Metodologias de Desenvolvimento de Software (MDS)

Mestrado em Desenvolvimento de Software e Sistemas Interactivos

Apontamentos elaborados por JCMetrôlho

1

NOTA PRÉVIA

- Estes apontamentos, que serão apresentados e discutidos nas aulas TP, não dispensam a consulta da documentação complementar, sobre as várias metodologias, que será também fornecida ao longo do semestre aos alunos.

2

APRESENTAÇÃO

- Corpo docente
- Contactos do docente
- Horários letivos e de atendimento
- Forma de avaliação
  - Consultar a Ficha de Unidade Curricular (UC) aprovada e publicada para o presente ano letivo
- Bibliografia recomendada
- Forma de iteração (sumários, pautas, moodle, etc)
- Marcação de datas de avaliação

OBJECTIVOS DA UC DE MDS

"Reforçar o conhecimento e experiência prática dos alunos sobre desenvolvimento ágil de software: planeamento de produtos, releases e iterações, qualidade, planeamento, refactoring, pair programming, integração contínua. Dar a conhecer algumas das variantes mais conhecidas de processos ágeis. Apreensão dos conhecimentos primordialmente através da sua aplicação prática em casos de estudo reais a desenvolver ao longo do semestre usando metodologias como o SCRUM ou XP (Extreme Programming). Utilização de ambientes de desenvolvimento integrado (IDE) que suportem e incentivem o desenvolvimento ágil de software: Ex. Outsystems Community Edition. Pretende-se que os alunos adquiram competências para serem capazes de conhecer bem técnicas e técnicas de desenvolvimento ágil de software (Scrum, Extreme Programming e outras metodologias conhecidas). Desenvolver espírito crítico para identificar o paradigma a aplicar num dado problema. Dominar técnicas e ambientes de desenvolvimento de Software. " Fonte: FUC da UC.

CONTEÚDOS PROGRAMÁTICOS DA UC DE MDS

" Introdução às Metodologias de desenvolvimento de software; Processos de desenvolvimento de software (tradicionais, interativos, incrementais, Ágeis); Comparativo da metodologia Waterfall com metodologias ágeis; A necessidade de métodos ágeis para desenvolvimento de software; Planeamento; Introdução às *user stories*; Software ágil: valores e princípios fundamentais; Principais Práticas das Metodologias Ágeis; Principais abordagens usadas nas metodologias ágeis; Scrum; eXtreme Programming; Qualidade de Software: Test driven development."

Fonte: FUC da UC.

5

CONTEÚDOS PROGRAMÁTICOS DA UC DE MDS

Na componente mais laboratorial será usada a plataforma OUTSYSTEMS

https://www.outsystems.com/platform/

6

METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE

José Carlos Metrôlho,

7

NOTA PRÉVIA

- Estes slides são uma adaptação dos slides disponibilizados pelos autores do livro "*Software Engineering: Modern Approaches: An Object-Oriented Perspective*", 2nd Edition, Eric J. Braude e Michael E. Bernstein, John Wiley & Sons inc, ISBN-10: 0471692085 (2011).
- Também são usados como fontes os slides disponibilizados pelos autores e informação que consta dos seguintes livros:
  - Frank Tsui, Orlando Karam e barbara Bernal, Essentials of Software Engineering, 3rd Edition, Jones&Bartlett Learning, ISBN 978-1-4496-9199-8 (2014);
  - Sérgio Guerreiro, Introdução à Engenharia de Software, FCA, ISBN:978-972-722-795-2 (2015).
  - Ian Sommerville, Software Engineering, 10 edição, Pearson Education Limited, ISBN: 10: 1-292-09613-6 (2016).
- A leitura dos livros referidos anteriormente é recomendada aos alunos como complemento ao apresentado nas aulas.
- Além do anteriormente referido, são usadas ainda outras fontes (livros, artigos, documentos web, etc.) que serão oportunamente citadas no rodapé dos respetivos slides.

8

A SABER …

- O que é o processo de desenvolvimento?
- Quais as principais atividades dos processos de desenvolvimento de software?
- Quais os principais tipos de processos/metodologias de desenvolvimento?
- Que processo escolher para determinado projeto?
- Ferramentas CASE, para apoio ao processo de desenvolvimento

9

O PROCESSO É DEFINIDO NA FASE DE PLANEAMENTO

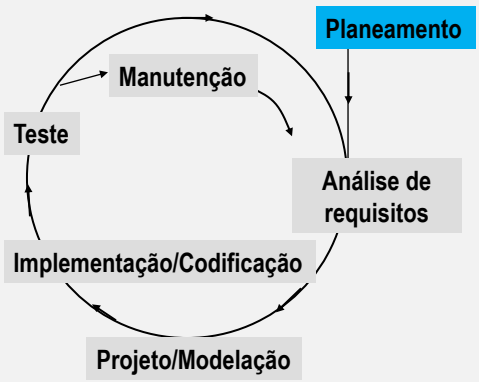


Figura é tradução da existente em "*Software Engineering: Modern Approaches: An Object-Oriented Perspective*", 2nd Edition, Eric J. Braude e Michael E. Bernstein, John Wiley & Sons inc, ISBN-10: 0471692085 (2011).

10

PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

" O processo de desenvolvimento de software consiste na definição sistemática de etapas distribuídas de comunicação, cooperação, execução e gestão tendo como objetivo construir, de forma eficiente e eficaz(no menos tempo possível), um novo produto de software devidamente verificado e validado. *In Sérgio Guerreiro, Introdução à Engenharia de Software, FCA, ISBN:978-972-722-795-2 (2015).*

11

PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

- Um projeto de Software é composto por várias atividades
  - E.g. planeamento, desenho, implementação, teste, etc.
- As atividades são organizadas em fases
- O processo de software*:*
  - Estabelece a ordem e a frequência das fases
  - Especifica os critérios de passagem de uma fase para a seguinte
  - Define os resultados intermédios e finais previstos do projeto

12

ATIVIDADES DE **EXECUÇÃO** DO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

- Comunicação
- Planeamento
- Modelação do sistema
- Implementação
- Instalação
- Formação
- Manutenção

13

ATIVIDADES DE **SUPORTE** DO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

- Gestão do projeto (acompanhamento e controlo)
- Revisões técnicas e formais
- Garantia da qualidade do software
- Gestão de risco

14

OS PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

- NÃO devem acrescentar
  - sobrecarga
  - documentação desnecessária
  - derrapagem de prazos
  - etc.
- Se aplicados corretamente ajudam a:
  - Cumprir prazos
  - Mais qualidade do software
  - Mais sustentabilidade (facilidade de manter ao longo do tempo) do software

15

OS PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

- **Clássicos**
  - Definem uma sequência de etapas estática e que satisfaz exclusivamente os requisitos identificados . É assumido que é possível identificar totalmente as necessidades para o produto de software final.
    - Ex: Cascata (*Waterfall*), Espiral (*Spiral*), Incremental.
- **Ágeis**
  - Ao contrário dos processos clássicos, têm a preocupação de avaliar constantemente o que está a ser feito e verificar se está correto.
    - Ex: SCRUM, eXtreme Programming (XP), Kanban, *Rational Unified Process* (RUP).

16

FASES DO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Começo
O produto é concebido e definido

Planeamento
Cronograma inicial, recursos e custos são estimados

Análise de Requisitos
Especificar o que o produto deve fazer. **O que implementar**?

Desenho
Especificar as partes e como devem encaixar. **Como implementar?**

Implementação
Escrever código

Teste
Executar o produto com dados de teste

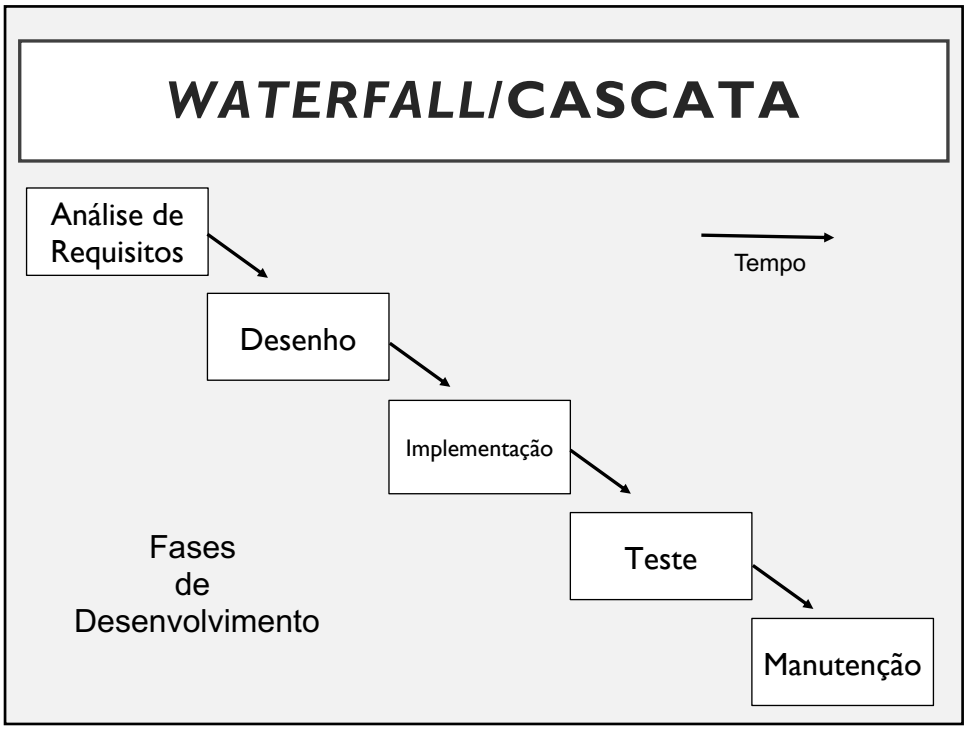
Manutenção
reparar defeitos e adicionar capacidade

17

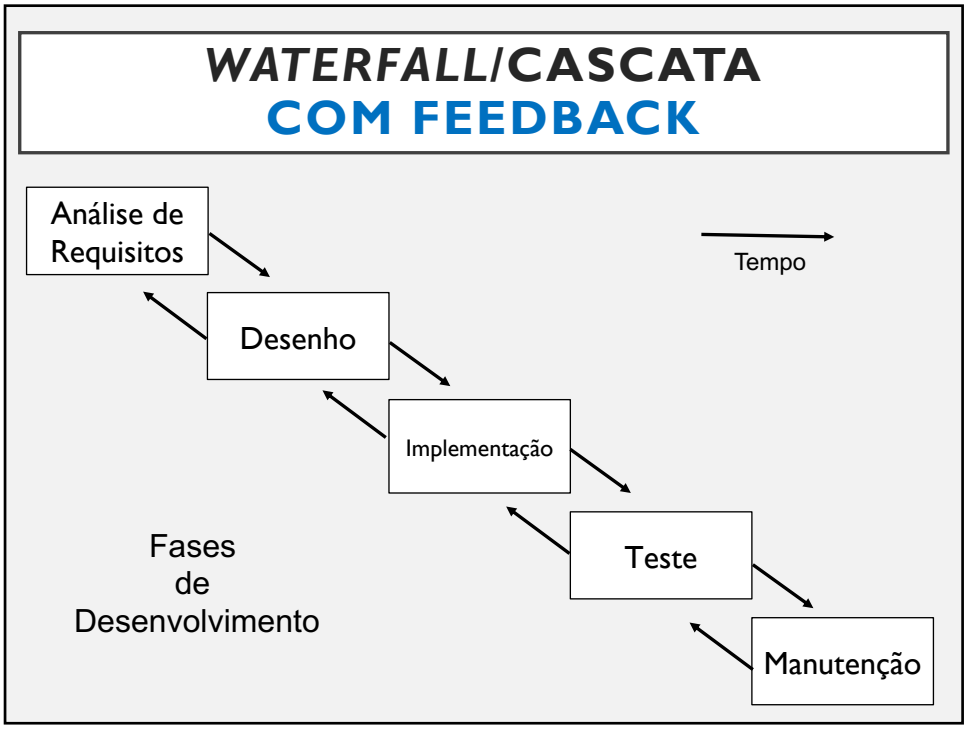
PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE CLÁSSICOS

- *Waterfall* | Cascata
- Iterativo e Incremental
- Prototipagem
- Espiral
- *Rapid Application Development (*RAD)

18


WATERFALL/CASCATA
Análise de Requisitos
Desenho
Implementação
Teste
Manutenção
Tempo
Fases de Desenvolvimento

19


WATERFALL/CASCATA
COM FEEDBACK
Análise de Requisitos
Desenho
Implementação
Teste
Manutenção
Tempo
Fases de Desenvolvimento

20

VANTAGENS DO PROCESSO WATERFALL/CASCATA

- Simples e fácil de entender por todos os intervenientes
- Usado à muitos anos
- Fácil de gerir devido à sequencialidade na execução
- Facilita a alocação de recursos
- Funciona bem para projetos onde os requisitos sejam bem conhecidos à partida

Leitura complementar recomendada: The Waterfall Model: Advantages, disadvantages, and when you should use it

21

DESVANTAGENS DO PROCESSO WATERFALL/CASCATA

- Requisitos têm de ser conhecidos muito cedo
- Difícil de estimar de forma confiável
- *Feedback* dos *stakeholders* só após a fase de testes
- Problemas podem só ser descobertos na fase terminal do projeto
- Falta de paralelismo
- Uso ineficiente de recursos

Leitura complementar recomendada: The Waterfall Model: Advantages, disadvantages, and when you should use it

22

INCREMENTAL

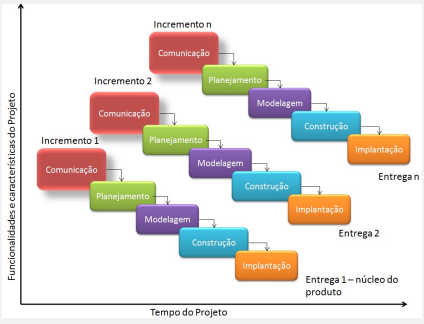
- O desenvolvimento incremental é uma estratégia de preparação e programação em que várias partes do sistema são desenvolvidas em momentos ou taxas diferentes e integradas à medida que são concluídas.

Fonte: Using Both Incremental and Iterative Development, Dr. Alistair Cockburn, Humans and Technology, https://pdfs.semanticscholar.org/1194/2a1d43800a6bbd67ba9cca1dbb4c244cd762.pdf

23

INCREMENTAL

"O desenvolvimento é dividido em etapas, denominadas "incrementos", que produzirão incrementalmente o sistema, até a sua versão final."
"Em cada incremento é realizado todo o ciclo do desenvolvimento de software, do planeamento aos testes do sistema já em funcionamento. Cada etapa produz um sistema totalmente funcional, apesar de ainda não cobrir todos os requisitos."



Fonte: http://engenhariadesoftwareuesb.blogspot.com/2012/12/blog-post.html

24

INCREMENTAL
PROS

- Apresenta vantagens, especialmente se os requisitos não estão claros inicialmente.
- O primeiro incremento é normalmente constituído do núcleo do sistema. Isto é, os requisitos básicos são implementados, e os detalhes suprimidos. Esse produto será entregue para uma avaliação, que poderá detetar, inicialmente, problemas que poderiam ser de dimensões muito maiores se detetados somente na entrega do produto final.
- Outra vantagem para o desenvolvedor é que, em contato com o sistema, o cliente esclarece seus requisitos e suas prioridades para os próximos incrementos, além de contar com os serviços da versão já produzida.
- A construção de um sistema menor é sempre menos arriscada que a construção de um grande;
- Se um grande erro é cometido, apenas o último incremento é descartado;
- Reduzindo o tempo de desenvolvimento de um sistema, as chances de mudanças nos requisitos do usuário durante o desenvolvimento são menores.

Fonte: http://engenhariadesoftwareuesb.blogspot.com/2012/12/blog-post.html

25

INCREMENTAL
CONTRAS

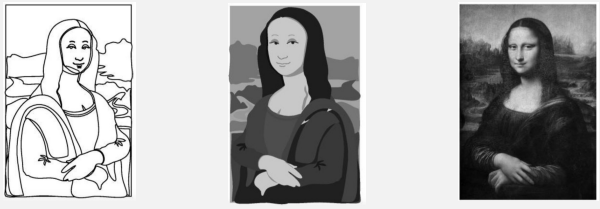
- É difícil dividir um projeto em incrementos autónomos
- O somatório de recursos e tempo consumidos pelos diversos incrementos é maior do que se o projeto fosse desenvolvido integralmente em conjunto
- A gestão do processo de desenvolvimento é mais complexa do que no caso do waterfall
- O cliente ou utilizador final obtém o sistema de forma gradual. Terá de ser capaz de disponibilizar o produto de software ao longo do tempo.
- A sobreposição de vários incrementos pode causar dificuldades, se houver dependências sequenciais de informação entre os componentes.

26

ITERATIVO

"Desenvolvimento iterativo é uma estratégia de agendamento de retrabalho, na qual é reservado um tempo para rever e melhorar partes do sistema."

No desenvolvimento iterativo o ciclo de vida de um projeto é composto por várias iterações. Uma iteração inclui tarefas como análise requisitos, desenho/modelação, implementação, teste e implementação, em várias proporções, dependendo da altura em que são realizadas no ciclo de desenvolvimento. As iterações nas fases iniciais e de elaboração concentram-se nas atividades de gestão, requisitos e desenho/modelação. As iterações na fase de construção concentram-se no desenho, implementação e teste. E as iterações na fase de transição concentram-se no teste e implementação.



Fonte: Using Both Incremental and Iterative Development, Dr. Alistair Cockburn, Humans and Technology, https://pdfs.semanticscholar.org/1194/2a1d43800a6bbd67ba9cca1dbb4c244cd762.pdf

27

ITERATIVO

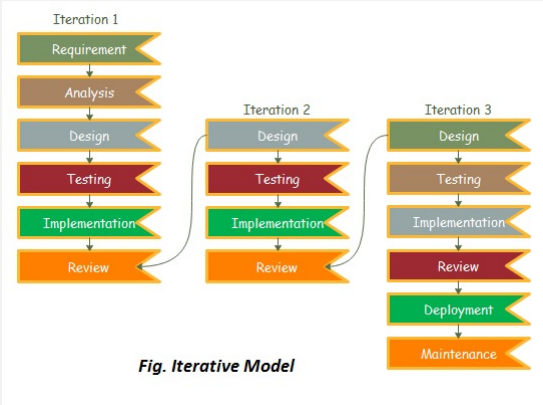


Fig. Iterative Model

Fonte: https://www.javatpoint.com/software-engineering-iterative-model

28

ITERATIVO
PROS E CONTRAS

Pros

- Entregas intermédias facilitam a identificação e correção de erros entre os componentes do software.
- Necessidades não especificadas nas fases iniciais podem ser desenvolvidas nos incrementos.
- Os incrementos podem ser desenvolvidos por menos pessoas.
- Entrega dos incrementos permite aferir melhor o o cumprimento do prazo especificado.
- O Modelo iteratvo inclui o uso do software pelo utilizador para que as mudanças sejam feitas de acordo com o seu feedback.
- É flexível e fácil de gerir

Contras

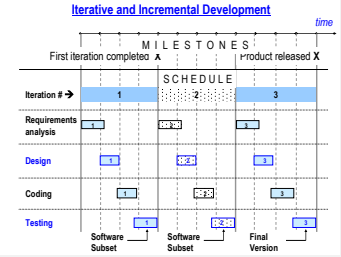
- Podem surgir problemas relativos à arquitetura do sistema, porque nem todos os requisitos estão reunidos na frente de todo o ciclo de vida do software.
- O modelo iterativo precisa ser relativamente pequeno.
- O número de iterações não pode ser definido no início do processo.
- O fim do processo não pode ser previamente definido.
- Gestão e manutenção do sistema completo podem ser complexas.

Fonte: https://www.slideshare.net/danieladeoliveirafranciosi/modelo-incremental-engenharia-de-software

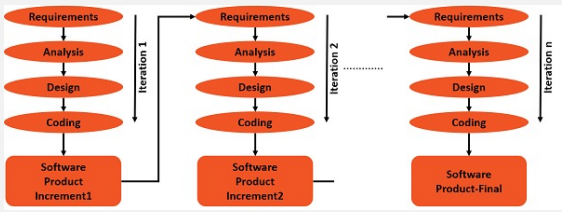
29

ITERATIVO E INCREMENTAL

No processo incremental e iterativo, inicialmente, uma implementação parcial de um sistema total é construída para que fique em um estado de entrega. Em iterações posteriores, maior funcionalidade é adicionada. Os defeitos, se houver, da entrega (iteração) anterior são corrigidos e o produto em funcionamento é entregue. O processo é repetido até que todo o desenvolvimento do produto esteja concluído. As repetições desses processos são chamadas **iterações**. No final de cada iteração, um **incremento** de produto é entregue.



Fonte: *Software Engineering: Modern Approaches: An Object-Oriented Perspective*", 2nd Edition, Eric J. Braude e Michael E. Bernstein, John Wiley & Sons inc, ISBN-10: 0471692085 (2011).



Fonte: https://www.tutorialspoint.com/adaptive_software_development/sdlc_iterative_incremental_model.htm

30

30

ITERATIVO E INCREMENTAL
PROS E CONTRAS

Pros

- Permite desenvolver requisitos priorizados primeiro.
- A entrega inicial do produto é mais rápida.
- Os clientes obtêm funcionalidades importantes cedo.
- Reduz o custo de entrega inicial.
- Cada versão é um incremento do produto, para que o cliente tenha um produto em funcionamento o tempo todo.
- O cliente pode dar feedback para cada incremento do produto, evitando surpresas no final do desenvolvimento.
- As alterações de requisitos podem ser facilmente acomodadas.

Contras

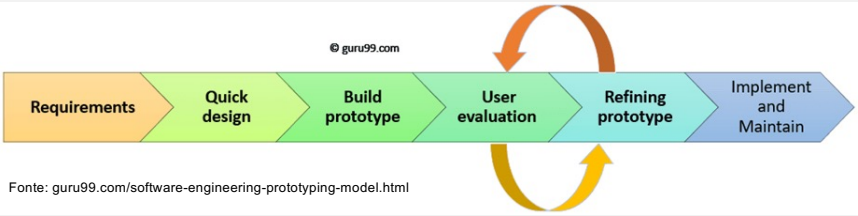
- Requer planeamento eficaz de iterações.
- Requer desenho eficiente para garantir a inclusão da funcionalidade necessária e fornecer alterações posteriormente.
- Requer definição antecipada de um sistema completo e totalmente funcional para permitir a definição de incrementos.
- São necessárias interfaces de módulo bem definidas, pois algumas são desenvolvidas muito antes de outras.
- O custo total do sistema completo pode ser maior..

Fonte: https://www.tutorialspoint.com/adaptive_software_development/sdlc_iterative_incremental_model.htm

31

31

PROTOTIPAGEM



Fonte: guru99.com/software-engineering-prototyping-model.html

Em Engenharia de Software, a metodologia Prototipagem é um modelo de desenvolvimento de software no qual um protótipo é construído, testado e melhorado, quando necessário, até que um protótipo aceitável seja alcançado.

1) São 6 as etapas do processo de prototipagem: 1) Recolha e análise de requisitos, 2) Projeto rápido, 3) Construir um protótipo, 4) Avaliação inicial do utilizador, 5) Protótipo com melhorias, 6) Implementar produto e manutenção.
2) Os tipos de modelos de prototipagem são: 1) Protótipos de descarte rápido 2) Protótipo evolutivo 3) Protótipo incremental 4) Protótipo extremo.
3) Reuniões regulares são essenciais para manter o projeto no prazo e evitar atrasos dispendiosos na abordagem de prototipagem.
4) A funcionalidade ausente pode ser identificada, o que ajuda a reduzir o risco de falha, pois a prototipagem também é considerada uma atividade de redução de risco.
5) A prototipagem pode incentivar solicitações de alteração excessivas.

32

TIPOS DE PROTOTIPAGEM

- Protótipo descartável
- Protótipo evolutivo
- Protótipo incremental
- Protótipo extremo

Fonte: guru99.com/software-engineering-prototyping-model.html

33

TIPOS DE PROTOTIPAGEM
PROTÓTIPO DESCARTÁVEL

O descarte rápido é baseado no requisito preliminar. É rapidamente desenvolvido para mostrar como o requisito será visualmente. O feedback do cliente ajuda a gerar alterações no requisito, e o protótipo é criado novamente até que o requisito seja avalisado.

Nesse método, um protótipo desenvolvido será descartado e não fará parte do protótipo final. Essa técnica é útil para explorar ideias e obter feedback instantâneo para os requisitos do cliente.

Fonte: guru99.com/software-engineering-prototyping-model.html

34

TIPOS DE PROTOTIPAGEM
PROTÓTIPO EVOLUTIVO

O protótipo desenvolvido é melhorado de forma incremental com base no feedback do cliente até que seja finalmente aceite. Isso ajuda a economizar tempo e esforço. Isso porque desenvolver um protótipo do zero para cada interação do processo pode às vezes ser muito frustrante.

Este modelo é útil para um projeto que utiliza uma nova tecnologia que não é bem compreendida. Também é usado para um projeto complexo em que todas as funcionalidades devem ser verificadas uma vez.

É útil quando o requisito não é estável ou não é entendido claramente no estágio inicial.

Fonte: guru99.com/software-engineering-prototyping-model.html

35

TIPOS DE PROTOTIPAGEM
PROTÓTIPO INCREMENTAL

Na prototipagem incremental, o produto final é dividido em diferentes pequenos protótipos e desenvolvido individualmente. Eventualmente, os diferentes protótipos são agrupados em um único produto. Este método é útil para reduzir o tempo de feedback entre o utilizador e a equipa de desenvolvimento.

Fonte: guru99.com/software-engineering-prototyping-model.html

36

TIPOS DE PROTOTIPAGEM
PROTÓTIPO EXTREMO

O método de prototipagem extrema é usado principalmente para desenvolvimento web.

É composto por três fases sequenciais:

1. Protótipo básico da página no formato HTML.
2. Simular o processo de dados usando uma camada de serviços de protótipo.
3. Os serviços são implementados e integrados ao protótipo final.

Fonte: guru99.com/software-engineering-prototyping-model.html

37

PROTOTIPAGEM
PROS E CONTRAS

Pros

- Os utilizadores estão ativamente envolvidos no desenvolvimento. Portanto, erros podem ser detetados no estágio inicial do processo de desenvolvimento de software.
- A funcionalidade ausente pode ser identificada, o que ajuda a reduzir o risco de falha, pois a prototipagem também é considerada uma atividade de redução de risco.
- Ajuda o membro da equipa a comunicar efetivamente
- A satisfação do cliente existe porque o cliente pode ver o produto em um estágio muito inicial.
- Quase não haverá chance de rejeição de software.
- O feedback mais rápido do utilizador ajuda a obter melhores soluções de desenvolvimento de software.
- Permite que o cliente compare se o código do software corresponde à especificação do software. Isso ajuda a descobrir a funcionalidade que falta no sistema.
- Também identifica as funções complexas ou difíceis.
- Incentiva a inovação e o desenho flexível.
- É um modelo simples e fácil de entender.
- O protótipo serve como base para derivar uma especificação do sistema.
- O protótipo ajuda a entender melhor as necessidades do cliente.
- Os protótipos podem ser alterados e até descartados.
- Um protótipo também serve como base para as especificações operacionais.
- Os protótipos podem oferecer formação antecipada para futuros utilizadores do sistema de software.

Fonte: guru99.com/software-engineering-prototyping-model.html

38

38

PROTOTIPAGEM
PROS E CONTRAS

Contras

- A prototipagem pode ser um processo lento e demorado.
- O custo do desenvolvimento de um protótipo pode ser considerável, especialmente se o protótipo for descartado..
- Algumas vezes, os clientes podem não estar dispostos a participar do ciclo de iteração por mais tempo.
- Pode haver muitas variações nos requisitos de software sempre que o protótipo é avaliado pelo cliente.
- Pode ser produzida má documentação porque os requisitos dos clientes estão em mudança.
- Pode ser difícil para os programadores de software acomodar todas as alterações exigidas pelos clientes.
- Depois de ver um modelo de protótipo inicial, os clientes podem pensar que o produto real será entregue em breve.
- O cliente pode perder o interesse no produto final quando não estiver satisfeito com o protótipo inicial.
- Os desenvolvedores que desejam criar protótipos rapidamente podem acabar criando soluções de desenvolvimento abaixo do padrão.

Fonte: guru99.com/software-engineering-prototyping-model.html

39

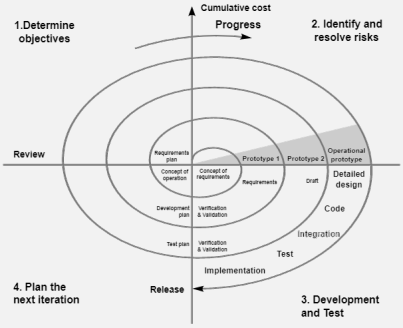
39

MODELO EM ESPIRAL

- Barry Boehm, 1988
- Um dos processos iterativos e incrementais mais antigos e conhecidos
- Processo orientado a riscos
- O projeto começa no centro e cada ciclo da espiral representa uma iteração
- O objetivo de cada ciclo é aumentar o grau de definição e implementação do sistema, enquanto diminui o grau de risco

40

MODELO EM ESPIRAL



Fases do modelo espiral

Planeamento: Nesta fase, a equipa analisa as demandas para ver se são viáveis. Se forem, as equipas começam a planear a fase de desenvolvimento.

Análise de risco: Aqui, a equipa estuda os possíveis problemas que podem surgir, bem como as possíveis maneiras de enfrentar esses desafios.

Engenharia: Com tudo planeado e previsto, os responsáveis por esta etapa começam a trabalhar.

Avaliação: Quando esta etapa termina, o cliente avalia o resultado e fornece feedback à equipa. Este feedback será usado para o planeamento da próxima iteração.

Fonte: https://www.intellectsoft.net/blog/spiral-model-sdlc/

41

MODELO EM ESPIRAL (BARRY BOEHM, 1988)

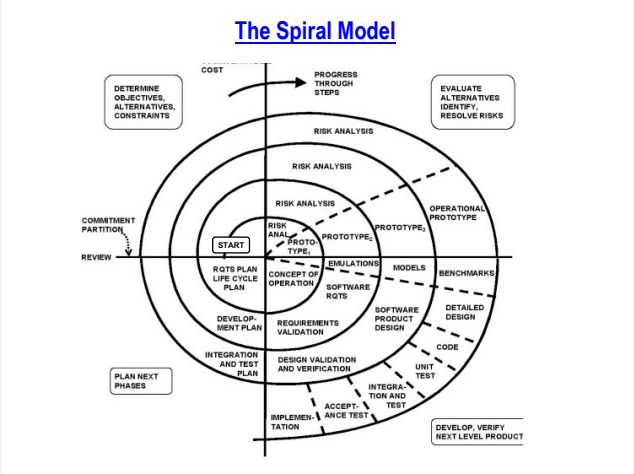


Figura de "*Software Engineering: Modern Approaches: An Object-Oriented Perspective*", 2nd Edition, Eric J. Braude e Michael E. Bernstein, John Wiley & Sons inc, ISBN-10: 0471692085 (2011).

42

MODELO EM ESPIRAL
ETAPAS ITERATIVAS

1. Identificar objetivos e restrições críticas
2. Avaliar alternativas de projeto e processo
3. Identificar riscos
4. Resolver um subconjunto de riscos usando análise, emulação, *benchmarks*, modelos e protótipos
5. Desenvolver as entregas do projeto, incluindo requisitos, desenho, implementação e teste
6. Planejar para os próximos e futuros ciclos - atualizar o plano do projeto, incluindo cronograma, custo e número de iterações restantes
7. Revisão, pelas partes interessadas, das entregas da iteração e seu compromisso de prosseguir com base no objetivo atingido

43

MODELO EM ESPIRAL
PROS

- Os **riscos são geridos no início** e durante todo o processo - os riscos são reduzidos antes que se tornem problemáticos
- O **software evolui à medida que o projeto avança** - erros e alternativas pouco adequadas são eliminados mais cedo.
- O **planeamento faz parte do processo** - cada ciclo inclui uma etapa de planeamento para ajudar a monitorar e manter um projeto no caminho certo.

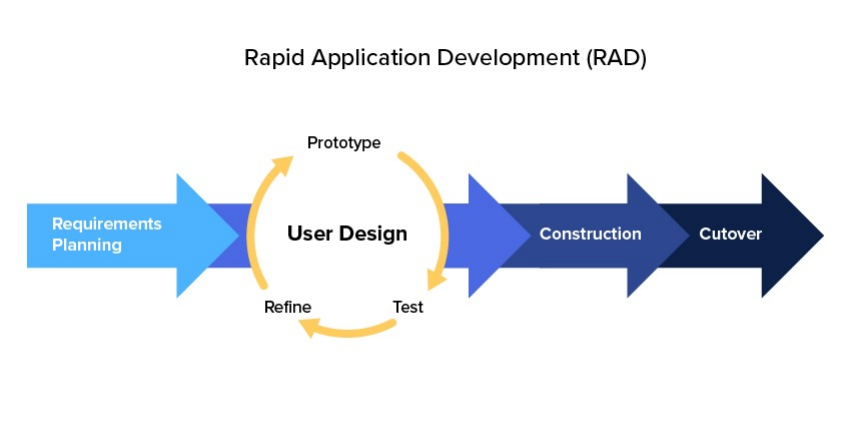
44

MODELO EM ESPIRAL MODEL
CONTRAS

- **Complexo** - a análise de risco requer conhecimento altamente especializado. Existe inevitavelmente alguma sobreposição entre as iterações.
- Pode ser **exagerado para projetos pequenos** - a complexidade pode não ser necessária para projetos menores. Não faz sentido se o custo da análise de risco for uma parte importante do custo geral do projeto.

45

MODELO
RAPID APPLICATION DEVELOPMENT



Fonte: https://kissflow.com/rad/rapid-application-development/

46

MODELO RAD

Etapas

- Definir os requisitos
- Construir o Protótipo
- Receber *feedback*
- Finalizar software

Fonte: https://kissflow.com/rad/rapid-application-development/

47

MODELO RAD
PROS

- Os requisitos podem ser alterados a qualquer momento
- Incentiva e prioriza o *feedback* do cliente
- As avaliações são rápidas
- O tempo de desenvolvimento é reduzido drasticamente
- Mais produtividade com menos pessoas
- O tempo entre protótipos e iterações é curto
- A integração não é um problema, pois integra-se desde o início do projeto.

Fonte: https://kissflow.com/rad/rapid-application-development/

48

MODELO RAD CONTRAS

- Precisa de forte colaboração entre membros da equipa
- Não é para equipas grandes
- Precisa de desenvolvedores altamente qualificados
- Necessita do utilizador ao longo do ciclo de vida do produto
- Apenas adequado para projetos com pouco tempo de desenvolvimento
- Mais complexo de gerir do eu outros modelos
- Sistemas devem poder ser modularizados

Fonte: https://kissflow.com/rad/rapid-application-development/

49

IBM RATIONAL UNIFIED PROCESS (RUP)

O processo pode ser descrito em duas dimensões ou em dois eixos:

- o eixo horizontal representa o tempo e mostra o especto dinâmico do processo à medida que é promulgado, e é expresso em termos de ciclos, fases, iterações e marcos.
- o eixo vertical representa o aspeto estático do processo: como é descrito em termos de atividades, artefactos, trabalhadores e fluxos de trabalho.

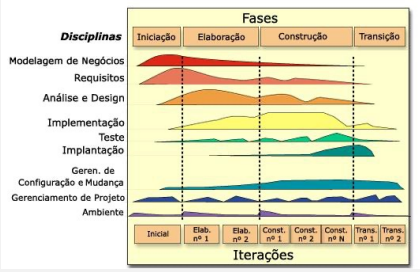


Imagem retirada de https://pt.wikipedia.org/wiki/IBM_Rational_Unified_Process

Fontes: https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf , https://alvinalexander.com/java/java_oo/java_oop_tutorial_1_6Rational_Unified_Process.shtml e ftp://public.dhe.ibm.com//software/pdf/br/RUP_DS.pdf

50

RUP: FASE DE INICIAÇÃO

- O resultado da fase inicial é:
  - Um documento de visão: uma visão geral dos requisitos do projeto principal, dos principais recursos e das principais restrições.
  - Um modelo inicial de casos de uso (10% a 20%) concluído.
  - Um glossário inicial do projeto (opcionalmente pode ser parcialmente expresso como um modelo de domínio).
  - Um caso comercial inicial, que inclui o contexto comercial, critérios de sucesso (projeção de receita, mercadoreconhecimento etc.) e previsão financeira.
  - Uma avaliação de risco inicial.
  - Um plano de projeto, mostrando fases e iterações.
  - Um modelo de negócios, se necessário.
  - Um ou vários protótipos

Fonte:
https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

51

RUP: FASE DE ELABORAÇÃO

- O resultado da fase de elaboração é:
  - Um modelo de casos de uso (pelo menos 80% completo) - todos os casos e atores de uso foram identificados e a maioria das descrições de casos de uso foram desenvolvidas.
  - Requisitos suplementares que capturam os requisitos não funcionais e quaisquer requisitos que não sejamassociado a um caso de uso específico.
  - Uma descrição da arquitetura de software.
  - Um protótipo de arquitetura executável.
  - Uma lista de riscos revisada e um caso de negócios revisado.
  - Um plano de desenvolvimento para o projeto geral, incluindo o plano de projeto de granulação grossa, mostrando iterações "e critérios de avaliação para cada iteração.
  - Um caso de desenvolvimento atualizado especificando o processo a ser usado.
  - Um manual preliminar do usuário (opcional).

Fonte: https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

52

RUP: FASE DE CONSTRUÇÃO

- O resultado da fase de construção é no minimo:
  - O produto de software integrado nas plataformas adequadas.
  - Os manuais do usuário.
  - Uma descrição da versão atual.

Fonte:
https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

53

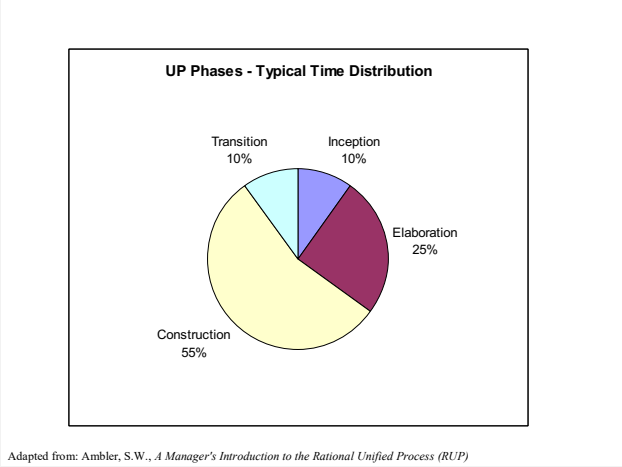
RUP: FASE DE TRANSIÇÃO

- O resultado da fase de transição incorpora:
  - "Teste beta" para validar o novo sistema contra as expectativas do utilizador
  - operação paralela com um sistema legado que está substituindo
  - conversão de bases de dados operacionais
  - treino de utilizadores e de quem mantem o produto
  - implantar o produto nas equipes de marketing, distribuição e vendas

Fonte:
https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

54

RUP: PROPORÇÃO TEMPORAL DAS FASE



Adapted from: Ambler, S.W., *A Manager's Introduction to the Rational Unified Process (RUP)*

Fonte: "*Software Engineering: Modern Approaches: An Object-Oriented Perspective*", 2nd Edition, Eric J. Braude e Michael E. Bernstein, John Wiley & Sons inc, ISBN-10: 0471692085 (2011).

55

RUP: ITERAÇÕES

- Cada fase do RUP pode ser dividida em iterações.
- Uma iteração é um ciclo completo de desenvolvimento que resulta em uma release (interna ou externa) de um produto executável, um subconjunto do produto final em desenvolvimento, que cresce de forma incremental de iteração para iteração até conseguir o resultado final.
- Explora os benefícios de uma abordagem iterativa, relativamente ao processo tradicional em cascata, O processo iterativo tem as seguintes vantagens:
  - Riscos são mitigados antes
  - Mudança é mais gerenciável
  - Maior nível de reutilização
  - A equipa do projeto pode aprender ao longo do caminho
  - Melhor qualidade geral

Fonte:
https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

56

RUP: PROS

A abordagem do RUP tem vantagens sobre os processos sequenciais tradicionais, nomeadamente:

- Melhor gestão
- Feedback regular às partes interessadas
- Gestão de risco aprimorado
- Implementação dos requisitos reais
- Conhecer cedo o que realmente funciona
- Os desenvolvedores concentram-se no que realmente interessa

Fontes: ASHRAF ANWAR,A Review of RUP (Rational Unified Process) , International Journal of Software Engineering (IJSE), Volume (5) : Issue (2) : 2014, pg 8-24, URL: https://www.cscjournals.org/manuscript/Journals/IJSE/Volume5/Issue2/IJSE-142.pdf .

Scott W. Ambler , A Manager's Introduction to The Rational Unified Process (RUP), Ambisoft, URL: http://www.ambysoft.com/downloads/managersIntroToRUP.pdf

57

USER STORY

Uma declaração breve e simples de requisitos da perspectiva do utilizador.

Quem?
O quê ?
Porque?

Como <função de utilizador>,
Quero <objectivo>,
Para <benefício desejado>.

Critérios de aceitação

58

CRITÉRIOS DE ACEITAÇÃO

- "Um critério de aceitação é um conjunto de condições ou regras de negócios que a funcionalidade ou recurso deve satisfazer e atender, para ser aceite pelo Dono do Produto / Partes Interessadas."

Fonte e exemplos em: https://rubygarage.org/blog/clear-acceptance-criteria-and-why-its-important

59

EXEMPLO

- Como **consumidor**, quero **poder ver o meu consumo diário de energia** para **poder começar a entender como reduzir meus custos ao longo do tempo**

- Critério(s) de aceitação:
  - Ler os dados do medidor a cada 10 segundos e exibir no portal com incrementos de 15 minutos e exibir no visor doméstico todas as leituras
  - Ler os medidores para obter novos dados, conforme disponíveis, e mostrar no portal a cada hora e no display em casa após cada leitura
  - Nenhuma tendência de vários dias por enquanto (outra história).
  - Etc ...

Fonte: A User Story Primer By Dean Leffingwell with Pete Behrens, 2009, Leffingwell, LLC. All Rights Reserved.

60

OS 3 C DE UMA USER STORY

1. **Cartão**- Um cartão com 2 a 3 frases força a brevidade. Descreve apenas o tópico do item, uma descrição de alto nível do comportamento do sistema desejado e por que é importante.
2. **Conversa**- Tem de haver espaço a debate para apurar com mais detalhe o que é pretendido com a US. A discussão entre os utilizadores de destino, equipa, dono do produto e outras partes interessadas, permite determinar mais detalhe necessário para implementar a intenção da US.
3. **Confirmação**- representa as condições de satisfação que serão aplicadas para determinar se é cumprido ou não a intenção e os requisitos mais detalhados.

Fonte: https://www.visual-paradigm.com/guide/agile-software-development/what-is-user-story/

61

INVEST EM USER STORIES

- Independent
- Negotiable
- Valuable
- Estimable
- Small
- Testable

Ver significado em:
http://xp123.com/xplor/xp0308/index.shtml

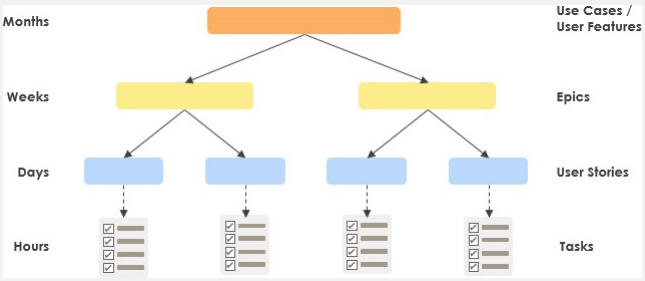
62

PRODUCT BACKLOG

- Uma lista priorizada de todas as US que podem ser entregues
- Novos itens podem ser adicionados a qualquer momento no Backlog do Produto
- Os itens são definidos e priorizados pelo Dono do Produto, com a contribuição de outras pessoas
- Os membros da equipa estimam itens no Backlog do Produto em relação um ao outro, usando uma escala predeterminada

63

GRANULARITDADE DOS ITEMS DO PRODUCT BACKLOG



Fonte: https://www.visual-paradigm.com/scrum/theme-epic-user-story-task/

64

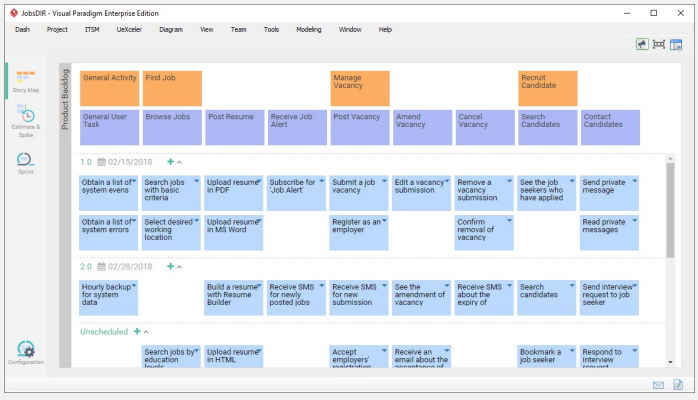
GRANULARIDADE DOS ITEMS DO *PRODUCT BACKLOG*

- Temas ou épicos não podem ser concluídos num sprint; portanto, são divididos em mais US e subsequentemente num grupo de tarefas relacionadas. Épicas são entregues em versões. Mas mesmo pequenas USde épicos diferentes podem ter algo em comum. Esse grupo de US é chamado de tema.

Fonte: https://www.visual-paradigm.com/scrum/theme-epic-user-story-task/

65

USER STORY MAPPING & PRODUCT BACKLOG



Fonte: https://www.visual-paradigm.com/scrum/theme-epic-user-story-task/

66

CICLOS DE VIDA DUMA US

Vários estados possíveis:

- Pendente
- *To-do*
- Discutindo
- Em desenvolvimento
- Confirmando
- Acabado

Fonte: https://www.visual-paradigm.com/guide/agile-software-development/what-is-user-story/

67

BENEFICIOS DAS USER STORIES

- Enfatizam a comunicação verbal.
- São compreensíveis tanto pelos clientes quanto pelos desenvolvedores, incentivando maiores níveis de participação do cliente.
- São do tamanho adequado para o planeamento.
- São adequadas para o desenvolvimento iterativo.
- O conhecimento acumula-se dentro da equipa, não apenas o Dono do Produto.

Fonte: https://www.thedroidsonroids.com/blog/how-to-write-user-stories-why-they-are-crucial-for-successful-app-development

68

PROCESSOS ÁGEIS

Manifesto Ágil
(valores *core*)

Indivíduos e Interações	**sobre**	Processos e Ferramentas
Software Funcionando	**sobre**	Documentação Abrangente
Colaboração do Cliente	**sobre**	Negociação de Contrato
Responder à Mudança	**sobre**	Seguir um Plano

Fonte: https://agilemanifesto.org/

69

PROCESSOS ÁGEIS

12 Princípios
(como as pessoas da organização se devem comportar umas com as outras)

Fonte: https://www.visual-paradigm.com/scrum/agile-manifesto-and-agile-principles/

70

PROCESSOS ÁGEIS

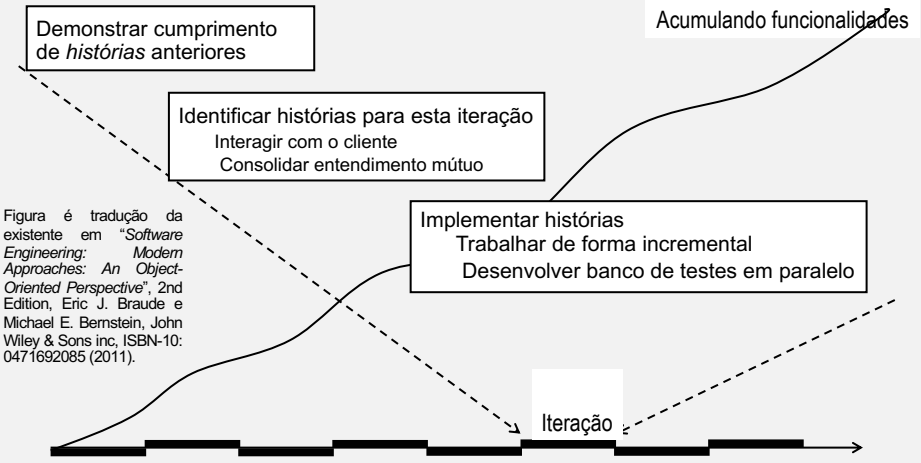
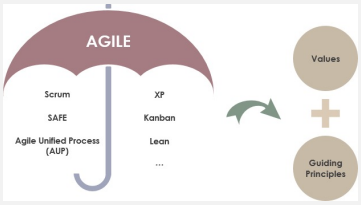


Figura é tradução da existente em "*Software Engineering: Modern Approaches: An Object-Oriented Perspective*", 2nd Edition, Eric J. Braude e Michael E. Bernstein, John Wiley & Sons inc, ISBN-10: 0471692085 (2011).

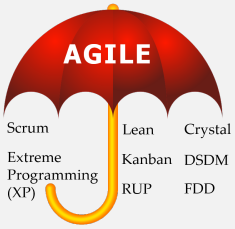
tempo; iterações com duração igual →

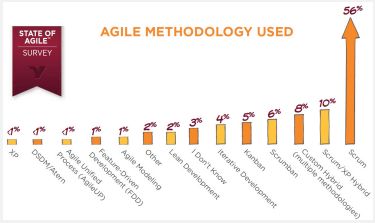
71

PROCESSOS ÁGEIS



Fonte: https://www.visual-paradigm.com/scrum/agile-manifesto-and-agile-principles/





Fonte: https://masterofproject.com/blog/3525/agile-frameworks-methodologies

72

PROCESSOS ÁGEIS
SCRUM

Scrum: *"A* ***framework*** *within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value."*

Fonte: ScrumGuides.org, The Scrum Guide™

Meet Jeff Sutherland

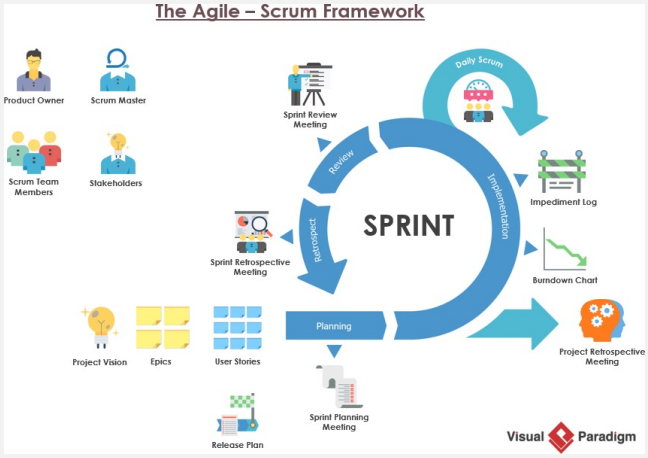
Jeff is the co-creator of Scrum and a leading expert on how the framework has evolved to meet the needs of today's business..

Meet Ken Schwaber

Ken Schwaber co-developed the Scrum process with Jeff Sutherland in the early 1990s to help organizations...

73

PROCESSOS ÁGEIS
SCRUM



Fonte: https://www.visual-paradigm.com/scrum/what-is-agile-and-scrum/

74

PROCESSOS ÁGEIS
SCRUM

Funções/Papéis/*Roles*

Product Owner / Dono do Produto (1 elemento)

- Age como uma única voz
- Sabe o que precisa ser construído e em que sequência deve ser feito
- Normalmente, um gerente de produto
- Responsável por maximizar o valor do produto resultante
- Responsável pela gestão do *Backlog* do Produto

Scrum Master

- Representa a gestão do projeto
- Normalmente preenchido por um gestor de projeto ou *Team Leader*
- Responsável por promover valores e práticas de Scrum
- Cabe-lhe a tarefa de remover impedimentos

Equipa de Desenvolvimento

- 3 a 9 elementos
- Funções cruzadas (controlo de qualidade, programadores, designers de interface, etc.)
- Os membros devem estar a tempo inteiro
- A equipa é auto-organizada
- Mudanças de membros devem ocorrer apenas entre *sprints*

Esta informação não dispensa a leitura de: https://www.scrumguides.org/scrum-guide.html

75

PROCESSOS ÁGEIS
SCRUM

Processo

Kickoff Meeting

- Uma forma especial de *Sprint Planning Meating*
- Reunião antes do início do Projeto

Sprint Planning Meating

- Reunião no início de cada Sprint entre o Dono do produto, o Scrum Master e a equipa
- Máximo de 8 horas para um sprin de um mês
- Confirmar o *Backlog* do Produto e determinar a meta do sprint. (Participantes: *Product Owner*, Scrum Master, Equipa Scrum)
- Criar o *Backlog* do Sprint (Participantes: Scrum Master e Equipa Scrum)

Sprint

- Uma iteração de um mês, durante a qual são incrementadas funcionalidades do produto
- Nenhuma influência externa deve interferir com a equipa Scrum durante o Sprint
- Cada Sprint começa com o *Daily Scrum Meating*

Esta informação não dispensa a leitura de: https://www.scrumguides.org/scrum-guide.html

76

76

PROCESSOS ÁGEIS
SCRUM

Daily Scrum Meating

Processo

- Uma sessão curta realizada todos os dias antes que a equipa comece a trabalhar
- Participantes: Scrum Master (que modera) e a equipa Scrum
- Todos os membro da equipa devem esclarecer 3 questões:
  - O que fizestes desde o último Scrum?
  - O que vais fazer no próximo Scrum?
  - O que está está a impedir de avançar o trabalho?
- **NÃO** é uma sessão de solução de problemas
- **NÃO** é uma maneira de coletar informações sobre os que estão atrasados
- É uma reunião em que os membros da equipa se comprometem uns com os outros e com o Scrum Master
- É bom para o Scrum Master acompanhar o progresso da equipa

77

Esta informação não dispensa a leitura de: https://www.scrumguides.org/scrum-guide.html

77

PROCESSOS ÁGEIS
SCRUM

Sprint Review Meating

Processo

- É realizada no final de cada Sprint
- Funcionalidades criadas durante o Sprint são apresentadas ao Dono do Produto
- Esta é uma reunião de no máximo quatro horas para Sprints de um mês.
- Esta é uma reunião informal, não uma reunião de status, e a apresentação do Incremento visa obter *feedback* e promover a colaboração.
- A Revisão da Sprint inclui os seguintes elementos:
  - Os participantes incluem a equipa Scrum e as principais partes interessadas, convidadas pelo Dono do produto;
  - O Dono do produto explica quais itens do Backlog do produto foram “Concluídos” e o que não foi "Concluído";
  - A equipa de desenvolvimento discute o que correu bem durante o Sprint, quais problemas que surgiram e como esses problemas foram resolvidos;
  - A equipa de desenvolvimento demonstra o trabalho "concluído" e responde a perguntas sobre o incremento;
  - O Dono do produto discute o Backlog do produto como está. Projeta provavelmente datas de entrega e metas com base no progresso até a data (se necessário);
  - Todo o grupo colabora no que fazer em seguida, para que a Sprint Review forneça contribuição valiosa para o subsequente *Sprint Planning*;
  - Revisão de como o mercado ou o uso potencial do produto pode ter mudado o que é a coisa mais valiosa a seguir;
  - Revisão do cronograma, orçamento, recursos potenciais e mercado para os próximos lançamentos antecipados de funcionalidade ou capacidade do produto.

78

Esta informação não dispensa a leitura de: https://www.scrumguides.org/scrum-guide.html

78

PROCESSOS ÁGEIS SCRUM

Sprint Retrospective

Processo

- A Retrospetiva do Sprint é uma oportunidade para a equipa Scrum se auto-avaliar e criar um plano para melhorias a serem implementadas durante a próxima Sprint.
- A Retrospetiva do Sprint ocorre após a Sprint Review e antes do próximo planeamento do Sprint. Esta é uma reunião de no máximo três horas para Sprints de um mês.
- O objetivo da Retrospetiva da Sprint é:
  - Inspecionar como foi o último Sprint em relação a pessoas, relacionamentos, processos e ferramentas;
  - Identificar e ordenar os principais itens que foram bem e as possíveis melhorias;
  - Criar um plano para implementar melhorias na maneira como a equipa Scrum faz seu trabalho.

Esta informação não dispensa a leitura de: https://www.scrumguides.org/scrum-guide.html

79

79

PROCESSOS ÁGEIS SCRUM

Artefactos

Product Backlog

- O Product Backlog é uma lista ordenada de tudo o que se sabe ser necessário no produto. É a única fonte de requisitos para quaisquer alterações a serem feitas no produto. O Dono do Produto é responsável pelo Backlog do Produto, incluindo seu conteúdo, disponibilidade e pedido.
- O Product Backlog nunca está completo. O desenvolvimento inicial estabelece os requisitos inicialmente conhecidos e melhor compreendidos. O Product Backlog evolui à medida que o produto e o ambiente em que será usado evoluem. O Product Backlog é dinâmico; ele muda constantemente para identificar o que o produto precisa ser apropriado, competitivo e útil. Se um produto existe, o Product Backlog também existe.
- O Product Backlog lista todos os recursos, funções, requisitos, melhorias e correções que constituem as alterações a serem feitas no produto em versões futuras. Os itens do Product Backlog têm os atributos de uma descrição, ordem, estimativa e valor. Os itens do Product Backlog geralmente incluem descrições de teste que comprovarão sua integridade quando "Concluído".

Esta informação não dispensa a leitura de: https://www.scrumguides.org/scrum-guide.html

80

80

PROCESSOS ÁGEIS
SCRUM

Sprint Backlog

Artefactos

- O Sprint Backlog é o conjunto de itens do Backlog do produto selecionados para o Sprint, além de um plano para fornecer o incremento do produto e atingir a meta do Sprint. O Sprint Backlog é uma previsão da equipa de desenvolvimento sobre qual funcionalidade será no próximo incremento e o trabalho necessário para entregar essa funcionalidade em um incremento "Concluído".
- O Sprint Backlog torna visível todo o trabalho que a Equipa de Desenvolvimento identifica como necessário para atingir a meta do Sprint. Para garantir a melhoria contínua, inclui pelo menos uma melhoria de processo de alta prioridade identificada na reunião Retrospetiva anterior.
- O Sprint Backlog é um plano com detalhes suficientes para que as mudanças em andamento possam ser entendidas no Daily Scrum. A Equipa de Desenvolvimento modifica o Sprint Backlog em todo o Sprint, e o Sprint Backlog surge durante o Sprint. Esse surgimento ocorre quando a equipa de desenvolvimento trabalha com o plano e aprende mais sobre o trabalho necessário para atingir a meta do Sprint.
- Conforme novo trabalho é necessário, a equipa de desenvolvimento o adiciona ao Sprint Backlog. À medida que o trabalho é executado ou concluído, o trabalho restante estimado é atualizado. Quando os elementos do plano são considerados desnecessários, eles são removidos. Somente a equipa de desenvolvimento pode alterar seu Sprint Backlog durante um Sprint. O Sprint Backlog é uma imagem altamente visível e em tempo real do trabalho que a Equipe de Desenvolvimento planeja realizar durante o Sprint, e pertence exclusivamente à equipa de desenvolvimento.

81

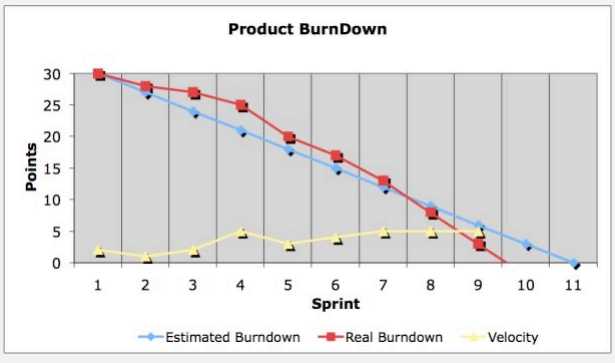
Esta informação não dispensa a leitura de: https://www.scrumguides.org/scrum-guide.html

81

PROCESSOS ÁGEIS
SCRUM

Burndown Chart

Artefactos

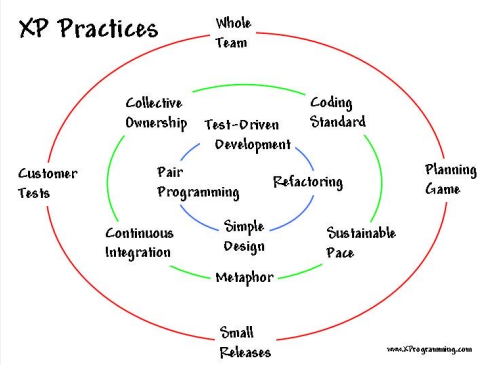


Fonte: https://www.scrum-institute.org/Burndown_Chart.php

82

PROCESSOS ÁGEIS
EXTREME PROGRAMMING (XP)

XP é uma metodologia de desenvolvimento de software baseada em valores de simplicidade, comunicação, feedback, coragem e respeito. Conduz a equipa ao uso de práticas simples, com feedback suficiente para permitir à equipa fazer os ajustes necessários ao bom desenrolar do desenvolvimento.



Fonte: https://ronjeffries.com/xprog/what-is-extreme-programming/#metaphor

83

PROCESSOS ÁGEIS
EXTREME PROGRAMMING (XP)

Valores/Princípios

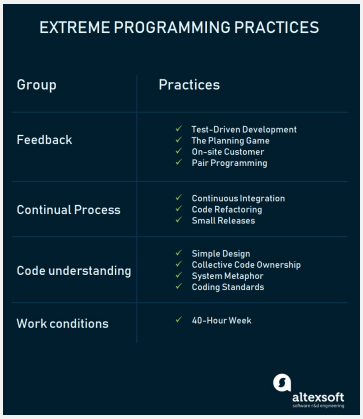
- **Comunicação**: todos numa equipa trabalham em conjunto em todas as etapas do projeto.
- **Simplicidade**: os desenvolvedores esforçam-dr para escrever código simples, agregando mais valor a um produto, pois economiza tempo e esforços.
- ***Feedback***: os membros da equipa entregam software com frequência, obtêm feedback sobre ele e aprimoram um produto de acordo com os novos requisitos.
- **Respeito**: todas as pessoas designadas para um projeto contribuem para um objetivo comum.
- **Coragem**: os programadores avaliam objetivamente os seus próprios resultados, sem desculpas e estão sempre prontos para responder às mudanças.

Fonte: https://www.altexsoft.com/blog/business/extreme-programming-values-principles-and-practices/

84

PROCESSOS ÁGEIS
EXTREME PROGRAMMING (XP)

Práticas/Ações



Fonte: https://dzone.com/articles/extreme-programming-values-principles-and-practice

85

PROCESSOS ÁGEIS
EXTREME PROGRAMMING (XP)

Desenvolvimento Orientado a Testes

A qualidade do software deriva de ciclos curtos de desenvolvimento que, por sua vez, permitem receber feedback frequente. E feedback valioso vem de bons testes. As equipas XP usam técnicas de desenvolvimento orientado a testes (ex. TTD).

Planning Game

Esta é uma reunião que ocorre no início de um ciclo de iteração. A equipa de desenvolvimento e o cliente reúnem para discutir e aprovar os recursos de um produto. No final do jogo de planeamento, os desenvolvedores planejam a próxima iteração e release, atribuindo tarefas para cada um deles.

On-site Custumer

O cliente final deve participar totalmente do desenvolvimento. O cliente deve estar presente o tempo todo para responder às perguntas da equipa, definir prioridades e resolver disputas, se necessário.

Pair Programming

Prática que requer que dois programadores trabalhem em conjunto no mesmo código. Enquanto o primeiro desenvolvedor se concentra em escrever, o outro analisa o código, sugere melhorias e corrige erros ao longo do caminho. Esse trabalho em equipa resulta em software de alta qualidade, partilha mais rápida do conhecimento, mas leva de 15 a 60% mais tempo. Nesse sentido, é mais razoável para projetos de longo prazo.

Fonte: https://dzone.com/articles/extreme-programming-values-principles-and-practice

86

PROCESSOS ÁGEIS
EXTREME PROGRAMMING (XP)

Code Refactoring

O objetivo dessa técnica é melhorar continuamente o código. Refatorar é remover a redundância, eliminar funções desnecessárias, aumentar a coerência do código e, ao mesmo tempo, desacoplar elementos. Mantenha seu código limpo e simples, para que você possa entendê-lo e modificá-lo facilmente quando necessário, seria o conselho de qualquer membro da equipa do XP.

Integração Contínua

Os desenvolvedores mantêm o sistema totalmente integrado. As equipas do XP levam o desenvolvimento iterativo para outro nível porque confirmam o código várias vezes ao dia, o que também é chamado de entrega contínua. Os programadores discutem quais partes do código podem ser reutilizadas ou compartilhadas. Dessa forma, sabem exatamente que funcionalidade precisam desenvolver. A política de código compartilhado ajuda a eliminar problemas de integração. Além disso, o teste automatizado permite que os desenvolvedores detetem e corrijam erros mais cedo, antes da implantação.

Pequenas *Releases*

Fornecer a primeira versão rapidamente e desenvolver ainda mais o produto, fazendo pequenas e incrementais atualizações. Pequenas versões permitem que os desenvolvedores recebam feedback com frequência, detetam bugs cedo e monitoram como o produto funciona na produção.

Fonte: https://dzone.com/articles/extreme-programming-values-principles-and-practice

87

PROCESSOS ÁGEIS
EXTREME PROGRAMMING (XP)

Design simples

O melhor design para software é o mais simples que funciona. Se alguma complexidade for encontrada, ela deverá ser removida. O design correto deve passar em todos os testes, não ter código duplicado e conter o menor número possível de métodos e classes.

Padrões de codificação

Uma equipa deve ter práticas comuns de codificação, usando os mesmos formatos e estilos para escrever código. A aplicação de padrões permite que todos os membros da equipa leiam, compartilhem e refatorem o código com facilidade, rastreie quem trabalhou em determinados trechos de código e torne o entendimento mais rápido para outros programadores. O código escrito de acordo com as mesmas regras incentiva a propriedade coletiva.

Propriedade do código coletivo

Essa prática declara a responsabilidade de toda uma equipa pelo design de um sistema. Cada membro da equipa pode rever e atualizar o código. Os desenvolvedores que têm acesso ao código sabem o lugar certo para adicionar um novo recurso. A prática ajuda a evitar a duplicação de código. A implementação da propriedade coletiva do código incentiva a equipa a cooperar mais e a vontade de trazer novas ideias.

Metáfora do Sistema

A metáfora do sistema representa um design simples que possui um conjunto de certas qualidades. Primeiro, um desenho e sua estrutura devem ser compreensíveis para novas pessoas. Eles devem poder começar a trabalhar nele sem gastar muito tempo examinando as especificações. Segundo, a nomeação de classes e métodos deve ser coerente. Recomendo que leia tambem: https://hygger.io/blog/system-metaphor-in-extreme-programming/

Fonte: https://dzone.com/articles/extreme-programming-values-principles-and-practice

88

PROCESSOS ÁGEIS
EXTREME PROGRAMMING (XP)

Semana de 40 horas

Os projetos XP exigem que os desenvolvedores trabalhem rápido, sejam eficientes e sustentem a qualidade do produto. Para aderir a esses requisitos, eles devem se sentir bem e descansados. Manter o equilíbrio entre vida profissional e pessoal evita que os profissionais se esgotem. No XP, o número ideal de horas de trabalho não deve exceder 45 horas por semana.

Fonte: https://dzone.com/articles/extreme-programming-values-principles-and-practice

89

PROCESSOS ÁGEIS
KANBAN

Início dos anos 40. O primeiro sistema Kanban foi desenvolvido por Taiichi Ohno para a Toyota, no Japão.

O Kanban é um método visual para gerir o trabalho conforme ele se move através de um processo. O Kanban permite visualizar o processo (o fluxo de trabalho) e o trabalho real passando por esse processo. O objetivo do Kanban é identificar possíveis constrangimentos no seu processo e corrigi-los para que o trabalho possa fluir através dele de maneira econômica, a uma velocidade ou taxa de transferência ideais.

Fonte: https://www.digite.com/kanban/what-is-kanban/

90

PROCESSOS ÁGEIS
KANBAN

Visualizar o fluxo de trabalho

A representação visual do processo de desenvolvimento ajuda a determinar o estado atual das tarefas.

Limitar *Work in Progress* (WIP)

Se restringir o número de tarefas ativas para todas as fases do projeto, pode controlar-se os recursos disponíveis e evitar ociosidade.

Medida contínua e melhoria do ciclo de vida

A possibilidade de fazer alterações durante o processo de trabalho é uma característica distintiva da metodologia Agile.

Fonte: https://xbsoftware.com/blog/software-development-life-cycle-sdlc-all-about-kanban/

91

PROCESSOS ÁGEIS
KANBAN

Fonte: https://www.digite.com/kanban/what-is-kanban/

92

PROCESSOS ÁGEIS
OUTROS...

Abordamos os processos ágeis mais representativos, no entanto existe Informação na internet sobre outros processos que pode ser consultada. Como trabalho complementar, sugere-se que o aluno façam uma pesquisa e análise sobre os mesmos...

93

QUALIDADE

- ***Quality Assurance* (QA):** atividades para medir e melhorar a qualidade de um produto e do processo
- ***Quality control* (QC):** atividades para **validar** e **verificar** a qualidade do produto através da **deteção** de falhas e "**correção**" dos defeitos
- Necessita de boas técnicas, processos, ferramentas e equipa

Ver mais em: Frank Tsui, Orlando Karam e barbara Bernal, Essentials of Software Engineering, 3rd Edition, Jones&Bartlett Learning, ISBN 978-1-4496-9199-8 (2014);

94

SIGNIFICADO DE "QUALIDADE"

- **Definições tradicionais**
  - ***Cumprir os requisitos***
  - ***Ajustado à utilização que vai ter***
- ***Verificação***: verificar se o software está em conformidade com os requisitos (se o software evoluiu dos requisitos corretamente)
- ***Validação:*** verificar se o software atende aos requisitos do utilizador(se está apto para uso)

Ver mais em: Frank Tsui, Orlando Karam e barbara Bernal, Essentials of Software Engineering, 3rd Edition, Jones&Bartlett Learning, ISBN 978-1-4496-9199-8 (2014);

95

VERIFICAÇÃO *vs* VALIDAÇÃO

- Verificação: "*Are we building the* ***product right***".
  - O software deve estar em conformidade com as especificações.
- Validação: "*Are we building the* ***right product***".
  - O software deve fazer o que o utilizador realmente pretende.

96

V & V

- O objetivo da V & V é estabelecer a confiança de que o sistema é adequado para a finalidade.
- Depende do objetivo do sistema, das expectativas do utilizador e do ambiente de marketing
  - Objetivo do Software
    - O nível de confiança depende de quão crítico o software é para uma organização.
  - Expectativas dos Utilizadores
    - Os utilizadores podem ter fracas expectativas de certos tipos de software.
  - Marketing
    - Obter um produto no mercado mais cedo pode ser mais importante do que encontrar defeitos no programa.

Fonte: Ian Sommerville, Software Engineering, 10 edição, Pearson Education Limited, ISBN: 10: 1-292-09613-6 (2016).

97

DESENVOLVIMENTO ORIENTADO A TESTES

- Escreva casos de teste unitários ANTES do código!
- Os casos de teste "são" / "tornam-se" requisitos
- Força o desenvolvimento em pequenos passos
- Passos:
  1. Escreva caso e código de teste
  2. Verificar (falha ou é executado)
  3. Modifique o código para que seja bem-sucedido
  4. Executar novamente caso de teste, testes anteriores
  5. *Refactoring* até ... ter sucesso

Ver mais em: Frank Tsui, Orlando Karam e barbara Bernal, Essentials of Software Engineering, 3rd Edition, Jones&Bartlett Learning, ISBN 978-1-4496-9199-8 (2014);

98

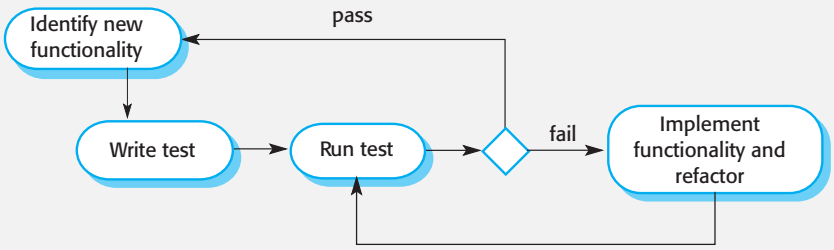
DESENVOLVIMENTO GUIADO POR TESTES

- O TDD (*Test-driven Development*) é uma abordagem para o desenvolvimento de sofware na qual se considera testes e desenvolvimento de código.
- Os testes são escritos antes do código e "passar" nos testes é fator crítico do desenvolvimento.
- Desenvolve-se o código incrementalmente, juntamente com um teste para esse incremento. Não se passa para o próximo incremento até que o código que se desenvolveu passe no teste.
- O TDD foi introduzido como parte de métodos ágeis, como **Extreme Programming**. No entanto, também pode ser usado noutros processos de desenvolvimento.

Fonte: Ian Sommerville, Software Engineering, 10 edição, Pearson Education Limited, ISBN: 10: 1-292-09613-6 (2016).

99

TEST-DRIVEN DEVELOPMENT



Fonte: Ian Sommerville, Software Engineering, 10 edição, Pearson Education Limited, ISBN: 10: 1-292-09613-6 (2016).

100

ATIVIDADES DO PROCESSO TDD

- Começa por ser identificado o incremento da funcionalidade necessária. Normalmente, isso deve ser pequeno e implementável em código.
- Escreve-se um teste para esta funcionalidade e implementa-se como um teste automatizado.
- Executa-se o teste, juntamente com todos os outros testes que foram implementados. Inicialmente, não se implementa a funcionalidade para que o novo teste falhe.
- Implementa-se a funcionalidade e executa-se novamente o teste.
- Depois de todos os testes terem sido executados com êxito, passa-se a implementar a próxima funcionalidade.

Fonte: Ian Sommerville, Software Engineering, 10 edição, Pearson Education Limited, ISBN: 10: 1-292-09613-6 (2016).

101

VER SEGUINTE INFORMAÇÃO...

IBM -> Test-driven development

https://www.ibm.com/garage/method/practices/code/practice_test_driven_development/

102

VANTAGENS DO TDD

- Cobertura de código
  - Cada segmento de código que se escreve possui pelo menos um teste associado; portanto, todo o código escrito tem pelo menos um teste.
- Teste de regressão
  - Um conjunto de testes de regressão é desenvolvido de forma incremental à medida que um programa é desenvolvido.
- Depuração simplificada
  - Quando um teste falha, deve ser óbvio onde está o problema. O código recém-escrito é verificado e modificado.
- Documentação do sistema
  - Os testes em si são uma forma de documentação que descreve o que o código deve fazer.

Fonte: Ian Sommerville, Software Engineering, 10 edição, Pearson Education Limited, ISBN: 10: 1-292-09613-6 (2016).

103

QUANDO PARAR DE TESTAR?

- **Resposta simples, parar quando**
  - Todos os casos de teste planeados terem sido executados
  - Todos os problemas encontrados terem sido corrigidos
- **Outras Técnicas:**
  - *Parar quando não se encontrar mais erros*
  - *Injetar defeitos - testar até que todos (ou% dos) erros encontrados*
- *NÃO deve ser por falta de tempo..*

Fonte: Frank Tsui, Orlando Karam e barbara Bernal, Essentials of Software Engineering, 3rd Edition, Jones&Bartlett Learning, ISBN 978-1-4496-9199-8 (2014);

104