QUALIDADE DE SOFTWARE

AULA 5

Prof.ª Maristela Regina Weinfurter Teixeira

CONVERSA INICIAL

Além de utilizarem técnicas e ferramentas, as metodologias ágeis agregam algo muito importante: as métricas que nos auxiliam na construção de software com qualidade. Se nosso ambiente de desenvolvimento ágil é bagunçado e sem métricas de gerenciamento, o modelo não será ágil. O modelo ágil preza pelas métricas para que as atividades e outros assuntos gerenciais possam ser medidos. Se não conseguimos medir algo, não conseguimos gerenciá-lo.

Nesta etapa, falaremos sobre ferramentas e práticas relacionadas à qualidade de software, tais como Sprint Burdown Chart, Velocity, Controle Chart, *Lead Time* e dívida técnica.

Avaliaremos algumas métricas relacionadas ao método Lean Kanban e também a outros métodos ágeis. Mesmo que nosso foco seja mais a nível de Scrum, é importante lembrar que muitas empresas acabam não adotando unicamente uma metodologia ágil, mas tudo o que há de bom e alinhado ao negócio em questão. Encontraremos no mercado times utilizando termos e técnicas do Scrum, XP e Kanban no mesmo projeto. Isso porque já pudemos notar que há várias coisas interessantes em cada uma dessas metodologias até aqui demonstradas.

Muitas vezes, não conseguimos elaborar nossos códigos e outros artefatos exatamente como gostaríamos ou precisaríamos tecnicamente falando. Então, entregamos o que é imprescindível e jogamos para dívida técnica itens que servirão para os momentos de refatoração de nossos códigos. Tempo e orçamento são coisas importantes para o modelo ágil, mas qualidade e um bom gerenciamento, por meio de métricas realistas, é mais do que complementar, é fundamental.

TEMA 1 – MÉTRICAS DE SOFTWARE: DAS CLÁSSICAS ÀS ÁGEIS

A máxima “aquilo que não pode ser medido não pode ser gerenciado” pode ser parafraseada para “aquilo que não pode ser medido não pode ser melhorado”, quando falamos em qualidade de software. Na engenharia de software, essa área ainda precisa de muito foco. As métricas devem ser bem pensadas antes de serem colocadas em uso.

Há muitos exemplos em que as métricas selecionadas foram realmente prejudiciais à transformação da qualidade de uma organização.

O entendimento sobre o que medir e quais resultados estamos buscando é elemento básico sobre quais métricas e em qual estágio da maturidade da qualidade devemos utilizar. Métricas concretas permitirão conversas oportunas e fornecerão dados tangíveis para melhoria contínua e discussões retrospectivas. Por exemplo, se nosso foco inicial é alcançar maior produtividade, podemos começar com as seguintes métricas (Nader-Rezvani, 2018):

- Número real versus planejado de histórias/pontos (incluindo histórias planejadas de dívida técnica);
- Frequência de código quebrado;
- Cobertura de código;
- Tempo médio para reparo/resolução (MTTR).

O *lead time* e o tempo de ciclo costumavam ser as métricas populares para ajudar a aumentar a velocidade em várias seções do processo de entrega de código. Uma métrica importante que ajudará a ser previsível com os clientes é medir o tempo de fluxo. Isso abrange o tempo total que leva desde a primeira solicitação do cliente até a conclusão. Com o tempo, esse tipo de dados pode ajudar as organizações a quantificar a probabilidade de concluir uma determinada porcentagem de trabalho em uma cadência definida. Afinal, alcançar o maior valor para o cliente requer a entrega consistente das melhores soluções.

As principais áreas em que o Agile pode ser eficaz são (Nader-Rezvani, 2018):

- Alcançar uma melhor previsibilidade de lançamento e aumentar a qualidade;
- Alcançar um nível mais alto de produtividade;
- Obter feedback valioso e oportuno dos clientes sobre os valores fornecidos;
- Elevar a satisfação do cliente;
- Pontuação *Net Promoter Survey* (NPS) refletindo maior qualidade do produto.
- Atrair e reter funcionários mais felizes.

As métricas podem ser categorizadas em duas seções diferentes:

- Principais indicadores: dados que fornecem visibilidade sobre a qualidade do software antes mesmo de ser lançado;
- Indicadores de atraso: relacionam-se com os dados que estão disponíveis após a implantação do software.

À medida que a maturidade é alcançada nas organizações, o monitoramento de indicadores de liderança e de atraso permitirá que as equipes considerem os fatores com maior impacto na qualidade, realizem análises de causa raiz e de volume e criem histórias para abordar esses itens como parte da melhoria contínua da qualidade jornada.

As métricas têm um conjunto de medidas que precisam ser observadas periodicamente para que o estado e a qualidade do processo sejam julgados. Essas avaliações requerem:

- Definição de métricas/medidas apropriadas a serem usadas, como os atributos de qualidade serão avaliados;
- Definição de um conjunto de metas de medição esperadas, podem ser limites simples ou cálculos muito complicados com base em várias medições realizadas;
- Um processo de coleta de dados implementado para realizar periodicamente as medições necessárias, incluindo ferramentas de coleta de dados adequadas identificadas e implantadas no projeto;
- Análise de dados realizada em momentos adequados e julgamentos feitos sobre qualidade;
- Armazenamento, reutilização e comparação de dados para determinar os níveis atuais de qualidade, conformidade, tendências e previsões futuras;
- Os dados precisam ser adequadamente protegidos, incluindo dados relacionados a pessoas, finanças e requisitos e/ou medições confidenciais.

Uma ampla gama de modelos foi desenvolvida para especificar e capturar métricas de qualidade de software. Analisamos brevemente vários aqui, com vistas a métricas mais recentes para plataformas baseadas em nuvem, métodos ágeis e sistemas intensivos de software em grande escala.

Muitas métricas de nível de código foram desenvolvidas. Os clássicos incluem linhas de código (geralmente uma medida de produtividade ruim), complexidade ciclométrica, análise de pontos de função, coesão, acoplamento e vários tipos de análise de complexidade e tamanho. Embora historicamente aplicados ao código-fonte, muitos também podem ser aplicados a modelos de nível de design, especialmente quando usados para atividades de engenharia orientadas a modelos e até mesmo potencialmente a modelos de configuração. Uma variedade de métricas para interface de usuário de sistemas também foi desenvolvida, muitas derivadas de Interação Humano-Computador (IHC) e pesquisa e prática de usabilidade. Esses têm sido aplicados a interfaces web, mais recentemente a interfaces móveis, e estão cada vez mais sendo aplicados a interfaces ubíquas, hápticas, realidade virtual, toque, gesto, fala e outras interfaces mais centradas no ser humano. Essas métricas incluem taxas de conclusão simples, tempo da tarefa, satisfação do usuário, taxas de erro, várias medidas de interação e métricas de estilo de marketing, como taxas de retorno e taxas de conversão.

O teste de software tem sido historicamente usado como um importante mecanismo de garantia de qualidade no desenvolvimento de software. Muitas métricas foram desenvolvidas para dar suporte à garantia de qualidade por meio de testes. Esses incluem defeitos/bugs por linhas de código, cobertura de código de teste, localização de falhas e identificação de criticidade de defeitos localizados.

Métricas em nível de processo são usadas para qualificar e quantificar aspectos de qualidade associados ao processo de desenvolvimento de software empregado por uma equipe. Exemplos incluem gráficos de *burn-down* comumente usados em métodos ágeis para rastrear o progresso, taxas de conclusão de tarefas, caminhos críticos e horas (e outros recursos) gastos em atividades de desenvolvimento e garantia.

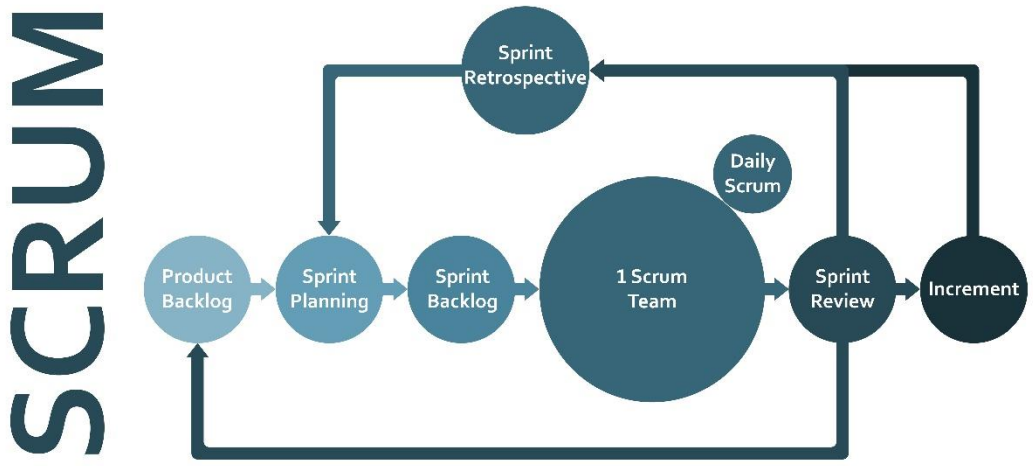
Sistemas orientados a serviços e baseados em nuvem trouxeram novas demandas para a avaliação do desempenho em tempo de execução de sistemas de software com vistas ao atendimento de atributos de qualidade nessa área. Tais métricas incluem as tradicionais, como disponibilidade do serviço, duração da interrupção, tempo médio entre falhas, tempo de conclusão e tempo de resposta para solicitações.

Medidas mais recentes são necessárias para avaliar a qualidade do serviço e a entrega de aplicativos em nuvem, incluindo capacidade de rede e dispositivo de armazenamento, capacidade do servidor (em termos de poder de computação), capacidade do servidor web (número de solicitações simultâneas, usuários suportáveis etc.), instância *start up*/*shut down* para medição da elasticidade da nuvem, tempo médio para transição e tempo médio para recuperação do sistema após falha.

TEMA 2 – SPRINT BURDOWN CHART

As metodologias ágeis utilizam-se de ferramentas para um conjunto de processos e de artefatos. Um desses artefatos é chamado Sprint Burdown Chart. Foi adaptado dentro das metodologias ágeis para demonstrar aos times de desenvolvimento o progresso dentro de cada Sprint. Esse tipo de gráfico não é exclusividade de frameworks ágeis. O fluxo de processos do Scrum pode ser visualizado no gráfico 1. Tudo parte do *Product Backlog*, que é utilizado dentro da *Sprint Planning* e que refina os cards que passam para a *Sprint Backlog*. O time, então, trabalha sobre os cards da *Sprint Backlog* acompanhados na Daily Scrum. Ao término da Sprint, há uma revisão que pode ou não sofrer algumas melhorias (*Increment*) e, finalmente, ocorre a Sprint Retrospective. Essa última serve de subsídio para auxiliar na *Sprint Planning*, bem como em mudanças no próprio *Product Backlog*.

Figura 1 – *Scrum Framework*



Crédito: KostianynL/Shutterstock.

Revisando aspectos das principais cerimônias da Metodologia Scrum, podemos observar o momento de ação sobre o gráfico Burdown:

- ***Sprint Planning*:** atividade inicial de qualquer Sprint. Durante a planning, todos os objetivos desse novo ciclo são estabelecidos, e o *Sprint Backlog* é criado, bem como o gráfico *Burdown* é inicializado. Nessa reunião são feitas todas as negociações do time e definido o objetivo da Sprint. Uma vez tudo definido e aceito pelo time, o PO garante que a equipe mantenha suas atividades alinhadas aos itens do *Sprint Backlog*. Os três papéis listados na Figura 1 participam da *planning*. O PO aponta o *backlog* do produto e a equipe desenha em conjunto o backlog da Sprint. O Scrum master media as discussões. Uma vez todos de acordo, a Sprint é iniciada;
- ***Sprint Review***: cerimônia que determina o encerramento da Sprint e o planejamento da mesma, que não deve durar mais do que duas horas. Nesse momento, a equipe apresenta o que foi desenvolvido, mostra novos recursos funcionando e demais artefatos. Aqui, vale ressaltar que como se trata de entrega de software, ele deve estar funcionando. Não se pode considerar apenas apresentações estáticas do que seria o produto. Além do time envolvido na Sprint, é bem comum que outras partes interessadas apareçam nessa reunião para celebrar as entregas, visto que geralmente estão ansiosas para ver os resultados, ou seja, novas *features* implementadas;
- ***Sprint Retrospective***: a retrospectiva, ou simplesmente chamada retro, dura de 15 a 30 minutos, e o time discute o que está funcionando e o que não está funcionando na dinâmica das atividades. Ela é realizada na sequência da *Sprint Review*, e todos devem participar, inclusive as partes interessadas (clientes, CEO etc.). O Scrum master é responsável por essa atividade e questiona o grupo da seguinte forma:
    - O que gostaríamos de fazer na sequência?
    - O que gostaríamos de parar de fazer?
    - O que vamos continuar fazendo na próxima Sprint?
- **Daily Scrum Meeting:** diariamente o time reúne-se para uma reunião que deve durar no máximo 15 minutos. Quando presencial, a *daily* é feita com os participantes de pé (*Stand Up Meeting*). Assim, garante-se que a

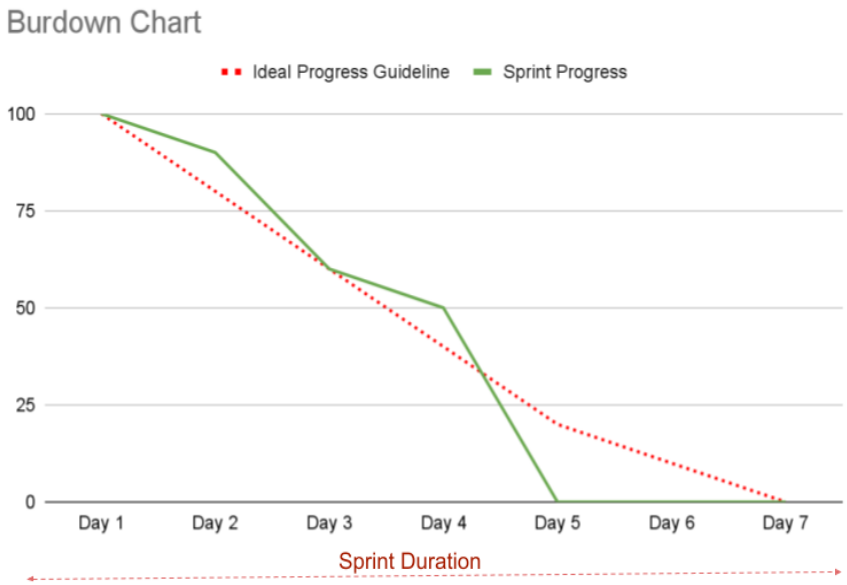
mesma não leve mais de 15 minutos. O Scrum master facilita a *Daily* e questiona o time de forma simples:

- O que fizeram ontem?
- O que vão fazer hoje?
- O que está bloqueado?

Esse tipo de gráfico demonstra como o trabalho do time está progredindo, mantendo a equipe coesa em relação ao que está ocorrendo com as atividades e com cada envolvido. Também ajuda na observação do progresso que pode estar muito lento em relação ao planejado. Ele é relativamente simples de ser interpretado e sua simplicidade o torna uma ferramenta excelente para o rastreamento do progresso das atividades. O gráfico marca no eixo horizontal os dias da Sprint e no eixo vertical os pontos do planejamento que compõem a Sprint, sempre indo do máximo de pontos da Sprint, que corresponde à velocidade da equipe até zerar.

A primeira linha do gráfico é traçada por uma diagonal que vai do ponto máximo ao ponto zero do eixo vertical. A linha pontilhada vermelha do gráfico da Figura 2 representa essa diagonal.

Figura 2 – *Burndown Chart*



Crédito: Maristela Regina Weinfurter Teixeira/Google Sheet.

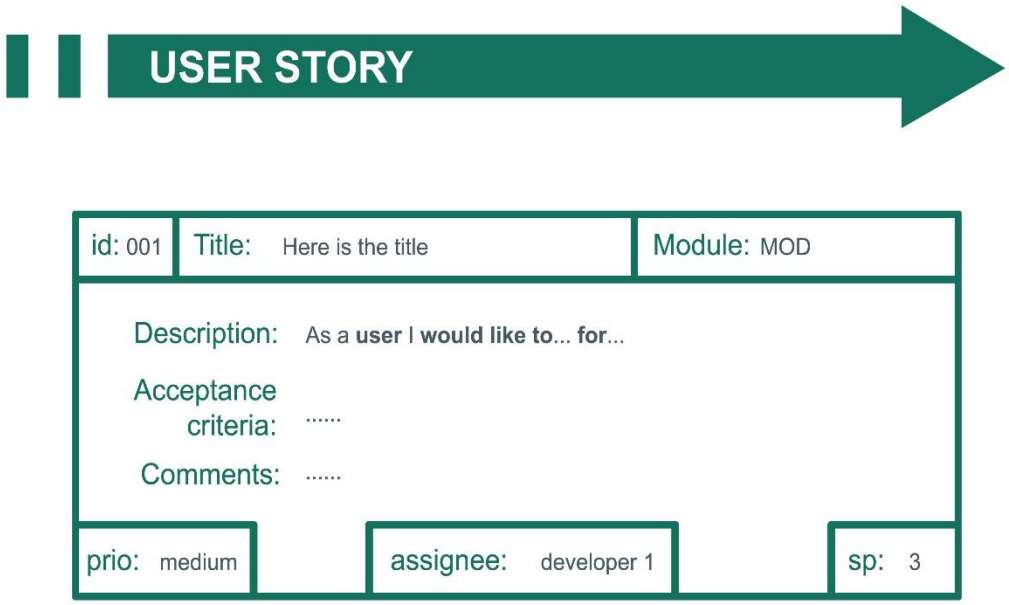
Ao término do primeiro dia de trabalho, após a daily, o time deve calcular o total de pontos das *User Stories* (US) – representadas na Figura 3 – concluídas

para que a linha azul (Figura 2) comece sua descida. Esses pontos são o somatório das US finalizadas.

Porém, como toda ferramenta, ela tem algumas limitações. Por exemplo, os gráficos geralmente focam um pequeno pedaço do contexto total. Ou seja, se houver alguma alteração no escopo do projeto, nós não teremos clareza e indicação das mudanças no nosso escopo mais específico.

Eles demonstram apenas as horas de trabalho e não se as US foram completadas ou ainda estão em andamento. Sua utilidade está relacionada com a exatidão das estimativas, porém estimativas nem sempre são bem elaboradas. Outro problema é que o gráfico não demonstra os motivos do aumento de trabalho restante, que pode vir por conta do aumento da estimativa ou remoção de trabalhos concluídos.

Figura 3 – Exemplo de como descrever uma US



Crédito: Amaial/Shutterstock.

Podemos tornar o uso do gráfico *Burdown* mais prático com as seguintes dicas:

1. Escrever atividades pequenas no *Sprint Backlog*. Refine o quanto puder para que ela caiba dentro da Sprint e seja útil;
2. Ao escrever suas atividades, pense com clareza em todos os itens do *Product Backlog* (para o PO). Qualidade não deve ser só compromisso do time de SQA. Com isso, escreve de forma clara, objetiva e que realmente entregue valor;

3. Objetivos simples e curtos garantem um bom planejamento diário, gestão do *Sprint backlog* e gestão de riscos;
4. Não parar de questionar e sanar todas as dúvidas para que a US seja desenvolvida com clareza nos objetivos.

O *Burndown Chart*, se aplicado adequadamente, é uma excelente ferramenta para auxiliar todo o time de desenvolvimento. Ele ajuda a promover reflexões sobre o processo, sobre a capacidade do time, objetivos diários, além de absorver demandas não planejadas.

TEMA 3 – MEDINDO A PRODUTIVIDADE

A engenharia de software propõe três dimensões importantes para medição de produtividade no desenvolvimento de software:

- **Velocidade**: a rapidez com que o trabalho é feito;
- **Qualidade**: quão bem o trabalho é feito;
- **Satisfação**: quão satisfatório é o trabalho.

Ao tentar definir metas de produtividade ou medir a produtividade, é importante considerar todas essas três dimensões, porque elas trabalham juntas de forma sinérgica. Embora a produtividade seja muitas vezes considerada em termos de aumento da produção (maior velocidade), um aumento na velocidade pode não corresponder a uma melhoria real da produtividade se houver uma queda correspondente na qualidade dessa produção. A velocidade e a qualidade juntas compõem a eficiência e a eficácia gerais do trabalho, enquanto a velocidade e a qualidade podem afetar a satisfação de diferentes maneiras. Um aumento na velocidade pode levar à redução de custos (e melhorar a satisfação dos gerentes), mas ao mesmo tempo pode levar ao aumento do estresse para os desenvolvedores (reduzindo sua satisfação e, por sua vez, incorrendo em custos futuros) (Sadowski, 2019).

A dimensão de **velocidade** captura como a produtividade é frequentemente conceituada em termos de tempo gasto em uma tarefa ou o tempo gasto (ou custo) para atingir uma determinada quantidade de trabalho. Como se pode conceituar ou medir, a velocidade é altamente dependente da tarefa, e o tipo de tarefa precisa ser considerado, bem como a granularidade, a complexidade e a rotina de uma tarefa específica. Por exemplo, as métricas de velocidade (*Velocity Metrics*) do desenvolvedor podem incluir o número de *story*

points por Sprint ou o tempo necessário para o código ir do início até a implantação.

A dimensão **qualidade** engloba o bom trabalho na produção de artefatos e software ou a qualidade dos serviços prestados. A qualidade pode ser uma consideração interna em um projeto ou externa a um projeto.

As métricas de qualidade em um projeto de software podem incluir contagens de características negativas, como defeitos pós-lançamento ou classificações autorrelatadas de atrasos devido à dívida técnica.

A **satisfação** da engenharia é um conceito multifacetado, o que torna difícil entender, prever ou medir. Essa dimensão captura fatores humanos de produtividade e tem vários subcomponentes possíveis, incluindo fatores fisiológicos como fadiga, medidas de conforto da equipe, como segurança psicológica e sentimentos individuais de fluxo/foco, autonomia ou felicidade. O aprendizado ou o desenvolvimento de habilidades que podem impactar positivamente a qualidade a longo prazo, a retenção do desenvolvedor ou a velocidade podem se manifestar como um aumento na satisfação. Para os desenvolvedores, a satisfação pode ser impactada pela eficácia real ou percebida de seu trabalho pessoal ou do trabalho de sua equipe.

Essas três dimensões da produtividade podem ser vistas de diferentes ângulos. Podem ajudar a restringir um objetivo de pesquisa e fornecer uma perspectiva sobre os métodos subsequentes que podemos usar para entender ou medir a produtividade. Os vários ângulos de visões das dimensões são (Sadowski, 2019):

- **Partes interessadas**: diferentes partes interessadas (stakeholders) podem ter objetivos e interpretações variadas de qualquer tipo de medição de produtividade. Antes de tentar entender e medir a produtividade, é essencial identificar quais stakeholders são de interesse e o que é importante para esses stakeholders;
- **Contexto**: projetos específicos, fatores sociais e culturais mudarão as percepções de produtividade. Por exemplo, se os desenvolvedores sentirem que ajudar os outros é valorizado por sua equipe, eles sentirão que o tempo gasto respondendo a perguntas é produtivo. O contexto de desenvolvimento subjacente (projetos de código aberto versus projetos focados em lucros) afeta as metas de produtividade;

- **Nível**: cada nível representa uma escala específica na qual a produtividade é considerada. Desenvolvedores individuais, equipes, organizações e a comunidade ao redor levarão a diferentes percepções de produtividade, e as metas de produtividade também podem estar em conflito entre esses diferentes grupos. Uma intervenção que pode beneficiar um nível pode não ser válida em todos os níveis;
- **Período de tempo**: as percepções de produtividade variam muito de acordo com o período de tempo considerado (prazos mais curtos, como dias, semanas ou Sprints, ou prazos mais longos, como meses ou anos). Por exemplo, uma mudança de processo pode diminuir a velocidade no curto prazo, mas levar a um aprendizado aprimorado da equipe ao longo do tempo e, assim, acelerar a velocidade por um período de tempo mais longo. Da mesma forma, os aprimoramentos de velocidade de curto prazo podem levar à fadiga e diminuir a satisfação do desenvolvedor por um longo período de tempo.

3.1 Velocity Metric

Velocity é uma métrica utilizada por times de desenvolvimento de software que utilizam o método Scrum. O Scrum Master é responsável pela mensuração do projeto e um dos grandes desafios é justamente mensurar atividades de desenvolvimento de software. Uma medição simples e comum é a soma do número de horas, porém alguns acreditam que é uma forma falha. Foi muito utilizada em metodologias precursoras às metodologias ágeis.

Nesse caso, velocidade não é aceleração, mas pode ser uma medida de capacidade de quanto a equipe é estável e consegue trabalhar junta. A velocidade de uma equipe (independentemente de usar pontos ou contagens de stories) varia conforme ela aprende a trabalhar em conjunto. A velocidade de uma equipe também pode variar em outras circunstâncias (Rothman, 2017):

- A equipe está aprendendo um novo domínio;
- As histórias variam em tamanho e podem ser bastante grandes;
- A equipe alterou a duração da iteração.

Existem duas desvantagens potenciais para medir a velocidade. Dissemos que a velocidade é a taxa de variação. É isso que entregamos

(recursos ou pontos de stories) ao longo do tempo. Se alterarmos a duração da iteração, alteraremos o tempo, que é uma das entradas para a taxa de mudança.

Ao entregarmos pontos de story e diminuirmos ou aumentarmos o tamanho das histórias, alteraremos uma das entradas para a taxa de mudança. Se mantivermos a mesma duração de iteração e usarmos recursos em vez de pontos, as medições de velocidade serão mais precisas.

Outra forma de medição é feita por meio de **Story Points**, que têm origem nas US. A métrica utiliza a sequência de Fibonacci em conjunto com questões, como:

- O projeto é parecido com algum anterior?
- Como foi o esforço para executá-lo?
- Existe algum bloqueio que possa causar atrasos?
- Dependemos de outras áreas ou fornecedores que podem afetar a conclusão de nossas atividades?

Uma técnica utilizada para o planejamento das estimativas é o *Planning Poker*, originado na metodologia Scrum. Cada membro do time recebe uma pilha de cartas que *Story Points* diferentes. Uma pessoa faz a leitura dos projetos que a equipe tem definido. Em seguida, cada integrante do time exibe uma carta que acredita que represente o esforço para concluir determinado projeto. Pontos importantes que devem ser levados em consideração:

1. *Story points* não são horas;
2. Média não é um bom resultado;
3. Ele não funciona muito bem com atividades pequenas ou enormes;
4. Manter os pontos sempre atualizados é muito importante.

A **t-shirt size**, que significa tamanho de camiseta, tem base nas medidas P, M, G e para cada item a ser mensurado, como no caso de camisetas para uma pessoa. A ***issue count***, a qual conta o número de itens, é bastante utilizada no framework Kanban e sugere-se que seja utilizada com times mais experientes. O Scrum Master escolhe as técnicas conforme fiquem mais aderentes ao projeto e ao time.

Para que a medição funcione, a grande questão está em sermos coerentes com nossas escolhas e não mudarmos constantemente. E como já comentamos, queremos conhecer o ritmo do nosso time, em qual velocidade ele trabalha e consegue fazer as entregas com qualidade. De nada nos vale que

haja uma entrega rápida se a mesma é parcialmente ou nada testada, se ela realmente entrega aquilo que estava proposto no *backlog*.

A experiência é a parte importante dentro de um time para a adoção da métrica, é importante que se passem duas a três Sprints para que, então, consigamos ter uma ideia se as métricas estão fazendo sentido.

Quando falamos em velocidade do time, devemos levar em consideração não somente as habilidades individuais de nossos devs, mas todo o contexto do projeto, a cultura da empresa e a coesão do time.

Em algumas ferramentas de gerenciamento de projetos ágeis, como no caso do Jira, a velocidade pode ser calculada automaticamente no relatório de Gráfico de Velocidade, desde que seja parametrizado adequadamente. Ele pode ser configurado para utilizar *Story Points*, Horas ou *Issue Count*. O importante é que seja adequadamente configurado pelo Scrum Master ou PO.

O que podemos fazer se nossos cálculos demonstrarem que nossa velocidade não está bem? O importante não é que saiamos tirando conclusões precipitadas, mas que possamos refletir com o time, tentando levantar o que está causando o problema de velocidade em nossas entregas.

TEMA 4 – *LEAD TIME* & *CYCLE TIME* PARA MEDIR O TRABALHO

Nosso propósito aqui é estimar da forma mais correta possível nossas atividades dentro de um prazo razoável. Às vezes, queremos saber se nossas estimativas realmente estão corretas, pois parece-nos que demoramos uma eternidade para concluí-las. Para isso, mediremos o tempo que levamos para desenvolver algo de duas maneiras: tempo de ciclo (*Cycle Time*) e tempo de espera (*Lead Time*).

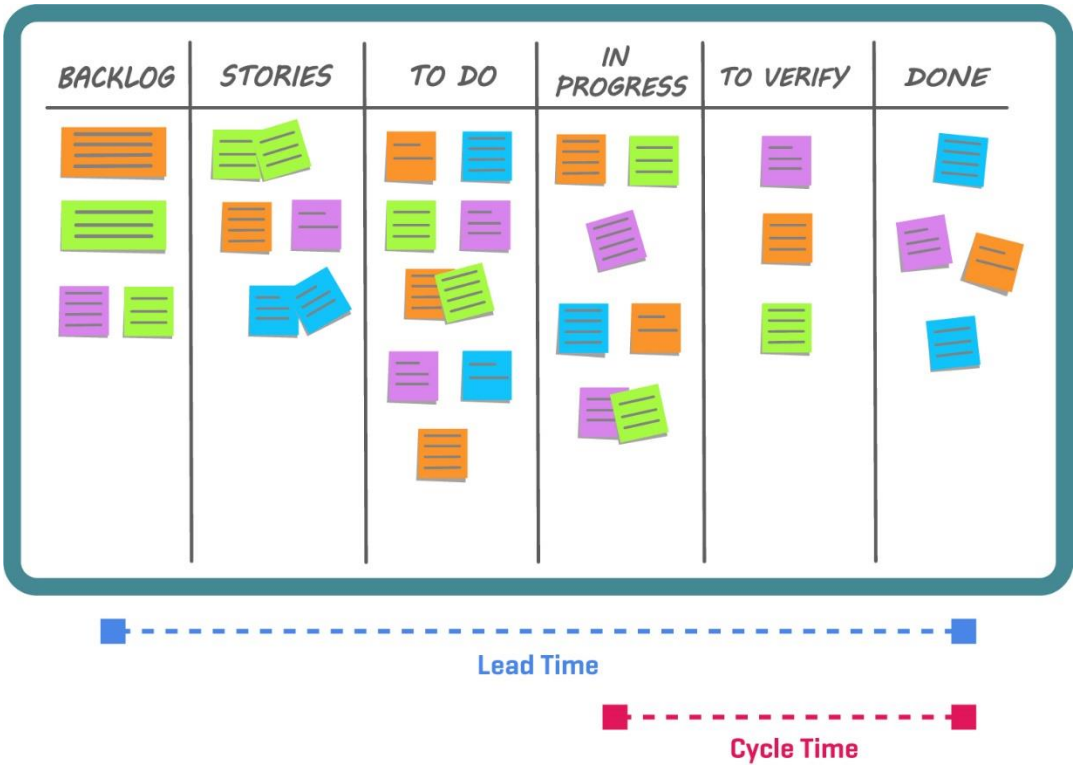
O tempo de ciclo tem por duração desde o momento em que o card é colocado na coluna “em trabalho” até o momento da entrega. Essa métrica corresponde ao tempo que o time trabalha dentro de cada Story.

Enquanto isso, o tempo de espera corresponde à duração de tempo desde quando um card vai para a “lista de pendências” até liberarmos o recurso para o cliente. O *Lead Time* é todo o tempo desde o *backlog* até o uso do cliente. Se você tiver uma área de preparação antes de mover seu produto para produção, seu *lead time* será pelo menos o tempo da equipe mais esse tempo (Rothman, 2017).

A Figura 4 ilustra a diferença entre o tempo de ciclo e o tempo de espera. Aqui está o que aconteceu quando uma equipe mediu seu tempo de ciclo como feedback para suas estimativas.

A equipe adicionou um dado: o dia e a hora em que o cartão passou de uma coluna para a próxima à medida que avançava no fluxo.

Figura 4 – Representação de *Lead Time* e *Cycle Time*



Crédito: Viktoriia Ablohina/Shutterstock.

4.1 Método Ágil Kanban

O método ágil Kanban tem o objetivo de trabalhar com projetos que catalisem resultados enxutos dentro das empresas. O que queremos ao adotar uma metodologia Kanban é que nossos sistemas complexos se adaptem à cultura Kaizen. Kaizen é focada em melhoria contínua para os resultados econômicos, sociológicos e dos negócios, ou seja, usa cinco propriedades como condições para o estímulo do comportamento enxuto:

1. Visualização do fluxo de trabalho;
2. Limite do WIP;
3. **Métricas e gerenciamento para o fluxo de trabalho**;
4. Adoção de políticas de processo explícitas;

5. Adoção de modelos para **reconhecimento de oportunidades de melhoria**.

A metodologia ágil Kanban tem foco na qualidade, na redução dos trabalhos em progresso, nas entregas frequentes, no equilíbrio entre demanda e receita, gestão de prioridades e fontes de variabilidade para melhoria e previsibilidade.

Falando rapidamente sobre o formato do Kanban, temos:

1. **Delivery Planning:** reunião para planejamento das entregas das funcionalidades do software;
2. **Kanban Delivery:** (*downstream*) conversão das opções e execução do backlog;
3. **Kanban Meeting:** reunião curta e rápida, de preferência em pé, como no Scrum, que tem por finalidade tomar decisões para que o trabalho flua;
4. **Kanban Discovery:** (*upstream*) é a discussão das opções de priorização das atividades para o time e para as entregas.

4.2 Métricas Lean

As métricas são ferramentas importantes e devem ser levadas a sério pelas equipes de desenvolvimento. As Métricas Lean precisam da colaboração de todo o time de desenvolvimento para que se meça e quantifique sua produtividade. Existem muitas métricas enxutas que podem ser usadas para entender um processo. As métricas comumente usadas incluem (Iswanto, 2020):

- Tempo de processo (P/T) é o tempo usado para executar um processo;
- Tempo de espera (W/T) é o tempo usado para esperar antes do início do processo;
- Lead Time (L/T) é o processo total e consiste em tempo de processo e tempo de espera;
- *Work in Progress* (WIP) é o número de itens na lista de espera de trabalho;
- Porcentagem completa e precisa (% C/A) é a porcentagem de tempo de uma entrada recebida com informações completas e precisas;
- O tempo de mudança (C/O) é o tempo necessário para mudar de uma atividade para outra, também é chamado de tempo de configuração;
- *Takt time* é a razão entre o tempo de trabalho líquido disponível para a demanda do cliente. *Takt* vem do alemão e significa *ritmo*. *Takt* descreve

as etapas de produção necessárias para atender às demandas dos clientes. Essa métrica é importante para entender a rapidez com que o processo deve ser executado para poder atender às demandas.

Vamos falar um pouco sobre três métricas mais conhecidas do Lean Kanban:

1. *Lead Time* (Tempo de Espera);
2. WIP (*Work in Progress);*
3. *Throughput* (Vazão de atividades entregues).

Essas três métricas serão discutidas separadamente ao longo deste tópico.

4.3 WIP (*Working In Progress*)

O WIP compreende atividades de desenvolvimento que estão em execução ou parcialmente acabadas e ficam em uma fila de espera de concluídas para implantação (*deploy*). Práticas Lean e metodologias ágeis sempre comentam que o ideal é limitarmos o trabalho em andamento, porque serve como um inventário que agrega valor ao cliente e pode se transformar em desperdício caso não seja mais necessário no futuro.

Então, aquilo que chamamos de sistema empurrada passa a ser chamado de puxado, no qual não é mais possível iniciarmos atividades antes de finalizarmos outra em andamento. Por exemplo, no WIP podemos evitar desperdício trocando dez atividades em andamento com 50% de conclusão, porém nenhuma concluída, por cinco atividades completas e cinco que vão iniciar.

4.4 *Lead & Cycle Time*

Dentro da área de desenvolvimento, o *lead time* está associado ao framework Kanban, e dentro do quadro de backlog, traça a tarefa em todas as suas fases: de sua criação até o status de *done*.

Do outro lado, temos o conceito de *Cycle Time*, que determina o tempo de uma atividade a ser trabalhada a partir do momento em que ela entrou em *doing* até o momento da finalização (*done*). Nesse caso, sempre menor ou igual ao *Lead Time*, conforme a Figura 5.

Tanto o *Lead Time* quanto o *Cycle Time* são métricas que presumem a utilização de horas ou dias para finalização de cada atividade. O cliente se importa com o *Lead Time* e o time técnico foca geralmente no *Cycle Time*. Lembrando que o esforço é sempre diferente do prazo. Concluir uma atividade em um dia não significa que ela será entregue nesse mesmo dia.

Ela será entregue ao cliente dentro do tempo do *Lead Time*, logo, esse *Lead Time* pode estar alto. E o que pode gerar esse número alto de horas ou dias para entrega?

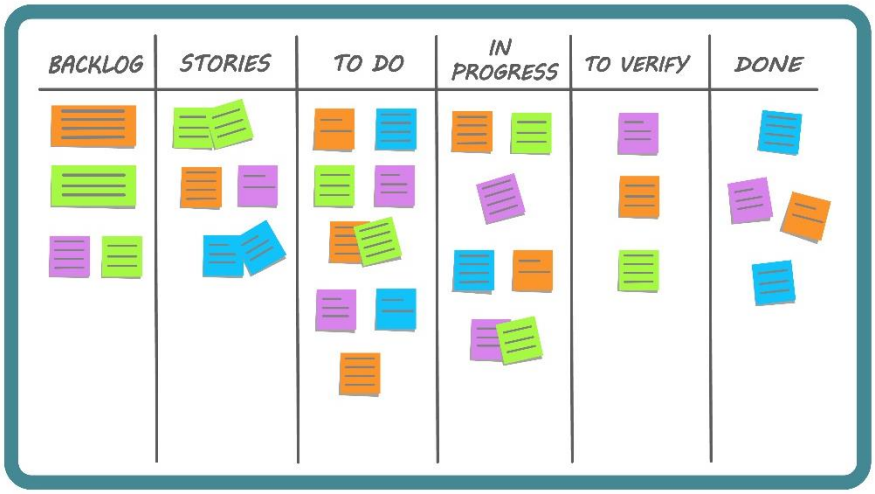
Em bloqueios por conta de processos de infraestrutura, por exemplo, a atividade faz parte de uma demanda que deve ser entregue com outras demandas de um determinado tipo e WIP não limitado.

Voltando-nos novamente à origem do *Lead Time* e do *Cycle Time*, esbarramos novamente no Kanban. O sistema Kanban (sistema puxado com origem no método de produção Toyota) permite a visualização do fluxo de trabalho na busca de geração de valor. Já o método Kanban é uma abordagem evolutiva e incremental de mudanças.

O objetivo do Kanban é sempre reunir pessoas comprometidas em mudanças de problemas que sejam claras e explícitas. Com ele, conseguimos otimizar nossos fluxos de trabalho, melhorar nossa comunicação e colaboração, além de transformarmos o ambiente de trabalho para que seja mais previsível e estável.

Para gerenciamento das atividades, utilizamos um quadro Kanban, conforme a Figura 5:

Figura 5 – *Kanban Board*



Crédito: Viktoriia Ablohina/Shutterstock.

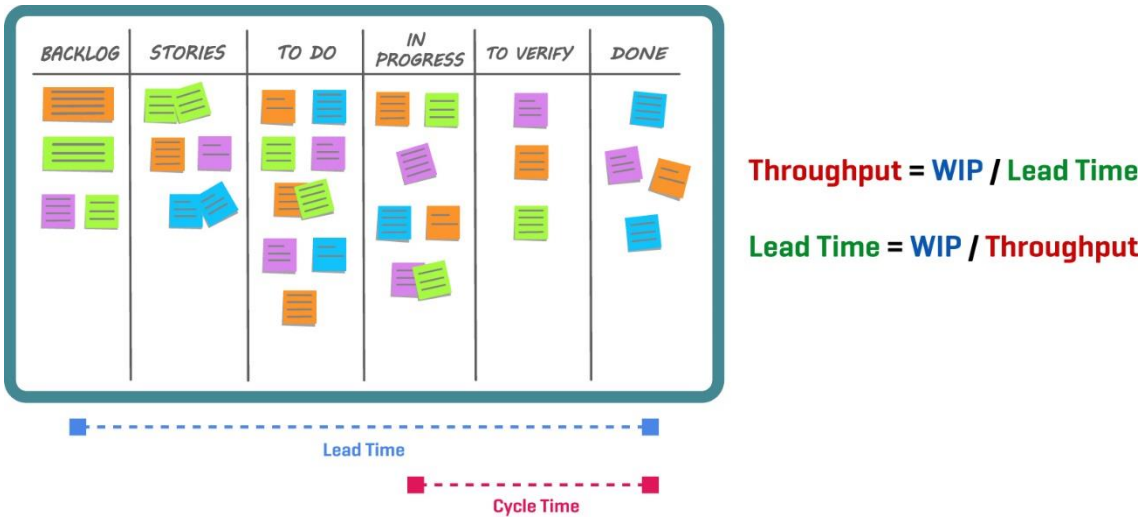
Lead Time e ***Cycle Time*** fazem parte do Lean Kanban, que combina práticas para geração de valor e trabalho criativo de forma ágil com foco na maximização da entrega de valor para o cliente. Outros termos importantes para Lean Kanban são WIP e *Throughput*.

WIP define o trabalho em progresso, em execução naquele ponto do processo. É importante sua sinalização no quadro Kanban porque conseguimos visualizar a quantidade de tarefas que estão em andamento a fim de confrontá-las com a capacidade de entrega do time. Quando limitamos o WIP, nosso trabalho torna-se mais estável e previsível, gerando confiabilidade nas entregas. O contrário cria filas maiores e aumento de risco nas entregas.

Outro termo importante dentro do Kanban é a métrica de ***Throughput***. Essa métrica demonstra a quantidade de tarefas entregues num determinado período. A visão da performance do time fica em evidência, e tudo aquilo que atrapalha a produtividade, como as filas de atividades, é melhor compreendido e melhorado.

Para calcularmos o *Throughput* e o *Lead Time*, temos (Figura 6):

Figura 6 – Métricas: *Throughput*, WIP e *Lead Time*



Crédito: Viktoriia Ablohina/Shutterstock.

Quanto melhor o *Throughput*, melhor a qualidade na validação das atividades anteriores, que deve ser menor, priorizadas, eliminando etapas se for possível e classificando as atividades para identificação das prioridades.

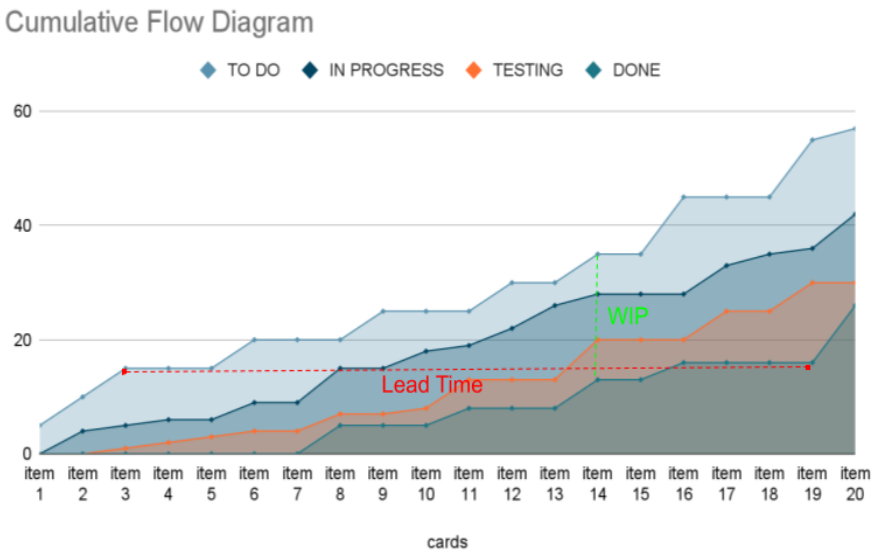
Quando o WIP não é adequado e limitado, os cards são muito grandes e pouco concisos, entre outras situações, a variação do *Throughput* torna-se instável.

Como podemos resolver o problema da variabilidade de *Throughput*? Por exemplo, determinando que *deploys* ocorrem num determinado dia da semana, estabelecendo melhor o espaçamento e periodicidade das reuniões, bem como limitando seu tempo, procurando manter o WIP mais enxuto.

4.5 *Cumulative Flow Diagram*

O *Cumulative Flow Diagram* (CFD) é um diagrama de fluxo acumulado que representa o registro de forma cumulativa da quantidade de demandas que passam pelas etapas de fluxo de trabalho (*to do*, *in progress*, *testing* e *done*), visto na Figura 7:

Figura 7 – *Cumulative Flow Diagram* exibindo o número de cards em cada coluna do quadro Kanban



Crédito: Maristela Regina Weinfurter Teixeira/Google Sheet.

Sua estrutura é muito simples e permite a visualização da capacidade de entrega por meio do status das tarefas. É muito bom para ajudar na identificação de gargalos ou bloqueios do time de desenvolvimento. No Eixo vertical temos a quantidade de tarefas e, no eixo horizontal, a linha do tempo. Um bom fluxo inclina suavemente, sem quebras ou saltos.

É vantajoso utilizarmos o CFD quando precisamos deixar visualmente expostas as disfunções do fluxo de trabalho. Os gargalos são bem visíveis, assim como o desbalanceamento entre as etapas do fluxo e a previsão de que o time

está fora do caminho para entrega dos itens do backlog. São situações importantes para gerenciarmos, pois, no caso de estarem fora do desejável, o gráfico o deixará bem exposto.

Cada etapa é representada por uma faixa gráfica de cor diferente. Os projetos normalmente estão divididos em estágios do desenvolvimento para acompanhamento, como em fila, em execução, em testes ou completadas.

É um modelo que não é engessado e pode variar de acordo com a natureza de cada projeto, seguindo suas especificidades. A vantagem do diagrama de fluxo cumulativo é que ele pode ser usado para muitas coisas:

- A linha feita é um *burn up* e pode ser usada para planejamento de projetos;
- A distância horizontal entre as linhas dá uma dica sobre o desempenho do seu sistema;
- A distância vertical entre as linhas dá uma dica sobre o que está errado com o seu sistema.

Ganhamos na visualização do andamento do projeto, no rastreamento de projetos ágeis, na melhoria de foco e no aumento de produtividade.

TEMA 5 – DÍVIDA TÉCNICA

Empresas têm convivido com o gerenciamento da dívida técnica. Esse gerenciamento deve ser proativo, para que o controle de custo das mudanças integre a tomada de decisão nas entregas de software.

Esse termo define o equilíbrio entre velocidade e retrabalho na busca de entregas de software de qualidade funcional (Kruchten, 2019). A dívida técnica começou a se traduzir em impacto financeiro, não necessariamente uma dívida do ponto de vista contábil, mas é um ponto de enormes custos financeiros no futuro. Inclui manutenção, inovação, obsolescência, integridade de dados etc.

O conceito de dívida técnica é bem aceito entre os desenvolvedores e lideranças e gera mudanças de curto prazo para manter o produto viável por décadas. Dessa forma, a dívida técnica também pode ser vista como uma espécie de investimento estratégico e uma forma de mitigar o risco.

Há dois conceitos importantes dentro do mundo ágil e é muito importante a discussão sobre eles, pois estão alinhados à agilidade e qualidade. Apesar de adotarmos todos os critérios de qualidade possível, nem sempre conseguimos

desenvolver ou código ou requisitos considerando todos os fatores importantes, especialmente quando falamos do lançamento de um MVP. Às vezes, também precisamos lidar com custos, outras necessidades mais urgentes, enfim, são muitas as situações que podem nos levar a não implementar o software como desejávamos.

E nessa linha de raciocínio, temos o que resta disso tudo, a chamada dívida técnica, pois não temos condições de implementar tudo de uma única vez. Quanto mais complexa é a ideia de nosso código, maior será nossa dívida técnica. Precisamos contornar caminhos dentro do desenvolvimento de software com abordagens mais simplistas e menos completas para conseguirmos avançar rapidamente nas entregas. Porém tais atalhos nos geram dívidas quase como dívidas financeiras na vida pessoal. Caso não sejam pagas, elas se acumularão e poderão alcançar níveis incontroláveis. Nosso código não muito certo vai nos cobrar juros dessa dívida.

Esse conceito tomou largas proporções no mundo de desenvolvedores e consegue descrever exatamente a entropia (mundo **dev** em desordem) inevitável no desenvolvimento de qualquer software. Às vezes, essa desordem ocorre por conta do projeto ao ser otimizado em decorrência de fatores externos, ou então por conta de escolhas não boas, ausência de informações importantes, revisões feitas sem a devida atenção e erros mesmo do desenvolvedor. Indiferentemente das razões, a dívida técnica é uma consequência inevitável no desenvolvimento de software.

Quanto mais a dívida técnica aumenta, mais cresce o custo para efetuarmos a mudança. A dívida técnica é uma fonte de complexidade que aumenta além dos níveis ideais. A capacidade de resposta do cliente diminui e perdemos a agilidade nas entregas, o que é essencial num modelo ágil.

Um dos grandes desafios da dívida técnica é que é difícil quantificar e priorizar aquilo que ficou para trás. À primeira vista, todas as atividades parecem essenciais e importantes. Há uma vertente que tenta criar uma espécie de monetização para dívida técnica, e seu custo de reparo está relacionado ao número de linhas. Por exemplo, se o código possui 10.000 linhas, o custo para corrigi-lo será de 20.000 dólares. Com essa situação de monetização da dívida técnica, estabelecemos um limite de crédito para a dívida técnica. Dependendo de como estamos a utilizando, podemos estabelecer que quando a dívida chega a 0,25 dólar por linha de código, está na hora de suspendermos o

desenvolvimento de novas *features*. A equipe, então, concentra-se na refatoração para redução do débito técnico aceitável.

A dívida técnica foi cunhada em 1992 por Ward Cunningham como uma metáfora em relação ao prejuízo que temos com as coisas que ficam para trás no desenvolvimento de software e retornam no formato de correções ou adaptações e implementações.

Apesar de falarmos anteriormente sobre a dívida técnica a nível de código, essa pode ocorrer em função do planejamento do projeto, do desenho da arquitetura, da documentação, dos testes, dos defeitos, dos requisitos funcionais e não funcionais, da infraestrutura, do corpo técnico de profissionais, da automação de testes, da usabilidade etc. Assim, ela pode surgir de diversas fontes relacionadas ao desenvolvimento de software.

A métrica do débito técnico permite que a inspeção contínua do código ocorra por meio de um feedback muito rápido dentro do processo de desenvolvimento de software, podendo mudar a proficiência do processo de software para saída do processo. A avaliação qualitativa é substituída pela mediação quantitativa da qualidade de software com o uso do débito técnico. A introdução desse conceito melhora muito o gerenciamento do processo de desenvolvimento de software.

Vimos que o processo de “consertar” um software é caro, porém ignorar esse assunto pode custar muito mais caro. Uma prática da análise ágil diz que devemos priorizar as dívidas técnicas. Equipes ágeis identificam, rastreiam, priorizam, monitoram e pagam sua dívida técnica. Geralmente, essas dívidas estão associadas às US, equilibrando, assim, entre o novo desenvolvimento e o acerto da dívida.

Um rastreador então é definido sempre que uma dívida técnica é apresentada ou descoberta. Tanto o sistema de rastreamento de dívidas como o sistema de rastreamento de defeitos não precisam ser complicados. Um gráfico simples na parede usando cartões de índice para capturar “histórias de dívidas técnicas” é uma maneira eficaz de um time gerenciar sua dívida técnica.

Geralmente, o PO, com auxílio do time, prioriza as histórias de dívidas técnicas, bem como estima o valor da eliminação de cada uma. Ele ainda garante que a prioridade das novas histórias de usuário nem sempre supere a prioridade das histórias de dívida técnica. Os líderes de equipe técnica devem defender

oportunidades para agendar histórias de dívidas em iterações de desenvolvimento.

Podemos elencar alguns aspectos que nos auxiliam na mensuração da dívida técnica:

1. Duplicidade de código;
2. Coesão de código;
3. Fragilidade de builds;
4. Fragilidade de testes automatizados;
5. Tamanho dos métodos das classes;
6. Cobertura ineficiente de testes unitários.

Algumas equipes de projetos ágeis alocam de 15 a 20% da capacidade da equipe em cada iteração para redução da dívida, deixando 80 a 85% para o desenvolvimento de histórias de usuários. Outras equipes ágeis ocasionalmente designam iterações explícitas de redução da dívida que evitam o desenvolvimento da história do usuário em favor do trabalho de redução da dívida. Ainda, outras equipes ágeis encontram maneiras de eliminar a dívida técnica durante o desenvolvimento da história do usuário. Seja qual for a abordagem, é essencial monitorar e pagar continuamente a dívida técnica.

FINALIZANDO

Conhecemos ferramentas, técnicas, conceitos e formas de medição dentro do modelo ágil que nos direcionam a pensar, planejar e gerenciar o desenvolvimento de software com a garantia da qualidade que almejamos, tanto para o processo quanto para o produto.

As metodologias e frameworks ágeis, que foram evoluindo desde os anos 2000, trazem técnicas e ferramentas simples e fáceis de serem implantadas para quantificarmos o quanto estamos sendo produtivos e entregando produtos na hora certa, com os requisitos certos e com um código limpo.

Acabamos fazendo uma combinação de ferramentas e técnicas para melhor gerenciarmos nossos projetos de desenvolvimento de software. A adaptação no uso de metodologias para o nosso time de desenvolvimento e para o time de SQA traz maior aderência com qualidade no desenvolvimento de software. Scrum, Kanban, XP e todos os demais apresentam características similares, porém com técnicas diferentes.

É aí que o mercado acaba evoluindo para Scrumban, ou coisas do tipo. Temos várias empresas trabalhando com a metodologia Scrum no sentido de processo de desenvolvimento de software, com adaptações, com programação em pares do XP para garantir melhoria na construção e validação do código, o quadro Kanban para estabelecer *done*, *doing* e *testing* e todas as etapas do desenvolvimento de software em um mix de ferramentas de qualidade ágeis que estudamos até aqui.

O importante, tanto no uso do modelo ágil quanto nas métricas ágeis, é justamente aprender a adaptá-las à realidade do time e da cultura da empresa, além claro, das questões financeiras e cronogramas de que dispomos.

REFERÊNCIAS

ISWANTO, A. H. **The lean enterprise**: tools for developing leadership in a lean culture. New York: Productivity Press, 2020.

KRUCHTEN, P. et al. **Managing technical debt**: reducing friction in software development. San Francisco: Addison-Wesley Professional, 2019.

NADER-REZVANI, N. **An executive's guide to software quality in an agile organization**: a continuous improvement journey. Los Altos: Apress, 2018.

ROTHMAN, J. Create your successful agile project. Pragmatic Bookshelf, 2017.

SADOWSKI, C. **Rethinking productivity in software engineering**. Los Altos: Apress, 2019.