41951- ANÁLISE DE SISTEMAS

Metodologias de desenvolvimento, processos de software e OpenUP.

Ilídio Oliveira | v2022-11-18

universidade de aveiro
departamento de eletrónica, telecomunicações e informática
deti

Objetivos de aprendizagem

- Identificar atividades comuns a todos os projetos (ciclo de vida)
- Distinguir projetos sequenciais de projetos evolutivos
- Descrever a estrutura do Unified Process (fases, objetivos, iterações)
- Identificar as principais atividades exigidas na atribuição do projeto
- Mapear disciplinas técnicas nas fases do OpenUP

Fases do SDLC: um modelo geral para a engenharia de software

SDLC - Systems Development lifecycle (a.k.a. Software Development lifecycle)

Quatro fases fundamentais: planeamento, análise, desenho e implementação.

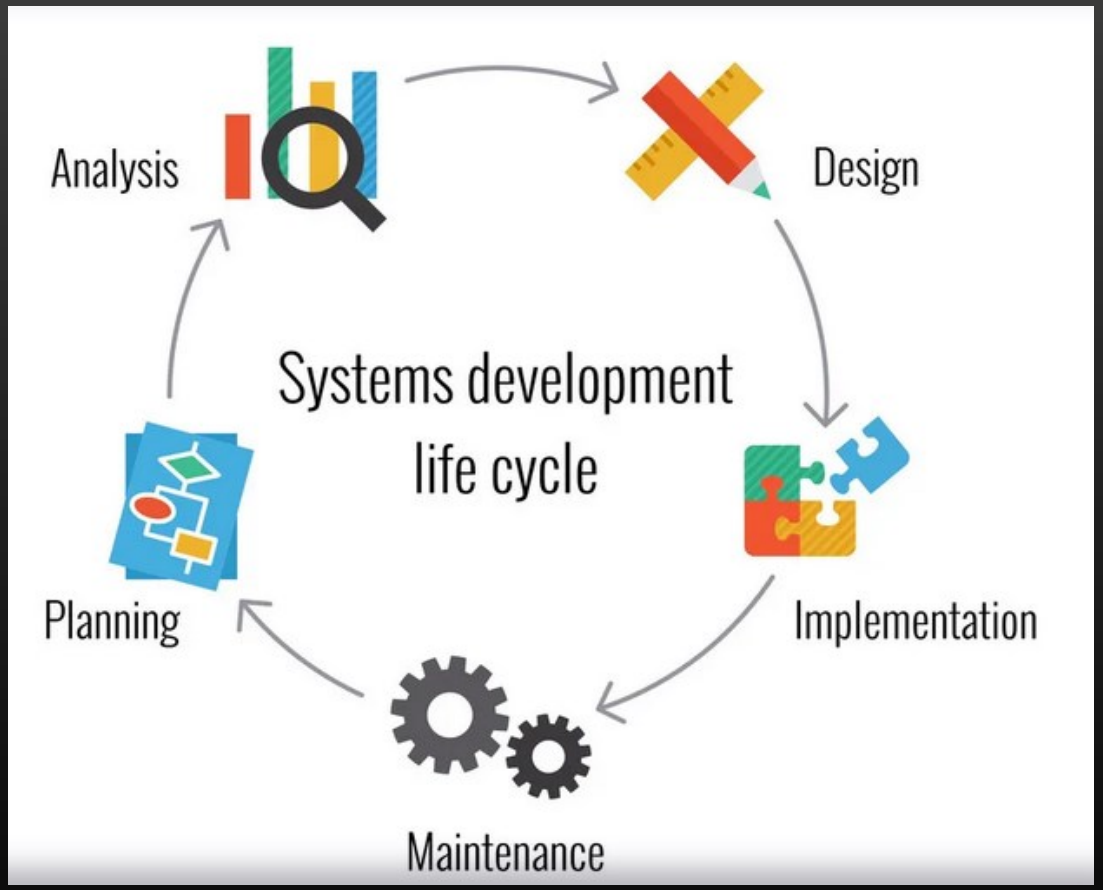
Diferentes projetos podem enfatizar diferentes partes do SDLC, mas todos os projetos têm elementos destas quatro fases.

Cada fase é composta por uma série de atividades, em que aplica disciplinas técnicas para produzir resultados previstos

Alguns autores representam a fase de manutenção

Ideia de evolução do produto

→ fazer crescer o produto de software, como uma entidade "viva"



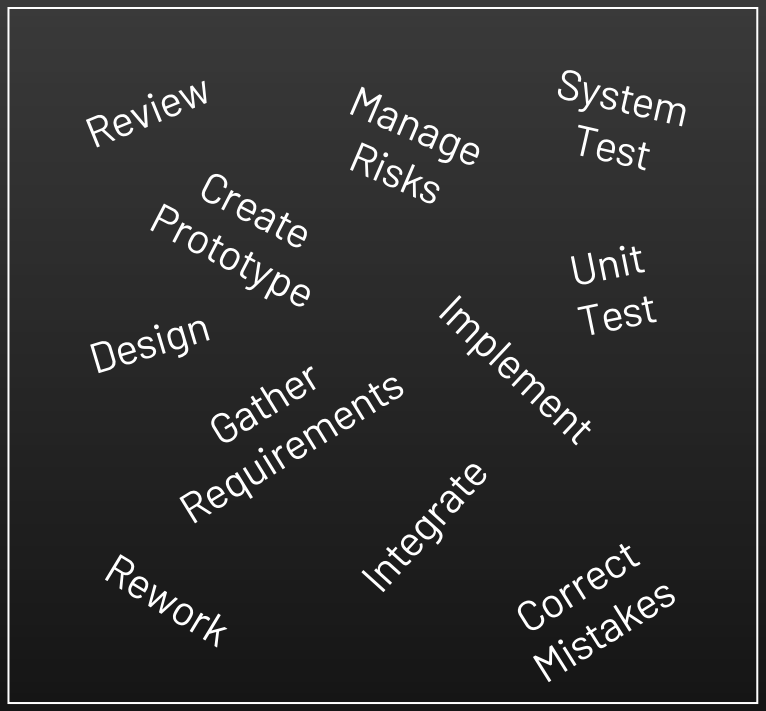
E.g.: Hoffer et al, "Modern Systems Analysis and Design", 5th ed.

Table 1-2 Products of SDLC Phases

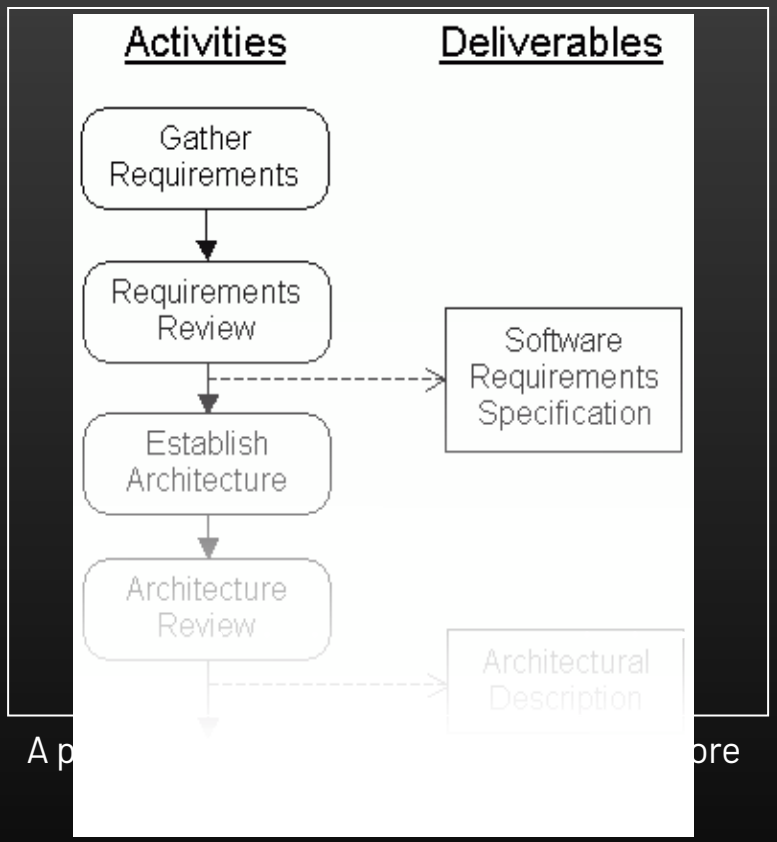
Phase	*Products, Outputs, or Deliverables*
Planning	Priorities for systems and projects; an architecture for data, networks, and selection hardware, and IS management are the result of associated systems;
	Detailed steps, or work plan, for project;
	Specification of system scope and planning and high-level system requirements or features;
	Assignment of team members and other resources;
	System justification or business case
Analysis	Description of current system and where problems or opportunities are with a general recommendation on how to fix, enhance, or replace current system;
	Explanation of alternative systems and justification for chosen alternative
Design	Functional, detailed specifications of all system elements (data, processes, inputs, and outputs);
	Technical, detailed specifications of all system elements (programs, files, network, system software, etc.);
	Acquisition plan for new technology
Implementation	Code, documentation, training procedures, and support capabilities
Maintenance	New versions or releases of software with associated updates to documentation, training, and support

Credit: Hoffer et al, "Modern Systems Analysis and Design", 5th ed.

Como aplicar o SDLC de forma sistemática?



Developing software without a defined process is chaotic and inefficient



A p ore

"It is better not to proceed at all, than to proceed without method." -- Descartes

Metodologias de desenvolvimento (para projetos de software)

Modelo para as atividades, papéis e resultados do processo de desenvolvimento.

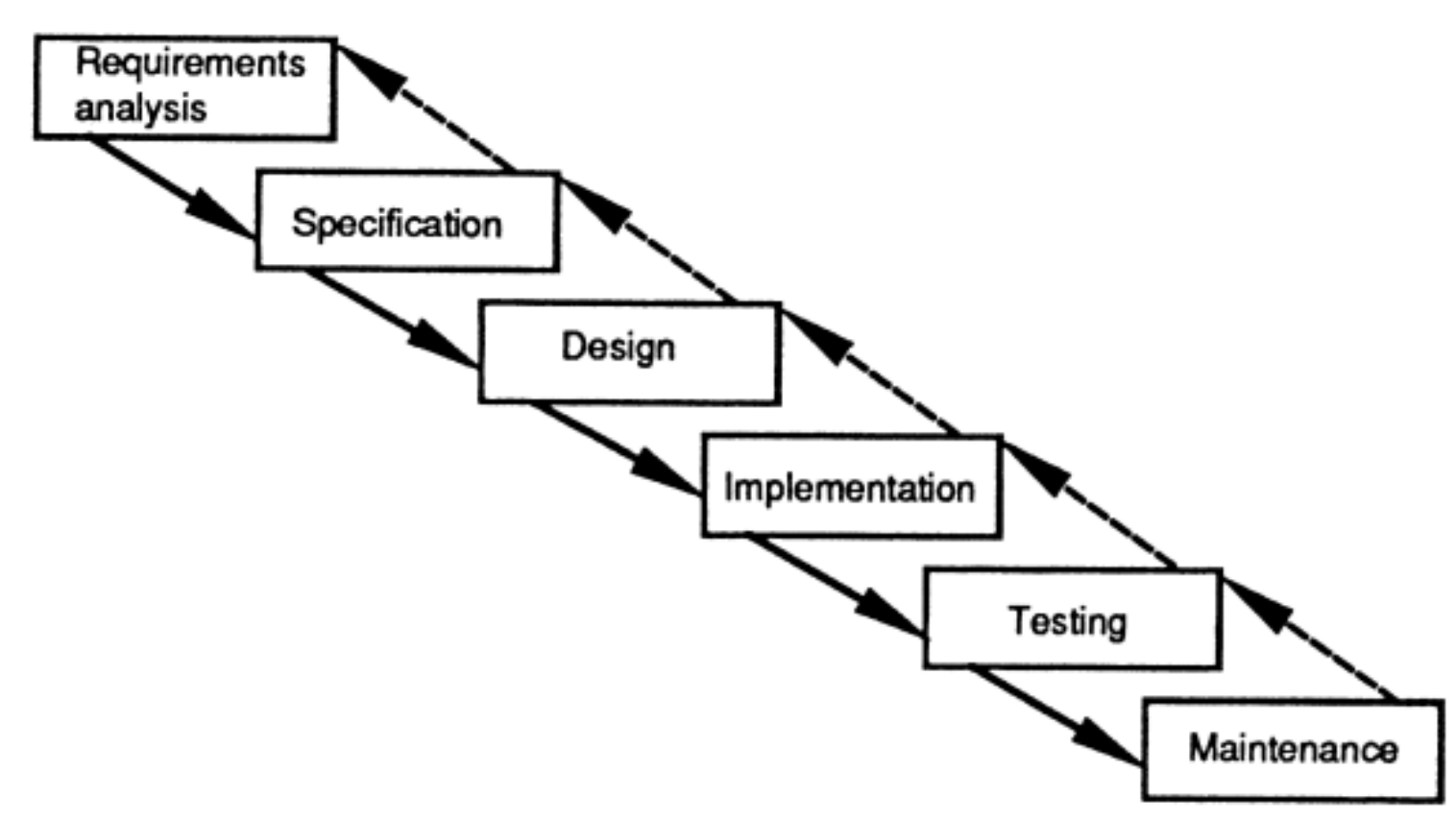
Metedologias para o desenvolvimento de sistemas – visão geral Dennis *et al*

Metodologia	Exemplo
Estruturada	• Waterfall • Desenvolvimento paralelo
Rappid Aplication Development (RAD)	• Prototyping • Desenvolvimento faseado
Object-Oriented Systems Analysis and Design (OOSAD)	• Unified Process
Agile Developement	• Extreme Programming • "SCRUM"
Personalizadas/à-medida	

Metedologias para o desenvolvimento: uma divisão quanto ao encadeamento das atividades

Tipo de meetodologia	Exemplo
Linear/estruturada	• Waterfall
Evolutiva	• Prototyping • Spiral model • Métodos ágeis

“Classical” engineering approach: Waterfall model



W. Royce, “Managing the Development of Large Software Systems,” *Proc. Westcon*, IEEE CS Press, 1970, pp. 328-339.

Waterfall model advantages

Simple and easy to understand and use.

Easy to plan

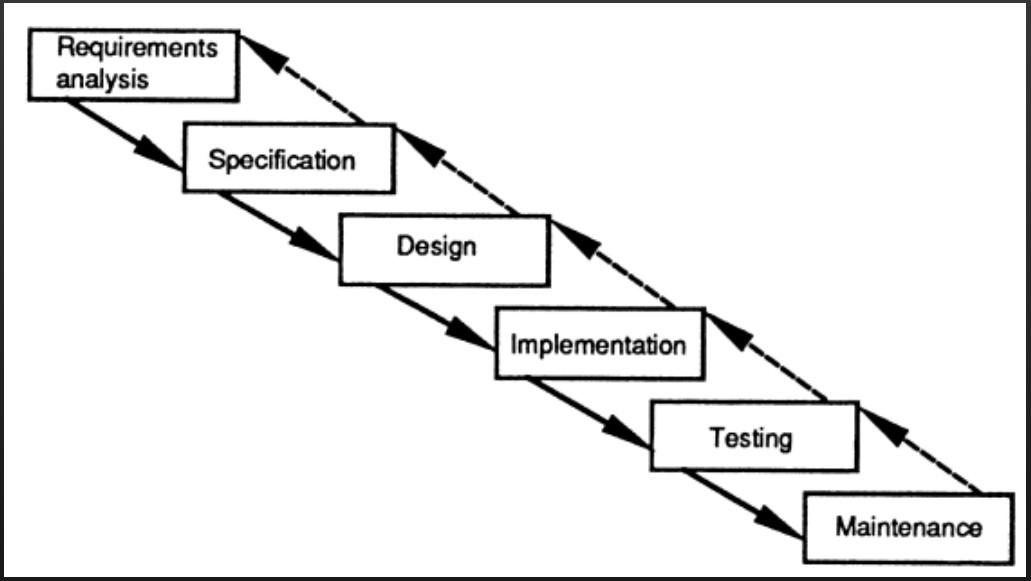
A schedule can be set with deadlines for each stage of development and a product can proceed through the development process like a car in a car-wash, and theoretically, be delivered on time.

Easy to manage

each phase has specific deliverables and a review process.

Phases are processed and completed one at a time.

Works well where requirements are stable and well understood



Waterfall model disadvantages

Problems

Difficulty of accommodating change after the process is underway.

Poor model for long and ongoing projects.

No working software is produced until late during the life cycle.

Not suitable for the projects where requirements are uncertain or at the risk of changing.

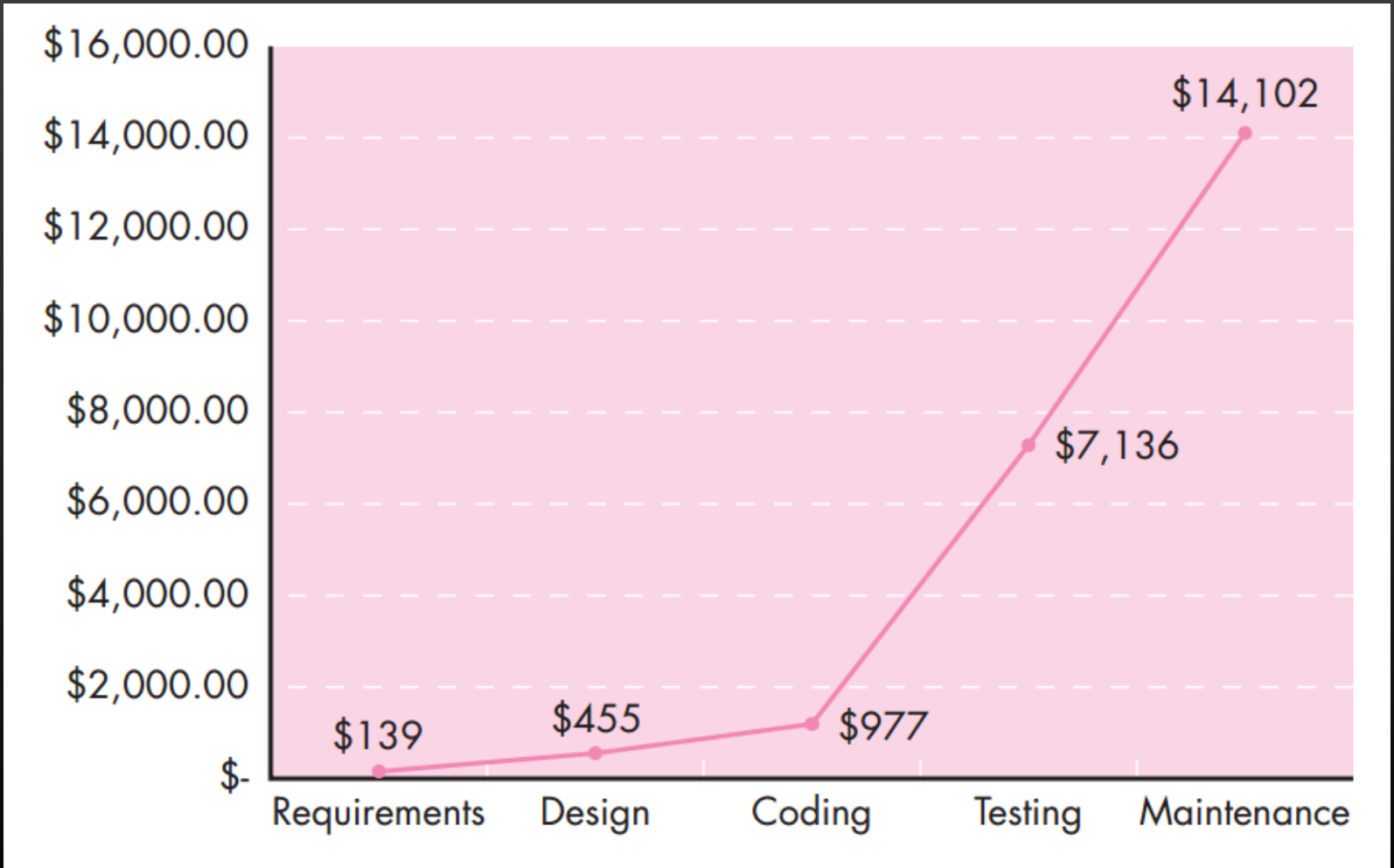
Why it may fail?

Real projects rarely follow the sequential flow that the model proposes

It is often difficult for the customer to state all requirements explicitly

The customer must have patience. A working version of the program(s) will not be available until late in the project time span

The cost of correcting an error raises **exponentially** along the sw lifecycle

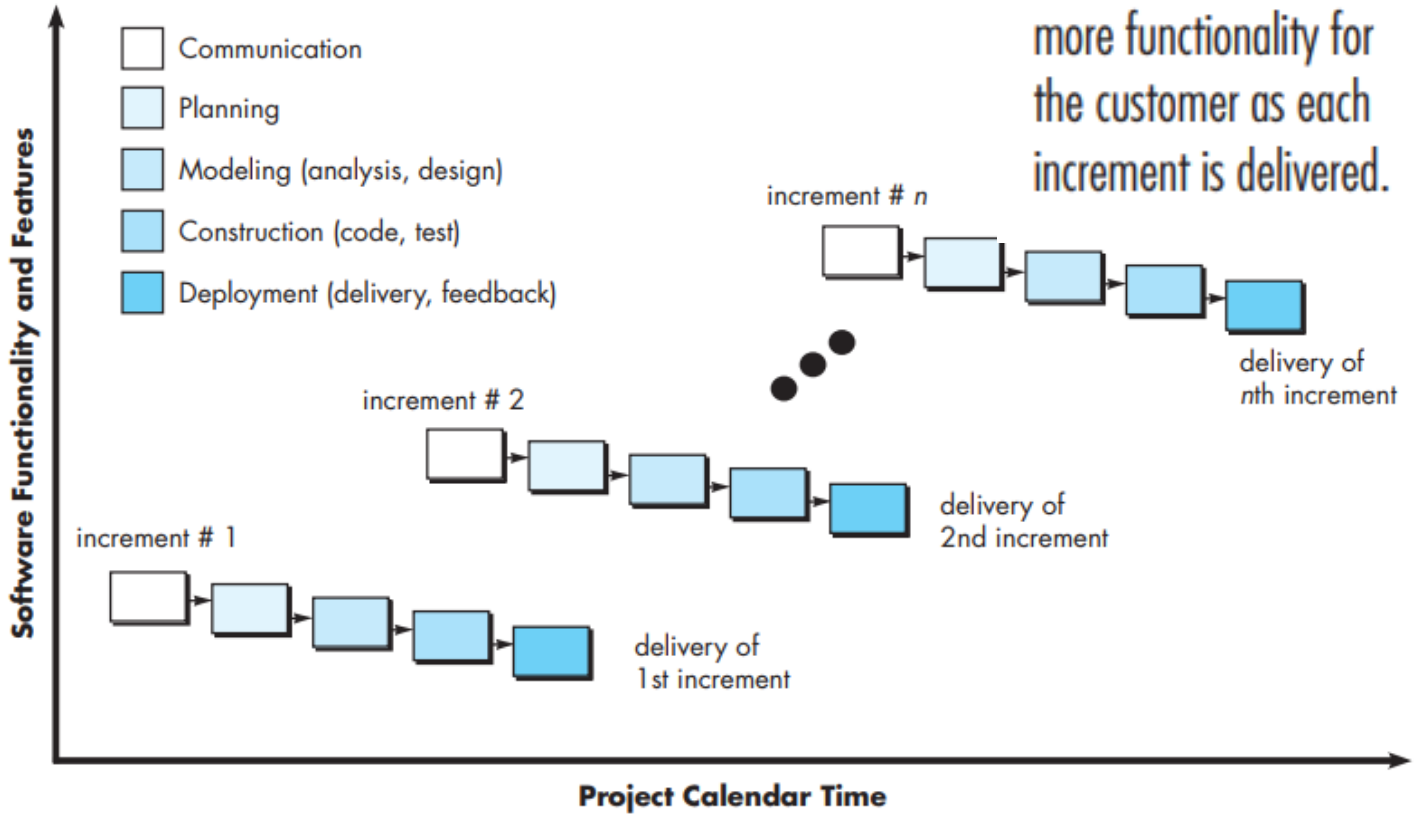


Boehm, B., and V. Basili, "Software Defect Reduction Top 10 List," IEEE Computer, vol. 34, no. 1, January 2001, pp. 135–137. http://doi.ieeecomputersociety.org/10.1109/2.962984

The incremental model delivers a series of releases, called increments, that provide progressively more functionality for the customer as each increment is delivered.

FIGURE 4.3

The incremental model



Phased & incremental

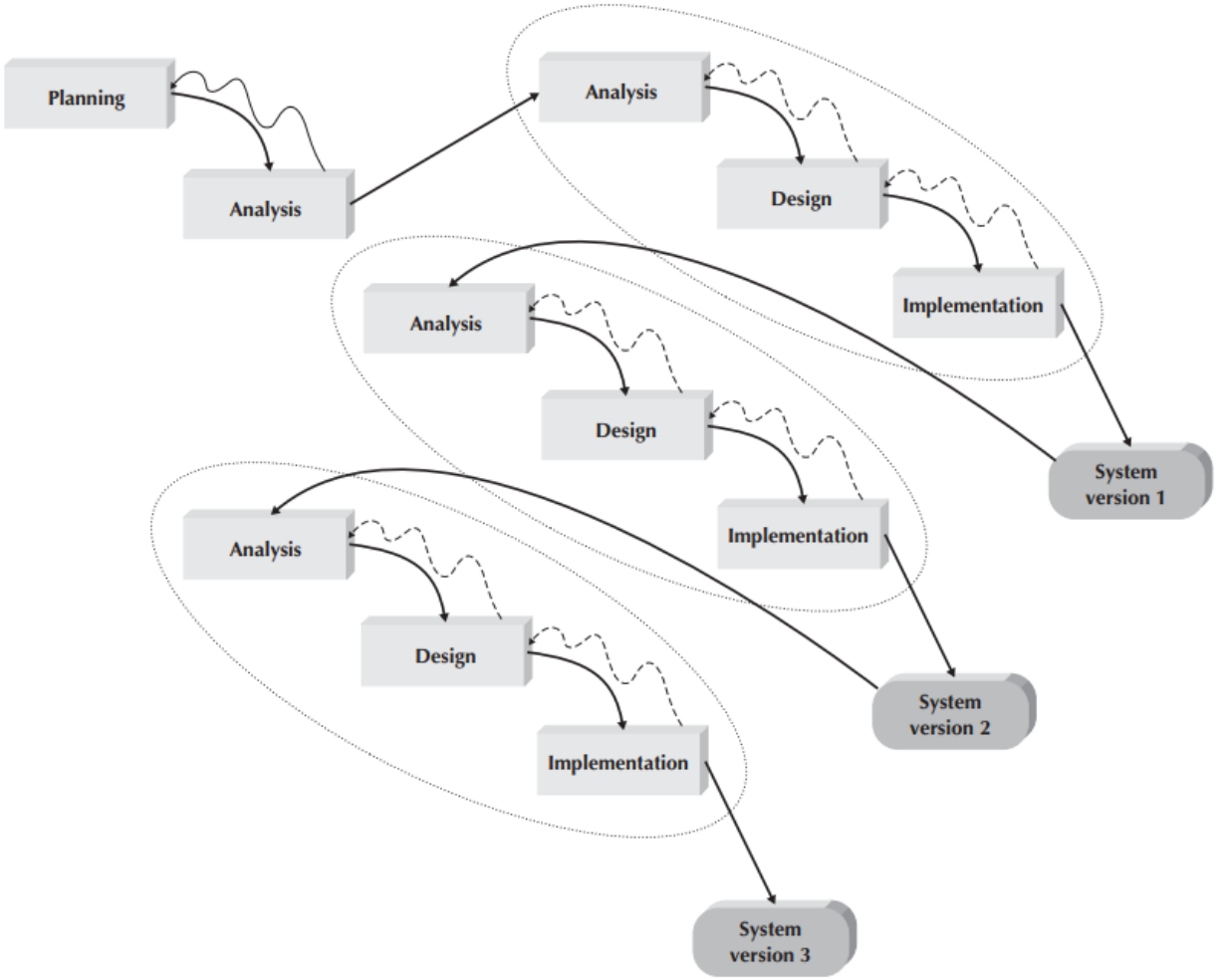
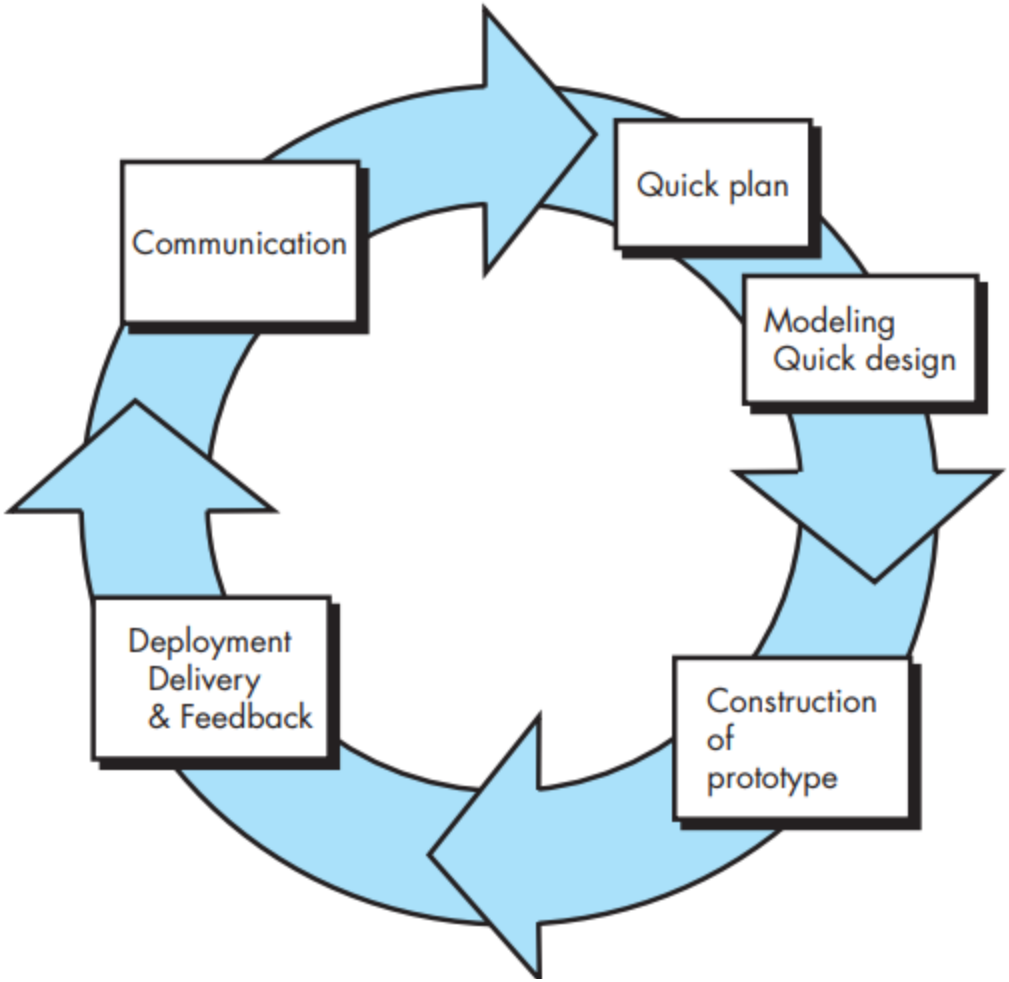


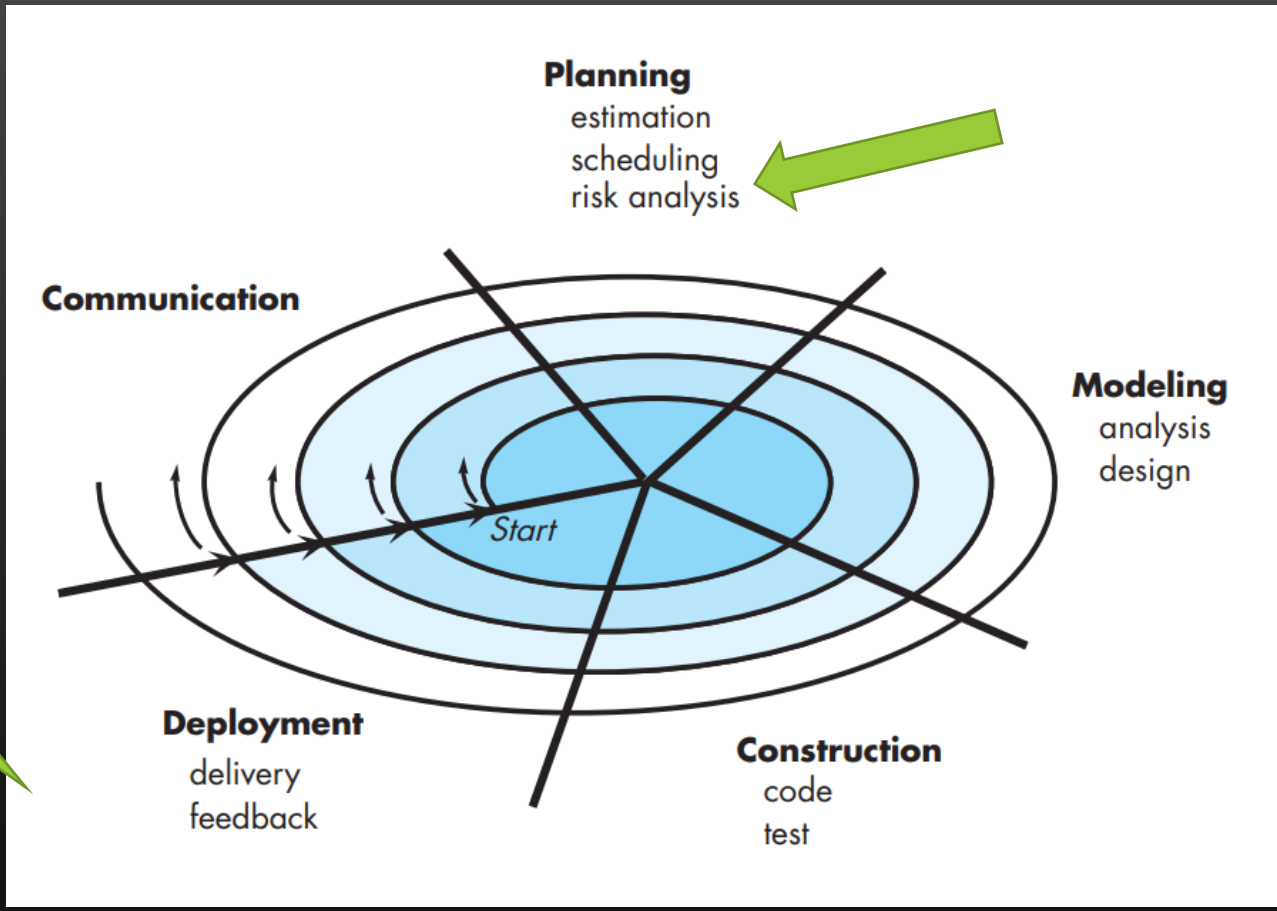
FIGURE 1-4 A Phased Development-Based Methodology

Evolutionary: protyping



Although problems can occur, prototyping can be an effective paradigm for software engineering. The key is to define the rules of the game at the beginning; that is, all stakeholders should agree that the prototype is built to serve as a mechanism for defining requirements. It is then discarded (at least in part), and the actual software is engineered with an eye toward quality.

Evolutionary: spiral model



Deployments don't need to be working software

Using the spiral model, software is developed in a series of evolutionary releases. During early iterations, the release might be a model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.

Linear or evolutionary processes?

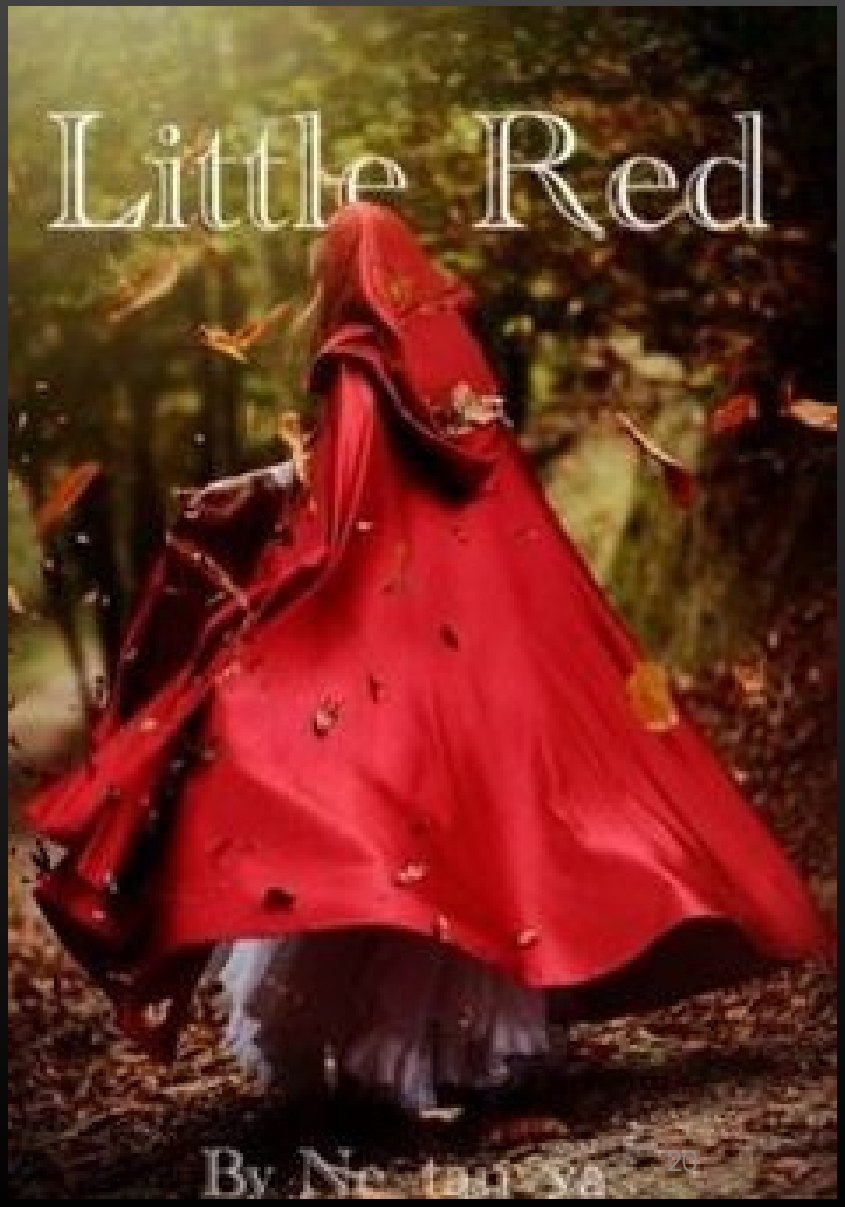
Plan-driven/linear processes

all of the process activities are planned in advance and progress is measured against this plan.

Evolutionary processes

planning is incremental and it is easier to adapt to changing customer requirements.

https://bit.ly/3DGbEsa



O Unified Process

Software process

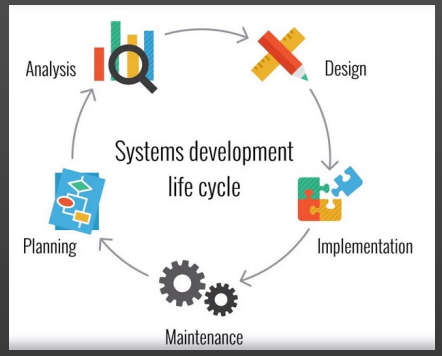
O SDLC é concretizado usando um processo de software sistemático.

Um processo de software é um guião para as atividades e tarefas que são necessárias para construir software de qualidade.

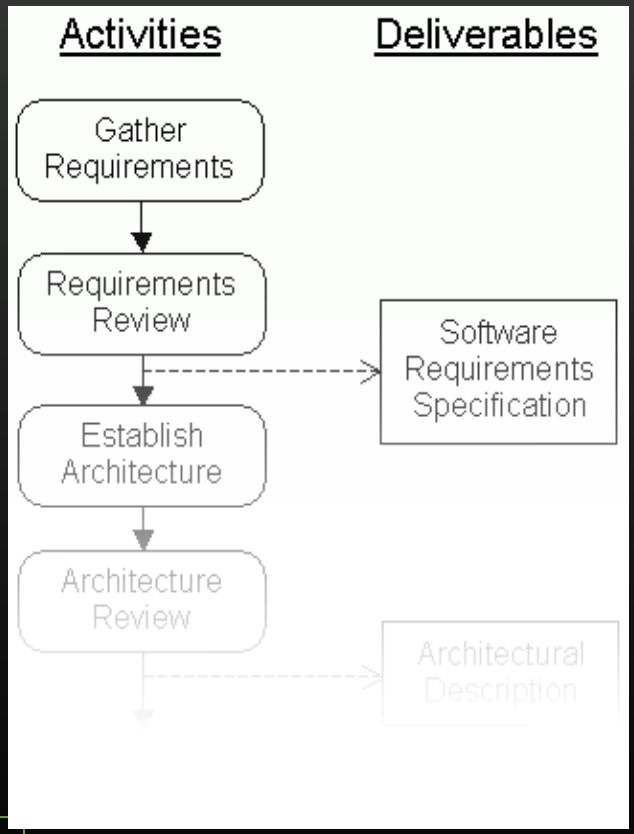
Por que precisamos de um processo explícito?

- As falhas ocorrem frequentemente
- Criar sistemas complexos não é intuitivo
- Os projetos apresentam problemas de implementação (terminados fora do prazo, acima do orçamento ou entregues com menos funcionalidades do que o previsto)

SDLC (visão geral do desenvolvimento)



Processo (dá o "guião")



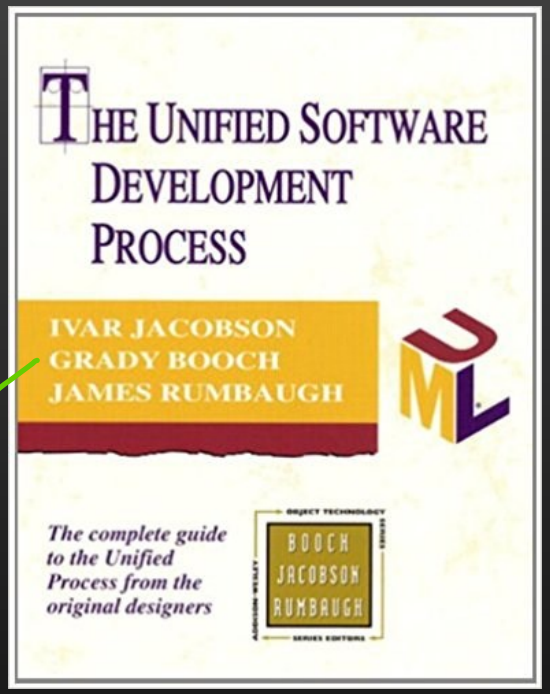
Credit: Dennis et al, "*Systems Analysis and Design: An Object Oriented Approach with UML*", 5th ed.

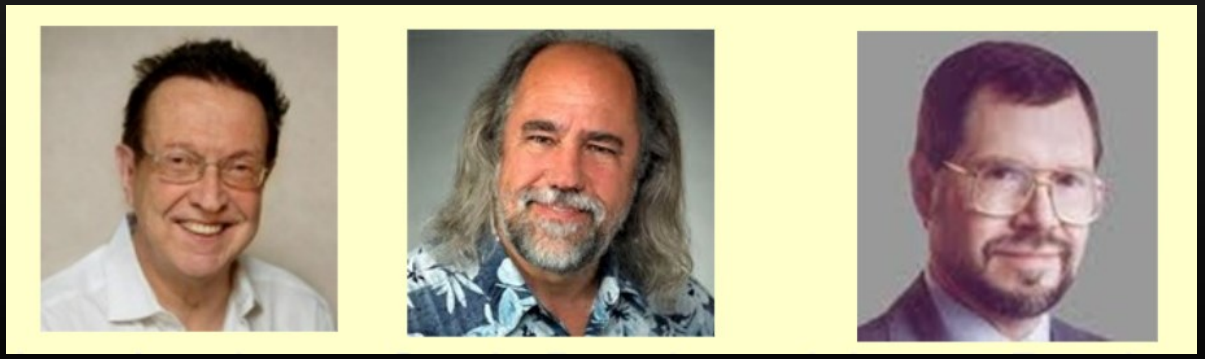
Unified Process/Open Unified Process

Uma tentativa de um processo de genérico

Pode ser adaptado para projetos concretos

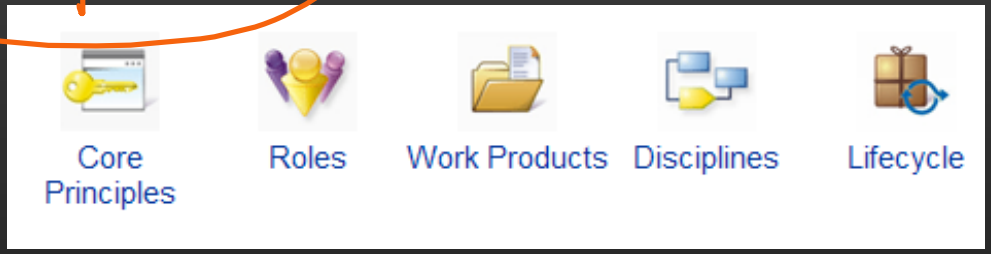
O OpenUP é uma versão “livre" do Unified Process





O SDLC é concretizado em processos de desenvolvimento

Open UP



Um processo especifica:

O quê?
Quem?
Como?
Quando?

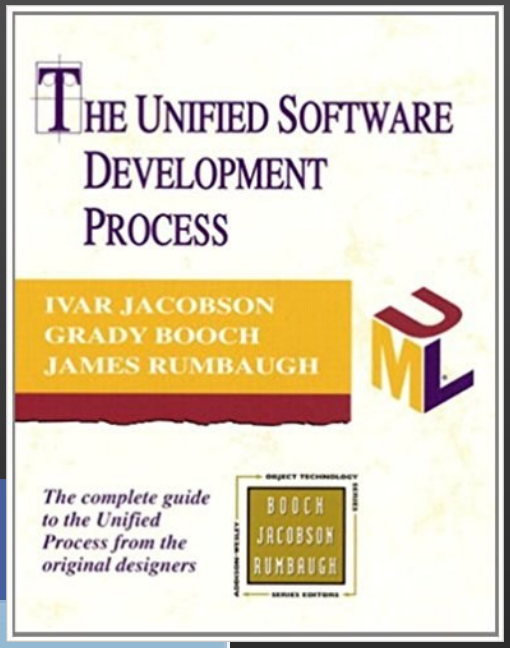
Um processo inclui:

Papéis
Fluxos de trabalho
Procedimentos
Modelos (dos resultados esperados)

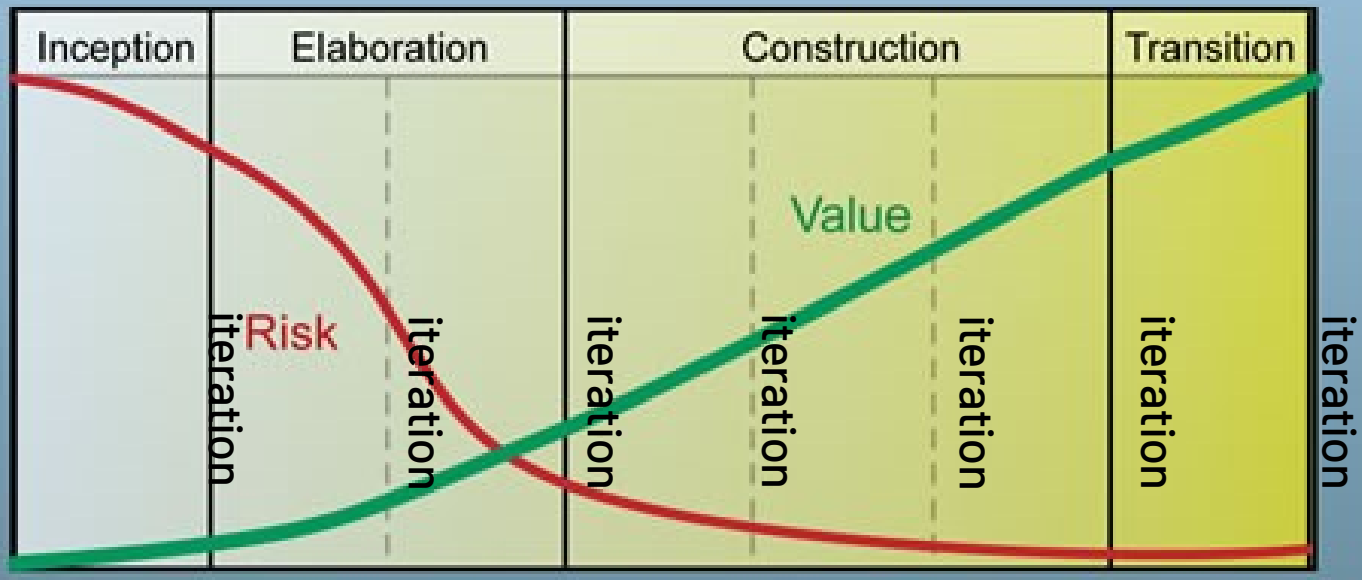
http://sweet.ua.pt/ico/OpenUp/OpenUP_v1514/

Qual é o melhor processo?

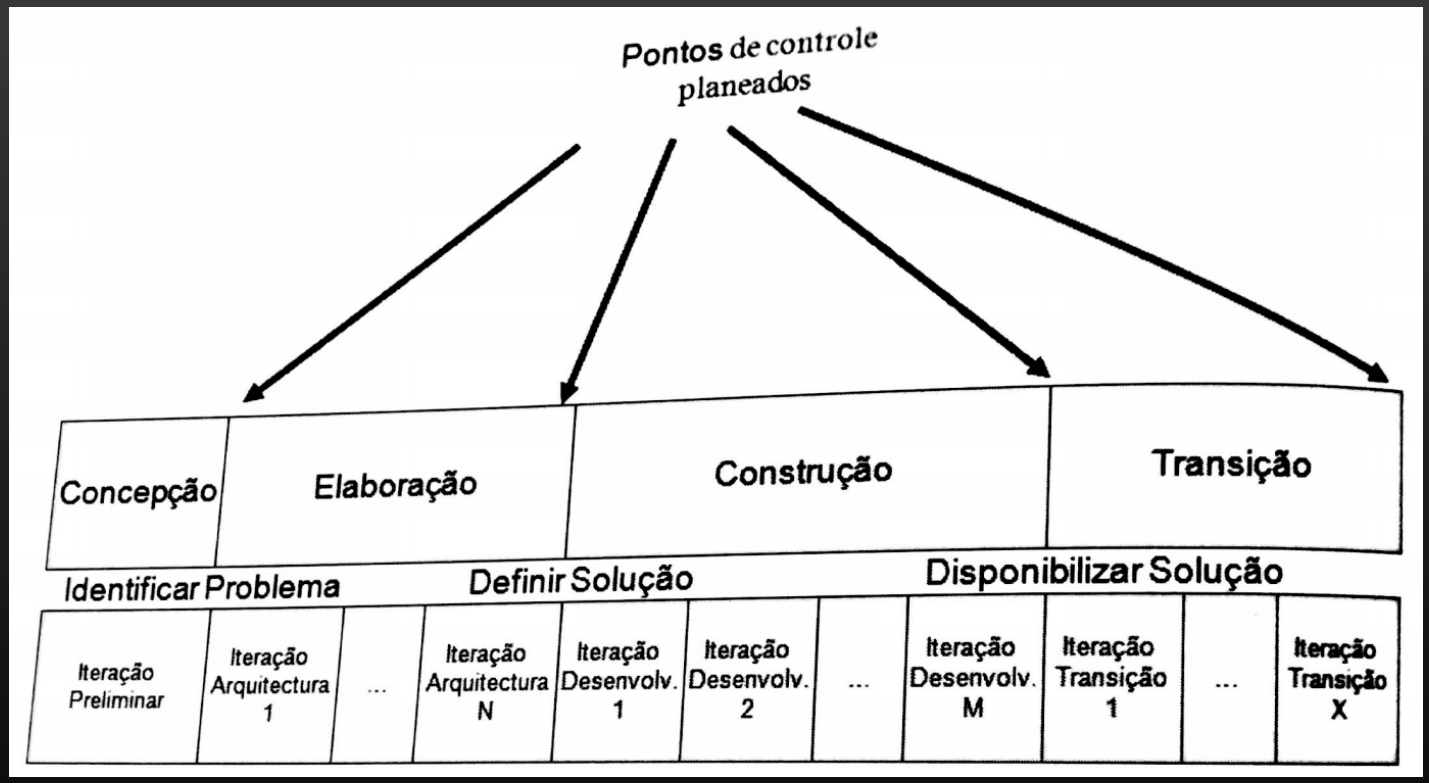
Não há um único "melhor processo"
As organizações devem seleccionar (ou personalizar) o seu processo.



Project Lifecycle



PT: Fases, iterações e pontos de controlo

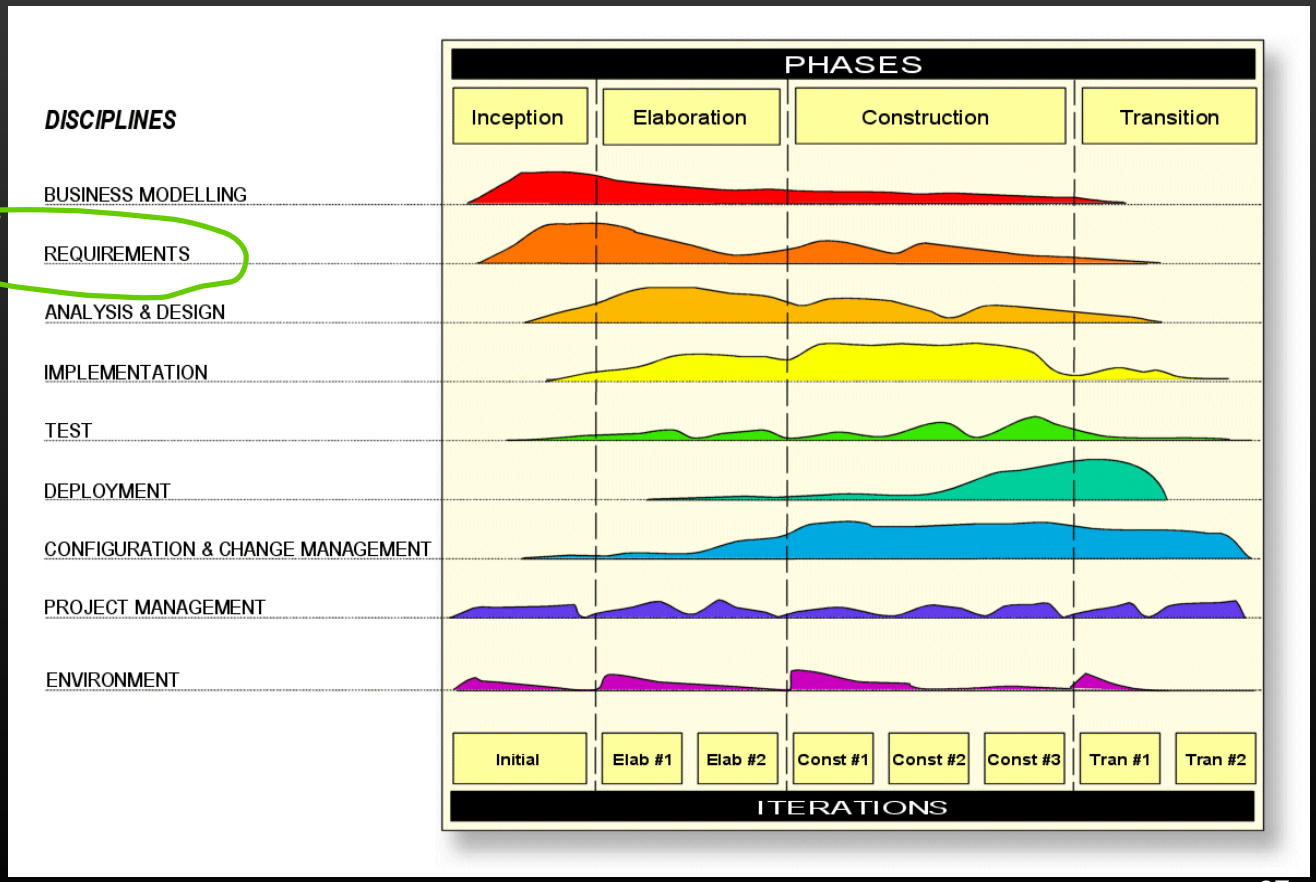


Visão geral do OpenUP/Unified Process

O UP oferece uma abordagem ao SDLC concebida como uma matriz, cruzando diferentes disciplinas técnicas com iterações (evoluções) no projecto.

(Nota: fases UP ≠ fases SDLC)

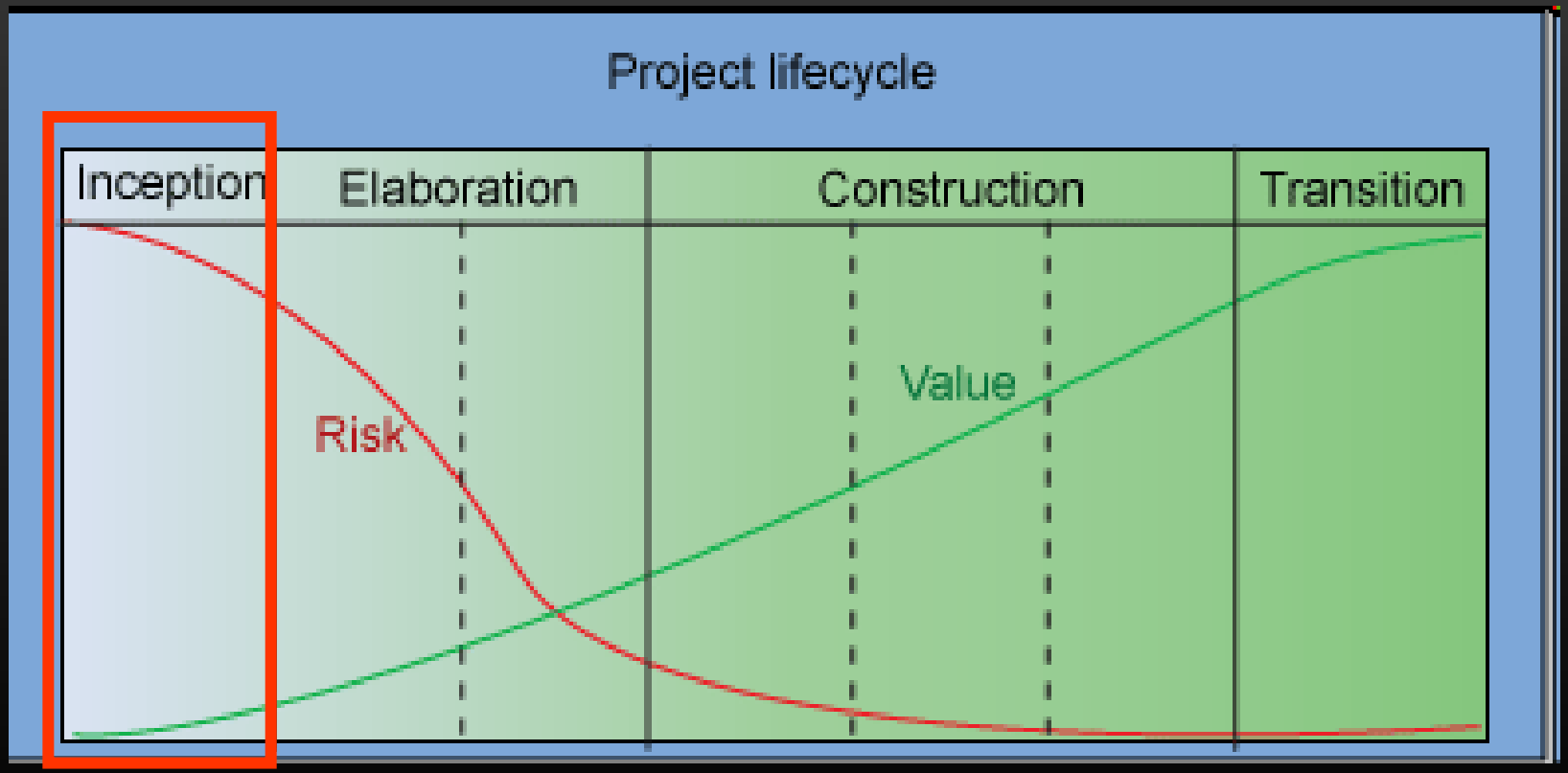
A análise dos requisitos é realizada principalmente no início do projeto (requisitos básicos), mas também durante as iterações (requisitos evolutivos).



The phases: Inception

Figura Project lifecycle

Do we agree on project scope and objectives, and whether or not the project should proceed?



Inception: Know What to Build

Typically one short iteration

Produce vision document and initial business case

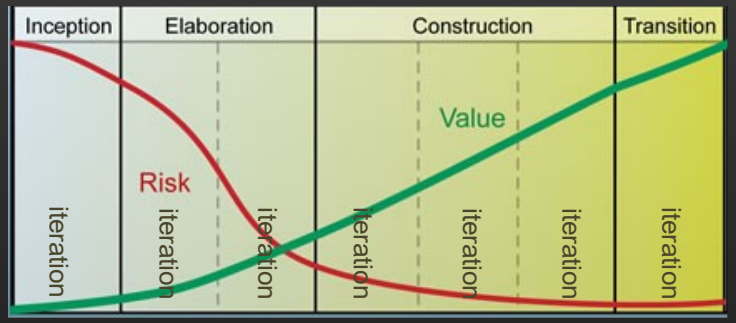
Develop high-level project requirements

- Initial use-case and (optional) domain models (10-20% complete)
- Focus on what is required to get agreement on 'big picture'

Manage project scope

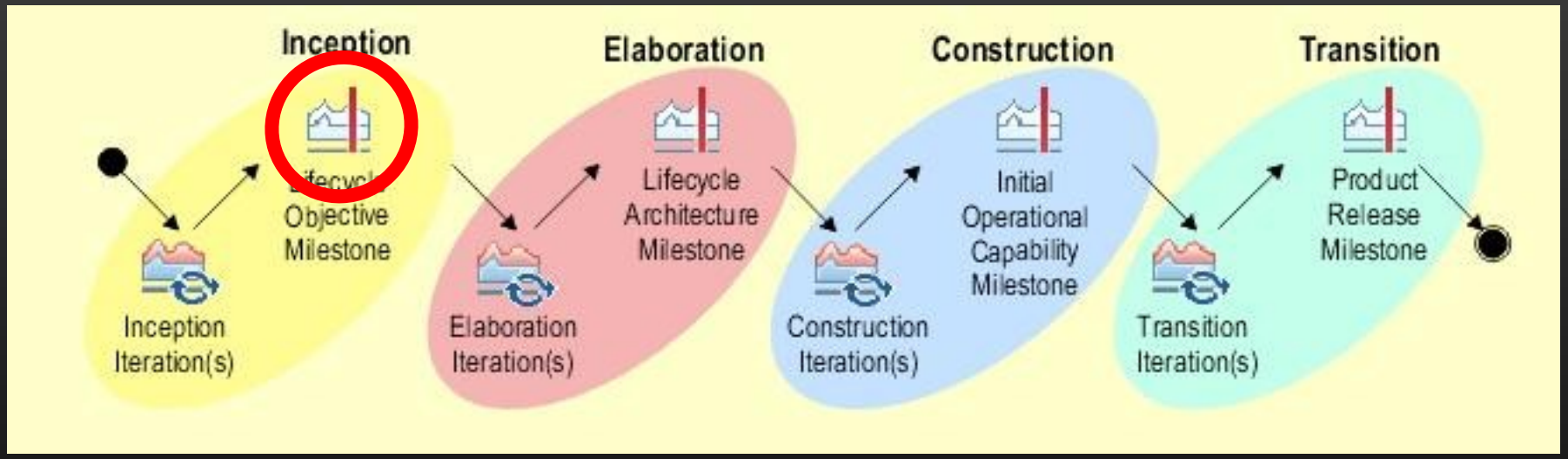
- Reduce risk by identifying key requirements
- Acknowledge that requirements will change
- Manage change, use iterative process

Produce conceptual prototypes as needed



Credit: Per Kroll (IBM)

Milestone: Inception



Objectives Milestone. At this point, you examine the cost versus benefits of the project, and decide either to proceed with the project or to cancel it.

Elaboration: Know How to Build It by Building Some

Elaboration can be a day long or several iterations

Balance

mitigating key technical and business risks with producing value (tested code)

Produce (and validate) an executable architecture

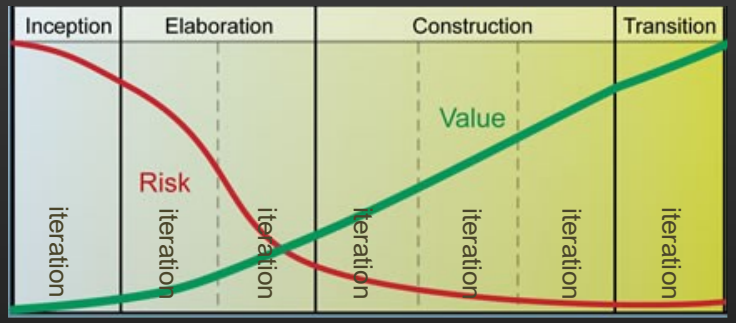
Define, implement and test interfaces of major components. Partially implement some key components.

Identify dependencies on external components and systems. Integrate shells/proxies of them.

Roughly 10% of code is implemented.

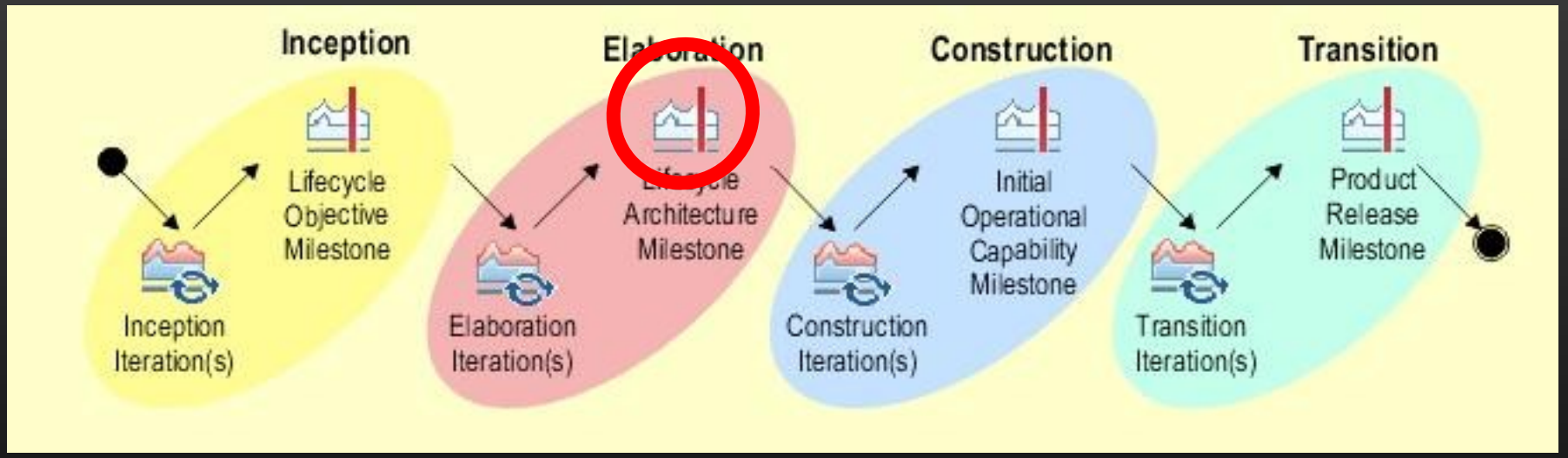
Drive architecture with key use cases

20% of use cases drive 80% of the architecture



Credit: Per Kroll (IBM)

Milestones: Elaboration



Architecture Milestone. At this point, a baseline of requirements is agreed to, you examine the detailed system objectives and scope, the choice of architecture, and the resolution of the major risks. The milestone is achieved when the architecture has been validated.

Construction: Build The Product

Incrementally define, design, implement and test more and more scenarios

- Incrementally evolve executable architecture to complete system
- Evolve architecture as you go along

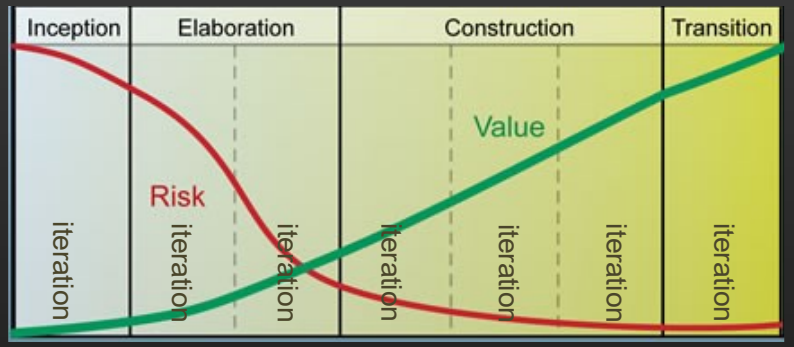
Frequent demonstrations and partial deployment

- Partial deployment strategy depends greatly on what system you build

Daily build with automated build process

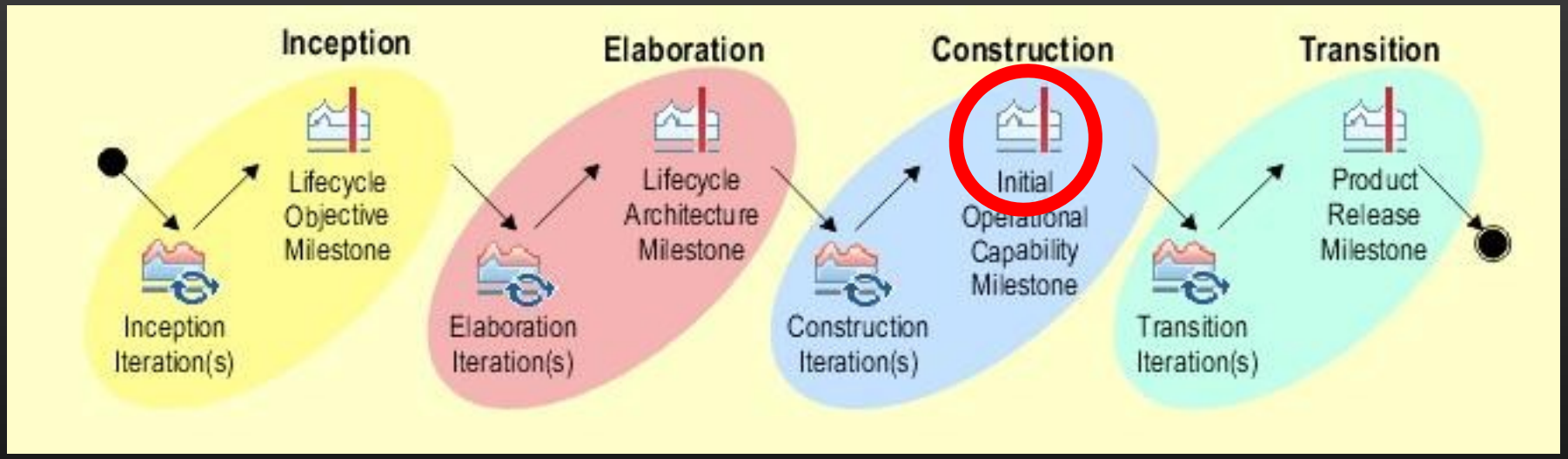
You may have to have a separate test team if you have

- Complex test environments
- Safety or mission critical systems



Credit: Per Kroll (IBM)

Milestones: Construction



Initial Operational Capability Milestone. At this point, the product is ready to be handed over to the transition team. All functionality has been developed and all alpha testing (if any) has been completed. In addition to the software, a user manual has been developed, and there is a description of the current release. The product is ready for beta testing.

Transition: Stabilize and Deploy

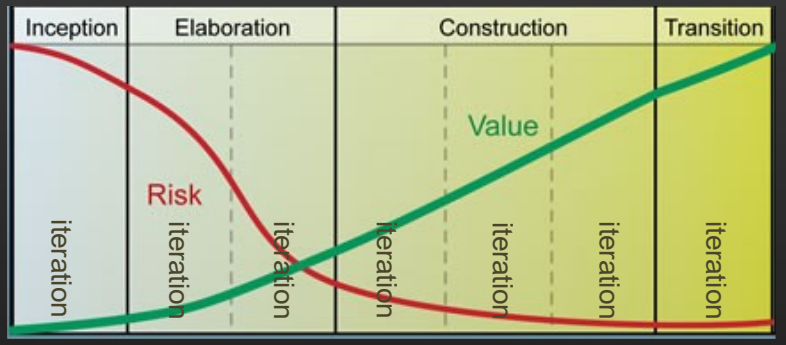
Project moves from focusing on new capabilities to **stabilizing** and tuning

Produce incremental ‘bug-fix’ releases

Update user manuals and deployment documentation

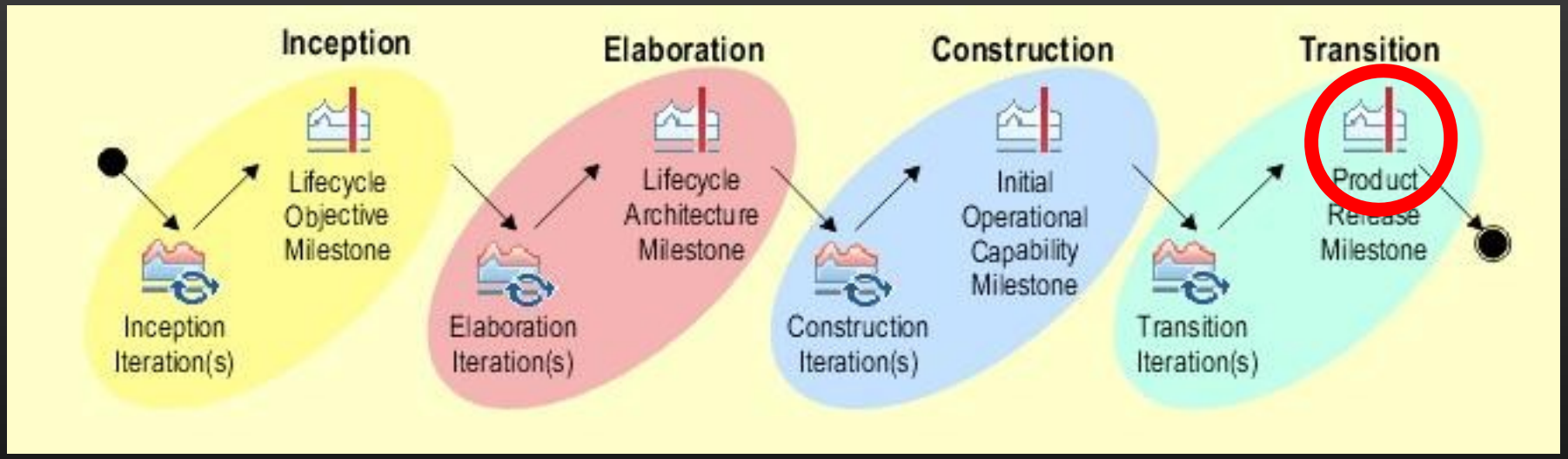
Execute cut-over

Conduct “post-mortem” project analysis



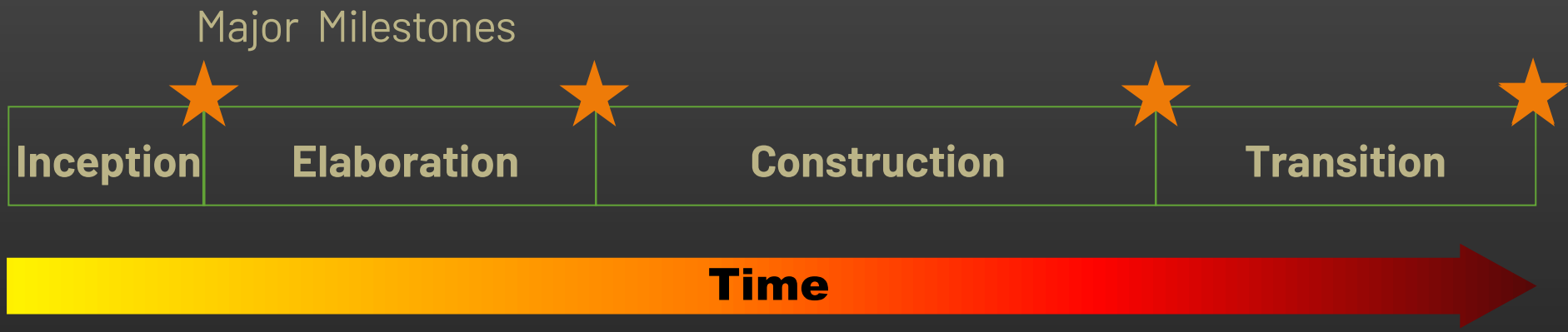
Credit: Per Kroll (IBM)

Milestones: Transition



Product Release Milestone. At this point, you decide if the objectives were met, and if you should start another development cycle. The Product Release Milestone is the result of the customer reviewing and accepting the project deliverables.

Recap main control points (lifecycle milestone)



Inception: **Agreement on overall scope**

Vision, high-level requirements, business case
Not detailed requirements

Elaboration: **Agreement on design approach and mitigation of major risks**

Baseline architecture, key capabilities partially implemented
Not detailed design

Construction: **Agreement on complete operational system**

Develop a beta release with full functionality

Transition: **Validate and deploy solution**

Stakeholder acceptance, cutover to production

Métodos “ágeis”

“Embrace change”

https://learning.oreilly.com/library/view/extreme-programming-explained/0321278658/

Em vez de:

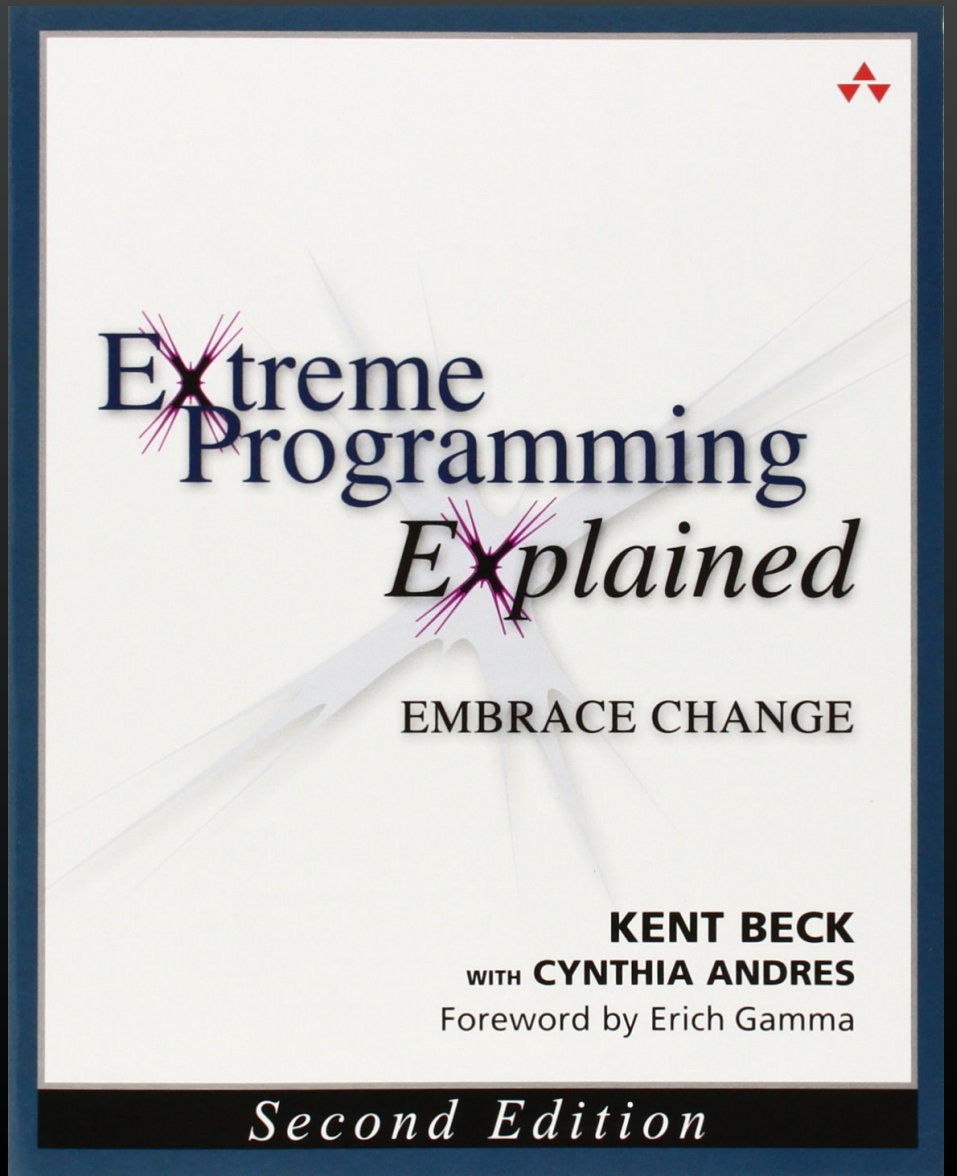
→ Lutar contra a inevitável a mudança que ocorre no desenvolvimento de software

→ Tentando (sem sucesso) especificar, congelar e "assinar" num conjunto de requisitos congelados e conceber antes da implementação

desenvolvimento iterativo e evolutivo:

→ Baseia-se numa atitude de abraçar a mudança e a adaptação como fatores inevitáveis e, na verdade, essenciais.

→ Mas: isto não quer dizer que o desenvolvimento iterativo incentive um processo descontrolado e reativo.



Manifesto para o Desenvolvimento Ágil de Software.

Ao desenvolver e ao ajudar outros a desenvolver software,
temos vindo a descobrir melhores formas de o fazer.
Através deste processo começámos a valorizar:

Indivíduos e interacções mais do que processos e ferramentas

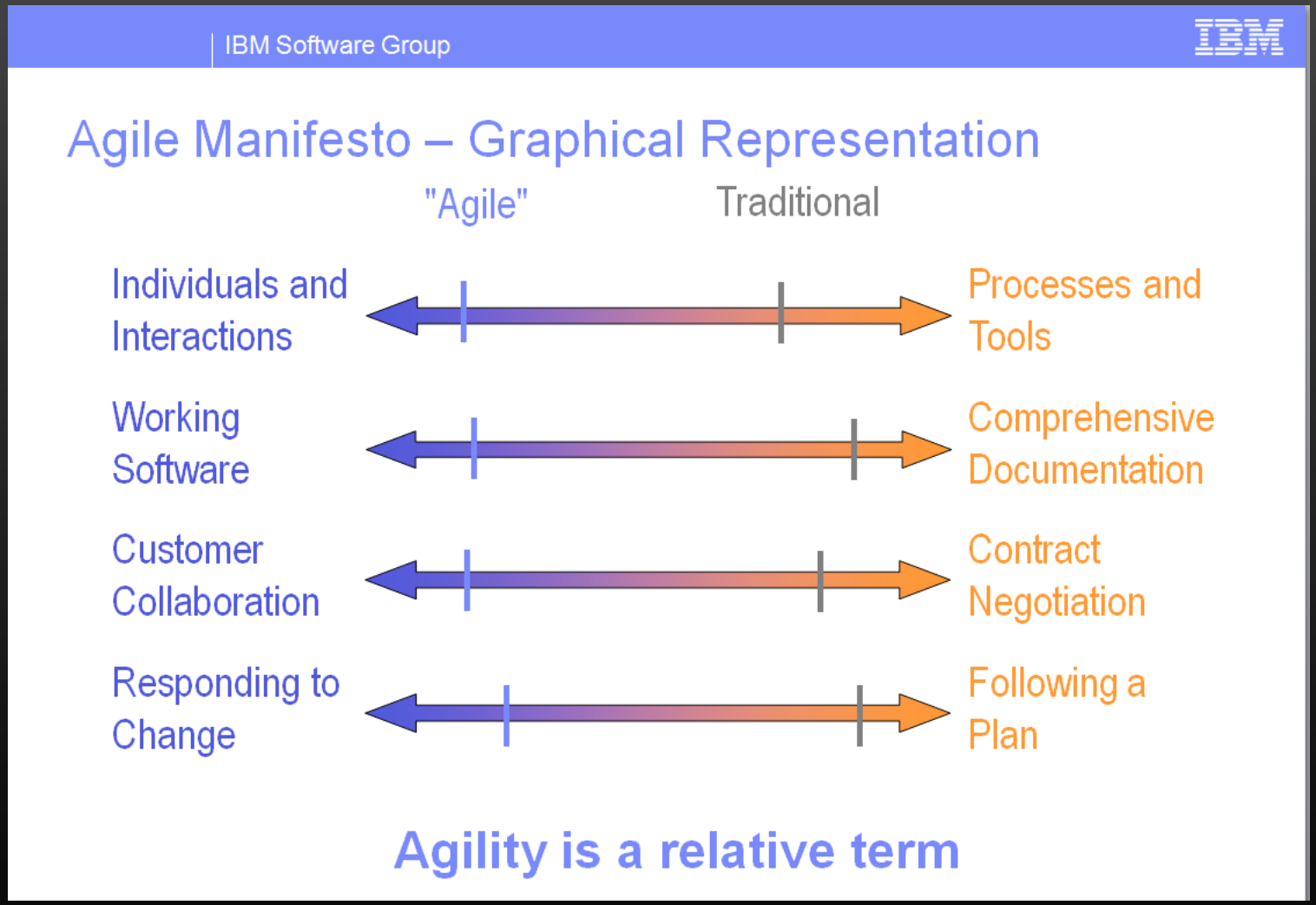
Software funcional mais do que documentação abrangente

Colaboração com o cliente mais do que negociação contratual

Responder à mudança mais do que seguir um plano

Ou seja, apesar de reconhecermos valor nos itens à direita,
valorizamos mais os itens à esquerda.

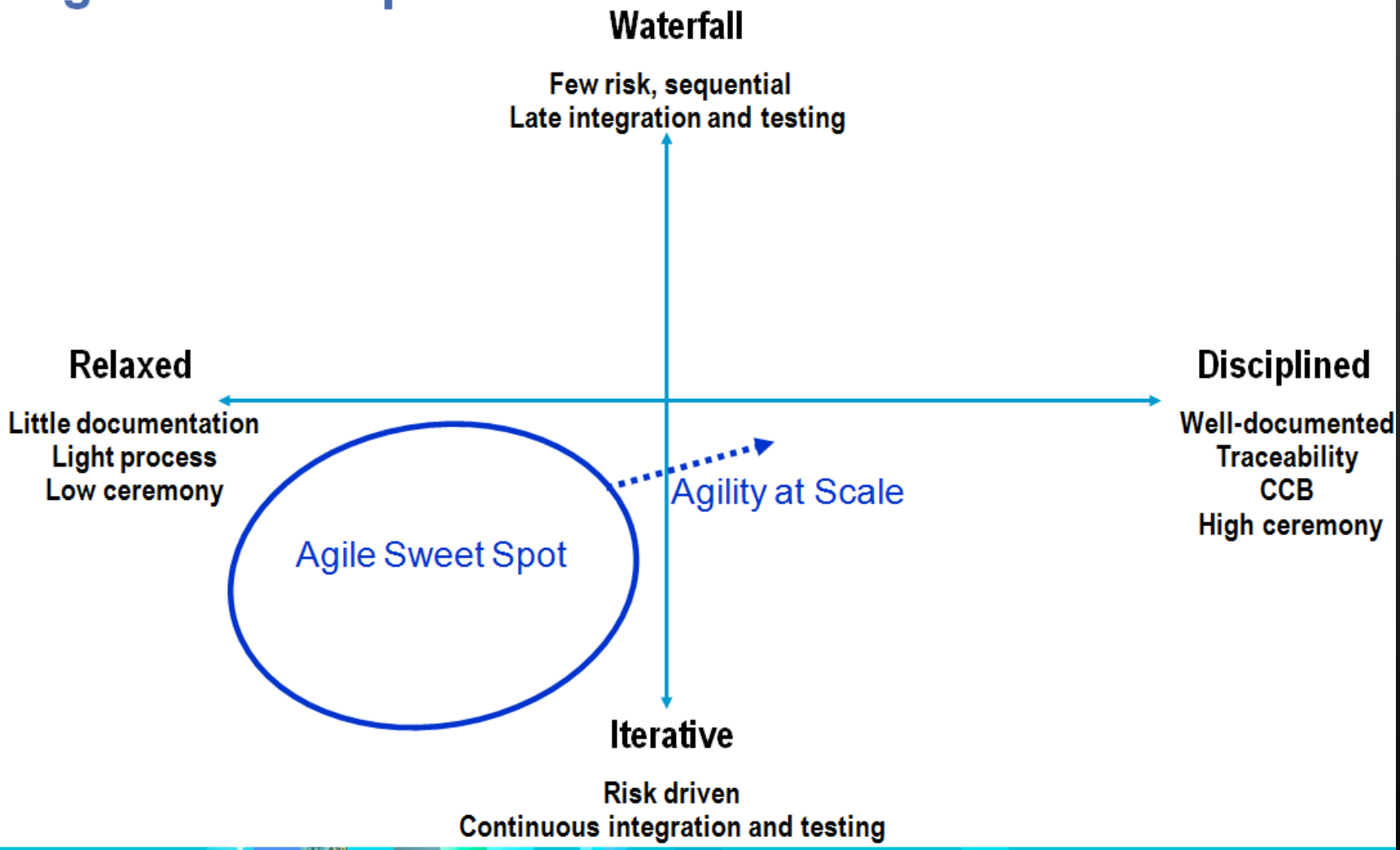
O desenvolvimento ágil de software



http://agilemanifesto.org

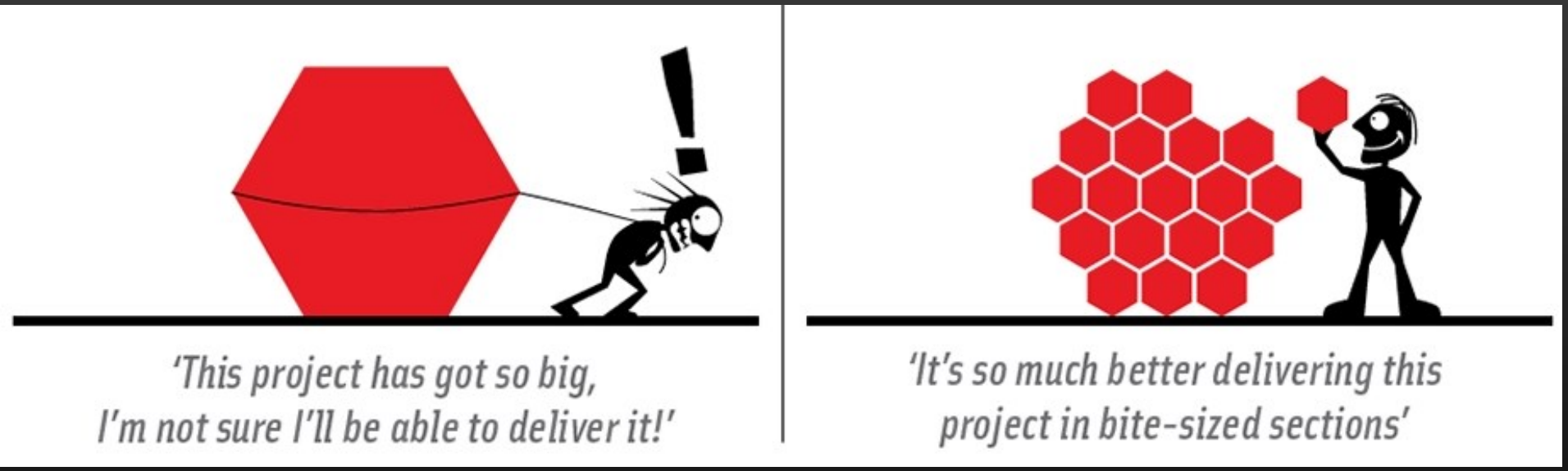
Credit: Per Kroll (IBM)

Agile Sweet Spot



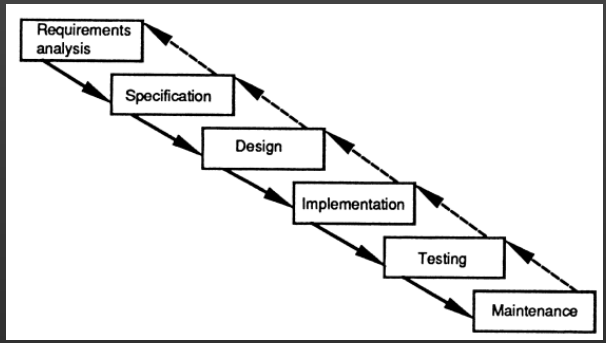
Credit: Per Kroll (IBM)

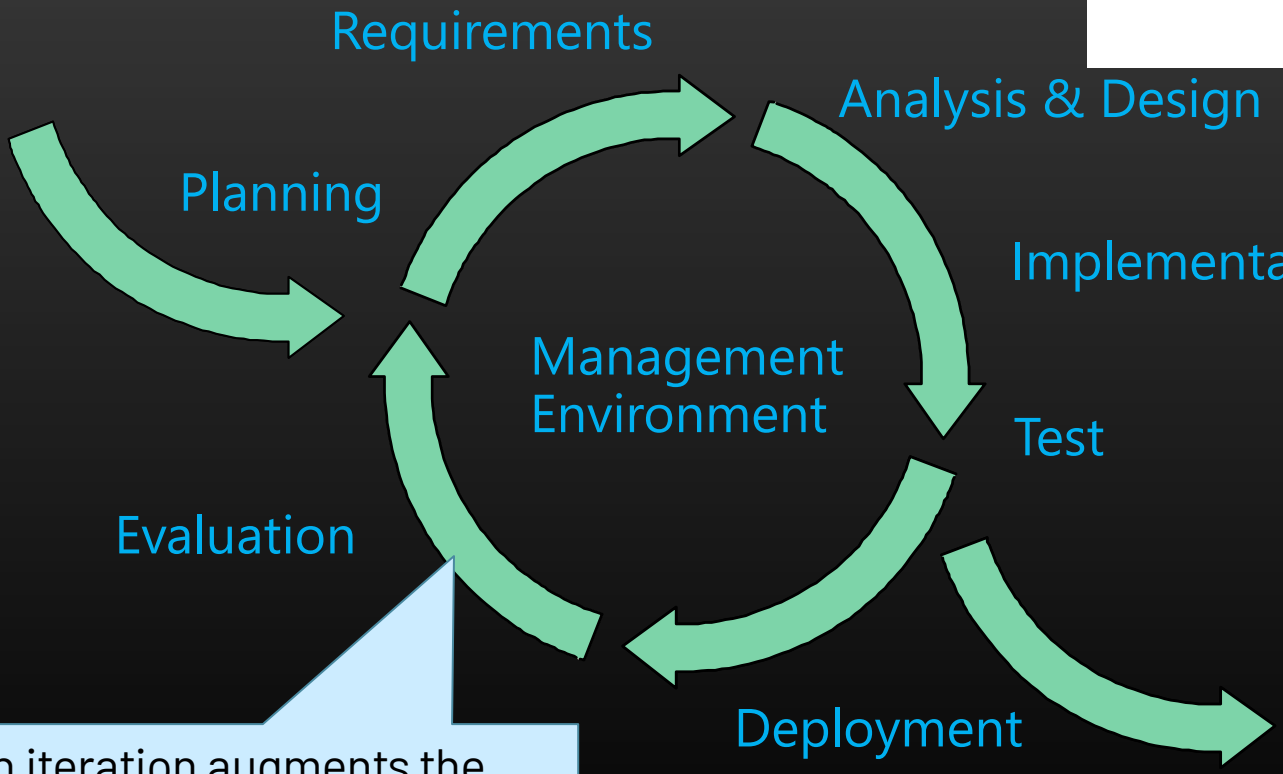
One project? Micro-projects?



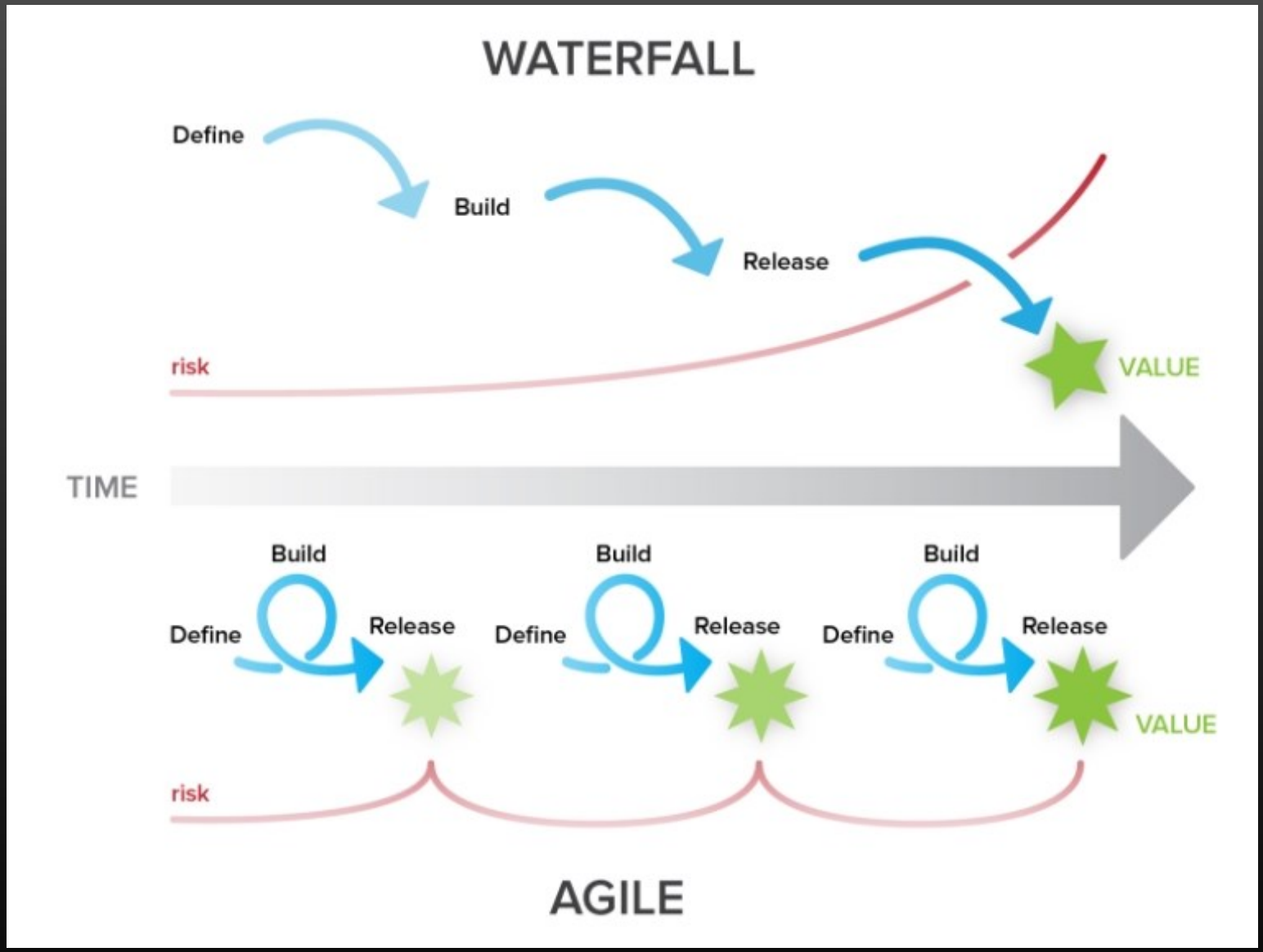
https://blog.ganttpro.com/en/waterfall-vs-agile-with-advantages-and-disadvantages/

Iterative development focuses on short and value-oriented cycles → Agile



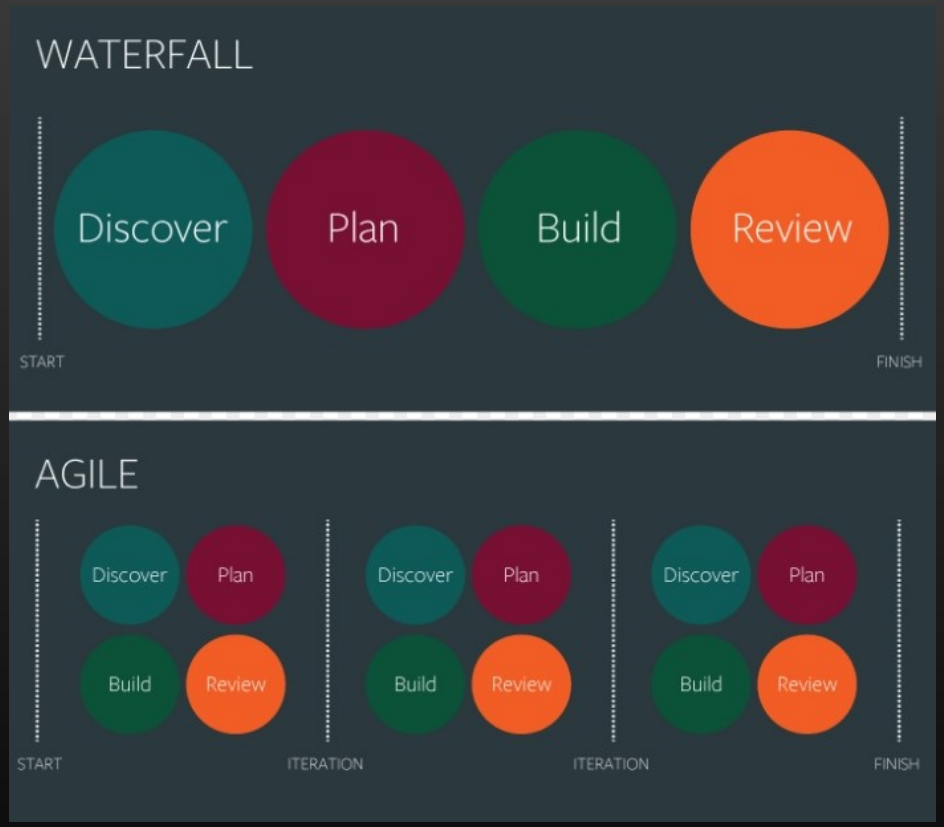


Each iteration augments the solution by integrating some executable result



https://blog.ganttpro.com/en/waterfall-vs-agile-with-advantages-and-disadvantages/

To be (agile) or not to be...



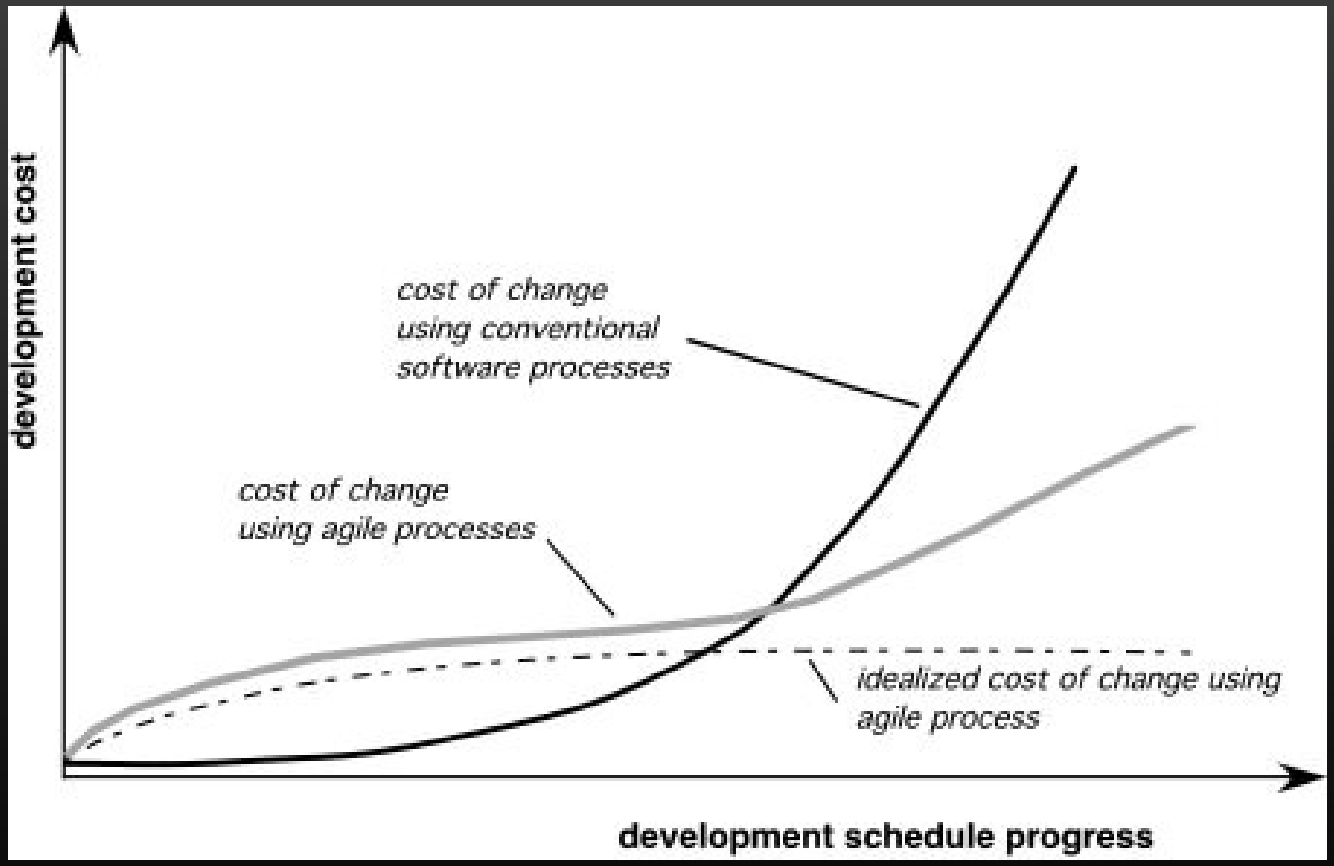
Iterative development (short cycles) *vs* linear development through stages?

Frequent business interaction *vs* fluctuations in stakeholder's participation?

Best to have good collaboration or a good plan?

Welcome changes to mitigate risk *vs* avoid changes to control risk?

Agility and the Cost of Change



The cost of change increases nonlinearly as the project progresses

Agility: key points

What is "Agility" in software development?

Effective (rapid and adaptive) response to change

Effective communication among all stakeholders

Drawing the customer onto the team

Organizing a team so that it is in control of the work performed

Yielding ...

Rapid, incremental delivery of software

Is OpenUP an agile process?

An Agile Process...

Is driven by customer descriptions of what is required (scenarios)

Recognizes that plans are short-lived

Develops software iteratively with a heavy emphasis on construction activities

Delivers multiple 'software increments'

Adapts as changes occur

Agile & Scrum

Scrum != software process

É um conjunto de práticas para gestão de equipas

Encaixa bem nas ideias de:

- desenvolvimento evolutivo
- Desenvolvimento ágil

Suportada em várias ferramentas

Entidades na gestão de um projeto à moda da "Scrum"

Backlog

- Ordenado por prioridade
- Features valem pontos

Iterações

- *timeboxed*

Iteração atual

- Atribuição e trabalho

Readings & references

Core readings	Suggested readings
• [Pressman'15] – Chap. 4, 5	• [Dennis'15] – Chap 1.