Desenvolvimento Ágil de Software

Metodologias de Desenvolvimento de Software

Pedro Salgueiro
pds@uevora.pt
CLV-256

Desenvolvimento rápido de software

- Desenvolvimento e entrega rápida de software é muito importante
  - Dinâmica de negócios é rápida:
    - os requisitos alteram-se rapidamente
    - difícil de produzir um conjunto de requisitos estável
  - Software tem de adaptar-se rapidamente para refletir as dinâmicas dos negócios

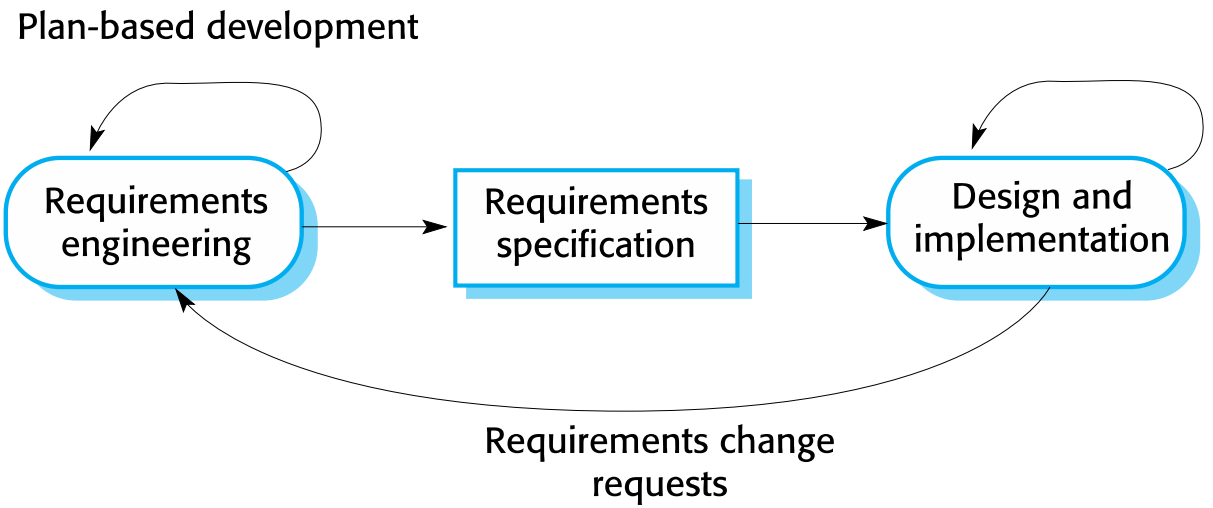
- Métodos baseados em planos
  - Essenciais para alguns tipos de software
  - Pouco adequados para as necessidades de alguns negócios

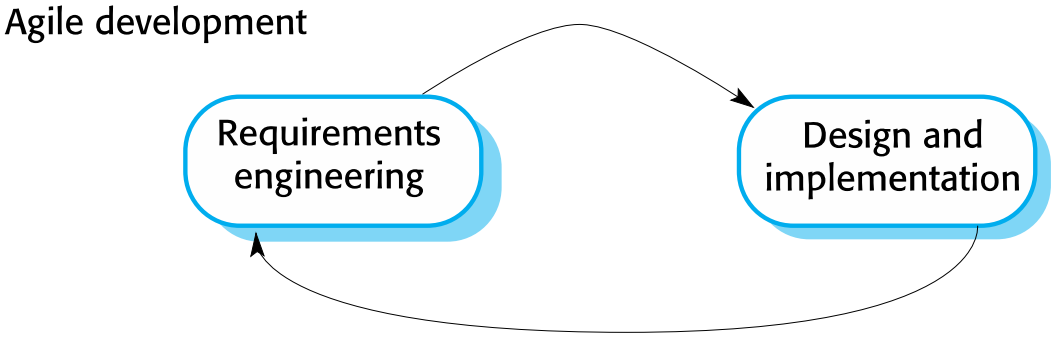
- Métodos Ágeis de Desenvolvimento
  - Surgiram no final dos anos 90
  - Objetivo de reduzir drasticamente o o tempo de entrega de do software

Desenvolvimento rápido de software

- Especificação, desenho e implementação estão intercaladas
- Sistemas são desenvolvidos como uma série de versões
  - Todos os interessados participam na avaliação das versões
- Entregas frequentes de novas versões
  - Avaliação
- Ferramentas de apoio ao desenvolvimento
  - e.g.: ferramentas que executam os testes de forma automática
- Documentação mínima
  - Foco em código funcional

Planos e métodos ágeis





Planos e métodos ágeis

- Desenvolvimento baseado em planos
  - Etapas de desenvolvimento separadas
    - Output de cada etapa é planeado com antecedência
  - Pode não ser o modelo *waterfall*
    - É possível fazer desenvolvimento incremental, baseado em planos
  - Iterações ocorrem dentro de cada atividade
- Métodos ágeis
  - Especificação, desenho, implementação e testes
    - Intercalados
    - Output de cada etapa é decidida num processo de negociação durante o processo de desenvolvimento

Métodos ágeis

- Insatisfação com as metodologias “pesadas” de desenvolvimento de software entre os anos 1960 e 1990 deram origem aos métodos ágeis
  - Foco no código em vez do desenho
  - Baseadas em abordagem iterativas
  - Têm como objetivo a entrega rápida de software rapidamente e a evolução rápida por forma a responder às alterações dos requisitos

- Objetivo das metodologias ágeis
  - Reduzir o peso do processo de desenvolvimento de software
    - Reduzindo a documentação existente
  - Responder de forma rápida às alterações dos requisitos, evitando refazer muito trabalho

Agile manifesto

- *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.*

Princípios do Agile Manifesto

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

Aplicação de métodos ágeis

- Desenvolvimento de software de pequena/média escala
- Desenvolvimento de sistemas à medida,
  - onde existe uma participação ativa do cliente no processo de desenvolvimento
  - não existam muitas regras externas que afetem o software
- Foco em equipas pequenas e muito bem integradas
  - Difícil de escalar métodos ágeis para sistemas grandes

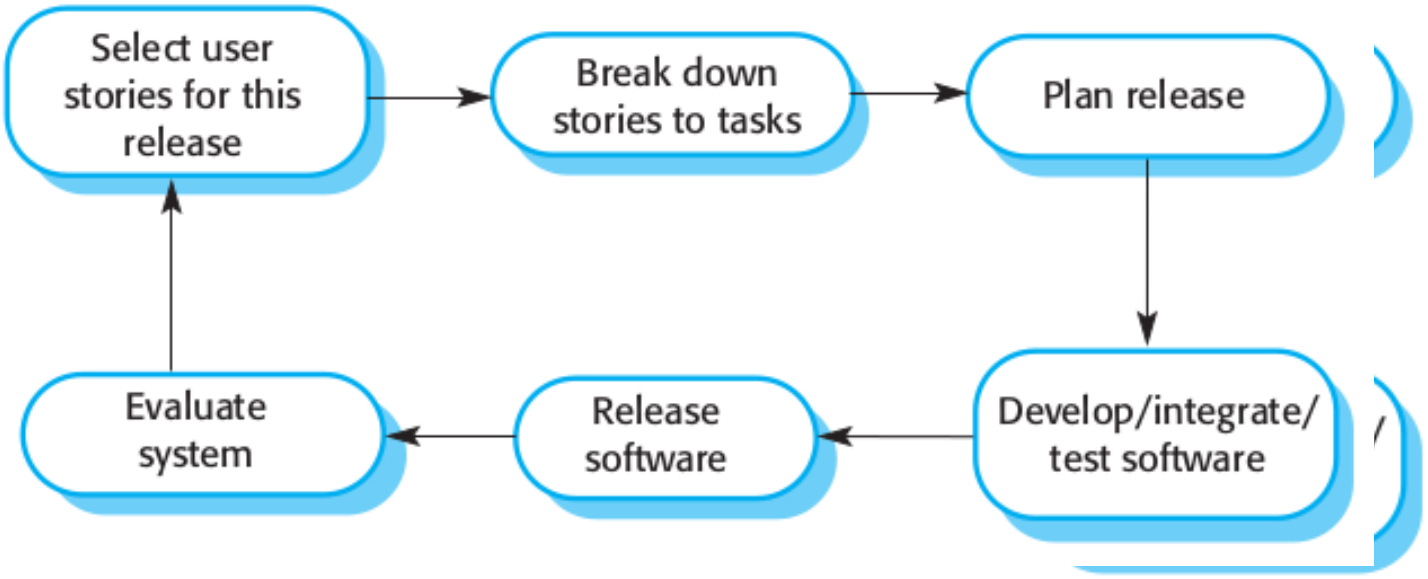
Técnicas de desenvolvimento ágil

Extreme Programming

- Método ágil
  - Criado no final dos anos 90
  - Introduziu muitos dos conceitos das técnicas de desenvolvimento ágil
  - Influenciou muitos outros métodos

- Leva a abordagem de desenvolvimento iterativo ao “extremo”
  - Várias versões por dia;
  - Incrementos são entregues aos clientes a cada 2 semanas;
  - Todos os testes devem ser executados antes de fazer uma nova versão
  - Uma versão nova só é aceite se todos os testes passarem com sucesso

Extreme Programming – ciclo de *releases*



Extreme Programming – boas práticas

Principle or practice	Description
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'. See Figures 3.5 and 3.6.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Extreme Programming – boas práticas

Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site customer	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

XP e métodos ágeis

- Desenvolvimento incremental
  - Através de *releases*/verões pequenas e frequentes
- Participação do cliente
  - A tempo inteiro
  - Fazer parte da equipa de desenvolvimento
- Dar mais importância às pessoas do que aos processos
  - Programação por pares
  - “Pertença” conjunta
  - Usar processos que evitem longas horas de trabalho
- Alterações
  - Suportadas através de releases/versões regulares
- Manter a simplicidade do código
  - *Refactoring* constante do código

XP – práticas influentes

- Extreme programming
  - Focado em aspetos técnicos
  - Não é fácil de integrar com aspetos de gestão
- Métodos ágeis
  - Usam algumas das práticas de XP
  - XP raramente é usado da forma como foi originalmente definido
- Práticas mais usadas
  - *User stories* para especificação
  - *Refactoring*
  - Test-driven development
  - Programação por pares

User stories como requisitos

- Em XP, o cliente ou utilizador faz parte da equipa de desenvolvimento
  - é responsável por fazer decisões sobre os requisitos
- Requisitos de utilizador são expressos através de cenários ou “user stories”
  - Equipa de desenvolvimento divide-as em tarefas
  - Base para estimar o calendário e o orçamento
- Cliente escolhe as “histórias” para incluir na próxima “release”/versão
  - Considerando as suas prioridades e a estimativa do tempo necessário para a implementação

Exemplo de *user story*: “Receitar medicamento”

Prescribing medication

The record of the patient must be open for input. Click on the medication field and select either ‘current medication’, ‘new medication’ or ‘formulary’.

If you select ‘current medication’, you will be asked to check the dose; If you wish to change the dose, enter the new dose then confirm the prescription.

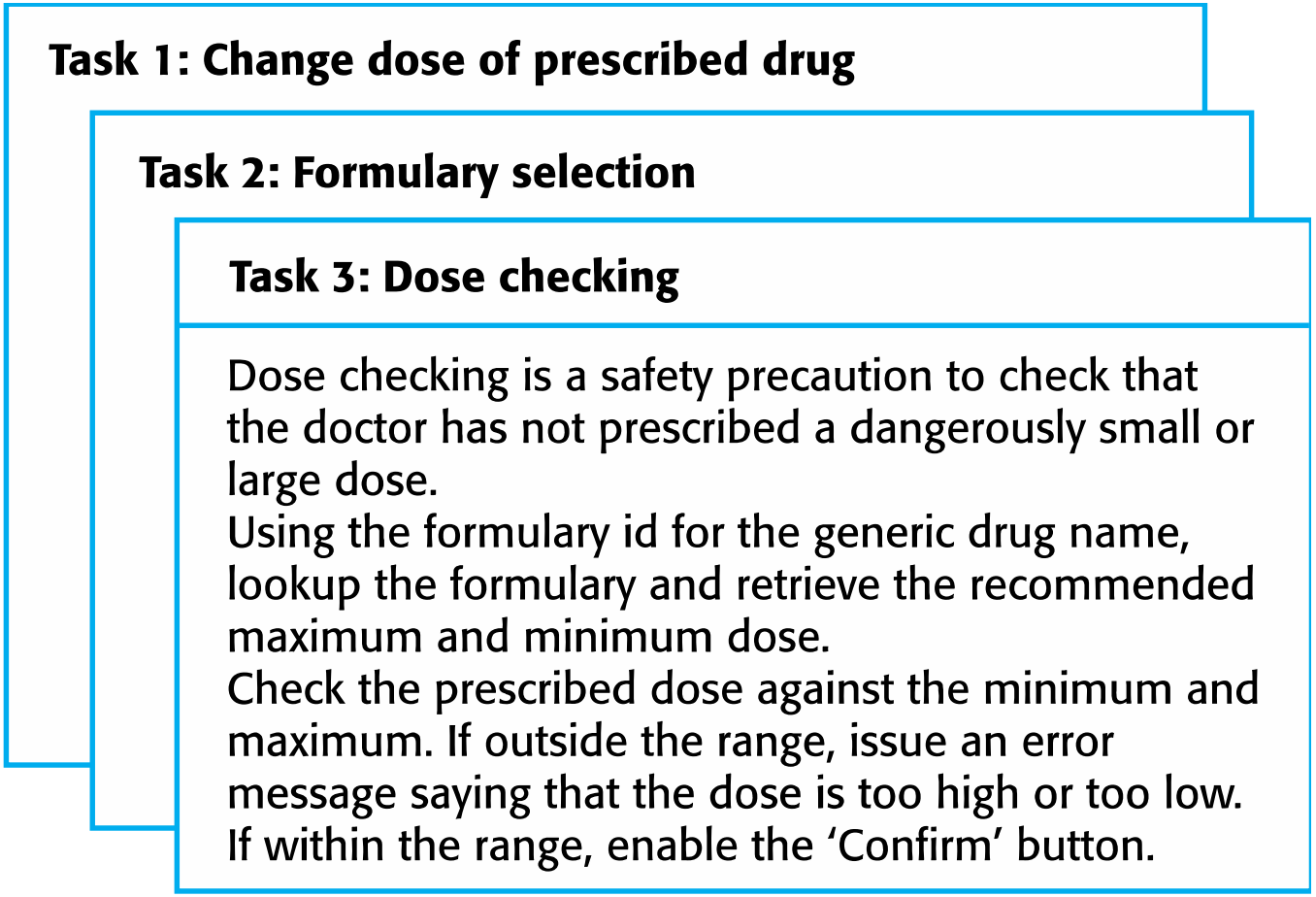
If you choose, ‘new medication’, the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

If you choose ‘formulary’, you will be presented with a search box for the approved formulary. Search for the drug required then select it. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

After you have confirmed the prescription, it will be displayed for checking. Either click ‘OK’ or ‘Change’. If you click ‘OK’, your prescription will be recorded on the audit database. If you click ‘Change’, you reenter the ‘Prescribing medication’ process.

Exemplo de tarefas: “Receitar medicamento”



Refactoring

- Consenso generalizado em Eng. de Software
  - Desenhar software para ser facilmente alterado
  - Vale a pena gastar tempo e esforço e antecipar alterações
  - Reduz custos mais tarde no ciclo de vida do software

- XP
  - Defende que não se deve antecipar alterações
  - Não é possível antecipar as alterações de forma fiável
  - Propõe melhorias constantes do código
    - Refactoring
    - Por forma a tornar as alterações mais fáceis de implementar

Refactoring

- A equipa de desenvolvimento procura por possíveis melhorias no software e implementa essas melhorias
  - mesmo que não existe uma necessidade imediata
- Melhora a compreensão do software e reduz a necessidade de documentação
- Alterações são mais fáceis de implementar porque o código está bem estruturado e fácil de ler
- No entanto, algumas alterações requerem fazer o refactoring da arquitetura
  - Processo mais dispendioso

Refactoring – exemplos

- Reorganização da hierarquia de classes por forma a remover código repetido
- “Arrumar” e renomear métodos e atributos por forma a torna-los mais fácil de perceber o que são
- Substituir código por chamadas a métodos que foram/estão incluídos em bibliotecas

Testes antes da implementação

- Testes em XP
  - Conceito central
  - Todos os testes são executados após qualquer alteração
- Características dos testes em XP
  - Testes antes do desenvolvimento
  - Testes incrementais criados a partir dos cenários
  - Participação dos utilizadores no desenvolvimento e validação dos testes
  - Uso de ferramentas que permitam a execução automática dos testes
    - Sempre que uma nova versão/release é criada

Test-driven development

- Escrever testes antes do código
  - Ajuda a clarificar os requisitos que devem ser implementados
- Testes
  - São escritos como programas
  - Podem ser executados de forma automática
  - Incluem um “marcador” que indica que foi executado com sucesso
  - Baseados em *frameworks* de testes, e.g.: Junit
- Todos os testes são executados automaticamente
  - Quando uma funcionalidade nova é adicionada
  - Permite verificar que a nova funcionalidade não introduziu erros

Participação do cliente

- Papel do cliente no processo de testes
  - Desenvolver testes de aceitação para os cenários que vão ser desenvolvidos na próxima versão/release do sistema
- Cliente faz parte da equipa de desenvolvimento
  - Responsável por escrever testes à medida que o software é desenvolvido
  - Todo o código novo é validado por forma a garantir que está de acordo com o que o cliente precisa
- No entanto
  - Quem assume o papel de cliente tem pouco tempo para dedicar ao processo de desenvolvimento
  - Cliente assume que fornecer os requisitos é suficiente
  - Podem não estar dispostos a participar no processo de testes

Caso de testes - Exemplo: Verificação de doses

Test 4: Dose checking

Input:
1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

Tests:
1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose * frequency is too high and too low.
4. Test for inputs where single dose * frequency is in the permitted range.

Output:
OK or error message indicating that the dose is outside the safe range.

Automação de testes

- Testes escritos como componentes executáveis
  - Antes da implementação das funcionalidades
  - Testes devem
    - ser isolados/stand-alone
    - invocar a funcionalidade a ser testada
    - verificar se o output está de acordo com as especificações
- Usar um framework de testes
  - Facilita a criação e execução de testes
- Com testes automáticos
  - É possível executar os testes de forma fácil rápida
  - Sempre que uma funcionalidade é adicionada, os testes podem ser executados.
    - Problemas criados pelas novas funcionalidades são detetados imediatamente

Programação por pares

- Programadores trabalham aos pares
  - Desenvolvimento de código em conjunto
- O código é partilhado por vários elementos da equipa
  - O conhecimento sobre o código é mais partilhado por toda a equipa
- Processo informal para rever código
  - Cada linha de código é vista por mais do que uma pessoa
- Encoraja o *refactoring*, pois toda a equipa beneficia

Programação por pares

- Programadores sentam-se na mesma estação de trabalho para desenvolver software
- Pares são criados de forma dinâmica por forma a que todos os membros da equipa trabalhem com todos os restantes membros durante o processo de desenvolvimento
- A partilha de conhecimento que ocorre na programação por pares é muito importante e reduz o risco para o projecto se algum membro da equipa sair
- Existem estudos que mostram que a produtividade da programação por pares é equivalente a duas pessoas a trabalhar independentemente

Gestão de projetos ágeis

Gestão de projetos ágeis

- Responsabilidade dos gestores de projecto
  - Garantir que o software é entregue a tempo, de acordo com o orçamento planeado

- Gestão de projetos típica
  - Baseada em planos
  - Gestores criam um plano
    - Definem o que deve ser entregue e quando
    - Quem vai trabalhar trabalhar em que tarefa

- Gestão de projetos ágeis
  - Obriga a uma abordagem diferente
    - Adaptada aos métodos incrementais e às praticas usadas nos métodos ágeis

Scrum

- Método ágil de desenvolvimento de software
  - Com foco na gestão do desenvolvimento iterativo em vez de aspetos específicos dos métodos ágeis

- Composto por 3 etapas
  - Fase inicial
    - Planeamento geral do projeto
    - Estabelecer objetivos do projeto e desenhar a arquitetura do software
  - Sprint cycles
    - Em cada ciclo é desenvolvido parte do sistema
  - Fim do projeto
    - Terminar documentação necessária
    - Avaliar todos os aspetos do projeto
      - Que lições se podem tirar do projeto e aplicar nos próximos

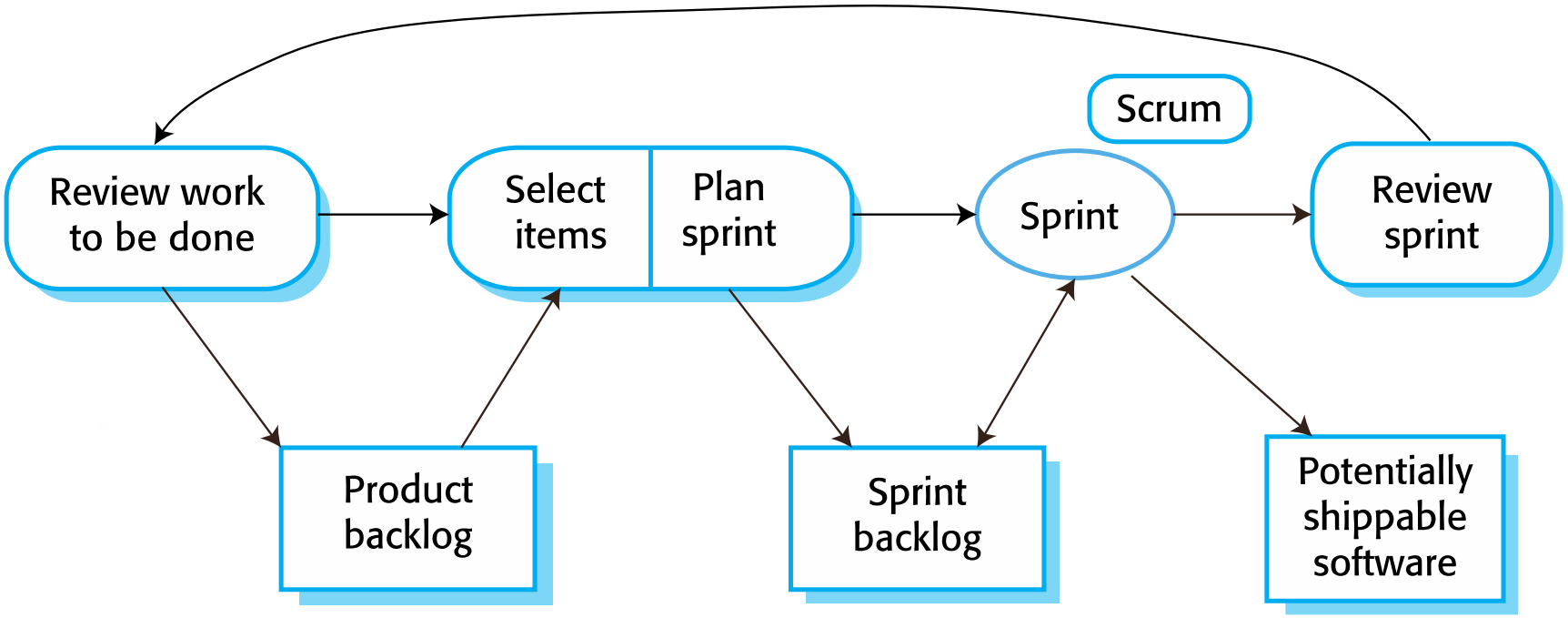
Scrum - terminologia

Scrum term	Definition
Development team	A self-organizing group of software developers, which should be no more than 7 people. They are responsible for developing the software and other essential project documents.
Potentially shippable product increment	The software increment that is delivered from a sprint. The idea is that this should be 'potentially shippable' which means that it is in a finished state and no further work, such as testing, is needed to incorporate it into the final product. In practice, this is not always achievable.
Product backlog	This is a list of 'to do' items which the Scrum team must tackle. They may be feature definitions for the software, software requirements, user stories or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation.
Product owner	An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development and continuously review the product backlog to ensure that the project continues to meet critical business needs. The Product Owner can be a customer but might also be a product manager in a software company or other stakeholder representative.

Scrum - terminologia

Scrum term	Definition
Scrum	A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team.
ScrumMaster	The ScrumMaster is responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference. The Scrum developers are adamant that the ScrumMaster should not be thought of as a project manager. Others, however, may not always find it easy to see the difference.
Sprint	A development iteration. Sprints are usually 2-4 weeks long.
Velocity	An estimate of how much product backlog effort that a team can cover in a single sprint. Understanding a team's velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance.

Scrum - ciclos de sprints



Scrum sprint cycle

- Sprints têm uma duração fixa
  - Normalmente 2 a 4 semanas
- Ponto de partida
  - *Backlog* do produto: lista de trabalho por fazer
- Fase de seleção
  - São escolhidas funcionalidades do *backlog*
    - Para serem implementadas durante o sprint
  - Envolve todos os membros da equipa de desenvolvimento que interagem com o cliente

Sprint cycle

- Depois de escolhidas as funcionalidades, dá-se inicio ao processo de desenvolvimento
  - Durante esta etapa a equipa de desenvolvimento é “isolada” do cliente
  - Comunicações passam pelo “Scrum master”
- O papel do “Scrum master” é o de proteger equipa de desenvolvimento de distracções externas
- No fim de cada sprint, o trabalho realizado é revisto e apresentado aos interessados.
  - Dá-se início ao novo sprint

Scrum – trabalho em equipa

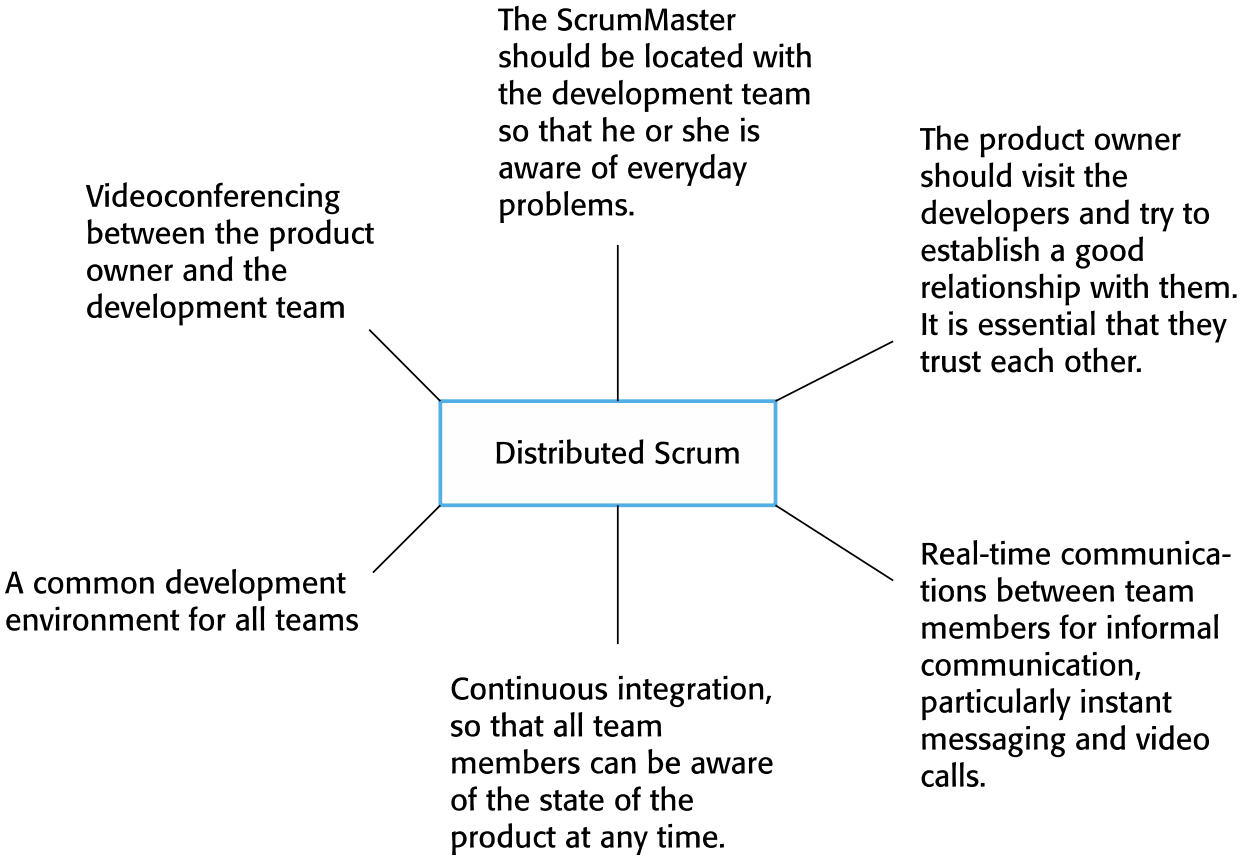
- Scrum master
  - Facilitador que marca reuniões
  - Gere o *backlog* que tem de ser feito
  - Regista decisões
  - Mede progresso do projeto, considerando o *backlog*
  - Comunica com os elementos externos ao projeto

- Toda a equipa participa em pequenas reuniões diárias
  - todos os membros partilham informação
  - descrevem o seu progresso desde a ultima reunião
  - quais os problemas que encontraram/surgiram
  - toda a equipa sabe o que se passa no projeto
    - Se surgirem problemas, consegue-se planear o trabalho futuro por forma a lidar com esses problemas

Scrum – benefícios

- Produto é partido em partes mais pequenas e de fácil compreensão
- Requisitos não estáveis não afetam o progresso do projeto
- Toda a equipa tem visibilidade sobre todo o projeto
  - Comunicação na equipa é melhorada
- Clientes têm entregas incrementais a tempo
  - Feedback mais rápido sobre o produto
- Aumento da confiança entre clientes e equipa de desenvolvimento

Scrum distribuído



Escalar métodos ágeis

Escalar métodos ágeis

- Métodos ágeis provaram ser bem sucedidos
  - Projetos de pequena e média escala
  - Equipas pequenas
- Razão para o seu sucesso
  - Melhoria na comunicação
  - Facilita a colaboração
- Escalar métodos ágeis
  - Projetos maiores e de maior duração
  - Várias equipas de desenvolvimento
    - Por vezes em diferentes locais geográficos

“Scaling out” e “Scaling up”

- Scaling up
  - Usar métodos ágeis
  - Projetos que não podem ser desenvolvidos por equipas pequenas
- Scaling out
  - Introdução de métodos ágeis em organizações grandes
  - Com grande experiência em desenvolvimento de software
    - Usando métodos não ágeis
- Importante ao escalar métodos ágeis: manter os conceitos fundamentais
  - Planeamento flexível, versões/releases frequentes, integração contínua, test-driven development e boa comunicação entre as equipas

Métodos ágeis - Problemas

- Muito informal
  - Incompatível com alguns aspetos legais relacionados com a contratação de serviços
- Apropriados para software novo
  - Não recomendados para manutenção de software
  - No entanto, grande parte dos custos de software está associado à manutenção de software existente
- Desenhados para equipas pequenas
  - No entanto, atualmente, grande parte de software envolve equipas distribuídas por todo o mundo

Métodos ágeis e manutenção de software

- A maior parte das empresas passam mais tempo a fazer manutenção de software, do que a desenvolver software novo.
  - É importante que os métodos ágeis garantam a manutenção de software, bem como o seu desenvolvimento inicial
- Dois aspetos importantes
  - Sistemas desenvolvidos através de métodos ágeis são mantiveis?
    - Considerando o foco no desenvolvimento e a minimização da documentação.
  - Métodos ágeis podem ser usados de forma eficiente para evoluir um sistema por forma a responder aos pedidos de alterações?
- Podem surgir problemas se a equipa de desenvolvimento original não for mantida

Métodos ágeis e manutenção de software

- Principais problemas
  - Falta de documentação do produto
  - Manter os clientes envolvidos no processo de desenvolvimento
  - Manter a continuidade da equipa de desenvolvimento

- Métodos ágeis
  - Toda a equipa deve conhecer e perceber todo o projeto

- Problema para projetos de longa duração
  - A equipa de desenvolvimento muda ao longo do tempo

Métodos ágeis e métodos baseados em planos

- Grande parte dos projetos incluem elementos de processos baseados em planos e de métodos ágeis. O correto equilíbrio depende de:
  - É importante ter uma especificação muito detalhada antes de começar a implementação?
    - Se sim, deve ser usado um método baseado em planos.
  - Uma entrega incremental, onde é possível obter um feedback rápido do cliente, é realista?
    - Se sim, pode-se usar métodos ágeis.
  - Qual o tamanho dos sistemas a serem desenvolvidos?
    - Os métodos ágeis são mais eficazes quando a equipa de desenvolvimento é pequena e consegue comunicar de forma não formal.

Aspetos técnicos, humanos e organizacionais

- Que tipo de sistema está a ser desenvolvido?
  - Sistemas que requerem muita análise previa podem necessitar de métodos baseados em planos
- Qual o tempo de vida esperado do sistema?
  - Sistemas com um tempo de vida grande normalmente necessitam de documentação mais exaustiva
- Que tecnologias estão disponíveis para apoiar o processo desenvolvimento?
  - Existem muitas ferramentas boas para apoiar os métodos ágeis
- Qual a organização da equipa?
  - Se a equipa de desenvolvimento estiver distribuída de forma geográfica, pode ser necessário uma documentação mais exaustiva.

Aspetos técnicos, humanos e organizacionais

- Existem aspetos culturais ou organizacionais que possam afetar o processo de desenvolvimento?
  - Tradicionalmente existe uma “cultura” de métodos baseados em planos. Pode ser um problema.
- Qual a qualidade da equipa de desenvolvimento?
  - Há quem defenda que é necessária uma equipa com mais qualidade quando usando métodos ágeis. Usando métodos baseados em planos, o trabalho “limita-se” a traduzir o desenho em código.
- O sistema está sujeito a regulamentação externa?
  - Se o sistema tiver de ser aprovado por um regulador externo, é possível que seja necessária uma documentação exaustiva do sistema.

Aspetos técnicos, humanos e organizacionais

- Qual a qualidade da equipa de desenvolvimento?
  - Existe quem defenda que os métodos ágeis requerem uma equipa de desenvolvimento de qualidade superior.
- Como é que a equipa de desenvolvimento está organizada?
- Quais as ferramentas de apoio ao desenvolvimento que estão disponíveis?

Bibliografia

- Software Engineering. Ian Sommerville. 10th Edition. Addison-Wesley. 2016. Capítulo 3.