1

O Manifesto Ágil

ALEXANDRE GOMES, RENATO WILLI E SERGE REHEM

O Manifesto Ágil é a pedra fundamental do conteúdo deste livro. Apesar de não ser anterior a todas as metodologias e práticas vistas, este documento sintetiza as questões básicas do chamado Movimento Ágil.

1.1 HISTÓRICO

Durante anos, a Engenharia de Software inspirou-se em processos de manufatura para a consolidação de seus métodos de trabalho. Nascida na segunda metade do século XX, buscou em setores emergentes da indústria da época grande parte das teorias e dos métodos de produção. Em especial, o campo automobilístico, em ampla ascensão industrial, teve importante papel para a constituição da nova indústria de TI. Graças ao modelo de produção em série de Henry Ford, altamente inspirado por Frederick Taylor, todo o pensamento tradicional da ciência do desenvolvimento de software desenrolou-se com intenso foco na padronização de componentes e processos e na mecanização do movimento.

Já em meados dos anos 90, começaram a surgir processos alternativos de desenvolvimento de software, em resposta àqueles tradicionais, considerados excessivamente regrados, lentos, burocráticos e inadequados à natureza da atividade. Esses novos processos foram apelidados de "leves" (*lightweight*), em oposição aos anteriores, "pesados" (*heavyweight*).

Em comum, ambos são baseados em desenvolvimento iterativo, no qual requisitos e soluções evoluem pela colaboração entre equipes auto-organizadas e *cross-funcional* (pessoas com diferentes expertises). Encorajam frequente inspeção e adaptação, uma filosofia de liderança, alinhamento entre o desenvolvimento e os objetivos das empresas ou dos clientes e um conjunto de boas práticas de engenharia que permitia entregas rápidas e de alta qualidade.

Essas metodologias só passaram a ser chamadas de ágeis após 2001, quando um grupo de 17 especialistas (veja o Quadro 1 a seguir) se reuniu na estação de ski *Snowbird*, em Utah, nos Estados Unidos, para discutir maneiras

de desenvolver software de uma forma mais leve, rápida e centrada em pessoas. Eles cunharam os termos "Desenvolvimento Ágil de Software" e "Métodos Ágeis" e criaram o Manifesto Ágil – amplamente difundido como a definição canônica do desenvolvimento ágil, composto pelos valores e princípios que veremos a seguir. O Manifesto foi publicado em 2001, e qualquer pessoa pode ser signatária. Mais tarde, algumas dessas pessoas formaram a Agile Alliance (2014), uma organização sem fins lucrativos que promove o desenvolvimento ágil.

Autores do Manifesto Ágil

Kent Beck
Criador da *ExtremeProgramming* – XP (Capítulo 4), *Test-Driven Development* (Desenvolvimento dirigido por testes, Capítulo 11) e JUnit (framework usado no desenvolvimento de testes de unidade). Uma das maiores referências do mundo ágil.

Jeff Sutherland e Ken Schwaber
Inventores do Scrum, que veremos no Capítulo 3.

Martin Fowler
Autor dos livros *Analysis Patterns*, *Planning Extreme Programming* e *Refactoring*, importante referência em design para desenvolvedores.

Dave Thomas e Andrew Hunt
Coautores do livro *O Programador Pragmático*, referência para desenvolvedores. Pregam a simplicidade e leveza no desenvolvimento, além de metodologias centradas em pessoas.

Alistair Cockburn
Criador da família de métodos ágeis chamada de Crystal.

Ward Cunningham
Criador do método de design CRC e contribuidor para outras metodologias, incluindo XP.

Arie van Bennekum
Ativamente envolvido no consórcio DSDM (*Dynamic Systems Development Method*).

Brian Marick
Representante da comunidade de testes e das ideias do que o *Agile Testing* pode ser.

Jim Highsmith
Autor do método *Adaptive Software Development* (ASD) e do livro com mesmo nome.

Robert C. Martin
Experiente em XP, autor do livro *Principles*, *Patterns*, *and Practices of Agile Software Development* e, mais recentemente, *Clean Code*.

Ron Jeffries
Primeiro coach em XP, proprietário do XProgramming.com e coautor do livro *Extreme Programming Installed*.

Jon Kern
Programador e arquiteto experiente em diversas linguagens, na época trabalhava na TogetherSoft. Foi representar Peter Coad, o dono da TogetherSoft e um dos criadores da FDD (Capítulo 6).

Mike Beedle
Adotou Scrum e XP como metodologias ágeis com sucesso em diversos projetos, coautor do livro *Scrum, Agile Software Development*, com Ken Schwaber.

Stephen J. Mellor
Também conhecido como Steve Mellor, autor de *Executable UML* e *MDA Distilled*, foi coordenador do *Advisory Board* da revista IEEE Software por dez anos.

James Grenning
Um dos criadores da técnica conhecida como *Planning Poker*. Autor de *Test-Driven Development for Embedded C*.

1.2 O MANIFESTO ÁGIL

O Manifesto Ágil é composto pela declaração de alguns valores e por 12 princípios, apresentados na próxima seção.

Há, na comunidade, um grande debate sobre o que é ser "ágil". Ao contrário de outras culturas de desenvolvimento, agilidade não está relacionada à obediência de protocolos preestabelecidos de produção, mas a novos padrões de comportamento e atitude. Portanto, uma equipe não pode se dizer "ágil" se não se comportar assim. Livros e artigos são ótimas fontes de conhecimento, mas nenhum time se torna ágil por sua simples leitura. Afinal, a agilidade não é outorgada, mas alcançada a cada pequena transformação diária de comportamento. Cada Método Ágil define suas próprias práticas, mas todos, em um momento ou outro, compartilham dos valores e princípios postulados pelo Manifesto Ágil.

Analisemos o que diz o Manifesto (2001) sobre novos valores do desenvolvimento de software:

> "Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através desse trabalho, passamos a valorizar:
>
> Indivíduos e interação mais que processos e ferramentas
> Software funcionando mais que documentação abrangente
> Colaboração com o cliente mais que negociação de contratos
> Responder a mudanças mais que seguir um plano
>
> Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda. "

Comecemos analisando a última frase, frequentemente suprimida nas citações, o que pode levar a muitas interpretações falhas e preconceitos relacionados aos Métodos Ágeis. Ela destaca que há, de fato, valor nos itens à direita. Isto é, há valor em processos, ferramentas, documentação, contratos e planos. Por muito tempo, no entanto, as soluções propostas para os desafios da engenharia de software focaram demasiadamente essas questões, deixando em segundo plano outras que também eram importantes – os itens à esquerda. Talvez o Manifesto seja uma tentativa de contrabalançar esses itens, pois aprendeu-se que eles são críticos para o sucesso dos projetos. Enfatizamos que processos, ferramentas, documentação, contratos e planos são muito importantes, mas que, se tivermos de priorizar entre eles e os itens da esquerda, escolheremos os da esquerda.

A primeira frase do Manifesto afirma que ainda "estamos descobrindo melhores maneiras de desenvolver software". Podemos inferir, a partir dessa declaração, que a engenharia de software ainda é muito incipiente, não está tão dominada quanto outras engenharias, como a civil ou mecânica (mesmo que todas evoluam a cada dia). Isso é muito natural, uma vez que a engenharia de software tem apenas cerca de 40 anos. Logo, não se propõe uma solução definitiva, mas bons indícios do que possa ser um melhor caminho rumo ao sucesso nos projetos. Ainda estamos em fase de aprendizado.

Analisemos agora, cada valor do manifesto:

- **Indivíduos e interação** mais que processos e ferramentas
  Passamos a crer tão cegamente nos processos e nas ferramentas que deixamos de nos comunicar. Esquecemos que são as pessoas que fazem software. Em vez de conversas e discussões, os desenvolvedores passaram a receber especificações escritas. Elas são importantes, sim, mas não comunicam tão bem como uma boa discussão presencial, ou esboços, rabiscos e modelos. Obviamente, ferramentas são importantes. É muito mais difícil fazer as coisas sem elas. Processos, igualmente. Ainda assim, não devemos deixar de valorizar as pessoas e não devemos deixar de nos comunicar. Isso faz parte de trabalho em equipe. Portanto, se essas questões começarem a disputar espaço, valorize mais o lado humano e você terá boas chances de obter melhores resultados.
- **Software em funcionamento** mais que documentação abrangente
  No início da engenharia de software (e em muitos locais até hoje), muitas organizações ficaram reféns de seus desenvolvedores. Como não havia documentação, todo o conhecimento estava em suas mentes. Perder uma dessas pessoas significava um prejuízo incalculável. A solução encontrada foi documentar os processos para a posteridade. Surgiram, então, as figuras de analistas de sistemas e documentadores, profissionais contratados não para programar, mas para produzir modelos gráficos e textuais. Talvez tenhamos errado na mão, e a proposta agora seja a de encontrar um ponto de equilíbrio. O Manifesto não nega a importância da documentação. No entanto, é preferível a entrega de software funcionando do que uma documentação abrangente, exagerada e cheia de desperdícios. Quando somos contratados, o resultado esperado é software funcionando, com qualidade. Documentação e manutenibilidade fazem parte dessa qualidade. Devemos refletir mais sobre "o que" documentar e "quando" documentar. Devemos refletir sobre o que é útil de fato e o que ficará defasado rapidamente ou sequer será lido algum dia. Isso gera um tremendo desperdício e encarece o que fazemos.
- **Colaboração com o cliente** mais que negociação de contratos
  Escopo é uma questão complexa e difícil de ser definida precisamente num texto de contrato. Além disso, desenvolver software é um processo de aprendizado: muito do que o sistema vai se tornar será aprendido ao longo do seu desenvolvimento. Vemos muitos fracassos nos projetos devido a essa dificuldade, pois o caminho normalmente seguido para resolver essa questão tem se mostrado oposto ao adequado. São criadas cláusulas e mais cláusulas com o objetivo de proteger tanto o contratante quanto o contratado, na tentativa de fechar o escopo o máximo possível, e são impostos processos complexos, burocráticos e frustrantes para mudanças. O resultado continua ruim.

  Esse é um ponto fraco do Manifesto e dos Métodos Ágeis, constantemente criticado devido à sua fragilidade e pessoalidade. É algo que

definitivamente ainda tem muito a evoluir. Sabemos que, quando a relação é bem construída, os resultados são melhores. No entanto, as partes precisam de alguma segurança contra atitudes de má fé. Para minimizar esse risco, normalmente os contratos têm espécies de "pontos de controle", em que a relação é reavaliada para se decidir pela continuidade ou descontinuidade do contrato sem ônus. Naturalmente, estando ambas as partes satisfeitas com a relação, mantém-se o compromisso. Caso contrário, busca-se o realinhamento de interesses e, não havendo acordo, suspende-se a continuidade do projeto. O Manifesto admite que é muito difícil se endereçar todas as complexas questões do desenvolvimento em contratos. Tentar criar muros de proteção não vai resolver nada se não houver colaboração entre a equipe e o cliente. Então, em vez de tentar resolver as coisas incluindo novas cláusulas, redigindo contratos super complexos, é preferível trabalhar em outro nível com o cliente, criando um clima de confiança e colaboração.

- **Responder a mudanças** mais que seguir um plano
  Como já mencionado, desenvolver software é um processo de aprendizado, tanto da equipe quanto do próprio cliente. Assim, é natural e inevitável que haja mudanças. Acreditamos que as mudanças são ótimas oportunidades para que o sistema desenvolvido seja mais aderente às necessidades do cliente, além de contribuírem muito para os resultados desejados. Por isso, devemos fazer o possível para recebê-las e acolhê-las de braços abertos.

  Em projetos, normalmente temos a ideia de que será traçado um plano no início do prazo e de que ele será seguido até o final. É muito difícil tomar tantas decisões acertadas no início do projeto, no momento em que menos se conhece a solução, principalmente em um ambiente instável como o nosso, em termos de tecnologia, pessoal e negócio. Até podemos seguir o plano, mas o resultado final pode não resolver o problema do cliente.

  Por isso, para acolhermos realmente as mudanças, precisamos replanejar o tempo todo. Os processos de planejamento ágil normalmente incluem ciclos PDCA em diversos níveis (diário, semanal, mensal, trimestral, etc.), em que há a oportunidade de reflexão e readequação dos rumos tomados pelo projeto.

1.3 OS 12 PRINCÍPIOS

Princípio é "[...] toda estrutura sobre a qual se constrói alguma coisa. São ensinamentos básicos e gerais que delimitam de onde devemos partir em busca de algo, verdades práticas que visam a treinar nossa mente para melhor discernirmos sobre os caminhos corretos a serem tomados nos objetivos. É através deles que podemos extrair regras e normas de procedimento (Dicionário online de português, 2014)".

Os 12 princípios do Manifesto Ágil complementam os valores, formando os pilares sobre os quais são construídos os chamados Métodos Ágeis.

Enunciados simples, mas de significado abrangente e profundo, são às vezes esquecidos no dia a dia por equipes "supostamente" ágeis, que cometem a falha comum de se ater a práticas específicas, sem buscar suas razões essenciais. Faremos um breve comentário sobre cada princípio. Se em alguns trechos formos repetitivos, tudo bem, assim vamos fixando os conceitos.

1. **Nossa maior prioridade é satisfazer ao cliente com entregas contínua e adiantada de software com valor agregado.**

Não é à toa que esse é o primeiro princípio. A evolução da engenharia de software tem trazido diversos processos, técnicas e ferramentas que, apesar de organizarem e documentarem o ciclo de vida do desenvolvimento de soluções, tornaram-se mais importante que o próprio software a ser entregue. Por outro lado, temos também, e não podemos deixar de ressaltar, os cérebros engenhosos dos analistas e programadores, ávidos por aplicar o "estado da arte" das mais recentes tecnologias, linguagens e ferramentas, colocando em risco a qualidade do produto e deixando em segundo plano as necessidades do cliente. O primeiro princípio do Manifesto Ágil resgata o maior objetivo que devemos ter em mente: entregar software funcionando com qualidade, com iterações rápidas e contínuas, sempre agregando valor de negócio ao cliente.

2. **Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Os processos ágeis tiram vantagem das mudanças, visando à vantagem competitiva para o cliente.**

Tradicionalmente, grande parte das metodologias de desenvolvimento de software buscou técnicas e ferramentas para desestimular a possibilidade de mudanças. Afinal, sabemos que, quanto mais tarde elas ocorrerem, maior será o custo de manutenção. Por que favorecê-las, então? São práticas comuns o levantamento inicial do máximo de requisitos possíveis, o estabelecimento de termos de compromisso de longo prazo para seu desenvolvimento (p. ex., assinaturas do cliente em todos os documentos de um modelo de casos de uso) e a criação de processos burocráticos para a solicitação de mudanças do compromisso preestabelecido. Acreditou-se por muito tempo que, com todos esses cuidados, o trabalho inicial de identificação de requisitos não seria prejudicado ao longo da construção do software, controlando, assim, prejuízos exponenciais da modificação tardia de funcionalidades.

Percebendo a ineficiência das práticas adotadas contra mudanças no decorrer do desenvolvimento, a filosofia ágil optou por discordar da premissa secular de que mudanças tardias são maléficas e adotou uma postura favorável à sua ocorrência. Agilistas, portanto, aceitam com naturalidade o fato de que transformações no escopo original de qualquer projeto são esperadas e muito bem-vindas. Com isso, mudanças de qualquer natureza passam a ser encaradas como algo normal.

A diferença agora é que, em vez de lamentar pela necessidade de modificação do plano original de trabalho, os Métodos Ágeis preparam-se com técnicas e ferramentas para responder o mais rápido possível a todo tipo de mudanças, que certamente é reflexo do aprendizado de alguma circunstância

até então não percebida pelos envolvidos. Dessa forma, o cliente livra-se das amarras de decisões precipitadas, refletidas em termos de compromisso prematuramente firmados e cuja rescisão lhe trará ônus, e beneficia-se do imenso potencial competitivo da adaptabilidade a novos cenários de mercado.

3. **Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.**

Você já deve estar percebendo o quanto os princípios estão interligados. Entregar com frequência software funcionando ao cliente, agregando valor e sendo capaz de responder rapidamente a mudanças só é possível com ciclos curtos. Os time-boxes (períodos de tempo pré-fixados e predeterminados) dão ritmo ao trabalho, e a equipe passa a ter consciência da sua velocidade, ou seja, passa a prever cada vez melhor o quanto é capaz de produzir em cada ciclo. O projeto passa por diversas iteracões de melhoria contínua, potencializando aspectos positivos e atuando nos pontos de melhoria identificados. Como consequência, a relação de confiança com o cliente (e entre os próprios membros do time!) só tende a aumentar ao longo do projeto.

4. **Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.**

Esse é um ponto difícil de implementar, mas nem por isso deve deixar de ser buscado. Clientes "tradicionais" gostariam de participar de poucas reuniões, falar sobre o que eles querem em seu sistema e aparecer um tempo depois para ver se o que pediram foi apresentado a contento. Quem trabalha com TIC já dever ter ouvido várias vezes a frase: "isto não foi exatamente o que pedi". Mesmo que os requisitos tenham sido bem elicitados e documentados, as necessidades de negócio mudam, porque a realidade do ambiente em que o sistema está inserido também muda (p. ex., cenário econômico, mudanças políticas, ações da concorrência).

Uma das maneiras de se evitar isso é adotando um processo constante de colaboração entre clientes e equipes de desenvolvimento, prioritariamente trabalhando juntos no mesmo ambiente. Esse princípio pode ser um pouco "impactante", pois muitos profissionais foram educados a não interagir diretamente com o cliente. "Isso só quem faz é o gerente ou os analistas de requisitos". A ação conjunta de times ágeis representantes diretos do cliente contratante possibilita um fluxo contínuo de apresentação, discussão e feedback, que é fundamental para a garantia de sucesso do projeto. Se você não consegue que estejam diariamente no mesmo ambiente, procure estratégias para maximizar os momentos de contato. É desnecessário mencionar o quão dependente esse princípio é dos anteriores.

5. **Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessários e confie neles para realizar o trabalho.**

Equipes ágeis são auto-gerenciadas. Não existe alguém dando ordens ou cobrando resultados. Em um primeiro momento, tais premissas podem soar como anarquia, como se cada um pudesse fazer o quisesse na hora que bem entendesse.

Em um time verdadeiramente ágil, o ambiente é de comunicação direta e constante, os feedbacks são frequentes e o comprometimento é de todos. A prioridade continua sendo a entrega constante de software funcionando, com valor agregado ao cliente. Metodologias como Scrum e XP apresentam alternativas para montagem do ambiente adequado (mobiliário, equipamentos, ferramentas de apoio, quadros de sinalização, etc.) e para o uso de práticas que favoreçam a criação de um clima motivador e de confiança mútua.

Um alerta: profissionais sem iniciativa rapidamente são desmascarados em um ambiente dessa natureza. A figura do gerente estilo "comando e controle", que vive cobrando resultados, cede espaço para o líder facilitador, que confia em seu time e está ali para servi-lo em prol de um objetivo maior.

6. **O método mais eficiente e eficaz de transmitir informação para a equipe e entre a equipe de desenvolvimento é a conversa frente a frente.**

Os problemas de comunicação parecem aumentar quanto mais a tecnologia se desenvolve. As relações humanas diretas vêm sendo gradativamente substituídas por instrumentos catalizadores do processo de comunicação. É inquestionável o papel que telefone, email e chat tiveram na dinamização do mundo moderno. Mensagens que antes demoravam dias para alcançar seus destinos são hoje entregues instantaneamente em qualquer canto do globo. Hoje, comunica-se muito mais que anos atrás.

Mesmo assim, apesar de tanta facilidade e conveniência, persistem na sociedade os mesmos problemas de comunicação há muito identificados e criticados pelos maiores experts em gestão de pessoas. Valendo-se apenas de objetivos quantitativos, a troca de mensagens em tempos de Internet ainda carece de qualidade.

Todo o aparato tecnológico disponível ainda não substitui a clareza e objetividade de uma boa conversa presencial, na qual está presente a importantíssima comunicação não verbal. Sutilezas, como gestos, entonação de voz e expressões faciais, não podem ser eficientemente transmitidas por meio eletrônico (o uso de letras maiúsculas e emoticons tenta minimizar o problema), o que dificulta a compreensão da mensagem. São apenas um conjunto de palavras e sentenças frias e, supostamente, objetivas.

O autor de um texto escrito geralmente o faz sob a premissa de que ali reside uma informação clara e de conclusões determinísticas. Talvez seja essa a matriz de todos os problemas associados à comunicação eletrônica. É importante que entusiastas da comunicação escrita atentem para a natureza humana e inexata de seus textos. Ao contrário de uma fórmula matemática, textos escritos não têm conclusões absolutas. A compreensão das intenções do autor dependem, em grande parte, da capacidade de interpretação do receptor da mensagem. Essa capacidade, por sua vez, depende da bagagem de conhecimento, da atenção dispensada, do estresse e mesmo do humor de quem está lendo a mensagem. Ademais, na comunicação remota, o autor não dispõe de qualquer oportunidade para esclarecimentos acerca de

suas reais intenções postas em palavras. Disso, emergem as mais diversas situações, com potenciais extremos de destruição de relações, gerando desgastes, atrasos e imenso desperdício de energia para a recomposição do ambiente agradável.

O Manifesto Ágil coloca que, dentre todos os tipos de troca de informação entre equipes de desenvolvimento de software, a mais eficaz é a da comunicação frente a frente. Quanto menos comunicação indireta, menores serão os riscos de má interpretação. Quanto mais frequentes forem as conversas presenciais, menos conflitos surgirão, menos energia será gasta para sua reversão e mais eficazes e sustentáveis serão os trabalhos.

7. **Software funcional é a medida primária de progresso.**

A engenharia de software se espelhou em outras indústrias do século XX, tradicionalmente manufatureiras, e nasceu baseada em processos teoricamente determinísticos e controláveis. Pelo acompanhamento dos artefatos gerados em cada etapa de trabalho, seria possível mapear o andamento de qualquer projeto, avaliar seus riscos e estimar seus prazos e custos. Em síntese, a Gerência de Projetos tradicional funciona desta forma:

- Busca-se identificar, no início do projeto, em alto nível de detalhamento, todas as atividades a serem executadas para entrega do produto final.
- Distribuem-se todas as atividades identificadas em um cronograma, atribuindo-lhes prazos e responsáveis.
- Acompanha-se o desenrolar dos trabalhos até a conclusão de todas as tarefas previstas.

Esse modelo, apesar de altamente aceito nas mais diferentes indústrias (TI, construção civil, promoção de eventos, etc.) demonstrou-se ineficiente e de pouco valor na engenharia de software. Por sua causa, criou-se uma cultura de supervalorização de documentos descritores do projeto. A entrega de artefatos tornou-se mais lucrativa que a entrega de software, sendo prática recorrente do mercado o faturamento de mais da metade do valor de um contrato de software estar vinculado apenas à entrega de modelos e diagramas abstratos.

O Manifesto Ágil, em sua segunda cláusula, e também o livro de Fried, Hansson e Linderman (2006), *Getting Real*, propõem que código funcionando seja mais importante que uma documentação extensa. Conforme já mencionados, documentos e especificações têm validade, mas priorizá-los em detrimento de um software bem feito e funcional é um erro. O 7º princípio do Manifesto ratifica esse discurso, esclarecendo que o bom andamento de um projeto de desenvolvimento de software deve ser mensurado, primordialmente, por meio da quantidade de software entregue e funcionando, que é o que, de fato, importa ao cliente final, e não pelo volume de documentos gerados.

8. **Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante sempre.**

Conceito proveniente da indústria manufatureira, a produtividade de uma equipe ainda é diretamente proporcional ao número de horas trabalhadas na linha de produção. Seguindo o mesmo raciocínio, empresas de TI tradicionais popularizaram o estereótipo do profissional de informática *workaholic*, constantemente estressado e com jornadas de trabalho sobre-humanas. Parte-se da premissa de que quanto mais tempo o profissional estiver em frente ao computador digitando código, maior será sua produção. Esse raciocínio não vale para um segmento produtivo pautado pela atividade intelectual, cujos resultados práticos são diretamente proporcionais à capacidade criativa de seus executores.

O desenvolvimento de software, há muito entendido como tarefa mecânica e repetitiva, finalmente está sendo compreendido como a arte de transformação de ideias em código. A criatividade de um desenvolvedor trabalhando em seu limite dificilmente estará fértil; por consequência, cada novo problema a ser transposto pode gerar complexidades desnecessárias e a introdução desmedida de defeitos, desconstruindo o que já está pronto. Prática comum na maioria das empresas de TI, esse processo cria um ambiente improdutivo, de alta rotatividade de pessoal e baixa difusão de uma cultura corporativa.

Para reverter tal cenário, o Manifesto Ágil revisa essa tradição industrial e sugere modelos de maior sustentabilidade para todos os envolvidos no processo de construção de software. O ponto-chave da proposta é a manutenção de ambientes que funcionem não em seus limites operacionais, mas em níveis nos quais sua sustentação seja viável por ilimitados períodos de tempo.

9. **Contínua atenção à excelência técnica e bom design aumenta a agilidade.**

Neste ponto, você pode estar se perguntando como é possível manter um ambiente sustentável de alta produtividade, priorizando a entrega contínua de código e a agregação constante de valor, estar sempre atento às necessidades de mudança e sem o poder de uma extensa e esclarecedora documentação. A resposta é óbvia, mas nem sempre é seguida: fazendo um bom código. Um código bem feito aliado a um projeto de qualidade elimina a necessidade de documentação exaustiva, reduz o retrabalho e facilita a tomada rápida de decisões. Dessa forma, viabiliza-se a entrega constante de versões funcionais e a resposta rápida a feedbacks do cliente. Uma teia de confiança é estabelecida entre todos os envolvidos, o que, naturalmente, possibilita a implantação de rotinas sustentáveis de trabalho, realimentando a cadeia e potencializando a consolidação de um círculo virtuoso de produção.

10. **Simplicidade – a arte de maximizar a quantidade de trabalho não realizado – é essencial.**

Essa é uma frase muito poderosa, que merece nossa atenção e reflexão. A arte de maximizar o trabalho não realizado pode parecer “coisa de

preguiçoso", como se quiséssemos arranjar desculpas para não fazer algum trabalho (o que ocorre – e muito – em todas as profissões). Na verdade, o que queremos dizer é: procure focar o que é realmente importante e que trará valor de negócio e vantagem competitiva ao seu cliente. Elimine o que não é importante. Esse princípio nos leva constantemente aos questionamentos do tipo: isso é realmente essencial? Há alguma forma de tornar isso mais simples?

O pós-guerra consolidou o modelo capitalista de produção, a indústria do consumismo e os valores do exagero. Todos os meios de comunicação vendem a ideia de que temos sempre de ter mais: mais roupas, mais carros, mais imóveis... E não foi diferente no mundo do software. Vimos, nos últimos anos, uma geração de aplicativos transbordando de funcionalidades. Não raro, ouvimos comentários de usuários do tipo "não utilizo 20% dos recursos do software X" ou "não sei para que serve metade desses botões". Esse excesso, entretanto, não agrega diferencial ao produto em questão e geralmente tem o efeito oposto. Afinal, mais funcionalidades representam mais complexidade de uso, e mais código potencializa mais defeitos.

Muitos confundem Métodos Ágeis com o desenvolvimento rápido de software. Na verdade, o ágil diz muito mais sobre eficiência, eficácia e efetividade do desenvolvimento do que sobre velocidade de programação. Ou seja, a energia gasta nas atividades do projeto deve ser aplicada para o desenvolvimento da coisa certa, que mais agregue valor ao cliente, e não apenas para geração incontida de código com pouco potencial de retorno do investimento aplicado.

A manutenção de um projeto simples, com poucas funcionalidades, além de reduzir a complexidade do projeto, facilitando sua manutenção, garante ao time tempo e energia para aprimoramento (simplificando ainda mais toda a arquitetura) ou para implementação de outra funcionalidade de valor extremo. Isso é nada mais que a aplicação prática do Princípio de Paretto (significado..., 2014) para o desenvolvimento de software: implementar apenas os 20% de funcionalidades que representarão os 80% de resultado (Koch, 2001).

O Manifesto Ágil, entretanto, não é o autor dessa filosofia pró-simplicidade. O Desenho Industrial há muito a defende, e a própria comunidade desenvolvedora de software também já a conhece (Instituto Nacional da Propriedade Industrial, 2014). Leander Kahney (2008), no livro em que descreve "A cabeça de Steve Jobs", dedica todo seu primeiro capítulo à "arte de dizer não" do fundador da Apple e o consequente mérito ao sucesso de seus produtos. Na gíria geek, geralmente em fóruns de discussões, propostas de complexidade absurda são comumente rechaçadas com KISS (*Keep It Simple Stupid*) (What..., 2000).

11. **As melhores arquiteturas, requisitos e design emergem de times auto-organizáveis.**

Esse tópico está estritamente relacionado ao princípio da emergência (Dicionário online de Português, 2014), que define o processo de formação de sistemas dinâmicos complexos a partir de regras simples.

Durante anos, acreditou-se na existência de mecanismos invisíveis de comunicação e liderança que justificassem toda a harmonia por trás da dinâmica de movimentação em um bando de pássaros ou em um cardume. Ultimamente, entretanto, a teoria da emergência tem sido a mais bem aceita para explicar todo sincronismo. Ela defende que existe um conjunto restrito e muito simples de regras que deve ser obedecido (por questões de sobrevivência) por todos os membros do grupo. Por exemplo, cada membro não pode se aproximar demais de outro, sob o risco de colisão, mas também não pode se afastar demais, para não ficar vulnerável aos predadores. Com isso, ao menor desvio de rota do vizinho, cada componente do grupo deve se reorganizar para manter válidas as regras básicas (de sobrevivência) do jogo. Assim, com essas duas simples regras, todo o grupo se sincroniza a ponto de parecer um único corpo aos olhos de um espectador, e a vida se perpetua.

No desenvolvimento de software, durante anos, acreditamos ser capazes de controlar todos os possíveis eventos que pudessem influenciar o bom andamento de qualquer projeto. Sob a batuta dos gerentes, o planejamento prévio e detalhado de cada passo do processo tornou-se o pilar fundamental dessa cultura de intenso controle. Por meio de cronogramas, planilhas e pilhas de documentos devidamente firmados, estabelecia-se a crença comum do domínio de todos os potenciais fatores de impacto aos trabalhos em curso. E, a qualquer imprevisto, incrementava-se ainda mais o hall de pontos de controle para ocasiões futuras, transformando a dinâmica de trabalho em um processo fechado, complexo, cada vez mais custoso e com necessidade de profissionais mais e mais experientes em todas as práticas de controle existentes.

O fato, entretanto, é que, mesmo com toda essa política de prevenção adotada e o aparente determinismo do plano preestabelecido, elementos surpresa continuam a surpreender os mais experientes gerentes de projetos, comprometendo suas estimativas de custo e prazo. Apesar de todos os documentos de controle e todas as assinaturas de comprometimento lavradas a ferro e fogo, desvios do plano original nunca deixaram de existir. Assim como a natureza ou o cérebro humano, o desenvolvimento de software é um processo de tamanha complexidade que qualquer tentativa de sistematização tende a ser falha. Ora, sabendo que há uma grande probabilidade de mudança do plano original, por que investir tanto tempo em seu detalhamento?

12. **Em intervalos regulares, o time reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.**

Chegamos ao último princípio do Manifesto Ágil e, até o momento, nada se falou sobre uma sequência de fases e/ou atividades que devem ser seguidas para a construção de um projeto de software de sucesso.

A engenharia de software tradicional vem insistindo na definição de uma "receita de bolo" ideal para o desenvolvimento de sistemas. Nenhum processo de desenvolvimento de software que conhecemos, entretanto, chegou a esse nível, o que nos sugere duas possíveis conclusões: devemos acreditar na existência do processo perfeito e insistir em sua busca, como defende a maioria dos profissionais da área, ou devemos crer que processos perfeitos

não existem e partir para outra abordagem, conforme a estratégia adotada pela comunidade ágil?

Inexistindo a definição perfeita de um processo de desenvolvimento, o princípio ágil defende que cada equipe, em cada projeto, deve encontrar, por méritos próprios, sua dinâmica de trabalho. Não existem, portanto, regras preestabelecidas. Ou melhor, quase não existem. Uma regra básica é a da melhoria contínua. A cada ciclo de trabalho, deve-se refletir sobre o que foi feito, aprender com o que não funcionou (descartando-o, se for o caso) e potencializar o que estiver dando certo. Naturalmente, as primeiras iterações do projeto serão de grande aprendizado. Passados os primeiros ajustes, a tendência é de convergência do ritmo, das expectativas e das prioridades dos envolvidos, reforçando o potencial de sustentabilidade dos trabalhos.

▸ REFERÊNCIAS

AGILE ALLIANCE . Orlando: Agile Alliance, 2014. Disponível em:<http://www.agilealliance.org/>. Acesso em: 13 abr. 2014.

EMERGÊNCIA. In: DICIONÁRIO online de português. [S.l.]: 7Graus, 2014. Disponível em:< http://www.dicio.com.br/emergencia/>. Acesso em: 13 abr. 2014.

FRIED, J.; HANSSON, H.; LINDERMAN, M. Getting real: the smarter, faster, easier way to build a successful web application. [S.l.]: 37signal, 2006. Disponível em:<https://basecamp.com/books/Getting%20Real.pdf>. Acesso em: 13 abr. 2014.

INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL. Desenho industrial. Brasília: CGCOM, 2014.

KAHNEY, L. A cabeça de Steve Jobs. Rio de Janeiro: Agir, 2008.

KOCH, R. O princípio 80/20: o segredo de se realizar mais com menos. Rio de Janeiro: Rocco, 2001.

MANIFESTO para desenvolvimento ágil de software. [S.l: s.n.], 2001. Disponível em: <http://www.agilemanifesto.org/iso/ptbr/>. Acesso em: 02 abr. 2014.

PRINCÍPIO. In: DICIONÁRIO online de português. [S.l.]: 7Graus, 2014. Disponível em:<http://www.dicio.com.br/principio/>. Acesso em: 13 abr. 2014.

SIGNIFICADO de diagrama de Pareto. [S.l.]: 7Graus, 2014. Disponível em: <http://www.significados.com.br/diagrama-de-pareto/>. Acesso em: 13 abr. 2014.

WHAT does KISS stand for? [S.l.: s.n., 2000]. Disponível em: <http://people.apache.org/~fhanik/kiss.html>. Acesso em: 13 abr. 2014.