Diagnóstico de um Projeto Ágil

GESTÃO DE PROCESSO DE SOFTWARE

João Nuno Almeida, Universidade do Minho, A75209
João Miguel Araújo, Universidade do Minho, A75364
Victor da Cunha, Universidade do Minho, A75693

09/06/2018

Conteúdo

1 Introdução **3**

1.1 História do desenvolvimento de software 4

2 Princípios e Virtudes de um Desenvolvimento Ágil **7**

3 State-of-the-Art **9**

3.1 Introdução aos Modelos Ágeis 9

3.2 Modelos Ágeis Mais Utilizados 10

3.2.1 Extreme Programming (XP) 10

3.2.2 Scrum 12

3.2.3 Crystal Methods 14

3.3 Caracterização de um Projeto Ágil 15

4 State-of-the-Practice **17**

4.1 Técnicas de Diagnóstico 17

4.1.1 Teste da Nokia 17

4.1.2 Teste de Karlskrona 18

4.1.3 Teste dos Pontos 20

4.2 Proposta de Diagnóstico para Projetos Ágeis 21

4.2.1 Análise Pergunta a Pergunta 23

4.2.2 Sistema de Classificação 29

4.3 Exemplo de Aplicação do Método de Diagnóstico 30

5 Conclusão **31**

Referências **32**

Anexo **33**

Teste da Nokia 33

Teste de Karlskrona 35

Teste dos 42 Pontos 37

1 Introdução

O desenvolvimento de software e a demanda por este tipo de produtos não param de aumentar. Podemos datar o principio desta tendência ao início da década de 90. Naquele tempo, a grande dificuldade sentida por empresas, independentes, ou qualquer entidade que se propusesse a iniciar o desenvolvimento de um novo software era ultrapassar as primeiras etapas do processo, nomeadamente de planeamento do projeto. Tratam-se de decisões que necessitam de ser tomadas antes de se iniciar o desenvolvimento, e que consistem essencialmente na estruturação de uma divisão de tarefas entre os envolvidos, assim como na calendarização da ordem cronológica de conclusão (expectável) dessas tarefas ao longo do tempo disponível e hierarquização das mesmas.

As técnicas de planeamento existentes na altura, eram demasiado estritas no que dizia respeito a tudo o que se passava e que tinha de ser seguido após ser iniciado o desenvolvimento. Implicavam um seguimento demasiado severo do que foi definido antes do desenvolvimento começar, na fase de planeamento, sem conceder liberdade para mudanças significativas, que por esse motivo, naturalmente implicavam custos e atrasos vultosos quando tinham de acontecer obrigatoriamente por motivos alheios à entidade (modificações no desejo do cliente) ou não (abandono de elementos da equipa).

Foi então, que por volta do mesmo período surgiram os primeiros conceitos de desenvolvimento ágil, i.e., um tipo de técnica de planeamento muito mais flexível no que diz respeito à ordem pela qual as tarefas têm de ser executadas, e no espaço de tempo disponível para as concluir, mas que apresenta resultados substancialmente melhores no que diz respeito à taxa de sucesso dos projetos [1]. Trata-se de uma técnica de planeamento também ela, melhor preparada para diminuir o impacto negativo, em todo o processo, causado por eventuais mudanças de opinião e dos desejos dos clientes que contrataram o software, quer em questões como, por exemplo, mudança de requisitos, ou alteração de decisões de design em fases avançadas do desenvolvimento.

Atualmente, a maioria dos projetos assenta sobre uma técnica de planeamento do tipo ágil [2], e é por esse motivo que outro tipo de problema está a surgir.

O que infelizmente está a acontecer é que muitos projetos estão a apresentar taxas de sucesso baixas apesar de usarem métodos ágeis, um facto que vai contra o que foi acima dito. As entidades que desenvolvem os ditos projetos, culpam as taxas de sucesso diminutas no conceito de desenvolvimento ágil e no seu fracasso como técnica de desenvolvimento, mas após uma análise mais cuidada o que é possível constatar é que não foi a técnica em si que falhou mas sim a sua aplicação em determinado contexto.

Como é quase *moda*, o recurso a técnicas de planeamento ágeis nos dias de hoje, o que falha em maior parte dos projetos é compreender como aplicar a técnica e ser capaz de responder positivamente à questão *"Enquadrar-se-á o meu projeto numa metodologia ágil?"*

Sendo este o mote do nosso ensaio, tentaremos de seguida idealizar uma ferramenta de diagnóstico, que mediante os resultados das respostas a um conjunto de questões permitirá retirar conclusões acerca da correta aplicação, ou não, da metodologia ágil, no projeto em

questão. De seguida, analisaremos os conceitos envolvidos para classificar um desenvolvimento como *Ágil*. Falaremos mais aprofundadamente, acerca da história das metodologias ágeis, nomeadamente, pioneiros e as primeiras aplicações a projetos. Daremos exemplos dos tipos de modelos mais usados, como por exemplo o Scrum e em que medida são classificados como modelos ágeis. Iremos enumerar as características mais marcantes dum modelo ágil que o permitem classificar como tal, tudo para nos auxiliar a que, após este estudo, resulte do mesmo uma ferramenta de auto-avaliação, no formato de uma proposta protótipo que carecendo inicialmente de validado profissional, será posteriormente aplicada com autorização de uma entidade a um projeto de desenvolvimento de software seu, validando se possível este modelo para utilizações futuras, caso as conclusões retiradas e a ajuda prestada à entidade no diagnóstico sejam relevantes.

É importante ainda referir, que este tipo de ferramenta que nos propomos a desenvolver, não se trata de nada pioneiro na indústria. Aliás, iremos basear a nossa ferramenta em testes já existentes, e usados por diversas entidades que desenvolvem software. O nosso objetivo passa por criar algo que configure uma melhoria aos produtos atualmente usados, e cujo conceito é baseado nos testes já existentes, i.e., tal como já tinha sido referido acima, um simples questionário (o mais sucinto e claro possível) que mediante as respostas dadas permita sumarizar o estado atual do projeto ao qual o mesmo é aplicado, no que diz respeito à metodologia de planeamento seguida.

No público alvo deste ensaio, incluem-se estudantes que pretendam usar este ensaio futuramente como nota de estudo. Professores que pretendam preparar-se e basear os seus estudos em alguma informação ou tópico aqui presente. Entidades que desenvolvam software, pois serão os mesmos os mais interessados em usar, as conclusões e resultados deste ensaio, para os transformar em algo útil na prática.

1.1 História do desenvolvimento de software

Se muitos acreditam que as metodologias ágeis apareceram depois do Manifesto para o desenvolvimento ágil de software, estas, na realidade, apareceram nos meados dos anos 90 como reação contra os métodos clássicos, geralmente caracterizados por uma pesada regulamentação, e para definir novos métodos que seriam capazes de incorporar mais facilmente as evoluções industriais e tecnológicas dentro do processo como também as mudanças de ideais por parte dos *stakeholders* do projeto.

De facto, muitas das práticas ágeis não são novas, sendo que muitas delas foram baseadas no modelo de desenvolvimento iterativo, uma técnica introduzida em 1975 (*Vic Basili and Turner*, 1975 [3]). Para compreender como os modelos ágeis foram criados iremos a seguir apresentar os diferentes modelos que foram desenvolvidos e que queriam responder as necessidades que os modelos ágeis respondem.

Segundo Kent Beck[4], o modelo em cascata, criado em 1970 por W.W. Royce, foi o primeiro modelo de desenvolvimento criado para encontrar e responder as necessidades dos utilizadores. No modelo original, Royce dividiu o desenvolvimento de software em 7 etapas distinta e sequenciais. Estas etapas são, por ordem:

- O levantamento e especificação de requisitos, onde é elaborado um conjunto rígido de requisitos de software;
- A conceção ou ***design***, onde os requisitos são modelados de maneira a criar uma representação do software que se quer criar;
- Implementação, onde , seguindo os modelos criados na etapa anterior, o software é criado;
- Integração, onde os diversos componentes de software criado são reunidos;
- Teste e depuração, onde se verifica que o software criado corresponde as expectativas definidas;
- Instalação;
- Manutenção de software, onde o software é mantido ao longo da sua vida, removendo defeitos e adicionando funcionalidades.

Muitos acreditavam que este modelo seria o mais eficaz pois os requisitos eram bloqueados logo na primeira etapa, não sendo permitidas renegociações ou mudanças nestes, mas este modelo revelou-se inadequado pois, como dito anteriormente, os utilizadores ou clientes mudam de ideias, logo, se fosse necessário mudar um requisito noutra etapa mais avançada seria necessário voltar ao inicio, descartando tudo o que foi feito até ao momento, algo impensável. O próprio *Royce* [5], na sua apresentação do modelo em cascata, tinha-o definido como um risco e um convite para falhas.

Considerando os riscos do modelo em cascata, mas compreendendo que este introduzia boas ideais, novos modelos iterativos e incrementais foram desenvolvidos cujo foco foi dividir o ciclo de desenvolvimento em fases, evoluindo a partir do modelo em cascata (Beck, 1999a), tomando o processo que está atrás deste e repetindo-o ao longo do ciclo de desenvolvimento.

Os modelos incrementais tentaram reduzir o tempo de desenvolvimento dividindo o projeto em etapas incrementais sobrepostas. Como no modelo em cascata, todos os requisitos eram analisados antes do inicio do desenvolvimento, mas, os requisitos são depois divididos em funcionalidades individuais, que irão ser introduzidos em incrementos. O desenvolvimento de cada incremento pode ser sobreposto, visto que estes são independentes, ganhando-se tempo através de *multitasking*.

Se os modelos incrementais evoluíram tendo como objetivo reduzir o tempo de desenvolvimento, os modelos iterativos evoluíram para melhor manejar as mudanças de requisitos e gerir os riscos dos projetos. Assim, modelos como o desenvolvimento iterativo e o modelo em espiral (Boehm, 1988)[6] vieram a aparecer. Estes modelos identificam fatores de riscos de maneira planeada e estruturada em diversos pontos do processo, em vez de tentar mitigá-los à medida que aparecem.

O desenvolvimento iterativo divide o projeto em si em várias iterações com tamanho variável, sendo que cada iteração produz um produto utilizável e constrói-se sobre o código e a

documentação produzida em iterações anteriores. A primeira iteração começa com as funcionalidades básicas do software, sendo que cada iteração que a segue irá adicionar novas funcionalidades ao produto. Cada iteração segue o modelo em cascata, começando o processo com a análise dos requisitos que serão introduzidos, seguido pelo design, a implementação e finalmente o teste do software produzido. O desenvolvimento iterativo lida com as mudanças de requisitos bloqueando-os no inicio de cada iteração, isto é, estes só serão modificados ou introduzidos na próxima iteração caso o processo de análise já esteja fechado na iteração em curso. Mesmo sendo necessário um conjunto de requisitos candidatos para a próxima iteração estes são facilmente modificados, permitindo melhor elasticidade na adoção de novas tecnologias e na integração das mudanças de opinião dos *stakeholders*.

O modelo em espiral também evita definir todo o sistema no início do projeto mas, enquanto que o modelo de desenvolvimento iterativo prioriza a implementação das funcionalidades em cada iteração, o modelo em espiral prioriza a implementação dos requisitos em função dos riscos atribuídos a estes. Estes dois modelos ofereceram um grande salto em agilidade comparado com o modelo em cascata mas muitos acreditaram que isto não era suficiente para poder responder às necessidades e à constante evolução do mundo dos negócios, pois estes modelos possuem fases de planeamento e analisa muito longas, continuando a focalizar-se sobre uma documentação extensa do projeto e dos seus componentes. Estes dois pontos fazem com que estes modelos não sejam, pelas definições atuais, considerados ágeis.

2 Princípios e Virtudes de um Desenvolvimento Ágil

Em 2001, quando o *Manifesto for Agile Software Development* foi lançado com autoria de 17 indivíduos, foram acentuados um conjunto de princípios bastante claros, que no fundo esclarecem o principal motivador para esta mudança de paradigma que tem ocorrido no desenvolvimento de *software* [7]. Aqui passamos a explicitar o seu conteúdo:

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> **Individuals and interactions** over processes and tools
>
> **Working software** over comprehensive documentation
>
> **Customer collaboration** over contract negotiation
>
> **Responding to change** over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more.

O desenvolvimento no paradigma ágil pretende adaptar-se aos novos tempos, em que os ciclos de vida do *software* são cada vez mais curtos e em que há uma exigência tanto por parte dos utilizadores como do próprio mercado e indústria para inovar mais rapidamente para competir, num espaço cada vez mais saturado com empresas e indivíduos que pretendem suceder e chegar ao topo.

Métodos de desenvolvimento passados eram demasiado rígidos (a título de exemplo, e para ajudar o leitor a visualizar melhor o pretendido, o que for expresso neste parágrafo terá como base a técnica de *Waterfall*), focados sobretudo em estabilidade e rigidez, adaptados a um paradigma mais empresarial que valorizava sistemas monolíticos que seriam usados durante anos. O processo de desenvolvimento envolvia um conjunto de passos rígidos que seriam seguidos à risca, com limites fixos para cada uma das fases do processo e guias específicos para a documentação de cada componente envolvido no produto. Caso fosse necessário alterar algum aspeto determinado em fases anteriores, seria preciso reiniciar o projeto desde essa fase, pois não havia margem para erros ou alterações. Isto, no entanto, não é para dizer que este tipo de desenvolvimento é completamente inadequado para todo o tipo de projetos efetuados hoje em dia. Continua a haver procura por certo tipo de *software* que de facto exige este tipo de cuidado com todas as fases do processo, mas a ideia que se pretende transmitir é que ao longo dos tempo, este paradigma de desenvolvimento deixou de se adequar a um grande número de projetos.

Com metodologias ágeis prezam-se sobretudo ciclos de desenvolvimento iterativos, isto é, componentes do *software* são trabalhados durante um curto período definido de tempo, com todas as fases associadas de planeamento, *design*, desenvolvimento e teste. Isto permite um maior foco por parte da equipa de desenvolvimento nos aspetos essenciais de cada parte da aplicação, apanhar defeitos e *bugs* no produto mais cedo, assim como aplicar quaisquer alterações de funcionalidade exigidas pelo cliente numa fase mais embrionária do desenvolvimento. Permite que o projeto seja mais tolerante a mudanças quer externas, quer internas.

Este principio da iteração no desenvolvimento permite também demonstrar um produto mais *barebones*, isto é, a mostrar ao cliente antecipadamente o *software* com as funcionalidades mais básicas implementadas durante essa iteração. Chamam-se a esses produtos *deliverables*. Estipula-se que a entrega destes ao fim de cada uma das iterações fomenta a confiança do cliente na equipa de desenvolvimento, do próprio projeto, e permite uma resposta imediata a quaisquer alterações que possam surgir na análise dos mesmos.

Tudo isto exige, claro, comunicação constante, não só entre todos os elementos da equipa de desenvolvimento, como com o próprio cliente, para garantir que no final seja entregue um produto de *software* de qualidade que satisfaça todas as partes interessadas.

No fundo isto sintetiza os princípios fundamentais do desenvolvimento ágil, e permite-nos retirar também as virtudes, como a transparência, comunicação, colaboração e eficácia. Passámos aqui a explicitar também os princípios que se encontram estipulados no manifesto ágil, mencionado anteriormente, que no fundo esclarecem todos os pontos aqui discutidos nesta secção.

> A nossa maior prioridade é, desde as primeiras etapas do projeto, satisfazer o cliente através da entrega rápida e contínua de software com valor.
>
> Aceitar alterações de requisitos, mesmo numa fase tardia do ciclo de desenvolvimento. Os processos ágeis potenciam a mudança em benefício da vantagem competitiva do cliente.
>
> Fornecer frequentemente software funcional. Os períodos de entrega devem ser de poucas semanas a poucos meses, dando preferência a períodos mais curtos.
>
> O cliente e a equipa de desenvolvimento devem trabalhar juntos, diariamente, durante o decorrer do projecto.
>
> Desenvolver projectos com base em indivíduos motivados, dando-lhes o ambiente e o apoio de que necessitam, confiando que irão cumprir os objectivos.
>
> O método mais eficiente e eficaz de passar informação para e dentro de uma equipa de desenvolvimento é através de conversa pessoal e directa.
>
> A principal medida de progresso é a entrega de software funcional.
>
> Os processos ágeis promovem o desenvolvimento sustentável. Os promotores, a equipa e os utilizadores deverão ser capazes de manter, indefinidamente, um ritmo constante.
>
> A atenção permanente à excelência técnica e um bom desenho da solução aumentam a agilidade.
>
> Simplicidade – a arte de maximizar a quantidade de trabalho que não é feito – é essencial.
>
> As melhores arquiteturas, requisitos e desenhos surgem de equipas auto-organizadas.
>
> A equipa reflete regularmente sobre o modo de se tornar mais eficaz, fazendo os ajustes e adaptações necessárias.

3 State-of-the-Art

Nesta secção iremos discutir, analisar e explicar o que são afinal em concreto, metodologias ágeis explicitando as suas características mais próprias. Iremos enumerar aqueles que são os modelos ágeis mais comuns e mais aplicados em situações reais assim como os seus prós e contras. Por fim, encerraremos esta secção com uma caracterização daquele que deve ser o ambiente de desenvolvimento dentro do domínio da entidade que está encarregue de desenvolver o software para que a aplicação de uma metodologia ágil reverta nos resultados esperados.

3.1 Introdução aos Modelos Ágeis

Para perceber qual é o significado de uma metodologia ágil e desse modo criar ferramentas para detetar a sua presença, é preciso perceber afinal o que significa ser ágil, no contexto do planeamento dum projeto. Como tudo na ciência, as definições mudam consoante os seus autores, no entanto, todas elas têm a intenção de descrever o mesmo conceito mas tomando caminhos diferentes. Segundo *Jim Highsmith*, a capacidade de ser ágil, é a capacidade de mudar rapidamente, mudar frequentemente e de apresentar resultados rapidamente. De modo a alcançar estas três características, independentemente da técnica ágil que a entidade aplique, é importante que estejam presentes os ideais de desenvolvimento iterativo, comunicação entre os envolvidos e redução (ou até mesmo total eliminação) de artefactos intermédios [8].

O desenvolvimento iterativo permite que a equipa de desenvolvimento se adapte de maneira rápida, facilitada e pouco custosa, a mudanças na visão do cliente, por exemplo, mudanças nos requisitos, alcançando deste modo a capacidade de mudar rapidamente e frequentemente.

O ideal de comunicação entre os interessados dentro da equipa de desenvolvimento, traduz-se na capacidade de propor novas ideias, ou rumos que o projeto poderá vir a seguir, e tomar decisões em concreto acerca dessas ideias sem criar burocracias e sem despender tempo significativo e importante.

A redução de artefactos intermédios, consiste na eliminação de pequenos componentes desenvolvidos por elementos da equipa em paralelo com o ramo de desenvolvimento principal, que pouco ou nada de relevante irão acrescentar ao produto final. A pronta identificação destes artefactos e consequente exclusão dos mesmos da etapa de desenvolvimento, irá permitir que os esforços dos membros da equipa sejam alocados noutras tarefas mais importantes, acelerando a conclusão do produto de software.

Apesar de estes serem os ideais, que em termos teóricos permitem classificar um projeto de ágil, é defendida dentro da comunidade científica a posição de que não basta seguir estas diretrizes, para que o dado projeto seja incluído na categoria dos projetos ágeis. Caso os envolvidos não estejam dispostos a encarar o projeto como ágil, o que irá acontecer é uma sabotagem mental, na forma de aplicação dos conceitos ágeis com uma intenção puramente iterativa e monolítica, transformando a metodologia realmente aplicada no projeto e deturpando as propriedades emergentes observáveis no mesmo. Segundo investigadores como *Jim Highsmith* e *Cockborn*, a única mudança verificada nas metodologias de abordagem de planeamento

desde os primórdios do desenvolvimento de software até aos dias de hoje, é a aceitação dos ideais ágeis. A metodologia já existia, o que mudou foi o reconhecimento da técnica como fonte de sucesso.

Imediatamente, e sem grande esforço adicional, podemos nos questionar, acerca da presença destes três ideais no nosso projeto e de maneira imediata aferir a presença de uma metodologia ágil.

3.2 Modelos Ágeis Mais Utilizados

Existem diversos métodos ágeis em uso nos dias de hoje. No entanto, poucos se destacam verdadeiramente pela sua singularidade, tratando-se essencialmente de variantes de outros métodos mais conhecidos, onde o que muda acaba por se tratar essencialmente das práticas que a entidade deve seguir para atingir os resultados comuns. Dentro dos métodos mais conhecidos, achamos relevante para o contexto deste ensaio destacar os seguintes [8]:

1. *Extreme Programming (XP)*;
2. Scrum;
3. *Crystal Methods*;
4. *Feature Driven Development*;
5. *Lean Development*;
6. *Dynamic Systems Development Methodology (DSDM*.

Devido à sua importância e uso cada vez maior na indústria decidimos focar-nos essencialmente nos métodos *Extreme Programming (XP)* e Scrum. Pela ajuda que dá na questão na comunicação e dinâmica de grupo, com técnicas próprias para tal, decidimos igualmente explicar de maneira mais aprofundada em que é que consiste o método *Crystal Method*. Segundo a nossa pesquisa, estes tratam-se dos métodos melhor documentados, devido a uma menor subjetividade na sua aplicação em comparação com todos os outros acima enumerados.

3.2.1 Extreme Programming (XP)

De todos as técnicas ágeis que surgiram nos anos mais recentes, a técnica de *Extreme Programming* foi aquela que mais adeptos criou no decorrer dos anos [8]. A técnica foi introduzida por *Beck and Jeffries* por volta do ano de 1998, e foi altamente popularizada por *Beck* no seu livro *Extreme Programming Explained: Embrace Change* lançado em 1999. A intenção de *Kent Beck* com o seu livro passou por esclarecer, quais as regras que devem ser seguidas de modo a aplicar esta técnica. A técnica *Extreme Programming* deve o seu nome precisamente a este aglomerado de regras, que foram vistas por todos como demasiado radicais (*Extreme*) para a época. A presença dos seguintes conceitos em determinado projeto, pode ser indicativo da presença de uma metodologia ágil no mesmo.

1. **Plano Inicial** - Todo o projeto basear-se-á num conjunto de iterações, que no seu todo resultarão no produto de software final. Isto permite uma redução significativa nos custos inerentes às mudanças de requisitos em fases adiantadas do projeto. No início de cada iteração, as partes interessadas reúnem e passam de novo por todas as fases comuns de escrita e priorização de requisitos, como se de um projeto novo se tratasse, respeitando-se e documentando os requisitos encontrados;

2. **Pequenos Lançamentos** - É importante que passado um número específico de iterações previamente negociado, exista um produto, não necessariamente completo, mas funcional. Após este primeiro lançamento, de x em x tempo devem ir sendo lançados novas versões, sempre funcionais. A componente mais importante do desenvolvimento de software é o código. Se não houver código funcional o produto não tem qualquer valor. É importante que, mesmo que seja difícil explicitar uma ideia e coloca-la em prática, o programador tente recorrer a uma solução mais simples para mostrar o que pretende, e tentar chegar à solução ideal a partir desse protótipo;

3. **Metáfora** - Deve ser definida uma metáfora entre as partes interessadas, i.e., um ou mais termos de comparação a partir dos quais o software será moldado. Uma das metáforas mais comuns no mundo da engenharia de software é a comparação do projeto e do produto a um automóvel ou à construção de uma casa;

4. **Design Simples** - As decisões de design, devem manter-se o mais simples possíveis e devem organizar o melhor que conseguirem a lógica do sistema. Isto irá permitir extinguir dependências complicadas de serem geridas à medida que a complexidade do projeto vai avançando;

5. **Testes** - Os testes para testar a validade e correção do código, devem ser escritos antes do próprio código. No final de cada iteração, um conjunto de testes específicos para a mesma devem ser verificados. Os testes unitários deverão testar todas as funções e funcionalidades envolvidas no códigos. Testes de aceitação deverão verificar se a perceção do significado dos requisitos é a mesma para todas as partes interessadas;

6. **Refactoring** - À medida que o design vai evoluindo, deve ser garantido que o mesmo se mantém o mais limpo possível, e que não introduz complexidade nem erros indesejados a secções de código "limpas"(*"Code Smells"*);

7. **Programação a Pares** - O método mais eficaz para programar é a pares;

8. **Integração Contínua** - Todos os engenheiros devem introduzir o produto do seu trabalho, e das iterações pelas quais ficaram responsáveis, num sistema integrado e partilhado por todos frequentemente, sem afetar a funcionalidade desse sistema, nem prejudicar a sua tolerância a testes;

9. **Propriedade Coletiva** - O código pertence a todos os envolvidos, e cada um pode fazer as mudanças que entender a qualquer altura;

10. **Cliente On-Site** - O cliente está sempre presente no processo de desenvolvimento, disponível para responder a perguntas, para validar *outputs* e para monitorizar o progresso do produto em qualquer altura;

11. **Semanas de 40 Horas** - Os requisitos e exigências de cada iteração deverão ser distribuídos de modo a que não ultrapassar um total de mais de 40 horas semanais de desenvolvimento do produto;

12. **Workspace Aberto** - Todos os envolvidos deverão trabalhar em conjunto, nas mesmas condições respeitando-se mutuamente e comunicando livremente, facilitando o esclarecimento de dúvidas e aumentando o sincronismo (garantir que todos estão *"na mesma página"*).

A combinação das 12 regras acima numeradas, irá resultar no aparecimento de 5 propriedades emergentes muito semelhantes aquelas que são (e devem ser) observáveis num projeto que adote uma metodologia ágil: comunicação, simplicidade, *feedback*, coragem e qualidade de trabalho. Comprovando assim, que a técnica XP segue corretamente a *framework* de desenvolvimento ágil.

No entanto, naturalmente é necessário estarem presentes determinados cenários no domínio do contexto de desenvolvimento, para que seja possível obter os resultados esperados da aplicação da técnica XP. A não presença de um ou mais dos seguintes cenários irá resultar num ou mais conflitos com as regras acima enumeradas.

1. **Tamanho da Equipa** - A equipa não deve ser menor do que 2 elementos nem maior do que 10 elementos;

2. **Tamanho da Iteração** - O mínimo de tempo para cada iteração deve ser de pelo menos 2 semanas;

3. **Equipas Distribuídas** - Não deve haver lugar a equipas distribuídas, todos devem trabalhar juntos e em conjunto.

3.2.2 Scrum

Juntamente com a técnica XP, a técnica Scrum é a mais usada no processo de desenvolvimento de software. Foi descrita formalmente pela primeira vez, em 1996 por *Ken Schwaber*, como sendo uma técnica que aceita a ideia de que o desenvolvimento de software possa vir a ser imprevisível e por esse motivo é pouco aconselhável perder tempo a definir rigorosamente um plano de desenvolvimento meticuloso. O termo é emprestado do Rugby, onde Scrum se trata de uma tática ofensiva, que consiste na junção dos jogadores de uma equipa em bloco, com o intuito de ultrapassar mais facilmente a equipa adversária e avançar no campo o máximo de distância possível na posse da bola [8].

O ciclo de desenvolvimento de um projeto que adote esta técnica, estará dividido em iterações, denominadas *sprints*. Por sua vez, cada *sprint* estará dividido nas seguintes fases:

1. **Planeamento Pré-Sprint** - O conteúdo retirado das reuniões pré-desenvolvimento, por exemplo, requisitos é guardado, nesta fase, num documento denominado *release backlog*. De seguida, é necessária a realização de uma reunião com os elementos da equipa de desenvolvimento, onde será estabelecido um acordo entre todos, acerca de quais as funcionalidades que serão selecionadas para implementação no sprint seguinte. Essas funcionalidades, derivam dos requisitos recolhidos, e serão colocadas num documento denominado *sprint backlog*, que consiste numa coleção de funcionalidades priorizadas, a completar até ao final do sprint em curso. No *sprint backlog* deve, ser feita referência ao objetivo do sprint em questão, relembrando todos os envolvidos da sua importância. O nível de detalhe a considerar na implementação, deve também constar deste documento.

2. **Sprint** - O documento *sprint backlog* é entregue a todos os envolvidos e a partir deste momento, todo o seu conteúdo é imutável até ao final do sprint. Cada um dos envolvidos, escolhe quais são as tarefas que pretende desenvolver e é encorajada a comunicação entre membros, na forma de reuniões diárias (usualmente matinais) onde é discutido o estado do sprint em curso, onde são retiradas dúvidas que possam entretanto ter surgido e onde se estabelece diálogo com o cliente, mantendo o mesmo a par do ponto da situação.

3. **Reunião Pós-Sprint** - Após o término de cada sprint é marcada uma reunião, onde é discutido o progresso do projeto e onde é demonstrado o sistema no seu estado atual. O sistema deve portanto manter-se sempre funcional, ao longo de todo o processo. Nenhum componente desenvolvido por um determinado elemento da equipa deve colocar em causa a funcionalidade do sistema, onde serão acrescentados os resultados do sprint.

Dito isto podemos sumarizar os princípios que deverão ser cumpridos de modo a alcançar uma metodologia ágil no projeto em questão. Segundo *Ken Schwaber*, para que o Scrum tenha sucesso, quando aplicado a um projeto é importante que:

1. Equipas de trabalho pequenas, de modo a maximizar as componentes de comunicação, partilha e sincronismo entre todos.

2. Capacidade de adaptabilidade por parte dos envolvidos, a mudanças tecnológicas, por exemplo paradigma de programação, em prol da qualidade do produto final.

3. Frequente criação de executáveis do sistema, que podem ser executados para realização de testes ou demonstrações.

4. Partição do trabalho a desenvolver de forma justa e equilibrada, fomentando o trabalho a pares entre os envolvidos.

5. Preocupação constante em testar e documentar frequentemente o produto.

6. Capacidade de dar o produto como concluído em qualquer altura necessária, reforçando a ideia de manutenção de um produto constantemente funcional.

No entanto, naturalmente é necessário estarem presentes determinados cenários no domínio do contexto de desenvolvimento, para que seja possível obter os resultados esperados da

aplicação da técnica Scrum. Se um ou mais destes cenários estiver em falta, as fases acima descritas da técnica Scrum estarão comprometidas e o projeto poderá desviar-se do caminho do sucesso.

1. **Tamanho da Equipa** - As equipas devem ter idealmente 7 elementos, nos quais se devem incluir um engenheiro de software, um engenheiro de qualidade e um responsável por documentação do sistema.

2. **Tamanho do Sprint** - A duração de cada sprint deve situar-se entre 1 a 6 semanas no máximo.

3. **Equipas Distribuídas** - O Scrum aceita o conceito de sprints paralelos, logo o conceito de equipas distribuídas, trabalhando cada uma no seu sprint respetivo não afetará negativamente o sucesso do projeto.

3.2.3 Crystal Methods

Esta técnica foi descrita pela primeira vez no início da década de 90, coincidindo com a época onde começaram a surgir os primeiros problemas relacionados com a rigidez das técnicas iterativas no seguimento do planeamento. *Alistair Cockburn* foi o primeiro a descreve-la, pois acreditava que o maior entrava no desenvolvimento de software passava pela falta de comunicação entre membros da equipa de desenvolvimento. Baseia-se no princípio de que a matéria escrita dificulta a compreensão dos conteúdos a comunicar [8].

Alistair Cockburn inspirou-se nas faces de um cristal para representar cada uma das iterações presentes no desenvolvimento de software. Inspirou-se na cor que o cristal apresenta para representar qual versão ágil presente no projeto. Como podemos visualizar na Figura 1, um projeto representado por um cristal transparente é o mais ágil que pode existir. Seguem-se as cores amarelo, laranja e vermelho para a representação de um projeto no limiar do ágil, no entender de *Cockburn*.

O critério usado para definir a cor do cristal que melhor caracteriza determinado projeto é unicamente o número de pessoas envolvidas na equipa de desenvolvimento. Visto que a única preocupação deste método é adotar uma maior flexibilidade na comunicação entre membros, quantos mais membros, mais difícil será o sucesso no processo de comunicação e menos ágil será o projeto recebendo uma cor mais intensa na escala criada por *Cockburn*. O que usualmente acontece nos dias de hoje, é a adaptação desta escala, para avaliação de outros aspetos sem ser a comunicação, por exemplo, a qualidade dos testes. Esta adaptação permite a uma equipa ter tantos cristais, quantos aspetos técnicos pretende avaliar. A presença destes cristais permite à equipa ter uma noção rápida e intuitiva do estado do processo.

No entender de *Alistair Cockburn* cada cristal começa o seu ciclo de vida com um determinado número de faces (comuns a todos os processos de desenvolvimento de software, como por exemplo, qualidade de testes, tamanho da equipa ou decisões de design) e cada vez que é acrescentada uma nova restrição é aumentado o número de faces do cristal e consequentemente a sua complexidade.

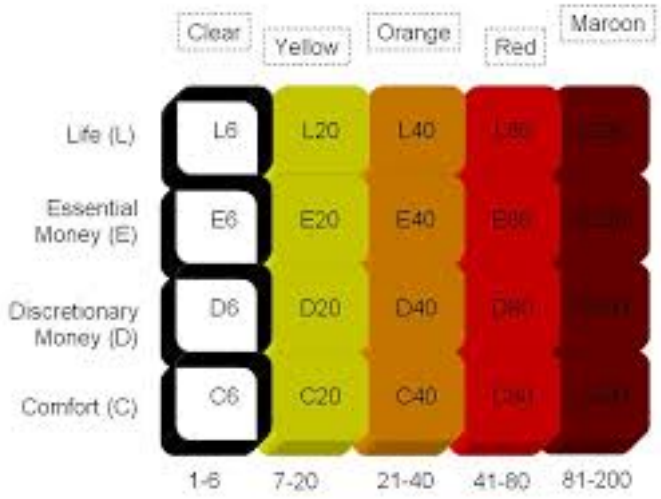

Figura 1: Escala de *Crystal Methods* definida por *Alistair Cockburn*

Os cenários mais favoráveis, para que esta técnica resulte e as propriedades emergentes da *framework* ágil surjam, são descritos por *Aistair Cockburn* da seguinte forma:

1. **Tamanho da Equipa** - O tamanho da equipa acaba por ser ilimitado, mas pode vir a influenciar a escala do cristal atribuída, dependendo das capacidades de comunicação dentro da equipa.

2. **Tamanho da Iteração** - O tamanho de cada iteração varia desde 1 semana até 4 meses para iterações mais complexas.

3. **Equipas Distribuídas** - Está presente o conceito de equipas distribuídas no recurso a esta técnica.

4. **Criticalidade do Sistema** - Suporta falhas que resultem da perca de conforto durante o ciclo de desenvolvimento, falhas em orçamentos descritivos, falhas em orçamentos fundamentais para o desenvolvimento e falhas no ciclo de vida do próprio projeto.

3.3 Caracterização de um Projeto Ágil

A principal característica que se verifica, após tomarmos conhecimento das práticas que cada técnica defende é a existência de um conjunto de propriedades emergentes (comunicação, simplicidade, *feedback*, coragem e qualidade de trabalho) que irão surgir, independentemente de qual técnica seja escolhida.

A diferença entre escolher uma técnica ou a outra, passa pelas consequentes mudanças que cada uma das técnicas vai exercer nas práticas quotidianas da equipa de desenvolvimento. A empresa deverá ter consciência de que, em algumas situações, as mudanças serão radicais e se calhar não compensam a adoção de uma metodologia ágil. Diversos fatores, como a volatilidade dos requisitos, irão definir ou não a necessidade de adotar uma metodologia ágil.

Um possível auto-diagnóstico, informal, que as empresas e em específico as equipas de desenvolvimento, podem realizar para perceber ou não a necessidade de uma metodologia

ágil, será questionarem-se se têm equipas de desenvolvimento de qualidade e se os clientes estão totalmente dedicados ao projeto, duas exigências fundamentais para a metodologia ágil resultar. Uma resposta negativa a uma destas perguntas, basta para excluir desde logo, a possibilidade de adotar esta metodologia, ou para as empresas perceberem se estão a usar corretamente o paradigma.

Uma hipótese não excluída pela comunidade científica, passa pela adoção parcial destes cenários, até que haja estrutura necessária em ambas as partes (equipa de desenvolvimento e clientes) para adotar o método na totalidade.

4 State-of-the-Practice

Enquanto que na secção anterior discutimos o aspeto mais teórico de toda a componente do desenvolvimento ágil, nesta secção iremos entrar na parte mais prática da metodologia, isto é, casos de estudo da aplicação das várias técnicas, métodos de teste para verificar a correta aplicação e implementação de todos os aspetos e um exemplo concreto aplicado por nós para a avaliação de um projeto como sendo ágil ou não.

4.1 Técnicas de Diagnóstico

A capacidade de diagnóstico do desenvolvimento a ocorrer numa equipa para verificar se de facto segue princípios ágeis constitui um elemento extremamente importante em qualquer projeto que se auto-intitule como ágil.

Existe a ocorrência de muitos casos em que equipas de desenvolvimento proclamam que estão a utilizar uma metodologia ágil para o desenvolvimento do seu produto, e aquando uma análise mais cuidada, podemos concluir que de facto se encontram maioritariamente a utilizar, por exemplo, a técnica *Waterfall* com elementos de *Agile* (frequentemente denominada de *Agilefall*) ou até mesmo *Agile* sem grande planeamento, com umas poucas pessoas a tomar as rédeas do projeto (também frequentemente denominada como *Cowboy Agile*).

Portanto, dentro deste domínio, identificámos algumas técnicas de diagnóstico que consideramos relevantes e que são utilizadas pela indústria, nomeadamente o Teste da Nokia, mencionado na subsecção a seguir [9]. Está claro que existem muitos mais métodos de avaliação da "agilidade"de um dado projeto. Os métodos de teste que aqui apresentamos representam apenas uma amostra que consideramos relevante para o efeito.

De notar que apesar das propostas de diagnóstico que iremos apresentamos a seguir, nenhuma em específico consegue cobrir todos os aspetos do desenvolvimento ágil e colocá-los numa forma mensurável. Representam, no entanto, técnicas concretas e práticas de efetuar um diagnóstico à equipa e ao projeto, que normalmente servem como bom indicador e permitem que o desenvolvimento seja bem direcionado.

4.1.1 Teste da Nokia

A empresa finlandesa mais conhecida mundialmente pelos seus telemóveis, utiliza metodologias ágeis no desenvolvimento dos seus vários produtos, mais concretamente Scrum. Lançado em 2007 pela *joint-venture* entre a Nokia Networks e a Siemens, este formulário permite testar se de facto o desenvolvimento que está a ser levado por uma equipa pode ser classificado de ágil, com um conjunto simples e claro de questões. Também existe um conjunto de questões para validar se a equipa também se encontra a utilizar mais especificamente a metodologia Scrum, mas para o nosso caso só nos interessa o primeiro conjunto de questões.

São no total 7 questões, todas de escolha múltipla [10]. Aqui passamos a exemplificar uma delas:

Questão 1 - Iterações:

a) Sem iterações

b) Iterações > 6 semanas

c) Tamanho variável < semanas

d) Iterações fixas que duram 6 semanas

e) Iterações fixas que duram 5 semanas

f) Iterações 4 semanas ou menos

Cada resposta tem a si associada uma cotação e, como se pode adivinhar, as primeiras respostas valem menos que as últimas. De facto, a resposta a) vale 0 pontos e a resposta f) vale 10 pontos. Quanto maior for a pontuação global, melhor se pode afirmar que a equipa em questão está a seguir metodologias ágeis.

Já em relação à questão escolhida, podemos claramente ver que o teste especifica que qualquer projeto que entre no domínio da filosofia ágil deve ter ciclos iterativos não superiores a 4 semanas. Isto corresponde à escala temporal que a Nokia considera aceitável, com base nas suas experiências, e que se adequam a uma esmagadora maioria dos projetos.

Em anexo [10] iremos colocar todas as questões que englobam o teste.

4.1.2 Teste de Karlskrona

O teste de *Karlskrona* surgiu pela primeira vez em 2008 na Suécia. Foi criado por um aglomerado de empresas suecas e alemães, cujo a área de operação era precisamente, o desenvolvimento de software. Proclama-se como um teste de auto-avaliação, que pretende avaliar as práticas diárias da empresa testada, no que diz respeito à metodologia usada para o desenvolvimento.

Consiste em 11 perguntas, de escolha múltipla, com quatro opções de resposta possíveis [11]. Mediante as respostas dadas nas 11 questões, é possível inferir através de uma escala definida pelos criadores, o ponto de situação no que diz respeito à metodologia usada pela entidade que desenvolve o software, no momento da realização do teste.

A escala acaba por ter a sua cota parte de subjetividade, visto que foi criada baseada na opinião dos autores do teste, e o que os mesmos entendem como sendo os aspetos necessários para que seja observável uma metodologia ágil num projeto.

Após responder ao teste, existem 5 categorias onde o projeto se pode inserir: *Waterfall*, 3 fases de transição *("primeiros passos", "meio caminho", "quase lá")* e ágil. A inserção do projeto numa destas categorias, varia de acordo com as respostas dadas às questões e a pontuação atribuída a cada uma.

O sistema de pontuação, implica a correta formatação das opções de resposta no próprio enunciado. Como podemos ver na Figura 2, as opções de resposta que manifestam a presença de metodologias não ágeis, são regra geral, colocadas na coluna do lado esquerdo do enunciado. As opções de resposta que manifestam a presença de metodologias ágeis, são regra geral, colocadas na coluna do lado direito do enunciado. Por este motivo, questões cuja

opção de resposta dada se encontra na coluna esquerda, são cotadas com 0 pontos e questões cuja opção de resposta dada se encontra na coluna direita, são cotadas com 1 ponto. O somatório das cotações finais das 11 questões será usado para inserir o projeto testado numa das 5 categorias acima referidas como é visível na Figura 3. É aconselhado aos tomadores do teste que o repitam novamente, passados alguns meses, para confirmar a manutenção da metodologia ágil no projeto, ou para aferir a evolução do projeto, caso o mesmo não tenha sido classificado como ágil no último teste realizado.

1. Do you regularly deliver fully working software (documented and tested)?
   ☐ No, we wait until everything is finished ☐ We gradually improve software and tests
   ☐ We deliver software and fix issues later ☐ We gradually improve and show to customer

Figura 2: Questão 1 do enunciado do teste

Survey evaluation: Each cross on the right column counts one point, the left column counts zero (max one point per question). Calculate the average amount of points for the whole team and look up the final grade below. Discuss the results with the team, repeat the test in a few months.

0-2 points = grade 1
3-4 points = grade 2
5-7 points = grade 3
8-9 points = grade 4
10-11 points = grade 5

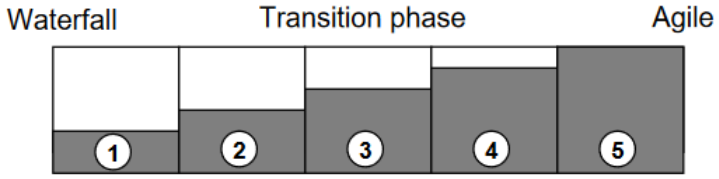

Figura 3: Auto-avaliação e análise de resultados

Naturalmente não existe margem para analisar todas as 11 questões neste ensaio. No entanto, usando o exemplo visível na Figura 2, da primeira questão do teste de diagnóstico de *Karlskrona*, podemos evidenciar a presença de práticas mais associadas a metodologias não ágeis nas opções de resposta da coluna esquerda, e ações e conceitos ágeis na coluna da direita. A questão pergunta se a equipa de desenvolvimento entrega software funcional regularmente (*"Do you regularly deliver fully working software (documented and tested)?"*, e é notória para quem conhece o *modus operandis* das metodologias ágeis a presença de conceitos e princípios ágeis, por exemplo na opção de resposta: *"We gradually improve and show to customer"*, que vai de encontro às políticas da técnica XP ou Scrum, de que o sistema deve manter-se funcional em todos os momentos visto que um produto não funcional, não tem valor absolutamente nenhum.

O remanescente do teste irá ser colocado nos anexos deste ensaio.

4.1.3 Teste dos Pontos

Este teste foi criado em 2013 com base no teste da Nokia, mencionado atrás, e pretende elaborar em cima do que já foi feito com este. Pretende fornecer um conjunto mais abrangente de tópicos que, segundo o autor, encapsulam melhor os princípios fundamentais que se devem seguir com a filosofia ágil, já que o teste da Nokia foca-se sobretudo em desenvolvimento com Scrum, como aludido há duas secções, e também tem um conjunto demasiado minimalista de questões.

Todos os elementos envolvidos no desenvolvimento devem preencher o teste, quer sejam os próprios *developers* como o(s) cliente(s). Passamos aqui a exemplificar alguns dos tópicos apresentadas:

1. A equipa tem o poder de tomar decisões.
2. A equipa encontra-se organizada e não precisa que a gestão lhe defina os objetivos a atingir.
3. A equipa compromete-se e toma responsabilidade pela entrega e está preparada para intervir em qualquer tarefa que a ajude a atingir o seu objetivo.

Em cada afirmação o indivíduo que estiver a responder ao teste deve assinalar com um 0 ou 1, em que 0 indica que a afirmação não se aplica ao estado atual do projeto e 1 aplica-se. Por exemplo, se o indivíduo não sentir que a equipa se encontra organizada, responde a 0 a afirmação 2 apresentada. Depois de todos os tópicos terem sido abordados, são somados todos os pontos atribuídos. Um teste deste tipo completamente preenchido só com uns perfaz 42 pontos (daí o título) [12]. No final, depois de todos os envolvidos no projeto terem preenchido o teste é feita a média com todos os resultados.

Tal como nas secções anteriores, o teste em completo será colocado no anexo deste ensaio.

4.2 Proposta de Diagnóstico para Projetos Ágeis

Neste secção iremos apresentar a nossa proposta de teste de diagnóstico de projetos ágeis. O nosso principal foco, inicialmente, foi tentar perceber os conceitos teóricos de uma metodologia ágil e as práticas quotidianas que cada uma das técnicas requer para que da sua aplicação resulte o sucesso do projeto e no surgimento das propriedade emergentes: comunicação, simplicidade, *feedback*, coragem e qualidade de trabalho. A partir daquelas que são as exigências de cada técnica em particular, e de algumas exigências comuns a todas as técnicas elaborámos um teste em formato de questionário, o mais sucinto possível, que no nosso entender pode ser respondido rapidamente e sem grande exigência, e que permita dar à equipa de desenvolvimento uma perspetiva da correta, ou não, aplicação da metodologia ágil ao seu projeto na forma de algo mensurável e com termos comparativos. Criamos questões o mais gerais possível para cobrir o máximo de técnicas possível e para não estender demasiado o tamanho do questionário em si. Cada uma destas questões, interroga as partes interessadas diretamente acerca da presença dos cenários necessários, no quotidiano de desenvolvimento de software, (acima descritos para cada técnica) para que a *framework* ágil apresente os resultados expectáveis e esteja efetivamente em utilização, resultando no surgimento das propriedades emergentes acima descritas.

A proposta por nós criada consiste em 12 questões, cada uma com 4 opções de resposta possíveis. As perguntas são curtas, à semelhança do teste de *Karlskrona*, e as opções resposta no nosso entender são diretas. É fácil o tomador do teste rever-se em pelo menos uma delas. O sistema de classificação consiste numa escala de 12 (pontuação mínima possível) a 100 (pontuação máxima) pontos, onde após a conclusão do teste, o tomador poderá inserir-se num de três grupos, consoante o intervalo de pontuação em que se encontra. Para efeitos de apresentação, a classificação atribuída a cada questão mediante a escolha de uma determinada opção, é apresentada à frente mesma, neste ensaio. Como é óbvio, para efeitos de teste em situações reais esta pontuação irá ser omitida, para eliminar qualquer possibilidade de influenciação por parte do tomador do teste, em escolher determinada opção de resposta.

Quando comparado com outros testes de diagnóstico que estudámos vemos vantagens no nosso testes. A qualidade do teste está ao nível de todos os outros, no que diz respeito aos assuntos abordados. Abordamos as questões que permitem classificar um projeto como ágil ou não, com o mesmo rigor e preocupação dos testes de diagnóstico já existentes, e não é nesse parâmetro que o nosso teste se evidência. São os pequenos pormenores e especificidades que elevam o nosso teste em comparação com os outros. O nosso foco de estudo incidiu em três testes de diagnóstico já existentes: *Karlskrona, Nokia e 42 pontos*. Essencialmente, a estratégia que adotamos de modo a elevar o nosso teste a um patamar superior aos outros, foi a de identificar o problema mais gritante em cada um dos testes estudados, e suprimi-lo aquando da criação do nosso.

No caso do teste de *Karlskrona*, é aquele que no nosso entender melhor configura um teste de diagnóstico de metodologias ágeis, devido ao seu reduzido tamanho e fácil identificação com as opções de resposta. Ainda assim, conseguimos identificar um ponto em que o nosso teste poderia suplantar o teste de *Karlskrona*: o sistema de classificação. No caso do teste de

Karlskrona, o sistema de pontuação baseia-se nas respostas dadas pelo tomador do teste, em que respostas dadas com opções do lado direito do enunciado são recompensadas com um ponto, e onde a respostas do lado esquerdo do enunciado não são atribuídos quaisquer pontos. Finalmente, o resultado final pode inserir-se em cinco categorias diferentes de abordagem ágil. O nosso teste simplificou este sistema de classificação em dois aspetos: a forma como a pontuação é dada e os intervalos de classificação. Ao atribuir pontuações distintas a cada opção de resposta (1 ponto, 3/4 pontos, 5/6 pontos e 8/9 pontos), não ficamos dependentes de qualquer formatação específica de como o enunciado é construído, podendo *baralhar* as mesmas. A nossa perceção de como os pontos vão ser atribuídos, foi inspirada precisamente no teste de *Karlskrona*. Podemos dizer que cada opção de resposta se insere numa das categorias que o teste de *Karlskrona* usa como classificação final do projeto, i.e., a opção de resposta com pontuação um, insere-se na *grade 1* de classificação e assim sucessivamente. A forma como classificamos o tomador no final do teste é que acaba por ser ligeiramente diferente. Tal como já foi dito, no teste de *Karlskrona* existem 5 opções de classificação, e no nosso entender algumas destas categorias, como por exemplo, *Nearly Reached (Grade 4)*, são demasiado específicas e ambíguas. De modo a evitar mal entendimentos sobre o significado de cada um dos intervalos de classificação, fomos mais pragmáticos e reduzimos para 3 o número de possibilidades de classificação.

No caso do teste *Nokia*, consideramos o número de questões e o sistema de pontuação adequado. O problema que identificamos neste teste, foi no extenso número de opções de resposta que, no nosso entender, poderia confundir o tomador do teste. No teste *Nokia*, o número de opções de resposta nunca é menos que 5 chegando a haver perguntas com 7 opções de resposta, algo que confunde o tomador e dificulta a sua identificação com apenas uma opção. Foi por este motivo que decidimos restringir o número de opções de resposta a apenas 4 por pergunta no máximo.

No caso do teste dos 42 pontos, consideramos o sistema de pontuação adequado, e a forma como o teste está construído algo inovador e diferente. No entanto, consideramos o teste demasiado extenso, mesmo tratando-se de um caso em que as opções de respostas são "Sim"ou "Não". Foi por isso que tomamos a decisão de restringir o nosso teste de diagnóstico a apenas 12 questões, focando-nos e questionando os tomadores, apenas naqueles que são os pontos mais importantes do desenvolvimento ágil.

Em suma, identificamos vantagens e desvantagens em todos os testes que estudámos. Tentamos sobretudo herdar os prós e eliminar os contras na construção do nosso próprio teste de diagnóstico. Acabamos por desenvolver um teste onde é notória a inspiração no teste de *Karlskrona* no que diz respeito à extensão do enunciado. Onde é percetível uma influência do teste *Nokia*, no sistema de classificação e cotação e onde as opções de resposta se mantiveram compreensíveis e simples como no teste dos 42 pontos.

4.2.1 Análise Pergunta a Pergunta

De seguida vamos, de maneira sucinta, analisar e fundamentar pergunta a pergunta a totalidade da nossa proposta de teste de diagnóstico ágil e iremos apresentar em anexo, de maneira integral, uma formatação formal do mesmo.

PERGUNTA 1

A equipa consegue entregar regularmente versões funcionais, mesmo que incompletas, do software ao cliente?

a. Não, o software é entregue/apresentado no seu todo ao cliente apenas na data final. (1 ponto)

b. Não, durante o desenvolvimento. (3 pontos)

c. Sim, mas os períodos de entregas variam muito (de 3 a 10 semanas) sendo que o software entregue pode não ser totalmente funcional. (6 pontos)

d. Sim, a equipa apresenta versões funcionais ao cliente. (9 pontos)

Com esta pergunta pretendemos determinar a existência, ou não, no projeto sobre o qual o tomador do teste participa, de uma das características mais importantes nos projetos que seguem uma metodologia ágil: a capacidade de manter um produto funcional ao longo do tempo (integrando os várias componentes desenvolvidos), e ter sempre algo a mostrar ao cliente em qualquer ocasião. Uma empresa com um projeto que siga uma metodologia ágil irá mostrar frequentemente ao seu cliente versões funcionais do produto, mesmo com adições constantes de novas funcionalidades a cada reunião. Se for esse o caso, as propriedades emergentes que devem surgir no quotidiano de desenvolvimento deste projeto irão passar pela: qualidade de trabalho e *feedback*.

PERGUNTA 2

2. A equipa de desenvolvimento é capaz de lidar com mudanças nos requisitos mesmo que estes acontecem num ponto avançado do desenvolvimento?

a. Não, os requisitos não podem ser modificados sem refazer grande parte do progresso realizado. (1 ponto)

b. Não, se estes aparecem tarde de mais. (3 pontos)

c. Sim, se não afetam requisitos anteriores. (6 pontos)

d. Sim. (9 pontos)

Com esta pergunta pretendemos determinar se no projeto do tomador do teste está presente uma das qualidades mais conhecidas das metodologias ágeis, nomeadamente, a capacidade de se adaptar a mudanças constantes nos requisitos impostos pelos clientes. Se estiverem a ser seguidos os princípios de uma metodologia ágil, com equipas distribuídas de 7 elementos e com sprints frequentes, não deverá haver influência de novos requisitos no trabalho já desenvolvido. Se for esse o caso, as propriedades emergentes que devem surgir no quotidiano de desenvolvimento deste projeto irão passar pela: qualidade de trabalho e simplicidade.

PERGUNTA 3

3. Planeou-se entregas de software em períodos relativamente curtos para o pro-

jeto?

a. Não, só existe uma única entrega, a final. (1 ponto)

b. Não, os períodos de entregas são cumpridos (ex: todos os 3 meses) ou variam muito. (3 pontos)

c. Sim, os períodos são curtos, mas variam em função das tarefas a realizar e o tempo restante. (6 pontos)

d. Sim, os períodos são curtos, são inferiores a 6 semanas e são estabelecidos no inicio do projeto com duração fixa. (8 pontos)

Com esta pergunta pretendemos determinar se no projeto em teste, existe o conceito de sprints, i.e., desenvolvimentos e, consequentemente, entregas frequentes de componentes do produto, que irão acrescentar valor ao produto e não irão afetar a sua funcionalidade de maneira negativa. Se as equipas estiverem preparadas para tal, não deverá haver problema em incluir no quotidiano de desenvolvimento os conceitos de *Backlog, ou Sprint Backlog*. Se for esse o caso, as propriedades emergentes que devem surgir no quotidiano de desenvolvimento deste projeto irão passar pela: qualidade de trabalho, simplicidade e *feedback*.

PERGUNTA 4

4. A equipa de desenvolvimento e o(s) cliente(s) participam ativamente no projeto com encontros para tirar dúvidas e participar no desenvolvimento?

a. Não, o cliente não quer responder às dúvidas da equipa de desenvolvimento, sendo que esta realiza apenas as tarefas pedidas. (1 ponto)

b. Não, por outras razões. (3 pontos)

c. Sim, o cliente está disponível para responder a dúvidas da equipa de desenvolvimento. (6 pontos)

d. Sim, são realizadas reuniões onde a equipa de desenvolvimento e o cliente interagem, resolvendo duvidas e propondo modificações. (8 pontos)

Com esta pergunta pretendemos determinar se no projeto em questão, existe interação entre cliente e equipa de desenvolvimento, na forma de reuniões, contactos pessoais ou impessoais. Pretendemos tirar ilações acerca do papel que o cliente desempenha no projeto. Num dos extremos o cliente não demonstra disponibilidade para realizar mais do que duas reuniões: a inicial de explicação, e a de entrega. No outro extremo, caso a metodologia seja puramente ágil, o cliente que contrata a empresa mostra disponibilidade para ter um representante em contacto constante com a equipa de desenvolvimento para esclarecer em qualquer instante alguma dúvida que surja, ou ajustar algum ponto estabelecido anteriormente. Se for esse o caso, as propriedades emergentes que devem surgir no quotidiano de desenvolvimento deste projeto irão passar pela: qualidade de trabalho, comunicação e *feedback*.

PERGUNTA 5

5. As pessoas que participam no projeto têm confiança entre elas e são motivadas para levar a cabo o projeto?

a. Não, a equipa não esta motivada, não gostando do projeto/cliente/produto. (1 ponto)

b. Não, os membros da equipa não se conhecem e não confiam uns nos outros. (3 pontos)

c. Sim, a equipa está motivada, mas alguns membros não confiam uns nos outros. (6 pontos)

d. Sim, a equipa está motivada e os membros confiam um nos outros. (9 pontos)

Com esta pergunta pretendemos determinar se, no projeto em teste, existe a capacidade dentro da equipa de desenvolvimento de agrupar os elementos, atribuindo a cada grupo as responsabilidades de um *Sprint Backlog*. Se houver confiança entre os elementos, várias equipas podem desempenhar em paralelo, várias tarefas em prol de um só projeto, fomentando a programação em conjunto e a inter-ajuda, qualidades defendidas por metodologias ágeis, como por exemplo, o *XP*. Se for esse o caso, as propriedades emergentes que devem surgir no quotidiano de desenvolvimento deste projeto irão passar pela: qualidade de trabalho, comunicação, *feedback* e coragem.

PERGUNTA 6

6. As pessoas que participam no projeto tomam decisões importantes frente a frente? Todos os membros da equipa participam nestas decisões?

a. Não, só os responsáveis do projeto tomam as decisões, sendo que a equipa raramente participa nas discussões. (1 ponto)

b. Não, as decisões são tomadas por telemóvel ou outros meios de comunicações com a maioria dos membros do projeto, sendo raramente tomadas decisões em reuniões presenciais. (3 pontos)

c. Sim, todos os membros participam nas decisões importantes, mesmo que não estejam diretamente presentes durante a tomada destas. (6 pontos)

d. Sim. (9 pontos)

Com esta pergunta pretendemos determinar se, no projeto em teste, existem reuniões frequentes e comunicação entre os envolvidos. Idealmente, o cenário que deve existir numa metodologia ágil passa por reuniões diárias onde é discutido o estado dos sprints, o estado do projeto em geral, assim como alguma alteração ou esclarecimento de dúvidas por parte do(s) representante(s) do cliente. Estas reuniões devem ser pessoais e deve ser promovido o diálogo entre as partes. Se for esse o caso, as propriedades emergentes que devem surgir no quotidiano de desenvolvimento deste projeto irão passar pela: qualidade de trabalho, comunicação e *feedback*.

PERGUNTA 7

7. Como é que a equipa do projeto avalia o seu progresso?

a. A equipa não efetua avaliações do progresso do projeto. (1 ponto)

b. A equipa avalia o seu progresso em função do tempo restante ou da etapa onde se situa segundo o planeamento efetuado. (5 pontos)

c. A equipa avalia o seu progresso em função da documentação já construída relativamente a documentação ainda a fazer. (4 pontos)

d. A equipa avalia o seu progresso em função do software desenvolvido face ao seu todo. (8 pontos)

Com esta pergunta pretendemos, de um modo mais indireto, tentar perceber se no projeto em questão a equipa tem o cuidado, de não só manter o software funcional em todos os momentos, mas também documentado, testado e suscetível a *refactoring*. Em abordagens ágeis, como o *XP* ou o *Scrum*, é essencial manter o código documentado, testado e limpo. Durante uma iteração é importante detetar prontamente quaisquer *code smells* de modo a eliminá-los. No final de cada iteração é importante criar executáveis do código documentado e funcional, para proceder a uma bateria de testes e demonstrações. Se for esse o caso, as propriedades emergentes que devem surgir no quotidiano de desenvolvimento deste projeto irão passar pela: qualidade de trabalho e *feedback*.

PERGUNTA 8

8. A equipa de desenvolvimento consegue manter um ritmo de desenvolvimento constante e sustentável (com uma cadência que se adequa ao projeto em questão)?

a. Não, estando inexperientes com as tecnologias usadas, a equipa não é capaz de acompanhar o ritmo de desenvolvimento exigido, chegando a haver atrasos para os prazos combinados. (1 ponto)

b. A equipa consegue manter um ritmo de desenvolvimento sustentável, mas inconstante, com alguns atrasos nos prazos de entrega. (3 pontos)

c. O ritmo de desenvolvimento da equipa é constante, mas não sustentável, com atribuição irregular de tarefas e subaproveitamento dos recursos. (6 pontos)

d. Sim. (8 pontos)

Com esta pergunta pretendemos, mais uma vez de modo indireto, tentar perceber se a equipa de desenvolvimento cumpre os conceitos de *Sprint e Sprint Backlog*. Numa metodologia ágil, um dos conceitos fundamentais consiste na introdução de um método de desenvolvimento que passe pela divisão e atribuição de tarefas pelas sub-equipas de desenvolvimento. As tarefas que essas sub-equipas deverão desempenhar estarão descritas num documento chamado *Sprint Backlog*, imutável a partir do momento em que se inicia o *Sprint*. Deve ser previamente estabelecida a duração do *Sprint* antes de se iniciar o mesmo. A equipa deverá focar-se em cumprir nesse prazo de tempo as tarefas estabelecidas no *Sprint Backlog*, integrando os componentes que desenvolveu com os componentes desenvolvidos pelas restantes sub-equipas. O processo deverá repetir-se em cadências semelhantes até ao final do projeto. Se for esse o

caso, as propriedades emergentes que devem surgir no quotidiano de desenvolvimento deste projeto irão passar pela: qualidade de trabalho, comunicação, simplicidade e *feedback*.

PERGUNTA 9

9. A equipa de desenvolvimento consegue manter boas práticas ao longo do desenvolvimento?

a. A equipa não seguiu qualquer tipo de planeamento ou boas práticas de codificação. (1 ponto)

b. Não, para entregar o produto foi necessário saltar algumas etapas do desenvolvimento. (3 pontos)

c. Foram seguidos alguns padrões e normas, mas há necessidade de efetuar refactoring do produto de tempos a tempos. (6 pontos)

d. Sim, a equipa seguiu padrões de desenvolvimentos e normas bem estabelecidas. (8 pontos)

Com esta pergunta pretendemos tentar perceber se durante o desenvolvimento a equipa de desenvolvimento aplicou no seu planeamento conceitos como *Sprint e Sprint Backlog*, ou realizou reuniões diárias onde se estabeleceram métodos e onde se tomarão decisões concretas acerca da práticas a seguir na implementação. A título de exemplo, o estabelecimento de práticas de codificação ou recurso a *frameworks*. Se for esse o caso, as propriedades emergentes que devem surgir no quotidiano de desenvolvimento deste projeto irão passar pela: qualidade de trabalho, comunicação, simplicidade e *feedback*.

PERGUNTA 10

10. A equipa de desenvolvimento organizou-se de maneira a minimizar a quantidade de trabalho?

a. Não, a equipa não planeou o projeto. (1 ponto)

b. A equipa não tem experiência suficiente e, como tal, elaborou um plano de projeto irrealista face as necessidades do mesmo. (5 pontos)

c. A equipa tem um plano de projeto, mas não se organiza sobre este, com vários membros a trabalhar à sua maneira. (4 pontos)

d. Sim. (8 pontos)

Com esta pergunta pretendemos verificar se a equipa de desenvolvimento segue as normas que estabelecem que num projeto ágil todos desempenham um papel específico que contribui para a produção de um produto final de qualidade. Num projeto ágil todos devem fazer parte de uma sub-equipa bem definida, de cerca de meia-dúzia de elementos, onde cada sub-equipa desempenha apenas as tarefas que lhe foram atribuídas (na forma de um *Sprint Backlog*), no tempo que foi estabelecido previamente, tempo este flexível dependendo das dificuldades da mesma. Se for esse o caso, as propriedades emergentes que devem surgir no quotidiano de desenvolvimento deste projeto irão passar pela: qualidade de trabalho, comunicação, simplicidade e *feedback*.

PERGUNTA 11

11. A equipa de desenvolvimento tem capacidade de se auto-organizar?

a. Não, o chefe da secção de desenvolvimento, que não participa no projeto, organiza as equipas de desenvolvimento. (1 ponto)

b. Não, é seguido um plano estabelecido pela empresa que é generalizado para todos os projetos que ocorrem nesta. (3 pontos)

c. Sim, cada equipa possui um chefe de equipa que organiza as tarefas para a equipa toda. (6 pontos)

d. Sim, a equipa no seu todo reúne-se para gerir e distribuir as tarefas. (8 pontos)

Com esta pergunta pretendemos perceber se a equipa de desenvolvimento se auto-organiza de modo a ser capaz de esquecer, de certa maneira, hierarquias para estabelecimento de prazos ou distribuição de sub-equipas pelas tarefas a realizar. Numa metodologia ágil, idealmente, o *project owner* deve manter-se ativo na fase de *Sprint Planning*, mas deve afastar-se durante a fase de implementação. Todos devem dar a sua opinião e ninguém se deve destacar numa posição de liderança. Se for esse o caso, as propriedades emergentes que devem surgir no quotidiano de desenvolvimento deste projeto irão passar pela: comunicação, simplicidade e *feedback*.

PERGUNTA 12

12. A equipa reúne-se regularmente para refletir sobre a sua eficácia e ajustar o seu comportamento se necessário?

a. Não, a equipa não reflete sobre a sua eficácia, mantendo sempre o mesmo comportamento. (1 ponto)

b. Não, são os responsáveis da equipa ou dirigentes que avaliam a eficácia, sendo aplicadas medidas para ajustar o desempenho quando se considerar necessário. (3 pontos)

c. A equipa só se reúne no final de cada projeto para refletir e aplicar alterações para o próximo projeto. (6 pontos)

d. Sim, a equipa reúne-se no final de cada fase do projeto para avaliar o progresso realizado face ao planeado e ajustar-se de acordo. (8 pontos)

Com esta pergunta pretendemos incidir na realização de reuniões, idealmente diárias, para discussão do ponto da situação de cada sub-equipa, ou para a redistribuição de novas tarefas. A equipa deve assiduamente documentar o código, discutir o mesmo, realizar baterias de teste e contribuir com ideias construtivas, de modo a contribuir para um projeto virtuoso. Se for esse o caso, as propriedades emergentes que devem surgir no quotidiano de desenvolvimento deste projeto irão passar pela: qualidade de trabalho, coragem, comunicação, simplicidade e *feedback*.

4.2.2 Sistema de Classificação

Uma vez respondidas as questões, será atribuída uma cotação a cada uma das respostas, sendo que a pontuação máxima disponível é de 100 pontos. Como se pode adivinhar, quanto maior for a pontuação, melhor se poderá classificar o projeto como ágil:

1. 100-71 – No geral o projeto segue princípios ágeis fielmente, e poderá ser classificado como tal;
2. 70-37 – O projeto segue alguns dos princípios ágeis, mas tem algumas inconsistências e, como tal, será preciso rever algumas das metodologias e processos utilizados;
3. 38-12 – O projeto em questão não pode ser considerado ágil, pois quebra demasiados dos princípios essenciais.

Em anexo iremos apresentar a versão integral do nosso teste de diagnóstico, com a formatação usada para efeitos de apresentação aos respondentes do mesmo.

4.3 Exemplo de Aplicação do Método de Diagnóstico

Nesta secção, a nossa intenção seria ade apresentar os resultados obtidos após a aplicação do nosso teste de diagnóstico num contexto real, se possível ao projeto de uma empresa dentro do mercado do desenvolvimento de software.

Abordamos várias fontes, que poderiam eventualmente conhecer tais equipas de desenvolvimento, que pudessem responder ao nosso questionário, e só temos de agradecer ao Professor Pedro Ribeiro e ao Professor João Miguel Fernandes, pelas suas disponibilidades, nesta tentativa. Demos ainda a nossa garantia, de que iríamos analisar todos os resultados recolhidos, e devolver o *feedback*, a cada respondente, quase como que de um agradecimento pelo tempo perdido se tratasse.

No entanto, por motivos que nos são alheios, nenhuma empresa respondeu em tempo útil ao dito teste de diagnóstico, tornando assim impossível a concretização do que tínhamos proposto nesta secção. Deixamos ainda uma menção ao facto de termos, de propósito, criado um *Google Forms* para onde convertemos o teste de diagnóstico, na esperança de facilitar a realização do mesmo. Ainda assim, não obtivemos resposta.

5 Conclusão

Podemos facilmente dividir o conteúdo deste longo relatório em duas partes: o estudo teórico e a aplicação prática. No estudo teórico, foram-se apresentado os princípios e virtudes seguidos pelos modelos ágeis, tendo-se apresentado os modelos mais usados na indústria. Como era de esperar, já existem alguns testes de auto-diagnóstico para empresas e equipas de desenvolvimento que usam metodologias ágeis ou que querem vir a implementar, sendo que os testes encontrados foram, geralmente, desenvolvidos focando-se no diagnóstico de um único modelo de desenvolvimento, como o Scrum.

Na fase seguinte, de aplicação prática dos conceitos teóricos adquiridos, concretizamos a nossa intenção de construir um teste de diagnóstico ágil preparado para ser aplicado na indústria, unindo para tal aquelas que são no nosso entender, as melhores características de cada teste, já existente, que estudamos. Criamos um questionário com 12 perguntas, cada uma com 4 opções de escolha e um sistema de classificação de 3 categorias. Cada pergunta foi devidamente analisada, para justificar o seu propósito. Cada opção de escolha foi pensada de modo a inserir o respondente num determinado patamar, no que toca à presença de uma metodologia ágil no projeto em teste. O sistema de classificação foi idealizado para que, mediante a cotação de cada opção de resposta, o respondente ficasse com uma perceção do estado do projeto em que se insere.

Deixamos ainda uma menção ao facto de, por motivos que nos são alheios, não termos conseguido reunir em tempo útil, resultados da aplicação do nosso teste, apesar dos esforços contínuos, de todos as partes interessadas, professores e nós próprios.

A aplicação de metodologias ágeis, é algo que está a tornar-se perigosamente numa moda. Até pode não ser a abordagem mais adequada, mas ainda assim as equipas de desenvolvimento não se atrevem a não usa-la. No nosso entender, a sua aplicação está a tornar-se demasiado corriqueira, e os aspetos positivos desta organização estão a perder-se. É por isso que responder a testes como este, apesar de muitas vezes serem considerados perdas de tempo, deviam ser prática corrente, porque na ânsia de tentar trazer conceitos positivos no ambiente de desenvolvimento, a má aplicação desta metodologia pode ter o efeito exatamente oposto.

Deixamos ainda uma ideia para futuro desenvolvimento. Algo que ao contrário do que acontecia com os testes de diagnósticos ágeis, ainda não existe no mercado: o desenvolvimento de um teste que determine se vale ou não a pena, para cada projeto recorrer a uma metodologia ágil.

Referências

[1] The Standish Group. Chaos report 2014.

[2] TechBeacon. Survey: Is agile new norm?

[3] Vic Basili and A.J. Turner. Iterative enhancement: A practical technique for software development, ieee transactions on software engineering, vol. 1, no.4, pp. 390-396, 1975.

[4] K. Beck. Embrace change with extreme programming, ieee computer pp.70-77, oct. 1999a.

[5] Winston W. Royce. Managing the development of large software systems.

[6] B Boehm. A spiral model of software development and enhancement, ieee computer, vol. 21, no. 5, pp. 61-72, 1988.

[7] Unknown. Agile software development - philosophy.

[8] Patricia Costa David Cohen Mikael Lindvall¨ Agile software development. Technical report, Fraunhofer Center Maryland, 20XX.

[9] Unknown. Agile software development - experience and adoption.

[10] Joe Little. The nokia test.

[11] Mayberg. Karlskrona test.

[12] Kelly Waters. How agile are you? (take this 42 point test).

Anexo

Teste da Nokia

Questão 1 - Iterações:

a) Sem iterações

b) Iterações > 6 semanas

c) Tamanho variável < semanas

d) Iterações fixas que duram 6 semanas

e) Iterações fixas que duram 5 semanas

f) Iterações 4 semanas ou menos

Questão 2 - Testar

a) Sem uma unidade para testar dedicada na equipa

b) Unidade testada

c) Funcionalidades testadas

d) Funcionalidades testadas assim que são completadas

e) Software passa testes de aceitação

f) Software é implementado

Questão 3 - Permitir especificação

a) Sem requisitos

b) Documento de requisitos enorme

c) User stories fracas

d) Bons requisitos

e) Boas user stories

f) O suficiente dentro do tempo de especificação

g) Boas user stories associadas às especificações como necessário

Questão 4 - Gestor do produto

a) Sem gestor do produto

b) Gestor do produto não compreende Scrum

c) Gestor do produto incomoda a equipa

d) Gestor do produto não está envolvido com a equipa

e) Gestor do produto tem o *backlog* do produto estimado pela equipa antes da reunião de planeamento do Sprint

f) Gestor do produto tem uma *roadmap* de lançamento com datas baseadas na velocidade da equipa

g) Gestor do produto motiva a equipa

Questão 5 - *Backlog* do produto

a) Sem *backlog* do produto

b) Múltiplos *backlogs* do produto

c) Um único *backlog* do produto

d) *Backlog* do produto tem boas user stories que satisfazem os critérios de investimento

e) Does dos sprints do *backlog* do produto estão num estado PRONTO

f) *Roadmap* do produto está disponível e é atualizada regularmente baseada em estimativas da equipa do *backlog* do produto

Questão 6 - Estimativas

a) Estimativas do *backlog* do produto não estimadas

b) Estimativas não produzidas pela equipa

c) Estimativas produzidas por planeamento

d) Estimativas produzidas por planeamento da equipa

e) Estimatica de erro < 10

Questão 7 - Perturbação da equipa

a) Gestor ou líder do produto perturba a equipa

b) Gestor do produto perturba a equipa

c) Gestores, líderes do projeto ou líderes da equipa dizem às pessoas o que fazer

d) Ter líder do projeto e papéis de Scrum

e) Ninguém perturba a equipa, apenas existem papéis de Scrum

Teste de Karlskrona

1. Do you regularly deliver fully working software (documented and tested)?

- No, we wait until everything is finished
- We gradually improve software and tests
- We deliver software and fix issues later
- We gradually improve and show to customer

2. How long are these delivery iterations?

- Depending on planned deadlines
- Every 8 weeks or less
- Every few months, very variable
- Every 4 weeks or less, always same length

3. How much is documented before start of implementation?

- Complete product is specified
- Basic product features, expected to change
- Little, someone decides what to do
- Clearly defined goals, details will be worked out

4. How often do you adjust or improve your way of working?

- Nearly never, management takes care
- We hold regular meetings and change things
- After something important went wrong
- Every day we indentify issues and take actions

5. Is there a collection of items and features describing the product?

- One big requirements document
- List prioritized by customer/business needs
- Some parts, not always up to date
- List with user stories and estimated difficulty

6. Do you know someone who is responsible for the product?

- No, we use documents to communicate
- One person is responsible and always available
- Group of people is responsible
- Direct contact to a customer representative

7. How is the team's work planned?

- Management decides what's needed
- Team implements high value items first
- Developers know what's important
- Team and customer plan multiple iterations

8. How good is the cooperation across team borders?

- Team leader or project leader takes care
- We all know the big picture and plan together
- Management never listens
- Teams and customers work actively together

9. How do teams track their work?

- Write down how many hours spent
- Estimate weekly the remaining work
- Write weekly status report
- Update daily how much work is remaining

10. Do you know if your team efficiency improves over time?

- I don’t know, we don’t measure this
- Team can compare efficiency of past iterations
- We measure time spent and overtime
- Team knows velocity and improves it

11. Is the team disrupted from work?

- Scope of work is regularly changed
- There are few unexpected changes in iterations
- Management often assigns new tasks
- Team can solely focus on planned work

Teste dos 42 Pontos

1. The team is empowered to make decisions.

2. The team is self-organising and does not rely on management to set and meet its goals.

3. The team commits and takes responsibility for delivery and is prepared to help with any task that helps the team to achieve its goal.

4. The team knows who the product owner is.

5. Each sprint/iteration has a clear goal.

6. All team members, including testers, are included in requirements workshops.

7. Requirements documentation is barely sufficient and the team collaborates to clarify details as features are ready for development.

8. Test cases are written up-front with the requirements/user story.

9. There is a product backlog/feature list prioritised by business value.

10. The product backlog has estimates created by the team.

11. The team knows what their velocity is.

12. Velocity is used to gauge how many user stories should be included in each sprint/iteration.

13. Sprints/iterations are timeboxed to four weeks or less.

14. Sprint budget is calculated to determine how many product backlog items/features can be included in the sprint/iteration.

15. The sprint/iteration ends on the agreed end date.

16. All tasks on the sprint backlog are broken down to a size that is less than one day.

17. Requirements are expressed as user stories and written on a card.

18. The team estimates using points which indicate the relative size of each feature on the product backlog/feature list.

19. The team generates burndown charts to track progress daily.

20. Software is tested and working at the end of each sprint/iteration.

21. The team is not disrupted during the sprint/iteration.

22. Changes are integrated throughout the sprint/iteration.

23. Automated unit testing is implemented where appropriate.

24. There is an automated build and regression test.

25. Stretch tasks are identified for inclusion in the sprint/iteration if it goes better than expected.

26. The Product Owner is actively involved throughout each sprint.

27. All code changes are reversible and it is possible to make a release at any time.

28. Testing is integrated throughout the lifecycle and starts on delivery of the first feature.

29. Impediments that hold up progress are raised, recorded on the whiteboard and resolved in a timely fashion.

30. When someone says 'done', they mean DONE! (ie shippable).

31. The team uses the whiteboard to provide clear visibility of progress and issues on a daily basis.

32. The sprint/iteration goal(s) is clearly visible on the board.

33. All user stories and tasks are displayed on the whiteboard for the duration of the sprint/iteration.

34. Daily scrums happen at the same time every day – even if the scrum master isn't present.

35. The daily scrum is resticted to answering the standard 3 scrum questions and lasts no more than 15 minutes.

36. There is a product demonstration/sprint review meeting at the end of each sprint/iteration.

37. All team members, including testers and Product Owner, are included in the sprint/iteration review.

38. The sprint/iteration review is attended by executive stakeholders.

39. There is a sprint retrospective at the end of each sprint/iteration.

40. Key metrics are reviewed and captured during each sprint retrospective.

41. All team members, including testers, are included in the sprint retrospective meeting.

42. Actions from the sprint retrospective have a positive impact on the next sprint/iteration.

Consulte e responda ao nosso teste de diagnóstico em:

https://docs.google.com/forms/d/e/1FAIpQLSet6nvfyw6HcWnoQfbLX0wdivDucveH_BMMFASsL6ATKzluGA/viewform

Teste de Diagnóstico Ágil Malvic
Gestão de Processo de Software

Data: _____ Nome da Empresa: _____

Resultado: _____ Nome do Projeto: _____

Descreva sucintamente o projeto a avaliar

Equipa

Nome do Respondente:_____

Papel no Projeto:_____

Email do Respondente (*):_____

Descrição Sucinta da Equipa (Nomes, Roles)

Nº Elementos Equipa:_____ Duração do Projeto:_____ Orçamento: _____

Cliente: _____

(*) - (Obrigatório para envio de feedback)

As metodologias ágeis têm como objetivo simplificar todo o processo de desenvolvimento de software dentro das empresas. A simplificação vem com um custo: a dificuldade em adotar as políticas corretas. Faça o seguinte teste e descubra em que ponto é que o seu projeto se encontra no que toca à adoção de abordagens ágeis.

O teste consiste em 12 perguntas. Deverá escolher APENAS UMA opção, aquela com que mais se identifica. Responda com sinceridade! Todo o conteúdo do questionário será confidencial.

---

1. **A equipa consegue entregar regularmente versões funcionais, mesmo que incompletas, do software ao cliente?**
   (a) Não, o software é entregue/apresentado no seu todo ao cliente apenas na data final.
   (b) Não durante o desenvolvimento (codificação).
   (c) Sim, mas os períodos de entregas variam muito (de 3 a 10 semanas) sendo que o software entregue pode não ser totalmente funcional.
   (d) Sim, a equipa apresenta versões funcionais ao cliente.

2. **A equipa de desenvolvimento é capaz de lidar com mudanças nos requisitos mesmo que estes aconteçam num ponto avançado do desenvolvimento?**
   (a) Não, os requisitos não podem ser modificados sem refazer grande parte do progresso realizado.
   (b) Não, se estes aparecerem quando a codificação já começou.
   (c) Sim, se não afetarem requisitos anteriores.
   (d) Sim.

3. **Planeou-se entregas de software em períodos relativamente curtos para o projeto?**
   (a) Não, só existe uma única entrega, a final.
   (b) Não, os períodos de entregas são compridos (ex: todos os 3 meses) ou variam muito.
   (c) Sim, os períodos são curtos, mas variam em função das tarefas a realizar e o tempo restante.
   (d) Sim, os períodos são curtos, são inferiores a 6 semanas e são estabelecidos no inicio do projeto com duração fixa.

4. **A equipa de desenvolvimento e o(s) cliente(s) participam ativamente no projeto com encontros para tirar dúvidas e participar no desenvolvimento?**
   (a) Não, o cliente não responde às dúvidas da equipa de desenvolvimento, sendo que esta realiza apenas as tarefas pedidas na reunião inicial.
   (b) Não, o cliente apenas se pronuncia através de contactos impessoais (Email, telefone...).
   (c) Sim, o cliente está disponível para responder a dúvidas da equipa de desenvolvimento.
   (d) Sim, são realizadas reuniões onde a equipa de desenvolvimento e o cliente interagem, resolvendo duvidas e propondo modificações.

5. **As pessoas que participam no projeto têm confiança entre elas e estão motivadas para levar a cabo o projeto?**
   (a) Não, a equipa não esta motivada, não gostando do projeto/cliente/produto.
   (b) Não, os membros da equipa não se conhecem e não confiam uns nos outros.
   (c) Sim, a equipa está motivada, mas alguns membros não confiam uns nos outros.
   (d) Sim, a equipa está motivada e os membros confiam um nos outros.

6. **As pessoas que participam no projeto tomam decisões importantes frente a frente? Todos os membros da equipa participam nestas decisões?**
   (a) Não, só os responsáveis do projeto tomam as decisões, sendo que a equipa raramente participa nas discussões.
   (b) Não, as decisões são tomadas por telemóvel ou outros meios de comunicações com a maioria dos membros do projeto, sendo raramente tomadas decisões em reuniões presenciais.
   (c) Sim, todos os membros participam nas decisões importantes, mesmo que não estejam diretamente presentes durante a tomada destas.
   (d) Sim.

7. **Como é que a equipa do projeto avalia o seu progresso?**
   (a) A equipa não efetua avaliações do progresso do projeto.
   (b) A equipa avalia o seu progresso em função do tempo restante ou da etapa onde se situa segundo o planeamento efetuado.
   (c) A equipa avalia o seu progresso em função da documentação já construída relativamente a documentação ainda a fazer.
   (d) A equipa avalia o seu progresso em função do software desenvolvido face ao seu todo.

8. **A equipa de desenvolvimento consegue manter um ritmo de desenvolvimento constante e sustentável (com uma cadência que se adequa ao projeto em questão)?**
   (a) Não, estando inexperientes com as tecnologias usadas, a equipa não é capaz de acompanhar o ritmo de desenvolvimento exigido, chegando a haver atrasos para os prazos combinados.
   (b) A equipa consegue manter um ritmo de desenvolvimento sustentável, mas inconstante, com alguns atrasos nos prazos de entrega.
   (c) O ritmo de desenvolvimento da equipa é constante, mas não sustentável, com atribuição irregular de tarefas e subaproveitamento dos recursos.
   (d) Sim.

9. **A equipa de desenvolvimento consegue manter boas práticas ao longo do desenvolvimento?**
   (a) A equipa não seguiu qualquer tipo de planeamento ou boas práticas de codificação.
   (b) Não, para entregar o produto foi necessário saltar algumas etapas do desenvolvimento.
   (c) Foram seguidos alguns padrões e normas, mas há necessidade de efetuar refactoring do produto de tempos a tempos.
   (d) Sim, a equipa seguiu padrões de desenvolvimentos e normas bem estabelecidas.

10. **A equipa de desenvolvimento organizou-se de maneira a minimizar a quantidade de trabalho?**
    (a) Não, a equipa não planeou o projeto.
    (b) A equipa não tem experiência suficiente e, como tal, elaborou um plano de projeto irrealista face as necessidades do mesmo.
    (c) A equipa tem um plano de projeto, mas não se organiza sobre este, com vários membros a trabalhar à sua maneira.
    (d) Sim.

11. **A equipa de desenvolvimento tem capacidade de se auto-organizar?**
    (a) Não, o chefe da secção de desenvolvimento, que não participa no projeto, organiza as equipas de desenvolvimento.
    (b) Não, é seguido um plano estabelecido pela empresa que é generalizado para todos os projetos que ocorrem nesta.
    (c) Sim, cada equipa possui um chefe de equipa que organiza as tarefas para a equipa toda.
    (d) Sim, a equipa no seu todo reúne-se para gerir e distribuir as tarefas.

12. **A equipa reúne-se regularmente para refletir sobre a sua eficácia e ajustar o seu comportamento se necessário?**
    (a) Não, a equipa não reflete sobre a sua eficácia, mantendo sempre o mesmo comportamento.
    (b) Não, são os responsáveis da equipa ou dirigentes que avaliam a eficácia, sendo aplicadas medidas para ajustar o desempenho quando se considerar necessário.
    (c) A equipa só se reúne no final de cada projeto para refletir e aplicar alterações para o próximo projeto.
    (d) Sim, a equipa reúne-se no final de cada fase do projeto para avaliar o progresso realizado face ao planeado e ajustar-se de acordo.

Observações

Tabela de Resultados

Questões	Opção	Pontos	Cotação
1, 2, 5, 6	(a)	1	
	(b)	3	
	(c)	6	
	(d)	9	
3, 4, 8, 9, 11, 12	(a)	1	
	(b)	3	
	(c)	6	
	(d)	8	
7, 10	(a)	1	
	(b)	5	
	(c)	4	
	(d)	8	

Escala de Classificação

• 100-71 – No geral o projeto segue princípios ágeis fielmente, e poderá ser classificado como tal;

• 70-37 – O projeto segue alguns dos princípios ágeis, mas tem algumas inconsistências e, como tal, será preciso rever algumas das metodologias e processos utilizados;

• 38-12 – O projeto em questão não pode ser considerado ágil, pois quebra demasiados dos princípios essenciais.