

Métodos ágeis

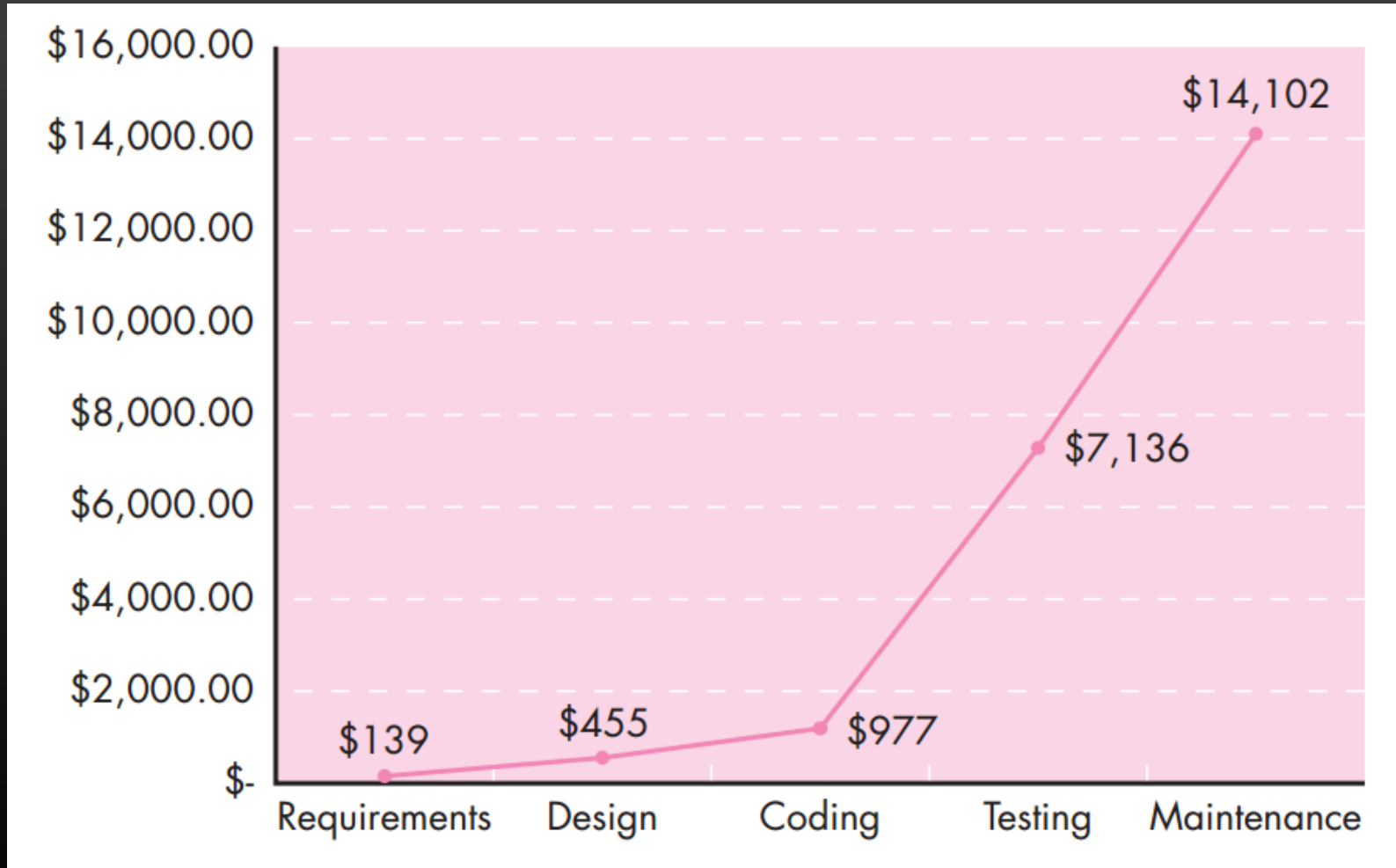
Ilídio Oliveira

v2021/12/17, TP10

Objetivos de aprendizagem

- Identificar vantagens de estruturar um projeto em iterações, produzindo incrementos frequentes.
- Caracterizar os princípios da gestão do *backlog* em abordagens ágeis.
- Identificar os papéis numa equipa de Scrum e as principais "cerimónias"
- Descrever a complementaridade entre o processo de software (e.g.: OpenUP) e uma metodologia de gestão de equipas (e.g.: Scrum)

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

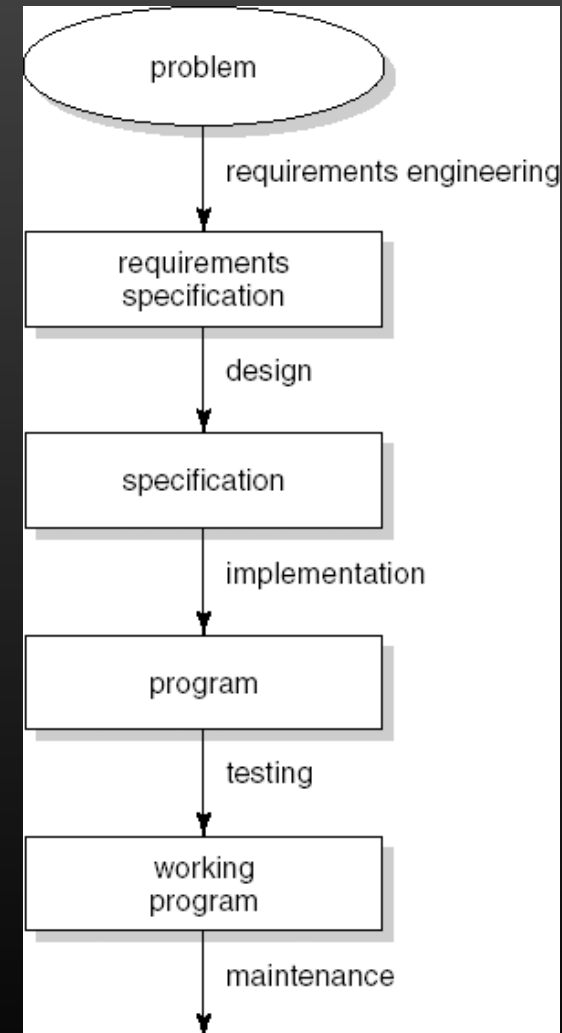


Boehm, B., and V. Basili, "Software Defect Reduction Top 10 List," IEEE Computer, vol. 34, no. 1, January 2001, pp. 135–137. <http://doi.ieeecomputersociety.org/10.1109/2.962984>

O processo de engenharia do software

Abordagem linear comum aos vários domínios de engenharia

Mas... é a única? É a “melhor”?...



Categorias para metodologias de desenvolvimento de software

Structured Analysis and Design

Estruturado para substituir o "ad – hoc" anterior

Abordagem passo-a-passo formal do SDLC que se move naturalmente de uma fase para a seguinte

e.g.: Waterfall

Rapid-Application Development

Fazer com que uma parte do sistema seja desenvolvida rapidamente e colocada nas mãos do utilizador (feedback antecipado e valor)

e.g.:
Phased/Incremental,
Prototyping

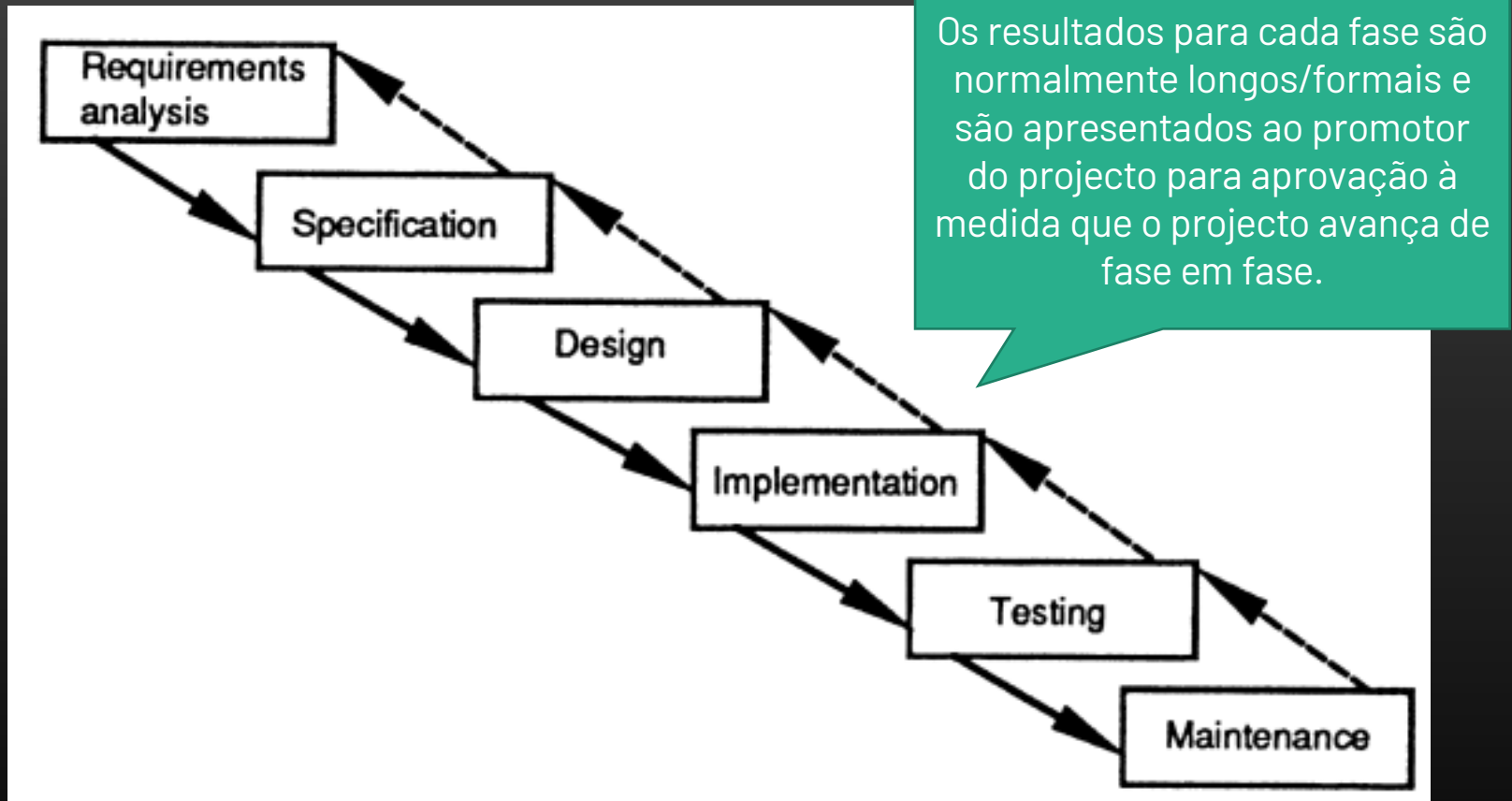
Agile development

Concentrar-se na eliminação de grande parte da sobrecarga de modelação e documentação; enfatizar o desenvolvimento de aplicações simples e iterativo.

Uma "atitude" para o desenvolvimento

e.g.: Extreme Programming

Abordagem "clássica" : Waterfall model



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

Vantagens do modelo de cascata

Simple e fácil de compreender e utilizar.

Plano linear

Pode ser estabelecido um calendário com prazos para cada fase de desenvolvimento e um produto pode prosseguir através do processo de desenvolvimento como um carro numa lavagem de automóveis, e teoricamente, ser entregue a tempo.

Fácil de gerir

cada fase tem resultados específicos e um processo de revisão.

As fases são tratadas e completadas uma de cada vez.

Funciona bem quando os requisitos são estáveis e bem definidos.

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.

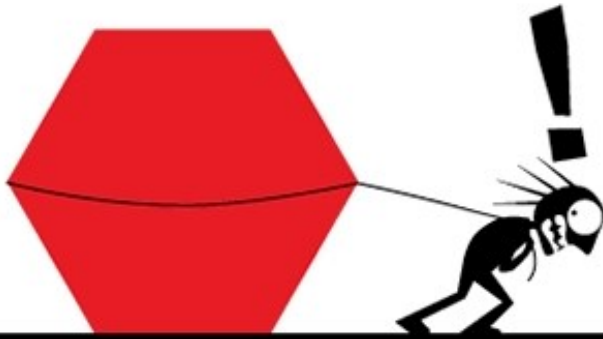
Why it may fail?

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

One project? Micro-projects?



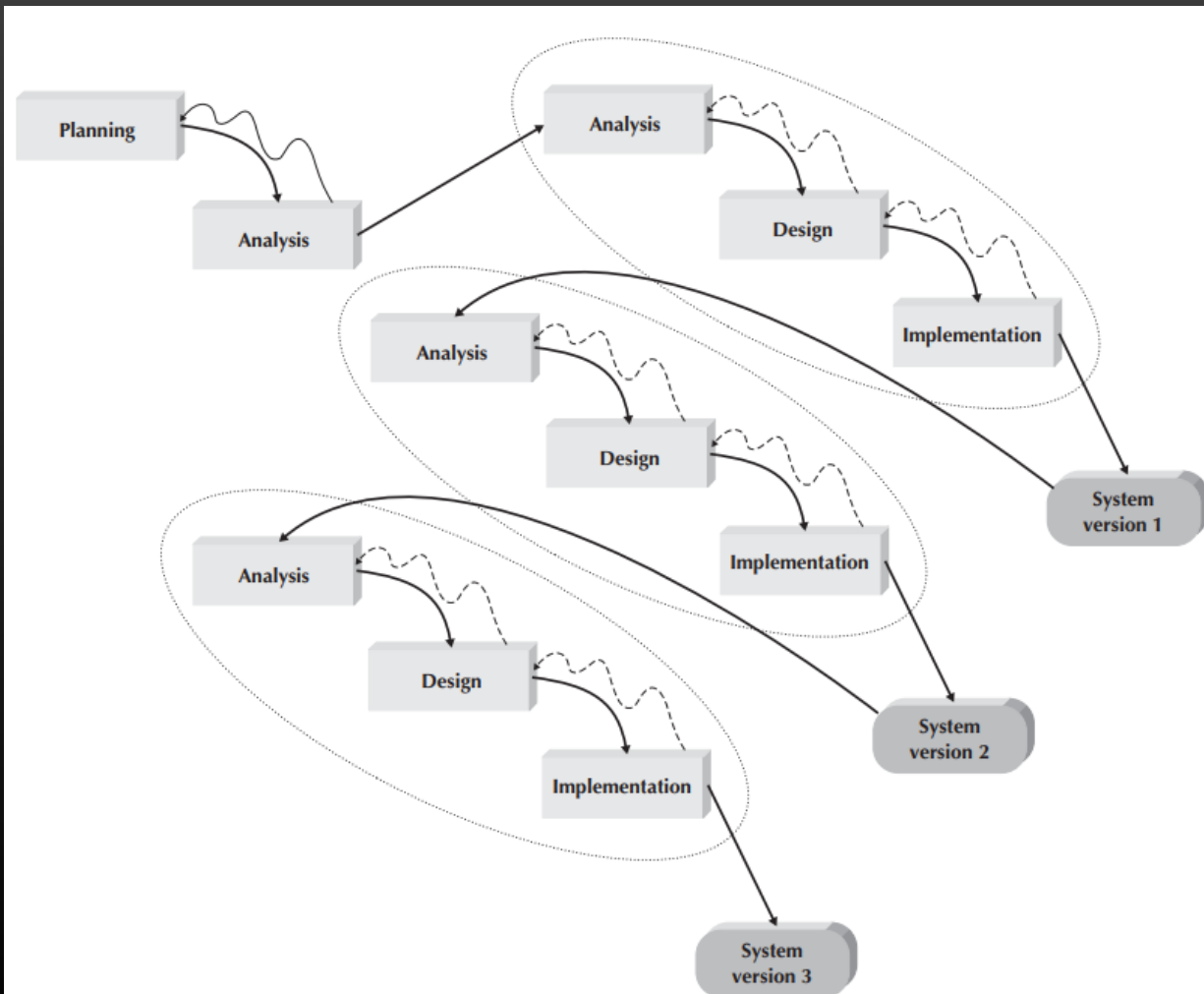
*'This project has got so big,
I'm not sure I'll be able to deliver it!'*



*'It's so much better delivering this
project in bite-sized sections'*

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

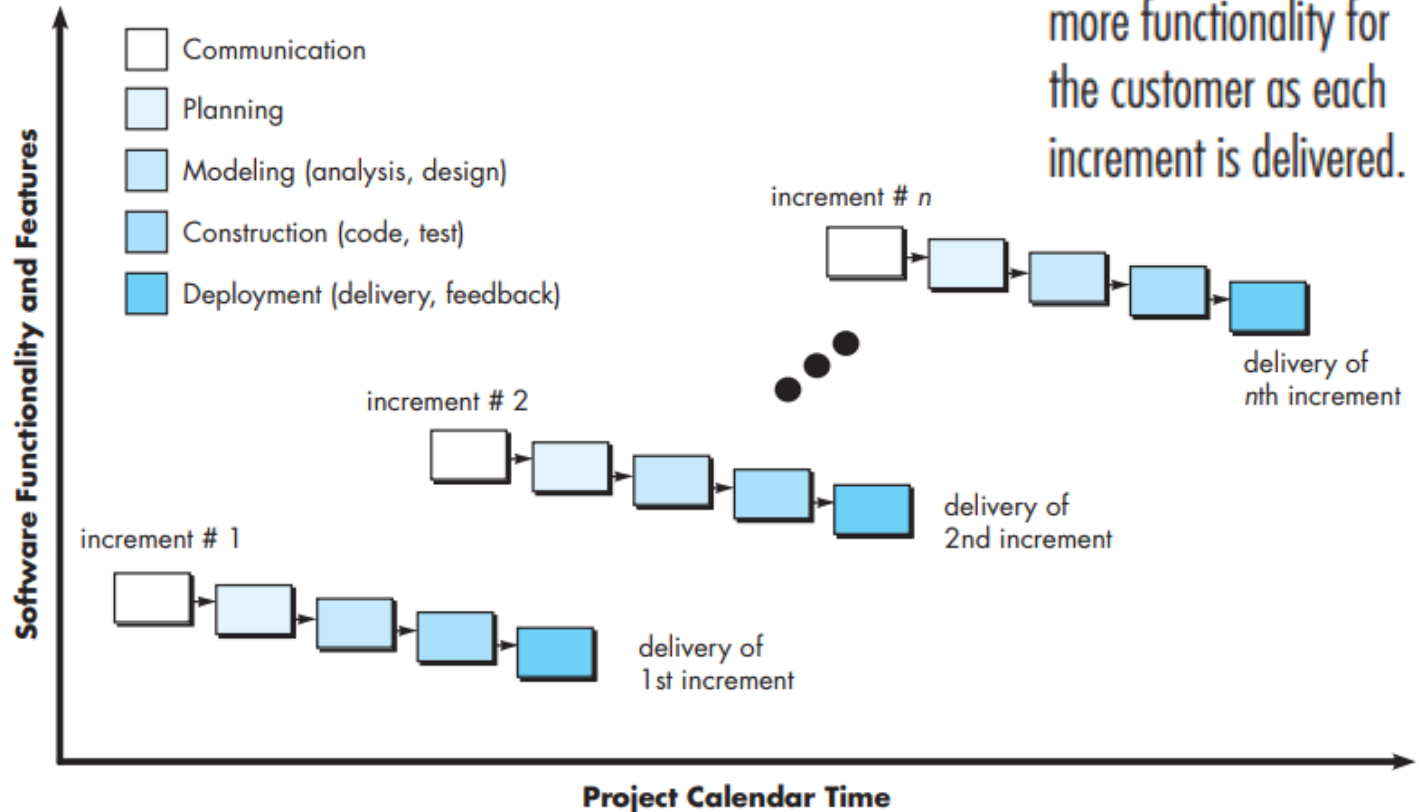
Phased / incremental



RAD: phased/incremental

FIGURE 4.3

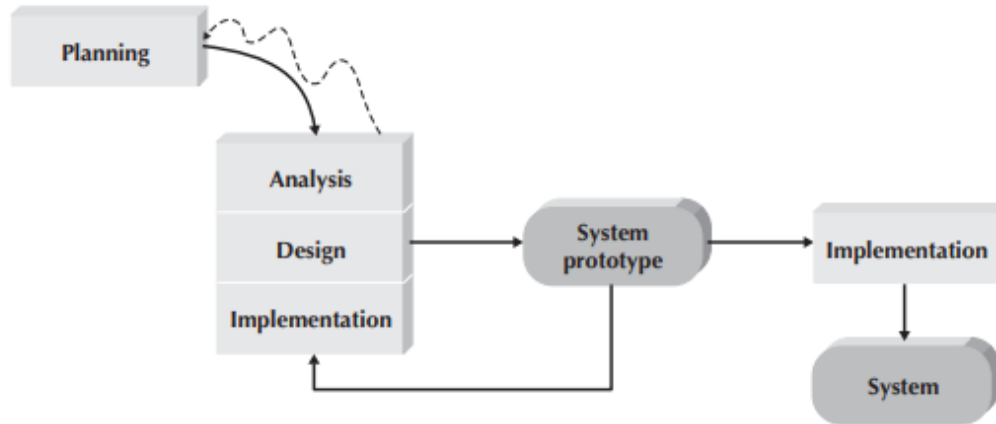
The incremental model



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

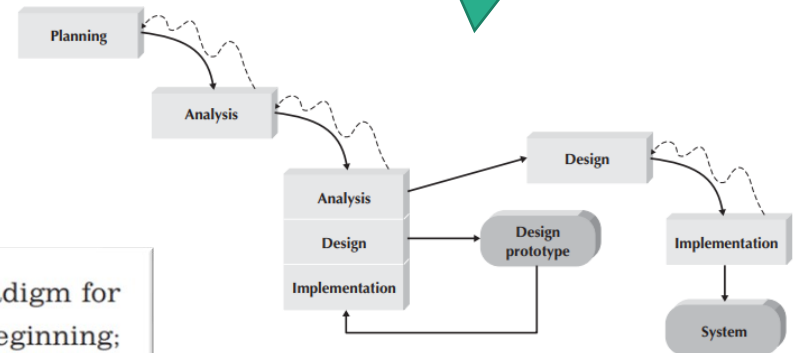
RAD: prototyping

FIGURE 1-5
A Prototyping-Based
Methodology



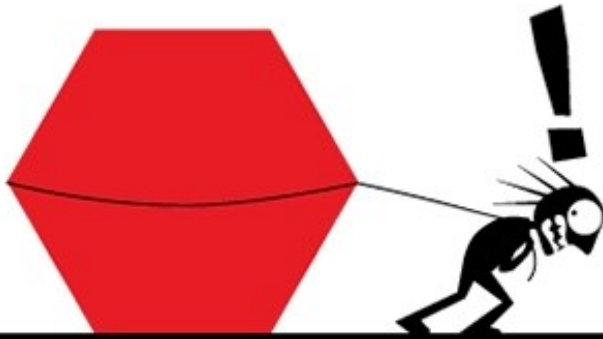
Throwaway prototyping:
a variation of
Prototyping in which the
prototype is discarded
(not used for the
implementation).

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.

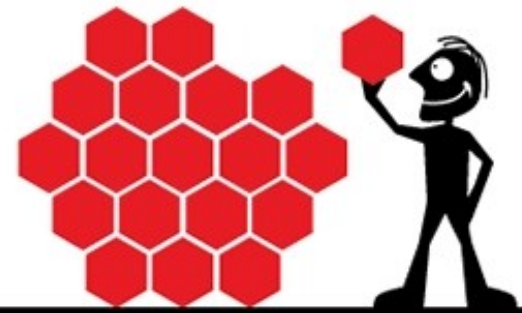


A Throwaway Prototyping-Based Methodology

Um grande projeto? Vários pequenos “projetos”?



*'This project has got so big,
I'm not sure I'll be able to deliver it!'*



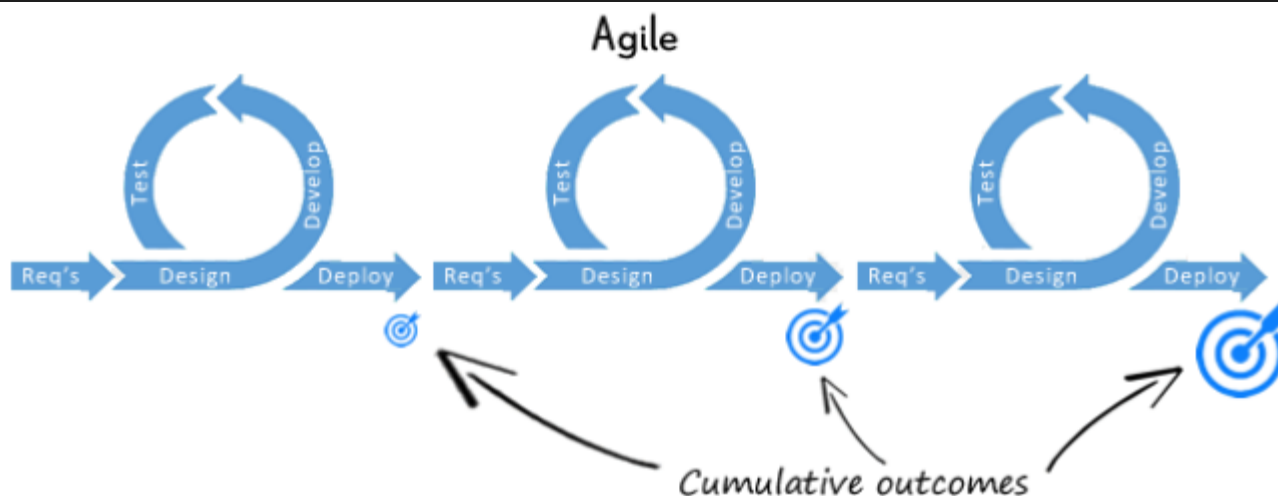
*'It's so much better delivering this
project in bite-sized sections'*

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

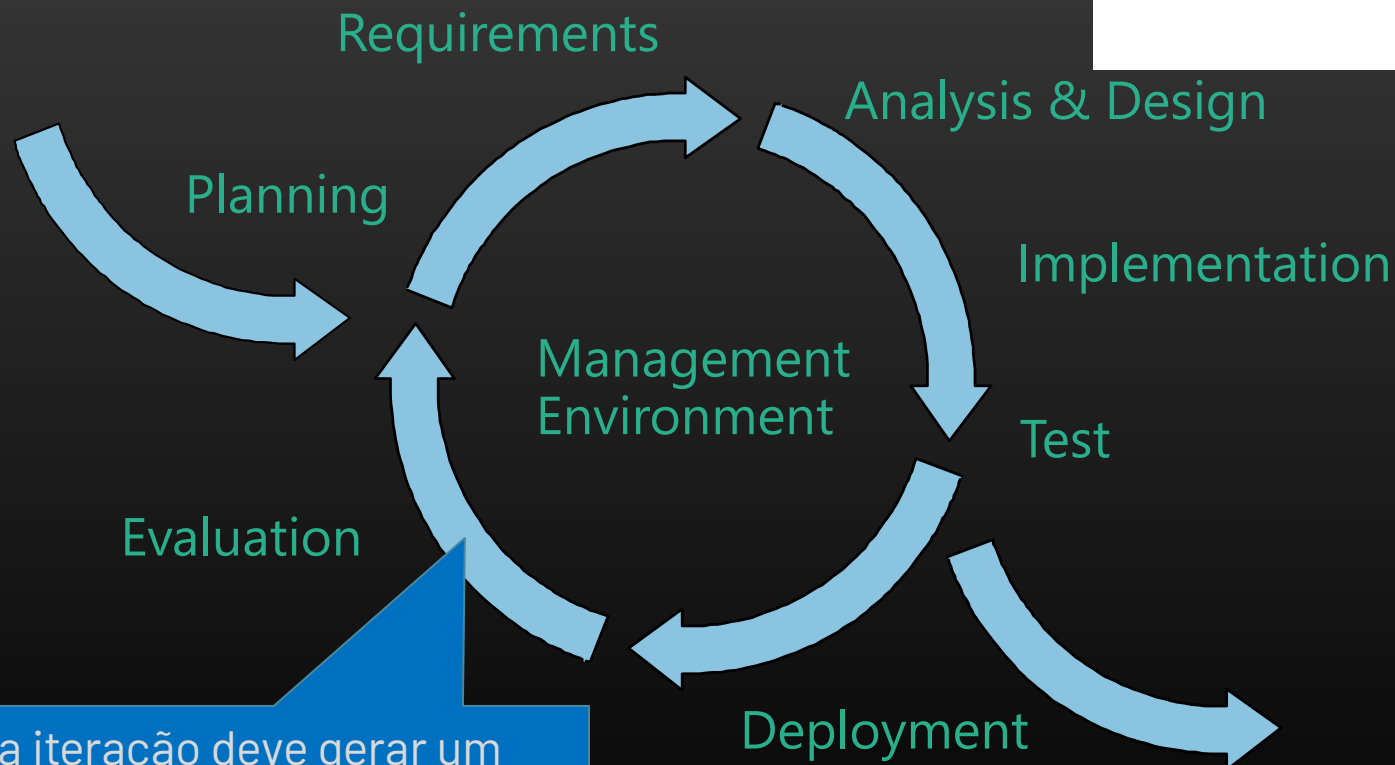
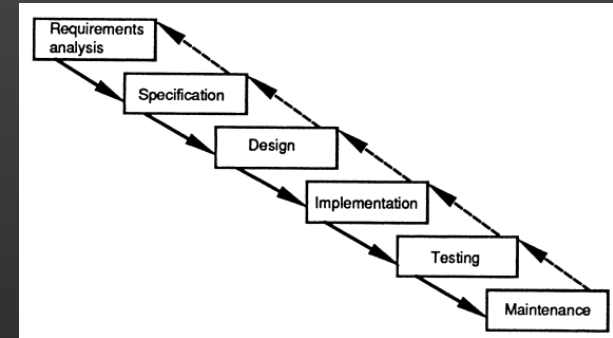
Desenvolvimento iterativo

Cada iteração envolve escolher um pequeno subconjunto dos requisitos, para projetar/desenhar, implementar e testar.

- Desenvolvimento conduzido por ciclos curtos
- Cada ciclo dá um incremento executável (parcial)
- Cada incremento é testado e integrado
- O feedback de cada iteração leva ao refinamento e adaptação da próxima.
- Reapreciação regular das prioridades

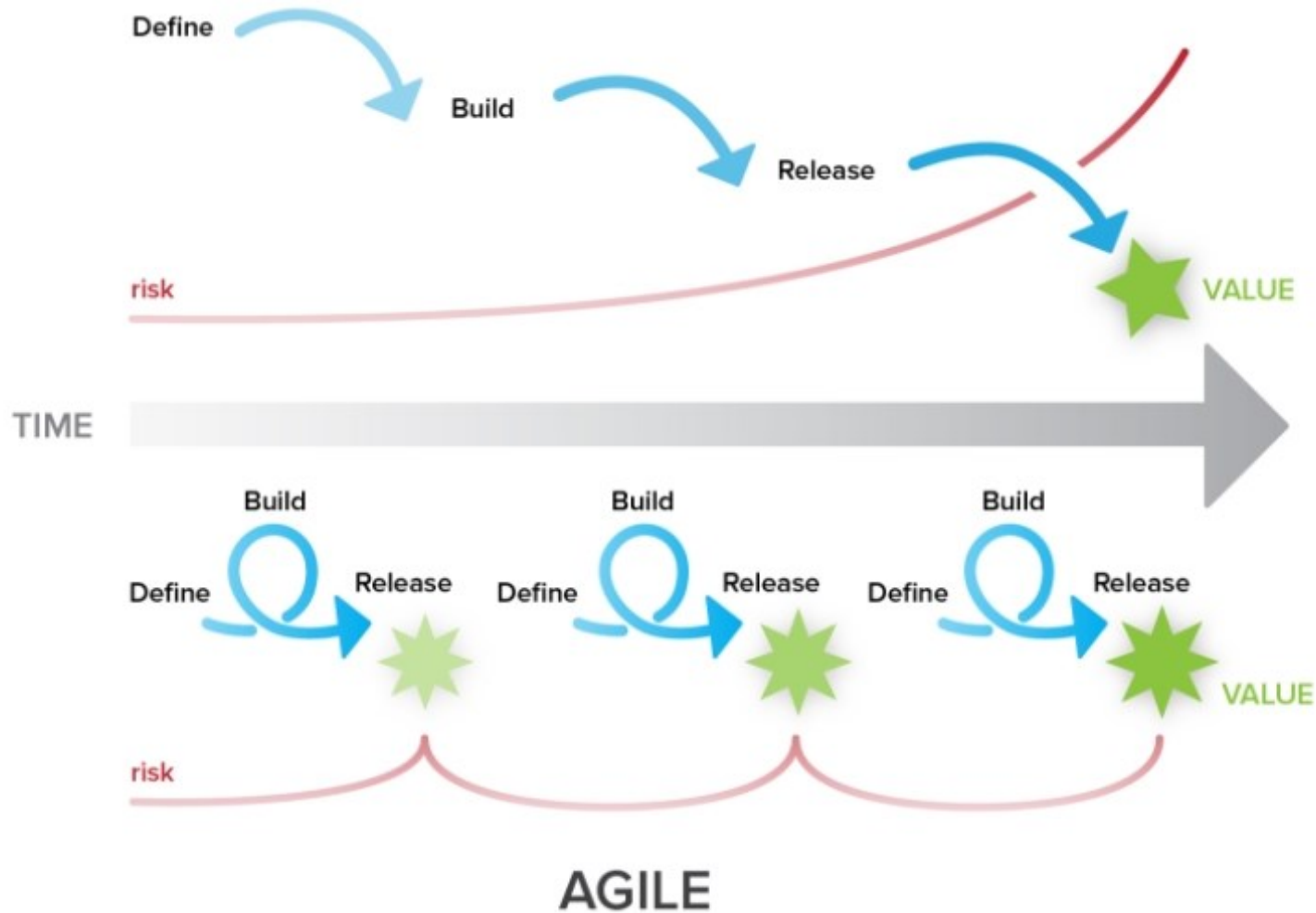


O desenvolvimento iterativo foca a entrega de valor orientada por ciclos curtos



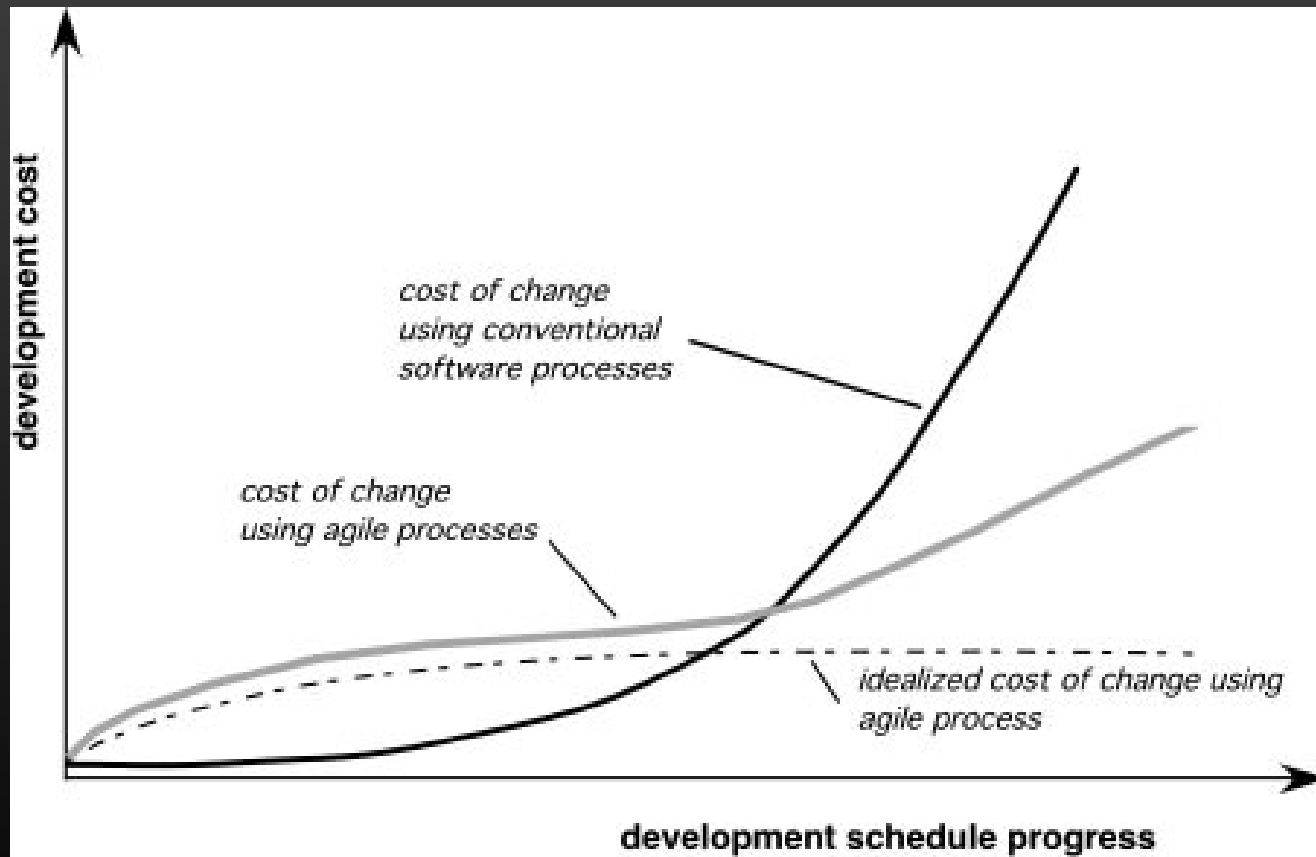
Cada iteração deve gerar um incremento funcional (entrega de valor)

WATERFALL



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

Agility and the Cost of Change



The cost of change increases nonlinearly as the project progresses

As equipas podem seleccionar entre várias metodologias

Ability to Develop Systems	Structured Methodologies		RAD Methodologies			Agile Methodologies	
	Waterfall	Parallel	Phased	Prototyping	Throwaway Prototyping	XP	SCRUM
With Unclear User Requirements	Poor	Poor	Good	Excellent	Excellent	Excellent	Excellent
With Unfamiliar Technology	Poor	Poor	Good	Poor	Excellent	Good	Good
That Are Complex	Good	Good	Good	Poor	Excellent	Good	Good
That Are Reliable	Good	Good	Good	Poor	Excellent	Excellent	Excellent
With a Short Time Schedule	Poor	Good	Excellent	Excellent	Good	Excellent	Excellent
With Schedule Visibility	Poor	Poor	Excellent	Excellent	Good	Excellent	Excellent

FIGURE 1-8 Criteria for Selecting a Methodology

Questões...

O sucesso de um projeto (de desenvolvimento de um SI) depende do seu tamanho?

As abordagens ágeis funcionam melhor que as abordagens sequenciais?

A abordagens ágeis funcionam melhor em certos tipos de projeto (e.g.: quanto ao tamanho)?

<https://www.infoq.com/articles/standish-chaos-2015/>

→ <http://bit.ly/3moxw1e>

A “normalização” do *agile*

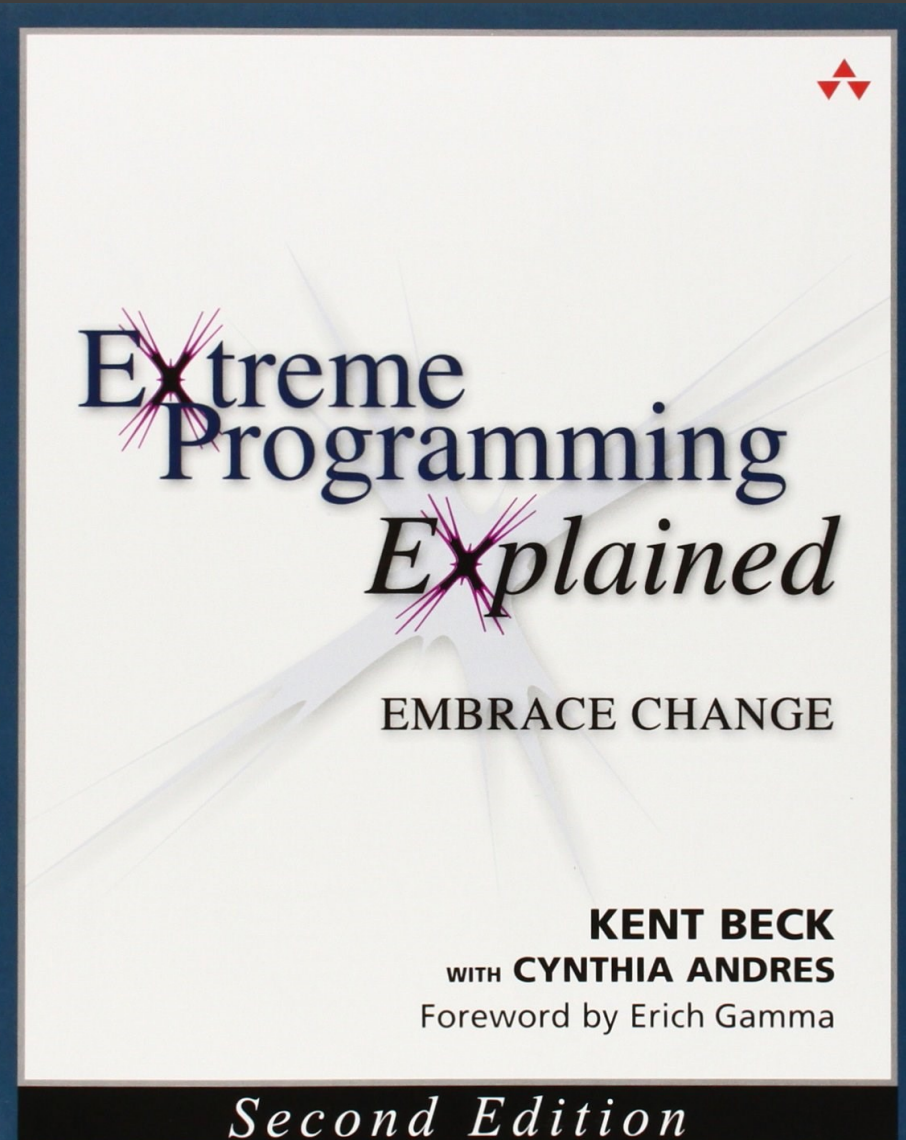
"Embrace change"

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

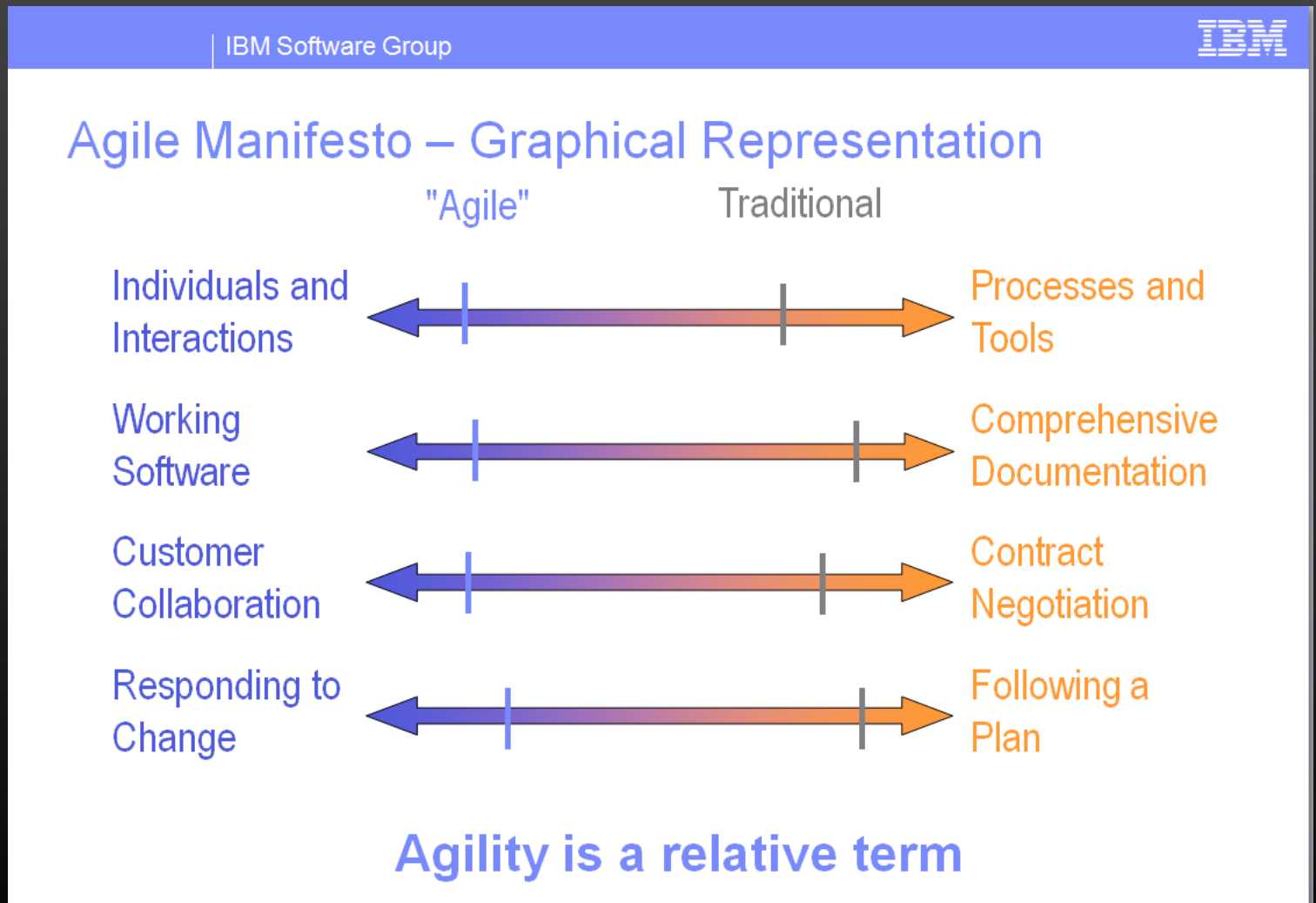
Software funcional mais do que documentação abrangente

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.

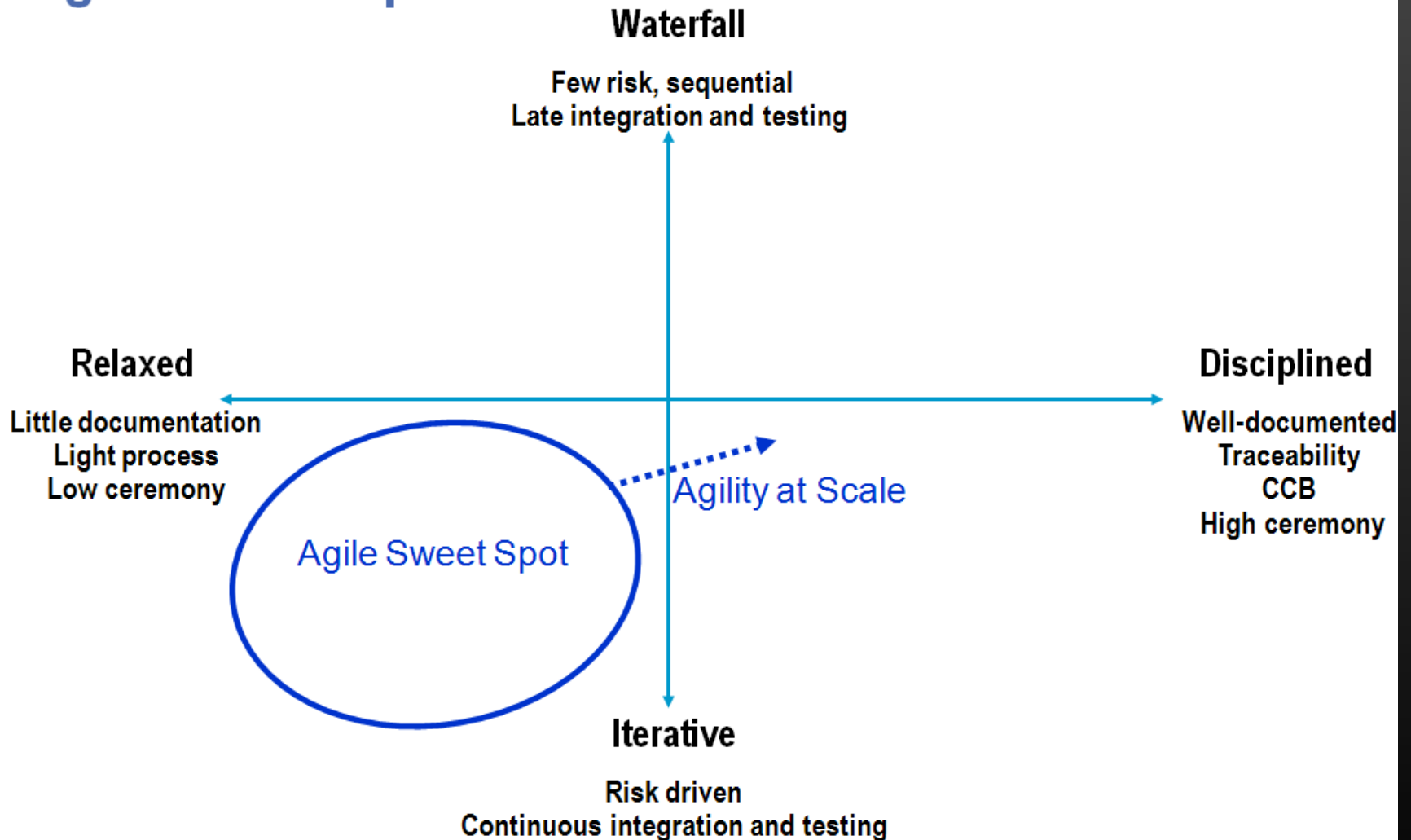
O desenvolvimento ágil de software



<http://agilemanifesto.org>

Credit: Per Kroll (IBM)

Agile Sweet Spot



Credit: Per Kroll (IBM)

Doze princípios para clarificar os valores

1. A nossa maior prioridade é, desde as primeiras etapas do projeto, a satisfação do cliente através da entrega rápida e contínua de valor (software implementado).
2. Aceitar alterações de requisitos, mesmo numa fase avançada do ciclo de desenvolvimento. Os processos ágeis potenciam a mudança em benefício da vantagem competitiva do cliente.
3. Fornecer frequentemente software pronto a funcionar. Os períodos de entrega devem ser de poucas semanas a poucos meses, dando preferência a períodos mais curtos.
4. As pessoas da área do negócio e a equipa de desenvolvimento devem trabalhar juntos, diariamente, durante o decorrer do projeto.
5. Desenvolver projetos com base em indivíduos motivados, dando-lhes o ambiente e o apoio de que necessitam, confiando que irão cumprir os objetivos.
6. O método mais eficiente e eficaz de passar informação para e dentro de uma equipa de desenvolvimento é através interações face-a-face (conversas diretas).
7. A principal medida de progresso é a entrega de software a funcionar.
8. Os processos ágeis promovem o desenvolvimento a um ritmo sustentável. Os promotores, a equipa e os utilizadores deverão ser capazes de manter um bom ritmo de trabalho, indefinidamente.
9. A atenção permanente à excelência técnica e um bom desenho da solução aumentam a agilidade.
10. Simplicidade – a arte de maximizar a quantidade de trabalho que não é feito – é essencial.
11. As melhores arquiteturas, requisitos e desenhos emergem das equipas que se auto-organizam.
12. A equipa reflete regularmente sobre o modo de se tornar mais eficaz, fazendo os ajustes e adaptações necessárias.

Destacando algumas características da abordagem “ágil”

Objetivos:

Satisfação do cliente

Mitigação de risco

Ritmo na equipa

Para atingir a agilidade é necessário:

- Ciclos curtos e entrega de valor frequente
- Envolvimento do “negócio”
- Resposta rápida à alteração de condições (e.g. alteração de requisitos)
- Colaboração e comunicação na equipa
- Aplicar práticas de construção de software evolutivas: integração em contínuo, desenvolvimento orientado por testes,...

Como é que o Unified Process encaixa?

Unified Process/Open Unified Process

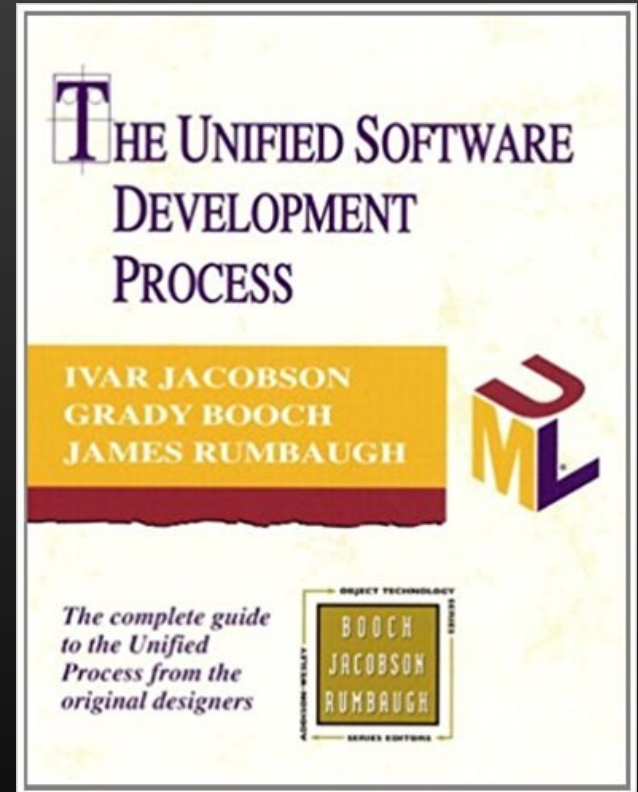
As metodologias são concretizadas em processos (de software)

UP fornece um processo que integra ideias do desenvolvimento incremental e iterativo

Uma tentativa para um processo de genérico

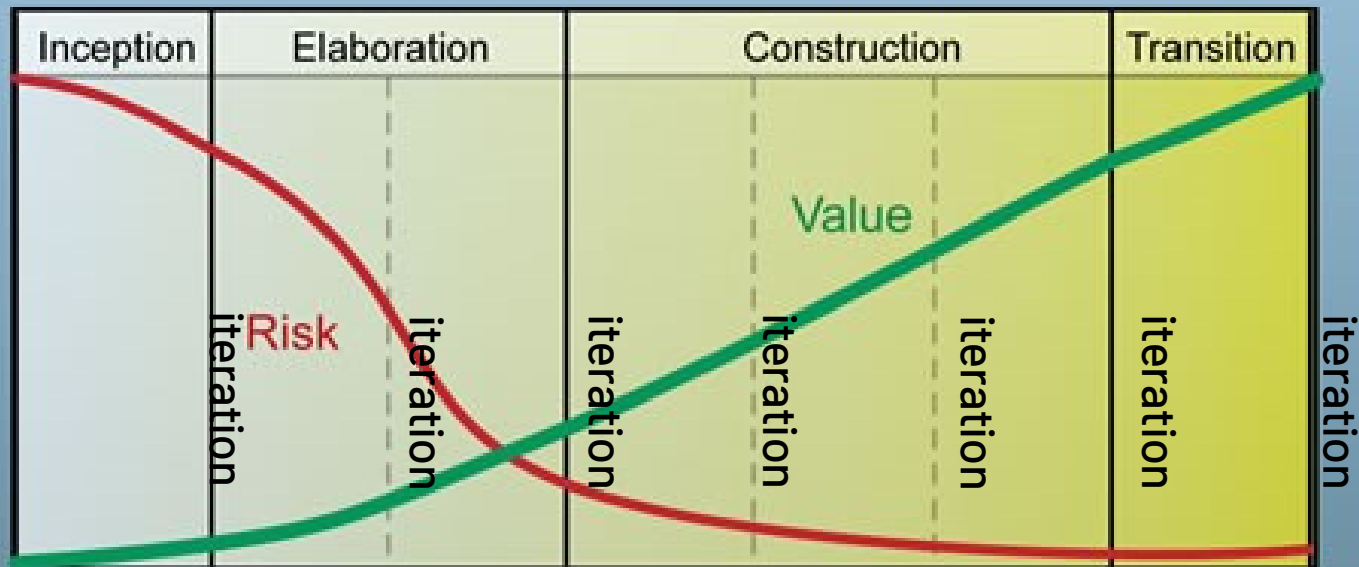
Pode ser adaptado ao projecto específico em mãos

O OpenUP é uma versão "livre" do Unified Process



Natureza iterativa

Project Lifecycle



THE UNIFIED SOFTWARE
DEVELOPMENT
PROCESS

IVAR JACOBSON
GRADY BOOCH
JAMES RUMBAUGH



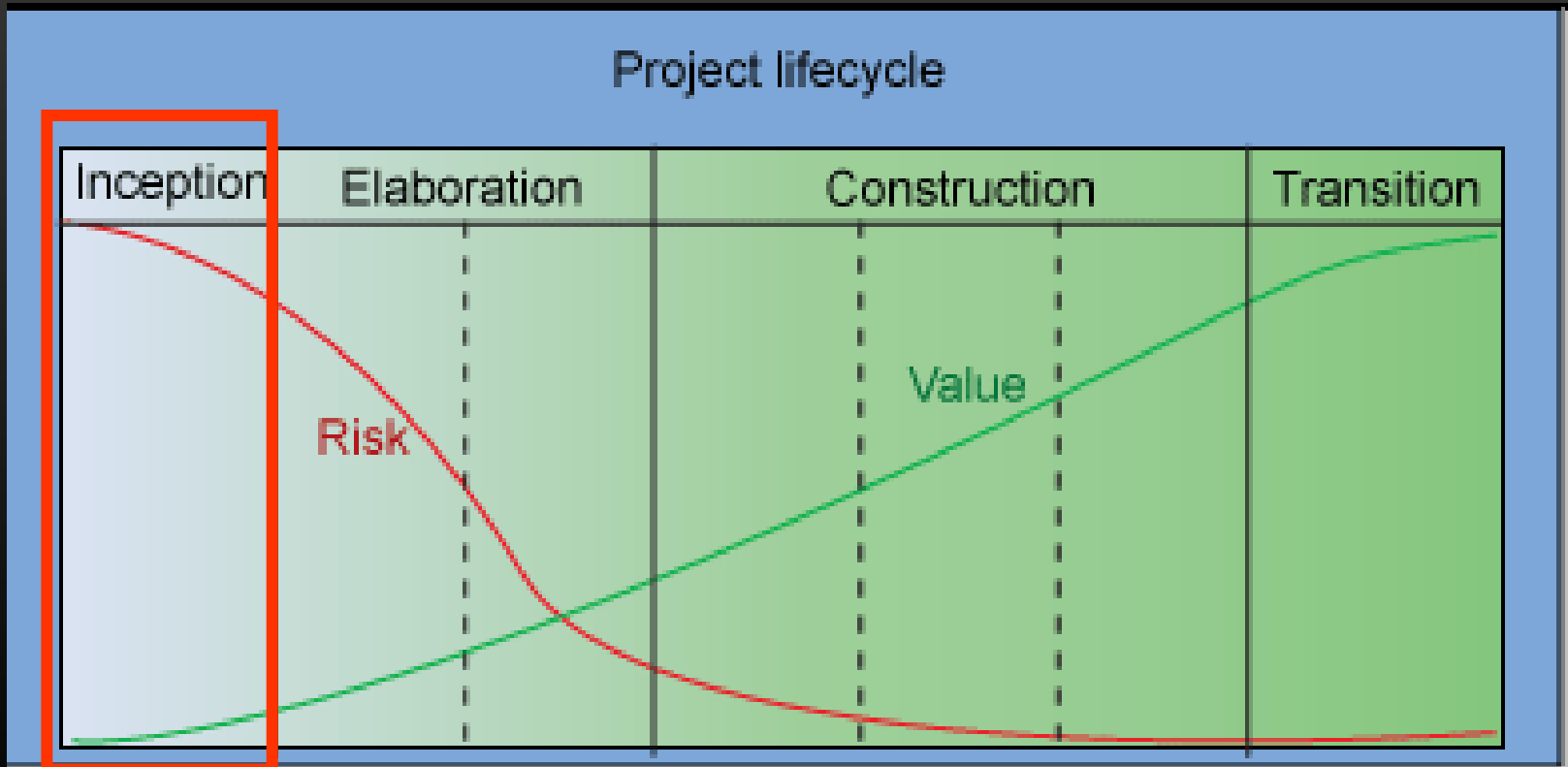
*The complete guide
to the Unified
Process from the
original designers*

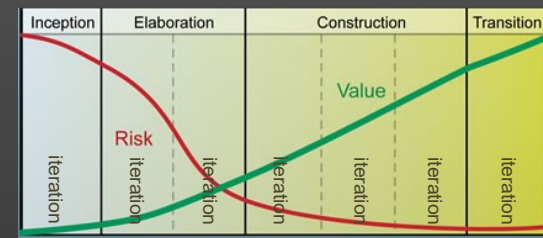


The phases: Inception

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

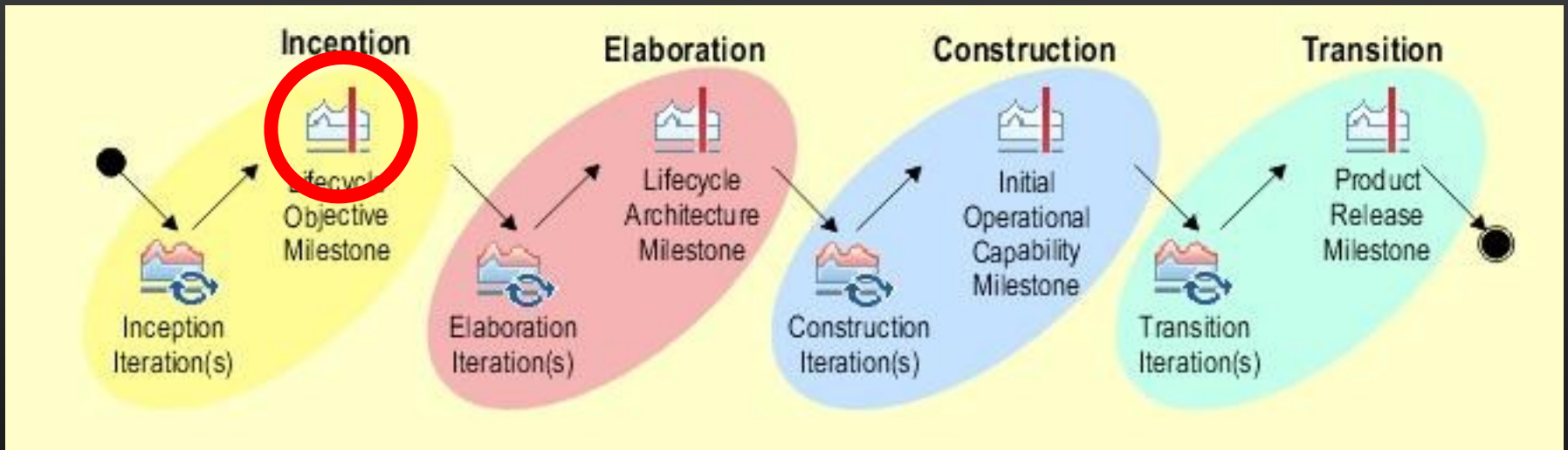
Acknowledge that requirements will change

Manage change, use iterative process

Produce conceptual prototypes as needed

Credit: Per Kroll (IBM)

Milestone: Inception



Lifecycle 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.

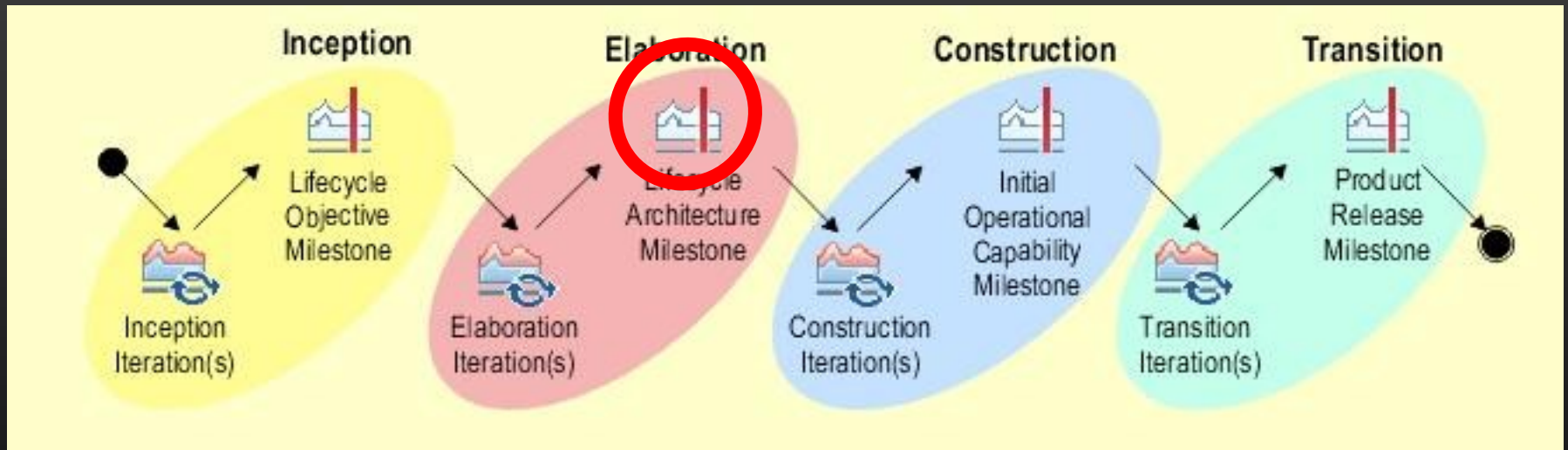
Roughly 10% of code is implemented.

Drive architecture with key use cases

20% of use cases drive 80% of the architecture

Credit: Per Kroll (IBM)

Milestones: Elaboration

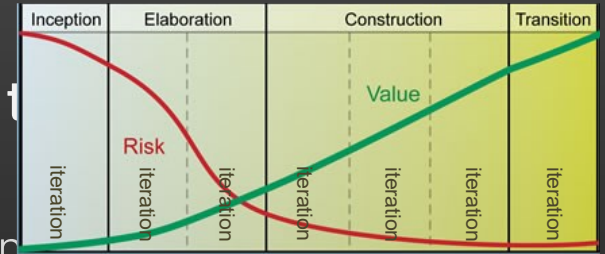


Lifecycle Architecture Milestone. At this point, a baseline of requirements is agreed to, you examine the detailed system objectives and scope, the choice of architecture, and the resolution of the major risks. The milestone is achieved when the architecture has been validated.

Construction: Build The Product

Incrementally define, design, implement and test 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

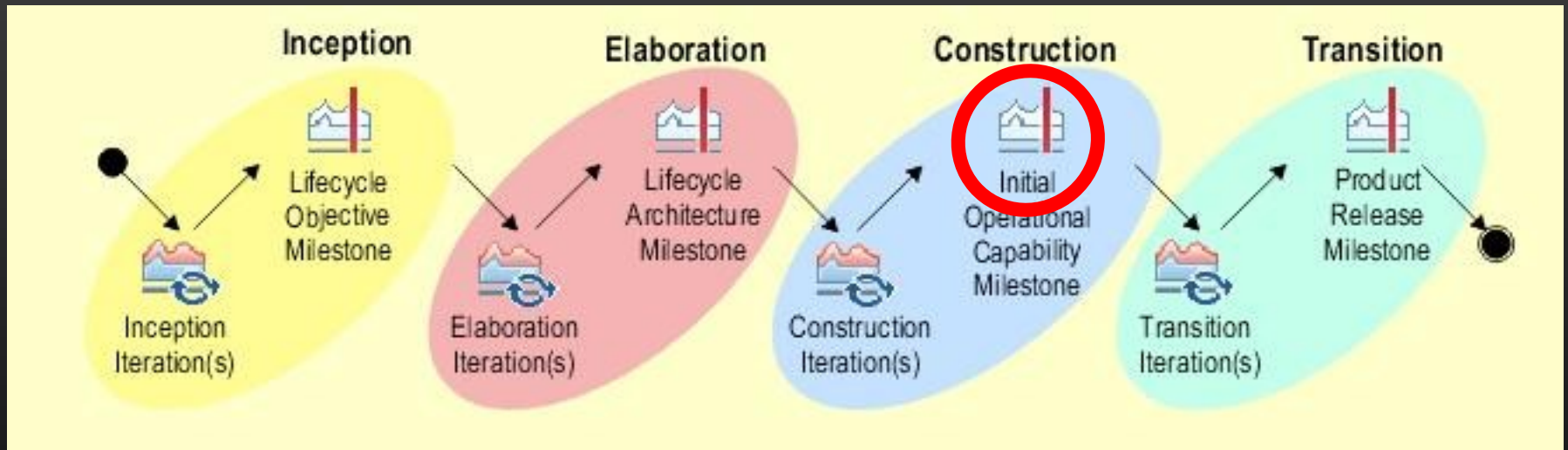
You may have to have a separate test team if you have

Complex test environments

Safety or mission critical systems

Credit: Per Kroll (IBM)

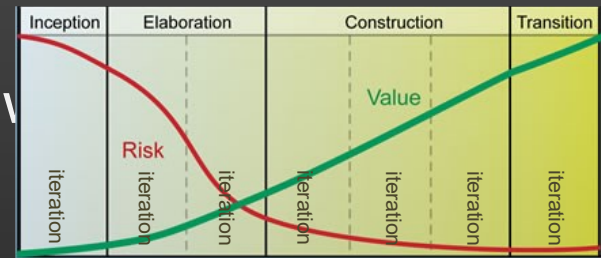
Milestones: Construction



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
stabilizing and tuning



Produce incremental 'bug-fix' releases

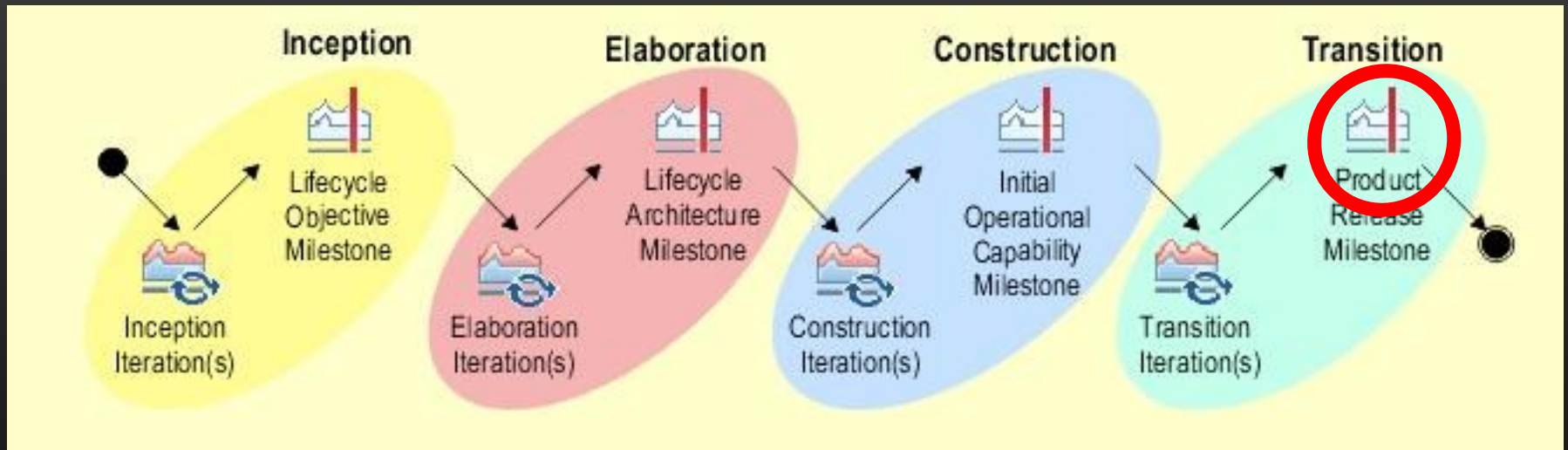
Update user manuals and deployment documentation

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 objective milestone)

Major Milestones



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

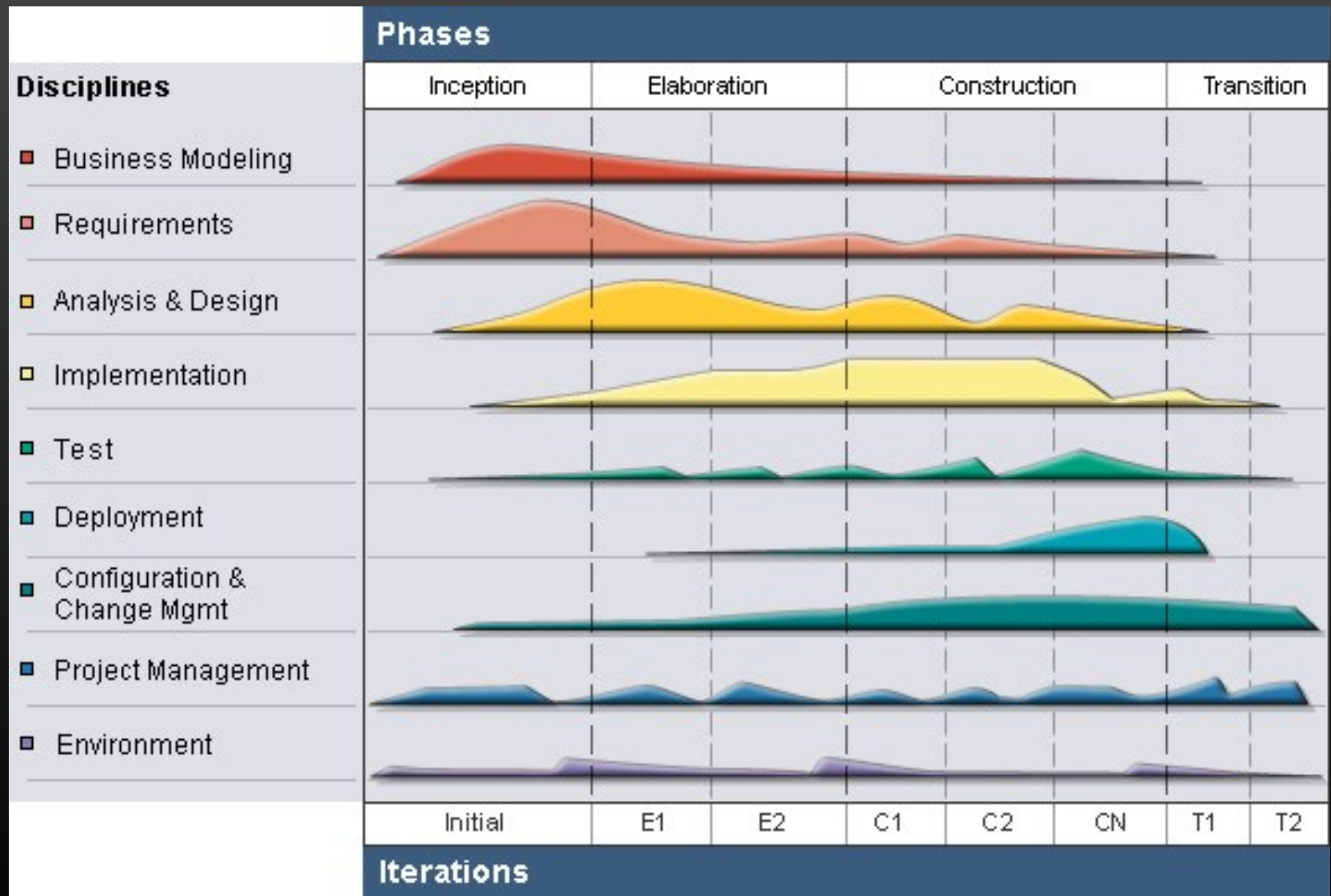
Construction: Agreement on complete operational system

Develop a beta release with full functionality

Transition: Validate and deploy solution

Stakeholder acceptance, cutover to production

Ciclo de vida do Unified Process



Um processo ágil...

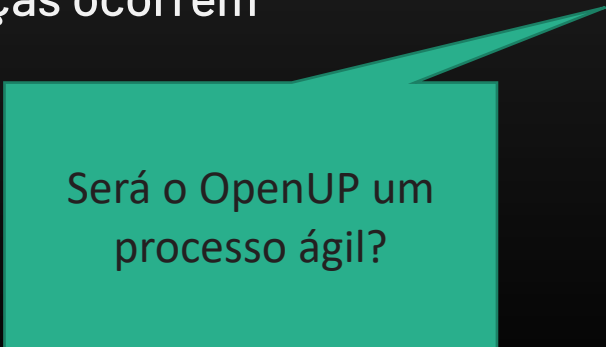
É guiado pelas descrições do cliente do que é necessário (cenários)

Admite que os planos são de curta duração

Desenvolve software iterativamente, com grande ênfase nas actividades de construção

Entrega múltiplos 'incrementos de software'.

Adapta-se à medida que as mudanças ocorrem



Será o OpenUP um processo ágil?

Algumas ideias a reter

- Dinamismo do negócio → dinamismo das metodologias de construção do software
- É preciso adotar metodologias que procurem mitigar o risco
- É preciso adotar a gestão explícita de prioridades
- O planeamento rigoroso “a-priori” é difícil e, geralmente, demasiado inflexível para a natureza dos projetos de sw
- A comunicação e a colaboração são a chave.
- Os métodos ágeis defendem o desenvolvimento evolutivo (iterative + incremental).