

TÓPICO 5

**Modelação de sistemas**

* Modelação de sistemas é o processo de desenvolver modelos abstratos de um sistema, em que cada modelo representa uma perspetiva diferente desse sistema
* Modelação de sistemas representa agora um sistema usando um tipo de notação gráfica que se baseia em notações UML
* Modelação de sistemas ajuda o analista a perceber a funcionalidade do sistema e os modelos que são usados para comunicar com os clientes
* Modelos já existentes do sistema são usados durante a fase de engenharia de requisitos. Eles ajudam a clarificar o que é que o sistema já faz e pode ser usado como uma base para discutir as forças e as fraquezas.
* Modelos do novo sistema são usados durante a fase de engenharia de requisitos para ajudar a explicar o propósito dos requisitos para os stakeholders. Engenheiros usam estes modelos para discutir as propostas de design e para documentar o sistema para implementação
* Num processo de engenharia movido a modelos, é possível gerar por completo ou parcialmente a implementação de um sistema.

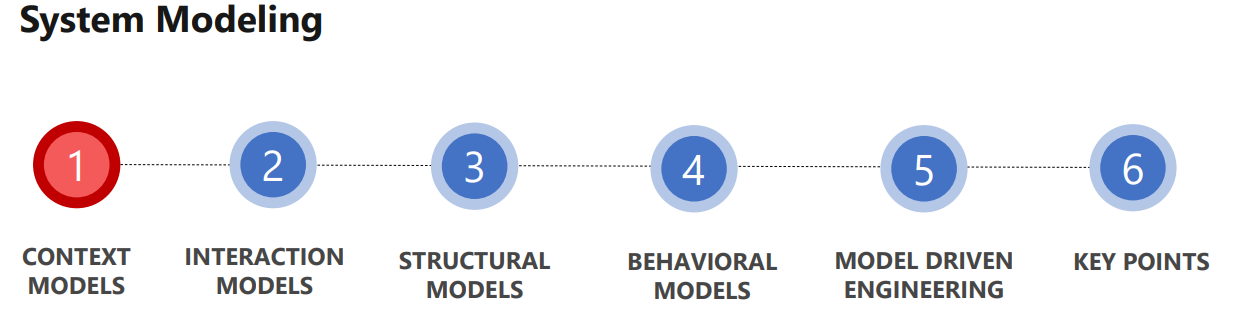
**Perspetivas do sistema**

* Perspetiva externa, onde modelamos o contexto e o ambiente do sistema
* Perspetiva interativa, onde modelamos as interações entre o sistema e o seu ambiente, ou entre os componentes do sistema
* Perspetiva estrutural, onde modelamos a organização de um sistema ou a estrutura dos dados que é processada pelo sistema.
* Perspetiva comportamental, onde modelamos a dinâmica do comportamento do sistema e como responde aos eventos

**Tipos de diagramas UML**

* **Diagramas de atividade** – mostram as atividades envolvidas no processo.
* **Diagramas de caso de uso** – que mostram as interações entre o sistema e o seu ambiente
* **Diagramas de sequência** – que mostram as interações entre os atores e o sistema e entre os componentes do sistema
* **Diagramas de classe** - que mostra os objetos das classes no sistema e as associações entre essas classes
* **Diagramas de Estado** – que mostram como é que o sistema reage a eventos internos e externos

Uso de modelos gráficos

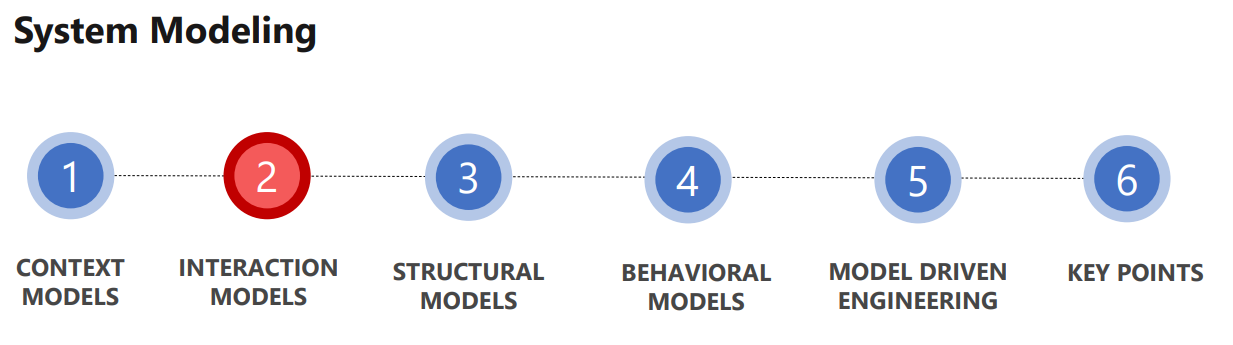
* Facilita a discussão acerca de um sistema proposto ou já existente
  + Modelos incorretos e incompletos são OK pois a sua função é suportar a discussão
* Uma maneira de documentar um sistema já existente
  + Os modelos devem ser uma representação accurate do sistema e não precisam de ser completas
* Como uma descrição detalhada do sistema que pode ser usada para gerar a implementação do sistema
  + Modelos devem estar corretos e completos.

**LIMITES DO SISTEMA**

* Os limites do sistema são estabelecidos para definir o que está dentro e fora do sistema
* A posição do limite do sistema tem um efeito profundo nos requisitos do sistema
* Definir um limite do sistema é um judgment politico

**PERSPETIVA DO PROCESSO**

* Contexto dos modelos mostram aos outros sistemas no ambiente, como é que um sistema não deve ser desenvolvido.
* Modelos de processo revelam como é que o sistema a ser desenvolvido é usado em processos de negócio
* Diagramas de atividade UML podem ser usados para definir modelos de processo de negócio.



**Modelos de interação**

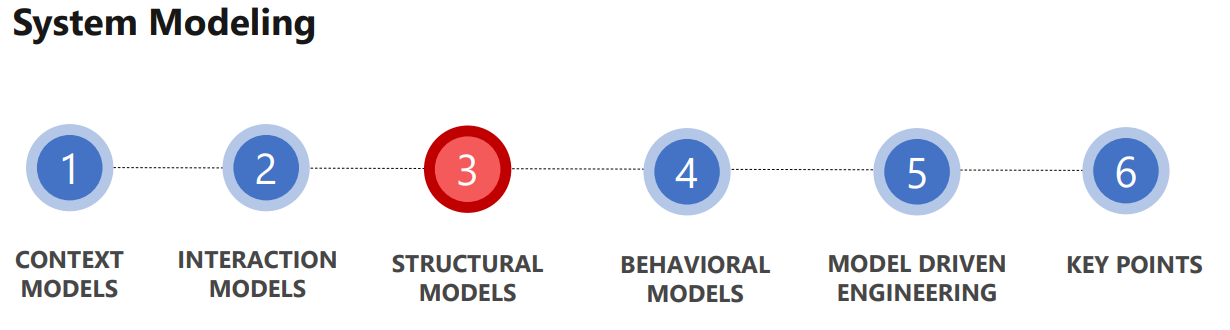
* Modelar interações de utilizadores é importante porque ajuda a clarificar os requisitos do utilizador
* Modelar interações Sistema – a – Sistema destaca os problemas de comunicação que podem vir a surgir
* Modelar interações entre componentes ajuda-nos a perceber se a estrutura do sistema proposta é capaz de nos oferecer a performance e a dependência desejada

**Modelação de casos de uso**

* Casos de uso foram desenvolvidos originalmente para suportar requisitos de elicitação e agora incorporados no UML
* Cada caso de uso representa uma tarefa descrita que envolve interações externas com o sistema
* Atores num caso de uso podem ser pessoas ou outros sistemas
* São representados através de diagramas para fornecer uma visão geral do caso de uso numa forma mais detalhada do texto

**Diagramas de sequência**

* Diagramas de sequência fazem parte do UML e são usados para modelar interações entre atores e objetos dentro do sistema
* Um diagrama de sequência mostra as sequências de interações que têm lugar durante o caso de uso particular
* Os objetos e os atores envolvidos são listados ao longo do topo do diagrama
* Interações entre objetos são indicadas através de setas.



**Modelos estruturais**

* Modelos estruturais do software mostram a organização do sistema em termos de componentes que fazem esse sistema e as suas relações
* Modelos estruturais podem ser modelos estáticos, que mostram a estrutura do design do sistema ou modelos dinâmicos, que mostram organização do sistema quando está a ser executado
* Criamos modelos estruturais do sistema quando estamos a discutir e a desenhar a arquitetura do sistema

Diagramas de classe

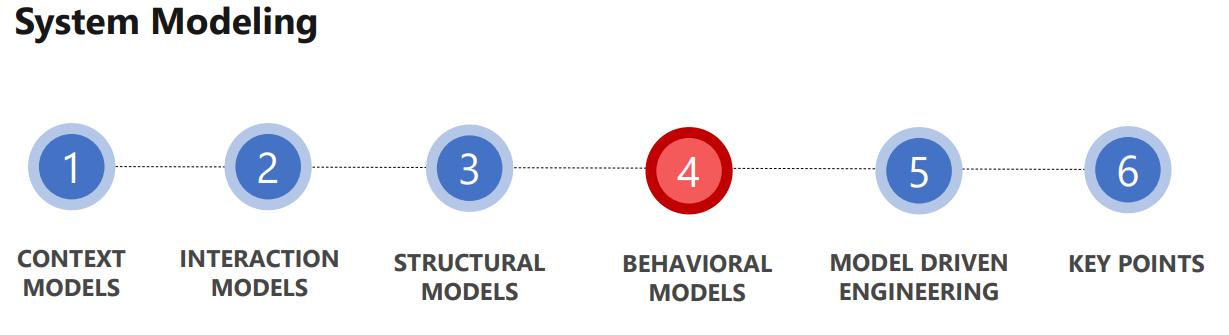
* Diagramas de classe são usados para desenvolver sistemas orientados a objetos para mostrar classes num sistema e as associações entre essas classes
* Uma classe de objeto pode ser pensado como uma definição geral de um tipo de objeto de sistema
* Uma associação é uma ligação entre classes que indica que existe uma relação entre estas classes
* Quando estamos a desenvolver modelos durante os early stages do processo de engenharia de software, objetos representam algo na vida real

Generalização

* Generalização é uma técnica que usamos para gerir a complexidade
* Em vez de aprender as características detalhadas de cada entidade, generalizamos.
* Em sistemas de modelação, é geralmente usado para examinar classes num sistema para ver se há um scope para generalização. Se mudanças forem propostas, então não tens que olhar para todas as classes no sistema para ver se foram afetadas pela mudança
* Em linguagens orientadas a objetos, como java, generalização é implementada usando classes de herança.
* Na generalização, os atributos e operações associados a classes de maior nível tbm estão associadas a classes de menor nível
* As classes de menor nível são subclasses que herdam os atributos e as operações das suas super classes.

Agregação de objetos e modelos

* Um modelos de agregação mostra como é que as classes que são coleção são compostas de outras classes



Modelos comportamentais

* Modelos comportamentais são modelos que se comportam dinamicamente, ou seja, à medida que o sistema está a correr.

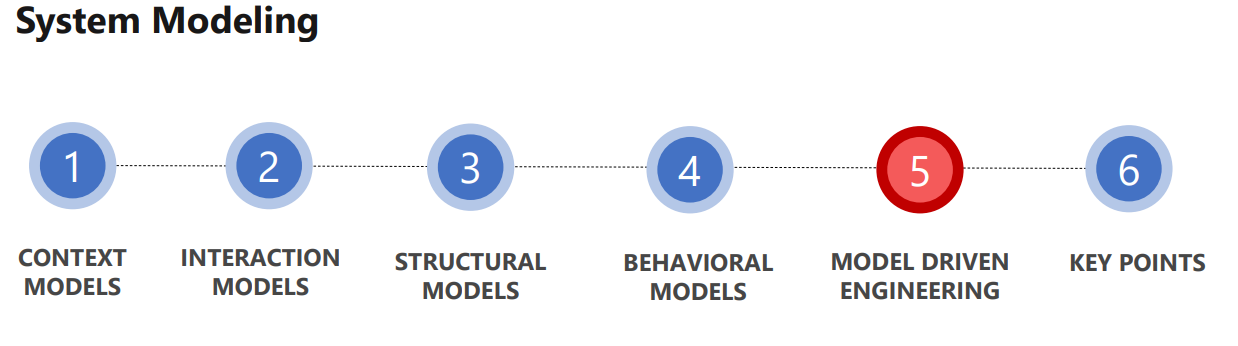
Data driven modeling

* Muitos sistemas de negócio são sistemas de processamento de dados que são primariamente corridos por dados. São controlados pelo input dos dados para o sistema, com relativamente pouco processamento externo.
* Mostram a sequência de ações envolvidas em processar os inputs dos dados gerando um output associado.

Event Driven Modeling

* Sistemas em tempo real são normalmente event-driven, com o mínimo de processamento de dados.
* Mostra-nos como é que um sistema responde a eventos internos e externos.

State machine models

* Estes modelam o comportamento do sistema em resposta a eventos internos e externos

Model Driven Engineering

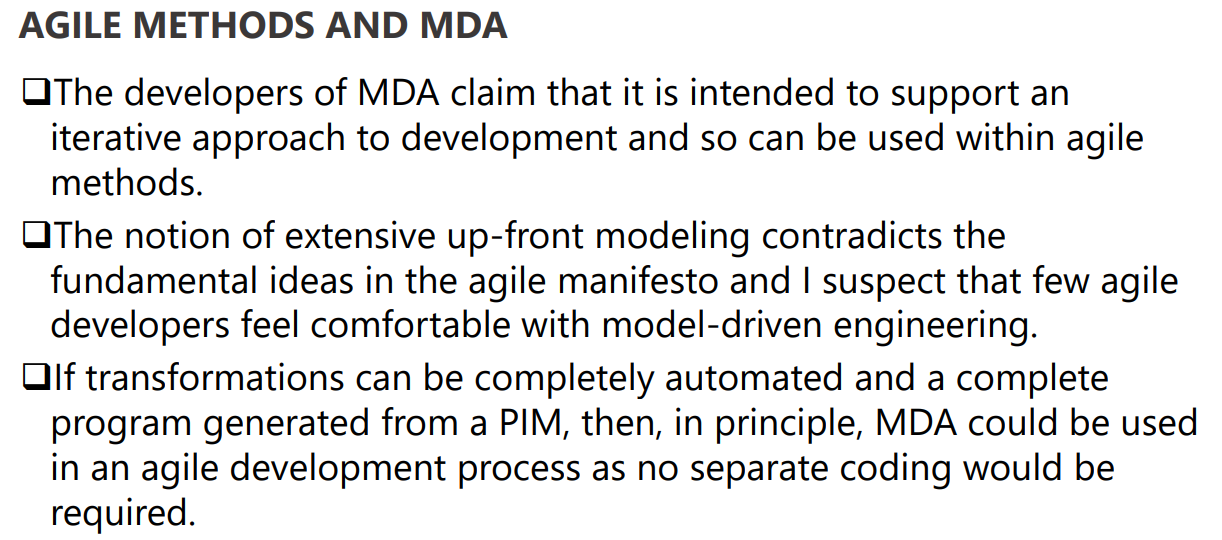
* É uma abordagem ao desenvolvimento de software onde os modelos em vez dos programas são o principal output do processo de desenvolvimento
* Os programas que executam numa plataforma hardware/software são gerados automaticamente a partir de modelos.

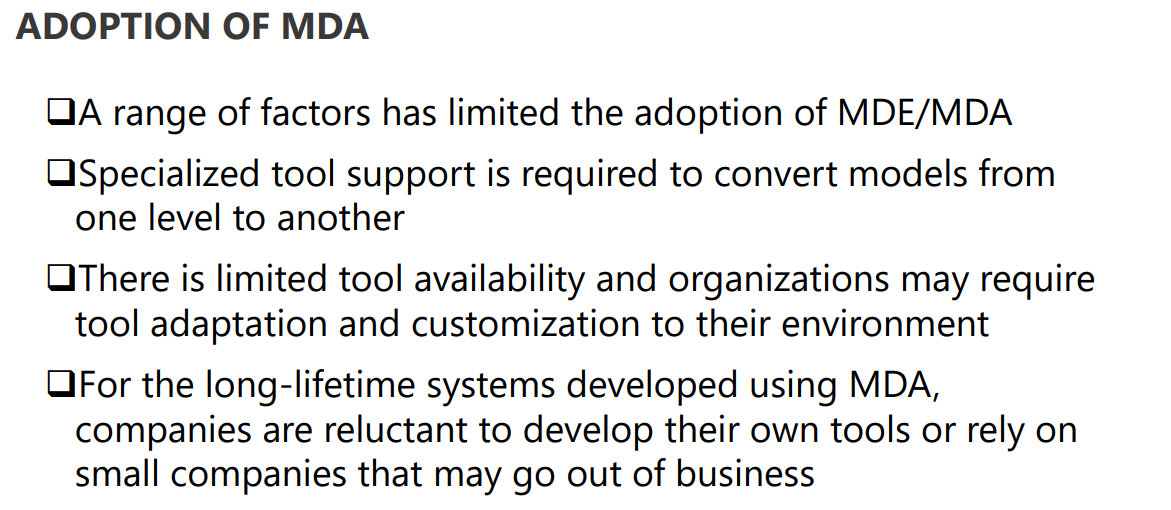
Usage of Model Driven Engineering

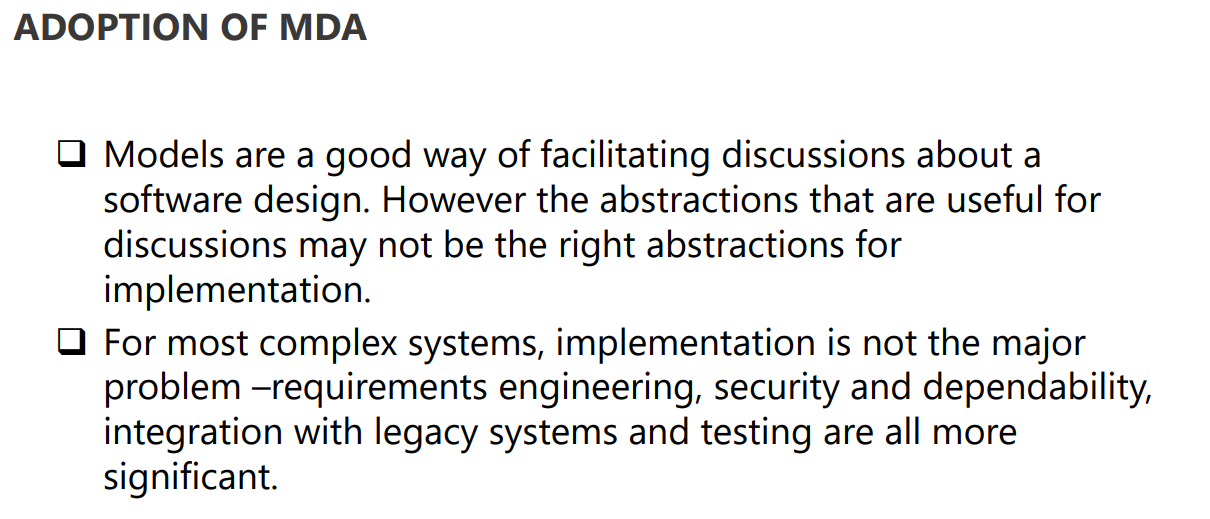
* Ainda é um estádio inicial do desenvolvimento, e é um pouco incerto se vai ter um impacto grande no processo de engenharia de software.
* Pros
  + Permite que os sistemas sejam considerados em níveis superiores de abstração
  + Gerar código automaticamente significa que é mais fácil adaptar sistemas a novas plataformas.
* Contras
  + Modelos para abstração podem não ser os corretos para implementação

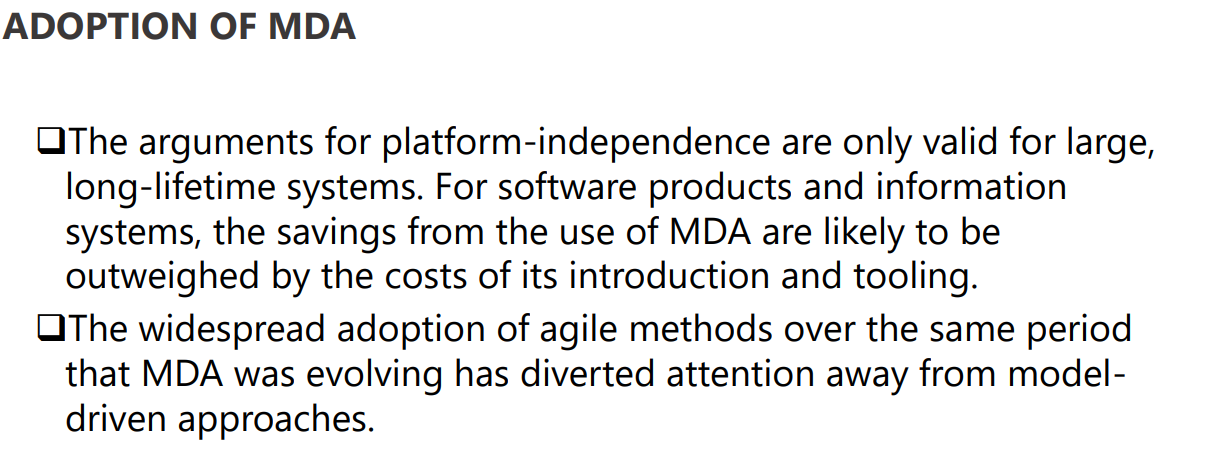
Tipos de modelos

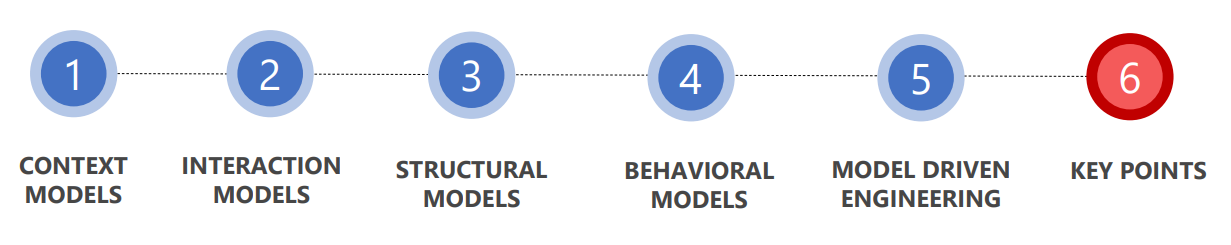
* Um modelos de computação independente (CIM)
  + Modela os dominiios de abstração importantes usados no sistema.
* Modelo independente (PIM)
  + Estes modelam a operação do sistema sem referenciar a sua implementação
* Modelos específicos de plataforma (PSM)



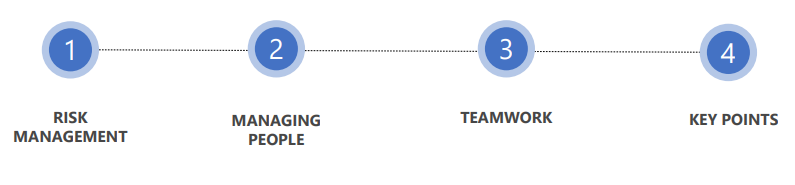








**KEY POINTS**

* Um modelo é uma visão abstrata de um sistema que ignora os detalhes do sistema. Os modelos complementares do sistema podem ser desenvolvidos para mostrar o contexto do sistema, interações, estrutura e comportamento.
* Modelos de contexto mostram o sistema que está a ser modelado a ser posicionado num ambiente com outros sistemas e processos
* Diagramas de caso de uso e diagramas de sequência são usados para descrever interações entre utilizadores e sistemas no sistema a ser desenhado. Casos de uso descrevem interações entre o sistema e atores externos. Diagramas de sequência adicionam informação a estes mostrando interações entre objetos do sistema
* Modelos estruturais mostram a organização e a arquitetura do sistema. Diagramas de classe são usados para definir estruturas estáticas de classes num sistema e as suas associações.
* Processos de software são atividades envolvidas em produzir o software do sistema. Modelos de processo de software são representações abstratas destes processos.
* Modelos de processos gerais descrevem a organização dos processos de software.
* Engenharia de requisitos é o processo de desenvolver especificações de software.

Gestão de projetos é necessário para definir os orçamentos para o desenvolvimento do software.

CRITÉRIO DE SUCESSO

* Entregar software ao cliente a tempo
* Manter os custos dentro do orçamento
* Entregar software que vai de encontro com as expectativas do utilizador
* Manter uma equipa de desenvolvimento funcional e com coerência.

Distinções em gestão de projeto

* O produto é intangível
  + O software não pode ser tocado ou visto. Os gestores dos projetos de software não conseguem ver o progresso.
* Muitos projetos são one-off
  + Projetos grandes de software costumam ser diferentes de projetos anteriores.
* Processos de software são variáveis e específicos a cada organização

Fatores que influenciam a gestão de projeto

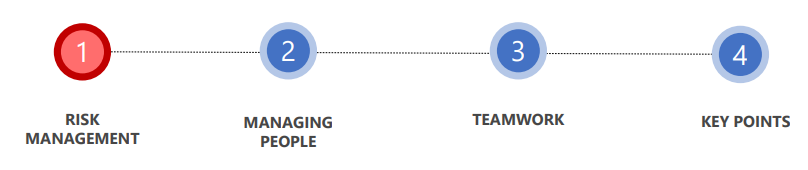
* Tamanho da empresa
* Clientes de software
* Tamanho do software
* Tipo de software
* Cultura organizacional
* Processos de desenvolvimento de software

Atividades de gestão universais

* Planeamento de projetos
  + Gestores de projetos são responsáveis por planear, estimar e definir as tarefas das pessoas.
* Gestão de risco
  + Gestores de projeto avaliam os riscos que podem afetar o projeto, monitorizam estes riscos e tomam ação quando os problemas vêm ao de cima.
* Gestão do pessoal
  + Gestores do projeto têm que escolher as pessoas para a sua equipa e estabelecer maneiras de trabalhar que o levam a uma performance efetiva da equipa.

Atividades de gestão

* Reportar
  + Gestores do projeto são usualmente responsáveis por reportar o progresso do projeto aos clientes a aos gestores da empresa.
* Proposal Writing
  + O primeiro estádio num projeto de software pode envolver escrever uma proposta para ganhar um contrato. A proposta descreve os objetivos do projeto e como vai ser.



**GESTÃO DE RISCO**

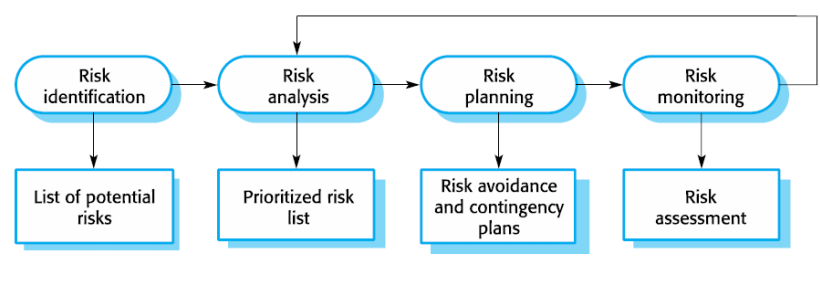
* Gestão de risco preocupa-se com identificar os riscos e desenhar os planos de forma a minimizar o efeito dos riscos no projeto.
* Gestão do risco de software é importante porque herda de algumas incertezas em desenvolvimento de software.
* Devemos antecipar os riscos, perceber os impactos destes riscos no projeto, no produto e no negócio, e tomar estes passos de forma a evitar estes riscos

**CLASSIFICAÇÃO DO RISCO**

* Existem duas dimensões de classificação de risco
  + O tipo de risco (técnico, organizacional)
  + O que é afetado pelo risco:
    - Riscos do projeto afetam o calendário e os seus recursos
    - Riscos do produto afetam a qualidade e a performance do software a ser desenvolvido
    - Riscos de negócio afetam a organização desenvolvendo o software.

**Processo de gestão de risco**

* **Identificação do risco**
  + Identificar projeto, produto e risco de negócio
* **Análise de risco**
  + Avaliar as consequências do risco
* **Planear o risco**
  + Desenhar os planos de forma a evitar ou minimizar os efeitos desse risco
* **Monitorizar o risco**
  + Monitorizar os riscos durante o projeto



**IDENTIFICAÇÃO DO RISCO**

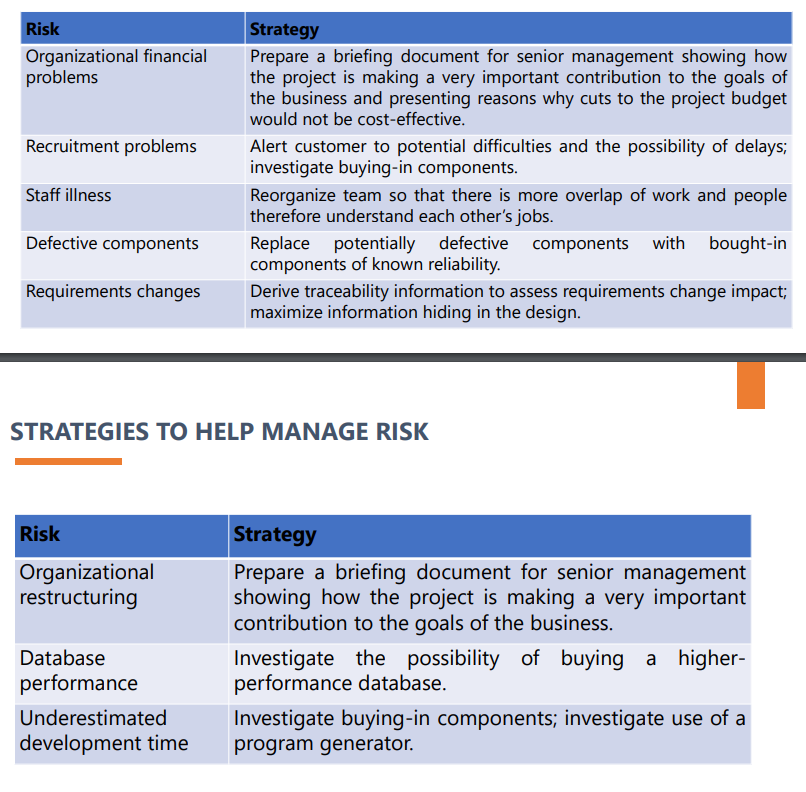
* Pode ser atividades de equipa baseadas nos projetos individuais do gestor.
* Alguns riscos comuns
  + Riscos tecnológicos
  + Riscos Organizacionais
  + Riscos com as pessoas
  + Riscos dos requisitos
  + Riscos de estimativa

**ANÁLISE DE RISCO**

* Avaliar a probabilidade e a seriedade de cada risco
* A probabilidade pode ser bastante baixa, baixa, moderada, alta ou muito alta
* As consequências do risco podem ser catastróficas, sérias, toleráveis ou insignificantes

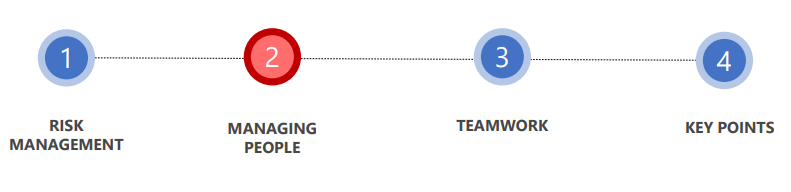
**PLANEAMENTO DO RISCO**

* Considera cada risco e desenvolve uma estratégia para gerir esse risco
* Estratégias de evasão
  + A probabilidade de o risco subir é reduzida
* Estratégias de minimização
  + O impacto do risco no projeto ou produto ser reduzido
* Planos de contingência
  + Se o risco subir, os planos de contingência são planos para lidar com esse risco



**MONITORIZAÇÃO DO RISCO**

* Avaliar cada risco identificado para decidir se está a ficar menos ou mais provável
* Avaliar se o efeito do risco mudou ou não
* Cada risco chave deve ser discutido nas reuniões de gestão de progresso



**GERIR PESSOAS**

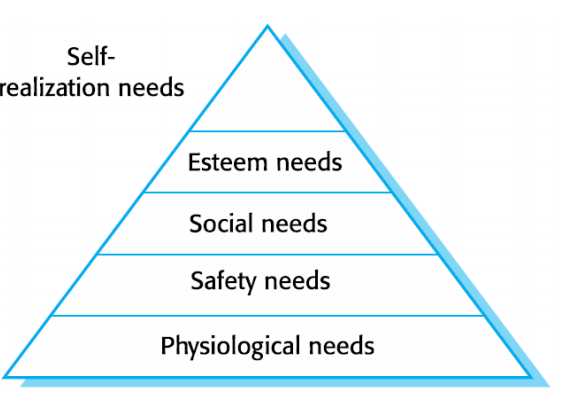
* Pessoas são ativos importantes numa organização
* A tarefas de um gestor é essencialmente orientado às pessoas.
* Baixa gestão das pessoas é um importante contributo para o falhanço do projeto

**FACTORES DA GESTÃO DAS PESSOAS**

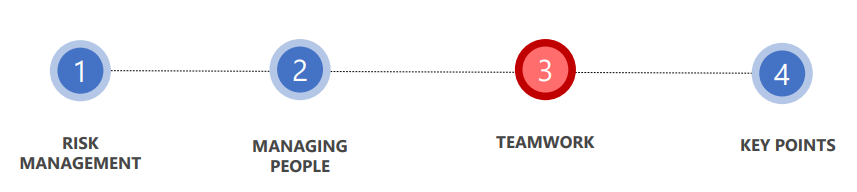
* Consistência
  + Membros da equipa devem ser tratados da mesma maneira
* Respeito
  + Os membros da equipa têm diferentes skills e estas diferenças devem ser respeitadas
* Inclusão
  + Envolver a equipa toda
* Honestidade
  + Ser honest sempre

**MOTIVAR AS PESSOAS**

* Uma das tarefas mais importantes de um gestor é motivar as pessoas que estão a trabalhar no seu projeto



**TIPOS DE PERSONALIDADE**

* Orientados às tarefas
  + A motivação para fazer o trabalho é o trabalho em sí
* Self – oriented
  + Objetivos pessoais
* Orientado à orientação
  + A principal motivação é a presença e as ações dos seus co-workers. As pessoas vão para o trabalho pq gostam de trabalhar

**TRABALHO DE EQUIPA**

* Maior parte da engenharia de software é um grupo de atividade
* Um bom grupo é um grupo coeso e tem espírito de equipa. As pessoas envolvidas são motivadas pelo sucesso do grupo bem como os seus objetivos pessoais.
* Interação de grupo é um fator determinante para a performance de grupo
* Flexibilidade na composição de um grupo é limitada

**COESÃO DO GRUPO**

* Num grupo coeso, os membros consideram o grupo a ser mais importante que qualquer individual nele
* As vantagens de um grupo coeso são
  + Qualidade do grupo é desenvolvido pelos membros do grupo
  + Os membros da equipa aprendem entre eles
  + O conhecimento é partilhado

**EFICIÊNCIA DE UMA EQUIPA**

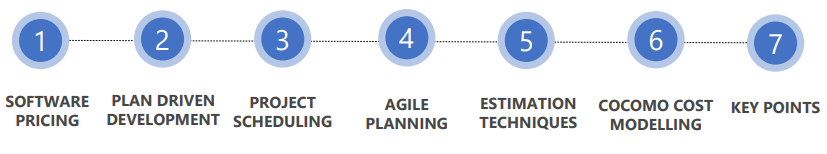
* Pessoas no grupo
  + É preciso um mix de pessoas num projeto de grupo.
* Organização do grupo
  + Um grupo deve organizado de forma a que os individuais contribuem para o melhor das suas habilidades e tarefas
* Comunicação técnica e de gestão
  + Boas comunicações entre os membros do grupo e entre a equipa de engenheiros de software.

**SELECIONAR OS MEMBROS DO GRUPO**

* O gestor ou líder da equipa têm como função criar um grupo coeso e organizar o seu grupo para que possam trabalhar juntos com eficiência
* Envolve criar um grupo com um balanço a nível técnico e de personalidades.

**PONTOS CHAVE**

* Uma boa gestão do projeto é essencial se os projetos de engenharia de software forem desenvolvidos no Schedule e dentro de orçamento
* A gestão de software é distinta de gestão de engenharia. Software é intangível.
* O risco de gestão envolve identificar os riscos do projeto e estabelecer a probabilidade que eles podem ocorrer e as consequências para o projeto se esse risco aparecer. Devemos fazer planos para evitar, gerir e lidar com estes riscos
* Gestão de pessoal envolve escolher as pessoas corretas para trabalhar no projeto e organizar a equipa
* As pessoas são motivadas pela interação com as outras pessoas, o reconhecimento da gestão e os seus pares, dando oportunidades para o seu desenvolvimento pessoal.



**Gestão de projetos de software**

* Planeamento de envolve dividir o trabalho em partes e distribuí-las por membros da equipa, antecipar os problemas que venham a aparecer e preparar soluções para estes problemas.
* O plano do projeto, que é criado no inicio do projeto, é usado para comunicar como é que o trabalho vai ser feito à equipa do projeto e aos clientes, e a ajudar a avaliar o progresso no projeto.

**Estádios do planeamento**

* No estádio da proposta, quando estamos a criar um contrato para desenvolver ou fornecer um sistema de software
* Durante a fase de start-up do projeto, quando tens que planear quem é que vai trabalhar no projeto, como é que o projeto vai ser divido nos diferentes incrementos, que recursos vão ser alocados ao longo da empresa.
* Periodicamente durante o projeto, quando modificas o plano com a experiência leve que ganhas e informação de monitorizar o progresso do trabalho.

**PROPOSTA DE PLANEAMENTO**

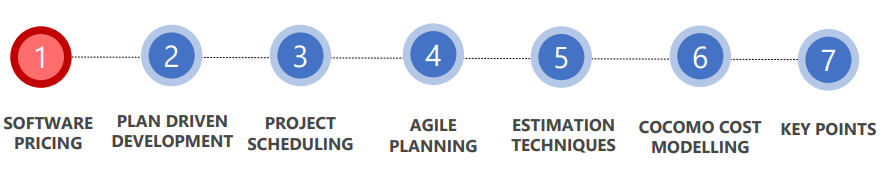
* Planear pode ser necessário só com requisitos de software outline
* O objetivo de planear neste estádio é fornecer informação que vai ser usada de forma a estabelecer um preço ao sistema
* O preço do projeto envolve estimar quanto é que um software vai custar para desenvolver, tomar como factores o custo do staff, custos de hardware, custos de software.

**PLANEAMENTO DE STARTUP DO PROJETO**

* Neste estádio, sabemos mais acerca dos requisitos do sistema, mas não temos informação de design ou implementação
* Criar um plano com detalhes suficiente de forma a tomar decisões acerca do budget projeto e staffing
* O plano de start-up deve definir a monitorização do projeto
* O plano de start-up ainda é preciso para desenvolvimento de agile para permitir que os requisitos sejam alocados no projeto.

**PLANEAR O DESENVOLVIMENTO**

* O plano do projeto deve ser alterado ao longo do progresso do projeto e sabemos mais acerca do software e o seu desenvolvimento
* A calendarização do projeto, custo-estimado e riscos devem ser revistos com regularidade



PREÇO DO SOFTWARE

* Estimativas são feitas para descobrir o custo do developer e de produção do software
  + Tomamos em conta o hardware, software, viagens, treino e custos de esforço
* Não existe uma simples relação entre o custo de desenvolvimento e o preço que é cobrado a um cliente
* Fatores organizacionais, económicos, políticos e de negócio influenciam o preço cobrado.

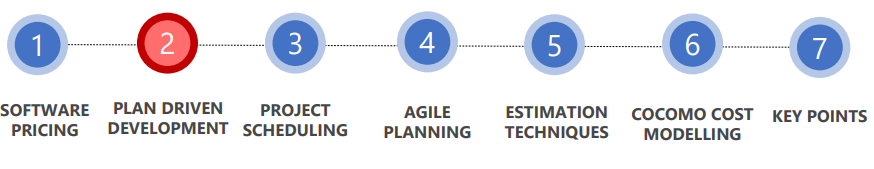
FACTORES QUE AFETAM O PREÇO DO SOFTWARE

* Oportunidade de mercado
* Incerteza do custo estimado
* Termos de contrato
* Volatilidade dos requisitos
* Saúde Financeira

**ESTRATÉGIAS DE PREÇO**

* **Abaixo do preço**
  + A empresa pode meter um sistema abaixo de preço de forma a ganhar um contrato que permita reter staff para oportunidades futuras.
  + Uma empresa pode meter um sistema de forma a ganhar acesso a novos mercados
* **Aumentar o preço**
  + O preço pode aumentar quando um comprador deseja um contrato com preço fixo e então o vendedor aumenta o preço para evitar riscos inesperados.

**PREÇO PARA GANHAR**

* O software ganha um preço de acordo com o que o developer acredita que o cliente está disposto a pagar.
* Se é menos do que o custo do desenvolvimento, a funcionalidade do software pode ser reduzida de acordo com uma funcionalidade a ser adicionada no futuro
* Custos adicionais podem ser adicionados ao longo da mudança dos requisitos e estes podem levar o preço a um nível mais elevado para compensar o défice do preço original

**PLAN DRIVEN DEVELOPMENT**

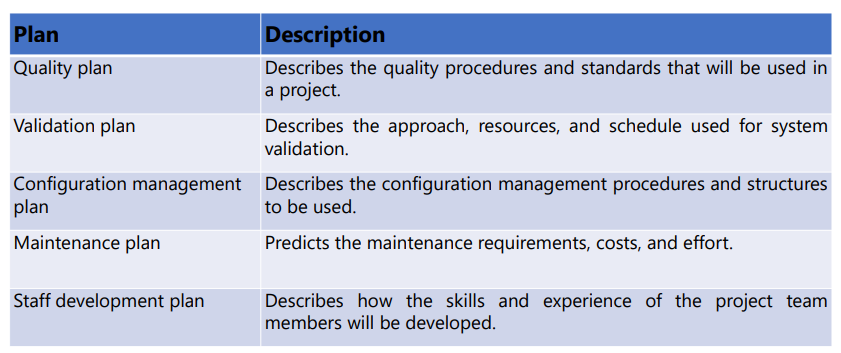
* Plan driven or plan-based development é uma abordagem à engenharia de software onde o desenvolvimento do processo é planeado ao detalhe.
* O plano do projeto é criado de forma a guardar o trabalho a ser feito, quem o vai fazer, o calendário de desenvolvimento e os produtos de trabalho
* Os gestores usam o plano para fazer suporte à decisão e uma maneira de medir o progresso.

**PLAN DRIVEN DEVELOPMENT PROS AND CONS**

* Os argumentos a favor de uma abordagem plan-driven são os problemas iniciais de planeamento (disponibilidade de staff, outros projetos etc) que devem ser tomados em conta, e os potenciais problemas e dependências que são descobertos antes do inicio do projeto.
* O principal argumento contra um desenvolvimento plan-driven é que muitas decisões ao inicio devem ser revistas devido às mudanças ao ambiente em que o software está a ser desenvolvido e usado

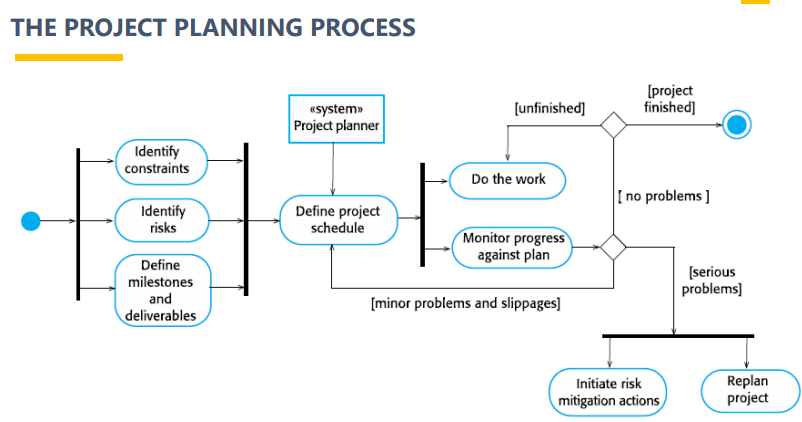
**CLASSIFICAÇÃO DO RISCO**

* Num projeto plan-driven, o plano do projeto define os requisitos disponíveis no projeto, um calendário e o trabalho a ser feito
* Secções do plano
  + Introdução
  + Organização do projeto
  + Análise de risco
  + Requisitos de hardware e software
  + Trabalho a ser feito
  + Calendarização
  + Mecanismos de monitorização e de reporting



**PROCESSO DE PLANEAMENTO**

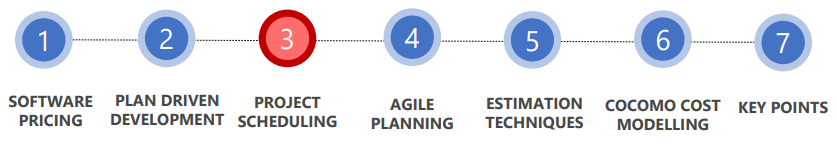
* Planeamento do projeto é um processo interativo que começa quando criamos o processo inicial durante a fase de start-up do projeto
* Mudanças no plano são inevitáveis
  + À medida que mais informação acerca do sistema e da equipa do projeto fica disponível, mais regularmente deve ser revisto o plano que reflete nos requisitos, calendário e mudanças de risco
  + Mudanças nos objetivos do negócio levam a mudanças nos planos do projeto.



**SUPOSIÇÕES DO PLANEAMENTO**

* Devemos fazer suposições realistas em vez de otimistas
* Problemas de qualquer tipo aparecem durante o projeto, e podem levar a atrasos
* As suposições iniciais e calendarização devem tomar em conta problemas inesperados em conta
* Devemos incluir contingência no plano de forma a que, se as coisas derem errado, a nossa data de entrega não seja seriamente perturbada

**RISCO DE MITIGAÇÃO**

* Se há problemas sérios com o desenvolvimento é muito provável que haja atrasos significativos, é necessário arriscar ações de mitigação para reduzir os falhanços do projeto
* Na conjunção destas ações, também devemos replanear o projeto
* Isto pode envolver renegociar as restrições e entregáveis do projeto com o cliente.

**CALENDARIZAÇÃO DO PROJETO**

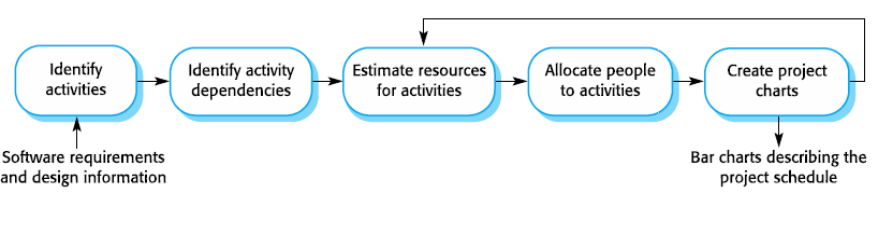
* Calendarização do projeto é o processo de decidir como é que o trabalho no projeto vai ser organizado em tarefas separadas, e quando e como vão ser executadas
* Estimamos o tempo necessário de calendário completando cada tarefa, o esforço necessário e quem vai trabalhar nas tarefas identificadas.
* Também temos de estimar os recursos necessário para completar cada tarefa.

**ATIVIDADES DE CALENDARIZAÇÃO DO PROJETO**

* Dividir o projeto em tarefas e estimar o tempo e recursos necessários para completar cada tarefa.
* Organizar as tarefas concorrentemente de forma a otimizar os trabalhadores.
* Minimizar as dependências das tarefas para evitar atrasos
* Dependências nos gestores do projeto intuição e experiência

**METAS E ENTREGÁVEIS**

* Metas são pontos na calendarização contra qual conseguimos avaliar o progresso
* Entregáveis são produtos do trabalho que são entregues ao cliente.



**PROBLEMAS DA CALENDARIZAÇÃO**

* Estimar a dificuldade dos problemas e o custo de desenvolver uma solução é difícil
* Produtividade não é proporcional ao número de pessoas a trabalhar numa tarefa
* Adicionar pessoas a um projeto tardio, faz com que seja mais tarde devido à comunicação das despesas gerais.
* O inesperado acontece sempre. Permitir sempre contingência no planeamento

**REPRESENTAÇÃO DA CALENDARIZAÇÃO**

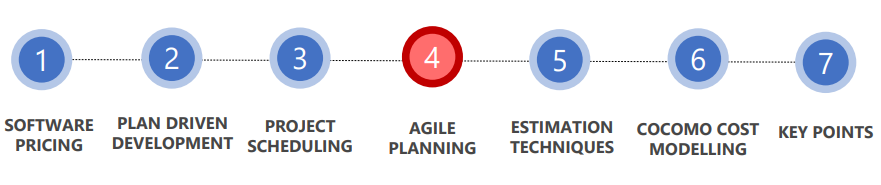
* Notações gráficas são normalmente usadas para ilustrar a calendarização do projeto
* Mostram o projeto divido em tarefas. Tarefas devem ser pequenas. Devem durar cerca de uma a duas semanas
* Gráficos de barras são os mais comuns para a representação da calendarização do projeto

**ATIVIDADES DO PROJETO**

* Tarefas são o elemento básico do planeamento
* Cada atividade tem
  + Duração num calendário em dias ou meses
  + Um esforço estimado, que mostra o número de dias ou meses necessários para completar o trabalho
  + Um prazo limite na qual a tarefa deve estar terminada
  + Um fim definido, que pode ser um documento.

**METAS E ENTREGÁVEIS**

* **METAS**
  + São os pontos no calendário contra qual podemos avaliar o progresso, por exemplo, a entrega do sistema para teste
* **ENTREGÁVEIS**
  + São produtos que são entregues ao cliente.



**PLANEAMENTO AGILE**

* Métodos agile de desenvolvimento de software são abordagens iterativas onde o software desenvolvido é entregue aos clientes em incrementos
* Fora as abordagens plan-driven, a funcionalidade destes incrementos não é planeada em avanço, mas é decida durante o desenvolvimento
  + A decisão no que incluir num incremento depende do progresso e as prioridades do cliente
* As prioridades do cliente e requisitos mudam portanto faz sentido haver um plano flexível que acomoda todas estas mudanças.

**ESTÁDIOS DO PLANEAMENTO AGILE**

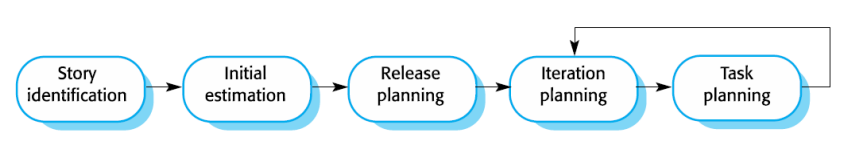
* Release planning, que olha em frente nos meses que se seguem e decide que características devem ser incluídas no lançamento do sistema
* Planeamento de Iteração, que tem um período mais curto e foca-se em planear o próximo incremento do sistema.

**ABORDAGENS AO PLANEAMENTO AGILE**

* Planear em scrum
* Baseado em gerir o backlog do projeto
* O jogo do planeamento
  + Desenvolvido originalmente como parte do Extreme Programming
  + Dependente das user stories como forma de medida do progresso do projeto.

**PLANEAMENTO STORY-BASED**

* O jogo de planeamento é baseado em user stories que refletem nas características que devem ser incluídas no sistema
* A equipa do projeto lê e discute as histórias e organiza-as por ordem de tempo que acham que vai demorar para a implementar
* Histórias recebem “pontos de esforço” refletindo o seu tamanho e dificuldade de implementação
* O número de pontos de esforço implementados por dia é medido dando uma estimativa da velocidade da equipa
* Isto permite que o total de esforço necessário para implementar um sistema possa ser estimado



**PLANEAMENTO DE LANÇAMENTO E ITERAÇÃO**

* Planeamento de lançamento envolve selecionar e refinar as histórias que vão refletir as características a ser implementadas no lançamento de um sistema e a ordem na qual as histórias vão ser implementadas.
* As histórias a ser implementadas em cada iteração são escolhidas, baseadas no número de histórias refletindo no tempo que demora em cada iteração

**ALOCAÇÃO DAS TAREFAS**

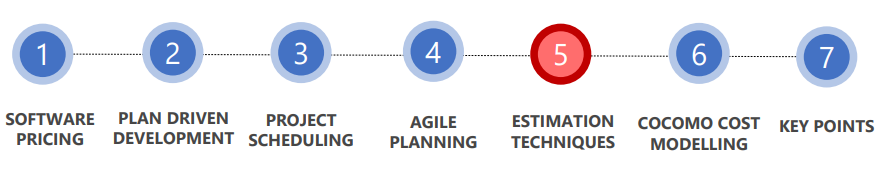
* Durante o estádio de planeamento de tarefas, os developers dividem as histórias em desenvolvimento de tarefas
  + Uma tarefa de desenvolvimento deve demorar entre 4-16 horas
  + Todas as tarefas que devem ser completadas para implementar todas as histórias
  + Developers escolhem as tarefas especificas que querem implementar.
* Benefícios desta abordagem
  + A equipa recebe uma visão geral das tarefas a serem completadas na iteração
  + Developers têm um senso de ownership nestas tarefas e é capaz de os motivar para completar a tarefa.

**ENTREGA DE SOFTWARE**

* O incremento do software é entregue no final de cada iteração do projeto
* Se as características a ser incluídas no incremento não podem ser completadas no tempo, o tempo de trabalho é reduzido
* A calendarização da entrega nunca é estendida

**DIFICULDADES COM PLANEAMENTO AGILE**

* Planeamento Agile é dependente da disponibilidade e envolvimento do cliente
* Pode ser difícil de organizar, pois os representantes dos clientes podem prioritarizar outros trabalhos e não estar disponíveis para o jogo de planeamento
* Mais tarde, alguns clientes podem ser familiares os planos tradicionais do projeto e pode ser difícil haver empenho no processo de planeamento agile.



**TÉCNICAS DE ESTIMATIVA**

* Organizações têm que fazer um esforço a nível de software e estimativa de custo. Há 2 tipos de técnicas e pode ser usado para isto:
  + **Técnicas baseadas de experiência –** A estimativa do esforço futuro dos requisitos é baseado na experiência do gestor de projetos passados e domínio de aplicação.
  + **Custo de algoritmo de modelação**

**ABORDAGENS BASEADAS EM EXPERIÊNCIA**

* Experiência baseada em técnicas caem sobre julgamentos baseados em experiência de projetos passados e o esforço é gasto nestes projetos nas atividades de desenvolvimento de software
* Tipicamente, identificamos os entregáveis a ser produzidos num projeto e os diferentes componentes de software ou sistemas que vão ser desenvolvidos
* Documentados numa spreadsheet, estimamos individualmente e computamos o esforço total necessário
* Normalmente ajuda ter um grupo de pessoas envolvidas no esforço e perguntar a cada membro do grupo para explicar a sua estimativa.

**PROBLEMAS COM ABORDAGENS BASEADAS EM EXPERIÊNCIA**

* A dificuldade com técnicas baseadas em experiência é que novos projetos de software podem não ter muito em comum com projetos anteriores
* Desenvolvimento de software muda rapidamente, e o projeto pode usar técnicas não-familiares como web-services, sistemas de aplicação configuração ou HTML.
* Se ainda não se trabalhou estas técnicas, a experiência antiga pode não ajudar a estimar o esforço necessário, fazendo com que seja mais difícil de produzir custos accurate e estimativas de calendário.

**MODELAÇÃO CUSTO DE ALGORITMO**

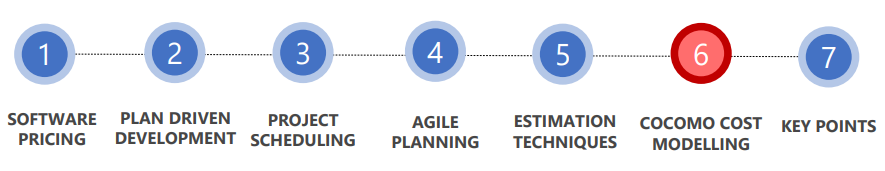
* Custo é estimado como uma função matemática de um produto, projeto e atributos do processo cujos valores são estimados pelo gestores do projeto
  + Effort = A´SizeB´M
  + A é uma organização-dependente constante, B reflete o esforço desproporcionado para projetos maiores e M é o multiplier que reflete o produto, processo e atributos das pessoas.

**ACCURACY DA ESTIMATIVA**

* O tamanho do sistema de software apenas é certo quando estiver acabado
* Alguns fatores que influenciam o tamanho final são
  + Uso de COTS e componentes
  + Linguagem de programação
  + Distribuição do sistema
* À medida que o progresso do processo avança o tamanho estimado fica mais certo
* The estimates of the factors contributing to B and M are subjective and vary according to the judgment of the estimator.

**EFICIÊNCIA DOS MODELOS DE ALGORITMO**

* O custo dos modelos de algoritmo são uma maneira sistemática de fazer uma estimativa do esforço necessário para desenvolver um sistema. Porém, estes modelos são complexos e difíceis de usar.
* Há muitos atributos e scopes consideráveis para a incerteza na estimativa de valores
* Esta complexidade significa que a aplicação prática do custo do algoritmo foi limitada a um número grande de empresas.



**CUSTO DE MODELAÇÃO COCOMO**

* Um modelo empírico baseado na experiência do projeto.
* Modelo bem documento e independente que não está ligado a um vendedor especifico.
* Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2.
* COCOMO 2 takes into account different approaches to software development, reuse, etc.