Ágil

Valores

Valores

- Agile Alliance Manifesto
  - Valores
    - Indivíduos e interações sobre processos e ferramentas
    - Software operacional sobre documentação abrangente
    - Colaboração de cliente sobre negociação de contrato
    - Resposta a mudança sobre seguimento de plano

Valores

- Indivíduos e interações sobre processos e ferramentas
  - Pessoas: ingrediente de sucesso mais importante
  - Trabalhar com outros, comunicando e interagindo

Valores

- Software operacional sobre documentação abrangente
  - Documentação extensa demanda tempo para produção e sincronização
  - Criar documento curto com análise racional do projeto e estruturas de mais alto nível no sistema

Valores

- Colaboração de cliente sobre negociação de contrato
  - Contrato com requisitos, cronograma e orçamento é falho
  - Melhores contratos governam o modo como time desenvolvedor e cliente trabalharão juntos

Valores

- Resposta a mudança sobre seguimento de plano
  - Habilidade de responder a mudança frequentemente determina sucesso de projeto
  - Construir planos que sejam flexíveis e prontos para adaptar a mudanças no negócio e tecnologia

Princípios

Princípios

- Valores inspiraram 12 princípios
  - Características diferenciadoras
    - Prioridade é satisfazer cliente com entrega contínua
    - Acolher requisitos mutáveis, mesmo tardios
    - Entregar software operacional frequentemente
    - Pessoas de negócio e desenvolvedores devem trabalhar juntos diariamente
    - Construir projetos ao redor de indivíduos motivados

Princípios

- Conversa face a face é o método mais eficiente e efetivo de transmitir informação para e dentro do time
- Software operacional é a medida primária de progresso
- Patrocinadores, desenvolvedores e usuários devem ser capazes de manter ritmo constante
- Atenção contínua a excelência técnica e bom projeto aumenta agilidade
- Simplicidade é essencial

Princípios

- Os melhores arquiteturas, requisitos e projetos emergem de times auto organizáveis
- Em intervalos regulares, o time reflete sobre como se tornar mais efetivo, então ajusta seu comportamento

Princípios

- Prioridade é satisfazer cliente com entrega contínua
  - Quanto menos funcional a entrega inicial, maior a qualidade na entrega final
  - Quanto mais frequente as entregas, maior a qualidade final

Princípios

- Acolher requisitos mutáveis, mesmo tardios
  - Mudança: conhecimento sobre o que é necessário para satisfazer mercado
  - Manter estrutura de software flexível para minimizar impacto de mudanças de requisitos

Princípios

- Entregar software operacional frequentemente
  - Entregar software operacional cedo (após primeiras poucas semanas)
  - Entregar software operacional frequentemente (a cada poucas semanas)

Princípios

- Pessoas de negócio e desenvolvedores devem trabalhar juntos diariamente
  - Para que projeto seja ágil, deve existir interação significativa e frequente entre clientes, desenvolvedores e envolvidos
  - Projeto de software deve ser continuamente guiado

Princípios

- Construir projetos ao redor de indivíduos motivados
  - Projeto ágil é aquele no qual pessoas são consideradas o fator de sucesso mais importante
  - Processo, ambiente, gestão, etc. são secundários e sujeitos a mudança se tiverem efeito adverso sobre as pessoas

Princípios

- Conversa face a face é o método mais eficiente e efetivo de transmitir informação para e dentro do time
  - Modo primário de comunicação é conversação
  - Time não demanda especificações, planos ou projetos escritos; pode criar por necessidade imediata e significativa, mas padrão é conversa

Princípios

- Software operacional é a medida primária de progresso
  - Progresso é mensurado por software que satisfaz a necessidade do cliente
  - Projetos ágeis não medem seu progresso em termos da fase em que se encontram

Princípios

- Patrocinadores, desenvolvedores e usuários devem ser capazes de manter ritmo constante
  - Time estabelece ritmo sustentável
  - Possibilitar a manutenção dos mais altos padrões de qualidade pela duração do projeto

Princípios

- Atenção contínua a excelência técnica e bom projeto aumenta agilidade
  - Qualidade é chave para velocidade
  - Membro de time ágil não produz código de baixa qualidade para melhorar quando tiver mais tempo

Princípios

- Simplicidade é essencial
  - Times escolhem o caminho mais simples que é consistente com seus objetivos
  - Fazem o trabalho hoje, confidentes que será fácil mudar se e quando problemas de amanhã surgirem

Princípios

- Os melhores arquiteturas, requisitos e projetos emergem de times auto organizáveis
  - Time determina melhor modo de preencher responsabilidades
  - Membros trabalham juntos em todos os aspectos do projeto

Princípios

- Em intervalos regulares, o time reflete sobre como se tornar mais efetivo, então ajusta seu comportamento
  - Time ajusta continuamente sua organização, regras, convenções, relacionamentos, etc.
  - Sabe que deve mudar com seu ambiente para permanecer ágil

Princípios Refinados

Princípios Refinados

- Revisão
  - Duplicação, declaração, prática
  - Organizacional x Técnico

Princípios Refinados

- Organizacional
  - Colocar o cliente no centro
  - Permitir auto organização do time
  - Trabalhar em um ritmo sustentável
  - Aceitar mudança

Princípios Refinados

- Organizacional
  - Desenvolver software mínimo
    - Produzir funcionalidade mínima
    - Produzir somente o produto requisitado
    - Desenvolver somente código e testes

Princípios Refinados

- Técnico
  - Expressar requisitos através de cenários
  - Desenvolver iterativamente
    - Produzir iterações de trabalho frequentes
    - Congelar requisitos durante iterações

Princípios Refinados

- Técnico
  - Tratar testes como um recurso chave
    - Não começar qualquer novo desenvolvimento até que todos os testes passem
    - Testar primeiro

Papéis

Papéis

- Responsabilidades
  - Atores em processo
    - Gerente
    - Dono do Produto
    - Time
    - Cliente
    - Treinador, Mestre

Papéis

- Gerente
  - Papel de apoio
  - Tutor, não cuidador

Papéis

- Dono do Produto
  - Define e mantém lista de características
  - Antes do desenvolvimento
    - As explica
  - Após
    - As aceita/rejeita

Papéis

- Time
  - Assume responsabilidades gerenciais, incluindo decisão de tarefas a implementar

Papéis

- Time
  - Auto organizado
    - Otimizar colaborações intensas dentro e através de fronteiras organizacionais para enfrentar desafios conforme surgem
  - Multi funcional
    - Time inclui todas as habilidades necessárias para entregar uma característica, independentemente de outros times

Papéis

- Time
  - Membros x Observadores
    - Membros devem dominar discussões, observadores devem oferecer opiniões quando solicitados

Papéis

- Cliente
  - Central
  - Interação constante
  - Indireção

Papéis

- Treinador, Mestre
  - Ajuda a manter valores, princípios e práticas ágeis
  - Exclusivo ou não

XP

XP

- Práticas
  - Cliente membro
  - Estórias de usuário
  - Ciclos curtos

XP

- Práticas
  - Testes de aceitação
  - Programação em par
  - Desenvolvimento dirigido por teste

XP

- Práticas
  - Propriedade coletiva
  - Integração contínua
  - Ritmo sustentável

XP

- Práticas
  - Local de trabalho aberto
  - O jogo de planejamento
  - Projeto simples

XP

- Práticas
  - Refatoração
  - Metáfora

XP

- Cliente membro
  - Clientes são membros do, e disponíveis para o, time
  - Se o cliente não pode estar perto, encontrar alguém capaz de substituir o cliente real

XP

- Estórias de usuário
  - Obter o sentido dos detalhes dos requisitos conversando com o cliente, sem capturá-los
  - Cliente escreve poucas palavras em um cartão, lembrete da conversa
  - Desenvolvedores escrevem estimativa no cartão com base no sentido obtido

XP

- Ciclos curtos
  - Entrega de software operacional a cada duas semanas
  - Software que satisfaz algumas das necessidades dos envolvidos
  - Sistema demonstrado no fim da iteração para obter feedback dos envolvidos

XP

- Testes de aceitação
  - Detalhes sobre estórias capturados na forma de testes de aceitação especificados pelo cliente
  - Testes para uma estória escritos imediatamente antes ou concorrentemente à implementação da estória
  - Agem para verificar se sistema comporta como especificado

XP

- Programação em par
  - Código escrito por pares de programadores trabalhando juntos na mesma estação
  - Um digita o código
  - Outro procura por erros e melhorias

XP

- Desenvolvimento dirigido por teste
  - Código escrito para fazer testes unitários de falha passarem
  - Primeiro escrever teste que falha pois funcionalidade inexistente
  - Então escrever código que faz teste passar

XP

- Propriedade coletiva
  - Um par tem o direito de dar saída a qualquer módulo e melhorá-lo
  - Programadores não são individualmente responsáveis por qualquer módulo particular

XP

- Integração contínua
  - Programadores dão entrada a seus códigos e integram muitas vezes por dia
  - O primeiro a dar entrada vence, os demais combinam

XP

- Ritmo sustentável
  - Time que dispara no início pode esgotar antes do fim
  - Hora extra somente na última semana em uma liberação

XP

- Local de trabalho aberto
  - Time trabalha junto em sala aberta
  - Cada mesa com duas ou três estações, com duas cadeiras
  - Paredes cobertas com gráficos, diagramas, etc.

XP

- O jogo de planejamento
  - Essência é divisão de responsabilidade entre negócio e desenvolvimento
  - Pessoas de negócio (os clientes) decidem importância de uma característica
  - Desenvolvedores decidem quanto sua implementação custará

XP

- Projeto simples
  - Fazer projetos tão simples quanto possível
  - Limitar foco para considerar somente estórias planejadas para iteração corrente
  - Não preocupar com estórias que virão

XP

- Refatoração
  - Conforme características são adicionadas e bugs tratados, estrutura do código degrada
  - Degradação revertida através de frequente refatoração
    - Prática de fazer série de transformações pequenas que melhoram estrutura do sistema sem afetar seu comportamento

XP

- Metáfora
  - É o todo que liga o sistema inteiro junto
  - É a visão do sistema que torna a localização e o formato de todos os módulos individuais óbvios
  - Se um módulo está inconsistente com a metáfora, então o módulo está incorreto

Scrum

Scrum

- Princípios usados para guiar atividades dentro de um processo
  - Requisitos
  - Análise
  - Projeto
  - Evolução
  - Entrega

Scrum

- Dentro de cada atividade, tarefas ocorrem dentro de um padrão de processo – sprint
- Trabalho conduzido dentro de sprint é adaptado ao problema e definido e frequentemente modificado em tempo real pelo time
- Quantidade de sprints para cada atividade depende da complexidade do produto

Scrum

- Enfatiza uso de conjunto de padrões de processo de software efetivos para projetos com
  - Cronograma apertado
  - Requisitos mutáveis
  - Criticidade de negócio

Scrum

- Cada padrão define conjunto de ações de desenvolvimento
  - Backlog
  - Sprints
  - Scrum meetings
  - Demos

Scrum

- Backlog
  - Lista priorizada de requisitos ou características de projeto
  - Itens podem ser adicionados a qualquer momento (introdução de mudanças)
  - Product manager avalia e atualiza prioridades

Scrum

- Sprints
  - Unidades de trabalho requeridas para satisfazer um requisito definido no backlog
  - Devem ser acomodadas em um período de tempo pré-definido (30 dias tipicamente)
  - Mudanças não introduzidas durante sprint
    - Proporciona aos membros do time ambiente de trabalho estável

Scrum

- Scrum meetings
  - Reuniões curtas (15 minutos tipicamente) realizadas diariamente pelo time
  - Três questões chave perguntadas e respondidas por todos membros
    - O que fez desde última reunião?
    - Quais obstáculos está encontrando?
    - O que planeja realizar até próxima reunião?

Scrum

- Scrum meetings
  - Scrum master
    - Líder de time
    - Conduz reunião e avalia respostas
  - Ajudam descobrir cedo problemas potenciais
  - Promovem estrutura de time auto organizável

Scrum

- Demos
  - Entregar incremento de software ao cliente para que funcionalidade implementada possa ser demonstrada e avaliada pelo cliente
  - Pode não conter toda funcionalidade planejada, mas aquelas funções que podem ser entregues dentro do período de tempo estabelecido

Dynamic Systems Development Method (DSDM)

DSDM

- Satisfaz restrições justas de tempo através do uso de prototipação incremental em ambiente de projeto controlado
- Filosofia emprestada de versão do princípio de Pareto (80% de aplicação em 20% de tempo)

DSDM

- Processo iterativo
  - Somente trabalho suficiente é requerido para cada incremento para facilitar avanço ao próximo incremento
  - Detalhe restante pode ser completado quando mais requisitos de negócio são conhecidos ou mudanças foram requisitadas e acomodadas

DSDM

- Três ciclos iterativos precedidos por duas atividades de ciclo de vida adicionais
  - Estudo de viabilidade
  - Estudo de negócio
  - Iteração de modelo funcional
  - Iteração de projeto e construção
  - Implementação

DSDM

- Estudo de viabilidade
  - Estabelece os requisitos e restrições de negócio básicos associados com a aplicação a ser construída
  - Então avalia se a aplicação é uma candidata viável para o processo DSDM

DSDM

- Estudo de negócio
  - Estabelece os requisitos de função e informação que farão com que a aplicação forneça valor de negócio
  - Também define a arquitetura básica e identifica os requisitos de manutenibilidade para a aplicação

DSDM

- Iteração de modelo funcional
  - Produz protótipos incrementais que demonstram funcionalidade ao cliente
    - Todos os protótipos evoluem na aplicação
  - Objetivo é coletar requisitos adicionais por meio do feedback dos usuários ao exercitarem o protótipo

DSDM

- Iteração de projeto e construção
  - Revisita protótipos para assegurar que foram criados em uma maneira que habilitará o fornecimento de valor aos usuários finais
  - Pode ocorrer concorrentemente ao modelo funcional

DSDM

- Implementação
  - Posiciona último incremento de software no ambiente operacional
  - Incremento pode não estar completo ou mudanças podem ser requisitadas conforme o incremento é posicionado
    - Trabalho continua retornando à iteração de modelo funcional

Feature Driven Development (FDD)

FDD

- Enfatiza colaboração entre pessoas em um time
- Gerencia complexidade usando decomposição baseada em características e integração de incrementos
- Comunicação de detalhes técnicos usando meios verbal, gráfico e textual

FDD

- Enfatiza garantia de qualidade encorajando
  - Estratégia incremental
  - Uso de inspeções
  - Aplicação de auditorias
  - Coleta de métricas
  - Uso de padrões

FDD

- Característica
  - Função de valor para cliente que pode ser implementada em duas semanas ou menos
- Ênfase na definição de características provê benefícios
  - Usuários podem descrevê-las mais facilmente, compreender relacionamentos, e revisar ambiguidade, erro ou omissão

FDD

- Características podem ser organizadas em agrupamentos hierárquicos relacionados a negócio
- Time desenvolve características operacionais a cada duas semanas
- Representações de projeto e código das características são mais fáceis de inspecionar
- Planejamento e execução são dirigidos pela hierarquia de características

FDD

- Modelo para definir uma característica

<ação> o <resultado> <por | para | de | a> um <objeto>

- Exemplos

Adicionar o produto ao carrinho de compra

Exibir as especificações do produto

FDD

- Um conjunto de características agrupa características relacionadas em categorias relacionadas a negócio

<ação> um <objeto>

- Exemplos

Fazer uma venda de produto

FDD

- Cinco atividades de framework
  - Desenvolver um modelo geral
    - Mais formato do que conteúdo
  - Construir uma lista de características
    - Uma lista de características agrupadas em conjuntos e áreas de assunto
  - Planejar por característica
    - Um plano de desenvolvimento

FDD

- Projetar por característica
  - Um pacote de projeto
- Construir por característica
  - Função de valor para cliente completa

FDD

- Compreensão do status do projeto é essencial para os envolvidos
  - O que foi alcançado com sucesso e problemas encontrados
- Se pressão de prazo é significativa, determinar se características estão adequadamente programadas é crítico

FDD

- Seis marcos durante projeto e implementação de característica
  - Demonstração de tarefa de projeto
  - Projeto
  - Inspeção de projeto
  - Código
  - Inspeção de código
  - Promover a build

Agile Unified Process (AUP)

AUP

- Cada iteração inclui
  - Modelagem
  - Implementação
  - Teste
  - Implantação
  - Gerência de configuração e projeto
  - Gerência ambiental

AUP

- Modelagem
  - Representações UML dos domínios de negócio e problema são criadas
  - Entretanto, para manter agilidade, tais modelos devem ser apenas bons o suficiente para permitir ao time progredir

AUP

- Implementação
  - Modelos são traduzidos em código fonte

AUP

- Teste
  - Time projeta e executa uma série de testes para descobrir erros e assegurar que o código fonte satisfaz seus requisitos

AUP

- Implantação
  - Entrega de um incremento de software e aquisição de feedback de usuários finais

AUP

- Gerência de configuração e projeto
  - Gerência de mudanças, gerência de riscos e controle de quaisquer produtos de trabalho persistentes
  - Rastrear e controlar o progresso do time e coordenar as atividades do time

AUP

- Gerência ambiental
  - Coordena uma infraestrutura de processo que inclui padrões, ferramentas e outras tecnologias de apoio disponíveis para o time

Modelagem Ágil

Modelagem Ágil

- Princípios
  - Modelar com propósito
  - Usar múltiplos modelos
  - Viajar leve
  - Conteúdo x representação
  - Conhecer modelos e ferramentas
  - Adaptar localmente

Modelagem Ágil

- Modelar com propósito
  - Ter objetivo específico em mente antes de criar modelo
    - Comunicar informação a cliente ou ajudar melhor compreender aspecto de software
  - Tipo de notação a ser usado e nível de detalhe requerido mais óbvios quando objetivo identificado

Modelagem Ágil

- Usar múltiplos modelos
  - Muitos modelos e notações podem ser usados para descrever software
  - Apenas pequeno subconjunto essencial para maioria de projetos

Modelagem Ágil

- Usar múltiplos modelos
  - Para discernimento necessário cada modelo deve apresentar aspecto diferente de sistema
  - Somente modelos que fornecem valor a suas audiências pretendidas devem ser usados

Modelagem Ágil

- Viajar leve
  - Manter somente modelos com valor de longo prazo
  - Cada produto de trabalho deve ser mantido consistente com mudanças
  - Ao decidir manter modelo agilidade trocada por conveniência de ter informação disponível a time

Modelagem Ágil

- Conteúdo x representação
  - Modelagem deve transmitir informação a audiência pretendida
  - Modelos
    - sintaticamente perfeito que transmite pouco conteúdo útil
    - com notação falha que fornece conteúdo valioso

Modelagem Ágil

- Conhecer modelos e ferramentas
  - Compreender forças e fraquezas de cada modelo e conhecer ferramentas usadas para criá-los

Modelagem Ágil

- Adaptar localmente
  - Abordagem de modelagem deve ser adaptada a necessidades de time

Planejamento

Planejamento

- Exploração inicial
  - Início: características significativas
    - Descoberta contínua
  - Característica escrita em estórias de usuário
    - Sem detalhes
  - Estimativa: pontuar estória

Planejamento

- Velocidade
  - Semanalmente: completar estórias
  - Velocidade: somar estimativas
  - 3-4 semanas: velocidade média
    - Prever trabalho semanal

Planejamento

- Planejamento de liberação
  - Desenvolvedores e clientes definem data para primeira liberação
  - Clientes selecionam estórias implementadas em liberação de acordo com velocidade

Planejamento

- Planejamento de iteração
  - Desenvolvedores e clientes definem tamanho de iteração
  - Clientes selecionam estórias implementadas em iteração de acordo com velocidade
  - Iteração termina em data especificada
    - Velocidade planejada para iteração é a medida da anterior

Planejamento

- “Finalizado”
  - Estória não finalizada até testes de aceitação passarem
  - Escritos em início de iteração
    - Definem detalhes de estórias
    - Autoridade final sobre funcionamento de estórias

Planejamento

- Planejamento de tarefa
  - Em início de iteração desenvolvedores dividem estórias em tarefas de desenvolvimento
  - Desenvolvedores escolhem tarefas de acordo com pontos implementados em iteração anterior
  - Time reúne em metade de iteração: metade de estórias deve estar completa

Planejamento

- Iterando
  - Clientes avaliam executável em final de iteração
    - Fornecem feedback em termos de novas estórias

Planejamento

- Rastreamento
  - Registrar resultado de iteração e usar para prever o que acontecerá em próximas
  - Gráfico de velocidade
    - Quantidade de pontos completos semanalmente
  - Gráfico de queima
    - Semanalmente, quantidade de pontos faltantes para completar liberação

Teste

Teste

- Test-Driven Development (TDD)
  - Escrever teste antes de código
  - Escrever teste suficiente para falhar
  - Escrever código suficiente para passar

Teste

- TDD
  - Cada função tem testes que verificam sua operação
  - Testes indicam que funcionalidade existente foi quebrada
  - Alterações em programa sem medo de quebrar algo importante
  - Testes indicam que programa continua funcionando

Teste

- TDD
  - Teste primeiro impõe diferente ponto de vista
  - Ver programa a partir de perspectiva de invocador
  - Interesse imediato por interface assim como função

Teste

- TDD
  - Projetar programa para ser testável
  - Para ser invocável e testável software deve ser desacoplado de seus arredores
  - Teste primeiro força a desacoplar software

Teste

- TDD
  - Testes atuam como forma de documentação
  - Para saber como invocar função existe teste que mostra
  - Testes atuam como exemplos que ajudam programador entender como trabalhar com código
  - Documentação executável

Teste

- Testes de Aceitação
  - Testes unitários necessários mas insuficientes como ferramentas de verificação
    - Verificam que elementos pequenos de sistema funcionam conforme esperado
    - Não verificam que sistema funciona apropriadamente como um todo

Teste

- Testes de Aceitação
  - Testes unitários são testes caixa branca
    - Verificam mecanismos individuais de sistema
  - Testes de aceitação são testes caixa preta
    - Verificam que requisitos de cliente foram satisfeitos

Teste

- Testes de Aceitação
  - Escritos por pessoas que desconhecem mecanismos internos de sistema
  - Podem ser escritos diretamente por cliente ou por analistas de negócio, testadores, ou especialistas em garantia de qualidade

Teste

- Testes de Aceitação
  - São documentação final de característica
  - Após cliente escrever testes que verificam correção de característica programadores podem lê-los para compreender característica
  - Documentação executável de características de sistema

Teste

- Testes de Aceitação
  - Escrever testes de aceitação primeiro afeta arquitetura
  - Para tornar sistema testável ele deve estar desacoplado em alto nível arquitetural
  - Interface de usuário deve estar desacoplada de regras de negócio

Teste

- Testes de Aceitação
  - Testes unitários levam a decisões superiores de projeto em pequena escala
  - Testes de aceitação levam a decisões superiores de projeto em escala arquitetural