Aula 3 - Metodologias de desenvolvimento de software

Jayme Anchante

15 de junho de 2020

Metodologias de desenvolvimento de software

Lista de methodologies

- Rapid Application Development (RAD), 1991
- Unified Process (UP), 1994
- Dynamic Systems Development Method (DSDM), 1994
- Scrum, 1995
- Crystal Clear, 1996
- Extreme Programming (XP), 1996
- Feature-driven development, 1997

ASD DevOps DAD DSDM FDD IID Kanban Lean SD LeSS MDD
MSF PSP RAD RUP SAFe Scrum SEMAT TSP OpenUP UP XP

Rapid Application Development

Conceito

Também conhecido como Rapid Application Building (RAB) é tanto um termo geral utilizado para se referir a abordagens adaptativas de desenvolvimento de software assim como um abordagem própria.

Abordagens RAD põem menos ênfase no planejamento e mais ênfase no processo adaptativo

Protótipos são utilizados em adição ou no lugar do design do software

Motivação

RAD foi uma resposta ao paradigma Waterfall desenvolvido nos anos 1970 e 1980

Waterfall foi criado a partir da engenharia tradicional para indústria e construção civil

Software é diferente, pois o conhecimento ganho com o processo de desenvolvimento pode retornar para os requisitos e o design da solução

Abordagens baseadas no planejamento tentam fixar os requisitos, a solução e o plano para implementá-la, tudo isso desencorajando mudanças. Abordagens RAD reconhecem que software é um processo intensivo em conhecimento e provêm maneiras de aproveitar este conhecimento adquirido durante o projeto para adaptar e melhorar a solução

História

A primeira abordagem RAD foi desenvolvido por Barry Boehm e é conhecida como Modelo Espiral

Ênfase no desenvolvimento de protótipos em vez da forma tradicional de especificações

James Martin desenvolveu a abordagem RAD na IBM nos anos 1980 e publicou o livro *Rapid Application Development* em 1991

RAD de Martin

1. Fase de planejamento dos requisitos: usuários, gerentes, desenvolvedores e outros stakeholders discustem as necessidades de negócio, escopo do projeto, restrições e requisitos de sistema; no fim a equipe ratifica as questões principais e obtém autorização da gestão sênior para prosseguir
2. Fase de design do usuário: usuários interagem com desenvolvedores para criar modelos e protótipos que representam todos os sistemas, entradas e saídas usando wireframes, representações, mockups
3. Fase de construção: desenvolvimento do software em si, usuários ainda participam e fazem sugestões no processo
4. Fase de transição: testes, implantação, treinamento do usuário

RAD - imagem

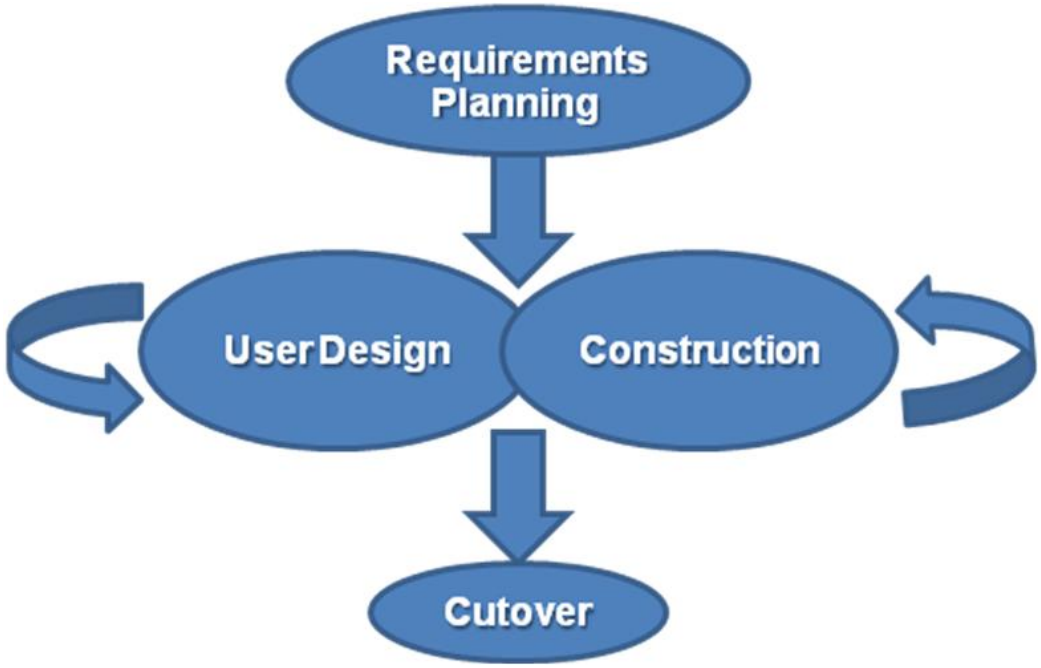

Figure 1: RAD

Vantagens

- Melhor qualidade: interação dos usuários com protótipos gera mais usabilidade, foco nos problemas de negócio
- Controle de risco: identificação prematura de riscos e ajustes baseados em experiências empíricas
- Projetos finalizados no cronograma e no orçamento: desenvolvimento de unidades incrementais reduz o risco de falhas catastróficas, com espaço para agir cedo no projeto

Desvantagens

- Risco de uma nova abordagem: no início sua adoção foi lenta e vista com suspeita
- Falta de ênfase em requisitos não funcionais: aqueles não vistos pelo usuário final
- Tempo dos recursos: mais interação entre usuários e desenvolvedores significa investir tempo destas pessoas no processo (paradoxo: quanto melhor a expertise de domínio de negócios, mais ele é obrigado a gerenciar o negócio e menos propenso ele vai estar em supervisionar o projeto)
- Custo de oportunidade entre flexibilidade e controle

Processo Unificado

Conceito

Processo Unificado é uma metodologia iterativa e incremental de desenvolvimento de software

Alguns dos refinamentos mais conhecidos são: Processo Unificado Racional (patenteado pela IBM), Processo Unificado Aberto e o Processo Unificado Ágil

É um processo extensível e customizável, utilizado de forma genérica para se referenciar a todos os refinamentos para não infringir patentes

Características

- Iterativo e incremental
- Centrado na arquitetura
- Foco no risco

Iterativo e incremental

As fases de Elaboração, Construção e Transição são divididas em uma série de iterações com limite de tempo

Cada iteração gera um incremento: lançamento de uma funcionalidade nova ou melhoria de uma antiga

Todas as iterações incluem trabalho em todos os processos (e.g. requisitos, design, planejamento e testes), mas sua importância vai mudando ao longo do projeto

Iterativo e incremental - imagem

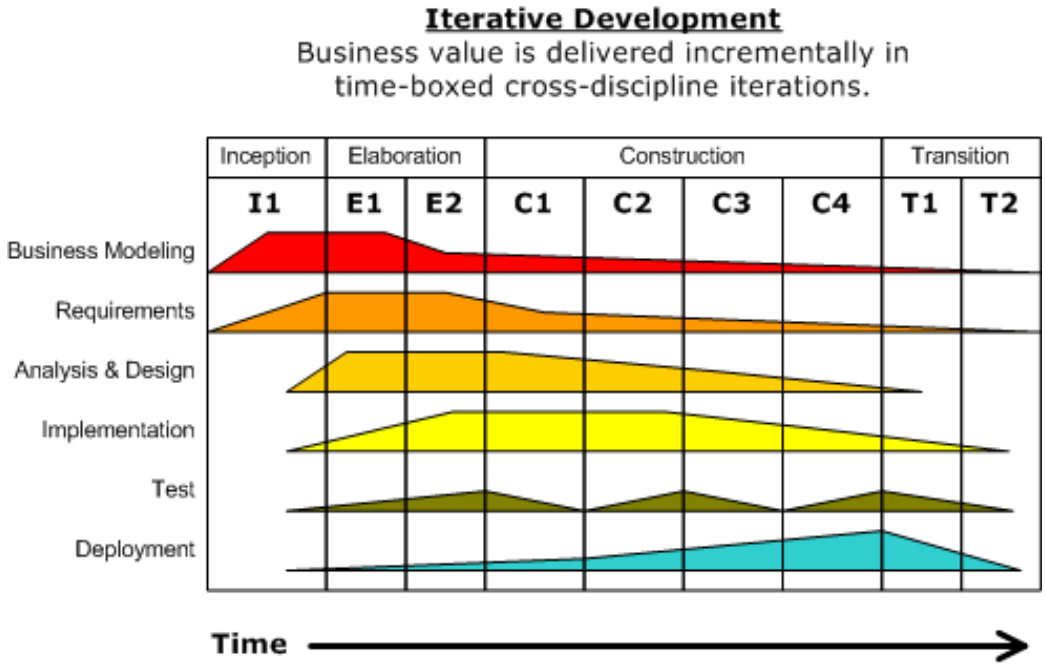

Figure 2: Iterações

Centrado na arquitetura

Suporte a múltiplas modelos e visões de arquitetura

Um dos mais importantes entregáveis é uma arquitetura mínima executável

Foco no risco

Necessidade que a equipe foque nos riscos mais críticos nas primeiras iterações do projeto

Os entregáveis de cada iteração - especialmente na fase de elaboração - devem ser selecionados para que os maiores riscos sejam mitigados primeiro

Fases do Processo Unificado

- Inception
- Elaboração (milestone)
- Construção (release)
- Transição (lançamento final)

Inception

Deve ser a mais curta das fases (uma longa duração é sinal de uma sobreespecificação, o que é contrário ao espírito da metodologia)

Objetivos:

- Estabelecimento
- Cronograma e estimativa de custos preliminar
- Viabilidade
- Comprar ou desenvolver?

Inception - finalização

Stakeholders devem verificar o estado do projeto e verificar o Lifecycle Objectives (LCO) Milestone:

- Escopo do projeto
- Requisitos iniciais
- Cronograma e orçamento
- Aceite de riscos
- Aceite de processo
- Visão de negócios
- Planejamento para a próxima fase
- Compliance com portfólio

Elaboração

Capturar a maioria dos requisitos, analisar riscos, validar a arquitetura do sistema

Criação de casos de uso

Entregável é um plano que inclue cronograma e orçamento para a fase de Construção

Construção

Maior porção do projeto

Características são implementadas numa série de curtas iterações em que cada uma resulta num lançamento do software

A última iteração da construção entregará o software pronto para a implantação na fase de Transição

Transição

Implantação para os usuários finais

Feedback dos usuários a partir do lançamento inicial resulta em refinamentos incorporados ao longo de diversas iterações

Refinamentos

Cada refinamento traz uma categorização diferente das disciplinas e dos fluxos de trabalho, colocando ênfase em diferentes artefatos do projeto, diferenças em o que acontece após a fase de transição

Agile Unified Process

Versão simplificada do Processo Unificado Racional desenvolvida por Scott Ambler

Inclui as técnicas ágeis de desenvolvimento orientado a testes, modelagem ágil, mudança de gerenciamento ágil, refatoração de banco de dados

AUP - disciplinas

1. Modelo: entender a organização, o problema, uma solução viável
2. Implementação: transforma o modelo em código e testes unitários
3. Teste: testes de integração e de qualidade
4. Implantação: planeja a implantação e como será disponível para os usuários finais
5. Gerenciamento de configuração: como acessar, controlar os artefatos do projeto
6. Gerenciamento de projeto: gerenciamento de riscos, de pessoas e coordenação fora do projeto
7. Ambiente: garantir que os guidelines são apropriados, as ferramentas necessário estão disponíveis ao time

AUP - imagem

Dois tipos de iterações: desenvolvimento e lançamento

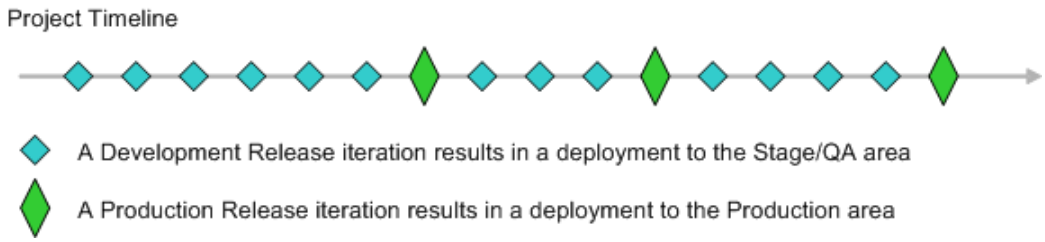

Figure 3: Iterações

Enterprise Unified Process

Desenvolvido por Ambler e Constantine em 2000

Além das 4 fases do Processo Unificado, existem Produção, e a Aposentadoria

Além das disciplinas de projeto do Processo Unificado Racional ainda colocar mais Operações e Suporte e mais sete disciplinas empresariais: Modelagem de Negócios Empresariais, Gerenciamento de Portfólio, Arquitetura Empresarial, Reutilização Estratégica, Gerenciamente de Pessoas, Administração Empresarial, Melhora em Processos de Software

Processo Unificado Aberto

Características essenciais da Processo Unificado (desenvolvimento iterativo, casos de uso, desenvolvimento orientado a cenários, gerenciamento de riscos, foco na arquitetura) sem as partes opcionais

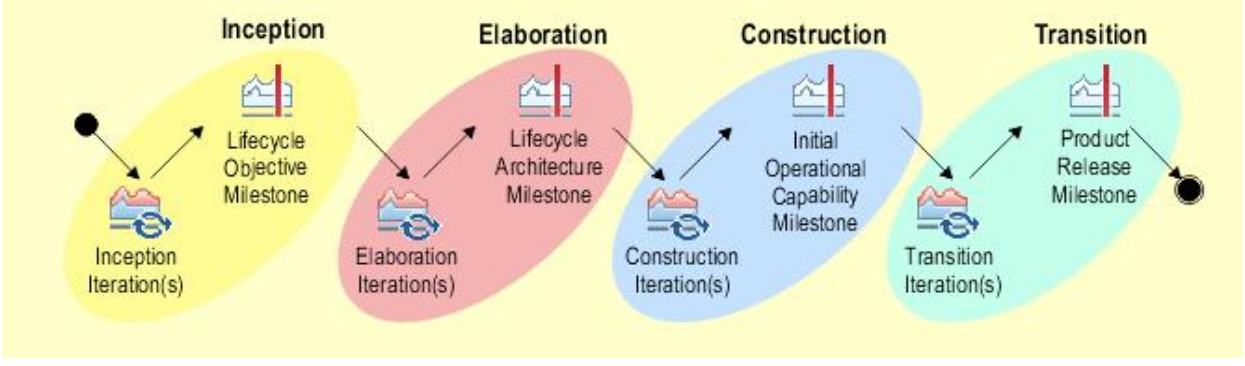

Figure 4: Iterações

Processo Unificado Racional

Metodologia adaptável criada pela empresa Rational Software, hoje divisão da IBM

A cada iteração, as tarefas são categorizadas em nove disciplinas: modelagem de negócios, requisitos, design, implementação, test e implantação (disciplinas de engenharia), configuração e mudança de gerenciamento, gerenciamento de projeto, ambiente (disciplinas de suporte)

Mesmas fases do Processo Unificado

Certificação IBM de Designer de Solução de Processos - RUP

Dynamic Systems Development Method (DSDM)

Conceito

Inicialmente uma metodologia específica que tentava disciplinar a abordagem flexível RAD.

Hoje é mais uma abordagem genérica de gerenciamento de projetos

Criado em 1994

Princípios

- Foco nas necessidades de negócio
- Entregar no prazo
- Colaboração
- Nunca comprometer a qualidade
- Construção incremental de bases robustas
- Desenvolvimento iterativo
- Comunicação contínua e clara
- Controle

Técnicas

- Timeboxing: quebra do projeto em porções menores, cada uma com orçamento fixo e data de entrega, cada porção tem seus requisitos priorizados e selecionados já que tempo e dinheiro são fixos
- MoSCoW: MUST have, SHOULD have, COULD have, WON'T have, metodologia de priorização
- Prototipagem
- Testes em cada iteração
- Workshop: reúne stakeholders para discutir requisitos, funcionalidades
- Modelagem: representação visual de certos aspectos do sistema
- Gerenciamento de configuração: com muitos entregáveis em cada iteração, precisa-se de boas práticas de gerenciamento

Papéis

- Patrocinador executivo: também conhecido como “Project Champion”, tem o poder e a responsabilidade de angariar recursos e orçamento, também toma decisões finais
- Visionário: levantar os requisitos inciais, tem a melhor visão dos objetivos de negócios dos sistemas e do projeto
- Embaixador dos usuários: representante dos usuários finais no projeto, dar feedback durante o desenvolvimento
- Usuário conselheiro: dá um ponto de vista importante dos usuários, traz conhecimento diário ao projeto
- Gerente de projeto
- Coordenador técnico: responsável pelo design da arquitetura e pela qualidade técnica do projeto
- Líder da equipe: garante que o time trabalha como um só
- Desenvolvedor de soluções: interpreta os requisitos de sistema e os modela desenvolvendo códigos e protótipos
- Tester de soluções: checagem técnica, faz comentários e documentação do código
- Escriba: reúne e documenta requisitos, acordos, decisões feitas

Fatores críticos de sucesso

- Aceitação do DSDM
- Comprometimento de envolvimento do usuário final
- Equipe qualificada
- Relacionamento solidário entre cliente e fornecedor

Scrum

Conceito

Metodologia ágil para desenvolvimento, entrega e manutenção de software

Normalmente para equipes pequenas (10 pessoas ou menos)

Quebra o trabalho em objetivos que devem ser completados em iterações finitas

Nome

Scrum é um termo que origina-se de um tipo de formação no rugby

Não existe uma patente nem um dono do Scrum

Figure 5: Origem do scrum

Ideias chave

- Autorganização da equipe
- Equipe reunida fisicamente
- Comunicação face a face
- Volatilidade dos requisitos
- Desafios imprevisíveis
- Aceita que o problema não pode ser definida antes de tudo, o foco deve ser a maximização da capacidade de entrega da equipe

Papéis

- Product owner
- Scrum master
- Time de desenvovimento

Product owner

Representa os stakeholders e é a voz dos clientes/usuários. É responsável pelo backlog do produto e maximizar o valor entregue pela equipe

Define o produto em termos do usuário (histórias do usuário), adiciona-os ao backlog e faz sua priorização. Uma equipe scrum deve ter apenas um PO, mas um PO pode ter várias equipes, não pode ser o scrum master

Foco na parte de negócios do desenvolvimento, sendo a ligação entre stakeholders e equipe. Não deve ditar a parte técnica da solução

Comunicação, empatia são habilidades essenciais

PO frente aos stakeholders

- Definir e anunciar lançamentos
- Comunicar entregas e status do equipe
- Compartilhar o progresso nas reuniões
- Compartilhar riscos, impedimentos, dependências com os stakeholders
- Negociar prioridades, escopo, orçamento e cronograma
- Garantir que o backlog é visível e claro a todos

Scrum master

Remoção de impedimentos para garantir que a equipe mantenha sua capacidade de entrega

Não é um gerente ou um líder de equipe, mas é um meio de campo entre a equipe e distrações e

Garante que o scrum seja seguido, é o facilitador das cerimônias e motiva a equipe a se aprimorar

Uma distinção entre o scrum master e o gerente de projetos tradicional são as responsabilidades de gerenciamento de pessoas (que o primeiro não tem)

Scrum master - responsabilidades

- Ajudar o PO a manter o backlog eficiente
- Ajuda a construir a “definição de pronto”
- É um coach da equipe
- Promove a autorganização
- Remoção de impedimentos
- Facilita eventos e cerimônias
- Educa os stakeholders quanto aos princípios do scrum

Equipe de desenvolvimento

Deve ter entre 3 e 9 membros e é responsável por construir incrementos de valor ao final de cada iteração/sprint

Normalmente desenvolvedores, mas pode incluir pesquisadores, arquitetos, especialistas em dados, testers, engenheiros

Não podem receber tarefas de ninguém a não ser do PO, e têm o scrum master como barreira de distrações

Encorajados a interagir com usuários, clientes e stakeholders para ganhar insights

Fluxo de trabalho

- Sprint
- Planejamento da sprint
- Daily
- Revisão da sprint
- Retrospectiva da sprint
- Refinamento do backlog

Sprint

Também conhecida como iteração ou timebox é a unidade básica de desenvolvimento

A duração é fixa e pré acordada a priori, entre uma e quatro semanas, normalmente duas

Começa com um planejamento e acaba com a revisão e a retrospectiva

Planejamento da sprint

- Discussão e acordo do trabalho a ser feito na sprint
- Seleção dos itens do backlog que entrarão na sprint
- Acordo do objetivo da sprint, uma descrição do entregável futuro
- Duração recomendada de quatro horas para uma sprint de duas semanas

Na primeira metade toda a equipe scrum seleciona os itens do backlog que todos acreditam que possam ser entregues. Na segunda metade a equipe de desenvolvimento detalha o trabalho em tarefas. Após o detalhamento, os itens podem ser quebrados em itens menores ou colocados de volta no backlog. Votação de quais tarefas a equipe irá entregar dentro do backlog da sprint

Daily

Todos na equipe de desenvolvimento participam, normalmente de pé

Começa sempre no horário mesmo que faltem alguns membros, deve acontecer no mesmo horário e local todos os dias e é limitada a quinze minutos

Todos são bem vindos, mas apenas o time de desenvolvimento pode falar

Tipicamente responde-se a três perguntas: o que fiz ontem para completar o objetivo da sprint; o que farei hoje para completar o objetivo da sprint; há impedimentos em bloqueiem a mim ou a equipe de atingir o objetivo da sprint

Daily - continuação

Todo impedimento deve ser anotado pelo scrum master e delegado a pessoa responsável

É possível que o scrum master mostre o status de entrega do time atráves de um gráfico de burndown

Se a equipe não vê valor nessas cerimônias, é papel do scrum master de mostrar valor e evangelizar as pessoas quanto ao scrum

Discussões detalhadas são proibidas, assim que a reunião acabar membros individuais podem se reunir para discutir problemas em detalhe, encontros conhecidos como sessão de desobstrução/impedimento ou after party

Revisão da sprint

Feita ao final da sprint, deve-se:

- Revisar o trabalho completado e o trabalho planejado não completado
- Apresentação para stakeholders
- Colaboração com stakeholders sobre próximos passos

Duração recomendada de duas horas para uma sprint de duas semanas

Retrospectiva da sprint

Reflexão da equipe sobre:

- Aprendizados na última equipe
- Identifica e concordância quanto a processos de melhoria

As três principais perguntas que deve-se responder são: o que aconteceu de bom na sprint, o que aconteceu de não tão bom, o que pode ser melhorado para a próxima sprint

Recomendação de duração de uma hora e meia para uma sprint de duas semana, tendo scrum master como facilitador

Refinamento de backlog

Antigamente conhecido como *grooming* é um processo contínuo de revisão do backlog, para manter os itens, claros, esmiuçados, com critérios de aceitação, identificação de dependências

Não é um princípio original do scrum, mas foi introduzido posteriormente como forma de gerenciamento do backlog

Artefatos

- Backlog do produto
- Backlog da sprint
- Incremento
- Extensão

Backlog do produto

Quebra do trabalho a ser feito e contém uma lista ordenada de requisitos do produto. Formatos comuns são histórios do usuários e casos de uso. Os requisitos definem as features, bug fixes e outros requisitos para garantir um produto viável. O PO faz a priorização de acordo com risco, valor, dependências, tamanho e cronograma

Backlog do produto é aquilo que será entregue ordenado na sequencia de entrega, deve ser visível a todos

Os itens devem conter a percepção de valor de negócio do PO e o esforço da equipe de desenvolvimento em pontos da história. Essas estimativas ajudam o PO a ver o ROI (return over investment) e a priorização dos itens

Backlog do produto - gerenciamento

Na sua forma mais simples, é uma lista de itens. Deve-se haver regras claras para como adicionar, remover e ordenar os itens

Backlog da sprint

Lista de itens que a equipe de desenvolvimento deve entregar na próxima sprint

A lista é selecionado do backlog do produto progressivamente do topo da lista até que a equipe acredite não conseguirá entregar mais em uma única sprint

A equipe deve ter em mente sua capacidade passada tendo em mente quanto esforça conseguirá ter

Itens são quebrados em tarefas menores, tarefas nunca são atribuídas por alguém que não seja a própria pessoa

Uma vez que o backlog da sprint é pactuado, nada pode ser adicionado a não ser pelo próprio time de desenvolvimento

Incremento

Produção que pode ser um potencial lançamento para atingir o objetivo da sprint

É formado por todos os itens completados na sprint atual e passadas

O incremento deve seguir a “definição de pronto”

Outros artefatos

- Gráficos de burndown
- Definição de pronto
- Definição de entregue
- Velocidade

Gráfico de burndown

Representação gráfica de quanto trabalho falta em relação ao tempo

Se a linha de tarefas a serem entregues está acima do ideal, a sprint está atrás do planejado

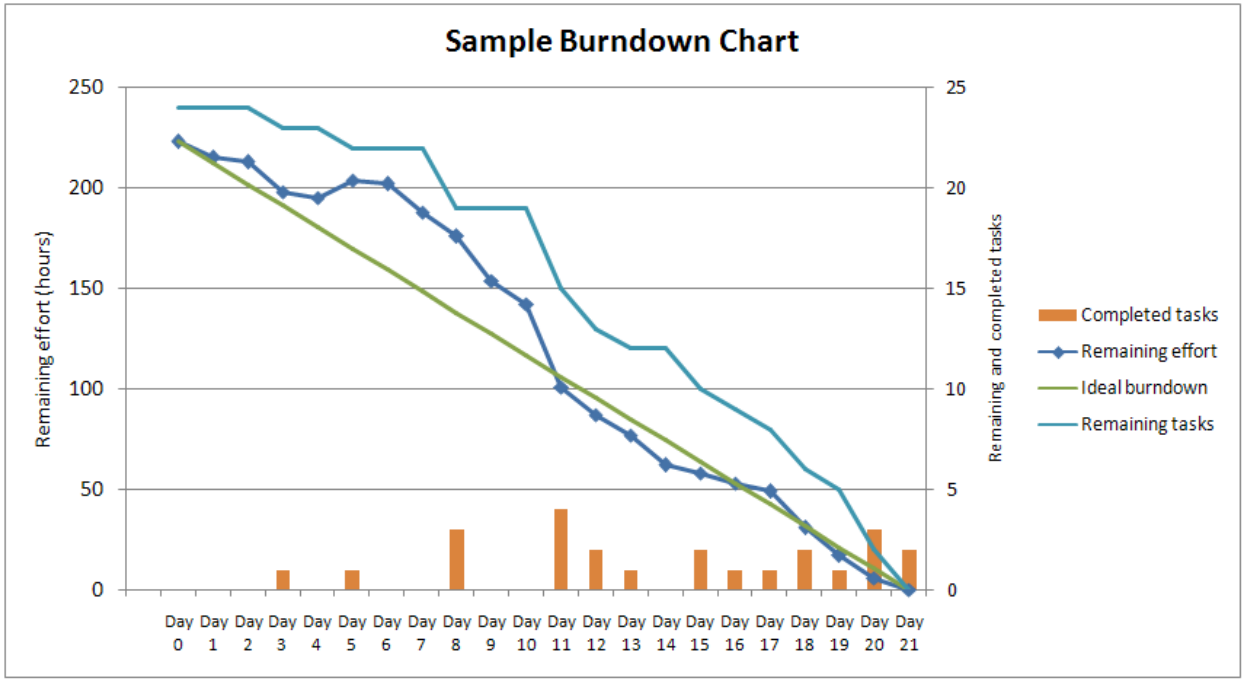

Figure 6: Gráfico de burndown

Definição de pronto

Critério para determinar se uma especificação é suficiente para começar a se trabalhar neste item

Definição de entregue

Critério para determinar se um item foi entregue pelo time de desenvolvimento, varia de time para time, mas deve ser consistente dentro do mesmo time

Velocidade

Capacidade total de esforço da equipe numa sprint

Spike

Período finito utilizada para pesquisar um conceito ou criar um protótipo

Usualmente entre sprints

Casos de uso

Lista de ações ou eventos de interação entre atores (em UML) e o software

História do usuário

Descrição informal em linguagem não técnica de uma ou mais características de um sistema de software

Escritas da perspectiva do usuário final

Eu como *cargo* posso *capacidade*, assim *recebe benefício*

Poker de planejamento

Esforço de resolução de itens do backlog é expresso em pontos da história seguindo a escala Fibonacci

Os pontos da tarefa podem ser definidos com acordo geral gamificada usando o poker de planejamento

Após a explicação da histório pelo PO, cada membro do time joga uma carta virada para baixo que significa a pontuação necessária para completar a tarefa. Após todos jogarem, as cartas são reveladas e as estimativas são discutidas, normalmente por aqueles que jogaram as cartas mais baixas e mais altas. Repete-se o processo até que um consenso seja alcançado

Ao jogar as cartas pra baixo, evita-se o viés cognitivo da ancoragem

Limitações

- Times estão dispersos geograficamente e temporalmente: scrum envolve interações face a face ou síncronas
- Times com pessoas altamente especializadas: desenvolvedores devem ter "habilidades em T" (barra vertical é a profundidade de sua expertise, barra horizontal é a habilidade de colaboração entre disciplinas)
- Produtos com muitas dependências externas: testes de aceitação de usuários ou coordenação com outras equipes pode causar atrasos e falhas nas sprints
- Produtos maduros ou com controles de qualidade específicos: produtos que necessitam de testes extensivos (e.g. aparelhos médicos, segurança de veículos)

Valores

Abordagem baseada no feedback, sustentado por três princípios de transparência, inspeção e adaptação

Cinco valores do scrum:

1. Comprometimento
2. Coragem
3. Foco
4. Abertura
5. Respeito

Adaptações

- Scrumban: uso concomitante de scrum e kanban
- Scrum de scrums: técnica para operar scrum em escala para muitas equipes trabalhando no mesmo produto
- Scrum em larga escala: dois níveis da abordagem, para equipes até oito pessoas, e para equipes com mais de oito até centenas de pessoas

Críticas

- Cerimônias do scrum podem prejudicar a produtividade das equipes e fazê-las perder tempo
- Podem vir a se tornar uma forma de microgerenciamento
- Presume que a quantidade de esforço necessário para completar uma tarefa pode ser quantificada, quando muitas vezes isso é imprevisível

Programação extrema

Conceito

Conhecido como XP (eXtreme Programming), é uma metodologia para melhorar a qualidade de software e a capacidade de resposta

Usa das melhores práticas e as leva a “níveis extremos”

História

Kent Beck começou o desenvolvimento do XP durante seu trabalho no projeto de folha de pagamento Comprehensive Compensation System (C3) na Chrysler, especialmente quando se tornou líder de projeto em 1996. Após um refinamento do método, publicou o livro "Programação extrema explicada" em 1999. O projeto C3 foi cancelado em 2000 quando a Daimler-Benz adquiriu a Chrysler

Atividades do desenvolvimento de software

1. Codificação: é o produto mais importante do desenvolvimento
2. Testes: se alguns testes eliminam algumas falhas, muitos testes eliminam muitas falhas. Testes em nível de sistema eram encorajados inicialmente ao final de todos os dias, mas hoje foram reduzidos para semanalmente
3. Escutar: ouvir o que os clientes dizem que o sistema deve fazer
4. Design: com sistemas complexos, é necessário pensar as dependências dentro do sistema, organizar a lógica dos módulos

Valores

1. Comunicação: dos requisitos para a equipe de desenvolvimento, documentação. Disseminação de conhecimento entre os membros da equipe
2. Simplicidade: começar com a solução mais simples, funcionalidades extras podem ser adicionadas mais tarde. Abordagem "You are not gonna need it" (YAGNI)
3. Feedback: do sistema (testes unitários), do cliente (testes de aceitação), da equipe (quando clientes vêm com novos requisitos e a equipa dá uma estimativa)
4. Coragem: sempre fazer design e código para hoje e não para amanhã. De estar confortável com refatoração. Para excluir código que é obsoleto
5. Respeito: pelos outros e consigo mesmo. Nunca colocar mudança que quebrem o sistema de alguma forma (compilação, falhar testes, atrasar o trabalho dos outros)

Regras

De código:

- Codar testes unitários antes
- Apenas um par integra o código de cada vez
- Deixar otimizações por último
- Sem hora extra

De testes:

- Todo código deve ter testes unitários
- Todo código deve passar os testes unitários antes de ser lançado
- Quando um bug for encontrado, um teste deve ser escrito antes do bug ser resolvido
- Testes de aceitação são feitos frequentemente e seus resultado são publicados

Princípios

- Feedback:
- Simplicidade
- Abraçar as mudanças

Práticas

12 práticas agrupadas em 4 áreas: feedback em escala granular, processo contínuo, entendimento comum e bem estar do programador

Feedback em escala granular

- Programação em par
- Jogo do planejamento
- Desenvolvimento orientado a testes
- Toda equipe

Programação em par

Duas pessoas trabalham no mesmo código/máquina, uma programa coda e outra revisa tudo, os papéis se invertem frequentemente. Economicamente aumenta o tempo de escrita entre 15 e 100%, mas reduz os bugs em 15%. É importante para o fluxo de conhecimento

Pode não performar se houver desengajamento de uma das partes ou se acontecer o "olhe o mestre" caso haja muita diferença de senioridade e a pessoa júnior ser o observador

Jogo do planejamento

É uma reunião que ocorre uma vez a cada iteração dividido em:

- Planejamento do lançamento: fase da exploração (requisitos do cliente na forma de histórios do usuário), fase do comprometimento (funcionalidade que serão incluídas e a data de entrega), fase da condução (o plano pode ser ajustado, requisitos adicionados/removidos)
- Planejamento da iteração: fase da exploração (requisitos transformados em tarefas), fase do comprometimento (tarefas são atribuídas aos programadores), fase da condução (tarefas são feitas)

Desenvolvimento orientado a testes

Testes são escritos antes do código, estimulando o programador a pensar quando seu código pode falhar

Toda equipe

O cliente não é aquele que paga as contas, mas sim aquele que usa o sistema. O cliente deve estar sempre próximo para responder respostas, interagir com o sistema, dar feedback

Processo contínuo

- Integração contínua
- Refatoração ou melhora de design
- Pequenos lançamentos

Integração contínua

Prática de fundir/mesclar todo o trabalho dos desenvolvedores numa mesma versão principal uma ou mais vezes por dia

Além de rodar testes localmente, é prática comum o servidor de versionamento rodar os testar para validar cada mudança de código

Melhora de design

Como XP defini que deve-se codar para hoje e implementar apenas o necessário, às vezes é necessário refatoração de código para acomodar mudanças de arquitetura

Entendimento comum

- Padrões de código
- Propriedade comum do código
- Design simples
- Metáfora do sistema

Metáfora do sistema

É uma história que todos (clientes, programadores, gerentes) podem contar sobre como o sistema funciona. São nomenclaturas para classes e métodos para que todos saibam para que servem

Bem estar do programador

- Ritmo sustentável

Se for necessário hora extra numa semana, é importante não haver na seguinte

Pessoas performam melhor e mais criativamente quando estão descansadas

Grandes facilitadores são fusões de código frequentes, cobertura de testes, lançamentos frequentes também reduzem problemas inesperados em produção e as horas extras para consertá-los

Controvérsias

Proponentes do XP dizem que ter um cliente no local pedindo mudanças informalmente o processo se torna mais flexível poupando dinheiro, já os críticos dizem que isso faz com que haja muito retrabalho e o escopo aumenta indefinidamente além do orçamento

- Requisitos são expressos como testes automatizados de aceitação e não como documentos formais
- Requisitos são definidos incrementalmente em vez de tentar descrever todos de uma só vez
- Desenvolvedores devem trabalhar em duplas
- Não existe um grande design inicial, tudo acontece durante o processo e incrementalmente
- Um representante do cliente pode se tornar um ponto de falha no projeto, há o perigo de microgerenciamento

Desenvolvimento orientado a características

Conceito

Uma mistura de práticas reconhecidas derivadas de funcionalidades valorizadas pelo cliente

História

Inicialmente desenvolvido por Jeff de Luca para atender um projeto de 15 meses com 50 pessoas para um banco de Singapura em 1997

Resultando em cinco processos

Etapas

1. Desenvolver um modelo geral: projeto começa com um passo a passo do escopo e seu contexto
2. Criar uma lista de características: definidas como "*ação resultado objeto*", e.g. "calcular o total de uma venda"
3. Planejamento de cada característica: produzir um plano de desenvolvimento e atribuir aos programadores
4. Design por característica: seleção das características que serão entregues nas próximas duas semanas
5. Construção por característica

Lean

Conceito

Tradução para software do “produção enxuta” industrial originalmente no modelo Toyota

Termo se originou no livro “Lean software development” de Mary e Tom Poppendieck publicado em 2003

Princípios

1. Eliminar resíduos: tudo que não entrega valor ao cliente (trabalho parcial, características extras, esperas, bugs)
2. Ampliar aprendizados: testar ideias com código, mostrar telas para os clientes para validar requisitos
3. Decidir o mais tarde possível: com muita incerteza, o sistema deve ser construído com capacidade para mudanças quando os fatos se tornarem mais certos
4. Entregar o mais rápido possível: para receber feedbacks, ganhar da concorrência
5. Empoderar o time: gerentes são incentivados a escutar os desenvolvedores para entender a situaçãoi
6. Construir integridade: com mais características entregues, aumenta a complexidade e fica mais difícil adicionar melhoras, refatorações são importantes para manter o código simples
7. Otimizar o todo: software não é a soma das partes, mas também o produto de suas interações

Kanban

Conceito

É um método para gerenciar e melhorar o trabalho entre sistemas e pessoas

Balanceamento entre a capacidade disponível e as demandas

História

David Anderson publicou o livro Kanban em 2010 descrevendo um projeto na Microsoft em 2004

Quadro

Seu design varia: itens de trabalho podem ser características ou histórias do usuários ou tarefas, colunas podem ser atividades do fluxo de trabalho (código, teste) ou status (fazendo, pronto)

Livro original define duas práticas: visualizar seu trabalho e limitar o trabalho em andamento

O livro Essential Kanban Condensed acrescenta: fazer políticas explícitas, controle do fluxo, ter feedbacks, melhorar a colaboração

Quadro - imagem

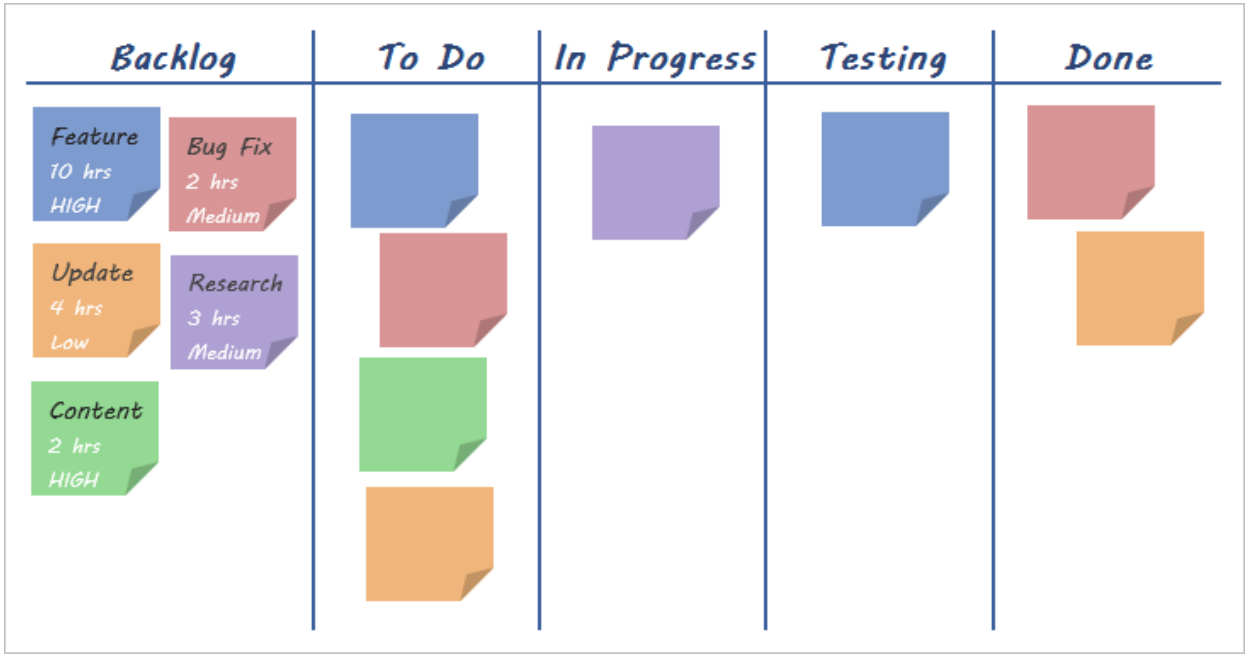

Figure 7: Quadro de Kanban

Disciplined Agile Delivery

Conceito

Construção a partir de diversas práticas ágeis como scrum, lean, kanban e outros

As metodologias do ágil focam em uma parte das atividades necessárias para entregar o proeto, em vez de cada empresa criar o seu ágil juntando as partes, o DAD oferece uma proposta metodologia completa que une as partes e preenche os vazios

História

A primeira iteração o DAD foi criada em 2015, desde então novas iterações foram feitas para revisar e adicionar metodologias e tecnologias, como DevOps

O DAD tem como referência o livro "Choose You WoW! A Disciplined Agile Delivery Handbook for Optimizing Your Way of Working" de Scott Ambler e Mark Lines

Em agosto de 2019, o Project Management Institute comprou o DAD

Princípios chave

- Pessoas primeiro: pessoas e suas interações são o fator determinante de sucesso
- Hibridização: adoção de diversas práticas ágeis
- Ciclo completo de entrega: não foca apenas na construção
- Completo: mostra como desenvolver, modelar, criar arquitetura, gerenciamento, requisitos, documentação, governança
- Contextual: abordagem focada no resultado e não uma prescrição de práticas
- Soluções consumíveis em detrimento de software funcionando: software ainda é um entregável importante, mas o foco deve ser mais holístico, como hardware, processos de negócio, estrutura organizacional
- Auto organização com governança

Ciclo

Apoia disciplinas ágeis baseadas no scrum e no lean baseadas no kanban

1. Agile: três fases sendo inception (sprint 0), construção e transição (sprint de entrega)
2. Lean: três fases baseadas no kanban
3. Entrega contínua ágil: fluxo contínuo de trabalho incremental com entregáveis frequentes
4. Entrega contínua lean: fluxo contínuo de trabalho lean
5. Exploração: fase experimental baseada no conceito de *lean startup* e *minimum viable product*
6. Programação: ciclo para coordenação da equipe

Objetivos

Existem 21 objetivos divididos em 4 fases: inception, construção, transição e objetivos continuados

Inception

- Formar a equipe
- Alinhar com objetivos de negócio
- Visão comum de projeto
- Explorar escopo
- Estratégia de arquitetura
- Planejar o lançamento
- Estratégia de teste
- Visão comum geral
- Garantir o financiamento

Construçao

Construir uma solução incremental

- Aprovar a arquitetura
- Acomodar mudanças de necessidade de stakeholders
- Produzir solução consumível
- Melhorar qualidade
- Acelerar entrega de valor

Transição

Lançar a solução em produção

- Garantir a prontidão em produção
- Implantar a solução em produção

Objetivos continuados

Melhorar e trabalhar de uma forma organizacional

- Desenvolver membros da equipe
- Coordenar atividades
- Gerenciar riscos
- Evoluir o WoW
- Alavancar a infraestrutura
- Governança de entregas

Papéis

- Stakeholder
- Dono do produto
- Membro da equipe
- Líder da equipe: além de líder é o coordenador das práticas ágeis
- Dono da arquitetura: define as decisões de arquitetura e facilita a criação e evolução de um design

Papéis potenciais

- Expert no domínio: PO representa um cliente no geral, talvez seja necessário alguém com uma expertise mais refinada
- Experts técnicos: administradores de banco de dados, experiência do usuário (UX), experts em segurança
- Tester independente
- Integrador