Desenvolvimento Ágil de Software

Metodologias de Desenvolvimento de Software

Pedro Salgueiro
pds@di.uevora.pt
CLV-256

Desenvolvimento Ágil de Software

- Outline
  - Métodos ágeis
  - Desenvolvimento plan-driven e ágil
  - Extreme programming
  - Scrum

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 reflectir as dinâmicas dos negócios

- 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

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 objectivo a entrega rápida de software rapidamente e a evolução rápida por forma a responder às alterações dos requisitos
- Objectivo 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-media escala
- Desenvolvimento de sistemas à medida,
  - onde existe uma participação activa do cliente no processo de desenvolvimento
  - não existam muitas regras externas que afectem o software
- Foco em equipas pequenas e muito bem integradas
  - Difícil de escalar métodos ágeis para sistemas grandes

Problemas dos métodos ágeis

- Pode ser difícil manter o interesse dos clientes no processo de desenvolvimento do software
- Os membros da equipa de desenvolvimento podem não estar “preparados” para o envolvimento intenso característico dos métodos ágeis
- Estabelecer prioridades pode ser complicado quando existem vários interessados no sistema
- Manter simplicidade requer trabalho extra
- Contractos podem ser um problema para os processos iterativos

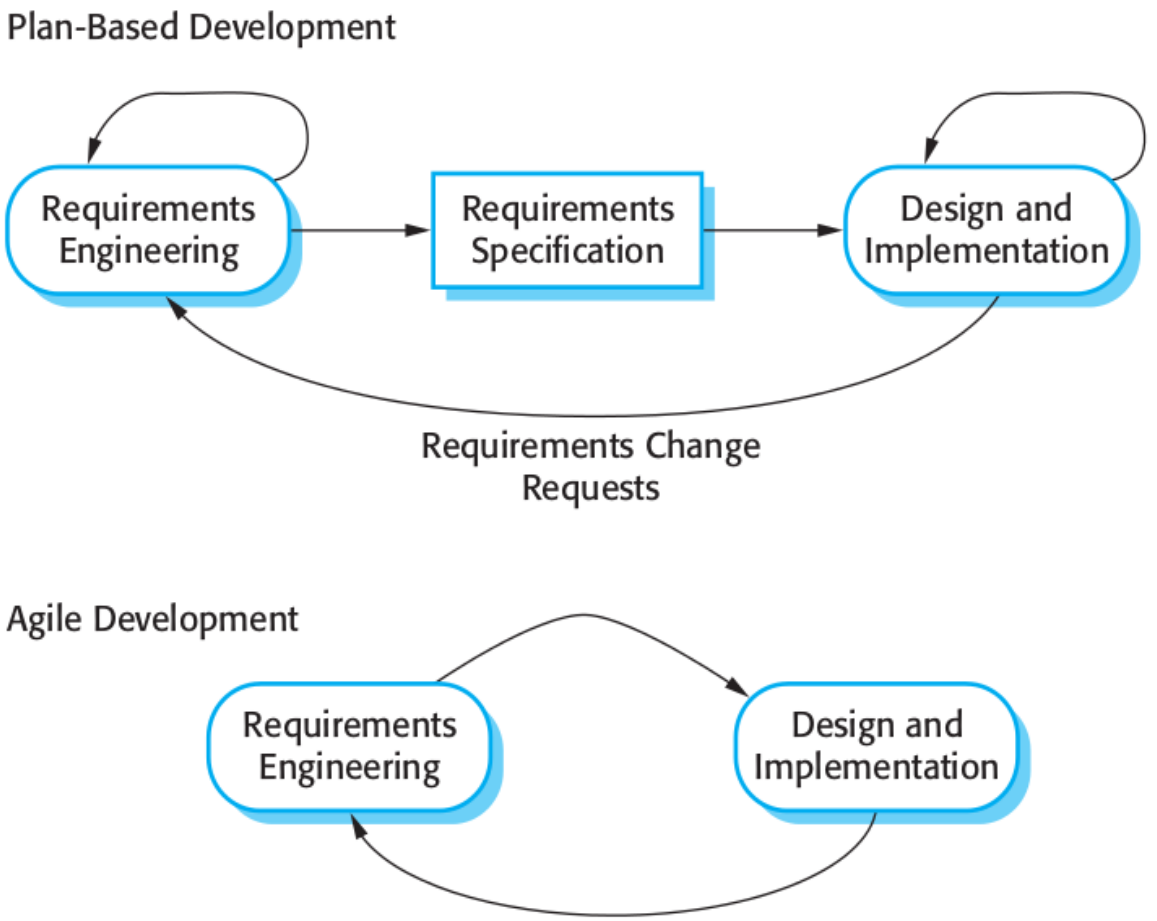
Métodos ágeis e manutenção de software

- 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 aspectos 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 métodos baseados em planos

- Desenvolvimento baseado em planos
  - Etapas de desenvolvimento separadas
  - Resultados produzidos no fim de cada etapa
  - Etapas previamente planeadas
  - Iterações dentre de cada etapa
- Desenvolvimento ágil
  - Especificação, desenho, implementação e testes estão intercalados
  - Outputs do processo de desenvolvimento são “negociados” durante todo o processo

Métodos ágeis e métodos baseados em planos



Aspectos técnicos, humanos e organizacionais

- Grande parte dos projectos incluem elementos de processos baseados em planos e de métodos ágeis. O correcto 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.

Aspectos 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.

Aspectos técnicos, humanos e organizacionais

- Existem aspecto culturais ou organizacionais que possam afectar 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.

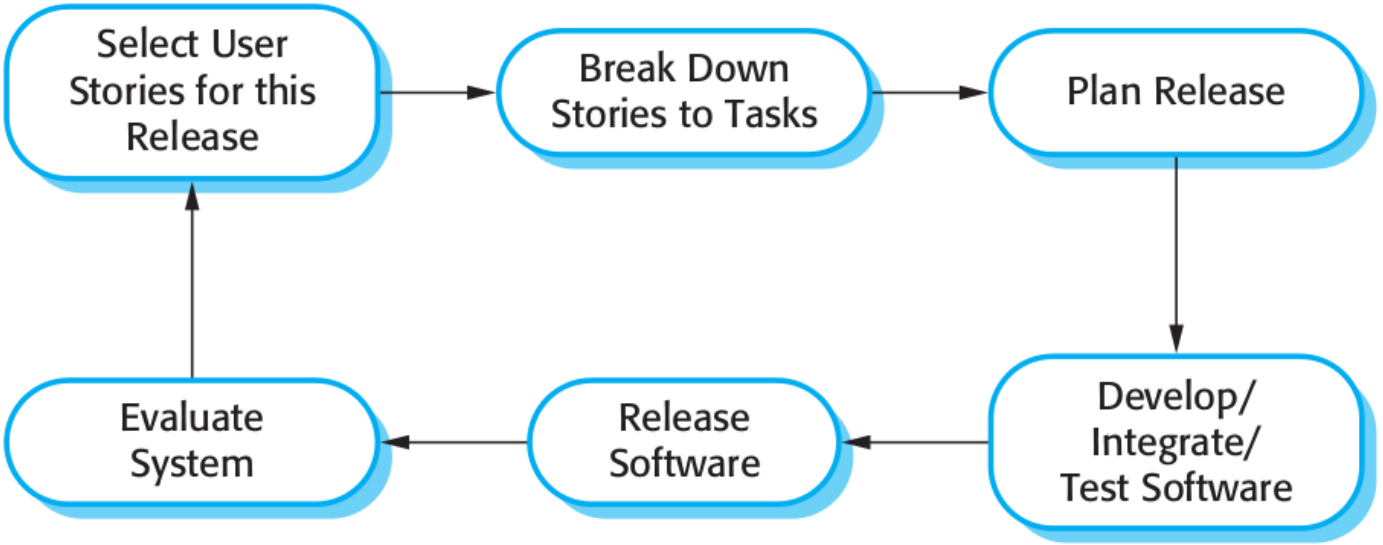
Extreme programming

- Um dos métodos ágeis mais conhecidos e usados
- Extreme Programming (XP) leva o conceito de desenvolvimento iterativo ao "extremo"
  - Podem existir novas versões várias vezes por dia;
  - Os incrementos são entregues aos clientes a cada 2 semanas;
  - Todos os testes devem ser executados/corridos em cada versão. As versões apenas são aceites se os testes correrem com sucesso;

XP e princípios ágeis

- Desenvolvimento incremental é apoiado através de “releases”/versões pequenas e frequentes
- Envolvimento do cliente permanente com a equipa de desenvolvimento
- Pessoas e não processos:
  - Programação por pares
  - O código nunca pertence apenas a uma pessoa
- Alterações são apoiadas através de versões/”releases” regulares
- Manter a simplicidade através de *refactoring* constante do código

Extreme programming - ciclo de entregas/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.

Extreme programming – boas práticas

- Em XP, o cliente ou utilizador faz parte da equipa de desenvolvimento e é responsável por fazer decisões sobre os requisitos

- Requisitos de utilizador 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

User story - exemplo

Prescribing Medication

Kate is a doctor who wishes to prescribe medication for a patient attending a clinic. The patient record is already displayed on her computer so she clicks on the medication field and can select current medication', 'new medication' or 'formulary'.

If she selects 'current medication', the system asks her to check the dose. If she wants to change the dose, she enters the dose and then confirms the prescription.

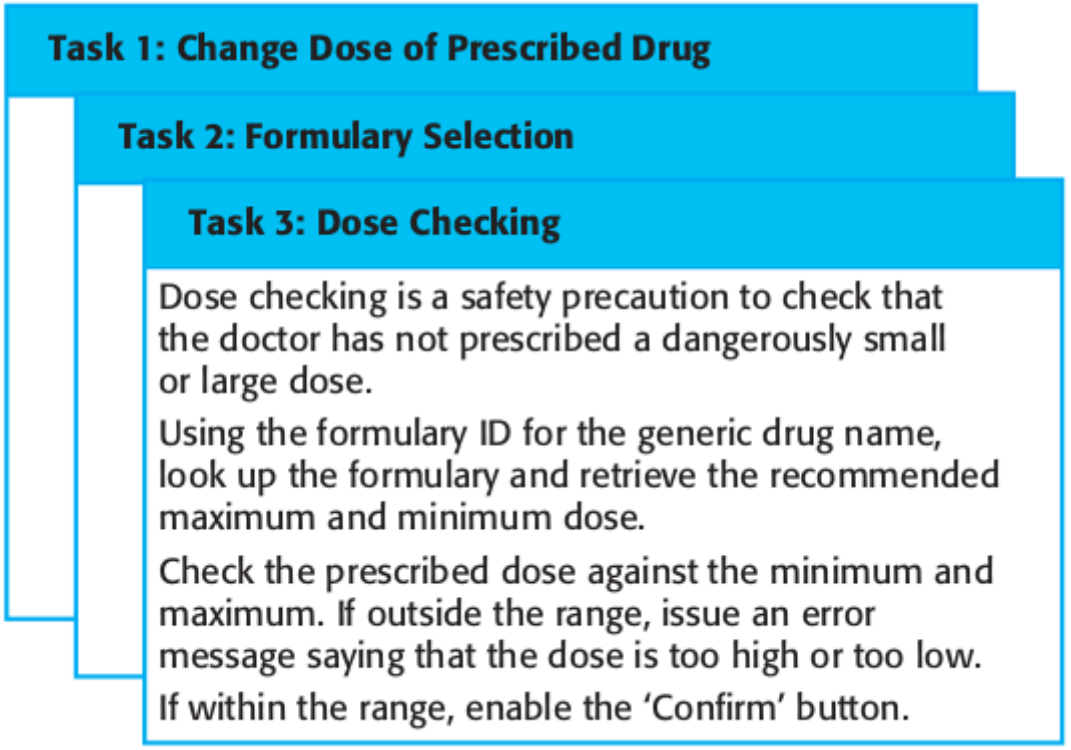
If she chooses 'new medication', the system assumes that she knows which medication to prescribe. She types the first few letters of the drug name. The system displays a list of possible drugs starting with these letters. She chooses the required medication and the system responds by asking her to check that the medication selected is correct. She enters the dose and then confirms the prescription.

If she chooses 'formulary', the system displays a search box for the approved formulary. She can then search for the drug required. She selects a drug and is asked to check that the medication is correct. She enters the dose and then confirms the prescription.

The system always checks that the dose is within the approved range. If it isn't, Kate is asked to change the dose.

After Kate has confirmed the prescription, it will be displayed for checking. She either clicks 'OK' or 'Change'. If she clicks 'OK', the prescription is recorded on the audit database. If she clicks on 'Change', she reenters the 'Prescribing medication' process.

User story - exemplo - divisão em tarefas



XP e alterações

- Bom senso no desenvolvimento de software
  - Desenhar software para ser alterado
  - Vale a pena gastar tempo e esforço a antecipar alterações
    - Reduz os custos a longo prazo
- XP defende que este esforço não vale a pena
  - As alterações não podem ser antecipadas de forma fiável
- Em vez disso, propõe a melhoria constante de código (*refactoring*), para 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 arquitectura
  - Processo mais dispendioso

Refactoring – exemplos

- Re-organização da hierarquia de classes por forma a remover código repetido
- “Arrumar” 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 e XP

- Testes são fulcrais em XP
  - Software é testando antes de ser implementado
- Testes em XP:
  - Testes antes do desenvolvimento
  - Testes incrementais a partir de cenários
  - Participação dos utilizadores no desenvolvimento e validação dos testes
  - Execução automática de testes por forma a garantir que todos os testes são corridos para todas as releases/versões

Testes antes do código

- Escrever testes antes do código ajuda a clarificar os requisitos a serem implementados
- Testes são escritos como programas
  - Poderem ser executados de forma automática
  - Recorre-se a frameworks de testes, e.g.: JUnit
- Todos os testes (novos e já existentes) são executados quando se adiciona uma nova funcionalidade
  - Permite verificar se a nova funcionalidade não introduziu erros

Descrição de testes - Exemplo

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

- Automação de testes significa que os testes são escritos como componentes executáveis
  - Devem ser stand-alone
  - Devem simular a submissão de inputs e verificar se o resultado está de acordo com a especificação
  - O framework Junit ajuda na criação de testes e na sua execução automática
- O uso de testes automáticos facilita a forma como estes podem ser executados
  - Sempre que uma nova funcionalidade é adicionada, os testes podem ser executados de forma simples, identificando potenciais problemas

Testes em XP - Dificuldades

- Os programadores preferem programar em vez de testar
  - Muitas vezes fazem atalhos quando estão a escrever testes
  - Exemplo: não incluir todas as excepções que podem ocorrer
- Alguns testes podem ser difíceis de escrever de forma incremental
- É difícil de perceber a completude/cobertura dos testes
  - Muitos testes não significa que abranja todas as situações

Programação por pares

- Em XP, os programadores trabalham aos pares, 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
- Existem estudos que mostram que a produtividade da programação por pares é equivalente a duas pessoas a trabalhar independentemente

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

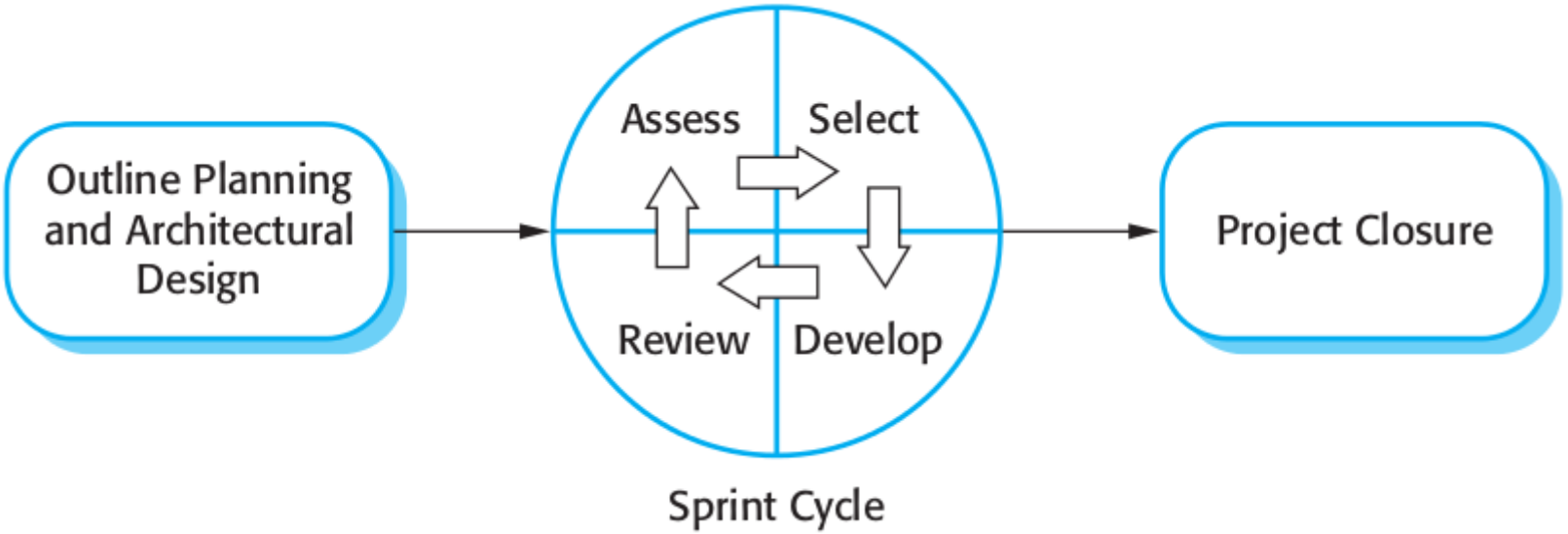
Programação por pares - vantagens

- Código não tem apenas um “dono”
  - Não existe apenas um responsável pelo código
- É um processo informal de revisão de código
  - Cada linha de código é vista pelo menos por duas pessoas
- Ajuda a fazer o refactoring do código

Scrum

- Abordagem genérica aos métodos ágeis
  - Com foco na gestão do desenvolvimento iterativo em vez de aspectos específicos dos métodos ágeis
- Composto por 3 etapas
  - Fase inicial
    - Planeamento geral do projecto
    - Estabelecer objectivos do projecto e desenhar a arquitectura do software
  - Sprint cycles
    - Em cada ciclo é desenvolvido parte do sistema
  - Fim do projecto
    - Terminar documentação necessária
    - Avaliar todos os aspectos do projecto
      - Que lições se podem tirar do projecto e aplicar nos próximos

Processo Scrum



Sprint cycle

- Sprints têm uma duração fixa
  - Entre 2 e 4 semanas
  - Correspondem a uma release/versão do sistema
- O ponto de partida para o planeamento é o “backlog” do produto
  - Lista de tarefas que ainda não estão feitas
- Fase de selecção envolve todos os membros da equipa que interagem com o cliente
  - Seleccionar quais as funcionalidades que serão desenvolvidas no sprint

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 projecto, considerando o backlog
  - Comunica com os elementos externos ao projecto

- 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 projecto
    - 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 afectam o progresso do projecto
- Toda a equipa tem visibilidade sobre todo o projecto
  - Comunicação na equipa é melhorada
- Clientes têm entregas a tempo
  - Feedback mais rápido sobre o produto
- Confiança entre clientes e equipa de desenvolvimento

Bibliografia

- Software Engineering, Ian Sommerville, 9th Edition, Addison-Wesley, 2010. Capítulo 3.