

Com todas as informações coletadas até aqui, a equipe *XPTO* poderia responder a pergunta feita pelo gestor de produto quanto à estimativa de entrega da seguinte maneira: "Temos grandes chances de finalizarmos o desenvolvimento da nova funcionalidade em um período de 10 a 18 dias, no entanto, se formos conservadores em nossa estimativa, podemos prever que o prazo pode ser de até 30 dias, com 95% de chance de acertarmos a estimativa que está sendo compartilhada, dado o nosso histórico recente de entregas".

COMPREENDENDO A DISTRIBUIÇÃO DE LEAD TIME

Se você já estudou distribuições de dados, perceberá que a distribuição de *lead time* da sua equipe dificilmente será simétrica. De acordo com Magennis (2014) e Zheglov (2013), figuras importantes na aplicação da estatística para projeções de entregas de software, o tipo de distribuição que observamos quando lidamos com *lead time* é a chamada Weibull.

De acordo com Zheglov (2013), a Weibull faz parte de uma família de distribuições composta pelos parâmetros de forma (beta) e de escala (neta). Ela pode assumir as características de diferentes tipos de distribuições, porque a mudança do beta pode ajustar a forma da curva de distribuição.

Quando beta é igual a 1, a distribuição de Weibull é idêntica a uma distribuição exponencial. Em contrapartida, quando beta é igual a 2, a distribuição de Weibull é parecida com a distribuição de Rayleigh.

Após observar um vasto conjunto de dados, Zheglov (2013) constatou algumas características baseadas no parâmetro da forma (beta):

- Quando o beta é menor do que 1, a curva da distribuição é convexa ao longo de todo o intervalo. Esse tipo comportamento no parâmetro beta ocorre muitas vezes em operações de TI, atendimento ao cliente e em outras áreas nas quais não há um trabalho planejado.

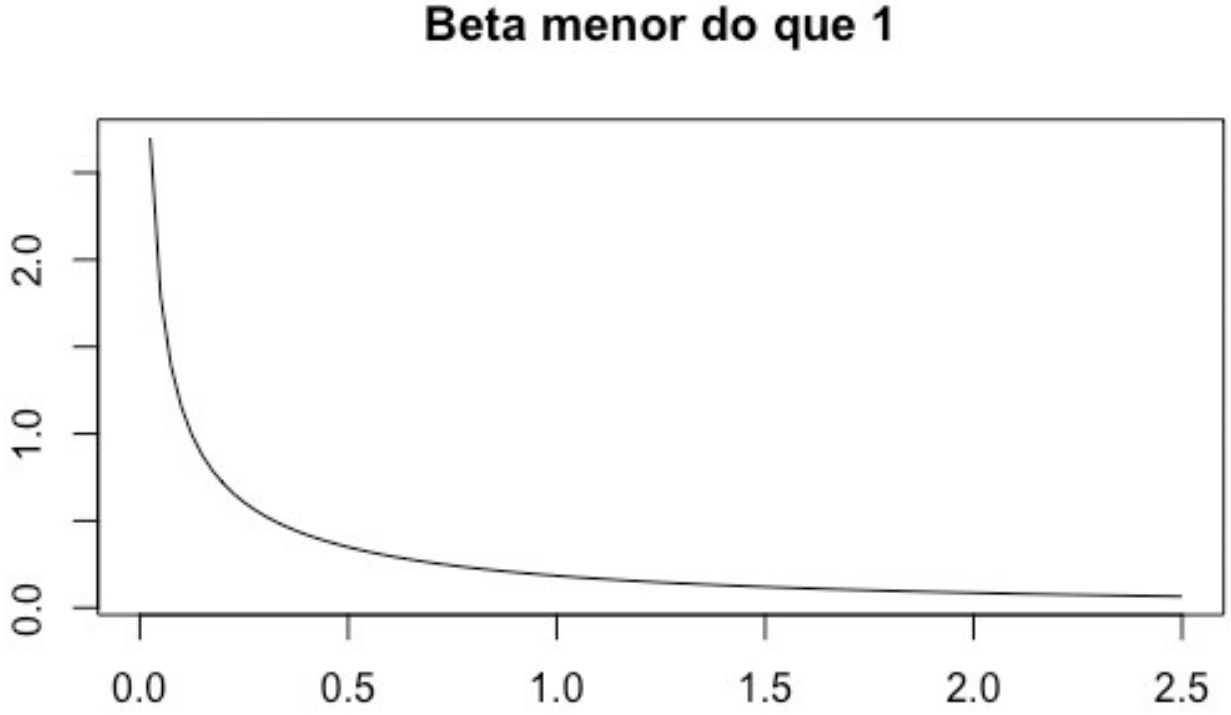


Figura 3.17: Distribuição de Weibull: beta menor do que 1

- Quando o beta está entre 1 e 2, a curva da distribuição é côncava no lado esquerdo até atingir o seu pico e, a partir dele, ela se torna convexa até o fim da distribuição. Esse tipo de comportamento acontece em ambientes em que há o desenvolvimento de novos produtos.

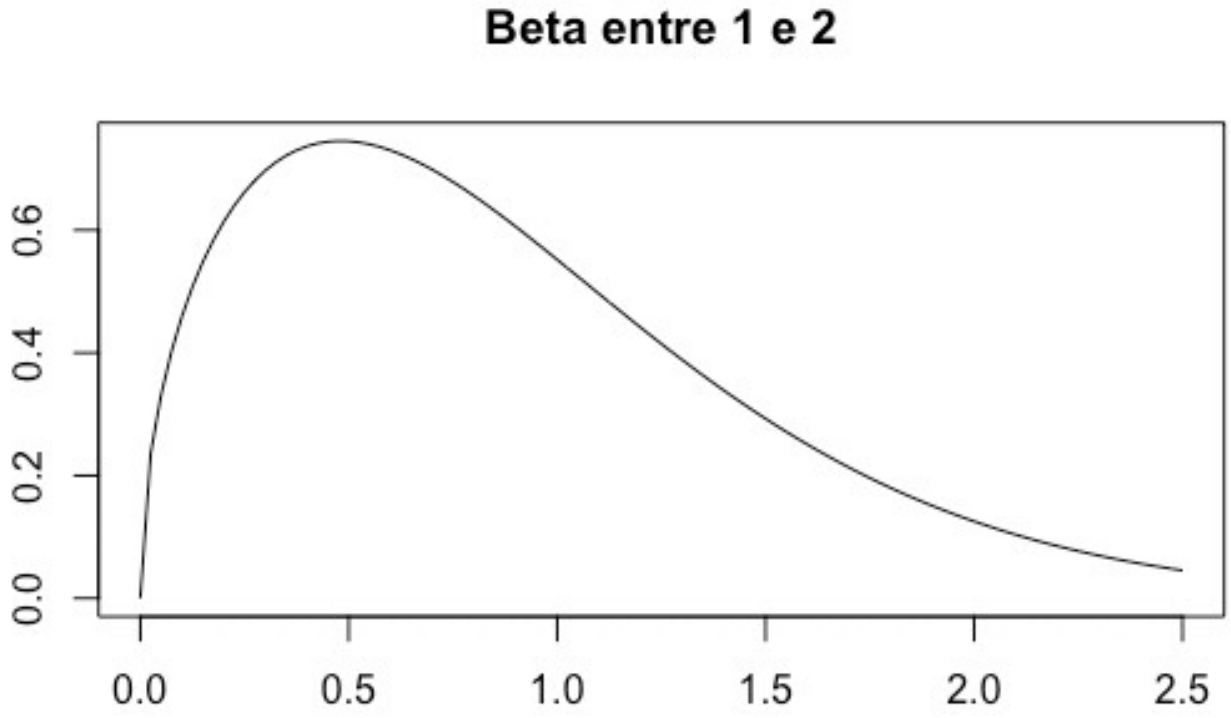


Figura 3.18: Distribuição de Weibull: beta entre 1 e 2

- Quando o beta é maior do que 2, a curva da distribuição é convexa no lado esquerdo, em seguida, se transforma em côncava, uma vez que sobe em direção ao pico, e se transforma em convexa novamente até o fim da distribuição. Esta forma de distribuição ocorre frequentemente em projetos de software em que o desenvolvimento acontece em um formato sequencial (cascata).

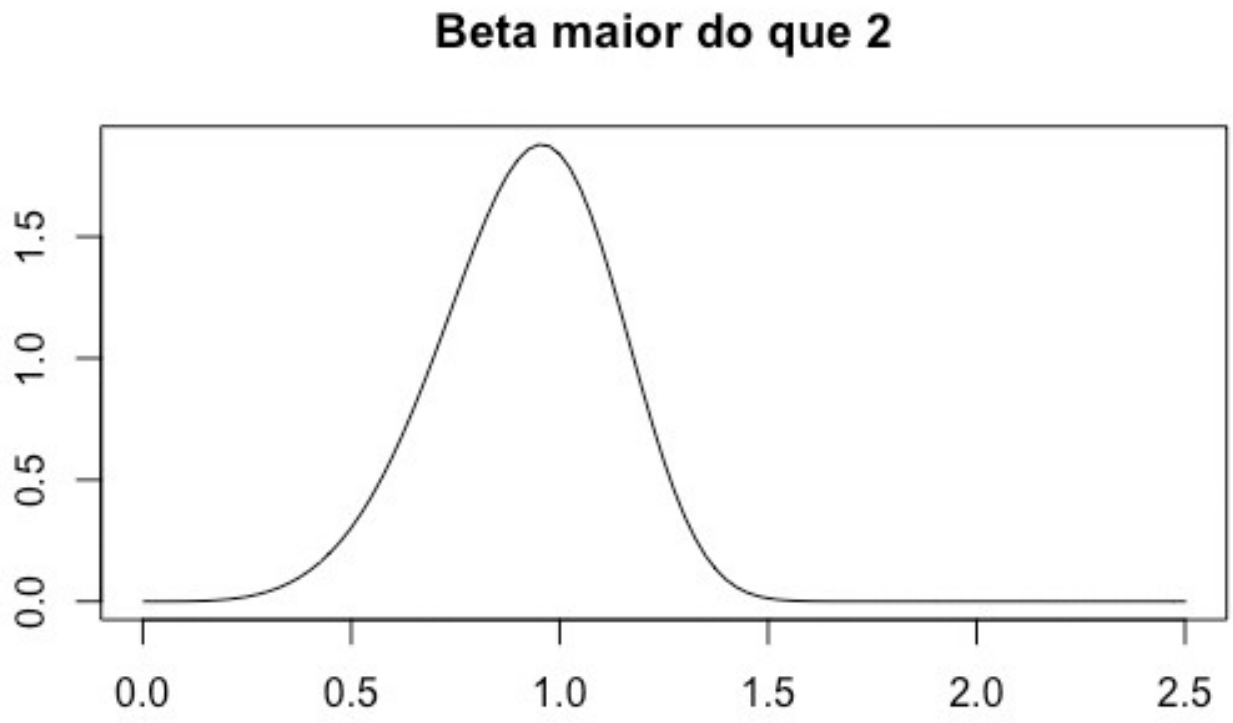


Figura 3.19: Distribuição de Weibull: beta maior do que 2

Formas de aprimorar a visualização da dispersão de lead time

Agora que aprendemos a extrair estatísticas descritivas para analisar a distribuição de *lead time*, gostaria de compartilhar algumas formas de visualizar a dispersão de dados para que você identifique em sua equipe casos extremos e avalie se o processo dela está com alta ou baixa variabilidade.

Todos os gráficos que serão expostos a seguir combinam no eixo horizontal as demandas que passaram pelo fluxo de desenvolvimento e no eixo vertical o *lead time* de cada demanda.

Uma dica que deixo é a de monitorá-los frequentemente (semanal ou quinzenalmente), de utilizá-los em cerimônias de retrospectiva (para fomentar a reflexão do aprendizado a partir do passado) e de compartilhá-lo em reuniões nas quais seja necessário dar visibilidade sobre a tendência do tempo que tem sido despendido para a entrega do que tem passado pelo fluxo de desenvolvimento.

O gráfico a seguir nos mostra a distribuição de *lead time* de uma equipe de desenvolvimento de produto. Nele são exibidos também 3 segmentos de reta que representam as referências de mediana (p50), percentil 75 (p75) e percentil 95 (p95).

Analisando os dados percebemos que a equipe possui uma baixa variabilidade entre o p50 e p75 (o *lead time* está em torno de 5 a 8 dias). Porém, quando comparamos o p75 com o p95, temos uma variabilidade maior (o *lead time* vai de 8 até 16 dias).

A dúvida que fica é: por que tamanha variabilidade entre o percentil 95 e o restante dos dados?

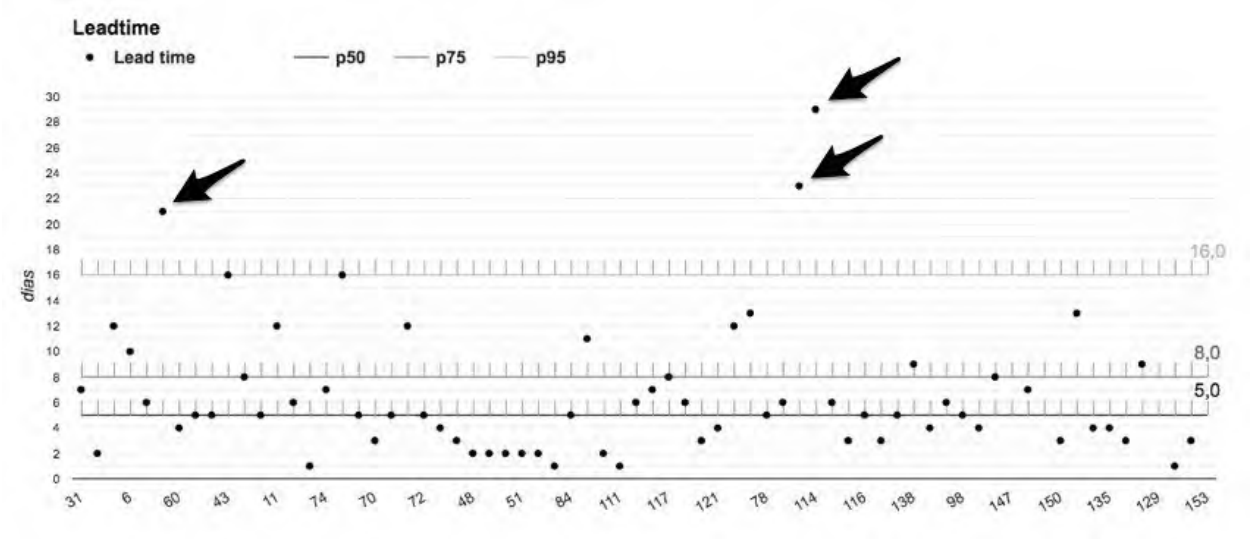


Figura 3.20: Análise de outliers

Ao verificarmos os casos extremos, no gráfico sinalizados pelas setas, chegamos à conclusão de que eles elevam o comportamento de *lead time*. Ou seja, eles são os responsáveis por distorcer o comportamento do conjunto de dados.

Venho observando alguns fatores que acabam sendo responsáveis por gerar situações de casos extremos e que você, assim como eu, já deve ter lidado em seu dia a dia. São eles:

1. Problemas externos ao desenvolvimento da demanda que são mapeados tardiamente (exemplo: integrações com sistemas externos que não funcionam ou ambientes que não estão disponíveis quando necessário);

2. Dependência de terceiros para a finalização de uma entrega (exemplo: o desenvolvimento de uma funcionalidade exige o trabalho de *design*, mas o especialista não faz parte da equipe);

3. Falta de definição do que se espera como resultado de uma demanda (exemplo: ausência de critérios na etapa de validação de uma funcionalidade ou da correção de um *bug*);

4. Mudança no escopo de uma demanda ao longo do desenvolvimento (exemplo: o escopo de uma funcionalidade absorve mais do que era esperado inicialmente por ela).

5. Subestimação da complexidade do desenvolvimento de uma demanda (exemplo: a equipe, de forma extremamente otimista, assume o desenvolvimento de uma entrega sem ponderar sua complexidade técnica ou de negócios, ou a equipe inicia o desenvolvimento de uma história de usuário mal dimensionada).

Se você estiver lidando com um processo estável, perceberá que as retas de percentil estarão próximas, como no exemplo a seguir, no qual observamos que 50% das entregas foram feitas em até 5 dias, 75% foram feitas em até 6 dias e de que 95% delas foram realizadas em até 9 dias.

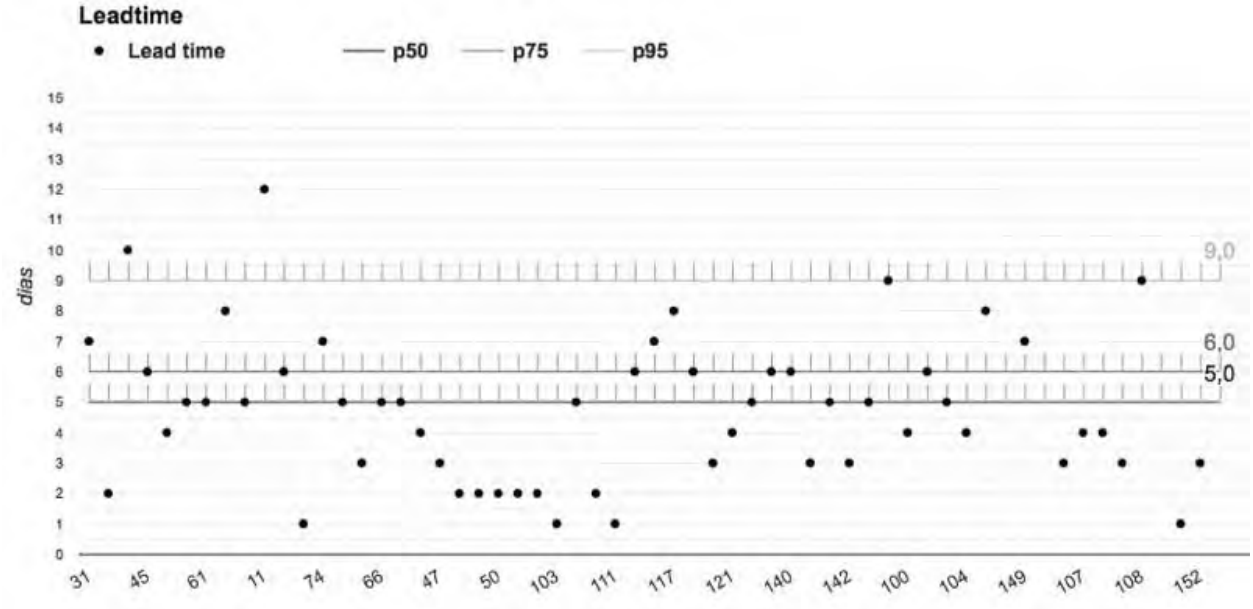


Figura 3.21: Exemplo de processo estável

3.9 Dicas

Antes de concluir, gostaria de compartilhar 6 dicas para você reduzir a variabilidade do *lead time*:

1. Faça um trabalho de mentoria com o(a) gestor(a) de produto (ou quem quer que seja o gerador de demandas) para que haja uma padronização na complexidade dos itens de trabalho (entregue valor com soluções simples);

2. Traga ferramentas para que a equipe diminua a complexidade e remova as incertezas técnica ou de negócios dos itens que serão trabalhados;

3. Analise a distribuição de *lead time* com frequência. No início do desenvolvimento de um projeto ou produto, a alta variabilidade do *lead time* será normal, porém, após algum tempo, é importante buscar um padrão;

4. Comunique aos *stakeholders* de um projeto ou produto qualquer a variação no *lead time* e desenvolva ações para controlar crescimentos;

5. Use os casos extremos (*outliers*) como uma oportunidade para aprender e melhorar o seu processo.

6. Limite o trabalho em progresso. Dessa forma, você estará monitorando o *lead time* de menos itens de trabalho e, consequentemente, melhorando a qualidade do que está sendo produzido pela equipe.

3.10 Recapitulando

Entender as demandas que passam pelo fluxo de desenvolvimento e analisar o tempo que elas levam para serem concluídas são atividades de alto valor quando estamos em busca de compreender formas de melhorarmos a construção de software

Neste capítulo, tivemos a oportunidade de discutir como a métrica de *lead time* pode nos oferecer insumos para a extração de uma visão do tempo que tem sido despendido para a geração de entregas dentro de uma equipe de desenvolvimento.

Todas as análises e visualizações que foram propostas podem ser feitas a partir de uma visão global do sistema, ou a partir de uma perspectiva por tipo de demanda (exemplo: análise independente do *lead time* de *bugs* e de funcionalidades).

Aprendemos também que termos em nosso fluxo de desenvolvimento demandas com *lead times* excessivamente distintos representa uma maior variabilidade no processo, o que enfraquece a previsibilidade das estimativas futuras.

A partir de agora, ao analisar o *lead time*, você passará a fazer questionamentos como:

- Quanto tempo tem sido necessário para entregarmos uma nova funcionalidade ou a correção de bug?
- Temos mantido um padrão no tempo de entrega?
- Se sim, como podemos manter?
- Se não, que tipo de mudança precisamos fazer para que o fluxo melhore?

Saiba que tudo o que vimos até aqui tem por objetivo fazer com que você e a sua equipe proponham melhorias no processo a partir de dados. Com as informações de *lead time* em mãos, vocês estarão aptos a identificar problemas, levantar fatores que estejam minando o fluxo de desenvolvimento e argumentarão a partir de fatos.

3.11 Referências

OHNO, T. *Toyota Production System:* Beyond Large Scale Production. Productivity Press, 1988.

MAGENNIS, T. *The Economic Impact of Software Development Process Choice - Cycle-time Analysis and Monte Carlo Simulation Results.* 48th Hawaii International Conference on System Sciences, 2015. Disponível em: http://focusedobjective.com/wp-content/uploads/2014/09/The-Economic-Impact-of-Software-Development-Process-Choice-Cycle-time-Analysis-and-Monte-Carlo-Simulation-Results.pdf.

THOMAS, S. *Lead time versus Cycle Time – Untangling the confusion.* Set. 2015. Disponível em: http://itsadeliverything.com/lead-time-versus-cycle-time-untangling-the-confusion.

ZHEGLOV, A. *How to Match to Weibull Distribution in Excel.* Ago. 2013. Disponível em: https://connected-knowledge.com/2013/08/01/how-to-match-to-weibull-distribution-in-excel/.

CAPÍTULO 4

Medir o número de entregas de uma equipe

4.1 Introdução

Geralmente, equipes que trabalham operando ou construindo software precisam compartilhar uma visão sobre a sua capacidade de atendimento aos problemas reportados pelos clientes, ou a compreender a tendência de entrega de novas funcionalidades.

Se você faz parte de uma equipe que não monitora a sua capacidade de entrega, saiba que em algum momento você será cobrado por isso. Caso você já monitore e esteja sofrendo com aquela sensação de trabalhar demais sem conseguir de fato entregar algo, provavelmente você precisa olhar com mais cuidado para o seu processo. Ou ainda, se a equipe na qual você atua está convivendo com momentos de um alto número de entregas e outros de escassez, você deve estar precisando ajustar algum ponto de gargalo no seu fluxo de trabalho.

Depois de aprendermos a analisar o tempo que um item de trabalho leva para ser concluído (*lead time*) e a importância de monitorar o trabalho em progresso (WIP), neste capítulo teremos a oportunidade de conhecer a métrica chamada *throughput*. Você perceberá que a análise dela lhe ajudará a responder questões como:

- Quantas demandas a equipe está conseguindo entregar dentro de um espaço de tempo?
- A equipe está vivendo um cenário no qual o número de entregas vem aumentando?
- Existe algum tipo de bloqueio atrapalhando a capacidade de entrega da equipe?

Caso você já conheça ou monitore o *throughput*, não deixe de conferir ao longo do capítulo a seção na qual compartilho a relação desta métrica com o *lead time* e o WIP, a partir de uma leitura de lei de Little. Tenho certeza de que as premissas por trás da lei serão úteis para que você melhore o dia a dia do processo da sua equipe.

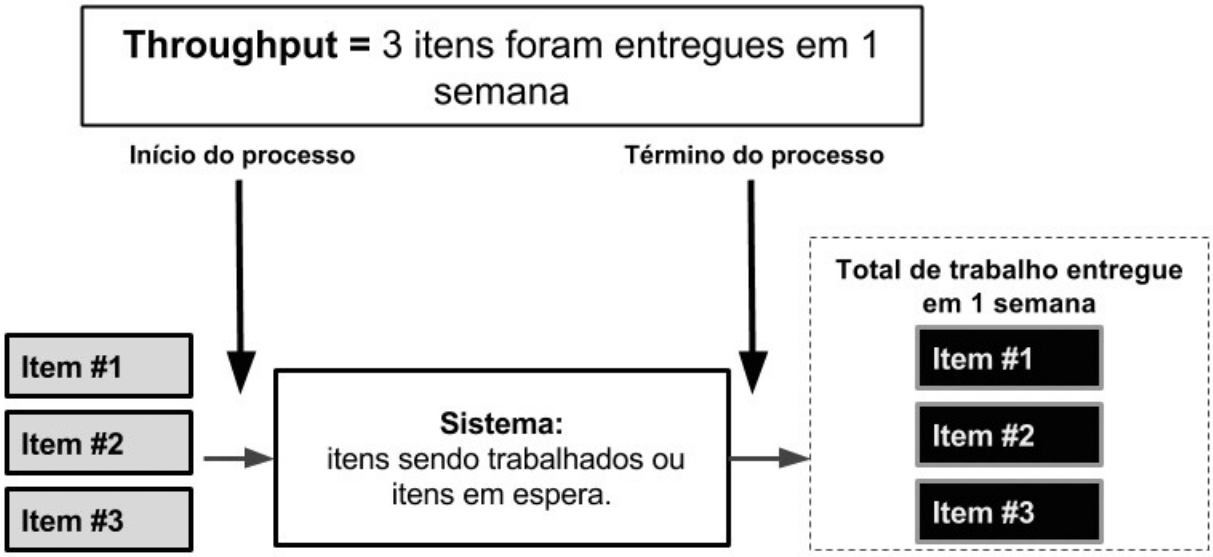
Outro tópico interessante que abordarei neste capítulo diz respeito às maneiras não tão convencionais de visualizar e analisar o *throughput*. A partir de gráficos e estatísticas descritivas, você e a sua equipe poderão aprimorar a forma como analisam o fluxo de entregas. Por fim, compartilharei um estudo de caso em que relato a rotina de uma equipe que trabalha na operação de um produto de software e que conseguiu ser mais eficiente através de uma série de aprendizados gerados da medição do *throughput*.

Lendo o conteúdo que será apresentado, você terá a oportunidade de aprimorar a saúde do processo da sua equipe e a dar mais visibilidade sobre a capacidade de entrega dela. Ao final, espero que você inclua em seu portfólio o *throughput* como mais uma métrica útil na proposição de melhorias baseadas em dados no processo de desenvolvimento de software.

4.2 O que é o throughput?

Throughput nada mais é do que uma medida que determina quantas unidades de trabalho foram completadas em uma unidade de tempo. Olhando pela perspectiva de vazão (ou saída), é possível afirmar que o *throughput* é uma medida que determina quanto o meu fluxo é capaz de processar.

Conforme ilustrado na figura seguinte, trazendo para o dia a dia de desenvolvimento de software, diríamos que, em determinada semana (unidade de tempo), o *throughput* de uma



equipe foi de três funcionalidades (quantidade de itens).

Figura 4.1: Exemplo de cálculo do throughput

Vale destacar que não há uma unidade de tempo universal para medir o *throughput*, portanto, caberá a você ou a sua equipe defini-la. Existem contextos em que faz sentido medir o *throughput* por dia, semana, iteração (*sprint*) etc.

Caso você esteja em uma situação na qual é preciso analisar quantos chamados sua equipe está atendendo para efetuar melhorias no processo de suporte aos clientes, medir o

throughput diariamente será interessante. Se você sabe que o responsável pela validação das demandas que são criadas pela equipe na qual você atua só pode estar presente junto do time quinzenalmente, sugiro avaliar o *throughput* por quinzena.

Agora, se a equipe em que você atua está lidando com uma entrega crítica onde a data de entrega é uma variável sensível, monitore semanalmente o *throughput* para que seja possível propor melhorias e ações dentro de um curto espaço de tempo. Afinal, responder às mudanças o quanto antes será um fator crítico para que haja a entrega de valor dentro do prazo esperado.

Uma dúvida que pode estar surgindo na sua cabeça neste momento é: *throughput* seria o sinônimo de velocidade (*velocity*), utilizada comumente em equipes que adotam o Kanban? A resposta é não.

Velocidade, conforme definição da Zheglov (2016), representa o total de *story points* entregues por uma equipe ao longo de uma iteração (*sprint*). *Story points* é uma medida usada por equipes na mensuração do esforço necessário para desenvolver determinado requisito, funcionalidade, história do usuário, *bug* etc.

No caso de *throughput*, estamos lidando com o total de itens de trabalho entregues ao longo de uma unidade de tempo que não precisa estar restrita ao limite de uma iteração, diferentemente da variável de velocidade que descreve o número de pontos entregues dentro de uma iteração. Como exemplo, ao final de uma iteração, sua equipe pode ter entregue 5 histórias de usuário e 30 *story points*.

Faço este parêntese para que você não caia na tentação de acreditar que *velocity* e *throughput* são métricas equivalentes. A partir do exemplo citado anteriormente, na iteração seguinte o time poderia continuar entregando 5 histórias de usuário, porém, entregando 20 *story points* e não 30. A motivação da diminuição na velocidade tem relação com a equipe ter trabalhado em demandas de menor esforço.

Como um estágio inicial, para se conhecer o quão difícil será construir algo, *story points* tende a ser uma informação útil para que a equipe meça quanto de esforço ela é capaz de produzir em um espaço de tempo. Com o passar do tempo, a equipe passa a conhecer mais das nuances do processo de construção de algum tipo de demanda. Nesta fase, utilizar a métrica de *story points* deixa de fazer sentido.

Ter em mãos o *throughput* auxiliará você a compreender qual tem sido o montante de trabalho entregue em um período de tempo (semana, quinzena, mês). Além disso, auxiliará a identificar se a equipe está criando uma tendência de aumento no número de entregas.

Por fim, será útil na identificação de algum fator que esteja bloqueando a cadência de entrega da equipe, gerando questionamentos como: será que a equipe está com tamanho ideal? Será que existe algum gargalo ao longo do processo?

4.3 Como visualizar o throughput de uma equipe?

Agora que definimos o que é *throughput*, compartilharei neste tópico formas de analisá-lo para que você possa aplicá-las em sua equipe. Um ponto importante de deixar claro é o significado de um item entregue.

Já trabalhei com equipes nas quais o conceito de pronto girava em torno da liberação de uma funcionalidade em um ambiente de aprovação. O problema de se considerar algo entregue, mesmo sabendo que existam etapas posteriores, é a falsa impressão de progresso. Afinal, do que adianta 5 demandas "prontas" para serem aprovadas, mas, sem uma expectativa de quando elas de fato serão consumidas pelos clientes ou usuários? Portanto, sempre que possível, considere um item finalizado quando este estiver gerando valor para o seu negócio ou usuário final.

A forma mais direta de representar o *throughput* se dá a partir de um gráfico de colunas. No exemplo que veremos a seguir, será criada uma visualização ao longo de um período de tempo (como semanas, *sprints*, meses etc.) de qual tem sido o número de entregas de uma equipe. Com o objetivo de compreender se há uma tendência de aumento ou queda na métrica de *throughput*, teremos também representada no gráfico uma média móvel das últimas 5 ocorrências de *throughput* (a definição do valor utilizado na média móvel fica a seu caráter).

Imagine uma situação em que você, como Gerente de Desenvolvimento da equipe XPTO, precise ajudar a equipe a explorar o *throughput* passado pensando na geração de melhorias futuras. Como a equipe realizou um trabalho de refinamento bem estruturado, as demandas desenvolvidas tiveram um esforço muito similar para serem construídas, e por isso você e a equipe optaram por analisar o *throughput* sem distinguir tamanho de esforço.

Existirão situações (não deixe de ler o estudo de caso) nas quais será útil analisar o *throughput* por tipo de demanda (como bugs, histórias do usuário, melhorias técnicas etc.), ou por tamanho de esforço da demanda (por exemplo, item simples, item com complexidade moderada ou item complexo). O que podemos observar no gráfico a seguir?

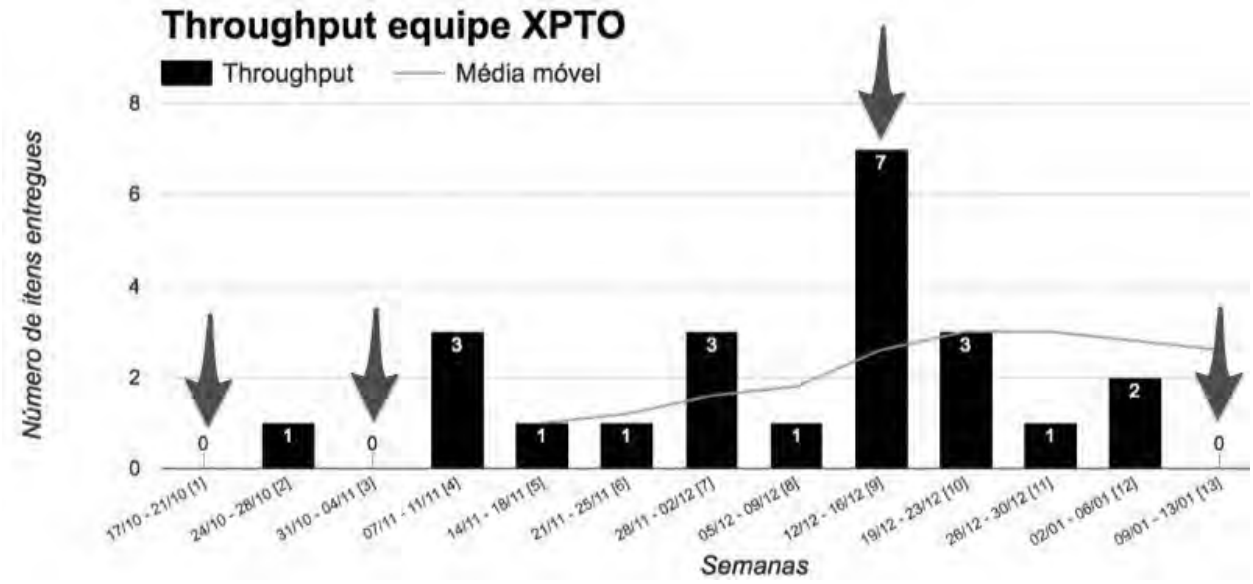


Figura 4.2: Gráfico de throughput da equipe XPTO

Primeiramente, observamos que em aproximadamente 1/4 das semanas (3 em 13) o *throughput* da equipe foi 0. Tal comportamento pode ser um sinal de alerta para que a equipe identifique fatores que estejam dificultando a fluidez do processo (ambiente indisponível, lentidão no processo de validação etc.).

Existirão situações em que será útil definir um limite inferior de *throughput*. E caso a equipe atinja ou opere abaixo de tal valor, medidas corretivas serão automaticamente tomadas para que o patamar de entrega retome o crescimento.

Outra informação que pode ser extraída a partir da visualização é a tendência de aumento ou queda no número de entregas a partir da média móvel. Voltando ao time XPTO, percebemos que o número de entregas por semana oscilou entre 1 a 3 itens entregues.

Portanto, é razoável dizer que, se as demandas que vierem no futuro tiverem um comportamento de incerteza e complexidade parecido com as que foram feitas no passado, a equipe tenderá a entregar no mínimo 1 e no máximo 3 itens por semana. Tal informação poderá ser útil quando a equipe estiver discutindo estimativas de entrega.

Ao olhar para a semana 9, percebemos um caso atípico, já que nela o time conseguiu finalizar 7 itens. Assim como é importante o time discutir as motivações do porquê o *throughput* foi baixo, o mesmo vale para casos extremamente positivos. Ao longo dos anos, venho observando que quando as equipes estão sob pressão ou se encontram em um estágio de otimismo, casos como o da semana 9 tendem a ser perigosos, por conta de a equipe acreditar que aquele será o comportamento padrão no futuro.

Acredito que você já tenha visto equipes projetando entregas a partir de cenários extremamente otimistas, não? Avaliar o que causou tamanho crescimento no *throughput* se faz necessário para que haja um alinhamento de expectativas, pois, assim com os vales (quedas), picos (aumentos) tendem a serem casos de exceção (como demandas reprimidas por conta de um atraso na aprovação de uma publicação, acúmulo de itens em homologação etc.).

Antes de finalizar este tópico, gostaria de compartilhar quatro situações que podem ocasionar uma queda no *throughput*:

1. Itens de trabalho mal definidos — como histórias de usuário com critérios de aceite ruins;
2. Débitos técnicos — como reescrever uma parte do software inesperadamente;
3. Gargalos no processo — como equipe de testes sobrecarregada e fluxo de publicação com problemas;
4. Mudança no escopo de um item de trabalho ao longo do desenvolvimento — como inserção de um requisito não funcional no meio de uma iteração.

Concluindo este tópico, gostaria de sugerir: meça o *throughput* semanalmente. Mesmo que você atue em um processo que lida com ciclos de trabalho quinzenais, ao analisar a vazão semanal, você identificará problemas com maior antecedência e poderá colocar em prática ações em um curto espaço de tempo, evitando surpresas indesejadas no futuro.

4.4 Qual a relação entre WIP, lead time e throughput?

Após aprendermos algumas maneiras de analisar o *throughput*, compete discutirmos a relação entre as métricas vistas nos capítulos anteriores (*lead time* e WIP) e o *throughput*.

Provavelmente você já leu ou ouviu falar da aplicação da lei de Little em processos de desenvolvimento de software. Conforme explicarei a seguir, a fórmula ilustrada a seguir é uma variação da fórmula original de Little. Ela nos diz que o tempo médio de um item no processo de desenvolvimento (*lead time*) é igual a divisão do número médio de itens no processo (WIP) pelo número médio de itens que saem do sistema por unidade de tempo (*throughput*).

$$L = W/\Lambda$$

Em que:

L = Média do *lead time*

W = Média de itens em WIP

λ = Média de *throughput*

O resultado da aplicação da variação da lei em equipes de desenvolvimento de software nos diz que, em geral, quanto mais coisas são trabalhadas em um determinado momento (em média), mais tempo levará para que cada uma dessas coisas termine (em média).

Ao alterar qualquer uma das variáveis da equação, você certamente afetará as outras. Em outras palavras, a fórmula revela possíveis alavancas para a realização de melhoria nos processos.

Agora que você compreendeu a existência de uma relação entre as variáveis, gostaria de deixar uma importante observação: pare de começar mais trabalho (WIP) quando estiver com demandas muito antigas no seu processo (*lead time* alto).

Ao contrário do que você possa imaginar, o *throughput* não aumentará porque o seu processo não estará dando vazão e os itens que estarão dentro dele passarão a ter o seu valor deteriorado (quanto mais tempo você demorar para entregar algo, menor será o *feedback* de utilidade da solução por aqueles que precisam dela). A melhor estratégia em situações como essa é reduzir o WIP, parando de começar e começando a terminar.

Ao utilizar a variação da lei de Little em equipes de desenvolvimento de software, buscamos compreender o que já aconteceu para tentar, de forma não determinística, e sim direcionar o presente a partir do histórico. É como se buscássemos melhorar o nosso processo a partir das evidências observadas no passado.

EXPLICANDO A LEI DE LITTLE

A lei de Little, criada por John Little no início dos anos 60, tem relação com um conjunto de estudos desenvolvidos para compreender sistemas de fila (SIGMAN, 2009). Segundo o próprio autor, um sistema de fila consiste em objetos discretos que podem ser chamados de itens, que chegam a alguma taxa em um sistema. Os itens podem ser carros em uma cabine de pedágio, pessoas em uma linha de um restaurante *fast food*, aviões em uma linha de produção ou instruções esperando para serem executadas em um computador.

Os itens entram no sistema, juntam-se a uma ou mais filas, eventualmente recebem um serviço e saem em um fluxo de saída. O serviço pode ser o atendimento na cabine de pedágio, um prato entregue no restaurante de *fast food* ou a configuração do painel de bordo de um avião. Na maioria dos casos, o serviço é o gargalo que cria a fila.

Como resultado da sua pesquisa, Dr. Little descobriu uma conexão entre a taxa média de entrada de itens no sistema, o número médio de itens no sistema e a tempo médio de um item no sistema. Matematicamente, a relação entre as três métricas se parece com a equação: $L = \lambda * W$.

Em que:

L = Número médio de itens no sistema de fila.

λ = Número médio de itens chegando ao sistema por unidade de tempo.

W = Tempo médio de um item no sistema.

Para exemplificar a aplicação da forma, Little usou o exemplo de um rack de vinho. Digamos que você tem um rack de vinho que, em média, tem 100 garrafas nele. Vamos ainda dizer que você reabastece o rack a uma taxa média de duas garrafas por semana. Tendo em mãos tais informações, podemos determinar quanto tempo, em média, uma determinada garrafa gasta alojada no rack. A partir da equação, temos que L é igual a 100 e λ é igual a 2. Aplicando tais valores, temos que uma dada garrafa de vinho fica, em média, 50 semanas no rack.

As premissas para a aplicação da lei de Little são:

1. Um sistema de filas estável — a média do WIP no início do período do cálculo deve ser muito próxima da média no final do período e a idade média dos itens que compõem o trabalho em progresso não deve estar mudando, nem para mais, nem para menos.

2. Um período de tempo arbitrariamente longo sob observação para garantirmos que o processo seja estacionário — média, variância e estrutura de autocorrelação não mudam no decorrer do tempo. Isto é o mesmo que dissermos que os dados estão em um plano liso, sem tendência, variância constante no decorrer do tempo, uma estrutura de autocorrelação constante no decorrer do tempo e nenhuma flutuação periódica.

3. Que o cálculo seja realizado usando unidades consistentes — por exemplo, se o tempo médio de um item no sistema é expresso em dias, então o número de itens chegando ao sistema também deve ser declarado em termos de dias.

Quando comparamos a fórmula originalmente criada por Little (1961) e a fórmula citada no início do capítulo, percebemos que existe uma diferença entre elas: a versão original lida com **taxa de entrada** e a versão adaptada para o contexto de desenvolvimento de software trata de **taxa de saída**.

Algumas mudanças nas premissas são necessárias quando passamos a olhar a lei de Little sob a perspectiva de *throughput* e não a partir da taxa de entrada. Segundo Vacanti (2015), são elas:

1. A entrada média ou taxa de chegada deve ser igual à taxa média de saída. É como se disséssemos que a quantidade média de demandas que entra no nosso processo de desenvolvimento é igual à quantidade média de demandas que sai do processo. Por exemplo, ao analisar as últimas semanas de trabalho da equipe, percebeu-se que, em média, entraram no processo e foram entregues duas histórias de usuário.

2. Todo o trabalho que é iniciado será eventualmente concluído e sairá do sistema. É comum entender que algo não faz mais sentido e tirá-lo do fluxo de desenvolvimento (como: histórias de usuário que a equipe de negócios percebeu não fazerem mais sentido, pois não beneficiarão os usuários de um produto). O ponto em questão não é se isso é certo ou não, apenas que a lei de Little deixará de ter validade se ocorrer algo assim no processo de desenvolvimento.

3. A quantidade de WIP deve ser aproximadamente a mesma no início e no final do intervalo de tempo escolhido para o cálculo. Assim como a taxa de entrada e saída, a quantidade de trabalho em progresso deve ser constante.

4. A idade média do WIP não deve aumentar nem decrescer. Em outras palavras, é como se o tempo médio necessário para entregar os itens que passam pelo processo seja o mesmo.

5. *Lead time*, WIP e *throughput* devem ser medidos usando unidades consistentes. Reforçando, se usamos dias para medir o *lead time*, teremos a mesma unidade para analisar o *throughput*.

Compreender as premissas por trás das equações é a chave para aplicar a lei corretamente. Baseado nas premissas anteriores, você, como orientador do processo de desenvolvimento da sua equipe, poderá sugerir diretrizes como:

- Só iniciaremos um novo trabalho aproximadamente na mesma proporção em que terminarmos o que já está no processo.

- Colocaremos todos os esforços necessários para terminar qualquer trabalho iniciado e minimizaremos a possibilidade de se trabalhar com itens que possam ser descartáveis. Dinâmicas de priorização e compreensão do que precisa ser feito são indicadas para que o esforço da equipe seja direcionado à entrega do que é necessário.

- Se o trabalho da equipe por ventura ficar bloqueado, faremos tudo o que pudermos para desbloqueá-lo o mais rápido possível. Para isso, se faz necessário o desenvolvimento do senso de comprometimento de todos os elementos da equipe, para que o fluxo de trabalho funcione.

- Antes de trazermos um novo item de trabalho para o fluxo de desenvolvimento, analisaremos se existe algo em WIP que esteja com um *lead time* alto e daremos foco para finalizá-lo.

As diretrizes compartilhadas são dicas para que você possa estabilizar o sistema de trabalho da sua equipe. Ao aplicá-las, você perceberá que o processo ficará mais previsível, o que trará mais segurança para quem estiver ao redor (por exemplo, equipe de negócios, clientes etc.).

Antes de concluir este tópico, gostaria de tranquilizá-lo quanto a necessidade de calcular as métricas a partir das fórmulas que vimos até aqui. Na realidade, o que realmente gostaria é que você esqueça a parte aritmética, afinal, como estamos aprendendo até aqui neste livro, podemos medir, sempre que precisarmos, *lead time*, WIP ou *throughput*.

Compreender a lei de Little trata-se de entender as premissas que precisam estar em vigor para que ela funcione. Utilize as premissas como um guia para refinar o seu processo. Sempre que o seu processo não seguir alguma das premissas, existe algum ponto demandando melhoria e a previsibilidade estará em risco. Tal imprevisibilidade pode se manifestar através de demandas com *lead times* altos, excesso de itens em aberto (aumento do WIP) ou falta de cadência de entrega (baixo *throughput*).

Deixando de olhar de forma combinada as variáveis que estão em torno do processo, você estará correndo o risco de gerar previsões de entrega otimistas demais, o que, quando não concretizadas, gera frustração (clientes e *stakeholders* ficam com as expectativas não atendidas) e perda em oportunidades de negócio. Enfim, lembre-se: você e sua equipe possuem mais influência sobre o que ocorre no fluxo de trabalho do que imaginam.

4.5 O uso do throughput a partir de um caso real

Neste tópico, teremos a oportunidade de analisar o *throughput* a partir da realidade de uma equipe que realizava a manutenção e evolução de um produto de software. O aprendizado que a equipe teve utilizando a métrica foi importante para a realização de melhorias no processo de atendimento aos usuários do produto e para o fluxo de trabalho.

Sem as análises que foram realizadas, provavelmente a equipe continuaria tendo dificuldades em ter visibilidade sobre a cadência de entrega dos diferentes tipos de demanda que passavam pelo processo. Espero que, ao final deste estudo de caso, você aprenda e aprimore as técnicas de análise compartilhadas.

Estudo de caso — Evolução e manutenção de uma plataforma de hospedagem de sites

O caso se passou em uma equipe de uma empresa focada em soluções digitais e que lidava diretamente com a manutenção e evolução de uma plataforma de hospedagem de sites.

A aplicação havia sido criada há 5 anos e contava com um total de 10.000 usuários ativos distribuídos pelo Brasil. Além disso, a receita gerada pela plataforma era responsável por 35% do faturamento da empresa que contava com outras soluções em seu portfólio.

Um dos grandes desafios da equipe era lidar com os contextos de manutenção, que envolvia correções de erros e apoio à equipe de suporte e de evolução do produto, a partir da criação de novas funcionalidades baseadas nas necessidades do mercado ou para se diferenciar da concorrência.

A priorização das demandas que passavam pelo fluxo de desenvolvimento era algo difícil. A equipe era pequena e precisava dar respostas rápidas aos problemas e as evoluções. Ela era composta de 3 desenvolvedores, 1 testador, 1 gestor de produto e 1 Agile Coach.

Por conta da característica de se trabalhar com múltiplas demandas, a equipe optou por utilizar o método Kanban como forma de gerir o fluxo de trabalho. Além disso, limitou-se a quantidade de itens que estariam abertos no processo (WIP) em 3 itens.

O ferramental de engenharia permitia que a equipe realizasse publicações automáticas no ambiente de homologação. No entanto, para o ambiente de produção, era necessária a intervenção de uma área terceira (operações).

Para as novas funcionalidades que seriam criadas para o produto, a equipe decidiu estimar o esforço necessário para criá-las a partir de uma categorização P, M e G. O objetivo de adotar tal técnica foi de que era muito mais fácil utilizar medidas relativas do que absolutas no processo de avaliação das demandas.

Itens avaliados como P eram aqueles nos quais a equipe enxergava baixa dificuldade na sua implementação, o que representava um esforço de no máximo 3 dias para serem completados. Os itens classificados como M eram aqueles com um grau de esforço próximo de 5 dias. Por fim, os itens G eram aqueles no qual o esforço seria maior do que uma semana de trabalho.

Antes de trazer uma demanda de manutenção para o fluxo de trabalho, a equipe avaliava se esta estava clara o suficiente para ser trabalhada, ou se deveria ser investigada melhor pela equipe de atendimento aos clientes. A partir do momento em que a equipe assumia a resolução de uma falha, ela deveria ser entregue por completo.

Semanalmente a equipe se reunia para definir as prioridades e para avaliar como andava a saúde do processo. Quando convinha, eram realizadas sessões de retrospectiva com o intuito de promover melhorias.

O fluxo de trabalho da equipe estava dividido nas seguintes etapas:

- **Para ser feito**: etapa na qual uma demanda recebida e analisada passava a fazer parte do fluxo de desenvolvimento da equipe.
- **Em desenvolvimento**: etapa em que uma demanda era implementada em um ambiente específico para os desenvolvedores.
- **Aguardando homologação**: etapa na qual os desenvolvedores liberavam determinada demanda implementada para a validação do gerente de produto no ambiente de homologação.
- **Em homologação**: etapa onde eram validadas e homologadas as demandas pelo gerente de produto.
- **Aguardando publicação**: etapa em que a demanda aguardava para ser publicada em produção pela área de operações.
- **Em produção**: etapa na qual os usuários eram notificados de que determinada demanda havia sido disponibilizada.

A equipe vinha sofrendo pressão de clientes, diretoria e equipe de suporte ao usuário por não dar visibilidade sobre o que estava sendo entregue para o produto. Portanto, decidiu-se que a métrica de *throughput* passaria a ser contabilizada quando uma demanda atingisse o estágio "Em produção".

Os tópicos que seguirão a exposição do caso mostrarão alguns problemas que foram identificados a partir da análise e do monitoramento do *throughput*. Você perceberá que os gráficos utilizados na análise combinam no eixo horizontal semanas e, no eixo vertical, o número de entregas realizadas.

Como informação complementar, baseado nos 5 últimos valores de *throughput*, foi traçada uma média móvel. Tal medida visa apresentar uma tendência de aumento ou queda nos valores de *throughput*.

Problema #1: throughput único

A equipe estava entusiasmada com a possibilidade de dar visibilidade sobre o seu ritmo de entrega, pois ela tinha o sentimento de que trabalhava demais, mas as pessoas não as reconheciam por tal.

Em uma reunião mensal entre os responsáveis pelo portfólio de produtos da empresa, o gerente do produto de hospedagem apresentou o *throughput* das 4 primeiras semanas de monitoramento da métrica.

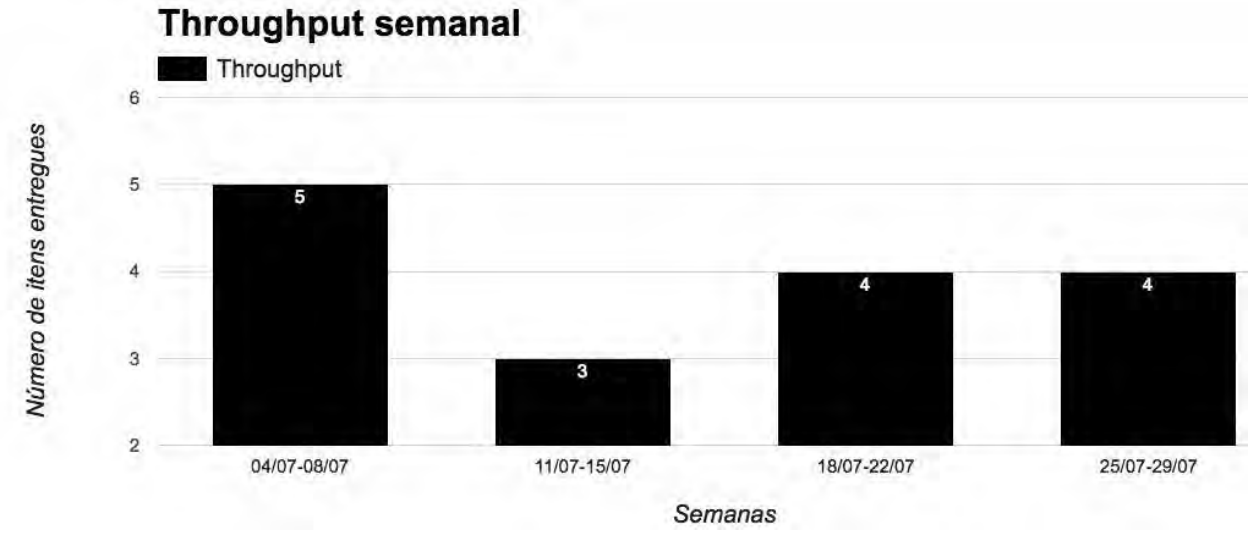


Figura 4.3: Estudo de caso: 4 primeiras semanas monitorando o throughput

Os participantes da reunião observaram que a equipe conseguiu entregar ao longo das semanas mais demandas (de 3 a 5) do que ela havia colocado como limite do fluxo de trabalho, que eram 3. O gerente de produto foi elogiado, pois conseguiu dar visibilidade de que o time produzira um bom volume de entregas de valor aos usuários da plataforma.

Após a apresentação dos resultados, um dos diretores que participavam do encontro questionou o gerente de produtos sobre o que de fato havia sido entregue. Segundo ele, o índice de satisfação quanto ao atendimento dos chamados tinha crescido, porém a empresa estava sendo questionada pelos seus clientes sobre a falta de novas funcionalidades no produto. O gerente do produto não soube responder e ficou com a pendência de avaliar se de fato o produto estava deixando de receber evoluções com frequência.

Figura 4.4: Estudo de caso: quebra na análise do throughput por tipo

Após uma nova análise do *throughput*, agora realizando uma quebra por tipo de demanda entregue, ele observou que:

1. Foram entregues 3 demandas de manutenção (*bug*) por semana.
2. Nas semanas 1, 3 e 4 foram entregues, respectivamente, 2 funcionalidades simples, 1 funcionalidade mais complexa e 1 funcionalidade simples.

A equipe concluiu que, proporcionalmente, estava colocando mais esforço na resolução de problemas no produto do que na construção de novas funcionalidades. O gestor de produto percebeu que as demandas priorizadas por ele estavam focando em demasia na área de suporte e atendimento ao cliente (interessado interno do produto) em detrimento ao usuário final (interessado externo do produto).

DEFINIÇÃO DE EFICIÊNCIA E EFICÁCIA

O que é eficácia?

Geralmente está ligado a fazer o que deve ser feito (fazer as coisas certas). Este conceito tem a ver com o foco em uma determinada direção. A pergunta orientadora é: estamos fazendo o que precisa ser feito para atingirmos nosso objetivo?

O que é eficiência?

> Geralmente está ligado a como fazer o que tem para ser feito. Tem como direcionamento realizar as operações com menos recursos (menos tempo, menor orçamento, menos pessoas, menos matéria-prima etc.). A pergunta orientadora é: estamos fazendo da forma mais otimizada possível?

O primeiro problema apontado no estudo de caso nos mostra a importância de identificar o significado por trás da métrica de *throughput*. Vimos que a equipe estava sendo muito eficiente no trabalho (vinha com um volume de entregas acima do que havia sido proposto), porém pouco eficaz (deixou de questionar o que era prioridade para o produto).

Se você deseja que a sua equipe seja eficiente e eficaz antes de assumir a entrega de qualquer trabalho, faça com que as pessoas pensem nas seguintes perguntas:

- Qual é o resultado esperado por quem está demandando (exemplo: cliente, usuário etc.)?
- Conseguimos simplificar a solução para que possamos validá-la junto ao usuário final, evitando assim retrabalho?
- Quais os tipos de demandas o time vem lidando no seu dia a dia? O que é prioridade no momento?

Ao priorizar as demandas que atravessarão o fluxo de desenvolvimento da sua equipe, não deixe de definir os respectivos valores de negócio. Tal informação será útil para que você alinhe as expectativas e defina o que será mais importante para o negócio junto aos interessados pelo resultado do que será produzido.

Algumas equipes podem trabalhar com o conceito de valor como sendo o retorno gerado a partir do investimento que será despendido em uma demanda, também conhecido como ROI (*Return on Investment*). Para calculá-lo, você deverá ter mãos o total esperado de receita para o desenvolvimento da nova demanda, dividido pelo custo total para desenvolvê-la.

Para exemplificar, imagine que uma funcionalidade gerará R$1.000,00 de receita e exigirá um custo de R$10.000,00. Podemos dizer então que, para cada R$1,00 retornado em receita, será gasto R$10,00, o que nos dá uma relação baixa de retorno versus o investimento. Idealmente, o valor expresso pelo ROI deve ser maior do que 1.

Outra forma de definir o valor de negócio de uma demanda é através do custo do atraso ou *cost of delay*. O custo do atraso é o impacto econômico do atraso de uma demanda. Em outras palavras, é o quanto a organização deixará de ganhar se a demanda não estiver pronta em uma determinada data. Para calculá-lo, você precisará estimar quanto de lucro a demanda que será entregue gerará.

Digamos que você espera que uma nova funcionalidade do seu produto, quando entregue, gere para a empresa um resultado de R$5.000,00 semanais. Neste exemplo, o custo do atraso é de R$ 5.000,00 por semana. Em outras palavras, para cada semana que a demanda estiver atrasada, a empresa perderá a oportunidade de ganhar R$ 5.000,00. Mais detalhes sobre a análise do *cost of delay* em projetos de software podem ser vistos em Plataformatec (2016).

Uma última forma de se calcular o valor de negócio de uma demanda é a partir de variáveis que sejam críticas para o seu contexto. Como exemplo, imagine que, para sua equipe, cada demanda deve ser analisada a partir do potencial de geração de receita, da geração de economia, da sua criticidade para o negócio etc.

As variáveis poderão ser ponderadas conforme maior relevância e a equipe atribuirá notas dentro de uma escala (escala de 1 a 5, por exemplo) para cada uma delas. Ao final, a equipe dará maior prioridade para as demandas que tiverem maior pontuação. Mais detalhes sobre essa aplicação desta abordagem podem ser vistos em Taller (2016).

Outro cuidado que deve ser tomado ao analisar o *throughput* diz respeito ao risco em se comparar equipes apenas levando em consideração o valor absoluto apresentado pela métrica. Para não cometer esse equívoco, lembre-se: equipes lidam com contextos distintos.

Sem conhecer a natureza das demandas, as etapas do fluxo de desenvolvimento, as limitações do sistema de trabalho, as restrições técnicas e de negócio, entre outras variáveis, ao colocar duas equipes na mesma régua (*throughput*), é como se você estivesse dizendo que maçã e banana são a mesma coisa. Portanto, como orientação, nunca compare o *throughput* entre equipes. Será uma análise distorcida e de pouco valor.

Problema #2: aumento no lead time gerando queda no thoughput

Após um novo alinhamento entre o gestor de produtos e as áreas de negócio, a equipe passou a focar esforços no desenvolvimento de evoluções do produto baseado no que traria maior valor ao negócio.

Acompanhando semanalmente o gráfico de *throughput*, a equipe começou a perceber que o número de entregas estava diminuindo. Um sinal de alerta foi disparado quando a equipe percebeu que, em duas semanas consecutivas, o *throughput* foi zero.

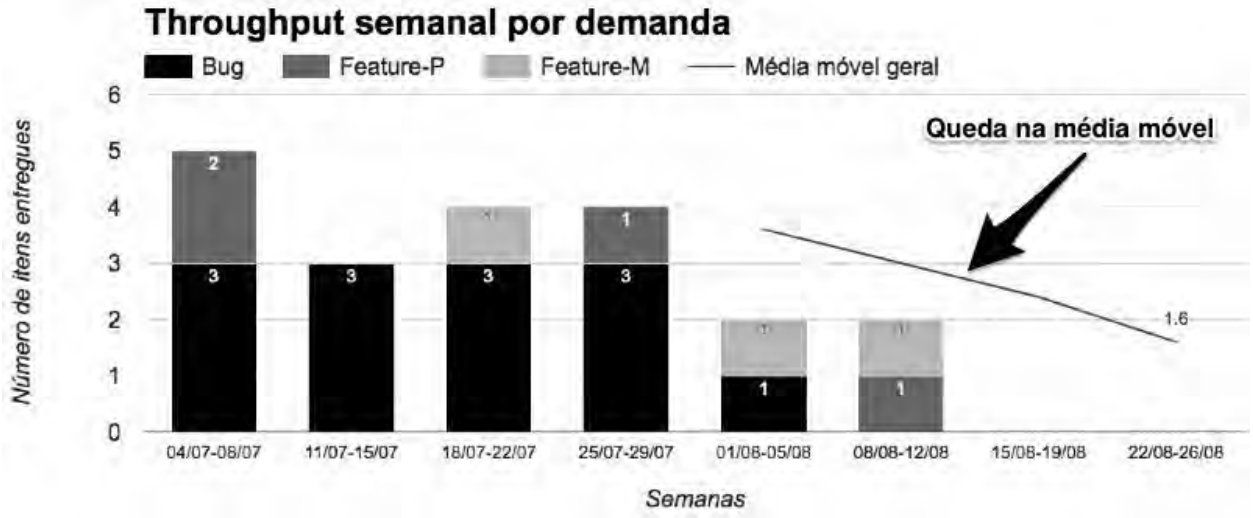


Figura 4.5: Estudo de caso: queda no throughput

Antes da reunião mensal de produtos, a equipe decidiu realizar uma retrospectiva com o objetivo de entender o que estava acontecendo, e identificou que as demandas de maior prioridade e que trariam maior valor tinham uma complexidade maior para ser finalizada.

Mesmo controlando o número de itens que passavam pelo processo (o limite era de 3), a equipe se deu conta de que estava carregando ao longo do fluxo de desenvolvimento 3 funcionalidades de esforço grande. Para sustentar ainda mais a análise, a equipe decidiu analisar a métrica de *lead time* e percebeu que as demandas em progresso estavam levando mais tempo do que as que haviam sido concluídas, o que ocasionou uma queda no *throughput*.

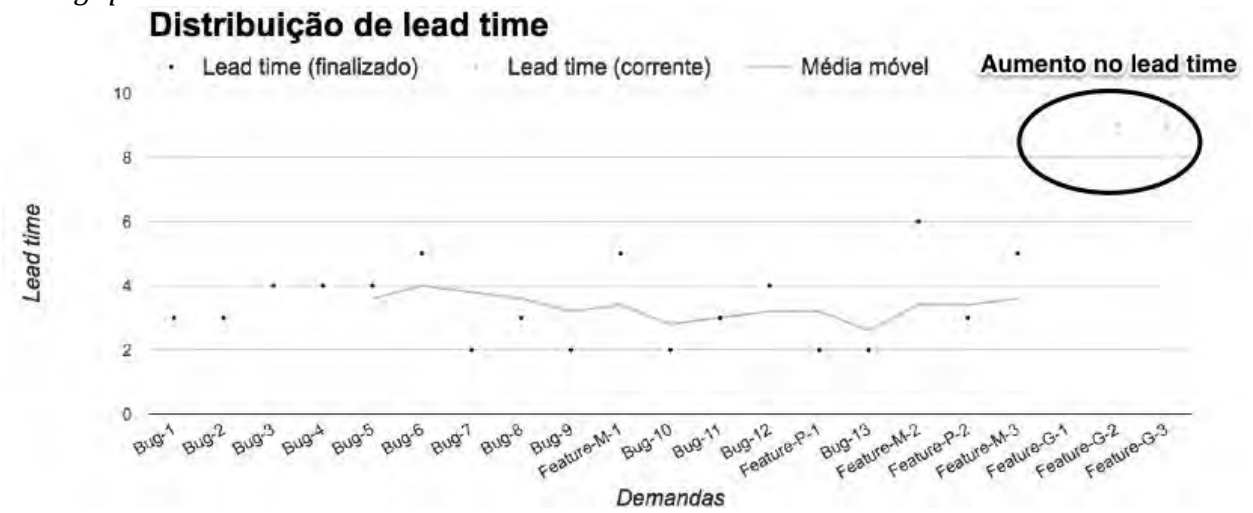


Figura 4.6: Estudo de caso: cruzando a análise do lead time com o throughput

A equipe identificou que, além da maior complexidade das demandas, existiam algumas incertezas técnicas que estavam dificultando a boa fluidez do desenvolvimento. Como plano de ação, a equipe definiu que, em casos em que existissem incertezas técnicas, provas de conceito seriam realizadas antes que fossem assumidos compromissos de entrega.

Na posse de tais informações, o gerente de produto conseguiu apresentar as motivações envolta da queda no *throughput* e conseguiu alinhar as expectativas junto aos diretores da empresa.

O segundo problema apontado no estudo de caso demonstrou a relação entre WIP, *throughput* e *lead time*. Dada a alta complexidade dos itens que estavam em WIP, naturalmente o *lead time* deles aumentou, o que afetou o número de entregas. Parece soar estranho, mas nem sempre ampliar o número de trabalho em progresso fará com que a sua vazão aumente, por isso se torna importante avaliar as três métricas em conjunto.

Como primeira dica para que a sua equipe não sofra os efeitos do aumento do *lead time*, reduza as incertezas das demandas. Quanto antes a equipe tirar dúvidas técnicas e de negócio, mais familiarizada com a solução ela estará. Descobrir problemas no início do desenvolvimento é mais barato do que em etapas futuras.

Ao longo do tempo, tenho percebido que as equipes não têm o hábito de avaliar um item que será construído antes da sua entrega já estar comprometida com os *stakeholders* de um projeto ou produto. O grande problema é que tais demandas indefinidas farão com que o processo se torne ineficiente. Os *stakeholders* não assimilarão a real capacidade de entrega do time (*throughput*).

A outra dica visa lembrá-lo de complementar a análise do *throughput* a partir das informações de *lead time*. Ao longo deste livro, estamos percebendo que as métricas servem como um software trabalham. Elas servem como forma de tangibilizar o trabalho que foi e está sendo feito. Lembre-se sempre: *lead time* baixo gerará mais *throughput*.

Acompanhando equipes que vem adotando métodos ágeis no processo de construção de software, tenho visto situações nas quais existe muito trabalho em progresso (excesso de WIP) aberto há um bom tempo (*lead time* alto) e com pouca entrega de valor (baixo *throughput*).

Para fazer com que sua equipe volte a trabalhar em um ritmo previsível, comece a melhorar o que está em progresso, compreendendo o que acontece com o fluxo de trabalho, administrando a capacidade de entrega e removendo possíveis bloqueios existentes. Reforce a mensagem de que é mais importante parar de começar e começar a terminar.

Por fim, reduza o tamanho dos entregáveis para que o *lead time* comece a ter um padrão. Excesso de variabilidade é um dos grandes detratores da previsibilidade, portanto, fomente

que a equipe trabalhe com demandas parecidas sempre que possível. Dessa forma, ela ficará mais segura da sua capacidade de entrega.

Problema #3: gargalos no processo

Depois de um período de duas semanas sem entregas, a equipe conseguiu colocar em produção na semana entre os dias 29/08 a 02/09 duas das três funcionalidades categorizadas como grandes.

Na semana seguinte, a equipe resolveu que aumentaria o limite WIP (número de demandas em aberto que seriam permitidas no fluxo de desenvolvimento) de 3 para 5. A decisão buscou aumentar a capacidade de paralelismo, pois existiam erros reportados pelos clientes há um tempo e que precisavam ser resolvidos.

Passadas duas semanas da decisão, a equipe percebeu que, mesmo aumentando o limite WIP e trabalhando com demandas de esforço menor (a equipe possuía domínio sobre o que estava sendo desenvolvido), o *throughput* não aumentava.

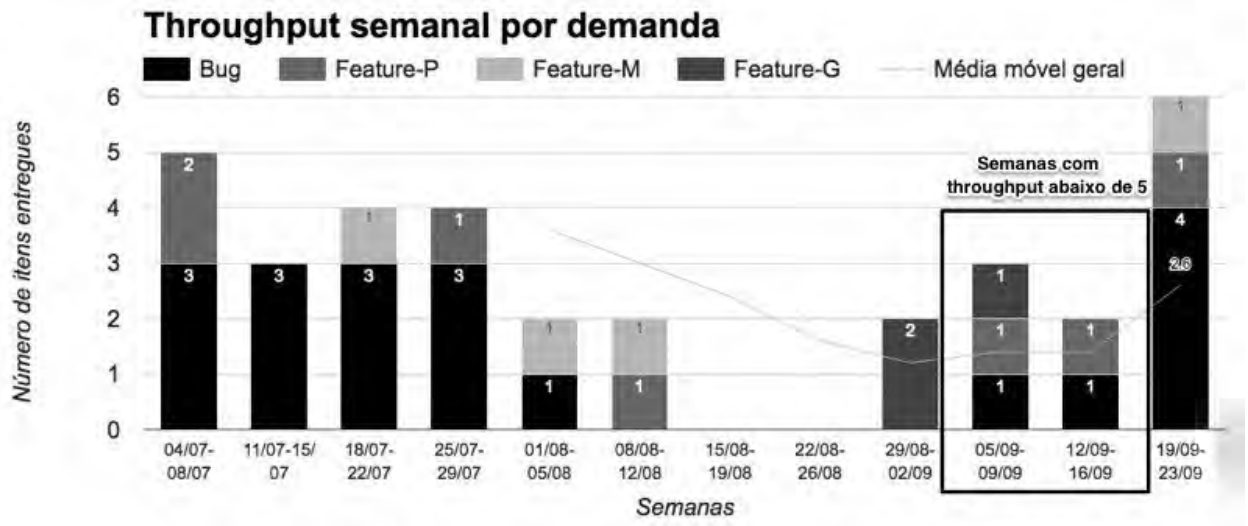


Figura 4.7: Estudo de caso: throughput abaixo do limite WIP

Ao analisar o motivo, a equipe percebeu que o gestor de produto e a área de operações estavam se tornando um gargalo no processo. Tal constatação ficou evidente quando a equipe visualizou o quadro que mapeava o fluxo de trabalho naquele momento.

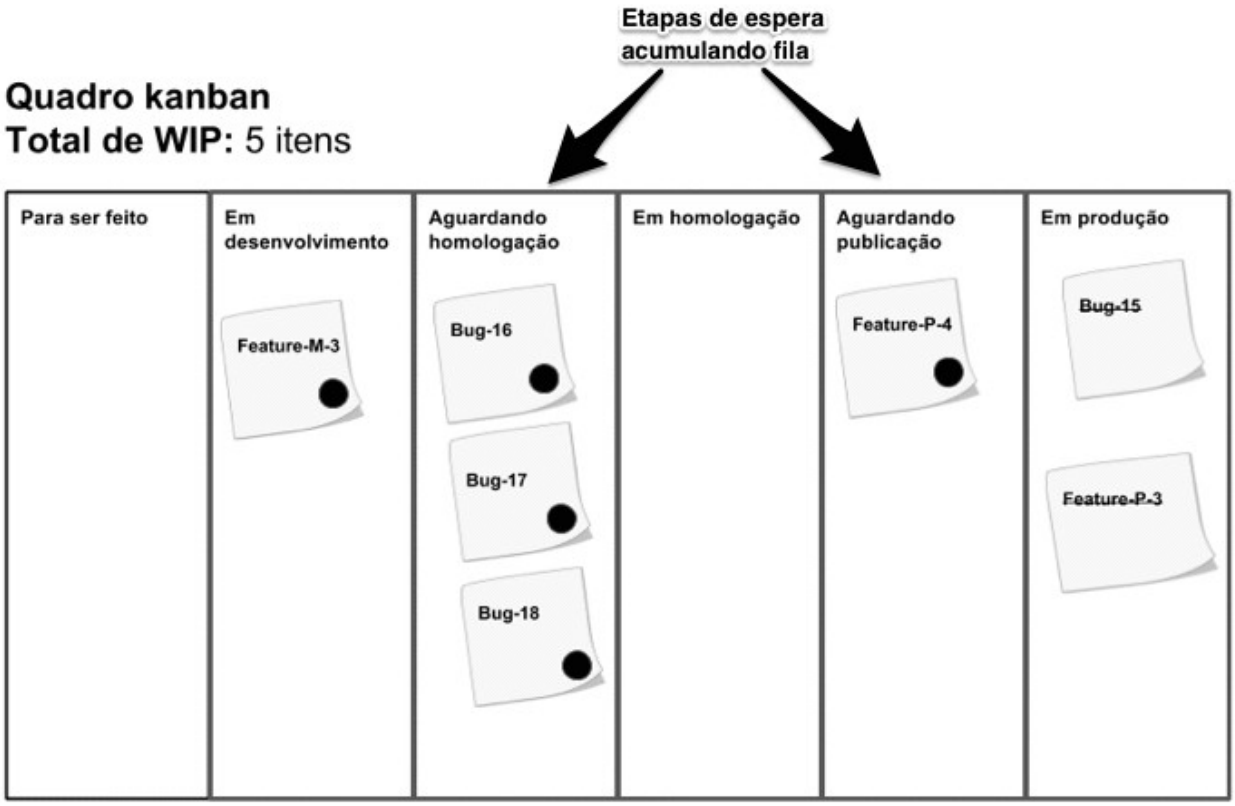


Figura 4.8: Estudo de caso: quadro Kanban evidenciando gargalos no processo

Mesmo tendo demandado um baixo tempo de desenvolvimento, a funcionalidade "Feature-P-4" carregava um alto valor de negócio, pois representava a disponibilização de um novo serviço pago na plataforma de hospedagem. Ao contatar a área de operações, o gerente de produto percebeu que tal importância não estava evidente, e por isso a demanda estava parada aguardando ser publicada.

Sobre os erros que estavam aguardando homologação, a equipe percebeu que existia uma ineficiência no processo, pois o gerente de produto não validava esse tipo de demanda. Isto é, bastava o crivo do testador que já era feito na etapa de "Em desenvolvimento".

Para que você e sua equipe não corram o risco de sofrer dos problemas expostos, gostaria de sugerir três ações de simples implementação. A primeira diz respeito ao conceito de classe de serviço.

Segundo Vacanti (2015), uma classe de serviço é uma política ou um conjunto de políticas em relação à ordem pela qual os itens de trabalho são puxados em um processo. Em outras palavras, significa que você pode tratar os itens que passam pelo fluxo de desenvolvimento de forma diferente.

No exemplo do estudo caso, como a equipe percebeu que os bugs não precisavam passar pela análise do gerente de produto, uma vez que esse tipo de demanda entrasse no fluxo, a equipe poderia definir uma regra de que as etapas de "Aguardando homologação" e "Em homologação" seriam ignoradas.

Equipes que adotam esse tipo de estratégia devem estar seguras de que realmente as etapas intermediárias eliminadas não produzirão valor para a demanda. Já vi situações nas quais a equipe, interessada apenas em aumentar o *throughput*, ao diminuir o *lead time* e controlar o WIP, deixou de lado questões relacionadas a qualidade do que estava sendo entregue. Qual foi resultado? Problemas e mais problemas sendo criados e precisando ser gerenciados.

A segunda ação diz respeito a importância de deixar visível e comunicar o valor de uma demanda. Para isso, promova a colaboração entre as pessoas envolvidas no fluxo de desenvolvimento. No estudo de caso, percebemos um certo distanciamento da área de operações. Um dos principais propósitos de se trabalhar a colaboração dentro de uma equipe é levar as pessoas a compartilharem objetivos e aprenderem juntas, visando superar desafios e construir produtos que gerem mais valor ao negócio.

Por fim, identifique gargalos o mais cedo possível a partir da visualização do fluxo de desenvolvimento. O hábito de olhar o quadro diariamente, seja ele físico ou virtual, é uma ótima forma de sensibilizar a equipe sobre o que está acontecendo.

Deixe as filas que podem ser formadas ao longo do processo explícitas, definindo etapas de espera entre as etapas de trabalho. No exemplo, só ficou clara a formação de uma fila, pois, dos 5 itens que estavam em aberto no fluxo da equipe, 4 deles estavam em etapas de espera ("Aguardando homologação" e "Aguardando publicação") e sem trabalho algum sendo empregado neles.

4.6 Técnicas para analisar o throughput de forma avançada

Neste tópico, compartilharei algumas formas não tão usuais de avaliar o *throughput* para que você melhore sua análise junto à equipe.

Dificilmente você terá acesso aos gráficos ou às informações expostas a seguir, a partir de softwares de gestão de projetos ou processo disponíveis no mercado. Por isso, mais uma vez sugiro que este trabalho seja feito utilizando planilhas eletrônicas ou ferramentas estatísticas como o R (para baixá-lo, acesse https://www.r-project.org/).

Assim como a métrica de *lead time*, a primeira análise que realizo do *throughput* vem das variáveis estatísticas de mínimo, máximo, média, moda, mediana e percentil. O motivo? A partir delas, consigo ter uma leitura dos limites inferiores e superiores do time, em outras palavras, os picos (máximo de entrega em determinado período de tempo) e vales (mínimo).

Também passarei a analisar se a média está muito distante do ponto que divide as informações do *throughput* no centro (mediana). Assim, compreenderei qual tem sido o comportamento de vazão em 75% dos casos, em 25% dos casos ou em 95% dos casos.

Em situações em que a equipe não mede o *lead time*, ter o *throughput* em mãos permite inferir se as demandas trabalhadas estão levando muito ou pouco tempo para serem concluídas. Como você pode saber disso? Observe: equipes que possuem um alto *lead time* são aquelas que têm problemas ou baixo *throughput*, e o contrário também é verdadeiro.

Portanto, sugiro que você e a sua equipe tenham em mãos as duas métricas para que possam usufruir dos benefícios de gerar as melhorias no processo a partir de uma análise cruzada.

Para ilustrar o uso das estatísticas descritivas ao olhar para o *throughput*, nada melhor do que vermos a aplicação de uma análise em uma situação que pode ser parecida com o que você esteja vivenciando hoje.

Imagine que a equipe *XPTO* participa de uma reunião semanal com outras equipes e que a pauta é tratar quantas funcionalidades novas estão sendo produzidas. Se a equipe não estiver mensurando a sua capacidade de entrega através de métricas como velocidade (número de pontos entregues em um ciclo) ou *throughput*, responder a essa pergunta será algo improvável ou baseado no achismo.

Para que a equipe não se comprometa com prazos de entrega impraticáveis, consultar as variáveis que serão apresentadas a seguir se torna pré-requisito para um bom planejamento e para construir uma relação de confiança com quem está fora do dia a dia da equipe.

Como premissa para a análise dos dados, identifique o que está por trás do número expresso pelo *throughput*. Questione se realmente a métrica está levando em consideração demandas de características semelhantes. Se ela estiver "escondendo" demandas de naturezas distintas, a análise será questionável. Resumindo, garanta que você e a equipe estão analisando demandas comparáveis (*throughput* de *bugs* pode ser bem diferente do que o *throughput* de funcionalidades, por exemplo).

No caso do time *XPTO*, a amostra de dados só levava em consideração novas funcionalidades que eram construídas. Extraindo algumas estatísticas descritivas da equipe, temos que:

- **Throughput mínimo:** 1 funcionalidade por semana.
- **Throughput máximo:** 7 funcionalidade por semana.
- **Média:** 2 funcionalidade por semana.

- **Moda:** 1 funcionalidade por semana.
- **Percentil 25:** 1 funcionalidade por semana.
- **Mediana:** 2 histórias por semana.
- **Percentil 75:** 3 funcionalidades por semana.
- **Percentil 95:** 6 funcionalidades por semana.

A primeira observação que podemos fazer baseado no *throughput* mínimo e no percentil 25 é de que ao menos uma funcionalidade foi produzida por semana. Caso a equipe *XPTO* enfrentasse uma situação de precisar desenvolver algo crítico ou de alto valor para o negócio, com a informação anterior em mãos, a área demandante ficaria segura de que haveria grandes chances de a demanda ser entregue no período de uma semana.

Seguindo com a análise, percebemos que a moda (valor mais frequente do número de entregas) não é uma boa medida para dizer qual tem sido o comportamento de entrega semanal da equipe. Seu valor é menor do que a mediana (responsável por dividir o conjunto de dados ao meio).

Se usássemos uma funcionalidade entregue por semana como sendo o comportamento padrão, estaríamos indo contra 50% da base histórica da equipe *XPTO*. Isso geraria uma visão pessimista demais.

Sendo assim, você deve estar pensando que o caminho seria utilizar a média, correto? No exemplo do time *XPTO*, a resposta poderia ser sim. Porém, existem casos em que a média não será uma boa referência, dado que seu valor pode ser distorcido por valores extremos. Ao longo dos anos venho, recomendando a mediana como métrica de referência, pois ela é menos sensível do que a média aos limites inferiores e superiores da distribuição.

Voltando ao exemplo da equipe *XPTO*, temos uma situação na qual a média e a mediana possuem o mesmo valor. Tal comportamento acontece por conta da distribuição não conter ocorrências de *throughput* distantes da mediana.

O percentil 75 nos permite observar que, em 75% das semanas, o *throughput* da equipe foi de até 3 demandas. Como tal medida está próxima da mediana, não seria irresponsável usá-la como uma perspectiva otimista de vazão.

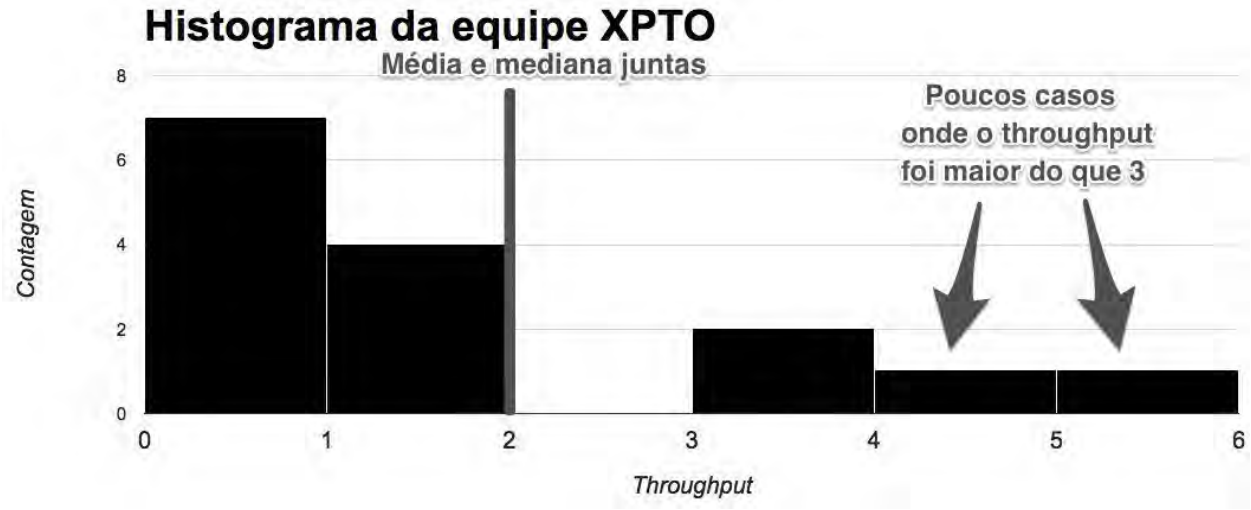


Figura 4.9: Histograma de throughput da equipe XPTO

Outra informação que pode ser retirada das estatísticas descritivas diz respeito ao que seria um *throughput* máximo de funcionalidades. Se por algum motivo, a equipe chegasse à conclusão de que consegue entregar mais de 6 funcionalidades por semana, poderíamos justificar, através do percentil 95, que em apenas 5% das semanas analisadas até então isso aconteceu. Dessa forma, a equipe não assumiria tal padrão de entrega por ser uma exceção.

Com todas as informações coletadas até aqui, a equipe *XPTO* poderia levar para a reunião semanal o seguinte diagnóstico: "Analisando nosso histórico recente de entregas, conseguimos produzir, com boa margem de segurança, até 2 funcionalidades novas por semana. Além disso, temos boas chances de entregarmos até 3 funcionalidades. Porém, dificilmente conseguiremos entregar mais do que 4 funcionalidades por semana".

As análises feitas neste tópico lhe dão insumo de ser transparente quanto a capacidade de entrega de uma equipe. Espero que você tenha compreendido que, a partir de estatísticas básicas, é possível identificar padrões (moda, média, mediana), distorções (média comparada com os valores mínimo e máximo) e probabilidades (percentil).

Agora, você e sua equipe estão aptos a olharem a cadência de entrega de uma forma estruturada, visando a busca por um processo mais previsível.

4.7 Recapitulando

Saber o número de entregas realizadas pela minha equipe é mais uma métrica para o portfólio de quem procura melhorar o processo de desenvolvimento de software.

Neste capítulo, discutimos como o *throughput* pode ser útil para analisar o volume de entregas produzidas ao longo de um espaço de tempo. Todas as análises e visualizações propostas podem ser feitas em uma visão global do sistema, ou a partir de uma perspectiva por tipo de demanda (exemplo: número bugs corrigidos por dia, número de funcionalidades entregues por semana etc.).

Utilizando uma variação da lei de Little, percebemos que existe uma relação entre as métricas de *lead time*, software e *throughput*. A partir das premissas apresentadas, aprendemos a importância de fazer com que as equipes coloquem todos os esforços necessários para trabalhar em demandas que serão úteis e para a finalização daquelas já iniciadas, removendo, quando necessário, possíveis bloqueios.

Por fim, antes de a equipe trazer um novo item de trabalho para o fluxo de desenvolvimento, é válido que ela analise a existência de algo em progresso que esteja com um *lead time* alto e dê foco para finalizá-lo. Afinal, funcionalidade não entregue nada mais é do que um amontoado de tecnologia que não teve sua validade posta a prova.

A partir do estudo de caso apresentado, exercitamos maneiras de interpretar o *throughput*, a fim de extrair informações que serão úteis para identificar falhas no processo de desenvolvimento de uma equipe de software. Fechamos o capítulo aprendendo como um conjunto de estatísticas descritivas pode aprimorar a interpretação por trás de um conjunto de dados de *throughput*.

A partir de agora, ao analisar o *throughput*, você passará a fazer questionamentos como:

- Qual tem sido a cadência de entrega da equipe?
- Temos mantido um padrão no número de entregas?
- Se sim, como podemos manter?
- Se não, que tipo de mudança precisamos fazer para que passemos a entregar mais e com qualidade?

Bem-vindo ao mundo das melhorias baseadas em dados. Com a apresentação da métrica de *throughput*, você e sua equipe estão preparados para identificar disfunções no processo, e levantar fatores que estejam minando o fluxo de desenvolvimento.

Além disso, terá fatos para compartilhar e argumentar a partir do contexto que estiverem inseridos. Adotando o que aprendemos até aqui, você terá um dia a dia mais transparente e previsível.

4.8 Referências

LITTLE, J. D. C. *A proof for the queuing formula:* L = λW. Case Institute of Technology, p. 383-387, Jun. 1961. Disponível em: http://pubsonline.informs.org/doi/10.1287/opre.9.3.383

COLUCCI, L. *Calculating Cost of Delay for software projects.* Nov. 2016. Disponível em: http://blog.plataformatec.com.br/2016/11/calculating-cost-of-delay-for-software-projects/

ZHEGLOV, A. *How to Match to Weibull Distribution in Excel.* Ago. 2013. Disponível em: https://connected-knowledge.com/2013/08/01/how-to-match-to-weibull-distribution-in-excel/.

SIGMAN, K. *Notes on Little's Law.* 2009. Disponível em: http://www.columbia.edu/~ks20/stochastic-I/stochastic-I-LL.pdf.

FERRARI, H. *BVP na prática.* Ago. 2016. Disponível em: http://blog.taller.net.br/bvp-na-pratica/

VACANTI, D. *Actionable Agile Metrics for Predictability.* Daniel S. Vacanti, Inc., 2015.

CAPÍTULO 5

Visualizar o fluxo de desenvolvimento de um time ágil

5.1 Introdução

Trabalhar com o desenvolvimento de software é um grande desafio. Afinal, temos de lidar diariamente com incertezas, mudanças, pessoas, expectativas, dependências, restrições etc.

Mapear um fluxo que garanta o desenvolvimento de software em um ritmo sustentável, ou até mesmo que proteja as equipes das incessantes demandas geradas por diferentes facetas envolvidas na construção de um produto ou projeto, é um dos grandes desafios de Gestores, Agile Coaches, Scrum Masters e time.

Com as etapas do fluxo de desenvolvimento estruturadas, as demandas (como funcionalidades, *bugs* etc.) passam a ser construídas seguindo a lógica pensada para o processo. E é aí que passam a surgir gargalos, alto tempo de entrega e baixa sensação de progresso.

Se você faz parte de uma equipe que possui um fluxo de desenvolvimento, em algum momento do seu dia a dia, você se deparará com a seguinte questão: como eu posso dar visibilidade de como está a saúde do meu processo?

Independente do seu método de trabalho, seja Kanban, Scrum, XP etc., desenvolver uma visão que apresente a quantidade de itens entregues, a quantidade de itens em progresso e o montante de trabalho a ser feito deve ser um tipo de prática difundida na rotina de qualquer equipe.

Dessa forma, você estará promovendo uma cultura que privilegiará: cadência de entrega; melhoria baseada em dados; visibilidade do fluxo de trabalho; e, principalmente, uma constante inspeção e adaptação do processo. Em resumo, você estará buscando entregar valor mais cedo aos seus usuários e clientes.

Neste capítulo, teremos a oportunidade de discutir a construção do CFD (*Cumulative Flow Diagram*), bem como o benefício de sua análise e sugestões de métricas que você poderá extrair dele.

Se você já conhece o CFD, não deixe de conferir o estudo de caso que será relatado mais adiante. Faça uma reflexão de como o aprendizado gerado pelo projeto citado poderá ajudá-lo a melhorar o seu dia a dia, afinal, sábio é aquele que aprende com os erros e acertos dos outros.

Ao final deste capítulo, espero que você entenda como o CFD pode se tornar uma importante ferramenta de monitoramento do processo de um time ágil.

5.2 O que é o CFD?

Primeiramente, é importante dizer que o CFD é um tipo de visualização que trata essencialmente de entradas e saídas. Como o próprio nome pode sugerir, o *Cumulative Flow Diagram* é uma excelente forma de compreender o fluxo de trabalho ao longo de um processo, afinal, o gráfico pode nos dar um panorama geral do que está acontecendo durante o desenvolvimento de um projeto ou produto.

O CFD é um instrumento de muito valor no processo de monitoramento em projetos ágeis, pois, usando-o, você poderá rapidamente analisar: quanto de trabalho foi realizado, quanto de trabalho está em progresso e quanto de trabalho ainda precisa ser feito.

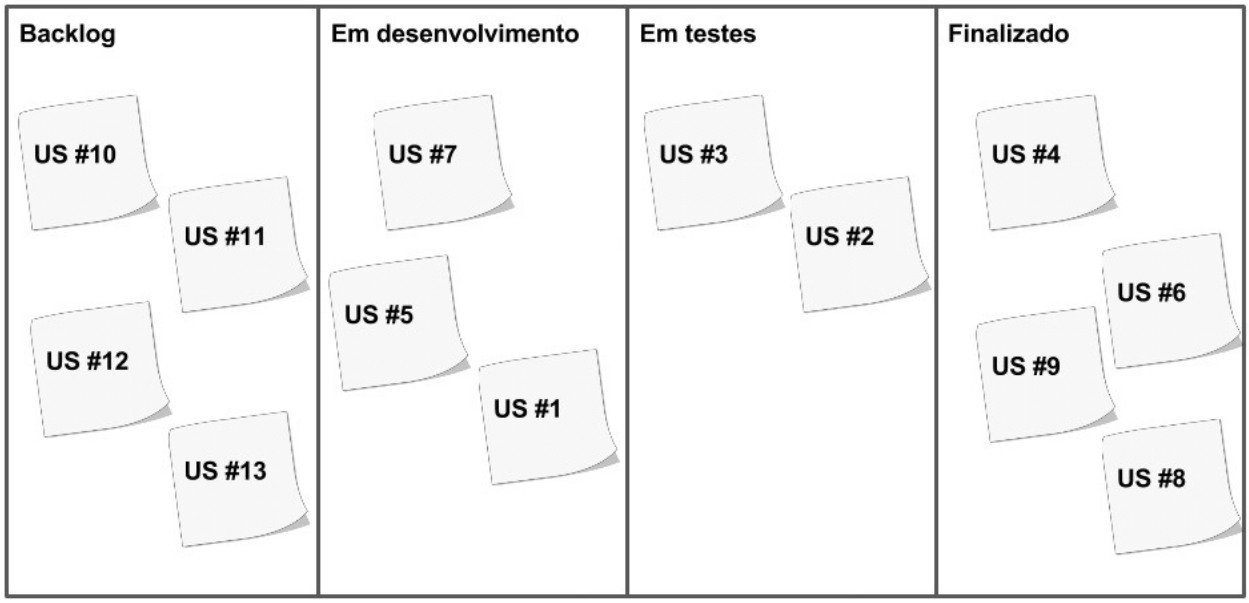
Visualizar o fluxo de um time a partir do CFD nos fornece *insights* quantitativos e qualitativos de potenciais ou reais problemas que possam ocorrer no contexto de trabalho do time. Afinal de contas, compreender a atual performance do processo é um dos primeiros passos para introduzir um sistema que tenha como fundação uma filosofia voltada à previsibilidade e melhoria contínua.

A estrutura do gráfico é muito simples. O eixo horizontal representa um período de tempo (semanas, *sprints* etc.) e o eixo vertical indica, de forma acumulada, o número de itens no processo (total de tarefas, total de histórias etc.). Cada área pintada no gráfico está relacionada a uma etapa do fluxo de trabalho (backlog, em progresso, finalizado etc.) e as curvas são basicamente o número de itens acumulados em tais etapas.

Vamos compreender como o que foi descrito anteriormente pode ser visto na prática. Imagine que o time XPTO possua o seguinte fluxo de trabalho:

- **Backlog:** primeira etapa do processo onde o time discute e detalha as funcionalidades priorizadas e que foram selecionadas para serem construídas.
- **Em desenvolvimento:** etapa na qual o time passa a construir as funcionalidades.
- **Em testes:** etapa na qual são realizadas validações e testes das funcionalidades construídas pelo time.
- **Finalizado:** última etapa do processo de desenvolvimento em que as funcionalidades são aprovadas e publicadas para os usuários finais.

A figura a seguir apresenta como as etapas descritas anteriormente estariam representadas em um quadro que expõe visualmente o trabalho do time.



Legenda:
US: User story = História do usuário.

Figura 5.1: Quadro mapeando o fluxo de trabalho do time XPTO

Mas, afinal, qual seria uma representação possível do CFD do time XPTO?

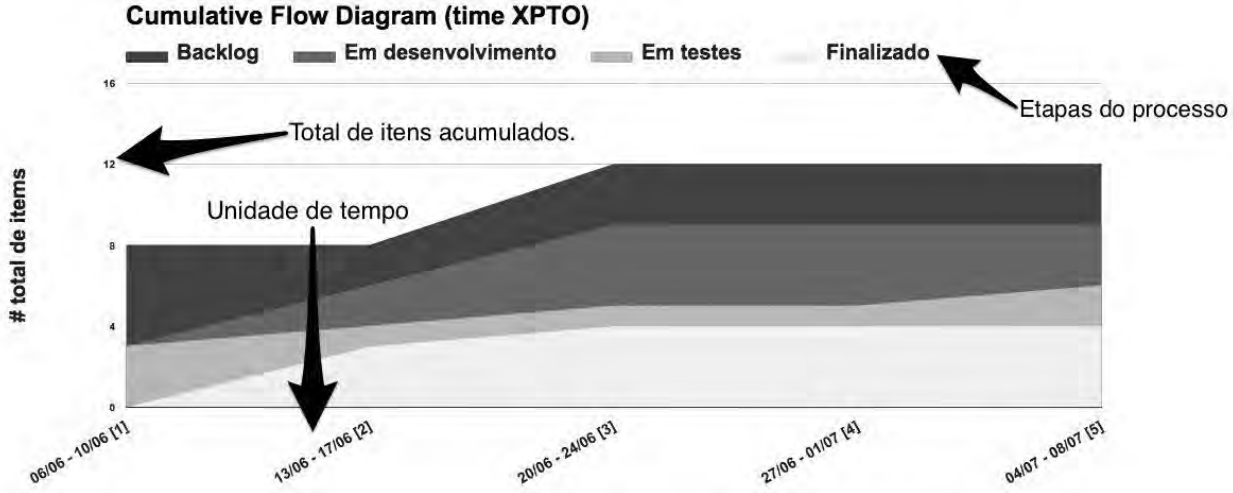


Figura 5.2: Exemplo de CFD a partir do processo do time XTPO

Conforme podemos observar no eixo horizontal da figura anterior, o time XPTO realiza um acompanhamento do seu processo de desenvolvimento semanalmente a partir de uma progressão do tempo da esquerda para a direita.

Um ponto importante de salientar é que a unidade de tempo e forma de ordenação definida para o CFD do seu time pode ser outra. Sendo assim, antes de fazer qualquer tipo de interpretação, compreenda em que unidade de tempo o gráfico está se comunicando e como ele está organizado.

As etapas do processo de desenvolvimento do time e o valor acumulado de itens que passaram por cada uma delas estão representadas no gráfico a partir das curvas "Backlog", "Em desenvolvimento", "Em teste" e "Finalizado". Ainda usando o exemplo de CFD, podemos dizer que, na quinta semana (04/07 até 08/07), o time XPTO possuía 12 itens que já haviam passado pela etapa de “Backlog”, 9 itens que passaram pela etapa de “Em desenvolvimento”, 6 que passaram pela etapa de “Em testes” e 4 que passaram pela etapa “Finalizado”.

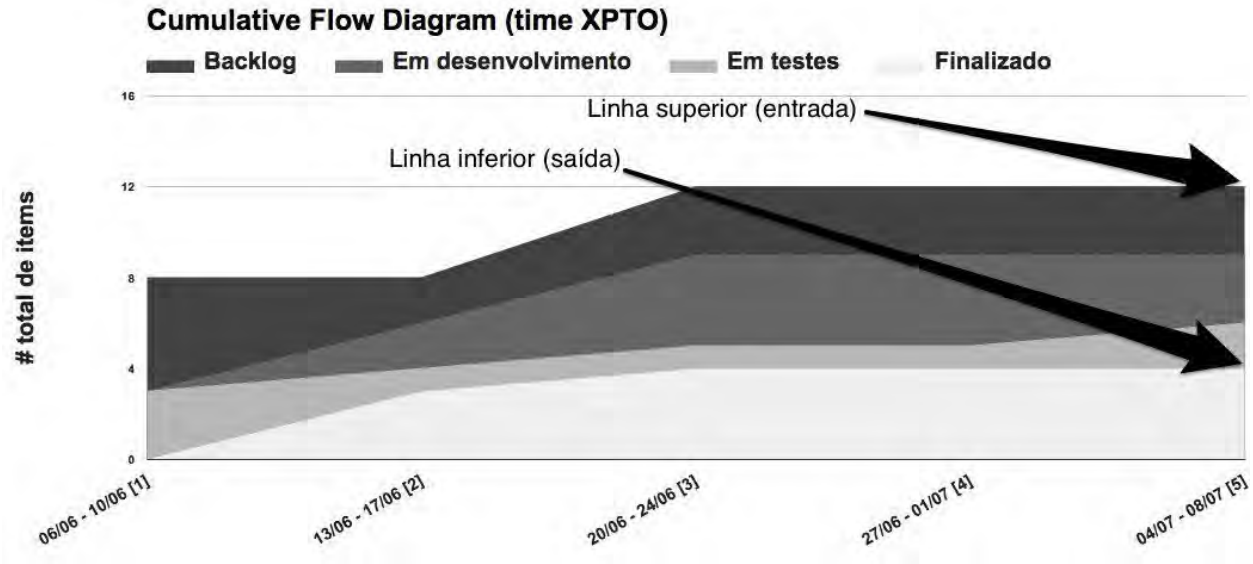


Figura 5.3: Linhas superior e inferior em um gráfico de CFD

Ainda sobre como ler o CFD, podemos analisá-lo a partir do conceito de entradas e saídas do processo (linhas superior e inferior, respectivamente).

Tendo como referência a quinta semana do time XPTO, ilustrada na figura anterior, podemos afirmar que 12 itens entraram para o processo de desenvolvimento (valor representado na linha superior, isto é, a primeira etapa do processo) e que 4 foram entregues (valor representado pela linha inferior, isto é, a última etapa do processo).

Para que a análise citada anteriormente faça sentido, é fundamental que o gráfico de CFD contenha apenas itens que passarão por todo o fluxo de trabalho. Isto significa dizer que, dado que o time assumiu o compromisso de desenvolver determinado item, este não poderá sair do processo, o que fará com que a linha superior nunca tenha o comportamento de queda.

A pergunta que você pode estar se fazendo neste momento é: "Afinal, o backlog de um time não pode diminuir?". A resposta é sim, porém, para que o CFD respeite a característica de ser uma análise **acumulativa**, o sistema de trabalho deve incluir apenas itens que já foram priorizados, analisados, refinados e estão prontos para entrar no fluxo.

Por esse motivo, torna-se relevante compreender o que está por trás da definição de "Backlog". No capítulo no qual trataremos sobre a visualização do gráfico de Burnup, explicarei por que ela sim deve representar a variação do escopo, dado que, no contexto do CFD, estamos discutindo e analisando o processo.

A melhor maneira de você capturar os dados para um CFD é rastrear a data em que os itens que passam pelo processo do time entram em cada etapa. Coletar tais dados será também insumo para que você possa analisar outras métricas, como o *lead time* e o *throughput*.

Mapear as entradas e saídas de um processo a partir de uma perspectiva acumulativa ao longo do tempo é uma das melhores ferramentas disponíveis para que você visualize a saúde do processo do seu time.

5.3 Como analisar o CFD a partir de um caso real

Nesta seção, teremos a oportunidade de analisar, a partir de um caso real, como o CFD foi útil para que um time pudesse avaliar a saúde do processo ao longo do desenvolvimento de um produto.

O CFD foi usado como forma de comunicar o progresso do trabalho para os principais interessados no resultado do projeto, bem como para identificar gargalos no processo e, principalmente, para gerenciar estoques nas etapas do fluxo de desenvolvimento.

Antes do início do projeto, foram realizados *workshops* educativos com o objetivo de ensinar o time a interpretar o fluxo de trabalho a partir do CFD.

Caso: desenvolvimento da nova versão de uma plataforma de B2B

O caso que será descrito se passou em uma equipe que trabalhou na recriação de uma aplicação B2B (*business to business*), responsável por conectar fornecedores de materiais esportivos e lojas de varejo.

A aplicação possuía 10 anos de existência e contava, em seu leque de usuários, com mais de 5.000 lojas compradoras e 800 fornecedores. Por mês, a plataforma transacionava mais de 10 milhões de reais e a empresa detentora da tecnologia era a principal líder deste tipo de solução no Brasil.

Em decorrência da chegada de novos concorrentes, alguns usuários começaram a migrar para as novas soluções, pois consideravam que a plataforma atual da empresa tinha baixo poder de inovação. As atualizações eram demoradas e as funcionalidades lançadas tinham pouco valor para quem utilizava a solução no dia a dia.

Um dos maiores desafios técnicos do projeto foi garantir a compatibilidade entre a nova e a velha tecnologia. Afinal de contas, o patrocinador colocou como restrição que a plataforma antiga deveria coexistir com a nova, já que ele não gostaria de gerar uma mudança drástica no dia a dia dos usuários.

O time teria 5 meses para realizar o primeiro grande lançamento da nova versão do produto, pois a área comercial da empresa gostaria de aproveitar uma feira do meio esportivo para promover a solução junto aos seus parceiros e usuários. A equipe que trabalharia no desenvolvimento era composta por: 4 desenvolvedores; 1 Agile Coach; 1 testador; e 1 gestora de produto.

O time decidiu trabalhar com o método Kanban e incorporou para o seu processo de desenvolvimento: cerimônias de planejamento, nas quais eram priorizadas e discutidas as funcionalidades que entrariam no fluxo de desenvolvimento; retrospectivas, onde eram discutidas melhorias de processo, principalmente, a partir das métricas coletadas; e refinamento, em que eram avaliadas as incertezas, complexidades, dependências e critérios de aceite das funcionalidades que seriam trazidas para o fluxo de desenvolvimento.

O fluxo de trabalho do time estava dividido nas seguintes etapas:

- **Em desenvolvimento**: etapa na qual a demanda que estava refinada e priorizada entrava para o fluxo de desenvolvimento do time. Neste momento, o time assumia, junto a gestora de produto, a responsabilidade de entregar a funcionalidade que acabara de entrar no processo.

- **Revisão de código**: etapa em que os desenvolvedores revisavam o código que havia sido produzido, a fim de garantir qualidade naquilo que estava sendo construído a partir da perspectiva arquitetural e técnica.

- **Aguardando testes**: etapa onde os desenvolvedores liberavam as funcionalidades criadas para a validação do testador que estava no time.

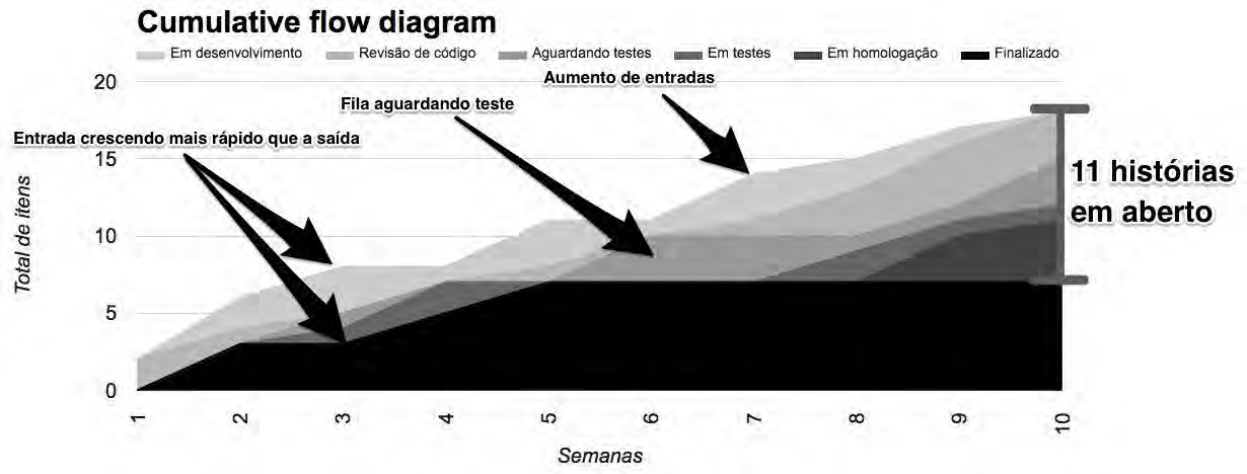
- **Em testes**: etapa na qual eram testadas as funcionalidades. Eram realizados testes exploratórios, de performance e de integração.

- **Em homologação**: etapa em que a gestora de produto validava se as funcionalidades desenvolvidas atingiam o objetivo de negócio que havia sido definido.

- **Finalizado**: etapa onde as funcionalidades eram publicadas no ambiente que seria futuramente o de produção.

O início: poucas entregas

As 10 primeiras semanas do projeto foram de muito aprendizado para todos os membros



do time.

Figura 5.4: Estudo de caso: dez primeiras semanas do projeto

Nas três primeiras semanas de trabalho, a taxa de entrada de demandas no fluxo era maior do que a taxa de saída. Na média, o time entregava uma história por semana e 3 novas histórias entravam para o fluxo de desenvolvimento.

Neste momento, os desenvolvedores, o testador e a gestora de produto entraram em um consenso de que nenhum novo item entraria para o fluxo de desenvolvimento na semana seguinte (quarta semana). Com tal decisão, o time foi capaz de reduzir o número de histórias em aberto. Ainda sobre o processo, nenhum outro tipo de gargalo era perceptível naquele momento e o time parecia começar a entender qual era o seu limite ideal de entradas e saídas.

Na quinta semana do projeto, o time recebeu a notícia de que o testador entraria de férias por duas semanas e só retornaria aos trabalhos na sétima semana. Mas, afinal, o que o CFD nos diz sobre o que ocorreu com o projeto?

- Ninguém do time de desenvolvimento se prontificou a testar as funcionalidades que estavam sendo criadas, portanto, um estoque de 3 funcionalidades se formou e perdurou até a volta do testador aos trabalhos.

- Na sétima semana, em vez de auxiliar o testador na validação das funcionalidades que haviam sido desenvolvidas durante a sua ausência, os desenvolvedores decidiram trazer novas funcionalidades para o fluxo.

- Da oitava a décima semana, mesmo com os desenvolvedores e o testador fazendo uma força tarefa para testar as funcionalidades, por conta de compromissos organizacionais, a gestora do produto ficou indisponível, o que gerou estoque na etapa de homologação.

- Na décima semana, o time se viu com um total de 11 histórias em aberto e com apenas 7 funcionalidades em produção. Tal cenário gerou uma certa desconfiança por parte do patrocinador do projeto, pois o time atingia a metade do prazo e os números mostravam uma projeção não muito animadora quanto ao que seria entregue até o evento.

O que podemos aprender deste início do projeto?

1. Identifique gargalos o mais cedo possível. Visualize os estoques que podem ser formados ao longo do processo, isto é, crie etapas intermediárias entre as etapas de trabalho. No caso, só ficou explícita a existência de um gargalo na fase de testes, pois o estoque da etapa "Aguardando testes" aumentou ao longo das semanas.

2. Estimule o time a olhar a vazão do processo. No caso, sabendo da ausência do testador, o time (gerente de produto, desenvolvedores e Agile Coach) deveria ter escalado uma pessoa que ficaria responsável pelos testes até o retorno do especialista. Dessa forma, o processo estaria mais estabilizado, isto é, sem gargalos e com uma relação mais equilibrada entre as entradas e saídas.

3. Monitore e controle a taxa de entrada e de saída do processo. Caso você perceba que a taxa de saída do processo é menor do que a taxa de entrada, cuidado, o time estará trabalhando acima da sua capacidade de entrega, o que gerará acúmulos de itens em aberto. Ambientes que lidam com esse tipo de cenário geralmente têm problemas de qualidade, possuem alta rotatividade por conta de estresse e não conseguem garantir uma cadência de entrega (o que gera baixa previsibilidade).

4. Quanto maior o número de demandas em aberto, mais difícil será estabilizar o processo. Crie uma cultura de entregas constantes e incrementais. Os 11 itens que estavam em aberto na décima semana compunham um ativo de funcionalidades que não representava valor para o patrocinador do projeto.

Momento chave: estabilizando o processo

No final da décima semana do projeto, o time realizou uma grande retrospectiva com o objetivo de aumentar as entregas, diminuir os gargalos do processo e aumentar a qualidade do produto que estava sendo desenvolvido. O CFD ilustrado a seguir demonstra o caminho que o time percorreu entre as semanas 10 e 20.

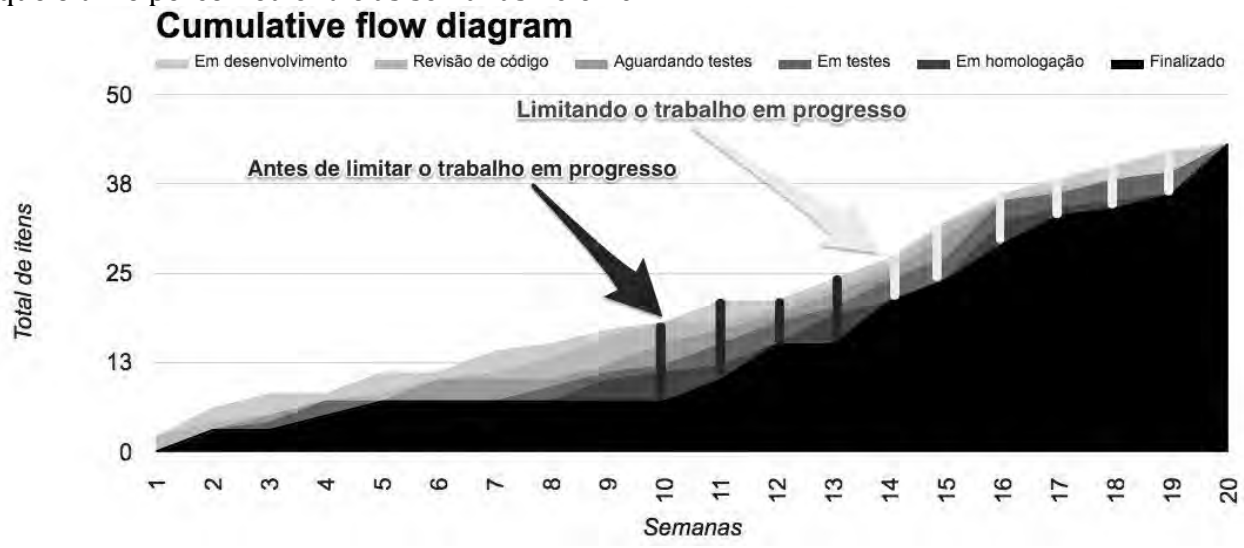


Figura 5.5: Estudo de caso: análise das dez últimas semanas do projeto

Entre as semanas 11 e 12, o time concentrou todos os esforços disponíveis para conseguir colocar em produção o maior número de funcionalidades possível. Para isso, desenvolvedores, testador e gestora de produto focaram em reduzir os gargalos localizados nas etapas de testes e aprovação (representadas por aguardando testes, em testes e em homologação).

A partir da semana 14, o time decidiu definir um limite para o total de itens que estariam em aberto no processo de desenvolvimento. O principal objetivo de tal decisão foi buscar garantir uma cadência na entrega e, principalmente, evitar gargalos nas etapas intermediárias do fluxo de desenvolvimento.

Sabendo que o prazo de entrega se aproximava, após a semana 15, a gestora de produto precisou tomar a decisão de controlar as funcionalidades que entravam no fluxo de desenvolvimento do time. Naquele momento, todos sabiam que os esforços deveriam estar alocados para a finalização da versão que seria apresentada no evento programado para acontecer na semana 20.

Três semanas antes da realização do evento, os desenvolvedores decidiram remover do processo a etapa de revisão de código, pois eles passaram a se organizar em duplas que

mudavam ao longo do dia. Dessa forma, todos participavam ativamente da construção e revisão do que estava sendo produzido.

Por fim, uma força tarefa foi realizada na semana 19 para que as 6 funcionalidades que estavam em aberto pudessem ser finalizadas.

O que podemos aprender com a segunda fase do projeto?

1. Controle o número de demandas que se encontram em aberto no processo. Monitore e limite a quantidade de itens que passam pelo fluxo. Essa é uma dica valiosa para aqueles que, de alguma forma, desejam trazer previsibilidade para o processo de desenvolvimento.

2. Fique atento caso alguma faixa desapareça (etapa do processo sem nenhum tipo de trabalho sendo feito). Caso alguma etapa do processo deixe de fazer sentido, talvez seja o momento de removê-la. No entanto, caso o time passe a pular etapas importantes, como por exemplo aquelas relacionadas à qualidade, cuidado, pois provavelmente as entregas produzidas podem estar repletas de defeitos e gerarão retrabalho no futuro.

3. Utilize o CFD como forma de discutir melhorias no processo junto do time. Dados e visualizações são ótima maneiras de remover o subjetivismo que pode existir nas discussões sobre como melhorar o processo a partir do que passou.

5.4 Métricas que podem ser extraídas de um CFD

Agora que tivemos uma visão geral sobre a estrutura do CFD, seus atributos e como utilizá-lo a partir da sua visualização, vamos dar nomes e definir algumas métricas que podem ser extraídas a partir das informações do fluxo que foram coletadas, e que às vezes não ficam tão visíveis graficamente.

Work in progress (WIP)

Dado que a linha superior de um CFD representa o total de itens que entraram no processo de desenvolvimento do time, e a linha inferior representa o acumulado de itens que saíram do sistema, podemos concluir que a diferença vertical entre as linhas, em qualquer intervalo de tempo, representará o total de trabalho em progresso (WIP) do time.

Ao longo dos anos, já me deparei com análises que consideravam o tamanho do WIP a partir da diferença dos valores acumulados de etapas intermediárias do processo. Tendo como exemplo o processo do time XPTO, é como se disséssemos que o WIP fosse a diferença entre o acumulado de itens das etapas “Em desenvolvimento” e “Finalizado”.

Caso queiramos analisar o sistema de trabalho como um todo, tal leitura estaria incorreta, pois estaríamos ignorando a primeira etapa do processo (Backlog) que demandou esforço do time para que as demandas pudessem ser definidas e priorizadas.

Voltemos ao exemplo de CFD do time XPTO para entendermos como tais diferenças geram ruído na análise. Imagine que, na terceira semana de trabalho do time, você precisasse compartilhar o total de itens em WIP.

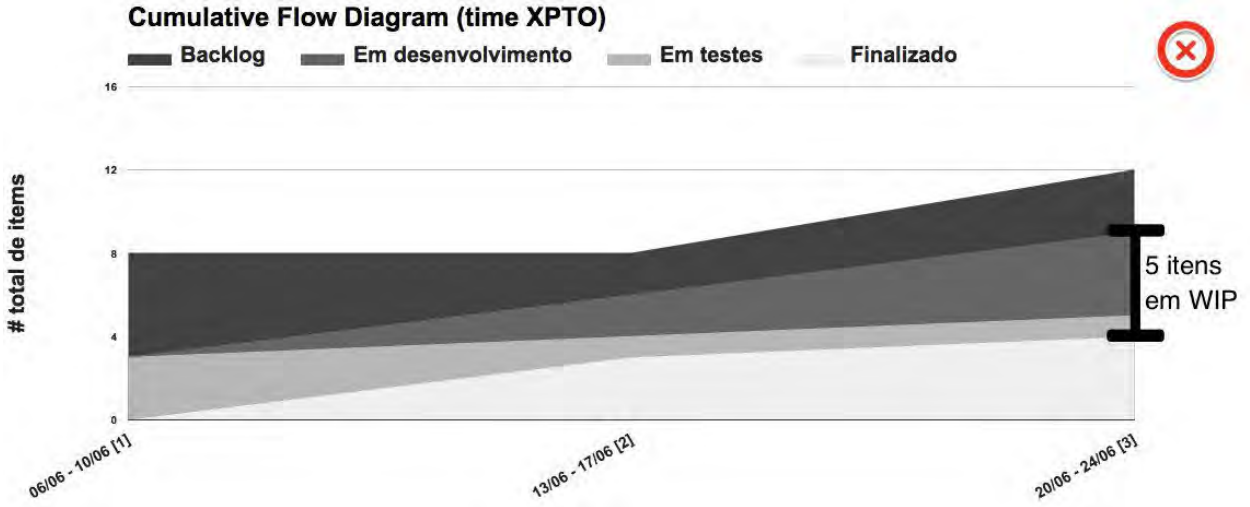


Figura 5.6: Considerando o WIP a partir de etapas intermediárias

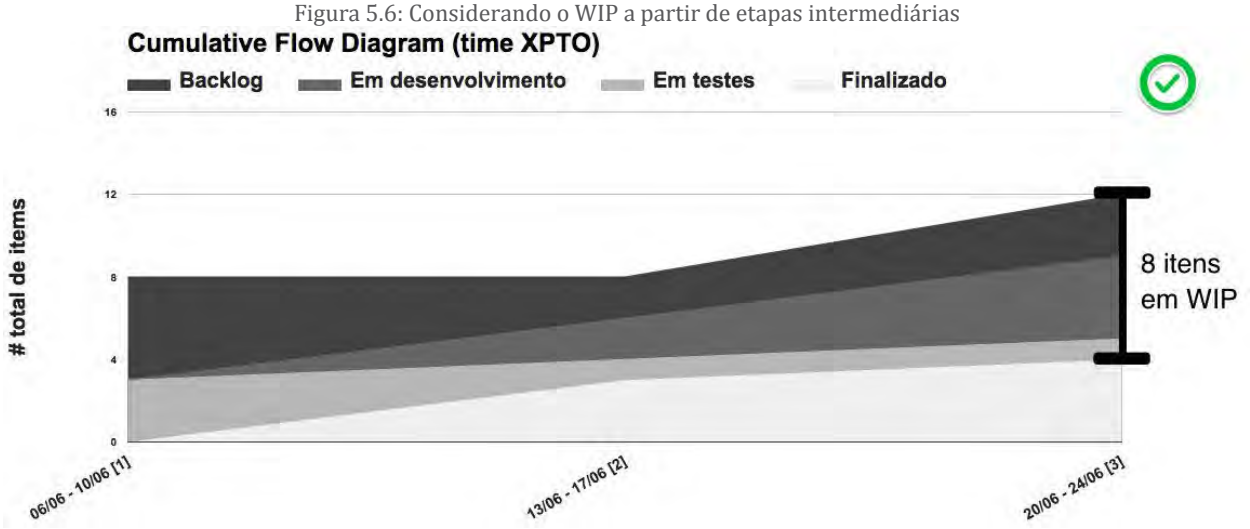


Figura 5.7: Análise do WIP de todo o processo do time XPTO

Analisando as figuras anteriores, observamos uma diferença quando extraímos a métrica de WIP a partir de etapas intermediárias, e quando calculamos a métrica a partir de uma

perspectiva de todo sistema. Na primeira figura, observa-se que o total de WIP no sistema de trabalho do time XPTO é 5 e não 8, afinal, o WIP está sendo medido a partir de etapas intermediárias.

Se você está usando tal abordagem, cuidado, pois é como se estivesse ignorando, no exemplo, o trabalho que foi despendido de análise e refinamento dos 3 itens que representam a diferença entre as etapas de “Backlog” e “Em desenvolvimento”.

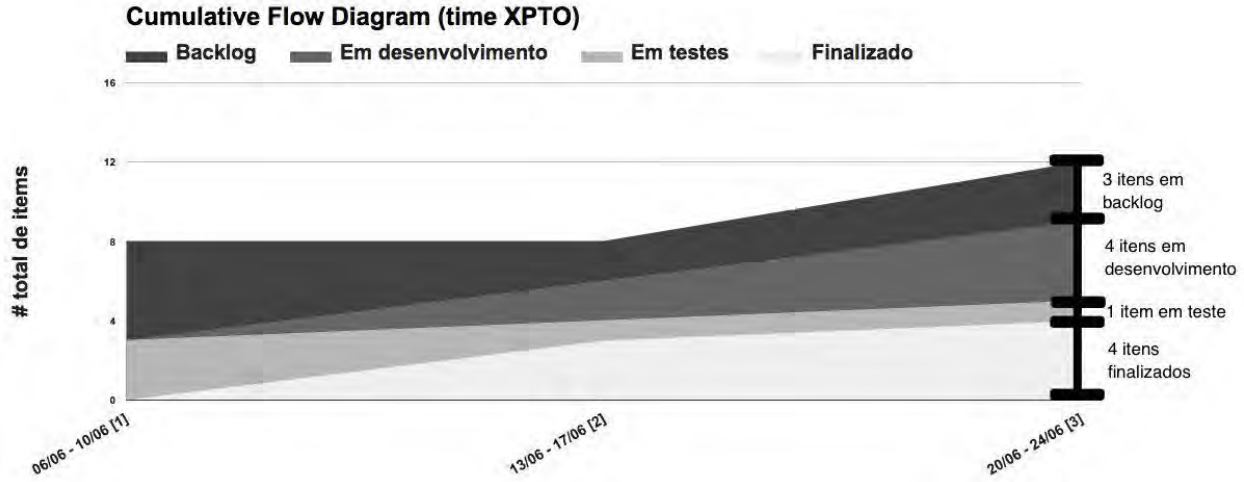


Figura 5.8: Análise do WIP por etapa do processo de desenvolvimento

Ainda sobre WIP, é possível extrair do CFD uma visão de trabalho em progresso em cada uma das etapas do processo do time. A figura anterior ilustra que, na terceira semana, o time XPTO tinha 3 itens sendo tratados na etapa de Backlog, 4 sendo trabalhados em desenvolvimento, 1 sendo testado, e 4 que haviam sido finalizados e publicados em produção.

Analisar o WIP a partir do CFD torna-se algo importante para qualquer pessoa que trabalha com desenvolvimento de software. Desenvolver tal visualização gera a possibilidade do time compreender a quantidade de itens que estão sendo trabalhados neste momento (como número de histórias em andamento na semana ou no *sprint*), *lead time*.

Além disso, também traz uma análise temporal de possíveis gargalos que possam ser formados no fluxo de trabalho como um todo (estoque de WIP só tem aumentado nas últimas semanas ou *sprints*, por exemplo), ou em determinadas etapas do processo — como o número de histórias em teste só tem aumentado, o que tem refletido na não entrega de novas funcionalidades aos usuários.

Em resumo, ter em mãos tal ferramental auxiliará na compreensão, definição e comunicação dos limites do sistema de trabalho, além da visualização de estoques.

Lead time médio

Além da possibilidade de analisarmos o tamanho do estoque de itens que estão em progresso (WIP) no fluxo de desenvolvimento, o CFD permite calcular, de forma aproximada, o tempo médio necessário para que um item saia do processo (*lead time*).

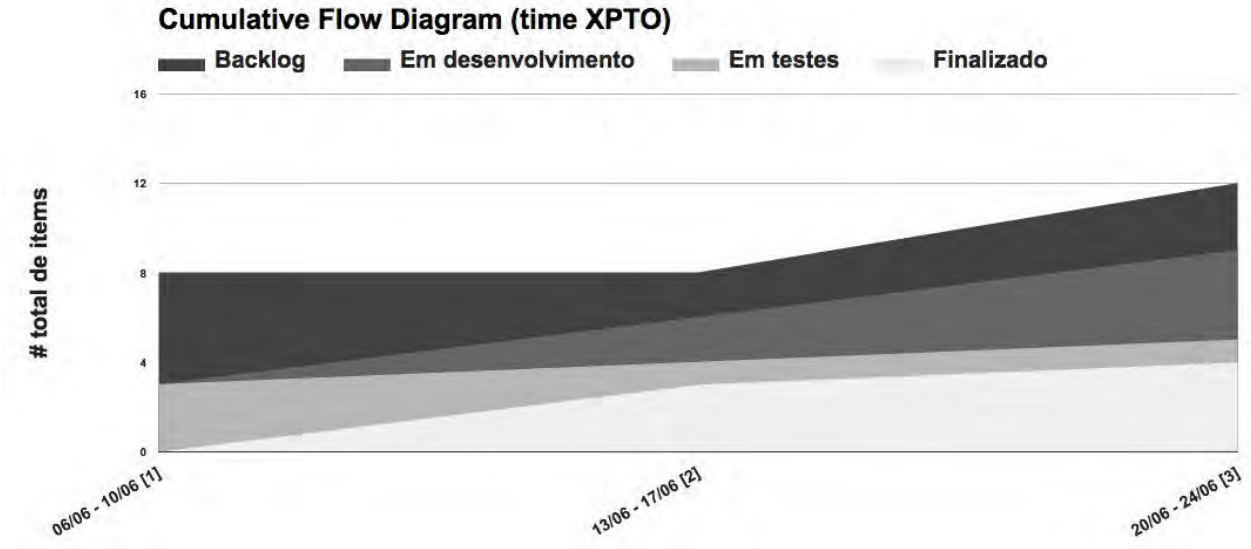


Figura 5.9: CFD do time XPTO na semana 3 do projeto

Para compreender melhor como extrair o *lead time* médio, voltemos ao CFD do time XPTO.

Conforme ilustrado na figura anterior, imagine que a partir da terceira semana do projeto o time já está mais confiante de que o fluxo de trabalho tenha atingindo o nível de maturidade desejado por todos.

Sendo assim, para dar visibilidade aos envolvidos (clientes, gestores de produtos etc.) de quantos dias estão sendo necessários para que sejam feitas as entregas, você acaba de receber a tarefa de extrair e mostrar ao time qual é o seu *lead time* médio. Para isso, você precisará realizar os seguintes passos a partir da visualização do CFD — note que o exemplo utilizará dias como unidade de tempo do *lead time*, no entanto, os passos serão os mesmos para qualquer outra unidade, como semanas, meses etc.

1. Defina um ponto na linha inferior do gráfico que corresponderá ao período de tempo que será analisado, e desenhe uma linha vertical. No caso que estamos analisando, tal período de tempo representa a semana 3, que corresponde aos dias 20/06 a 24/06.

2. A partir do ponto definido, desenhe uma linha horizontal da direita para a esquerda até que a linha inferior (etapa de saída do processo) toque a linha superior (etapa de entrada no processo) do CFD. No caso do time XPTO, temos o encontro entre as linhas na semana 1.

3. Dada a intersecção entre as linhas, o *lead time* médio será a diferença entre as datas das linhas superior e inferior. Como o CFD analisado utiliza no eixo horizontal o intervalo de tempo em semanas e desejamos extrair o *lead time* em dias, usaremos os últimos dias das semanas 1 e 3 para a realização do cálculo. Subtraindo 24/06 por 10/06, temos 14 dias.

4. Como último passo, adicione 1 ao resultado da subtração. No caso do time XPTO, temos que o *lead time* médio será de 15 dias. Mas, por que adicionar 01 ao resultado? Eu sempre aconselho a adição de uma "unidade de tempo" (no exemplo que estamos tratando, seriam dias), dado que o tempo mais curto que um item pode demorar para ser concluído é uma unidade.

   Imagine que um determinado item de trabalho comece e termine no mesmo dia (por exemplo, 24/06), qual será o seu *lead time*? Se fôssemos apenas subtrair 24/06 de 24/06, teríamos um *lead time* de 0 dias. Particularmente não concordo com esse tipo de visão, afinal, se considerássemos 0, estaríamos passando a ideia de que nenhum esforço foi necessário para a entrega deste item.

O passo a passo descrito anteriormente pode ser resumido na seguinte fórmula:

$$leadtimemedio = (datalinhasuperior - datalinhainferior) + 1$$

Figura 5.10: Fórmula para o cálculo do lead time médio

Voltando à análise do nosso exemplo, a partir de agora, o time poderá inferir que os novos itens que entrarão no fluxo de desenvolvimento levarão, aproximadamente, 15 dias para serem entregues. Tal dado pode ser útil em situações nas quais o time precise estimar datas de entregas para novas demandas.

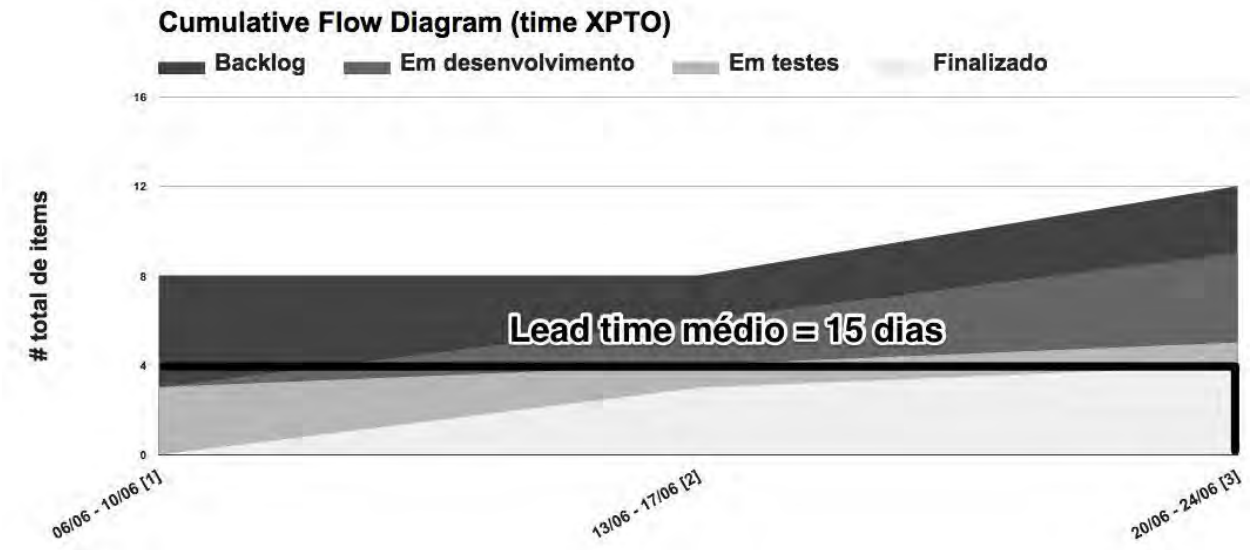


Figura 5.11: Exemplo de cálculo de lead time médio

É importante salientar que a métrica de *lead time* médio não deve ser considerada como uma meta, um comportamento padrão ou uma medida absoluta. Afinal de contas, pela natureza incerta que é desenvolver software, esse tipo de informação deve ser usado apenas como indicativo de tendência.

Throughput médio

Outra métrica que pode ser extraída da visualização do CFD é o *throughput* médio do processo, isto é, a taxa de vazão média de itens do fluxo de trabalho de determinado time.

Em algum momento de um projeto de desenvolvimento de software, o time precisará expor aos *stakeholders* qual tem sido, em média, a quantidade de funcionalidades entregues por um determinado período de tempo para que possam ser feitas projeções de quando o projeto será finalizado.

Para ilustrar como calcular o *throughput* médio, voltemos ao exemplo do time XPTO. Imagine que o time acaba de finalizar a quarta semana de trabalho no projeto e o gerente de produtos deseja saber quantos itens estão sendo entregues por semana, para se ter uma ideia da vazão do time. Como poderíamos calcular o *throughput* médio?

1. Determine um período de tempo. No exemplo, determinamos que vamos calcular o *throughput* médio das quatro primeiras semanas do projeto.

2. Descubra a "ascensão" da linha inferior, isto é, quantos itens saíram do sistema de trabalho naquele período de tempo. No caso do time XPTO, temos que 4 histórias foram entregues até a semana 4.

3. Por fim, divida a quantidade de itens que saíram do processo pelo período analisado. No caso do time XPTO temos que, em média, é entregue uma história por semana (resultado da divisão de 4 histórias por 4 semanas).

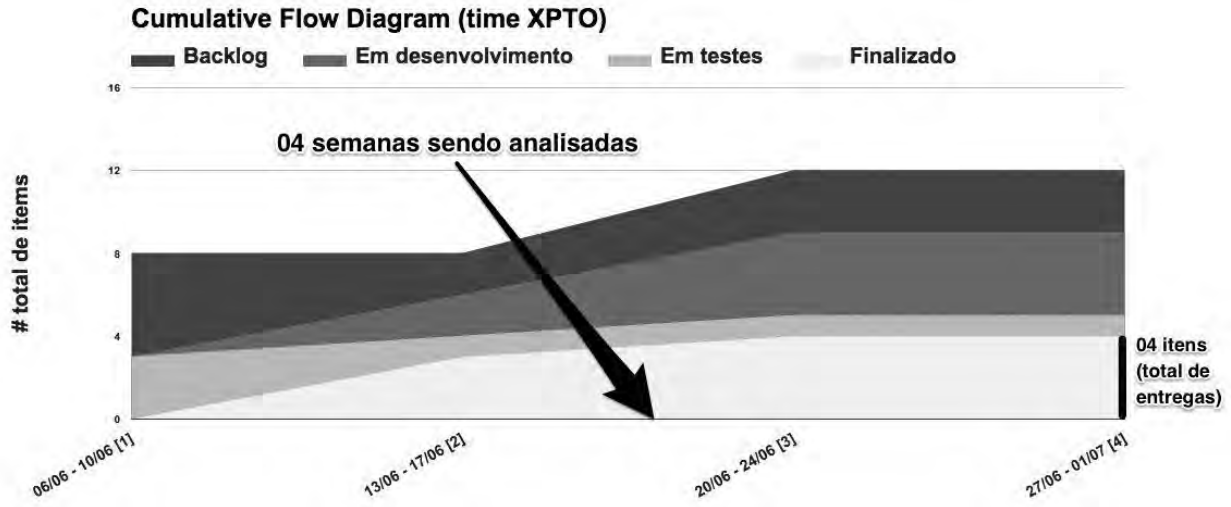


Figura 5.12: Exemplo de cálculo do throughput médio

O *throughput* médio nos dá a possibilidade de saber se o time está conseguindo garantir uma cadência de entregas ao longo do tempo.

É válido ressaltar que, como qualquer métrica, tal informação será útil e fará sentido dentro um contexto e para uma determinada unidade de análise (no caso, o time). Portanto, muito cuidado ao comparar o *throughput* médio de diferentes times.

Uma dica que costumo compartilhar sobre a análise do *throughput* é a de calcular a média a partir de um período de tempo mais recente.

Mudanças que aconteceram no fluxo de trabalho do time podem ter melhorado a taxa de vazão. E caso você esteja considerando momentos nos quais a vazão foi baixa (como produção por problemas de ambiente ou indefinições), a métrica média será influenciada por tal comportamento que, no entanto, dado o novo contexto, já não tem mais sentido.

No fundo, a ideia é remover da medida qualquer tipo de ruído que aconteceu no passado e que pode estar prejudicando uma visão mais realista de qual tem sido a cadência de entrega do time.

Análogo ao *throughput*, é possível extrair do CFD a taxa média de entrada de itens no processo de desenvolvimento. Os passos necessários para o cálculo desta métrica são os mesmos do *throughput*, exceto pelo fato de que queremos avaliar o número de entradas e não mais saídas no fluxo de trabalho.

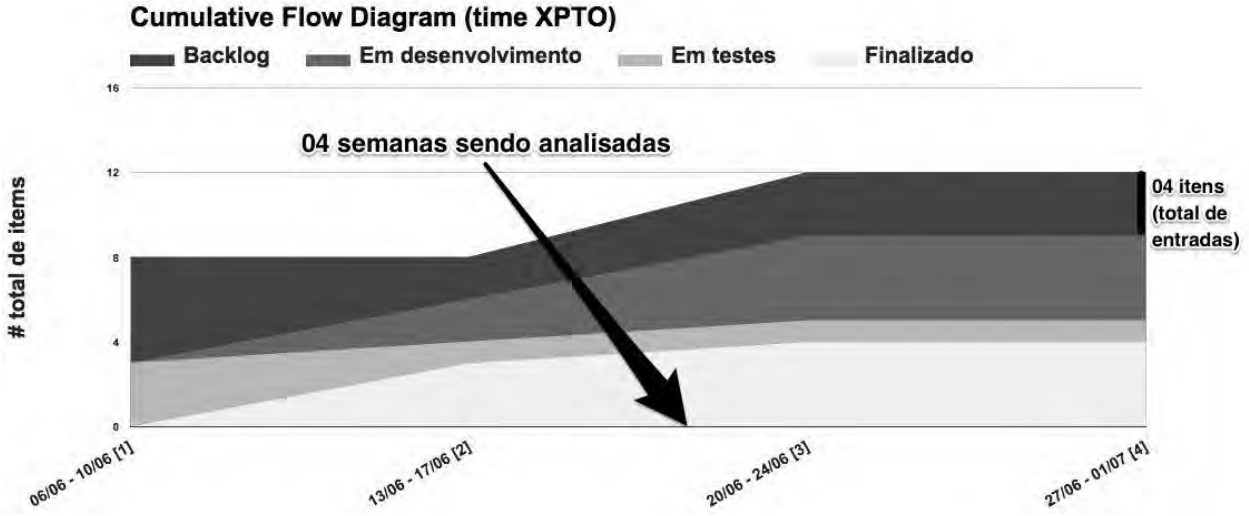
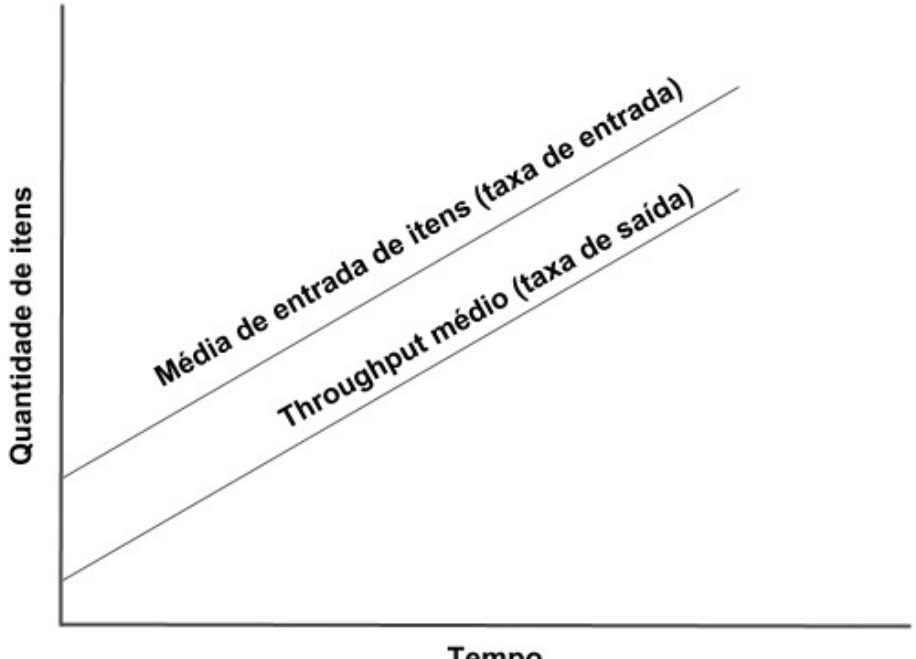


Figura 5.13: Exemplo de cálculo do taxa de entrada média

Utilizando como exemplo o time XPTO, conforme ilustrado na figura anterior, é possível dizer que a partir da primeira semana, 1 novo item entrou para o fluxo de desenvolvimento por semana, dado que a diferença entre o total de itens que passaram pela etapa de entrada do fluxo (Backlog) foi 4 (8 na primeira semana e 12 na última semana) e o total de semanas analisadas foi 4.

Analisar a taxa de entrada e o *throughput* médio juntos pode ser útil para compreender se um time tem conseguido manter uma constância de entrega (saída) a partir do que tem sido definido como necessidade (entrada).

Figura 5.14: Relação entre



throughput médio e média de entrada de itens

Caso o time tenha um comportamento parecido com o exibido nessa figura, ele terá dado o primeiro e, provavelmente, o mais importante passo para equilibrar o que entra e o que sai do processo de desenvolvimento. Buscar esse tipo de comportamento é algo fundamental para que haja um bom alinhamento de expectativas entre quem solicita uma funcionalidade de software e quem a constrói.

A pergunta que você pode estar se fazendo neste momento é: como eu consigo estimular o time a ter o tipo de comportamento ilustrado no último gráfico?

A melhor maneira de equilibrar entradas e saídas no desenvolvimento de projetos de software é garantindo uma quantidade constante de trabalho em progresso (WIP). Entenda como WIP constante um limite global de todo o sistema — o time pode ter, no máximo, 3 histórias em progresso, por exemplo —, ou uma determinação de quantos itens poderão ser trabalhados em cada etapa do fluxo — o time pode ter, no máximo, 2 histórias na etapa de teste).

Limitar o sistema tem se mostrado uma forma de garantir que os times passem a trabalhar em um ambiente com baixa variabilidade. Isso auxilia na previsibilidade das projeções de entrega.

5.5 Recapitulando

A geração da visualização do CFD pode ser feita manualmente (dará um pouco mais de trabalho, porém sua atualização pode ser realizada em cerimônias como o *daily*), por meio de planilhas eletrônicas ou até mesmo a partir de *softwares* especializados em gestão de projetos. Independente da forma como o time gerará o gráfico, o importante é dar visibilidade da evolução do fluxo para todos que estiverem envolvidos com o processo de desenvolvimento.

Dificilmente, você verá CFDs que sigam todas as dicas que compartilhei neste capítulo. Muito provavelmente você se deparará com visualizações bem diferentes das citadas nos exemplos, no entanto, não tenha medo.

Agora que você conhece os elementos do gráfico e as respectivas métricas (WIP, *lead time* médio, *throughput* médio e taxa de entrada), você saberá extrair ótimas ideias e, principalmente, saberá o que analisar.

Lembre-se: o gráfico só indicará problemas no processo. Todo o trabalho de análise e melhoria caberá a você e ao seu o time.

Uma forma de extrair informações úteis do CFD é através de perguntas, como por exemplo:

- O que está acontecendo com o nosso fluxo?
- Nossas métricas estão mostrando um bom ou mau caminho?
- Se estamos caminhando bem, como podemos manter?
- Se estamos com problemas, que tipo de mudança precisamos fazer para que o fluxo melhore?

O CFD garantirá responder as perguntas certas de forma antecipada e sugerirá as ações necessárias para aumentar a previsibilidade do processo.

Caso tenha interesse de saber mais sobre o assunto, não deixe de ler o excelente artigo escrito por Pawel Brodzinski (2013), ou o livro *Actionable Agile Metrics for Predictability* do Daniel Vacanti (2015).

5.6 Referências

BRODZINSKI, P. *Cumulative Flow Diagram.* Jul. 2013. Disponível em: http://brodzinski.com/2013/07/cumulative-flow-diagram.html.

VACANTI, D. *Actionable Agile Metrics for Predictability:* an introduction. Daniel S. Vacanti, Inc., 2015.

CAPÍTULO 6

Analisar a evolução do escopo e projetar prazos de entrega

6.1 Introdução

Desde que comecei a trabalhar com o desenvolvimento de software, tenho lidado com dois importantes aspectos: escopo de entrega e fluxo de desenvolvimento. O processo de alinhar as expectativas entre quem receberá o produto final e quem o constrói é geralmente árduo. Entretanto, quando acontece, aumenta as chances de sucesso da entrega, sem contar a diminuição de ruídos na comunicação entre as partes e o stress.

Ao monitorar o *throughput*, aprendemos que a equipe passa a ter insumos para analisar o ritmo de entrega, o que traz maior transparência para todas as pessoas envolvidas no fluxo de desenvolvimento. Se há uma expectativa quanto ao prazo de entrega, ao comunicarmos o quanto entregamos, invariavelmente teremos de lidar com a dúvida de quando o produto estará apto para uso.

Pensando em responder tal questionamento de forma assertiva, teremos a oportunidade de discutir neste capítulo o gráfico de *burnup* e como ele pode se tornar uma poderosa ferramenta de comunicação a ser usada com os *stakeholders*. Ao interagir com o conteúdo do capítulo, você aprenderá como o gráfico será útil para:

- Aumentar a visibilidade do processo de desenvolvimento de software, garantindo que as pessoas interessadas no produto estejam alinhadas quanto ao progresso.
- Apresentar, visualmente, como o crescimento do escopo pode influenciar o resultado entregue no final de um prazo estipulado.
- Comunicar projeções de entrega com o objetivo de garantir alinhamento quanto às expectativas de entrega.

Caso você já conheça ou utilize o *burnup*, confira o tópico onde apresento formas de simular cenários de projeção de entrega. Tenho certeza de que os métodos apresentados farão com que você melhore suas previsões.

Como último tópico do capítulo, teremos a chance de discutir como utilizar o *burnup* em desafios corriqueiros do dia a dia de equipes que estão criando ou evoluindo um software.

Ao final da leitura, você terá a chance de aprimorar a forma como lida e compartilha prazos de entrega a partir de métodos avançados de análise, levando informações mais seguras aos *stakeholders* e sendo mais previsível ao projetar o futuro.

6.2 Visualizar o escopo de um projeto através do gráfico de burnup

Equipes de desenvolvimento que trabalham em projetos ou em evoluções de produto de software onde há uma expectativa grande sobre a entrega comumente se deparam com o questionamento: quando o produto estará pronto para uso? Antes de pensar na data de entrega, se faz necessário analisar e alinhar o que se espera receber como resultado.

Clientes e usuários têm a expectativa de receber uma solução que, por vezes, possui mais funcionalidades do que de fato é necessário para provar a hipótese de um novo negócio, ou até mesmo para atender as necessidades do mercado.

Antes de partir para a construção de soluções mirabolantes de software que correrão o risco de não resolverem problemas reais e gerarão desperdício de tempo e dinheiro, a equipe, junto com aqueles que serão impactados pelo resultado do produto, devem pensar o que seria um MVP do que será produzido. Tal estratégia fará com que a equipe entregue rapidamente e de forma frequente pequenas partes da solução, sendo assim, serão obtidos *feedbacks* dos usuários o mais cedo possível. O software será construído incrementalmente, com as versões recém-criadas sendo inseridas ao produto já existente.

O QUE É UM MINIMUM VIABLE PRODUCT (MVP)?

Ries (2011) foi o criador do termo MVP e o definiu como sendo a versão de um produto que permite a equipe coletar o máximo de aprendizado validado sobre os clientes com o menor esforço possível.

Caroli (2015) define MVP como sendo a versão mais simples de um produto que pode ser disponibilizada para o negócio. O MVP determina quais são as funcionalidades mais essenciais para que se tenha o mínimo de produto funcional que possa agregar valor para o negócio (produto mínimo) e que possa ser efetivamente utilizado e validado pelo usuário final (produto viável).

A partir do momento que a equipe possui em mãos o escopo do produto que será produzido, o próximo desafio será gerir as mudanças que surgirão. Uma ferramenta que pode ser útil para esse desafio se chama gráfico de *burnup*.

O gráfico é composto por dois eixos: o horizontal apresenta um período de tempo, e o vertical demonstra o montante de trabalho que pode ser representado por número de histórias de usuário, número de *story points* etc.

Recomendo que você monitore o total de trabalho a partir de um número absoluto (como total de funcionalidades entregues; total de erros corrigidos) em vez de uma unidade de esforço (*story points* ou horas). A primeira medida demonstra de fato quanto foi produzido

ou o quanto ainda precisará ser feito, enquanto a segunda representa apenas um número de esforço.

Ainda sobre a estrutura do gráfico, ele possui duas linhas: o total de trabalho a ser feito e o total de trabalho concluído. A distância entre as linhas demonstra o quão distante aquela equipe está da entrega final. Idealmente as duas linhas se encontraram em algum instante do tempo.

Figura 6.1: Exemplo de visualização do gráfico de burnup

QUAL A DIFERENÇA ENTRE OS GRÁFICOS DE BURNDOWN E BURNUP?

Os gráficos de *burndown* e *burnup* são tipos de visualizações que gestores e equipes utilizam para monitorar e comunicar o progresso de uma entrega.

O gráfico de *burndown* mostra quanto trabalho está faltando ser concluído em um espaço de tempo (semana, quinzena, trimestre etc.). Já o gráfico de *burnup* apresenta quanto de trabalho foi feito e o total de trabalho necessário para a entrega. Tais gráficos são usualmente utilizados por equipes que trabalham com *frameworks* como Scrum ou algum tipo de metodologia ágil.

Figura 6.2: Exemplo de visualização do gráfico de burndown

Conforme ilustrado, o gráfico de *burndown* é muito simples. Uma única linha é traçada e alcança o valor 0 quando a entrega estiver concluída.

Enquanto o gráfico de *burndown* não exibe os avanços de entrega conquistados pela equipe, impedindo inclusive que pessoas que estão fora do dia a dia acompanhem o desempenho na medida em que são incluídas novas demandas, o gráfico de *burnup* permite que quem o visualize avalie rapidamente se tudo está em conformidade com o planejado ou se é preciso mobilizar algum tipo de plano de ação. Além disso, o gráfico de *bundown* não apresenta variações no escopo do que está sendo trabalhado.

Mudanças no escopo ocorrem quando um trabalho é adicionado ou removido. Vivenciamos mudanças quando um terceiro pede a inclusão de funcionalidades não previstas, ou quando é necessário remover algo do que havia sido planejado para que o prazo de entrega estipulado seja alcançado. O gráfico de *burndown* não mostra essa informação tão claramente quanto o gráfico de *burnup*, pois não possui, em uma mesma visualização, o tamanho do escopo (o que precisa ser feito) e o número de entregas realizadas até um determinado momento (o que já foi feito).

Dentro das equipes, sugiro a utilização do *burnup* como forma de responder às seguintes perguntas:

- O escopo do projeto tem crescimento de forma saudável?

- Quando finalizaremos o atual escopo de trabalho?

No contexto de equipes que trabalham com métodos ágeis, tal representação gráfica se torna essencial para o trabalho de gestores de produtos ou pessoas responsáveis por maximizar o valor do produto e o trabalho que é desenvolvido pela equipe. Os motivos?

1. Melhorar a comunicação — O gráfico de *burnup* consegue dar visibilidade de como está o status de uma entrega, e por isso funciona como uma ferramenta eficaz no gerenciamento das expectativas.

2. Responder a mudança — Acompanhando periodicamente o gráfico, a equipe poderá diagnosticar problemas, como por exemplo, aumento da quantidade de itens do escopo ou queda no ritmo de entrega, que possam estar afetando a entrega do produto que está sendo construído.

3. Analisar mudanças no escopo de entrega — Qualquer tipo de acréscimo existente no escopo estará explícito no gráfico e será útil para que a equipe discuta se de fato o que está sendo incluído trará valor para o projeto ou produto.

Agora que aprendemos a motivação de utilizar o gráfico de *burnup* e sua diferença para o gráfico de *burndown*, nada melhor do que um exemplo para compreender a utilidade da ferramenta.

Imagine que Luciana é a gestora de um projeto e faz parte da equipe *XPTO*, responsável por construir um novo aplicativo de vendas para um cliente. O projeto que ela estava atuando tinha o seguinte contexto:

- **Duração:** 24 semanas (6 meses).

- **Escopo da entrega:** após uma sessão de entendimento do problema, a equipe e todos os envolvidos com o projeto conseguiram pensar no que seria a versão mais enxuta para validar o aplicativo com os usuários.

- **Desafio do projeto:** garantir que o software estivesse disponível para uso antes do prazo esperado (6 meses).

Após buscar ferramentas que pudessem ajudá-la no acompanhamento do trabalho da equipe, Luciana percebeu que o gráfico de *burnup* seria útil para comunicar o progresso do projeto. O gráfico foi projetado para mostrar o total de histórias do usuário previstas no escopo e o número de histórias de usuário entregues.

No começo do projeto (6 primeiras semanas), Luciana se viu em uma situação onde, semanalmente, o escopo crescia a uma taxa de 5 histórias e, em contrapartida, a equipe

conseguia entregar 2 histórias. Ao analisar o gráfico e a divergência entre o aumento do escopo e a cadência de entrega, ela começou a procurar causas para o que estava acontecendo.

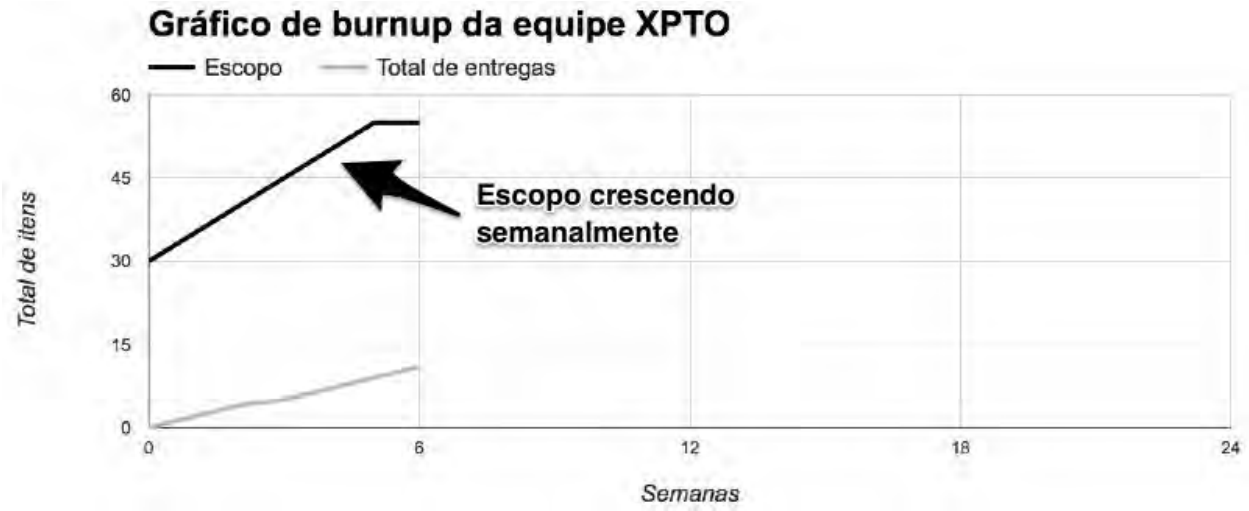


Figura 6.3: Momento inicial do projeto da equipe XPTO

No que diz respeito ao escopo, ela percebeu que o crescimento se dava por conta dos refinamentos que estavam sendo feitos após a sessão de mapeamento do produto. Mesmo entendendo que o escopo cresceria conforme a equipe fosse descobrindo detalhes sobre a solução, Luciana precisou alinhar com os *stakeholders* do projeto que, se tal taxa de crescimento do escopo se mantivesse, dificilmente a equipe conseguiria cumprir o prazo de entrega.

Luciana conseguiu tomar esse tipo de iniciativa, pois tinha em mãos dois indicadores que mostravam o quanto o time estava entregando por semana (taxa de saída) e quanto o escopo da entrega crescia (taxa de entrada). Tais indicadores são úteis para que você consiga administrar possíveis mudanças no escopo e para que a equipe acompanhe o seu ritmo de entrega.

No início de um projeto, é comum que o escopo cresça, afinal, a equipe está começando a se familiarizar com o produto que será construído. Porém, preste atenção se você começar a se deparar com um cenário no qual o escopo não para de aumentar, ou a taxa de entrega da equipe está menor do que a taxa de crescimento do escopo. Será o momento de priorizar ações que garantirão a entrega de valor do produto no futuro.

O QUE É UMA SESSÃO DE REFINAMENTO?

O processo de refinamento consiste em discutir de forma mais detalhada as demandas do escopo que serão trabalhadas em breve, isto é, os próximos da fila para quem trabalha com o método Kanban ou os do próximo *sprint* para uma equipe Scrum.

Tal cerimônia reduz excessos de dúvidas (técnicas ou de negócio) durante o desenvolvimento; diminui a constatação, durante o desenvolvimento, de que a complexidade é maior do que a esperada inicialmente; e auxilia na queda de retrabalho, porque o que foi desenvolvido não endereça o problema principal.

Outro ponto importante de se comentar é que uma demanda mal definida tende a ter um *lead time* alto. Vamos supor que devido a dúvidas, retrabalho e incertezas, uma demanda mal refinada tenha um *lead time* de 8 dias em vez de 5 dias de uma demanda bem definida. Vamos supor ainda que fossem necessários 2 dias para refiná-la melhor (se reunir com a área de negócios, remover algum bloqueio etc.).

O tempo total incluindo tal refinamento será de 7 dias (2 para refinamento, 5 de *lead time*), deixando de se desperdiçar 1 dia em comparação com os 8 da história mal compreendida. Dessa forma, pode ser melhor até mesmo deixar a equipe parada, sem desenvolver, do que colocar demandas no fluxo "a qualquer custo".

Voltando ao exemplo, passadas mais 6 semanas, Luciana continuava a se deparar com uma situação em que o escopo não parava de crescer. Neste momento, ela precisou sentar com o cliente a fim de alinhar maneiras para simplificar o escopo, porque se o projeto continuasse com tal comportamento, dificilmente a equipe conseguiria entregar algum produto que gerasse valor no prazo estipulado.

O *burnup* também foi útil para ela definir junto à equipe estratégias para melhorar o *throughput* (número de histórias entregues) baseado nos resultados apresentados naquele momento.

Figura 6.4: Momento crítico do projeto da equipe XPTO

Caso você se depare com a difícil situação de trabalhar em uma entrega onde há um prazo fixo e o cliente lhe pede a cada interação novas funcionalidades, tenha muita atenção ao aceitar novas demandas e explicite o risco de não entregar a solução esperada na data fixada.

Entregar software de forma iterativa (em curtos espaços de tempo) e incremental (adicionando novos recursos a cada ciclo) é fascinante, pois nos permite acompanhar e validar de forma contínua o produto que está sendo construído. Porém, se não for feita uma gestão das expectativas de quem está acompanhando o progresso do que está sendo produzido, a equipe corre o risco de sofrer a pressão por entregar cada vez mais e o produto pode nunca ter um fim claro.

Em situações nas quais a equipe sabe que não conseguirá entregar todos as funcionalidades esperadas na data que havia sido fixada, exercite com as pessoas uma outra maneira de pensar e as questione: o que é possível entregar na data prevista? Tal mudança de pensamento fará com que a equipe, o cliente e quaisquer pessoas envolvidas na entrega priorizem as funcionalidades que são de fato importantes e alinhem quando as funcionalidades com menor prioridade poderão ser entregues. Tendo tal atitude, você perceberá que a equipe se sentirá mais confiante e que o cliente terá um maior entendimento do que esperar como produto.

Depois de muito trabalho, discussões e melhorias, Luciana e a equipe chegaram ao final do projeto vivos e com uma versão funcional do novo aplicativo de vendas. O cliente estava satisfeito pelo resultado apresentado, pois conseguirá criar um novo canal de vendas para a empresa e já colhia resultados em seu faturamento. Antes de dar como finalizado o projeto, a equipe decidiu realizar uma sessão de lições aprendidas.

Um dos tópicos discutidos no encontro foi como o *burnup* havia sido útil e eis que surgem as seguintes razões:

- O gráfico trouxe visibilidade para o processo de desenvolvimento de software, garantindo que o cliente estivesse alinhando com o progresso das entregas que estavam sendo feitas.

- O gráfico foi uma ferramenta que demonstrou, visualmente, como o crescimento do escopo aumentava o risco de não haver uma entrega de valor no prazo esperado.

- A equipe utilizou a ferramenta como forma de garantir alinhamento sobre o prazo de entrega, pois a todo momento recorria a ela como insumo para analisar o cenário presente, baseado nas entregas e no escopo a ser feito, a fim de projetar o final do projeto.

- Ao monitorar o gráfico periodicamente, a equipe pôde detectar riscos e teve tempo para desenvolver manobras a fim de solucionar as ameaças que surgiram ao longo do tempo de projeto.

O exemplo apresentado nos mostrou uma série de benefícios para que você passe a utilizar o *burnup* no seu dia a dia de entregas de software. Percebemos que o gráfico nos traz uma maneira direta de enxergar o que está acontecendo com uma entrega já que combina ritmo de entrega e escopo.

Mudanças constantes no escopo é um dos grandes inimigos quando estamos produzindo ou evoluindo algum software. A partir de agora, você poderá utilizar o *burnup* como forma de deixar visível esse tipo de problema para qualquer tipo de pessoa impactada pela entrega. A aplicação da ferramenta poderá até mesmo ajudá-lo a convencer as pessoas a pararem de solicitar alterações e permitir que a entrega seja realizada.

Como observação final, sugiro que você e a equipe acompanhem semanalmente o gráfico de *burnup*. Tal hábito garantirá que todos estejam analisando, constantemente, formas de aumentar o número de entregas e a controlar o crescimento do trabalho a ser feito.

Uma pergunta que você pode estar se fazendo neste momento é: como eu poderia criar projeções de quando o total de entregas atingirá o escopo a ser feito? Ou em outras palavras, você deve estar buscando uma resposta para a pergunta que a maior parte das pessoas de fora de uma equipe faz: quando o software será entregue?

Para que você aposente a técnica de estimativa baseada no "achismo", apresentarei no próximo tópico uma coletânea de métodos para que você e sua equipe melhorem a árdua tarefa de antever o futuro.

6.3 Projetar cenários a partir do throughput

Quase todos os envolvidos na entrega de uma melhoria ou de um novo software estão habituados a se questionar sobre prazos de entrega. Toda vez que encaro esse tipo de pergunta, geralmente a respondo da seguinte maneira: "Bem, a única coisa que eu sei é que a incerteza é uma das poucas certezas para quem trabalha com desenvolvimento de software, portanto, vamos analisar as métricas da equipe para tentar projetar esse grande mistério com algumas técnicas analíticas".

Neste tópico, aprenderemos a utilizar o *throughput* como variável para projetar datas de entrega. Como primeira etapa, mostrarei a você porque você não deve utilizar a média de *throughput* como forma de projetar um prazo. Em seguida, veremos como utilizar algumas estatísticas descritivas para projetar cenários pessimista, mais provável e otimista.

Atenção ao usar o throughput médio

Como temos visto ao longo do livro, métricas são importantes, pois direcionam melhorias e são uma ótima ferramenta para ajudar equipes a focar suas pessoas e recursos no processo de desenvolvimento de software com qualidade, através de uma maior visibilidade do presente e com embasamento para projetar o futuro.

Uma das métricas-chave que aprendemos até aqui é o *throughput*. Só recordando, o *throughput* representa o número de itens entregues em um período de tempo.

Um ponto importante de se destacar é de que, quando usamos o *throughput* como variável para esboçar datas de entrega, precisamos de que os itens que passam pelo processo de desenvolvimento tenham um esforço parecido. Em outras palavras, é como se precisássemos de itens onde a variabilidade do tempo necessário para a entrega fosse baixa. O motivo? Garantir que o processo está estável o suficiente para que as projeções façam sentido.

A resposta clássica para a pergunta sobre quando uma entrega será concretizada é, geralmente, a seguinte: "baseado em nossa média de entrega, devemos levar x semanas para entregar o escopo atual". Veremos nos exemplos a seguir que utilizar a média pode ser arriscado quando você está buscando formas de idealizar o futuro. Compartilharei três histogramas representando diferentes casos para compreendermos como a média é sensível às circunstâncias dos dados históricos que estão sendo analisados.

Caso não conheça, o histograma é uma exibição gráfica que utiliza barras de alturas diferentes para realçar a frequência de vários pontos de dados dentro de um conjunto geral. Os gráficos apresentados a seguir são compostos pelo *throughput* no eixo horizontal (x) e a frequência de contagem no eixo vertical (y). O benefício deste gráfico é que ele nos oferece uma ideia geral da forma da distribuição dos dados que será importante para identificarmos a localização da média.

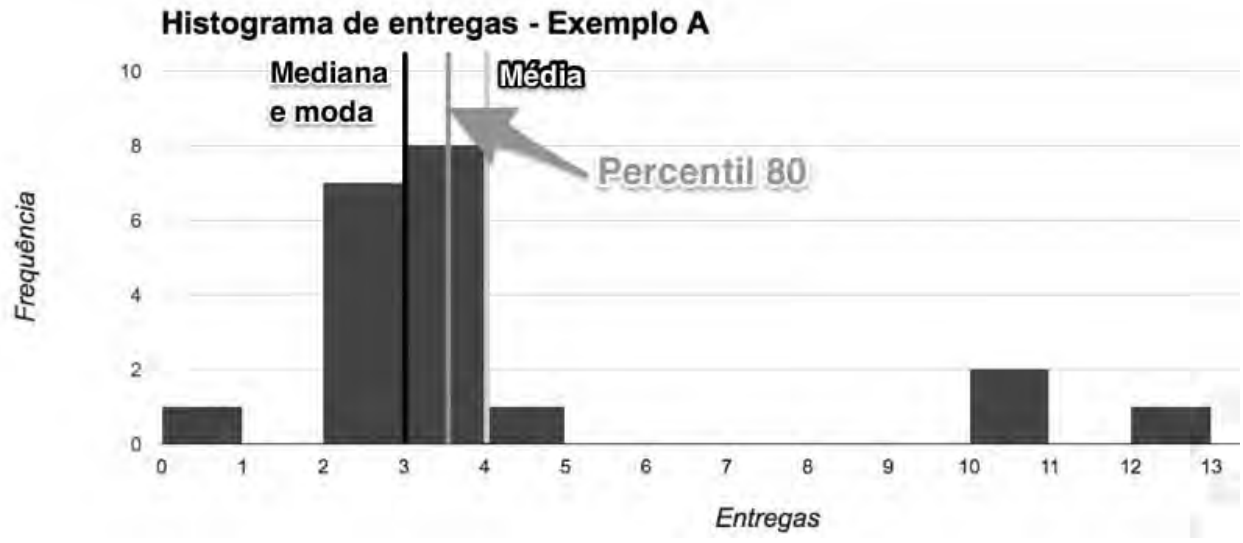


Figura 6.5: Média: caso 1

Analisando o primeiro caso, percebemos um cenário no qual a média de *throughput* é aproximadamente igual a 4 histórias por semana, e a mediana (número que separa o conjunto de dados no centro) é de 3 de histórias por semana.

Se pedíssemos para esta equipe estimar o seu *throughput* em uma semana qualquer, provavelmente ela responderia 3, como resultado da moda (valor que aparece mais vezes no conjunto de dados). E então, será que neste caso a média é uma boa referência para ser utilizada como padrão de entrega?

Mesmo que a média não esteja tão distante da mediana, temos um contexto onde 80% dos valores de *throughput* estão abaixo da média, o que significa dizer que os outros 20% estão acima. Consequentemente, a média representa um conjunto exclusivo de casos (ela está localizada no 1/5 final da amostra) e, portanto, não é uma referência confiável.

O exemplo do primeiro caso nos mostra que, para deduzir algum tipo de informação dos dados, é importante que conheçamos o tipo de distribuição apresentada. Por conta disso, não podemos dizer que a média, representada isoladamente é uma medida confiável. Precisamos de outras informações para ter uma análise mais acurada.

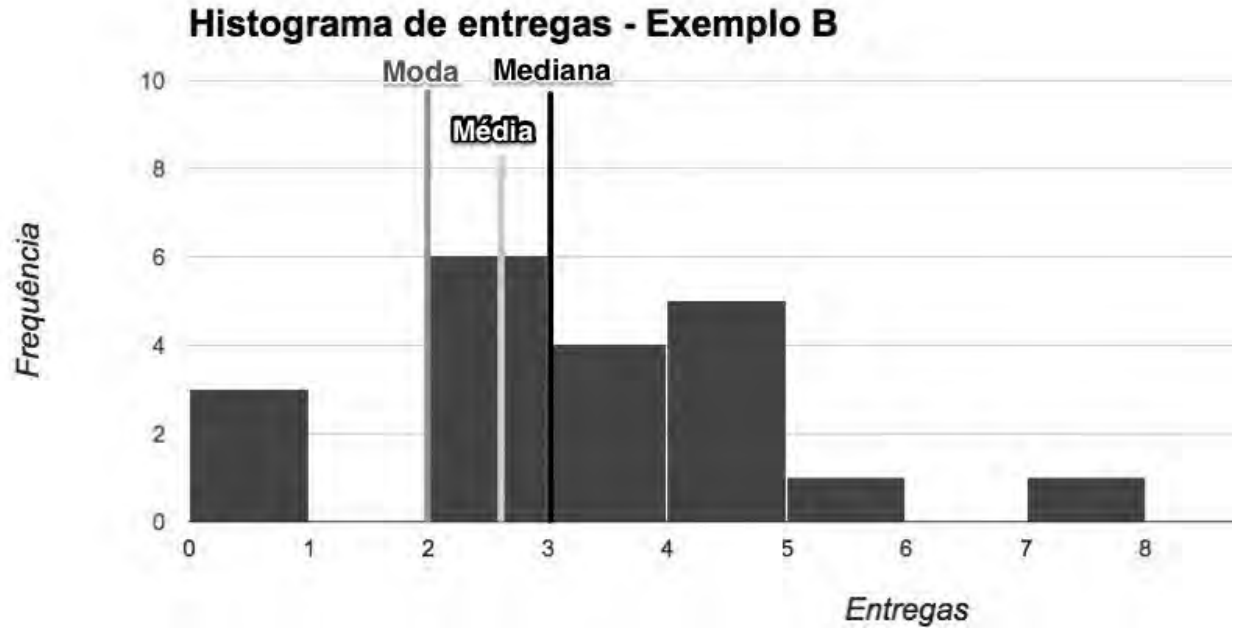


Figura 6.6: Média: caso 2

No segundo caso, temos uma situação em que a média está localizada abaixo da mediana, o que significa dizer que ela representa menos do que 50% do *throughput* da equipe. Portanto, utilizá-la neste caso como referência poderia soar como uma estimativa

pessimista, dado que em 50% dos casos ou mais o valor do *throughput* é maior do que o valor médio.

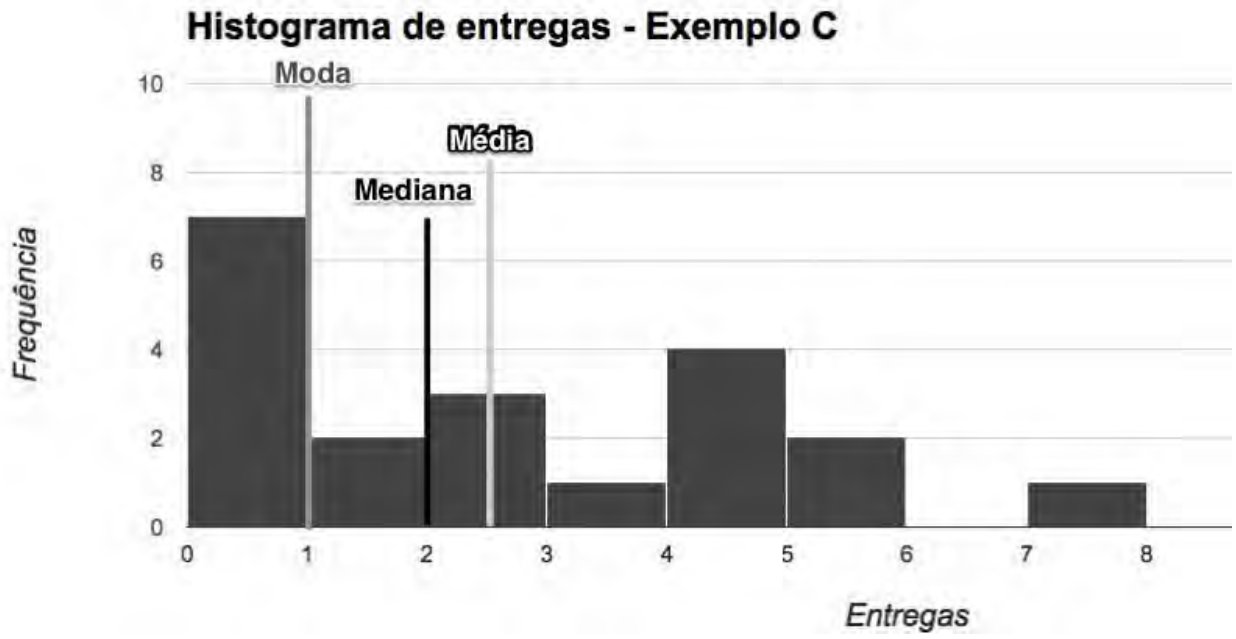


Figura 6.7: Média: caso 3

O terceiro caso nos apresenta uma situação na qual temos uma alta concentração semanas sem entrega (a moda é 0). Neste caso, será arriscado utilizar a média como referência para a projeção, pois, como pudemos observar, é comum o *throughput* ser zero. A pergunta correta a ser feita nesta situação seria: por quais motivos a equipe está passando tantas semanas com o *throughput* zerado?

Outra observação importante a ser feita sobre a média é que valores extremos podem enviesar o seu valor, o que pode fazer com que ela não represente a real tendência de valor central do conjunto de dados. Quando uma distribuição apresenta dados simétricos, as medidas de tendência central (média e mediana) serão basicamente as mesmas. Se os dados forem assimétricos, as medidas poderão se aproximar das observações mais extremas.

Pelo que vimos nos exemplos deste tópico, raramente as distribuições de *throughput* apresentarão média, mediana e desvio padrão idênticos. À medida que os dados se distorcem, a média perde sua capacidade de fornecer a melhor localização central, porque as extremidades da distribuição a empurraram para longe do ponto que representa a divisão dos dados.

A partir de agora, em vez de utilizar a média como resposta padrão para determinar a tendência de entrega de uma equipe, você analisará com mais cautela a distribuição de dados. No próximo tópico, você verá como algumas estatísticas descritivas poderão lhe apoiar no processo de projeção de um intervalo de entrega.

Criar cenários a partir das estatísticas descritivas

Aprendemos até aqui que a média pode não ser confiável quando há o desafio de se projetar a tendência de entrega de uma equipe. A seguir, teremos a oportunidade de descobrir como outras medidas oriundas da estatística descritiva poderão ser úteis quando alguém lhe perguntar qual o prazo para a entrega de um produto de software.

Quando lidamos com uma situação, como é o caso de quem trabalha com o desenvolvimento de software, onde a todo momento a única certeza que existe é a mudança, tentar ser determinístico ao projetar uma data de entrega é algo arriscado e, na grande maioria das vezes, gera frustração por nunca ser atingido.

O que será apresentado é fruto de uma corrente que projeta cenários futuros a partir de comportamentos do passado, ou seja, trabalhando com probabilidades. Pensando nisso, sugiro que, a partir de hoje, você pare de fixar prazos de entrega e passe a trabalhar com intervalos que vão de uma visão otimista para uma mais provável e, por fim, para um cenário pessimista.

A dúvida que pode estar pairando em sua cabeça agora é: como? A partir de variáveis como o mínimo, máximo, moda, mediana, média e percentil, podemos extrair valiosas informações para definir os cenários futuros.

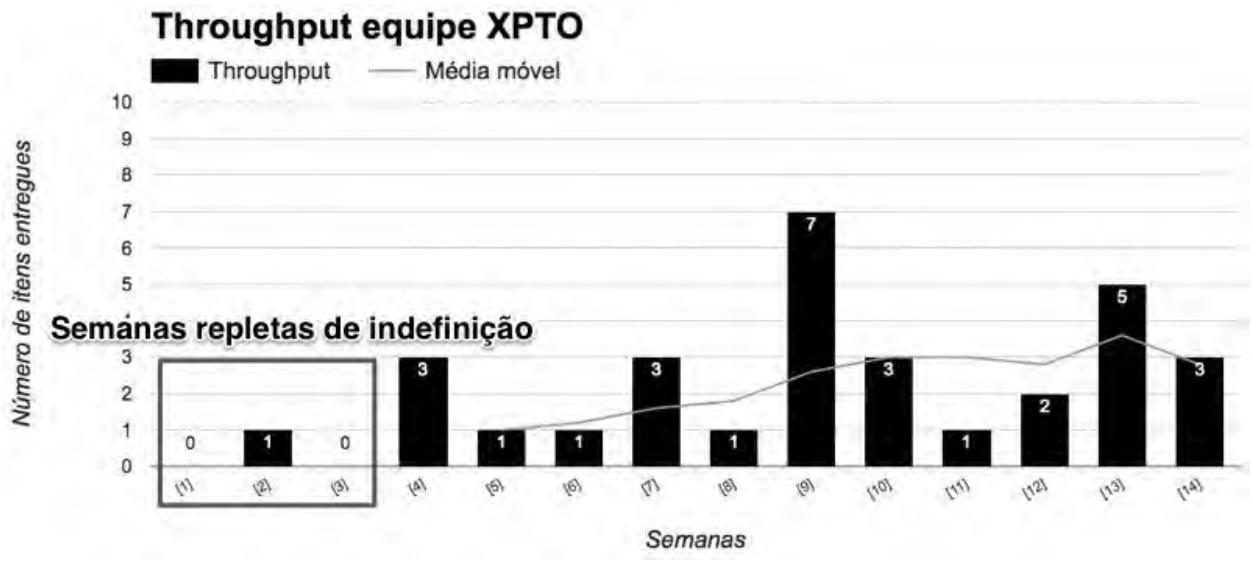
Vamos voltar ao exemplo da Luciana, gestora de produto da equipe *XPTO*. Ela está trabalhando em um novo projeto e já se passaram 14 semanas desde o seu início. O cliente tem uma expectativa de que a entrega seja feita até a semana 28. A equipe conseguiu definir um escopo que dificilmente sofrerá alterações, pois foi acordado que, se fosse necessária a criação de uma nova funcionalidade, outra seria despriorizada.

Antes de discutir com a equipe sobre quais seriam os cenários, Luciana decidiu extrair algumas informações do histórico de *throughput* para formular uma resposta para o cliente, que naquele momento estava agoniado por um prazo. Levando em consideração as 14 semanas, o time *XPTO* tinha o seguinte cenário:

- **Mínimo:** 0 (menor valor de entrega semanal da equipe).
- **Máximo:** 7 (maior valor de entrega semanal da equipe).
- **Moda:** 1 (valor de entrega semanal que mais vezes aconteceu).

- **Mediana:** 2 (representa que 50% das semanas tiveram 2 entregas).
- **Média:** 2 (valor médio somando todos os valores de *throughput* e dividindo pelo total de semanas).
- **Percentil 25:** 1 (representa que 25% das semanas tiveram um *throughput* menor ou igual a 1).
- **Percentil 75:** 3 (representa que 75% das semanas tiveram um *throughput* menor ou igual a 3).
- **Percentil 95:** 6 (representa que 95% das semanas tiveram um *throughput* menor ou igual a 6).

Em um primeiro momento, Luciana estranhou, pois não recordava que a equipe havia ficado semanas sem entregar. Inclusive, nas últimas semanas, o *throughput* vinha aumentando. Pensando nisso, ela complementou a análise que estava sendo feita avaliando



o número de entregas realizadas semanalmente pela equipe.

Figura 6.8: Caso XPTO: análise do throughput até a semana 14

Após visualizar o gráfico, ela recordou que a equipe realizara apenas uma entrega nas três primeiras semanas do projeto por conta de problemas de ambiente e por incertezas técnicas que precisavam ser sanadas na época.

Considerando que utilizar as três primeiras não ajudaria em sua análise, Luciana decidiu criar uma nova amostra de dados considerando o histórico de *throughput* a partir da quarta semana e o cenário desenhado foi:

- **Mínimo:** 1 entrega por semana.
- **Máximo:** 7 entregas por semana.
- **Moda:** 3 entregas por semana.
- **Mediana:** 3 entregas por semana.
- **Média:** 3 entregas por semana.
- **Percentil 25:** 1 entrega por semana.
- **Percentil 75:** 3 entregas por semana.
- **Percentil 95:** 6 entregas por semana.

Antes de seguir com a análise do caso, gostaria de aconselhá-lo a levar em consideração uma amostra de dados que descreva a realidade da equipe, assim como a Luciana fez no exemplo da equipe *XPTO*.

Por vezes, o passado da sua equipe pode ter sido ruim. No entanto, se o histórico recente apresenta uma melhora, utilize-o como referência. Uma informação útil para avaliar tendência é a média móvel, pois ela nos aponta, dentro de um intervalo de tempo determinado e não a partir da série histórica toda, se o *throughput* está crescendo ou diminuindo.

Voltando ao exemplo da equipe *XPTO*, Luciana decidiu traçar três possíveis perspectivas de entrega como forma de não ser determinística em suas projeções.

No caso do cenário otimista, Luciana utilizou como variável de referência o percentil 95. Ela tomou essa decisão, pois, em apenas 5% das semanas, a equipe conseguiu entregar mais do que 6 funcionalidades.

Sugiro que você utilize o percentil 95 e não o máximo, afinal, dificilmente a equipe conseguirá repetir o caso extremamente positivo. Geralmente, *throughput* muito alto e fora do padrão é vestígio de um passado de muito trabalho e de poucas entregas que culmina em um momento onde todo o esforço é entregue de uma vez. Tais situações acontecem, por exemplo, quando um ambiente fica indisponível por muito tempo, quando existe gargalo em uma etapa de aprovação ou onde é necessário o trabalho de um especialista (por exemplo, apenas um especialista em teste é responsável por validar as funcionalidades construídas pela equipe).

Pensando no cenário mais provável, Luciana lembrou de que a média pode, por vezes, carregar o viés dos casos extremos positivo e negativo. Ponderando essa questão, ela

decidiu utilizar, independente do valor apresentado, a mediana como tendência de entrega da equipe, pois, em 50% das semanas, a equipe entregou até 3 funcionalidades.

No conjunto de dados da equipe *XPTO*, a média, a mediana e a moda apresentaram o mesmo valor. No entanto, dependendo do histórico de *throughput*, tais medidas poderão ser bem diferentes como vimos no tópico anterior. Sendo assim, pensando em probabilidades, ao utilizar a mediana, você estará corroborando com 50% dos casos da sua amostra.

Para o cenário pessimista, Luciana decidiu utilizar o percentil 25. Mas, por quê? Ao ler a informação, ela chegou à conclusão de que a equipe conseguiu em 75% das semanas entregar mais do que 1 funcionalidade. Isso significa dizer que as chances da equipe entregar mais são boas, mas, neste momento o que ela queria era pensar no pior caso.

Em vez de utilizar o mínimo que pode ser um cenário pessimista demais, recomendo que você utilize o percentil 75. Afinal, em apenas 1/4 das situações sua equipe apresentará um resultado inferior ao apresentado pela medida.

Feitas as análises, Luciana sentou com a equipe e explicou o método que havia sido usado para realizar as projeções com o objetivo de validar se o que havia sido feito fazia sentido e refletia a realidade do projeto naquele momento.

Assim como fez Luciana, aconselho que você discuta com os especialistas (equipe) se de fato os números que estão sendo apresentados são coerentes. Queimar a etapa de avaliação da equipe e apresentar as projeções diretamente para qualquer interessado em saber os possíveis prazos de entrega pode ser um risco, pois você deixará de escutar as pessoas que conhecem os detalhes técnicos por trás do que está sendo construído. A pior coisa que pode acontecer ao projetar entregas é a venda de um prazo não realista.

Incrementando a visualização do gráfico de *burnup*, apresentarei a seguir uma nova versão onde serão projetados os cenários a partir de três retas oriundas da semana atual. Além disso, o número de entregas será explicitado por colunas (*throughput* acumulado) e o escopo por um gráfico de linhas.

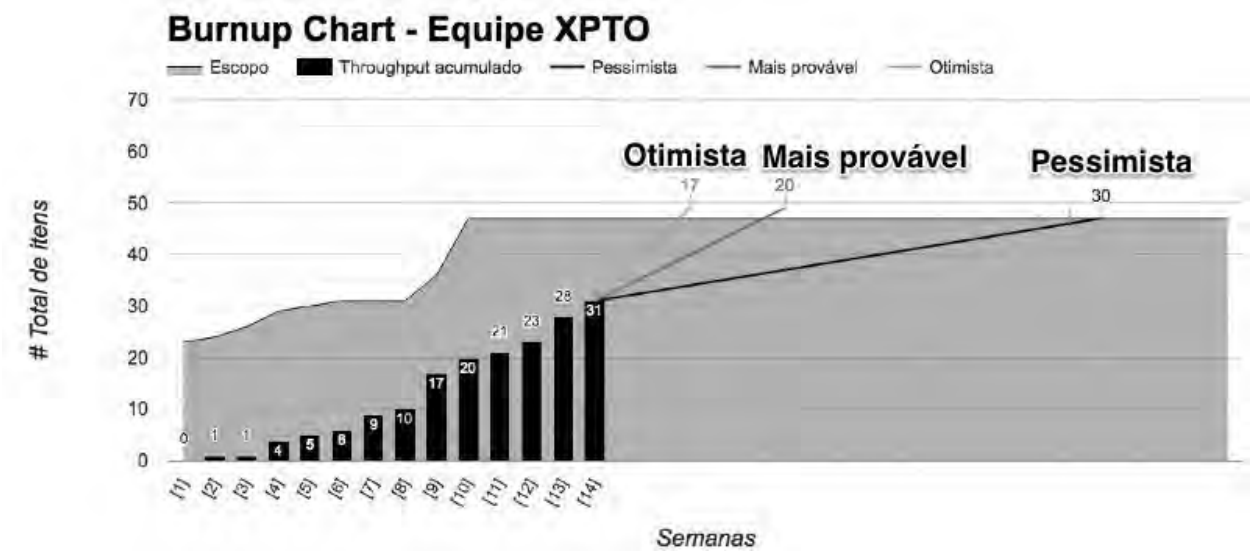


Figura 6.9: Caso XPTO: burnup com a projeção dos cenários

Retornando ao exemplo da equipe *XPTO*, depois de ter o aval da equipe quanto às projeções, Luciana apresentou o gráfico anterior para o cliente, explicitando os seguintes cenários:

- **Otimista:** onde seriam feitas 6 entregas por semana a partir do momento atual, o que representava a entrega do projeto em mais 3 semanas (semana 17).
- **Mais provável:** onde a equipe tinha 50% de chances de manter o ritmo de entrega que estava sendo apresentado até então, o que representava a conclusão do projeto em mais 6 semanas (semana 20).
- **Pessimista:** onde a equipe entregaria uma funcionalidade por semana e o projeto tomaria mais 16 semanas para ser finalizado (semana 30).

Em resumo, Luciana assegurou ao cliente que o projeto seria entregue entre as semanas 20 e 30. Tal informação o tranquilizou, pois a expectativa de prazo que ele tinha em mãos, que era a semana 28, estava dentro do intervalo exposto.

Projetar prazos de entrega não é uma atividade trivial, por isso deve ser tratado com muito cuidado para não gerar expectativas que não serão atingidas e, principalmente, para que se aumente as chances de se prever a entrega de algo dentro de intervalos baseados no que está sendo apresentado pela equipe (histórico).

Neste tópico, tivemos a oportunidade de utilizar uma forma probabilística para estimar prazos de entrega em vez de algo determinístico, o que significa dizer que agora você tem em mãos um método baseado em estatísticas para desenhar cenários futuros em vez de estimar às cegas.

A seguir, apresentarei como usar a técnica estatística de Monte Carlo como alternativa para se apresentar quais são as probabilidades de um determinado escopo ser entregue.

6.4 Utilizar o método de Monte Carlo para desenhar cenários de entrega

Vimos anteriormente que a estatística descritiva é uma ótima ferramenta para a construção de cenários. Nesta seção, apresentarei uma proposta de aplicação do modelo de Monte Carlo como técnica para simular probabilidades de uma entrega acontecer em um dado instante de tempo, baseado no histórico de *throughput*.

O que é a simulação de Monte Carlo?

A simulação de Monte Carlo é uma técnica estatística que tem como objetivo estimar a probabilidade de um certo resultado, por meio da geração de centenas ou milhares de cenários baseados em números aleatórios para as entradas em um sistema. Em outras palavras, a simulação pode ser definida como um método que se baseia em uma grande quantidade de amostragens aleatórias para se chegar em resultados próximos dos reais.

O nome Monte Carlo surgiu como alusão a uma de suas principais características: a aleatoriedade na qual ocorre os jogos de cassino. Geralmente, o método é aplicado a problemas altamente incertos em que a computação direta é difícil, impraticável ou impossível. Ela provou ser uma ferramenta útil em diferentes situações, como na física nuclear, exploração de petróleo e gás, finanças, seguros etc.

Um exemplo simples da aplicação de simulação de Monte Carlo se dá quando é preciso calcular a probabilidade de conseguir um número maior ou igual a 12 ao somar o lançamento de 3 dados de 6 faces. Para resolver esse problema com a simulação, a pessoa pode simular em um programa de computador o lançamento dos 3 dados 5.000 vezes. Imagine que, após todas as simulações, obtiveram-se os seguintes resultados:

Resultado da soma dos 3 dados	Número de ocorrências
3	21
4	70

5	134
6	240
7	370
8	509
9	581
10	586
11	636
12	**538**
13	**476**
14	**375**
15	**226**
16	**139**
17	**74**
18	**25**

Uma estimativa da probabilidade de a soma dos dados ser maior ou igual a 12 seria então a soma das ocorrências em que a soma foi maior ou igual a 12, dividido pelo número total de lançamentos de dados:

$$(538 + 476 + 375 + 226 + 139 + 74 + 23) \div 5000 = 37.06\%$$

Figura 6.10: Fórmula: resultados nos quais a soma dos dados foi maior do que 12

O resultado anterior diz que existe 37,06% de chance de a soma dos 3 dados lançados atingir um valor maior ou igual a 12. Da mesma forma que foi feito com 3, o modelo poderia ser replicado para qualquer quantidade de dados.

Como a simulação de Monte Carlo pode ser útil no desenvolvimento de software?

Para o contexto do desenvolvimento de software, podemos pensar em um algoritmo de simulação de Monte Carlo com o objetivo de determinar a probabilidade da finalização do escopo de um projeto em um certo momento no tempo (semana, mês etc.).

Ao longo deste livro, foram apresentadas métricas como *lead time* e *throughput*, que podem ser úteis, quando devidamente analisadas, para projetar resultados possíveis no futuro a partir do que aconteceu no passado.

Magennis (2011) apresenta um modelo que utiliza as informações do *lead time* de todas as etapas do processo de desenvolvimento de uma equipe como forma de projetar cenários de entrega. A seguir, apresentarei uma outra proposta que usa o *throughput* como variável base das simulações.

Antes que você passe a utilizar técnicas de projeção de cenários, sugiro que você faça uma reflexão sobre o quão saudável está o seu processo de desenvolvimento hoje. Se você tem um processo instável, os cenários futuros projetados pelas simulações não lhe dirão muita coisa.

A partir do que discutimos até aqui no livro, você pode identificar que o processo não está saudável quando:

1. Existe uma alta variabilidade no *lead time*, isto é, itens levam muito tempo para serem entregues e outros são finalizados rápido demais.

2. São observados gargalos ao longo do processo em decorrência do acúmulo de trabalho em etapas intermediárias. Alguns exemplos são: excesso de demandas aguardando validação; muito trabalho sendo estocado em uma etapa que exige o trabalho de um especialista (como teste ou *deploy*).

3. Há um excesso de trabalho em andamento (WIP elevado). A taxa em que as demandas entram no processo é maior do que a taxa de saída (entrega).

Quando estabilizado o processo, você estará apto a trabalhar com a simulação de Monte Carlo, que possui a seguinte estrutura:

1. Variáveis de entrada do sistema;

2. Distribuição probabilística usada para cada variável;

3. Variável de saída do sistema.

Ao trazer a estrutura apresentada para o dia a dia de uma equipe que monitora a informação de *throughput*, o modelo seria representado da seguinte maneira:

- **Variáveis de entrada do sistema:** *throughput* semanal.

- **Distribuição probabilística:** *bootstrap* com reposição da amostra de *throughput* já observada no projeto.
- **Variável de saída do algoritmo:** semana em que o escopo é finalizado, junto com a probabilidade de ocorrência.

O QUE É BOOTSTRAP?

O termo *bootstrap* se refere, em geral, a uma técnica ou método de simulação, que objetiva a obtenção de intervalos de confiança para as estimativas dos parâmetros de interesse, por reamostragem do conjunto de dados original.

A base da técnica é a obtenção de um "novo" conjunto de dados, por reamostragem do conjunto de dados original (EFRON; TIBISHIRANI, 1993).

Para ilustrar o uso do método de Monte Carlo, nada melhor do que recorrer ao exemplo da equipe *XPTO*.

Após a reunião de apresentação das projeções para o cliente, Luciana (gestora do projeto) decidiu avaliar se as projeções que haviam sido apresentadas eram realistas. Para isso, recorreu ao método de Monte Carlo como forma de levantar quais eram as chances de a entrega acontecer ao longo das semanas futuras.

Como primeiro passo para a aplicação do método, ela analisou qual havia sido o histórico de *throughput* da equipe e qual era o tamanho do escopo. Assim como havia sido feito com a análise descritiva dos dados, as 3 primeiras semanas do projeto não foram consideradas para as projeções, pois representaram um cenário passado que não se repetiria.

Semana	Throughput
4	3
5	1
6	1
7	3
8	1
9	7

10	3
11	1
12	2
13	5
14	3

Analisando os dados do projeto, Luciana percebeu que o tamanho do escopo era de 47 funcionalidades, sendo que 31 delas haviam sido entregues até a semana 14. Utilizando os dados históricos de *throughput*, Luciana aplicou o seguinte método:

1. A simulação começaria na semana 15. Para isso, gerou-se um valor de *throughput* aleatório para a semana 15 a partir do *bootstrapping*, com reposição do conjunto de *throughput* das semanas que já haviam acontecido, que no exemplo seriam as semanas 4 a 14. Para gerar o *throughput* futuro, o modelo escolheu de modo aleatório um valor dentro do vetor de *throughput* observado: [3, 1, 1, 3, 1, 7, 3, 1, 2, 5, 3]. Digamos que, em uma primeira iteração, a vazão gerada tenha sido de 2.

2. Em seguida, calculou-se a vazão acumulada da semana 15, que representou a vazão acumulada da semana anterior somada a vazão da semana que havia sido simulada. Nesse caso, a vazão acumulada simulada seria de 33 (31 + 2).

3. Como a vazão acumulada (33) ainda era menor que o escopo do projeto (47), o algoritmo simulou a semana 16 e assim sucessivamente até que a vazão acumulada coincidisse com o tamanho do escopo. Digamos que, para a semana 18, a vazão acumulada tenha atingido 47, que é igual ao tamanho do escopo. Logo, essa iteração da simulação seria finalizada e salva uma ocorrência de término do projeto para a semana 18.

4. Esse processo iterativo se repetiu mais 5.000 vezes, calculando o número de ocorrências de término para cada semana encontrada.

Após a execução do método, Luciana quis compreender em quais semanas o projeto tinha maiores chances de ser concluído. Para isso, ela avaliou a probabilidade acumulada de cada semana, que nada mais era do que soma das probabilidades anteriores a semana investigada.

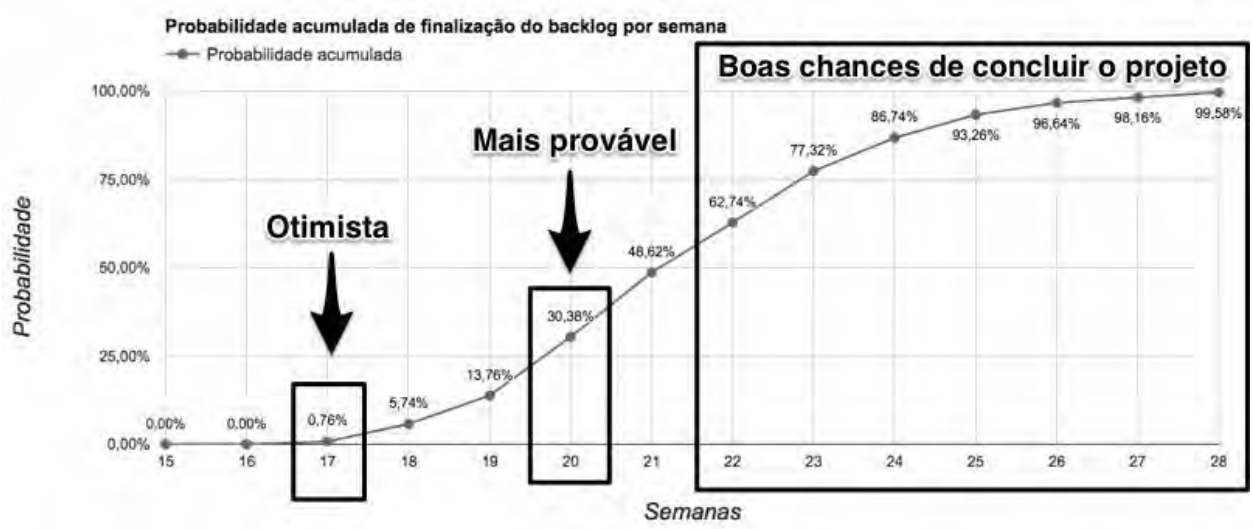


Figura 6.11: Probabilidade acumulada de finalização do escopo por semana da equipe XPTO

Luciana concluiu através desse gráfico que, a partir da semana 22, as chances de entrega do projeto estavam acima de 62% e que, a partir da semana 23, as oportunidades de conclusão saltavam para mais de 77%.

Outra informação útil que Luciana identificou foi de que realmente a projeção do cenário otimista que havia sido passada para o cliente dificilmente seria alcançada (o modelo de Monte Carlo apontava menos de 1% de chance de acontecer). Além disso, o cenário mais provável (semana 20) apresentava menos do que 50% de chances de ocorrer, o que poderia ser considerado um prazo arriscado para se assumir.

Ao aplicar o método de Monte Carlo, Luciana conseguiu ter ainda mais segurança que o projeto seria entregue antes da semana 28 (data esperada pelo cliente).

O pseudocódigo apresentado a seguir representa o método que foi desenvolvido no exemplo da equipe *XPTO*. Caso tenha interesse em baixar um modelo de planilha para realizar as simulações em seu projeto, acesse http://pages.plataformatec.com.br/spreadsheet-forecasting-software-project-completion-date.

```
# backlog_size representa o tamanho do escopo

# número de vezes que a simulação será executada
number_of_trials = 5_000

# variável que armazenará quantas vezes o escopo foi concluído em determinada semana
```

```
completion_occurrences = { }

# variável que armazenará a probabilidade estimada de entrega do escopo em determinada semana
probabilities_of_delivery = { }

for(trial in 1..number_of_trials) do
  # seleciona o valor da última semana da amostra de throughput
  week_index = last_week_observerd_index

  # seleciona o valor da última semana da amostra de throughput
  accumulated_throughtput =
     accumulated_throughtput_from_last_week_observed

  # realiza o loop enquanto o número de itens do escopo for maior do que o número de entregas acumuladas
  while(accumulated_throughtput < backlog_size) do

    # coleta aleatoriamente um valor de throughput da amostra de observações do passado
    throughput = generate_random_throughput_from_empirical_distribution

    # incrementa o valor do throughput acumulado
    accumulated_throughtput = accumulated_throughtput + throughput

    # incrementa o índice da semana
    week_index = week_index + 1
  end

  # atualiza quantas vezes o projeto foi completado com o índice da variável week_index
  completion_occurrences[week_index] = completion_occurrences[week_index] +1
end

# Divide as ocorrências de conclusão em cada semana pelo número de tentativas para calcular a probabilidade de entrega em cada
 semana
probabilities_of_delivery = completion_time_occurences / number_of_trials
```

O método de Monte Carlo é mais uma ferramenta que você pode adicionar ao seu portfólio quando precisar encarar o desafio de projetar cenários para prazos de entrega. Ele oferece como vantagem a possibilidade de escolha do nível de confiança da projeção dos cenários, pois, ao observar as probabilidades, o método gerará informações que permitirão você avaliar qual o nível de risco você estará disposto a assumir quando estiver cravando uma data para a entrega de algo.

Algumas equipes me perguntam se é interessante utilizar dados de projetos anteriores para projetar um projeto que acaba de começar. Minha recomendação é sempre a seguinte: comece a coletar dados do novo projeto e os utilize como parâmetro para esboçar o futuro.

Afinal, você terá pessoas diferentes trabalhando juntas, novas soluções serão criadas e um novo ambiente tecnológico será desbravado.

O que aconteceu em experiências passadas será útil como referência e ativo de lições aprendidas para a sua equipe. No entanto, dificilmente os acontecimentos do passado se repetirão no novo ambiente (projeto). No último tópico deste capítulo, darei algumas dicas de como lidar com a ansiedade de *stakeholders* que demandam definições de prazo.

Como disse anteriormente, antes de pensar em fazer qualquer tipo de projeção de cenários, avalie a estabilidade do seu processo. Ao longo do livro, compartilhei uma série de recomendações para que você possa melhorar a forma na qual sua equipe está criando software, e as métricas apresentadas (*lead time*, *throughput*, WIP etc.) são excelentes recursos no diagnóstico de melhorias. Portanto, projete cenários futuros apenas em ambientes em que há um processo estável.

6.5 Dicas de como utilizar o gráfico de burnup e as projeções no seu dia a dia

Gostaria de fechar este capítulo compartilhando algumas dicas de como o gráfico de *burnup* e os cenários projetados por ele podem ser úteis, independente do papel que você exerça em uma equipe de desenvolvimento de software. Imagine as seguintes situações:

1. Como diretor de uma empresa, você está aflito para saber quando poderá oferecer o novo produto para os seus clientes.

2. Como desenvolvedora, você tem a sensação de que o número de trabalho do projeto no qual você atua não para de crescer e está preocupada com a data de entrega que foi acordada com o cliente.

3. Como fornecedor, você precisa notificar seu cliente sobre como está o progresso do desenvolvimento do projeto no qual está trabalhando.

Acredito que, em menor ou maior intensidade, todos que trabalham em cenários de entrega já tiveram de conviver com os acontecimentos relatados. As dicas que transmitirei a seguir tem por objetivo garantir que você aumente a visibilidade sobre o que está acontecendo com o escopo de uma entrega.

No fundo, espero que, ao aplicá-las, você diminua ruídos de comunicação, traga mais transparência para os *stakeholders* e tenha domínio de uma ferramenta que permita ações que aumentem as chances de sucesso do produto que está sendo criado.

Para compartilhar as sugestões, farei uso de um novo exemplo. Imagine que Cláudio é diretor de tecnologia de uma *startup* que acaba de receber fundos para investir na evolução do seu produto. A equipe tinha o desafio de entregar novas funcionalidades aos clientes em 4 meses.

Ciente da importância do trabalho que viria pela frente, a equipe decidiu reunir-se para uma sessão de entendimento dos objetivos da *startup* com o produto. Ao levantar diferentes hipóteses e desenhar uma solução, a nova versão do produto exigiria o desenvolvimento de 23 novas funcionalidades.

Situação 1: primeira semana do projeto

A primeira pergunta que Cláudio fez para a equipe após o entendimento do escopo foi: quando teremos as novas funcionalidades disponíveis para os usuários?

A dúvida levantada no nosso exemplo é muito comum e gostaria de recomendar algumas técnicas antes que você ou a sua equipe pense em formular respostas diretas.

- **Dica 1:** não caia na tentação de usar dados do passado para prever o cenário de entrega de um novo projeto. O motivo é que novos contextos demandam novos dados. Como temos visto ao longo do livro, coletar *lead time*, *throughput* e WIP não exige grandes esforços e gera um ativo de discussão e ferramental para tomada de decisão riquíssimo. Portanto, caso não tenha dados em mãos, seja transparente com quem solicitou o prazo de entrega dizendo que é preciso monitorar algumas semanas no novo contexto para a geração de projeções futuras.

- **Dica 2:** caso a equipe tenha ciência de que a área de negócios possui uma data de entrega fixada, passe a monitorar e negociar o que será o escopo de entrega e não quando ele será entregue. Costumo dizer para as equipes em que atuo que é mais positivo pensar em "o que será possível entregar até determinada data" do que agir com o pensamento de que "não será possível entregar determinado escopo no prazo fixado". Prazo fixo exige um esforço de priorização tremendo, e por isso recomendo a utilização de técnicas como *MoSCoW* ou matriz de valor *versus* que lhe permitirão ser assertivo nas escolhas do que será parte do escopo de entrega.

- **Dica 3:** ao iniciar um novo projeto ou produto, tenha ciência de que as primeiras semanas de trabalho serão de pouquíssimas entregas de valor, pois: a equipe estará em processo de estruturação; questões relacionadas à configuração de ambiente precisarão ser respondidas; as incertezas de negócio ou técnicas serão altas e a equipe precisará de tempo para desvendá-las; o fluxo de desenvolvimento será posto à prova e poderá sofrer alterações etc.

A técnica de MoSCoW tem por objetivo auxiliar na priorização de requisitos ou funcionalidades. O acrônimo se refere à:

- **MUST HAVE**: são requisitos ou funcionalidades que devem ser desenvolvidos até a data de entrega. São considerados itens essenciais para o lançamento de um produto de software. Se a resposta para a pergunta "o que acontece se não entregarmos essa funcionalidade no final do projeto" for "cancelaremos o lançamento", então a funcionalidade deve ser categorizada como *must have*.

- **SHOULD HAVE**: são requisitos ou funcionalidades que devem ser considerados ao máximo, mas que não impactam no sucesso da entrega. São considerados itens importantes, mas que não são críticos. Pode ser difícil não entregá-los, mas sem eles o produto que será entregue ainda é viável.

Um item *should have* deve ser diferenciado de um *could have* a partir da revisão do nível de dor causada pelo não desenvolvimento em termos de valor para o negócio ou para os usuários afetados, por exemplo. O importante é ter clareza de qual critério está sendo usado.

- **COULD HAVE**: são requisitos ou funcionalidades que são desejados, mas não são necessários para a entrega. Geralmente são pequenos incrementos de baixo custo e que tem menos impacto se forem deixados de lado quando comparados com os itens que foram categorizados como *should have*.

- **WON'T HAVE**: são requisitos ou funcionalidades que não são necessários para a entrega neste momento. É importante deixá-los registrados na lista de requisitos priorizados para evitar que eles sejam mencionados em momentos futuros. Isso ajuda a gerenciar as expectativas de que alguns requisitos simplesmente não serão feitos na solução que está sendo criada neste momento.

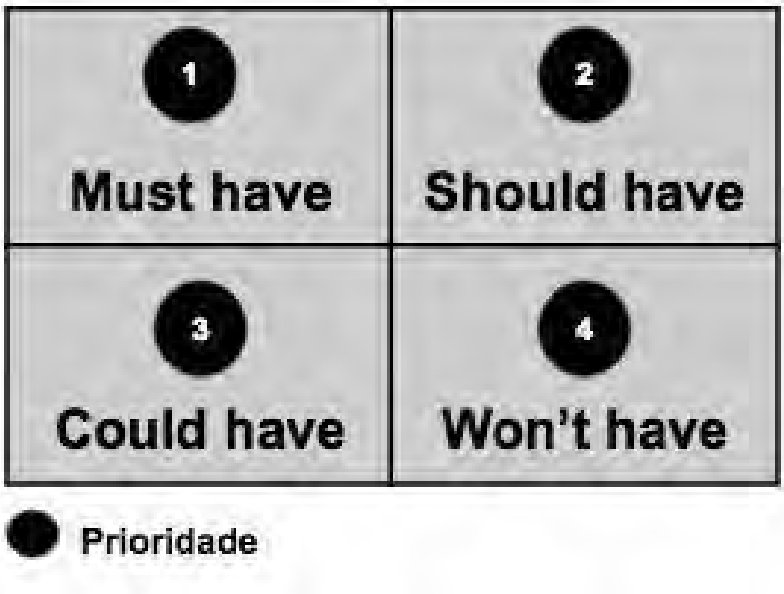


Figura 6.12: Priorização baseada na técnica de MoSCoW

A matriz de valor *versus* esforço confronta o valor de negócio esperado e o esforço que será necessário despender para a criação

de determinada funcionalidade. Evidentemente, ao utilizar tal técnica, você estará em busca das funcionalidades que tiverem o maior valor e o menor esforço. Em contrapartida, funcionalidades com baixo valor e muito esforço deverão ser descartadas.

Ao utilizar a matriz, tenha em mente a clareza do significado de valor e, se possível, utilize critérios quantitativos para medi-lo, como por exemplo: *ROI*, *cost of delay*, número de novos usuários que passarão a usar a plataforma etc.

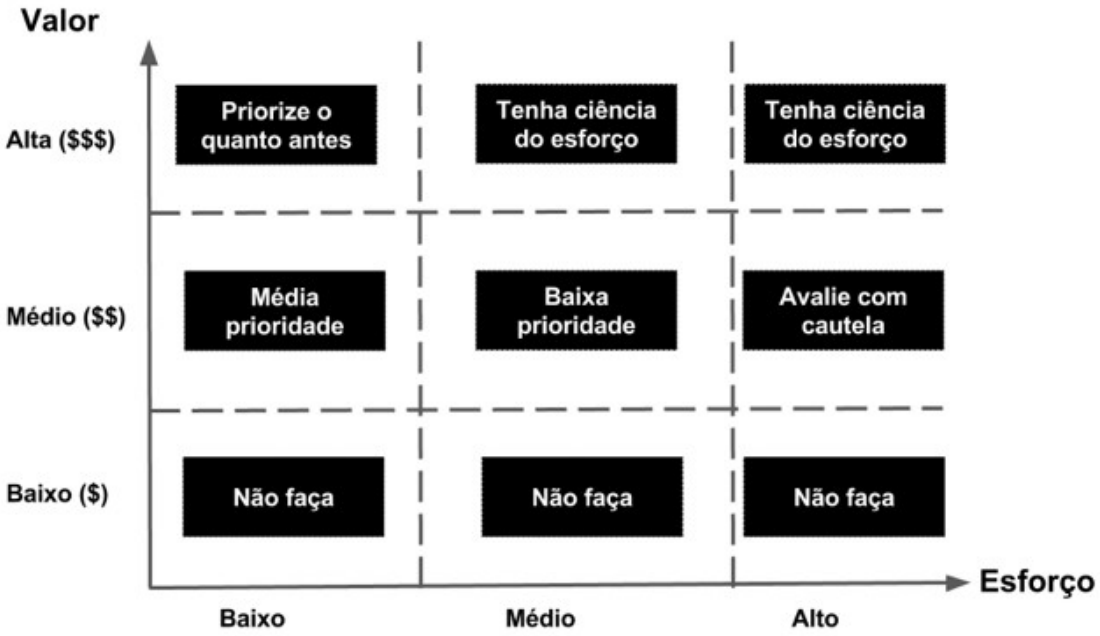


Figura 6.13: Priorização baseada na matriz de valor versus esforço

Situação 2: formas de negociar o escopo

Passadas 9 semanas de desenvolvimento do projeto de evolução do produto, Fernanda, desenvolvedora da equipe, percebeu algo estranho no gráfico de *burnup*. Ao avaliar as projeções, ela se deparou com um cenário no qual, apenas na estimativa otimista, a equipe atingiria o prazo de entrega mesmo sabendo que nenhuma nova funcionalidade seria incluída ao escopo.

Antes que a pressão dos investidores chegasse até a equipe, Fernanda sugeriu que Cláudio negociasse uma redução do escopo atual, pois, só assim, a equipe teria segurança de se comprometer com o prazo de entrega. Ela o alertou de que, se a equipe trabalhasse com o cenário apresentado no gráfico, tudo teria de dar certo para o prazo ser respeitado.

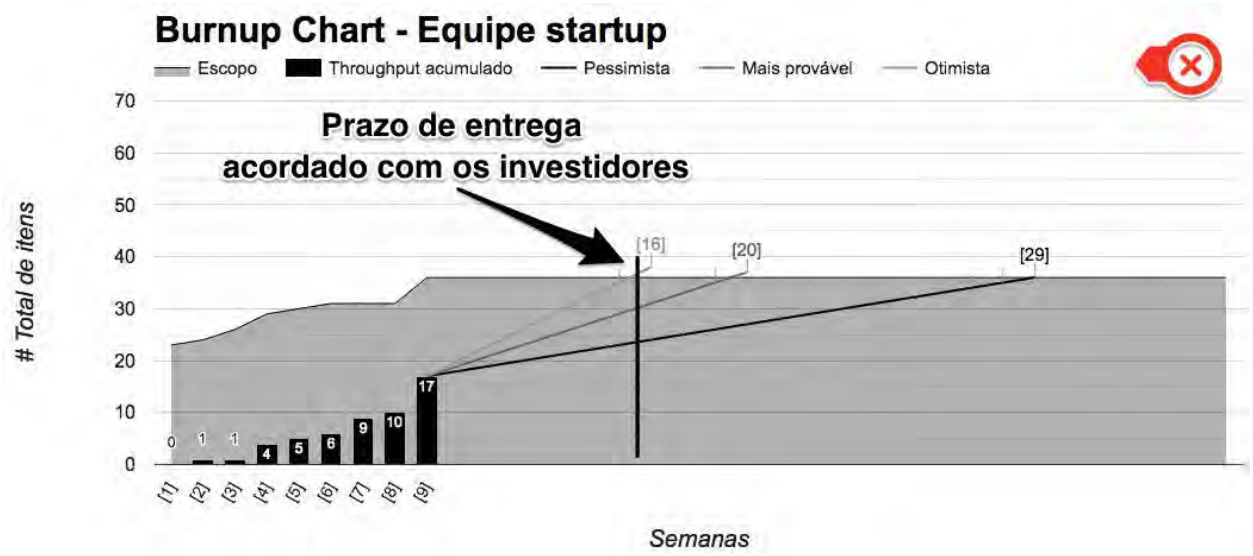


Figura 6.14: Projeção de entrega otimista

Caso você se depare com a necessidade de negociar o escopo, analise em primeiro lugar o que pode estar causando aumentos sucessivos no que será entregue. Crescimento no escopo geralmente é sinal de:

- Questões complexas que foram descobertas em determinado momento do projeto;
- Clientes e usuários que, ao interagirem com partes do produto que estão sendo construídas, começam a gerar novas demandas a partir de novas necessidades percebidas.

O grande desafio neste momento é conseguir garantir um alinhamento de expectativas. Portanto, tenha argumentos em mãos para negar a inclusão de um novo item no montante de trabalho da sua equipe.

Veja, em hipótese alguma estou dizendo que você deva dizer não para qualquer nova necessidade. Traga à tona quais são os benefícios esperados com o projeto como forma de avaliar se o que está sendo produzido e planejado de fato auxiliará no conquista das aspirações de negócio. Ao construir uma funcionalidade de software, você está em busca de melhorar algo, seja ele interno (como otimização de processos internos) ou externo (aumento no *ticket* médio pago pelos clientes, por exemplo).

Uma segunda recomendação que gostaria de fazer diz respeito aos intervalos dos cenários projetados. Caso você precise se comprometer com a entrega de um escopo em que o prazo esperado esteja entre as projeções otimista e mais provável, faça uma análise minuciosa dos problemas que poderão acontecer ao longo do caminho e crie planos de contingência no caso de algo dar errado. Qualquer deslize neste tipo de cenário fará com que a data esperada não seja atingida por falta de margem, logo, é importante garantir que as pessoas interessadas no resultado estejam cientes dos riscos.

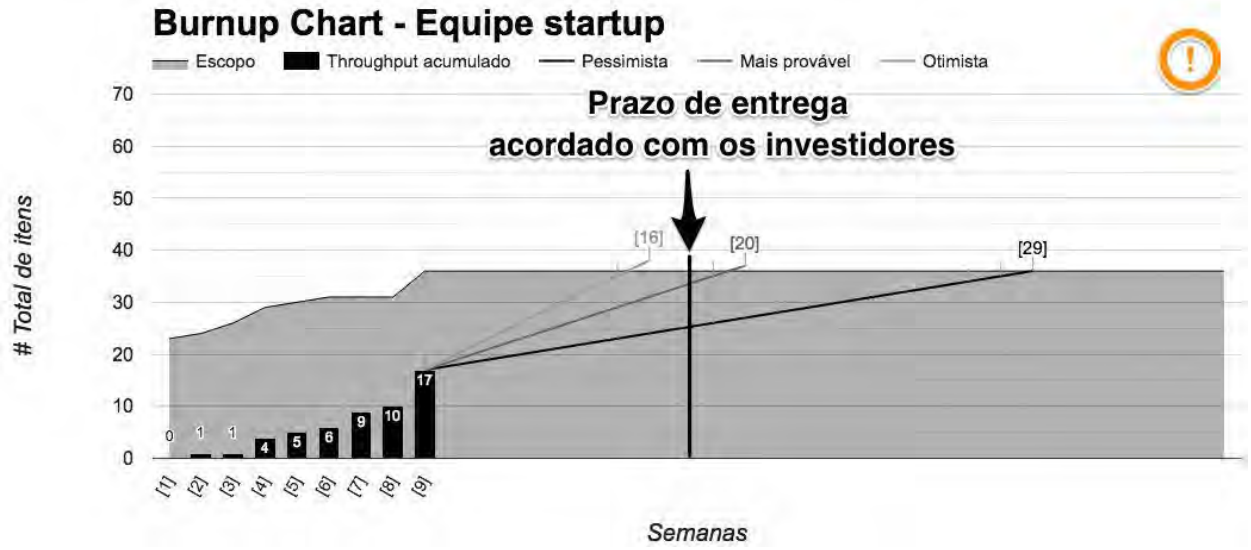


Figura 6.15: Projeção de entrega próxima da data mais provável

Caso você lide com um prazo já fixado, minha sugestão é que você tenha uma data alvo de entrega que esteja entre as projeções mais provável e pessimista. Lembre-se de que este tipo de situação leva em consideração um cenário futuro baseado em informações como a mediana, percentil ou até mesmo a média. Em outras palavras, você estará em um cenário no qual as chances de entrega estarão próximas de uma tendência do passado.

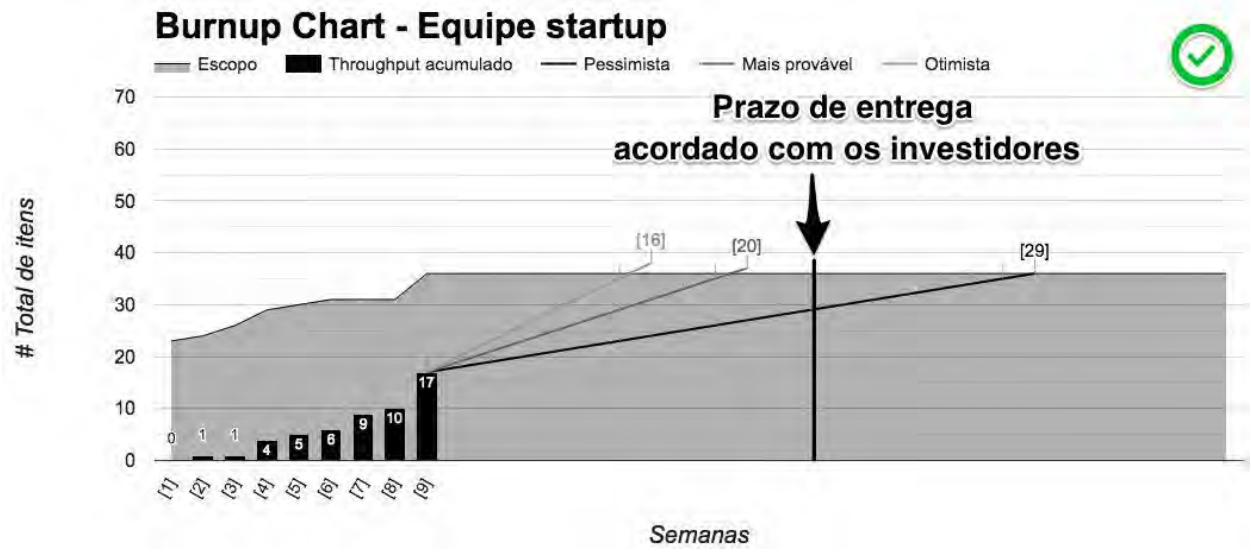


Figura 6.16: Projeção de entrega realista

Ainda falando sobre as projeções de cenários, tenha cautela ao apresentá-las para uma pessoa interessada no produto que está sendo construído. Por vezes, a pessoa pode assumir que as datas apresentadas são um compromisso de entrega, o que na realidade não é. Reforce sempre que possível a mensagem de que as informações apontadas são suposições provenientes do passado e que serão monitoradas periodicamente, a fim de se atingir o objetivo esperado dentro de um prazo factível.

Uma última dica que gostaria de compartilhar quanto à negociação de escopo diz respeito a uma pergunta que, por vezes, é esquecida pelos membros da equipe que é: "quanto o escopo que está sendo construindo apoiará os objetivos de negócio?".

Já tive a oportunidade de trabalhar em algumas iniciativas nas quais a equipe estava tão centrada em entregar determinado escopo dentro de um prazo, que os produtos entregues por vezes não se conectavam com todos as necessidades do negócio. Portanto, caso você e a sua equipe cheguem em um momento onde não será possível entregar um escopo determinado, foquem esforços na criação de soluções menores, mas que estejam alinhadas aos objetivos de negócio. Costumo dizer que é melhor termos soluções simples para provar hipóteses rapidamente do que soluções complexas que não serão úteis para usuário algum.

Situação 3: comunicar o progresso

Após apresentar a necessidade de redução do escopo, Cláudio ficou com a tarefa de comunicar, quinzenalmente, o progresso da entrega para os investidores. O objetivo deste encontro era oferecer visibilidade de quão perto a equipe estava da entrega da nova versão do produto.

Caso você lide com uma situação parecida com a de Cláudio, em que você esteja lidando com gestores de produto, gestores de projeto e diretores, que são pessoas que tendem a precisar de informações diretas e objetivas, minha recomendação é de que seja apresentado, durante um espaço de tempo pré-fixado, o gráfico de *burnup* com uma breve introdução sobre as entregas mais recentes, uma explicação sobre os parâmetros usados para a projeção de entrega, possíveis mudanças que surgiram desde a última apresentação do gráfico e bloqueios que estejam atrapalhando a equipe.

Sempre que precisei me comunicar com pessoas analíticas ou que estavam interessadas em saber qual seria o término de uma entrega, o gráfico de *burnup* foi o meu melhor aliado. Se possível, compartilhe a informação quinzenalmente ou, no máximo, mensalmente. Quanto mais distante for a frequência de compartilhamento, menores serão as possibilidades de as pessoas ajudarem na remoção de bloqueios e maior será a expectativa quanto a evolução da entrega, que quando não concretizada, gera frustração, conflito e pressão desnecessária.

O recado que gostaria de deixar sobre a comunicação do progresso é de que sempre dê visibilidade sobre o que está sendo feito para as pessoas que estão fora do time através de mensagens que façam sentido para elas. Para uma boa comunicação, basta criar uma mensagem com o formato claro, em uma periodicidade que faça sentido e para as pessoas certas.

6.6 Recapitulando

Saber medir o número de entregas realizadas pela equipe por vezes pode não ser suficiente quando encaramos o desafio de projetar cenários de entrega de um projeto ou da evolução de um produto de software.

Neste capítulo, discutimos como o gráfico de *burnup* pode ser útil para compreender tendências de crescimento de escopo, cadência de entrega e para comunicar cenários de entrega a partir de intervalos de tempo.

No que diz respeito às projeções, tivemos a oportunidade de aprender como extrair estatísticas do histórico de *throughput* para sermos mais assertivos quando é necessário gerar cenários futuros. Além disso, vimos como a média pode ser uma referência enviesada por casos extremos positivo e negativo e que, por isso, vale a pena usarmos outras medidas, como a mediana e o percentil para traçarmos o futuro.

Outra técnica apresentada neste capítulo foi o método de Monte Carlo. Ao utilizá-lo, passamos a ter a possibilidade de escolha do nível de confiança dos cenários projetados a partir das probabilidades apresentadas. Em linhas gerais, é como se trouxéssemos para a discussão o quão confiante estamos para entregar um montante de itens e o quão aberto ao risco estamos em não atingir determinado prazo de entrega, dado que quanto menor a probabilidade, menor será a chance de alcançar o prazo projetado.

Fechamos o capítulo aprendendo como utilizar o gráfico de *burnup* em situações comuns ao dia a dia de equipes que atuam no desenvolvimento de software.

A partir de agora, ao lidar com o gráfico *burnup*, você passará a fazer questionamentos como:

- Qual tem sido a taxa de crescimento do escopo?
- O *throughput* da equipe está em um ritmo saudável?
- A partir da data de entrega esperada, como estão as projeções baseadas no histórico de entrega da equipe?
- Se o prazo vem antes da projeção otimista, quais seriam propostas que poderíamos pensar para reduzirmos o escopo?
- Se o prazo está muito próximo do cenário realista, quais seriam os riscos que preciso mapear e que podem atrapalhar o alcance do prazo esperado?

Ao utilizar o gráfico de *burnup*, você terá em mãos uma excelente ferramenta para aumentar a visibilidade sobre o progresso de entrega e que suportará ações necessárias para aumentar a previsibilidade de entrega.

6.7 Referências

CAROLI, P. *Direto ao Ponto:* criando produtos de forma enxuta. Casa do Código, 2015.

EFRON, B.; TIBISHIRANT, R. J. *An introduction to the bootstrap.* New York: John Wiley & Sons, 1993.

MAGENNIS, T. *Forecasting and Simulating Software Development Projects:* Effective Modeling of Kanban & Scrum Projects using Monte-carlo Simulation. FocusedObjective, 2011.

RIES, E. *The Lean Startup.* Crown Trade Group, 2011.

CAPÍTULO 7

E agora, o que fazer?

Neste último capítulo, deixarei algumas sugestões de como você pode fazer melhor proveito das métricas e visualizações compartilhadas ao longo do livro.

Entender determinado problema a partir dos dados é uma ótima forma de incorporar uma filosofia de melhoria contínua, sem mudanças abruptas e caóticas, e de forma transparente para todos que estão envolvidos. Deixe a subjetividade de lado, análise o seu processo de desenvolvimento a partir de fatos (dados), traga as métricas para análise e crie software de qualidade.

As dicas que seguirão o capítulo têm por objetivo fazer com que você reflita sobre questões como: por que devo medir? O que preciso medir? Quando é necessário medir? Como medir?

7.1 Metrificar o processo, e não as pessoas

Todas as discussões abordadas tiveram o intuito de reforçar a importância em se analisar e aplicar mudanças no processo, e não nas pessoas. É importante que você tenha um ambiente no qual as pessoas sejam estimuladas a criar, que tenham autonomia e que estejam engajadas no propósito do ambiente em que elas se encontram.

Se você está interessado em medir quantas linhas de código um desenvolvedor criou na semana ou quantas funcionalidades um testador validou na última iteração, seu foco passa a ser a cobrança de performance, a comparação de desempenho e o microgerenciamento. No fundo, você cria uma cultura na qual há abuso de poder.

Tenha em mente a todo momento que as métricas serão úteis para evoluir o processo, compreender o contexto, gerenciar os riscos, analisar tendências, trabalhar com possibilidades e aumentar a visibilidade.

Ao construir ou analisar uma métrica, se questione: estou avaliando de fato algo que ajudará a evoluir o meu processo, ou estou querendo analisar o desempenho de uma pessoa? Elimine qualquer tipo de controle que seja inútil no seu objetivo de trazer visibilidade e previsibilidade para o fluxo de desenvolvimento da sua equipe.

7.2 Criar métricas como referência, e não cobrança

Em hipótese alguma utilize as métricas como forma de comparar equipes. O motivo é simples: contexto.

Não é porque o *throughput* da equipe A é de 10 funcionalidades por semana e o da equipe B é de 3, que a primeira entrega mais valor para o negócio do que a segunda. Evite cair na tentação de comparar equipes através das métricas. Já cansei de observar equipes que trabalham na mesma empresa e que possuem ritmos de entrega diferentes.

Ao analisar as métricas que foram apresentadas ao longo deste livro, tenha em mente que a sua unidade de análise deve ser apenas e, exclusivamente, a equipe que está sendo observada. Caso contrário, você deixará de levar em consideração o ambiente e as especificidades de cada equipe nas interpretações o contexto.

Por fim, lembre-se: amostras (equipes) distintas exigem análises específicas.

7.3 Criar e monitorar métricas de negócio

Ao longo do livro, falamos sobre métricas que ajudarão você e a sua equipe no aprimoramento da previsibilidade do processo de desenvolvimento. Para que o produto de software que está sendo construído ou evoluído gere resultado, é importante a criação e o monitoramento de métricas de negócio.

Passe tempo o suficiente entendendo o que os usuários do seu produto ou aplicação estão dispostos a utilizar, e estimule a criação de soluções focadas no que de fato importa.

Avalie a saúde financeira do seu produto, entenda quais são as funcionalidades mais utilizadas, monitore quanto as soluções criadas estão convertendo em negócios, converse com os seus usuários, pois assim você estará gerando valor para toda a cadeia envolvida.

7.4 Metrificar é diferente do que burocratizar

Processos são desenhados para eliminar os desperdícios, para otimizar os gargalos e para serem guiados por uma missão. Ao trazer uma forma de tangibilizar os resultados oriundos do seu processo, você está avaliando a saúde com a qual produz suas entregas.

Não caia no discurso de que metrificar é difícil ou que burocratizará o seu processo. Hoje existem softwares, planilhas eletrônicas e até mesmo controles em quadros físicos que permitem o monitoramento de métricas de processo.

Ao deixar de medir, você estará perdendo a chance de analisar sua cadência de entrega, e o quão rápido ou lento tem sido ao entregar demandas para o mercado. E dificilmente, você será assertivo quando tiver de interagir com *stakeholders* externos sedentos por prazos de entrega.

Equipes que incorporam métricas amadurecem seu processo de forma objetiva. Recomendo sempre quando a pauta de uma cerimônia de retrospectiva for melhoria de processo, que a equipe faça uso das métricas. Assim, a equipe terá a oportunidade de discutir objetivamente gargalos, desperdícios, bloqueios e ações para ajustar o seu fluxo de trabalho.

7.5 No estimates tem relação com métricas?

Uma pergunta que você pode ter feito ao longo da leitura do livro é: medir tem relação com o movimento *no estimates*?

Para mim, o termo *no estimates* é uma derivação de melhoria contínua. Não está relacionado ao Scrum ou o Kanban ou o XP, e é um convite para a discussão de alternativas ao processo de estimativa.

Quando uma pessoa me diz que na sua empresa é feito o trabalho de estimativa, eu a questiono: por que você precisa estimar? O que o faz continuar estimando? O que seria da sua empresa se não houvesse mais estimativas?

Não são perguntas simples de serem respondidas, mas elas geralmente revelam algum tipo de problema organizacional, como por exemplo:

- Pensamento orientado a projeto (as equipes e a empresa estão interessadas em concluir o projeto dentro do prazo, custo e escopo estipulado).
- Falta de confiança entre as pessoas.
- Falta de colaboração.
- Falta de visibilidade.
- Sistemas de comunicação lentos.
- Falta de capacidade da equipe em produzir pequenos incrementos de software com qualidade.

Resolver tais impasses não é uma atividade simples. Ao começar a trabalhar em cima deles, provavelmente você estará em uma situação onde estimar será uma opção, afinal, o assunto deixará de ser importante dado o nível de maturidade quanto à colaboração, visibilidade, confiança, integração, comunicação, qualidade de software etc. As pessoas da empresa estarão preocupadas em saber se estão tomando a melhor decisão do que será construído em vez de se preocuparem em predizer quanto tempo será necessário para desenvolver algo.

Em seu livro *No Estimates*, Vasco Duarte cita a diferença entre estimar e projetar. A partir da definição do dicionário Merriam Webster, ele traz que estimar é dar uma ideia geral de valor, tamanho ou custo de algo. Já projetar significa calcular ou predizer um evento futuro usualmente a partir da análise dos dados disponíveis. A grande diferença é que projetar utiliza dados, enquanto estimar não.

O objetivo deste livro não foi abordar técnicas ou a importância de estimar o esforço para desenvolver um produto de software. Todo o ferramental que foi apresentado corrobora com a ideia de usar dados para mostrar o que está sendo construído, avaliar problemas do presente, propor melhorias futuras e projetar cenários de entrega.

Ao aplicar as métricas aprendidas, você diminuirá o esforço despendido nas estimativas e projetará suas opiniões a partir de dados que refletem o seu contexto.

7.6 O que dizem outros autores sobre métricas?

Não poderia deixar de colocar algumas opiniões quanto ao uso das métricas por pessoas ativas na comunidade ágil. Os relatos seguintes são de pessoas que discutem, adotam, analisam e estudam formas de criar melhores softwares.

Alisson Vale — Software Zen — @alissonvale

Por que métricas de processo são importantes no desenvolvimento de software?

Tudo em desenvolvimento de software parece intangível. Não há objetos físicos para se contar; nossos itens de trabalho não têm uma etiqueta de preço atrelada a eles; é difícil fazer a conta de padeiro e saber o retorno obtido do nosso investimento; reconhecemos quando estamos sendo produtivos, mas não conseguimos medir produtividade.

Assumimos, muitas vezes, que vivemos em um mundo esfumaçado, imensurável. Estaríamos tão dominados pelas incertezas que os números, ou seriam obra de ficção, ou só poderiam ser obtidos caso voltássemos a ser escravos de processos pré-definidos, lineares e altamente prescritivos. Não precisa ser assim. Precisamos reduzir nossas incertezas e as métricas estão aí para nos ajudar.

Tomamos decisões o tempo todo. Quando não as tomamos, estamos apoiando de alguma forma quem as está tomando. Quando você reduz a incerteza sobre qualquer aspecto do seu projeto, você otimiza o processo decisório e o torna mais eficaz. Medimos porque queremos reduzir a incerteza sobre o que sabemos a respeito de nossos projetos. Medimos porque queremos nos tornar mais eficazes.

Métricas ajudam a fazer a coisa certa em um mundo repleto de oportunidades e incentivos para você tomar decisões erradas. Mais do que isso, métricas ajudam a validar suas

decisões. Projetadas no tempo, elas dirão se você está melhorando ou piorando determinado aspecto do seu sistema de trabalho.

É melhor trabalhar da forma X ou da forma Y? Experimente cada uma, meça, e avalie os resultados objetivamente.

É por meio de métricas que você vai ser capaz de remover a subjetividade enviesada das decisões que sempre o levam para o terreno da dúvida e do "achismo". Ser capaz de medir um sistema de trabalho, de tangibilizar seus resultados, e de usar essas medições para otimizar a sua performance é, sem dúvida, uma das mais importantes habilidades de qualquer gestor, líder ou agente de mudança na área de software.

Quais análises você indicaria para quem está começando com o assunto e por quê?

É importante começar definindo como você vai acompanhar o seu projeto. E isso tem a ver com a natureza do seu sistema de trabalho e com o que você quer observar. De forma geral, há métricas que são bem comuns e que serão úteis em quase todos os contextos.

O *lead time* é uma excelente forma de avaliar ritmo e previsibilidade. A Eficiência de Fluxo também pode ser muito útil para acompanhar a capacidade da equipe de lidar com delays e filas enquanto o trabalho é processado. Por fim, acompanhar a evolução do volume de trabalho em progresso (WIP) no sistema por meio de um gráfico CFD (*Cumulative Flow Diagram*) é uma ótima maneira de se assegurar que as coisas estão indo bem em termos de fluxo, ou de se antecipar a problemas dessa natureza.

Quais os benefícios de orientar melhorias no processo baseadas em dados?

Há inúmeros benefícios quando a adoção de melhorias em um dado sistema de trabalho é feita por um processo de intervenção suportado por dados. Primeiro, as métricas trazem uma objetividade na própria motivação da intervenção. Isso facilita a comunicação das ações junto às pessoas, justificando o esforço que precisarão fazer. As métricas também nos ajudam a saber, objetivamente, se uma dada mudança na forma de trabalho realmente contribuiu para a melhora que buscávamos.

Na prática, você deve começar o seu processo de melhoria identificando qual problema você quer resolver. A seguir, analise que métricas confirmariam que o problema de fato existe. Feito isso, colete os dados antes de fazer qualquer modificação no sistema.

Apresente os dados, discuta o problema com as pessoas e estabeleça uma hipótese de solução. Aplique e depois repita o processo de medição. Avalie o resultado, corrija suas hipóteses e repita o processo até ficar satisfeito com a melhoria que você buscava. Os Relatórios A3 de Problem Solving são excelentes ferramentas para ajudá-lo nesse processo de investigação, diagnóstico e solução de problemas.

Esse tipo de abordagem o ajuda a ser criterioso. Evita o desperdício de tempo e esforço com tentativas movidas por questões políticas ou emocionais. Traz as pessoas para

contribuírem colaborativamente no processo de construção da solução. E, é claro, permite uma adoção incremental de mudanças, o que é muito mais sustentável do que intervenções bruscas que tentam mudar tudo de uma vez, para que a empresa se alinhe a novos métodos que entram em moda todos os anos.

Quais métricas importam no processo de criação e manutenção de um produto de software (independente de ser de processo)? Por quê?

Desenvolvimento de software se dá de várias formas e em vários ambientes. Você pode ser uma empresa startup; um time em uma área de TI de uma grande empresa; pode prestar serviço de desenvolvimento para clientes; ou ainda trabalhar em uma fábrica de software para o governo. Se for desenvolvimento para terceiros é uma coisa; criação de produtos inovadores é outra; manutenção ou sustentação de software pode ser algo completamente diferente.

Entretanto, seja qual for o contexto, você vai precisar entender que seu projeto ou sua empresa é um sistema e tem de ser analisado como tal. O que você quer é acompanhar se o comportamento do sistema está alinhado com o seu propósito. Assim, pergunte-se "qual é o propósito do seu sistema?" e tente identificar que métricas o ajudam a entender se você está ou não alinhado com ele.

Pode ser que as métricas que mais importem para você sejam as de eficácia de produto (como aquisição, retenção ou conversão); pode ser que sejam métricas voltadas para previsibilidade (*lead time*, *due date performance*); ou para completude de escopo (*burnup/burndown*); ou de valor de negócio, como o *cost of delay*.

Em todas elas, os números vão revelar padrões que apontarão para o comportamento que o sistema apresenta. No final, o foco é na observação do sistema e do seu desvio ou alinhamento em relação ao seu propósito.

Em projetos de manutenção e sustentação de produtos, normalmente o perfil das demandas pode ser mapeado com o risco atrelado a elas ao longo do tempo. Há demandas que você tem que parar tudo para resolver, outras que podem aguardar alguns dias e outras que podem ser planejadas em um cronograma de entrega mais estável.

Isso acontece porque cada demanda tem um risco bastante diferente associado a ela. Quando os riscos são altos ou aumentam linearmente ou exponencialmente ao longo do tempo, usar um SLA com DDP é a melhor estratégia. Já quando o risco se mantém constante ao longo do tempo, o ideal é fazer uma seleção criteriosa das demandas mais importantes e encaixá-las em um roadmap de entregas de releases. Aí vale acompanhar com as métricas já discutidas até aqui (*lead time*, eficiência de fluxo e controle de WIP).

Em resumo, pode-se dizer que há primeiro um desafio em se descobrir o que se quer com as medições, depois o desafio de selecionar as métricas adequadas para aquele objetivo, o desafio de como medir e, por fim, o desafio de interpretar os dados e usá-los para melhorar continuamente os seus resultados.

Celson Martins — Crafters — @celsoavmartins

Por que métricas de processo são importantes no desenvolvimento de software?

As métricas de processo nos ajudam a enxergar o sistema e é natural que, enxergando melhor, melhores serão as intervenções e a análise destas intervenções. Entretanto, um problema escondido é a toxicidade da informação.

Tenho visto que informação limpa é uma raridade nos meios que vivo. A toxicidade não é só de dados, mas também de informação. Às vezes, é como achar agulha em um palheiro.

Quais análises você indicaria para quem está começando com o assunto e por quê?

As que se encaixarem melhor ao sistema a ser analisado. Quando estou considerando métricas para um sistema, principalmente no estágio inicial de análise, costumo optar pela simplicidade. Com o passar do tempo, métricas complexas podem ser derivadas das simples.

Hoje em dia, vejo três pontos fundamentais para análise básica de fluxo: Burnup, CFD e o Custo Médio da Demanda (CMD). O Rodrigo Yoshima desenvolveu um modelo de burnup muito interessante que usa regressão linear para exibir datas e ter alguma previsibilidade. Estamos, em conjunto com a Crafters e a Taller, desenvolvendo um modelo que vai estudar a distribuição destas datas e, com isso, tentar melhorar um pouco a previsibilidade de um sistema complicado.

Quais os benefícios de orientar melhorias no processo baseadas em dados?

Tomando cuidado com a toxicidade da informação, podemos obter informações detalhadas de pontos chave do processo e, com isso, compor a análise de riscos. Perceba que, quando falo de análise de risco, estou me referindo ao processo de análise e entendimento de assimetrias (distribuição de *lead time*), e não das matrizes de risco tradicionais.

Quais métricas importam no processo de criação e manutenção de um produto de software (independente de ser de processo)? Por quê?

Como respondi na outra pergunta, vai depender muito do estágio de análise em que aquele sistema se encontra. Hoje na Taller e na Crafters, temos um conjunto interessante de métricas, mas todas elas evoluíram a partir de métricas bem fundamentais.

O CMD nasceu neste contexto. O Fluxo Unificado nasceu neste contexto de análise e emergência. Guardado os devidos cuidados, também gosto muito das métricas estatísticas para análise do comportamento de uma distribuição e, junto com outras excelentes cabeças, estamos dia e noite trabalhando nestes modelos.

Rodrigo Yoshima — Aspercom — @rodigoy

Por que métricas de processo são importantes no desenvolvimento de software?

O Taiichi Ohno dizia: "Sem padrões não há melhoria". Nos últimos 10 anos, quando o Agile tomou o mercado, vi um distanciamento da comunidade com relação a métricas. Me parece que essa tendência está revertendo, principalmente pela influência do Kanban.

Métricas nos ajudam basicamente a entender o que acontece, agir e observar os resultados. Métricas nos servem de alerta caso alguma tendência mude. Métricas também nos permitem fazer previsões e melhorar nossa tomada de decisões. Fazer análises qualitativas é bom, mas, se for possível usar números, eu acho melhor.

Quais análises você indicaria para quem está começando com o assunto e por quê?

Acredito que o mercado inicialmente precisa entender que é possível ter números, gráficos e estatística para guiar suas decisões, mesmo dentro de sistemas complexos. Um dos grandes insights que emergiu do Kanban — e do trabalho de David J. Anderson — é reconhecer que o trabalho do conhecimento é "orientado a serviços". Logo, qualquer análise voltada a entender seu sistema de trabalho como um Fluxo e não como tarefas, projetos ou "Sprints" é muito saudável para quem está começando.

Quais os benefícios de orientar melhorias no processo baseadas em dados?

São inúmeros os benefícios! Para começar, o efeito para o aprendizado é enorme. Orientando as melhorias do processo com dados, você não só desconfia que suas ações estão corretas — VOCÊ SABE DE FATO!

Práticas mais modernas de gestão de mudança dizem que, para que a transformação ocorra inicialmente, você precisa tornar o problema ou a insatisfação palpável, mensurável e incômoda. Gráficos de tendências são perfeitos para isso.

Duas coisas aprendi nesses últimos anos sobre como influenciar a mudança: i) quem tem dados tem poder; ii) quem controla a narrativa controla o sistema. Esses dois itens juntos são armas poderosíssimas para influenciar um ambiente de trabalho de TI.

Há situações em clientes meus que o simples fato de medir a Carga de Falha e tornar esses dados públicos já acarretou em melhorias de até 40%. Ter dados que comprovem as melhorias também trazem confiança e capital social — isso geralmente se traduz em menos resistência da alta administração e uma mão mais aberta para financiamento de novas melhorias.

Quais métricas importam no processo de criação e manutenção de um produto de software (independente de ser de processo)? Por quê?

"Se você puder medir uma única coisa, meça o Custo do Atraso". Essa frase do Don Reinertsen ecoa na minha cabeça nos últimos 5 anos. A discussão que tem emergido, principalmente na comunidade Kanban, não é a necessidade ou não de usar o Custo do Atraso (Cost of Delay), mas sim a forma de usar.

Seja através de classificação (Classes de Serviço), cálculo financeiro e até qualitativamente (Taxonomias), o Custo do Atraso é determinante para saber se nossas mudanças e decisões de processo são economicamente vantajosas ou não.

Em seguida, creio que as métricas do Fluxo, como o *Cumulative Flow Diagram*, *throughput*, *lead time* e Carga de Falha, sejam importantes. Elas vão habilitar um entendimento sobre priorização, gargalos, previsibilidade e risco imprescindíveis para qualquer organização.

7.7 Metrificar para que mesmo?

Esse é apenas o ponto de partida para compreender o admirável mundo do processo de desenvolvimento de software a partir de métricas. Incorporar uma cultura que traga dados para sua equipe fará com que você monitore um processo que possui uma natureza essencialmente complexa (software), trazendo visibilidade do progresso para quaisquer interessados no que está sendo construído ou mantido.

Além disso, propor melhorias baseadas em dados é um excelente caminho para a remoção de análises subjetivas e, até certo ponto, vazias. No fundo, espero que, com o aprendizado deste livro, você estimule que as pessoas utilizem menos o recurso do *feeling* quando estiverem analisando o comportamento de uma equipe.

Por fim, metrificar é uma ótima forma de buscar previsibilidade quando estamos construindo um produto incerto.

Antes de concluir, gostaria de deixar algumas ressalvas:

- Métricas devem ser usadas para o bem, isto é, para evoluir o processo e não para gerar cobranças e comparações destrutivas.
- Números sem contextos são perigosos, portanto, ao analisá-los, tenha em mente a realidade que está envolta daquela unidade de medida.
- Procure tendências e fuja da precisão. Dada a complexidade que é criar um produto de software, não busque ser determinístico em um mundo que é receptivo por natureza a uma realidade probabilística.

Independente da métrica que estiver utilizando, lembre-se: meça o seu ambiente para ter o que melhorar.

Estou curioso para saber quais métricas de processo ou negócio você está aplicando. Compartilhe sua experiência comigo pelo e-mail raphalbino@gmail.com, ou no Twitter em *@rapha_albino*. Estou sempre interessado em tomar um café e conversar sobre o assunto.

Sumário