Métodos de Desenvolvimento de Software

↳ Geriz um projecto: para geriz um projecto significa planificaz, executaz e controlaz o processo.

↳ planeaz um projecto baseia-se:

qual é o trabalho

- qual é o trabalho que deve sez realigado (escopo) e quais os seus componentes de atividade correspondentes (work breakdown structure, aka WBS)
- quem vai executaz e geriz o trabalho a sez feito (matriz de responsabilidade)
- quando é que o trabalho vai sez feito (calendário)
- o custo do trabalho, materiais, etc

↳ executaz um projecto baseia-se:

- em realigaz o trabalho que é necessário fagez, de acordo com o que foi planeado
- manter as equipas e gezentes informados

↳ controlaz um projecto baseia-se:

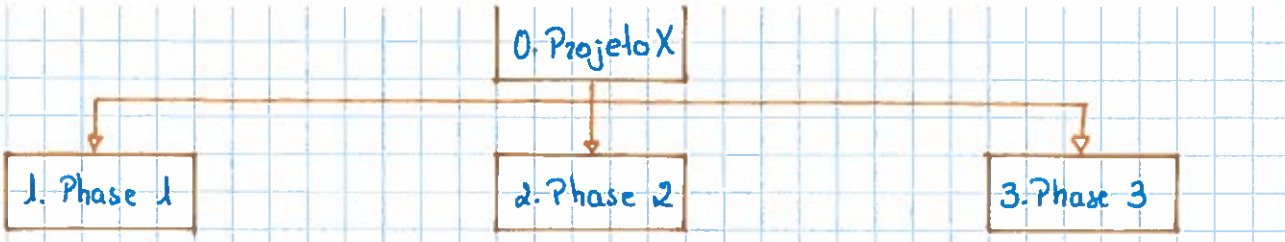
- monitorigaz e reportaz a execução do plano de gestão: scope, tempo, custo, como também qualidade e risco.
- tez o maior objetivo em manter a performance e os seus resultados em "linha" com os planos iniciais, com alguma margem de erro.

↳ Identificação de Atividades

↳ Work Breakdown Structure (WBS)

- define o escopo/scope do projeto ("to do" list)
- divide o trabalho em componentes: subdivide tarefas complexas em umas mais simples
- como constroiz um WBS: (1) incluiz 100% do trabalho/derivados
  (2) evitaz sobrepor elementos
  (3) planeaz primeiro o resultado, não as ações
  (4) tentar arranjar "the sweet spot" de detalhe
- exemplo:

(página seguinte)



- 1.1. Process requirements
- 1.2. Data requirements
  - 1.2.1. Identify all data entities
  - 1.2.2. Load data modeling tool
- 1.3. Conceptual design

- 2.1. Facilities
- 2.2. Equipment
- 2.3. People
  - 2.3.1. Full time
  - 2.3.2. Part time
  - 2.3.3. Temporary

- 3.1. Location
  - 3.1.1. Peru
  - 3.1.2. China
  - 3.1.3. Canada
- 3.2. Surveys
- 3.3. New designs

- existem diferentes maneiras de decompor as estratégias, como: product oriented ou process oriented
- quanto detalhe? the 8/80 rule (ofthumb):
  - nenhum work package deve ser menos de 8h ou mais de 80
  - grupos de tarefas, ou atividades, após completos, deveriam corresponder à completude dos níveis acima correspondentes

Planeamento de Actividades

Gantt charts (bar charts)

- têm pouco detalhe sobre as precedências da tarefa/atividade
- têm relações de dependência:
  - Finish to Start (FS): assim que A acaba, B pode começar (recomendada como dependência default no início do planeamento) 

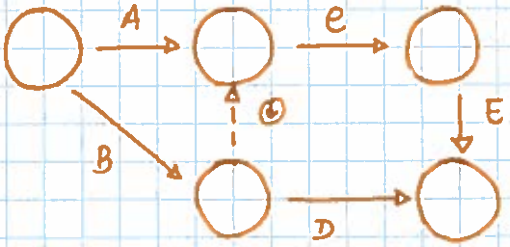
  - Start to Start (SS): assim que A começa, B pode começar
  - Start to Finish (SF): assim que A começar, B pode acabar
  - Finish to Finish (FF): assim que A acaba, B pode acabar
- project milestones: é um fim lógico para um stage no projecto, para progress reviewing. Está documentado, sumariza o trabalho feito, associado a uma tarefa ou grupo de tarefas. Representado por ◇

Pert diagrams:

- planeamento e controlo detalhado do projeto
- suporte para análise de project schedule
  - identificação do primeiro momento possível para completar uma atividade
  - suporte para a data de conclusão mais antiga (earliest)
  - comparação entre alternativas cenários de scheduling
  - controlo do planeamento do projecto

Activity On Arrow (AOA) Diagrams

- setas representam a execução de atividades
- nodos representam o fim (ou início) de atividades
- setas tracejadas representam atividades "fictícias" (dummmys) com tempo de execução nulo. Usadas para especificar pre-requisite relações, de maneira a preservar a integridade da network



@ a dummy arrow garante que e apenas começa depois de A e B estarem completas

- representação do tempo:
  - A → Atividade
  - T → tempo para concluir a atividade
  - EST → Earliest Start Time
  - EFT → Earliest Finish Time
  - LST → Latest Start Time
  - LFT → Latest Finish Time

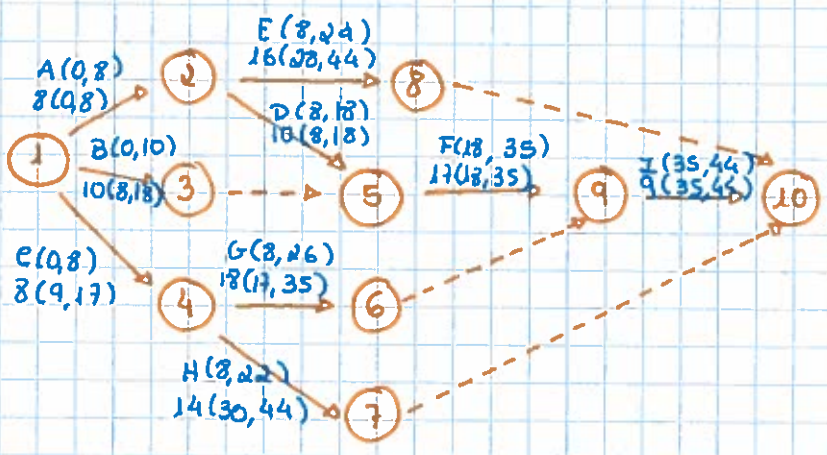
- algoritmo para a construção de AOA diagrams:

(1) identificar e listar todas as atividades
(2) atribuir a cada atividade um id único
(3) identificar e listar as dependencias entre atividades
(4) desenhar uma rede preliminar
(5) estimar a duração das atividades
(6) adicionar a duração das atividades
(7) calcular EST
(8) calcular LST
(9) fazer fine tune
(10) atribuir recursos

- caminho crítico: o caminho mais longo formado por atividades em que EST = LST. É o caminho que terá um impacto direto no projeto

↳ exemplo:

Act	Pred	Duz
A	—	8
B	—	10
C	—	8
D	A	10
E	A	16
F	D, B	17
G	C	18
H	C	14
I	F, G	9



caminho crítico: A → D → F → I = 8 + 10 + 17 + 9 = 44

↳ Activity On Node Diagrams

↳ cada atividade é representada por um nodo

ES	Duration	EF
Task Name		
LS	Slack	LF

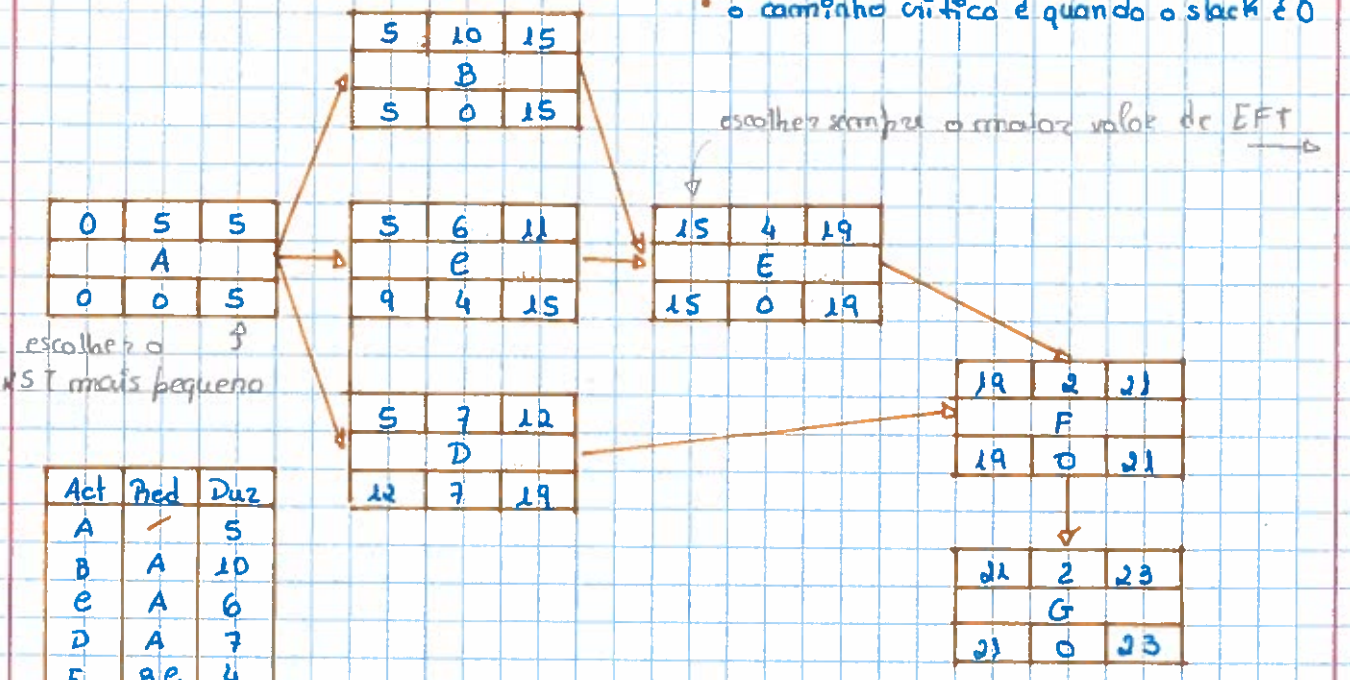
- ES → Early Start
- LS → Late Start
- EF → Early Finish
- LF → Late Finish
- duration = EF - ES
- slack = LF - EF (5)

↳ as dependencias são representadas por setas

↳ exemplo:

- o caminho crítico é quando o slack é 0

escolher sempre o maior valor de EFT →



Act	Pred	Duz
A	—	5
B	A	10
C	A	6
D	A	7
E	B, C	4
F	D, E	2
G	F	2

→ Earned Value Management (EVM)

↳ permite identificar:
- onde estão a ocorrer os problemas
- caso os problemas existentes sejam críticos
- o que vai ser necessário para o "put the project on-track?

↳ o verdadeiro trabalho executado é comparado com o que foi planeado com os custos associados. Conseguimos observar:
- schedule variance (desvio do prazo)
- cost variance (desvio do custo)
- schedule performance (índice de desempenho do prazo)
- cost performance (índice de desempenho do custo)

↳ existem elementos básicos de um EVM:

- planned value (PV): o custo-total do trabalho planeado a partir da data de relato (how much work should have been done?)
- actual cost (AC): o custo-total necessário para completar o trabalho até à data de relato (how much did the work cost)
- earned value (EV): o custo-total do trabalho completado/feito até à data de relato (how much work was executed)
- budget at completion (BAC): what's the project's total cost
- time at completion (TAC): what's the project's duration

↳ usa-se o earned value para prever os desvios:
- o valor do trabalho (até à data (EV)) é comparado com o custo desse trabalho
- o valor do trabalho performed até à data (EV) é comparado com o trabalho que devia ter finalizado até ao momento → (PV)

↳ schedule variance: diferença entre os custos orçamentos do trabalho realizado e o planeado.

- $SV = EV - PV$, → $sv > 0$, o projeto está a avançar conforme o plano
  → $sv < 0$, o projeto está a

↳ cost variance: diferença entre os custos orçamentados do trabalho realizado e o seu custo real.

- $CV = EV - AC$, → $cv > 0$, o projeto está sobreestimado
  → $cv < 0$, o projeto está superestimado

↳ schedule performance index (SPI): relação entre o trabalho planeado (EV) e o valor do trabalho planeado (PV).

- $spi = EV/PV$

↳ cost performance index (CPI): relação entre o trabalho planeado realizado (EV) e o valor real desse trabalho.

- CPI = EV/AC , pode ser visto como uma medição de produtividade

↳ interpretar o CPI e SPI:

CPI	
<1	cost overplanned
=1	cost according to plan
>1	cost below planned
SPI	
<1	delayed
=1	on time
>1	advanced

↳ O processo do Software

↳ atividades fundamentais do processo:

(1) Especificação do software
- definir funcionalidades e restrições

(2) desenvolvimento do software
- produzir software

(3) verificação e validação do software
- faz o que o cliente quer
- corresponde às especificações

(4) evolução do software
- de maneira a corresponder às necessidades do cliente

↳ especificação do software

- perceber e definir que serviços são requisitados do sistema e identificar as restrições para a operação e desenvolvimento do sistema.

- conduz a um documento de requisitos que é uma especificação do sistema para:
  - clientes e usuários finais (end-users) que recebem high level statements
  - os system developers recebem a especificação detalhada do sistema

- atividades principais:
  - requisitos de elicitação e análise (poderá desenvolver modelos de sistema e protótipos)
  - requirements specification (cria a documentos de requisitos do usário e sistema)
  - requirements validation (verifica o realismo, consistência e completude)

↳ design e implementação

- desenvolver um sistema executável para entregar ao cliente
- architectural design: identifica em geral a estrutura do sistema (componentes, relações e distribuição)
- interface design: interfaces entre componentes
- component selection and design: encontrar componentes reutilizáveis ou design new ones (detalhes para o programador)
- database design: design das estruturas de dados

↳ validation e verification (V & V)

- verificação: estamos a construir o projeto/sistema corretamente?
  - olhar para as especificações do sistema
- validação: estamos a construir o sistema correto?
  - está a par das expetativas do cliente
- stages:
  - teste dos componentes (programmer)
  - teste do sistema (integração)
  - teste de aceitação
- se for comercializado, às vezes o teste beta é feito (entregar o sistema a um grupo controlado de usuários - relatar problemas aos developers)

↳ evolução do software

- decorre durante todo o processo
- o software é continuamente alterado de maneira a corresponder às necessidades do cliente

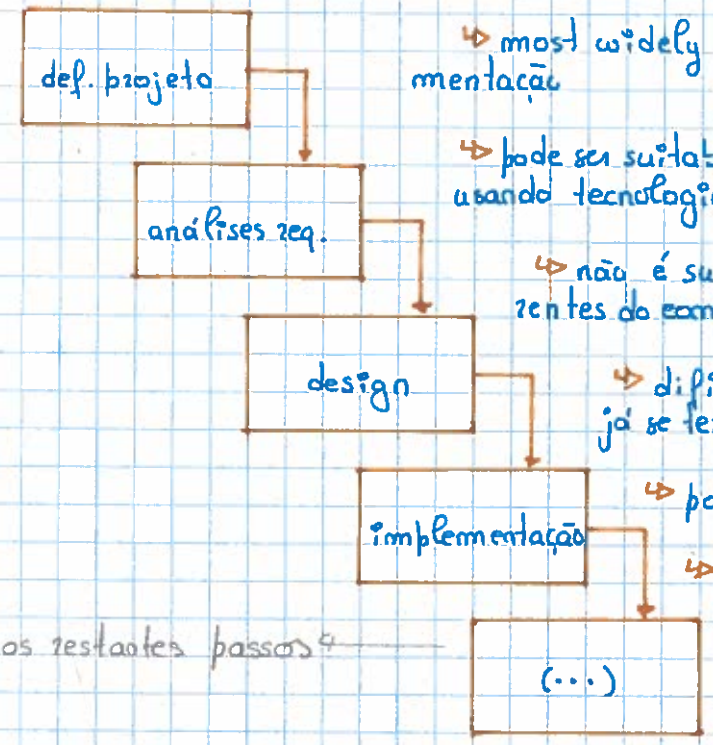
↳ características de um bom processo:

- compreensibilidade
- visibilidade
- suportabilidade
- aceitabilidade
- confiabilidade
- robustez
- manutenção
- rapidez

↳ passos para um processo genérico:

- definição do projeto
- análises requiridas
- design
- implementação do program
- component testing
- testing de integração
- teste do sistema
- entrega do sistema
- manutenção: corretiva, adaptativa, perfetiva

Waterfall Model (Plan driven)



↳ most widely used; cada passo resulta em documentação

↳ pode ser suitable para developments bem-compreendidos usando tecnologia familiar

↳ não é suitable para brand new e sistemas diferentes do comum por causa da incerteza

↳ difícil de acomodar a mudanças depois de já se ter começado o processo

↳ pode acomodar iteração, mas indiretamente

↳ versão que funciona apenas disponível muito no fim do processo

↳ encontra muitos blocking states

↳ however, reflete o tipo de processo modelo usado in other engineering approaches

↳ é ainda usado quando o software (projeto de) é parte de um larger system engineering project

↳ adequado para sistemas embutidos, críticos e grandes

↳ não adequado para comunicação informal em equipa, requisitos mudam rapidamente

↳ variante: formal system development
especificação

- modelo matemática de uma especificação do sistema é criado:
  - o modelo é refined usando transformações matemáticas a código executável

- adequado para sistemas críticos: sistemas com forte segurança, reabilidade ou requisitos de segurança (físicos)?

- problemas com o custo to de uma especificação formal

Rapid Application Development (RAD)

- parecido com o modelo waterfall mas usa um ciclo de desenvolvimento curto (60 a 90 dias para a completude)
- usa component-based construction e dá ênfase ao re-uso e generation do código
- usa múltiplicas equipas on scalable projets
- requere recursos pesados
- requer que os developers e clientes que estejam heavily committed
- problemas:
  - para grandes projetos requer muitos human resources para criar o número necessário de equipas
  - se o sistema não consegue ser modelizado, construir os modelos necessários para o modelo RAD será problemático
  - se high performance é um problema, requerer the tune das diferentes componentes do sistema pode não funcionar
  - pode não ser apropriado quando technical risks are high

Incremental Development

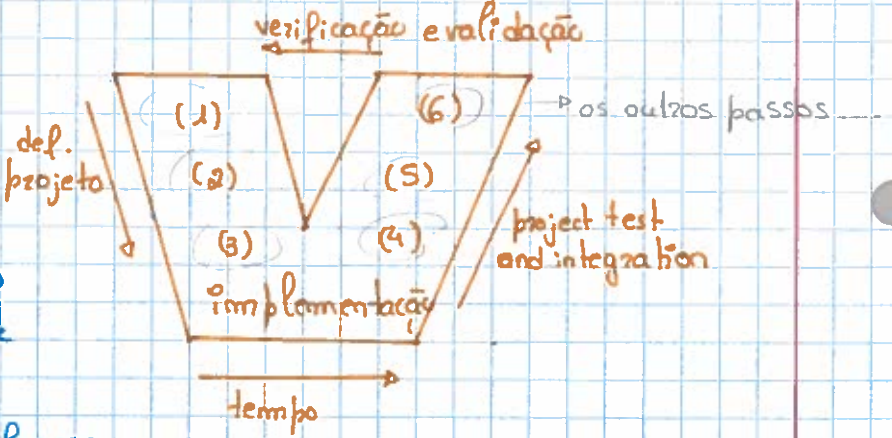
- aplica uma filosofia iterativa para ter um early feedback com several versions
- divide a funcionalidade do sistema em incrementos e usa uma sequência linear de desenvolvimento em cada incremento
- o primeiro incremento entregue é normalmente the core product (i.e. só funcionalidades básicas)
- revisões de cada incremento causam impacto no design de incrementos futuros
- lida bem com riscos
- pode ser implementado com waterfall
  - para tal é necessário planear os incrementos com antecedência
- extreme programming (XP) e outros métodos ágeis são incremental:
  - mas não implementam the waterfall model na ordem standard
  - os requisitos só são especificados antes de cada iteração
- vantagens:
  - reduz a quantidade de software a ser developed
  - reduz custos e riscos
  - fast delivery
- desvantagens:
  - requer compromissos que podem levar a um sistema que não corresponde às necessidades do cliente
  - o controlo sob a evolução perde-se

Incremental Process Model

- ideal quando o staff está indisponível para uma implementação completa
- incrementos podem ser planeados para gerir riscos técnicos
- positivo:
  - o custo de implementação é reduzido
  - mais fácil ter feedback do cliente
  - early delivery e deployment of useful software
- negativo:
  - o processo não é visível (not cos effective produzir documentação em todas as iterações)
  - as estruturas do sistema tendem a degradar
  - pode colidir com burocracia da organização

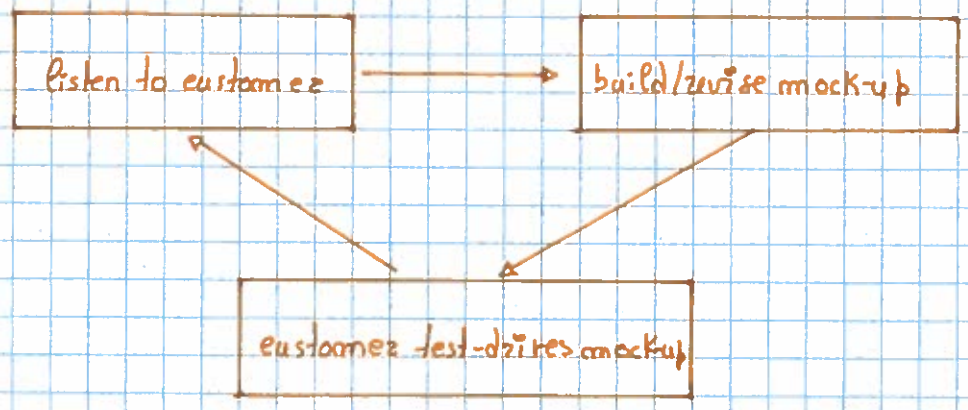
V-Model

- pros:
  - minimiza os riscos do projeto, devido a concerns explícitos (verificação, validação)
  - melhora e garante a qualidade
  - redução do custo total over the entire project e system life cycle
- cons: sofre os mesmos problemas que o modelo waterfall



Prototyping

- especificar requisitos é muitas vezes difícil
- os usuários muitas vezes não sabem o que querem até o verem
- prototyping involve construir um mock-up do sistema e usá-lo para ter feedback



- idealmente os mock-ups servem como mecanismos para identificar requisitos
- os usuários gostam do método, pois têm um feeling do sistema

↳ o menos ideal são os básicos para o produto completo:
- protótipos customam ignorar qualidade/performance/maintenance issues
- podem criar pressure from users para uma entrega rápida
- may use a less-than-ideal platform

↳ o processo não é visível (mais uma vez não é cost effective criar documentação para todas as versões)

↳ os sistemas acabam poorly structured (incorporar alterações torna-se cada vez mais caro e difícil)

↳ não é adequado para grandes, complex long-lived systems com equipas diferentes a desenvolver partes diferentes

↳ difícil de estabelecer uma arquitetura do sistema estável

↳ devia ser misturado com waterfall: evolutionary approaches para incertezas e waterfall para partes bem estabelecidas

↳ The Spiral Model

↳ ciclos de desenvolvimento through multiple (3-6) task regions
- customer communication
- planning
- risk analysis
- engineering
- construction and release
- customer evaluation

↳ incremental releases
- early releases may be paper or prototypes
- later releases become more complicated

↳ modelar o software until já não ser usado

↳ bom:
- não é à prova de bala, mas é considerado um dos melhores approaches
- é um approach realístico
- pode usar prototyping em qualquer fase de evolução do produto

↳ mau:
- requer excellent management and risk assessment skills