

47006- ANÁLISE E MODELÇÃO DE SISTEMAS

AMS: course presentation

Ilídio Oliveira

v2020/10/06

Key resources

Web page at Moodle

All learning materials
Assignments submission

Syllabus (dossier pedagógico)

Subjects covered
Grading (and other) rules

Course Calendar

Weekly plan

I. Oliveira

The screenshot shows a Moodle course page for 'Análise e Modelação de Sistemas' (47006). The page includes the following information:

Informação	Detalhe
código no paco	47006
área científica	Informática / Sistemas de Informação
créditos	8
escolaridade	ensino teórico (T) - 3 horas/semana
idioma(s) de lecionação	Português, Inglês
responsável	Ildio Fernando de Castro Oliveira

objectivos

Análise e Modelação de Sistemas é uma Unidade Curricular (UC) introduzida na área da engenharia de software. Serão apresentados conceitos básicos de análise e projeto de sistemas, requisitos, casos de uso e princípios OO. Tópicos de arquitetura de sistemas e design serão também abordados para suportar a UML e a OpenUP.

A metodologia OpenUP será usada como referência no processo de desenvolvimento de software. Os alunos terão a oportunidade de a seguir e praticar a aplicação de uma solução para um problema concreto. A UML será também apresentada e aplicada ao longo da UC para suportar o processo de modelação.

Mapping AMS in the ACM/IEEE curriculum guidelines



Software Engineering 2014

Curriculum Guidelines for Undergraduate
Degree Programs in Software Engineering

[Resource] Full Guidelines
document @eLearning

I. Oliveira

KA/KU	Title	Hours	KA/KU	Title	Hours
CMP	Computing essentials	152	DES	Software design	48
CMP.cf	Computer science foundations	120	DES.con	Design concepts	3
CMP.ct	Construction technologies	20	DES.str	Design strategies	6
CMP.tl	Construction tools	12	DES.ar	Architectural design	12
			DES.hci	Human-computer interaction design	10
			DES.dd	Detailed design	14
			DES.ev	Design evaluation	3
FND	Mathematical and engineering fundamentals	80	VAV	Software verification and validation	37
FND.mf	Mathematical foundations	50	VAV.fnd	V&V terminology and foundations	5
FND.ef	Engineering foundations for software	22	VAV.rev	Reviews and static analysis	9
FND.ec	Engineering economics for software	8	VAV.tst	Testing	18
			VAV.par	Problem analysis and reporting	5
PRF	Professional practice	29	PRO	Software process	33
PRF.psy	Group dynamics and psychology	8	PRO.con	Process concepts	3
PRF.com	Communications skills (specific to SE)	15	PRO.imp	Process implementation	8
PRF.pr	Professionalism	6	PRO.pp	Project planning and tracking	8
			PRO.cm	Software configuration management	6
			PRO.evo	Evolution processes and activities	8
MAA	Software modeling and analysis	28	QUA	Software quality	10
MAA.md	Modeling foundations	8	QUA.cc	Software quality concepts and culture	2
MAA.tm	Types of models	12	QUA.pca	Process assurance	4
MAA.af	Analysis fundamentals	8	QUA.pda	Product assurance	4
REQ	Requirements analysis and specification	30	SEC	Security	20
REQ.rfd	Requirements fundamentals	6	SEC.sfd	Security fundamentals	4
REQ.er	Eliciting requirements	10	SEC.net	Computer and network security	8
REQ.rsd	Requirements specification and documentation	10	SEC.dev	Developing secure software	8
REQ.rv	Requirements validation	4			

Course subject: analysis and specification of software-intensive systems

Systems analysis

Disciplines related to the characterization of the problem and specification of the technical solution

Development Process

Systematic engineering method. Defines activities, roles and outcomes

Visual modeling

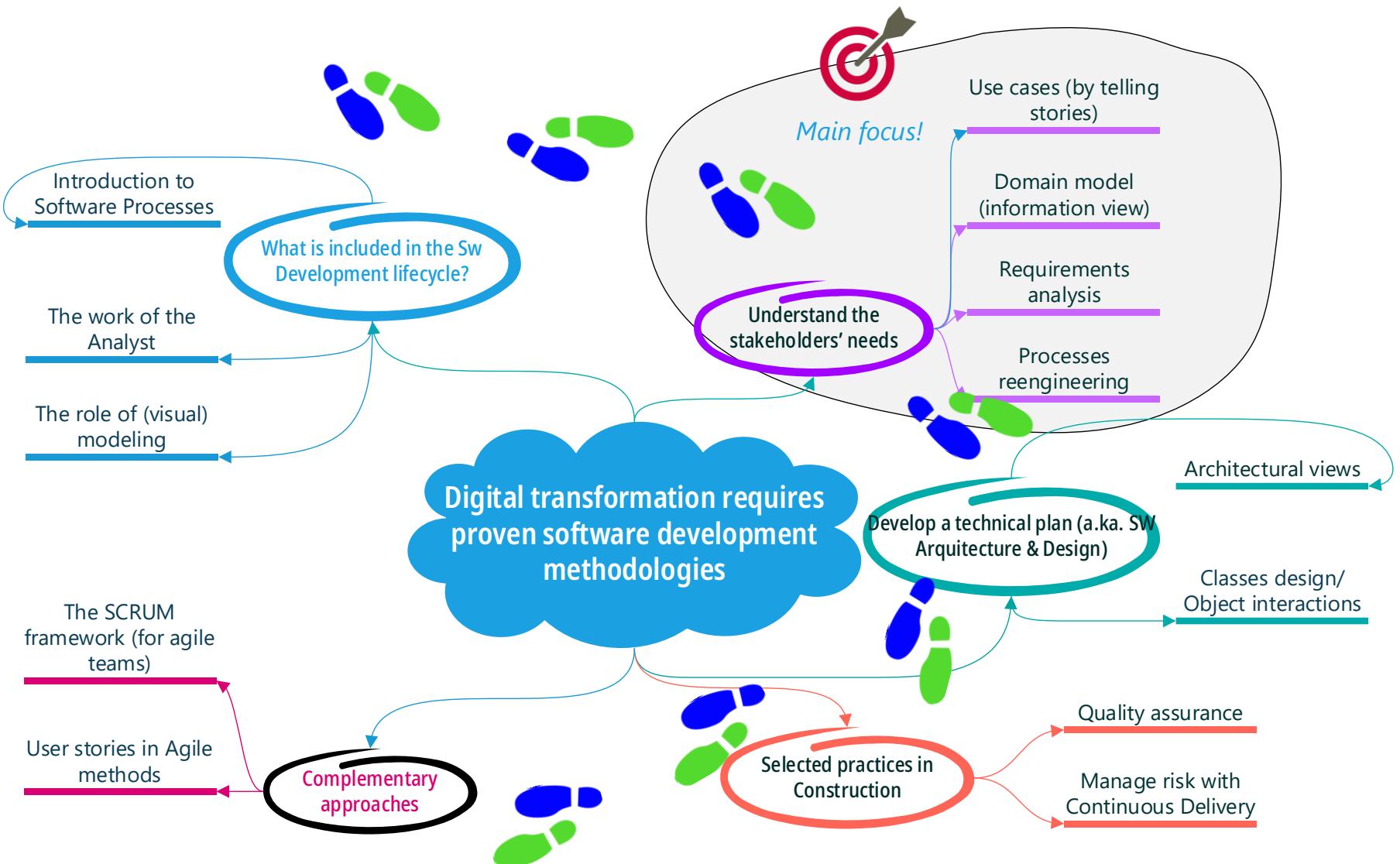
Unified Modeling Language - UML

CASE tools (computer-aided software engineering)

E.g.: VisualParadigm

1 Re	2 Ra	3 Ar	4 Dc	5 Bcs	6 Sa	7 Ut	8 Rca	9 At	10 Exm	11 Ri
Requirements elicitation	Requirements analysis	Atomic Requirements	Requirements Design	Basic Coding Skills	Static Code Analysis	User Testing	Root Cause Analysis	Code Analysis Tools	Exception Management	Risk Analysis
Design	Database Design	Design Patterns	Design Patterns	Infrastructure Basics	Testing Usability	Unit Testing	Defect Root Cause Analysis	Automated Code Analysis Tools	Performance Management	Requirements Management
Lean IT				Implementation Basics	Tools	Integration Testing	User Interface Design	Continuous Integration Tools	Configuration Management	Requirements Management Tools
Maintenance				Code Analysis	Management	Code Refactoring	Dynamic Code Analysis	Requirements Management Tools	Project Management Tools	Requirements Management Basics
19 Rt	20 Dp	21 Sc	22 Rg	23 Bi	24 Bo	25 Ad	26 Ol	27 Scb	28 Scc	29 Pac
Requirements Attributes	Design Patterns	Scrum	Re-engineering	Basics of ITIL	Big-O Notation	Algorithm Design	Object Oriented Languages	Software Security Basics	Scientific Computing	Numerical Mathematics
37 Rr	38 Ap	39 Ka	40 Rv	41 Do	42 Bm	43 Ds	44 Fl	45 Eb	46 Dbs	47 Gat
Requirements Reviews	Architecture Patterns	Kaizen	Reverse engineering	DevOps	Build Management	Data Structures	Functional Languages	Encryption Basics	Databases	Game Theory
55 Trm	56 Lsd	57-71	72 Pc	73 Mo	74 Ade	75 Aop	76 Di	77 Np	78 Sma	79 Pac
Task Management	Large-scale System Design	Agile Methodologies	Project Comprehension	Monitoring	Automation	Aspect Oriented Programming	Declarative Languages	Network Protocols	Smart Machines	Parallel Computing
87 Rem	88 Dn	89-103	104 Mp	106 Icm	105 Tdm	107 Dc	108 Pl	109 W's	110 Mi	111 Ai
Management of Requirements Portfolios	Design Notations	Soft Skills	Maintenance Planning	IT Change Management	Data Management	Procedural Languages	Protocol Computing	Web Application Security	Machine Learning	Artificial Intelligence
57 Agp	58 Pp	59 Td	60 Dd	61 Cd	62 Cy	63 Us	64 Bam	65 Sm	66 Sp	67 Pg
Agile Planning	Pair Programming	Test Driven Development	Definition of Done	Continuous Integration	Cloud Computing	User Stories	Bamboo	Stand-up Meeting	Space Solutions	Planning Game
89 Prs	90 Ts	91 Em	92 Crr	93 Crm	94 Ns	95 Rh	96 Is	97 Crt	98 Ma	99 Lea
Presentation Skills	Training Skills	Empathy	Creation of Relationships	Conflict Management	Negotiation Skills	Rhetoric	Intercultural Skills	Creativity Techniques	Marketing Basics	Leadership Basics
90 No	91 Im	92 Cr	93 Cm	94 Ns	95 Rh	96 Is	97 Crt	98 Ma	99 Lea	100 Go
No Overstaying	Imtrinsic Motivation	Creation of Relationships	Conflict Management	Negotiation Skills	Rhetoric	Intercultural Skills	Creativity Techniques	Marketing Basics	Leadership Basics	Good Manners
101 Co	102 Phf	103 St	104 Cm	105 Im	106 St	107 Ti	108 Sr	109 St	110 Pph	111 St
Collective Code Ownership	Physical Fitness	Stop Talking	Computer Modeling	Intrinsic Motivation	Standup Meetings	Team Light	System Monitoring	Stop Talking	Project Controlling	System Monitoring

<http://www.sw-engineering-candies.com/blog-1/periodic-table-of-software-engineering-know-how>



Note on cooperative learning

Team effort

Labs

Project assignment

Individual

Written exam

More on [Cooperative Learning.](#)

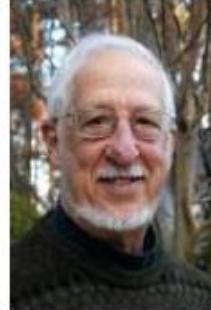
[Resource] Turning groups into teams @eLearning

COOPERATIVE LEARNING It leads to more and deeper learning and longer retention of information; greater development of high-level thinking, problem-solving, communication, and interpersonal skills; more positive attitudes toward engineering and science curricula and careers and greater retention in those curricula; and better preparation for the workplace.



Richard Felder
Engineer

Richard M. Felder is the Hoechst Celanese Professor Emeritus of Chemical Engineering at North Carolina State University. [Wikipedia](#)



How to study for AMS?

Attend the classes ;)

All topics in the Exam are addressed in classes, including some viewpoints/discussion questions.

Books

See references cited at the end of each presentation to find the relevant Chapters (from selected references)

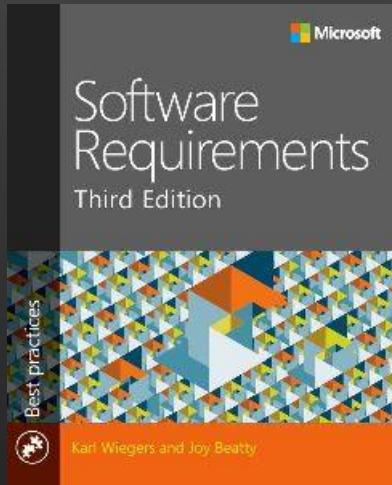
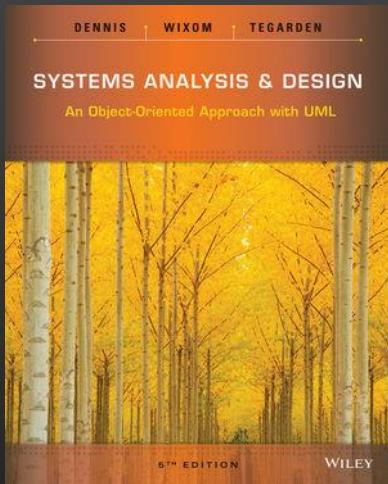
Labs & project

Actively participate in every assignment.

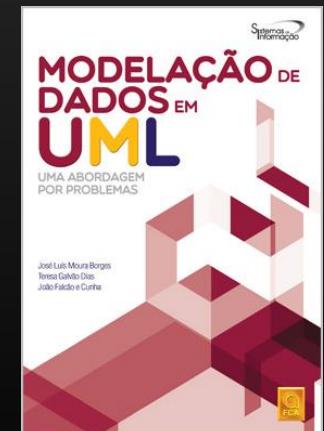
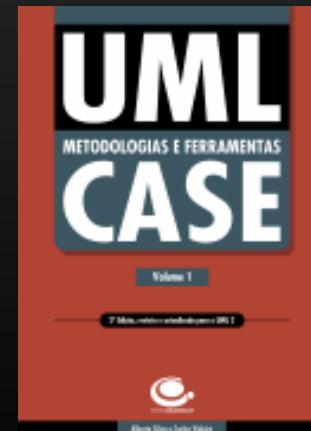
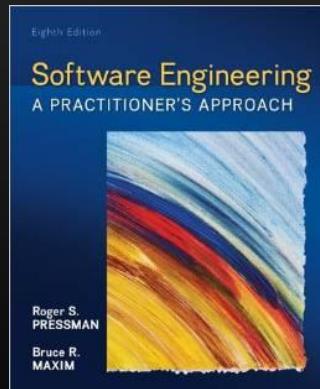
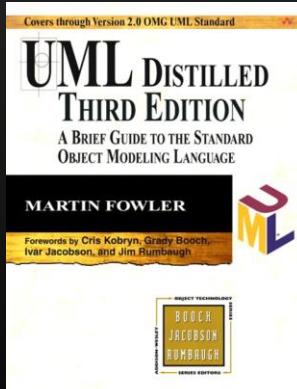
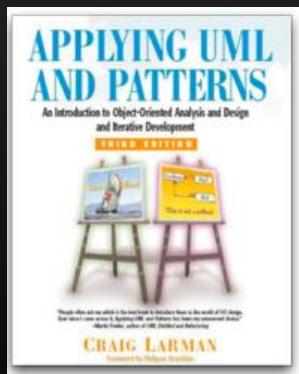
Pitfalls

- ✖ distribute the tasks and cut the discussion in lab assignment... everybody shoud go through the work.
- ✖ skip rotative “roles” in the group
- ✖ let the “smart volunteer” take all the responsibilities

Main references



Adopted text book.



47006- ANÁLISE E MODELÇÃO DE SISTEMAS

Digital transformation of the organizations and society (and its impact in software engineering)

Ilídio Oliveira

v2020/10/07

Learning objectives for this lecture

Describe cases of digital transformation of businesses

Identify three main axes in digital transformation

Explain what is the Systems Development Life Cycle

Explain how the business environment brings opportunities and risks to the software industry

Digital transformation and the strategic role of IS

Technological change has never occurred as rapidly, or on as large a scale, as today.

"Technological innovation enables – indeed, requires – companies to boost their agility and thus their competitiveness. That's why CEOs' top priorities in 2016 should be to digitize the core components of their business and rethink organizational design and governance processes. Catching this fast-moving – and rapidly growing – "digital wave" is the only way to avoid getting left behind."

PROJECT ■ SYNDICATE

PRINT



DOMINIC BARTON

Dominic Barton is the global managing director of McKinsey & Company.

JAN 15, 2016

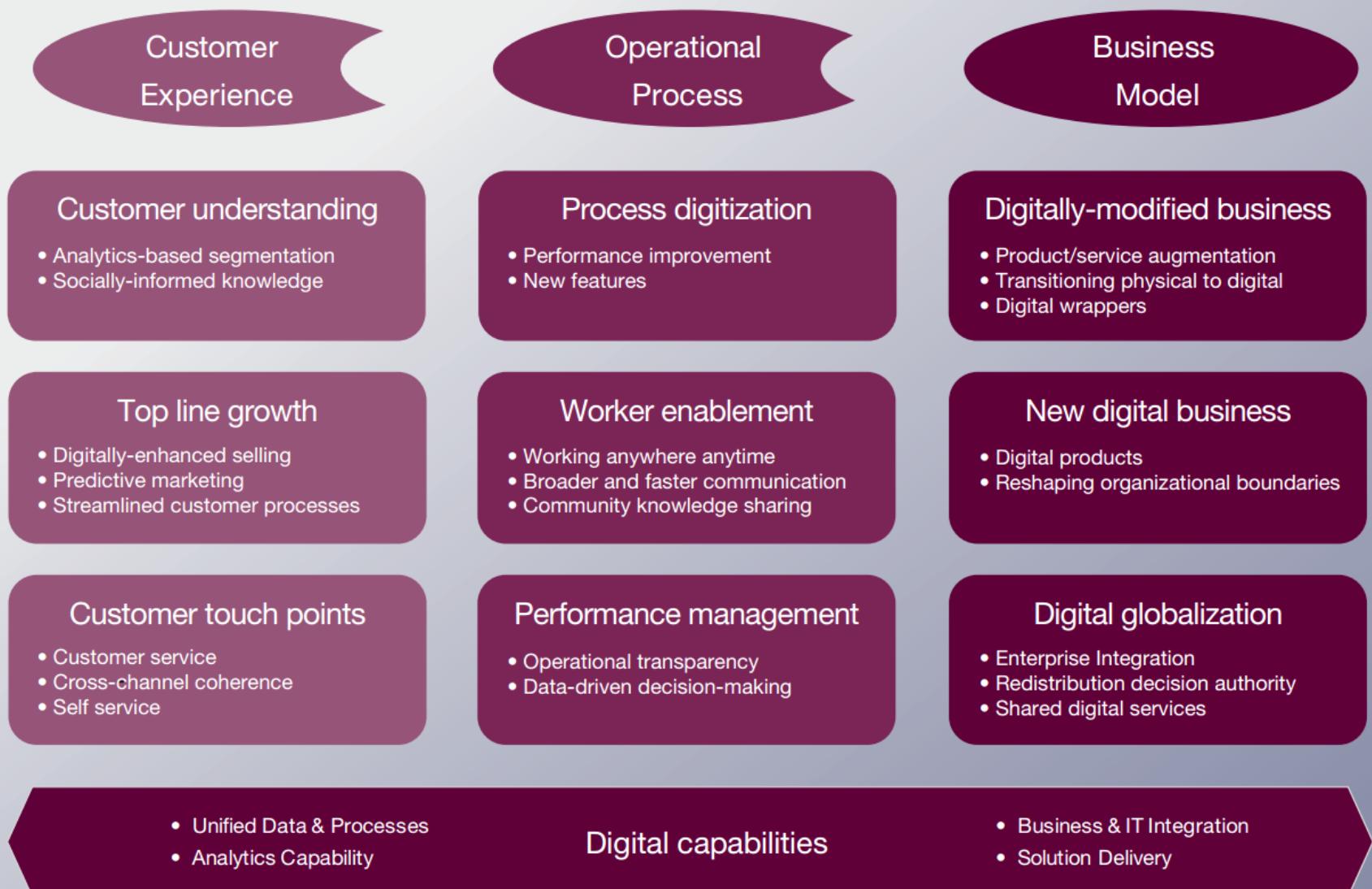
Catching the Digital Wave

NEW YORK – Technological change has always posed a challenge for companies. But, as we saw once again in 2015, it has never occurred as rapidly, or on as large a scale, as today. As innovation sweeps across virtually every sector, from heavy industry to services, it is transforming the competitive landscape, with the most advanced companies – rather than the largest or most established players – coming out on top.

For incumbents, the threat of displacement is very real. The average tenure of a company on the S&P 500 has fallen from 90 years in 1935 to less than 18 years today. Disruptive new players like Uber, which has upended the taxi industry, are tough competitors, often staking out market share by shifting more surplus to consumers. This is part of a broader trend of intensifying competition that, according to recent research from the McKinsey Global Institute, could reduce the global after-tax profit pool from almost 10% of global GDP today to its 1980 level of about 7.9% within a decade.

<http://prosyn.org/IxXI6OW>

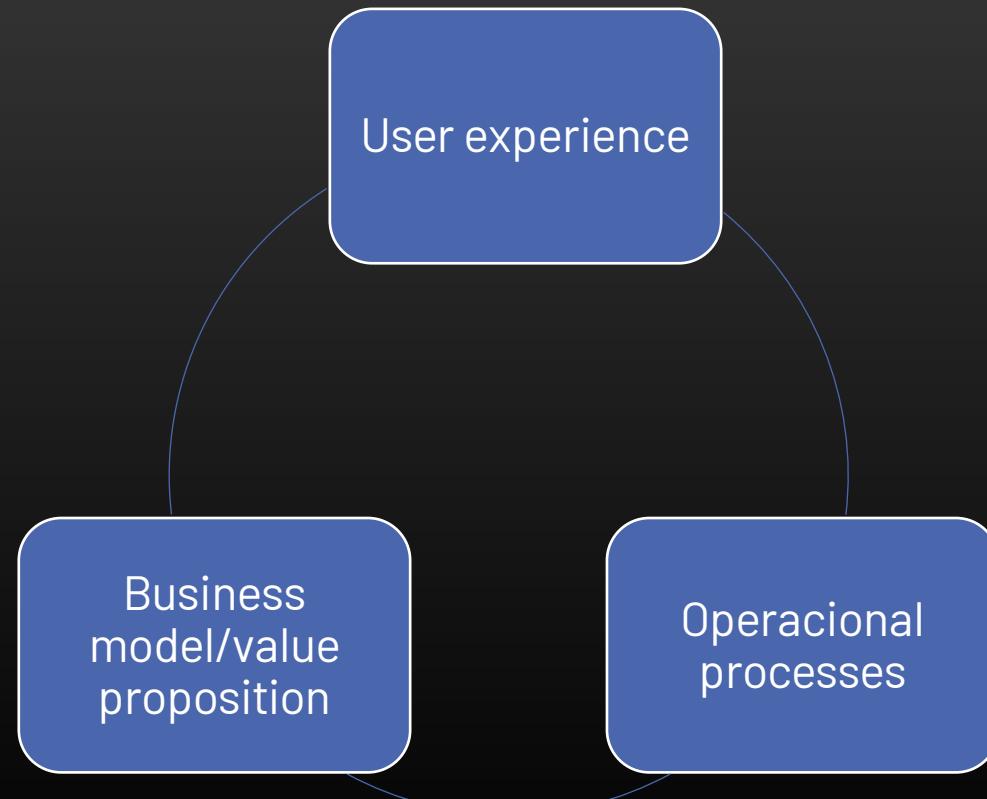
Figure 3: Building blocks of the digital transformation



“Digital Transformation: A Roadmap for Billion-Dollar Organizations”,
MIT Center for Digital Business

Digital transformation

The use of ICT to decisively improve the performance or value proposition of a company





Caixadirecta

O SEU BANCO SEMPRE DISPONIVEL.

Tudo o que precisa, onde estiver, como quiser.
Adesão gratuita.

[SAIBA MAIS >](#)

APP CAIXADIRECTA



UMA EXPERIÊNCIA ÚNICA
DE ACESSO SIMPLES E CÓMODO
AO SEU BANCO.

JÁ DISPONÍVEL PARA TABLET E SMARTPHONE IOS E ANDROID.

INOVAR NA CAIXA. COM CERTEZA.



SERVIÇO CAIXAUTOMÁTICA



Serviços Disponíveis[Página Principal](#)**Alunos**[Secretaria Virtual](#)[Horários de Turmas do 1ºAno](#)[Horários para 2014/2015](#)[Creditações OnLine](#)[Matrículas OnLine](#)[Candidaturas](#)[Candidaturas M23](#)[Candidaturas Especiais](#)[Candidaturas CET](#)[Candidatura Cursos Livres](#)[Candidaturas EI/internationalstudent](#)**Docentes**[Disciplinas](#)**[Secretaria Virtual para Estudantes da UA](#)**

Sistema de apoio aos **estudantes**, servindo de extensão à Secretaria dos Serviços Académicos.

**SGQ**

A partir de 26 de Janeiro, a Universidade de Aveiro (UA) implementa o Subsistema para a Garantia da Qualidade das Unidades Curriculares relativo ao 1º semestre do ano letivo 2014/2015.

A partir dessa data e até ao dia 22 de fevereiro, a UA promoverá o lançamento dos inquéritos pedagógicos junto dos estudantes. Os inquéritos são preenchidos eletronicamente, via PACO (<http://paco.ua.pt/>) ou diretamente em <http://sgq.ua.pt>.

Participa! A tua opinião é fundamental!

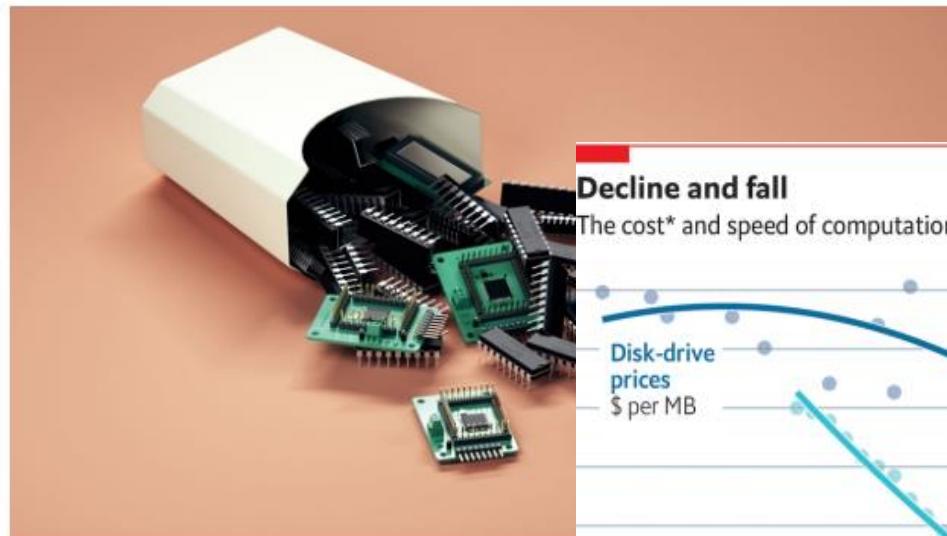
U B E R



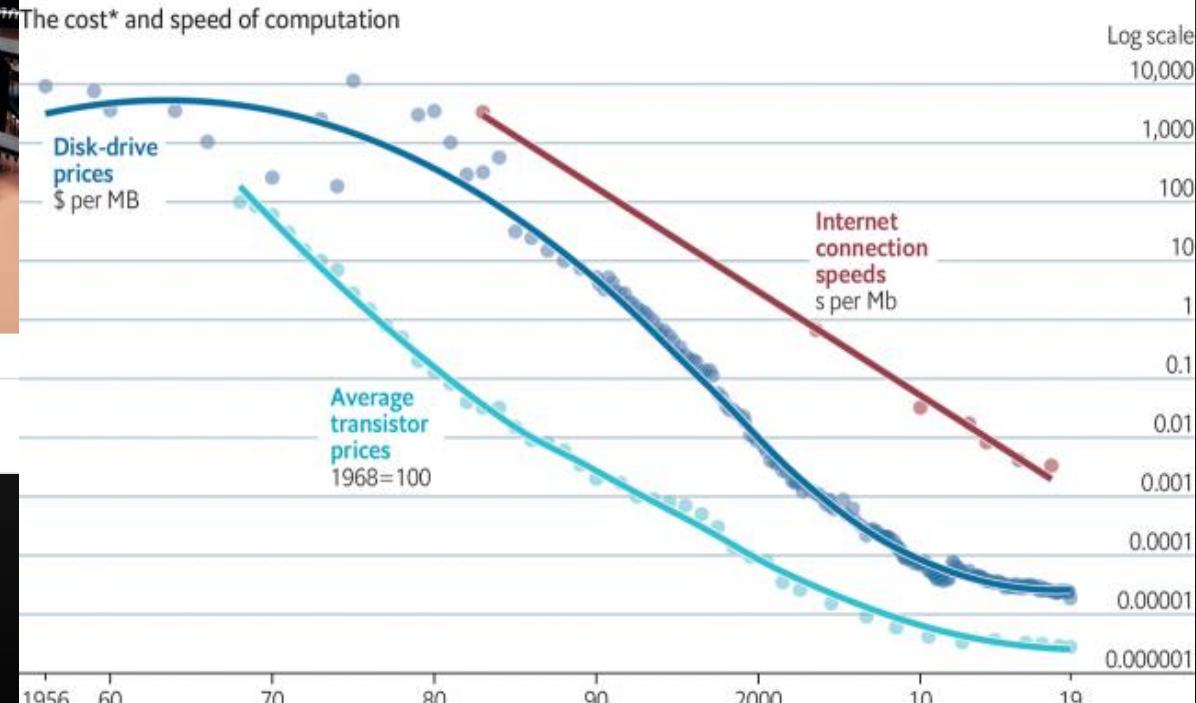
Ubiquitous computing

Drastic falls in cost are powering another computer revolution

The Internet of Things is the next big idea in computing



Decline and fall
The cost* and speed of computation



Print edition | Technology Quarterly >
Sep 12th 2019

Technology evolved ...

THE EVOLUTION OF TECHNOLOGY & Its Impact on the Development of Social Businesses



We are babies.

1960s

Technology has **little impact**.
It is a curiosity.

The company is king, but a benevolent king. Good focus on customer satisfaction, but customers have few options. Communications makes global business difficult so customers make geographic-based decisions.



We are still children.

1970s

Technology is for academics and has **little impact**.

Greater focus on margins and revenue. Customers become concerned about monopolies as customer satisfaction has less importance.



We are still children,
but we can **put out** what we want.

1980s

Technology invades the home and starts to **change behaviors**.

Customers become increasingly concerned about company practices and lack of customer satisfaction. Communications have improved to help customers make more informed decisions and to have better choices.



Like teenagers, we
now have some
control but don't know
what to do with it yet.

1990s

Technology is now everywhere. A great leap forward. It begins to **connect us** around the globe.

e-Commerce helps give customers a greater - and more informed - range of decisions. Companies use the web to make themselves more accessible but haven't begun truly focusing on customer relationships.



We are growing up, and
feeling pretty cool about it.

2000s

Technology enables more seamless communications across the globe. Growth is **explosive**, but like "explosions" is uncontrolled - all over the place.

Social Media allows customers to articulate their satisfaction with companies and make decisions based on the company's behavior, not just on price alone. Companies begin to react and change.



Welcome to adulthood!
2010s

Technology becomes **fully integrated** into our daily lives. We live more fully in a digital world.

Social Businesses are the evolution of companies now keenly aware that how they act and how they engage with customers can be more important than price, that the relationship is part of the value. Companies allow greater transparency into all aspects of the company and use social media channels to effectively engage with customers, but with a focus on WHAT the customer wants and HOW best to deliver it to the customer.

Systems development life-cycle

In 2015

MODERN RESOLUTION FOR ALL PROJECTS

	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

The Modern Resolution (OnTime, OnBudget, with a satisfactory result) of all software projects from FY2011-2015 within the new CHAOS database. Please note that for the rest of this report CHAOS Resolution will refer to the Modern Resolution definition not the Traditional Resolution definition.

Integrated Requirements Engineering: A Tutorial

Ian Sommerville, Lancaster University

Before developing any system, you must understand what the system is supposed to do and how its use can benefit the individuals or business that will pay for it. This involves understanding the application domain (airlines, railways, retail banking, games, and so on); the constraints; the specific functionality required by the stakeholders (the people who directly or indirectly use the system or the information it produces); and essential system characteristics such as performance, security, and dependability. *Requirements engineering* is the name given to a structured set of activities that help develop this understanding and that document the system specification for the stakeholders and engineers involved in the system development.

This short tutorial introduces the fundamental activities of RE and discusses how it has evolved as part of the software engineering process. However, rather than focus on established RE techniques, I discuss how the changing nature of software engineering has led to new challenges for RE. I then introduce a number of new techniques that help meet these challenges by integrating RE more closely with other systems implementation activities.

This tutorial introduces the fundamental activities of requirements engineering and discusses recent developments that integrate it and system implementation.

The fundamentals The RE process depends on the type of system being developed, the size and complexity involved, and the software used. For large military systems, there is normally a formal process that documents the system requirements. For smaller systems, the systems engineer might simply be a short vision of what software is expected to do.

Whatever the activities are, they are fundamen-

- **Elicitation.** Identify what the system needs to do and what requirements from the stakeholders are needed.
- **Analysis.** Understand the requirements, their overlaps, and their conflicts.
- **Validation.** Go back to the system stakeholders to verify the requirements.

■ *The need for rapid software delivery.* Businesses now operate in an environment that's changing incredibly quickly. New products appear and disappear, regulations change, businesses merge and restructure, competitors change strategy. New software must be rapidly conceived, implemented, and delivered. There isn't time for a prolonged RE process. Development gets going as soon as a vision for the software is available, and the requirements emerge and are clarified during the development process.

Towards an engineering process

Why do we need a formal process?

Failures occur (too) often

Creating systems is not intuitive

Projects are late, over budget or delivered with fewer features than planned

Systems development life-cycle (SDLC)

the process of determining how an information system (IS) can support business needs by designing a system, building it, and delivering it to users.

Role of the Analyst

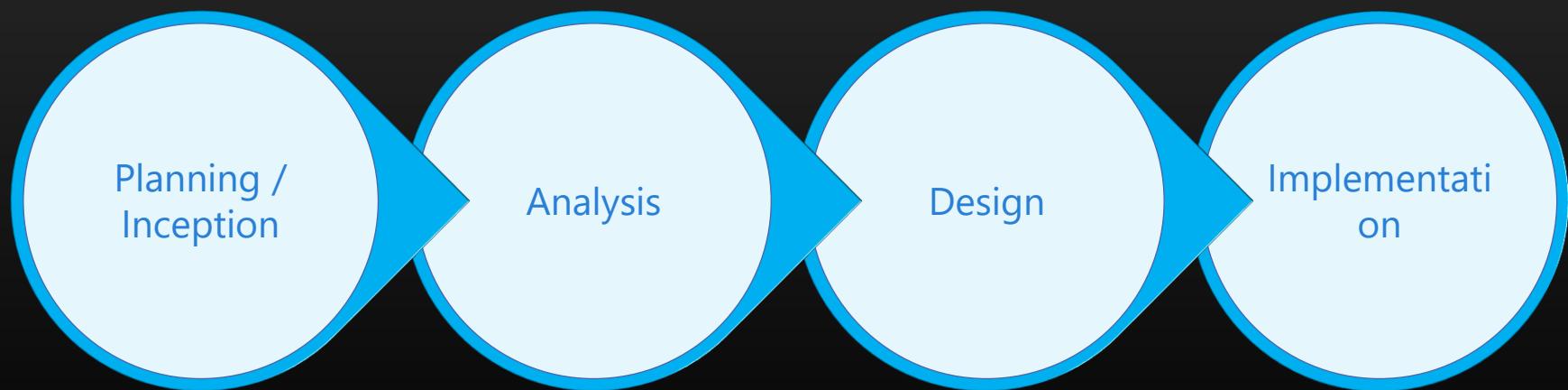
The key person in the SDLC is the systems analyst, who analyzes the business situation, identifies opportunities for improvements, and designs an information system to implement them. Being a systems analyst is one of the most interesting, exciting, and challenging jobs around.

The primary objective of a systems analyst is not to create a “awesome system”; instead, it is to create value for the organization.

SDLC phases

Four fundamental phases: planning, analysis, design, and implementation. Different projects might emphasize different parts of the SDLC or approach the SDLC phases in different ways, but all projects have elements of these four phases.

Each phase is itself composed of a series of steps, which rely upon techniques that produce deliverables (specific documents and files that provide understanding about the project).



The SDLC is instantiated in a specific development methodology

What is included in a “process”?



Core
Principles



Roles



Work Products



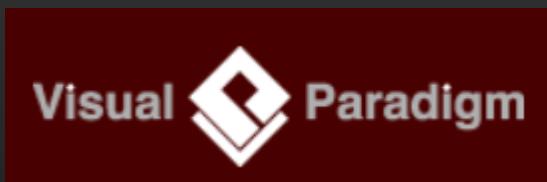
Disciplines



Lifecycle

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

Ferramentas CASE



Readings & references

Core readings	Suggested readings
<ul style="list-style-type: none">• [Dennis15] - Chap. 1	<ul style="list-style-type: none">• [Pressman14] - Chap. 1• “<u>Catching the digital wave</u>”• “<u>Digital Transformation: A Roadmap for Billion-Dollar Organizations</u>”, MIT Center for Digital Business

47006- ANÁLISE E MODELAÇÃO DE SISTEMAS
1st Semester, 2018/19

Visual Modelling with UML

Ilídio Oliveira | 2020/10/09

Learning objectives for this lecture

Justify the use of models in systems engineering

Enumerate advantages of visual models

Explain the organization of the UML

Identify the main diagrams in UML and their modeling viewpoint

Read and create Activity Diagrams

The SDLC is applied through software development processes

What is in a process?



Core
Principles



Roles



Work Products



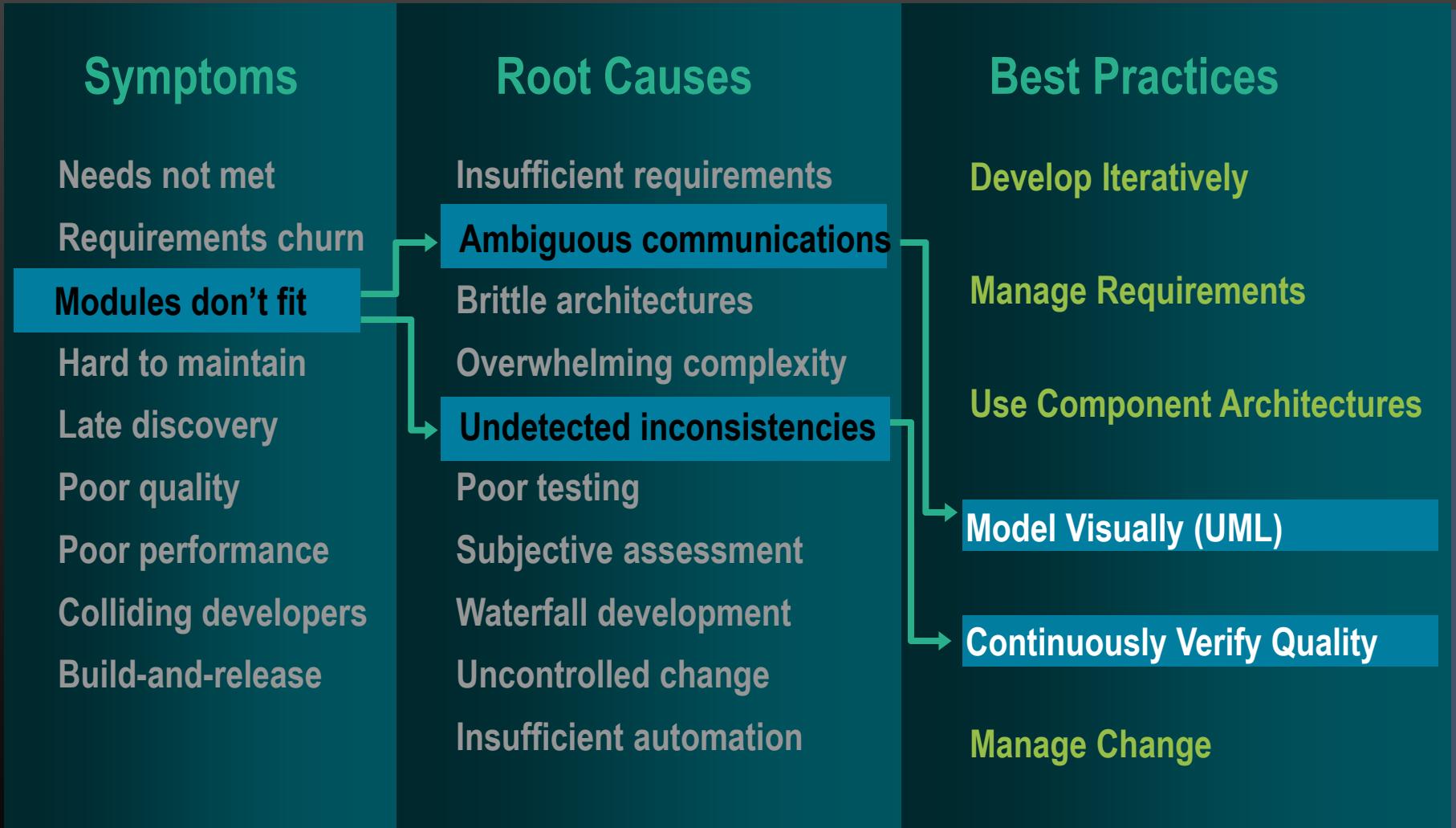
Disciplines



Lifecycle

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

Problems and solutions in the SDLC (a Rational Unified Process perspective)



Modeling

UML as a visual specification language

Usamos modelos visuais para captar partes do mundo/realidade

D Trumpet Version

Allegro Assai
from
Brandenburg Concerto #2

J. S. Bach
arranged by Mark Adler

Trumpet

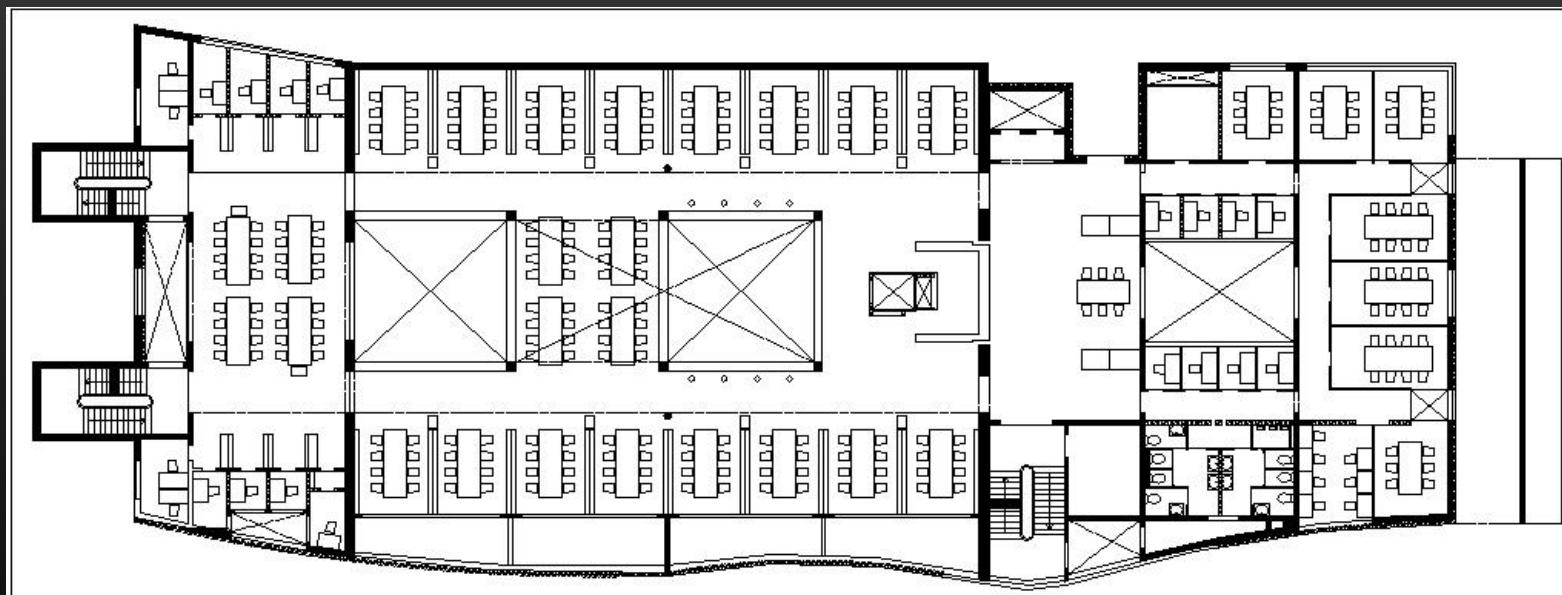
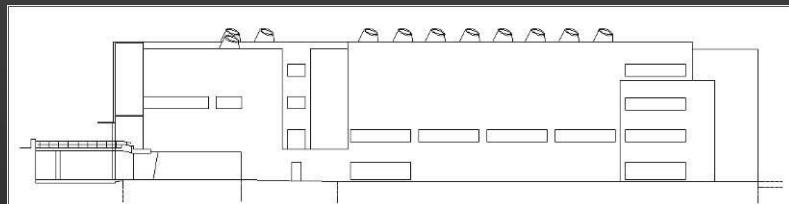
Organ



- Uma linguagem comum (escrever, ler)
- Especificações visuais são mais inteligíveis
- Compor: aplicar talento e disciplinas técnicas
- Orquestra: a prova que os modelos funcionam!



Um modelo é uma simplificação da realidade



Os modelos ajudam a gerir a complexidade

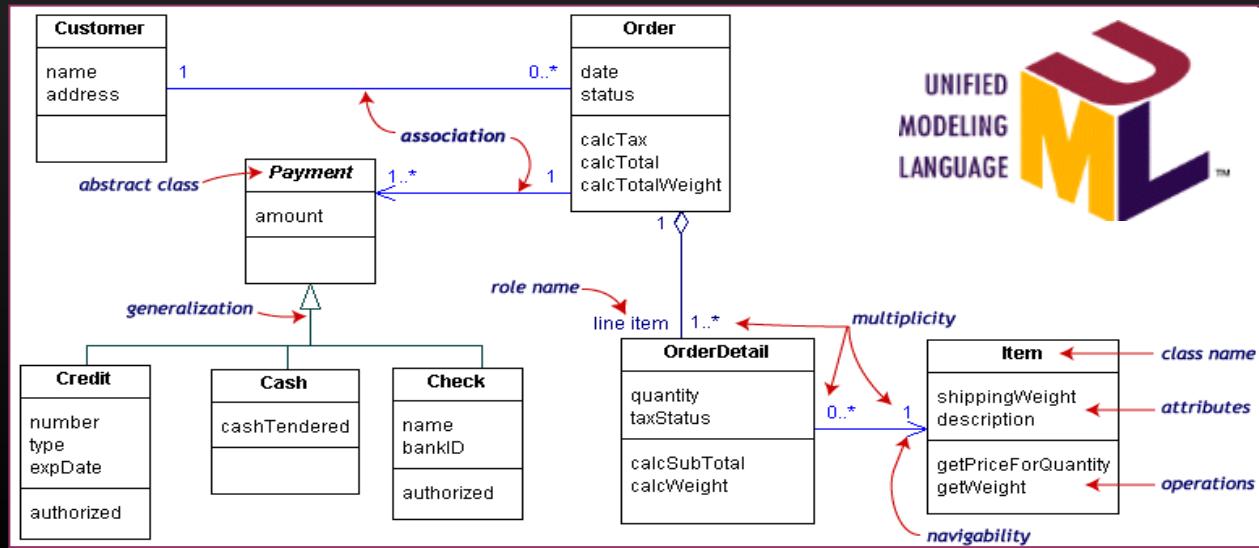
4 razões para usar modelos (G. Booch):

Ajudar a visualizar um sistema (*high-level*)

Especificar/documentar a estrutura e o comportamento do sistema
(antes de implementar)

Serve como referência para orientar construção ("planta")

Documentar as decisões (de desenho) que foram feitas



Modelação visual no desenvolvimento

UML 2: Unified Modeling Language

Linguagem de modelação normalizada

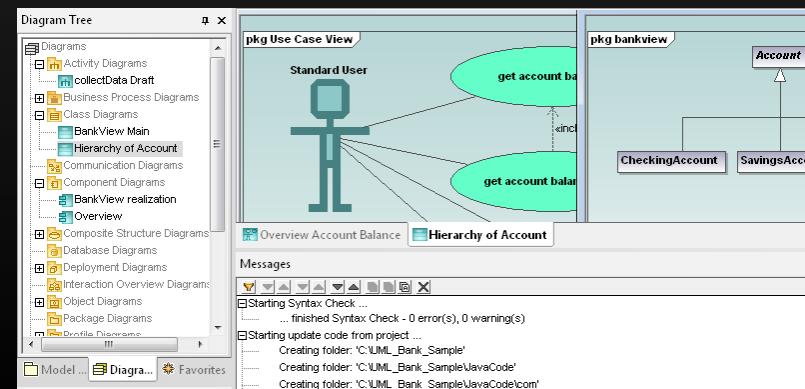
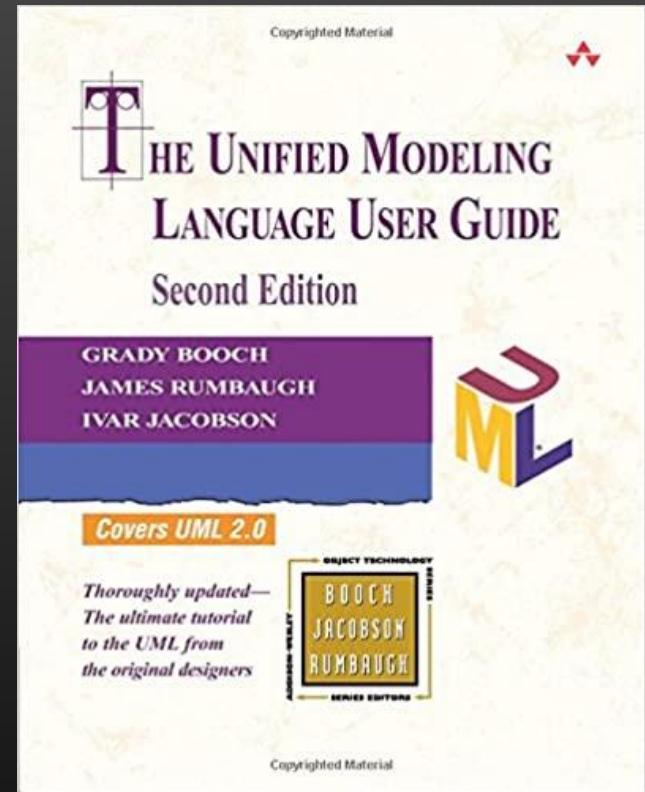
Benefícios

Promover a comunicação mais clara e sucinta

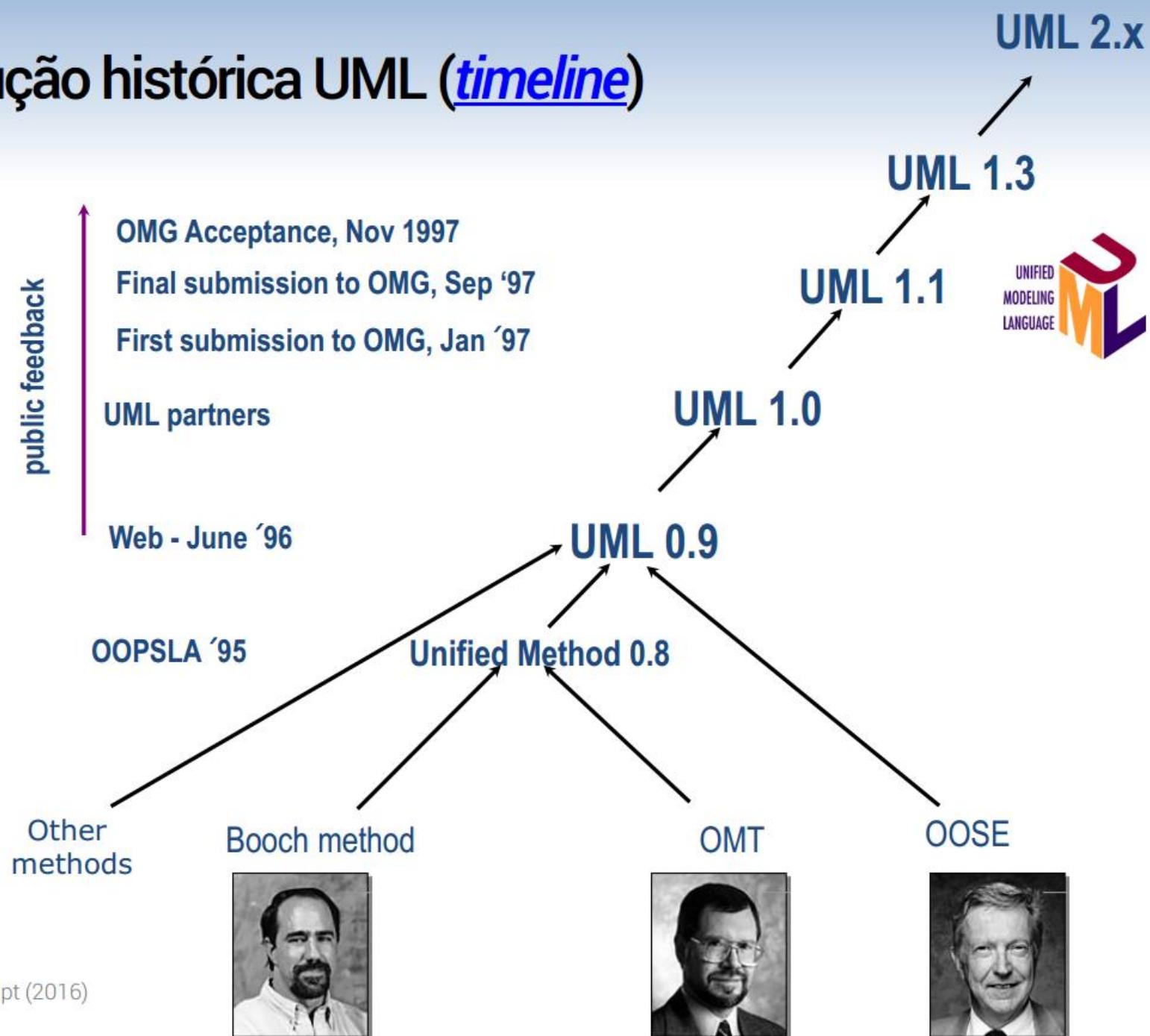
Manter o desenho (planeamento) e a implementação (construção) coerentes

Mostrar ou esconder diferentes níveis de detalhe, conforme apropriado

Pode suportar, em parte, processos de construção automática (gerar a solução a partir do modelo)



Evolução histórica UML (*timeline*)



UML é uma especificação do OMG

[ABOUT US](#)[RESOURCE HUB](#) ▾[OMG SPECIFICATIONS](#) ▾[PROGRAMS](#) ▾[MEMBERSHIP](#) ▾[MEMBERS AREA](#) ▾

ABOUT THE UNIFIED MODELING LANGUAGE SPECIFICATION VERSION 2.5.1

2.5.1 • UML • SPECIFICATIONS

UML®**Unified Modeling Language**

A specification defining a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems.

Title: Unified Modeling Language

Acronym: UML®

Version: 2.5.1

Document Status: formal ⓘ

Publication Date: December 2017

Categories: [Modeling](#) [Software Engineering](#)



Specification

Também reconhecida como um standard internacional ISO

The screenshot shows the ISO website's header and navigation. The top navigation bar includes links for Standards, About us, Standards Development, News, Store, and Members area. Below this is a secondary navigation bar with links for Standards catalogue, Online collections, and Graphical symbols. The ISO logo is on the left. The breadcrumb trail at the bottom of the header indicates the current page path: ISO Store > Store > Standards catalogue > By TC > JTC 1 Information technology > SC 7.

ISO/IEC 19505-1:2012[®]

Information technology -- Object Management Group Unified Modeling Language (OMG UML) -- Part 1: Infrastructure

Abstract

[Preview ISO/IEC 19505-1:2012](#)

ISO/IEC 19505-1:2012 defines the Unified Modeling Language (UML), revision 2. The objective of UML is to provide system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes.



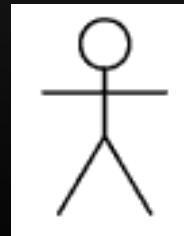
Aplicações principais da UML

Análise e desenho de sistemas de software

Estrutura e comportamento de sistemas baseados em software

- Elementos do modelo representam entidades do mundo do software

Especialmente adequada para o desenvolvimento por objetos (***object-oriented***)



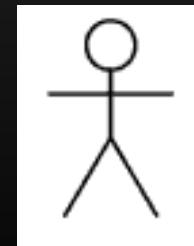
Analista

Domínio do problema (processos de trabalho,...)

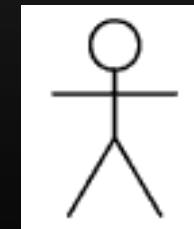
Especificar ou documentar o domínio de aplicação/negócio

- Elementos do modelo representam entidades do negócio

Não implica ou assume uma implementação em software

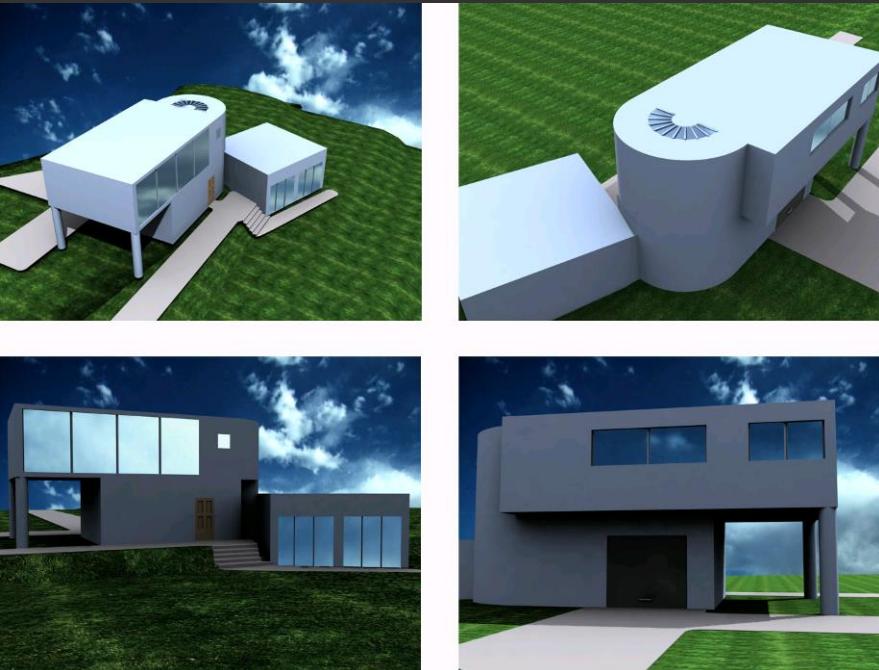


Programador



Arquiteto

Não há uma vista única, mas várias e complementares



Para que serve o sistema?

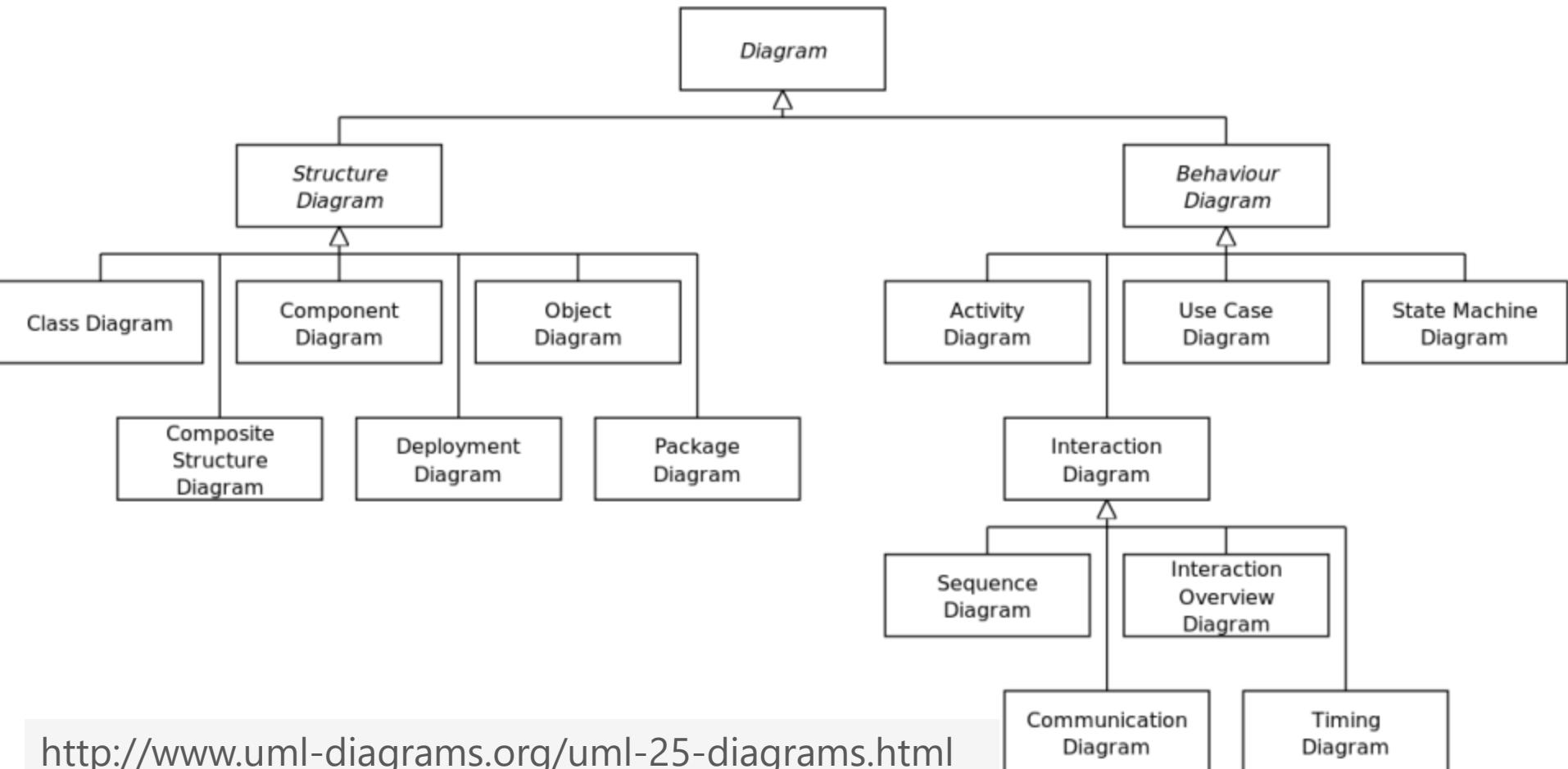
Quais são as estruturas de informação?

Decomposição funcional de atividades complexas

Visualizar a organização do software em partes e as suas interações

Etc.

Diagramas da UML 2.x

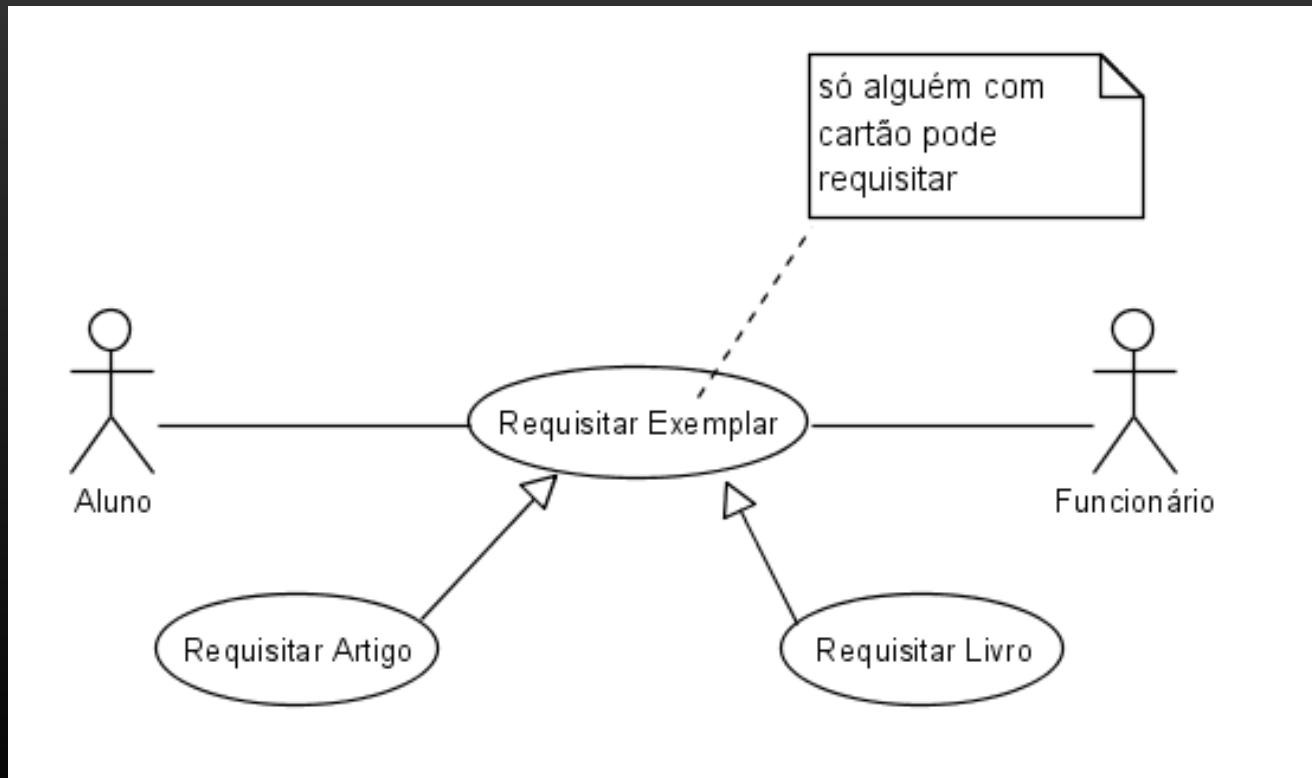


<http://www.uml-diagrams.org/uml-25-diagrams.html>

Elementos comuns

Anotações

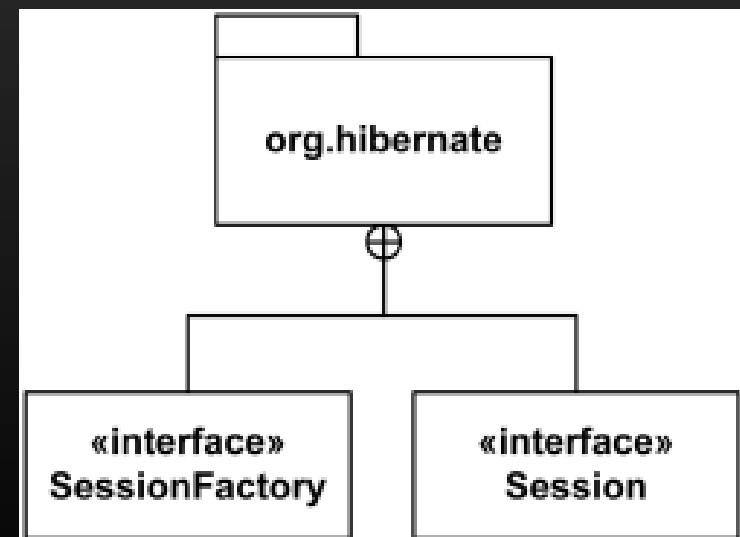
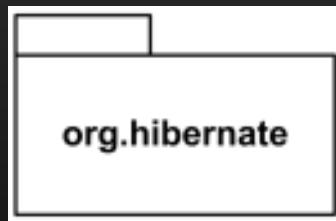
Um comentário que pode ser usado para anotar qualquer elemento



Pacotes

um mecanismo para dividir um modelo em partes

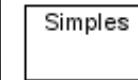
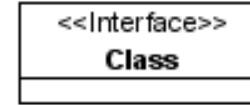
serve como mecanismo genérico para fazer agrupamentos



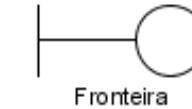
Estereotipo (*stereotype*)

uma especialização da semântica de um elemento do modelação

marcada com «...» ou com a alteração da decoração



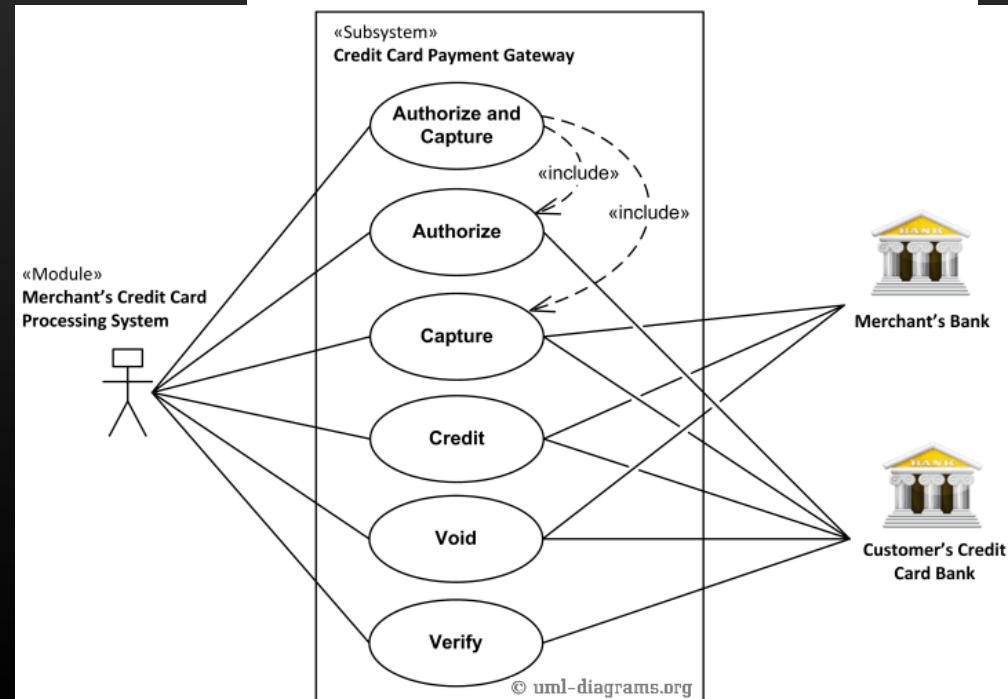
Simples



Fronteira



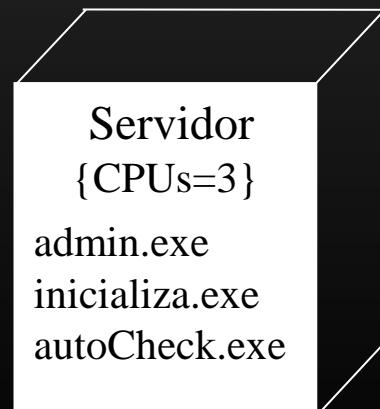
entidade



Valores etiquetados (*tagged values*)

Estender elementos do modelo
com uma linguagem
“computável” (pares
atributo/valor)

```
«Computer»  
{Vendor = "Acer",  
CPU = "AMD Phenom X4",  
Memory = "4 GB DDR2"}  
Aspire X1300
```

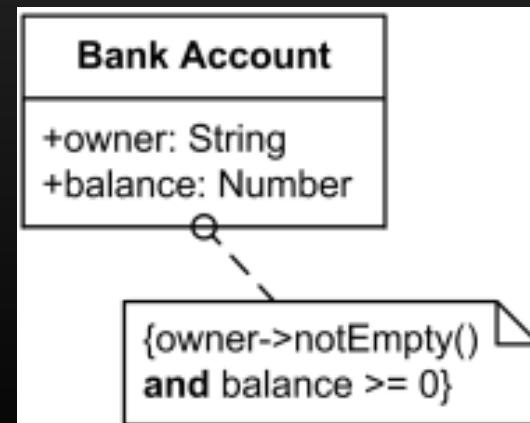
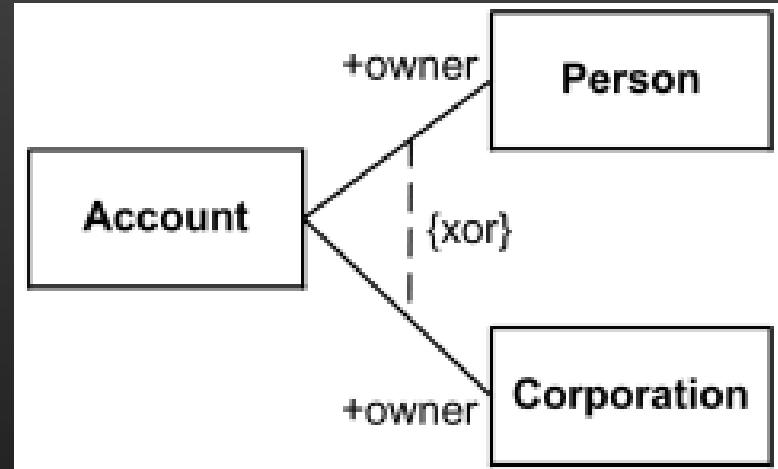
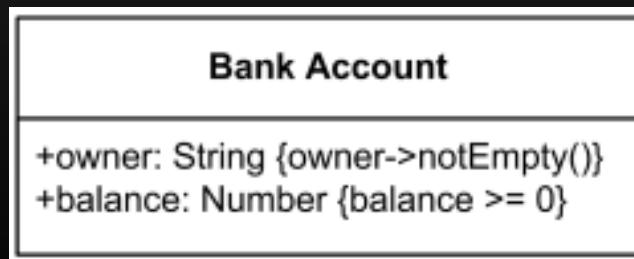


Restrições

Adicionar regras ao modelo ou condicionar a sua interpretação

condição ou restrição relacionada com um ou mais elementos

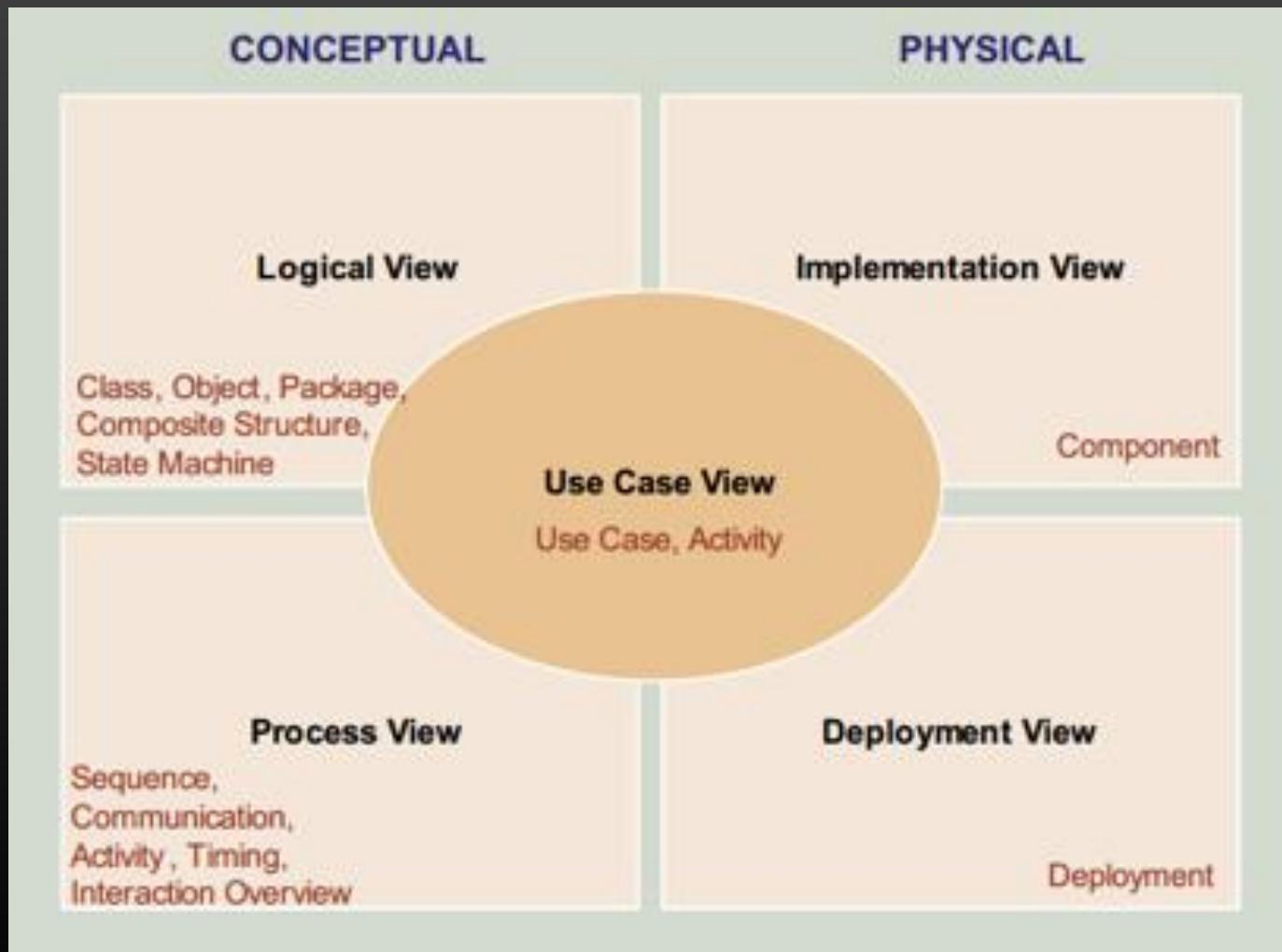
Linguagem própria para declarar restrições (OCL)



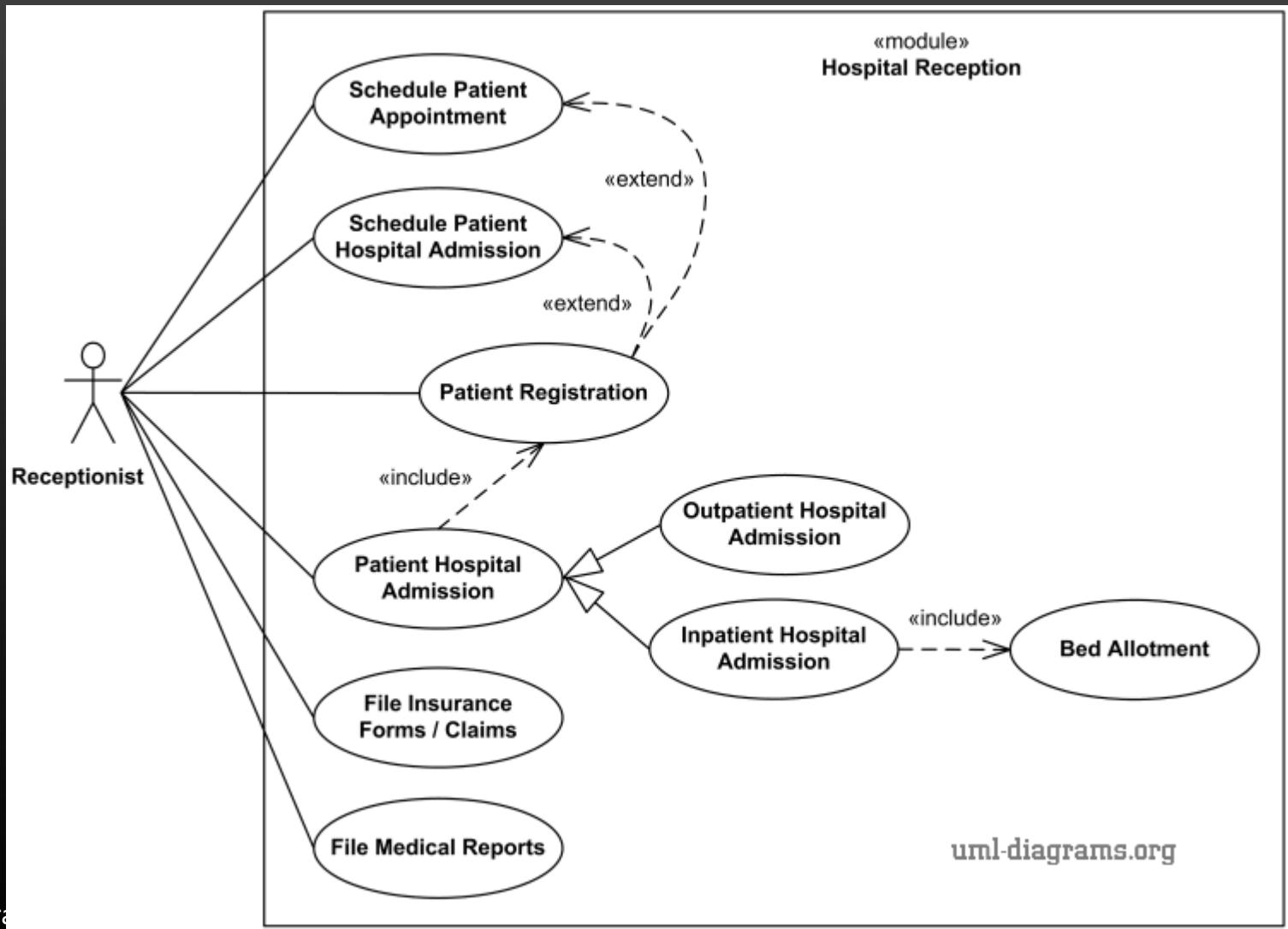
APLICAÇÃO DA UML AO LONGO DO PROCESSO DE DESENVOLVIMENTO

ILÍDIO OLIVEIRA ico@ua.pt
v2017-06-02

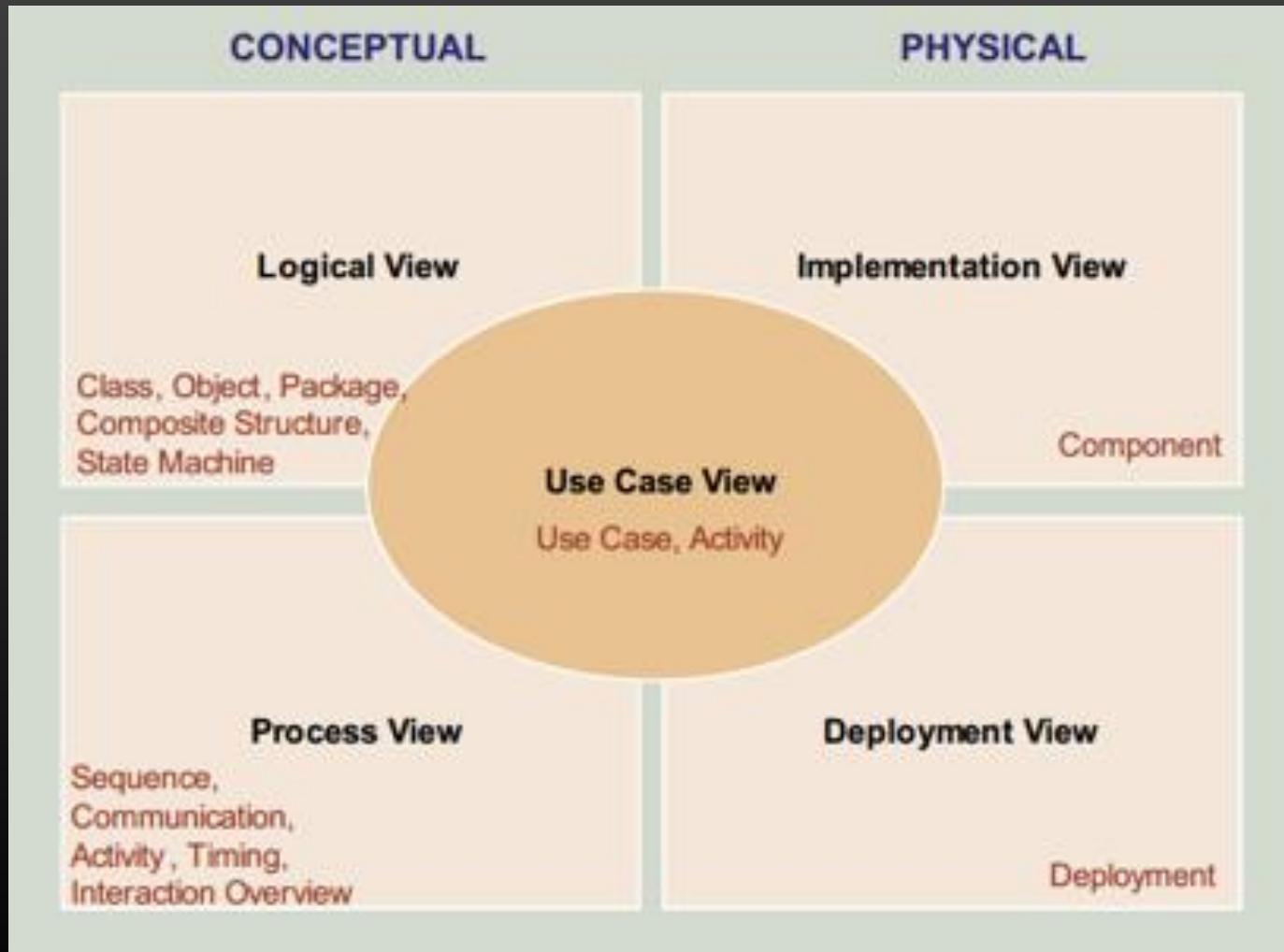
Diversos diagramas para abranger diferentes perspetivas de análise



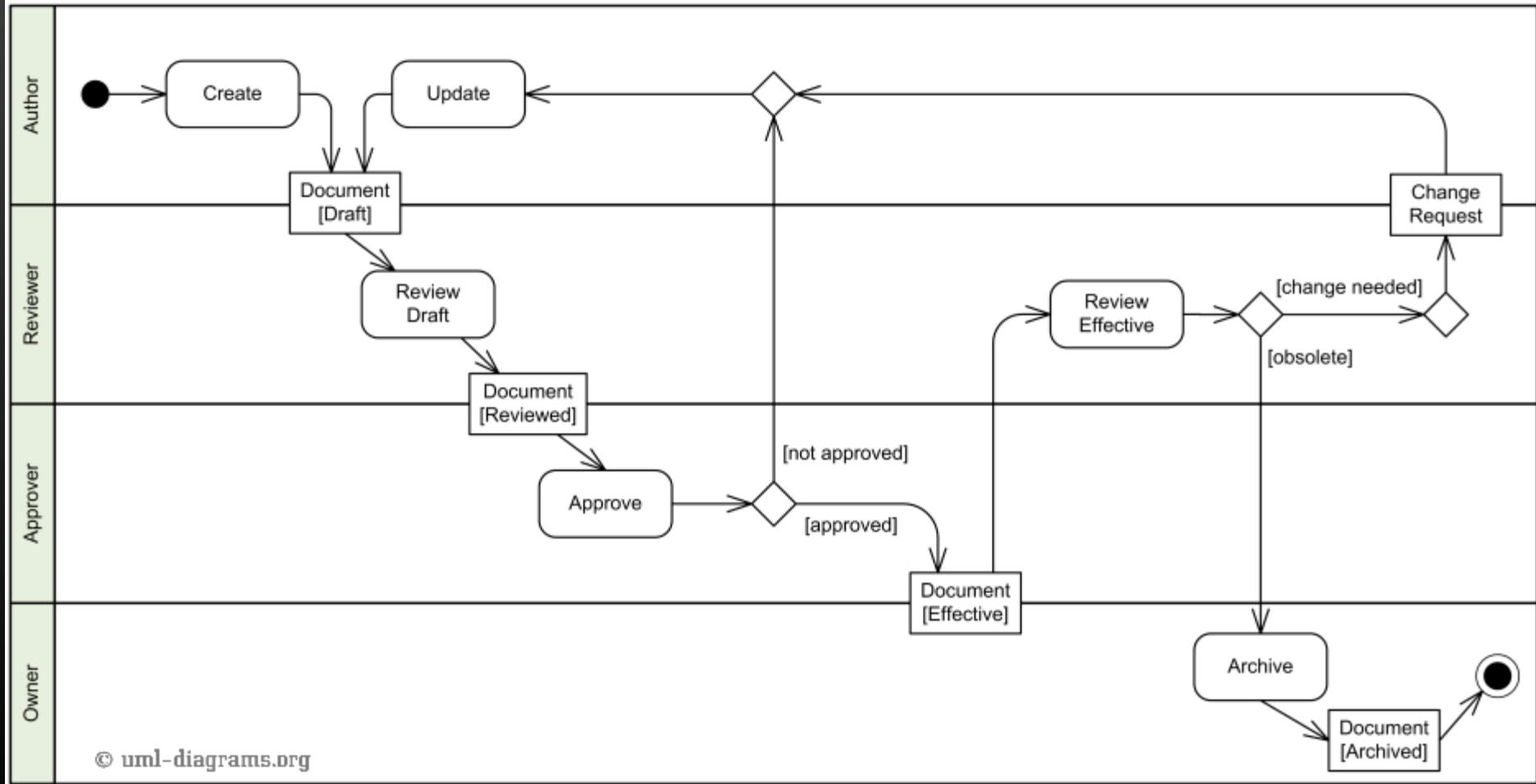
CaU do Sistema: organizar a funcionalidade do sistema em episódios de utilização



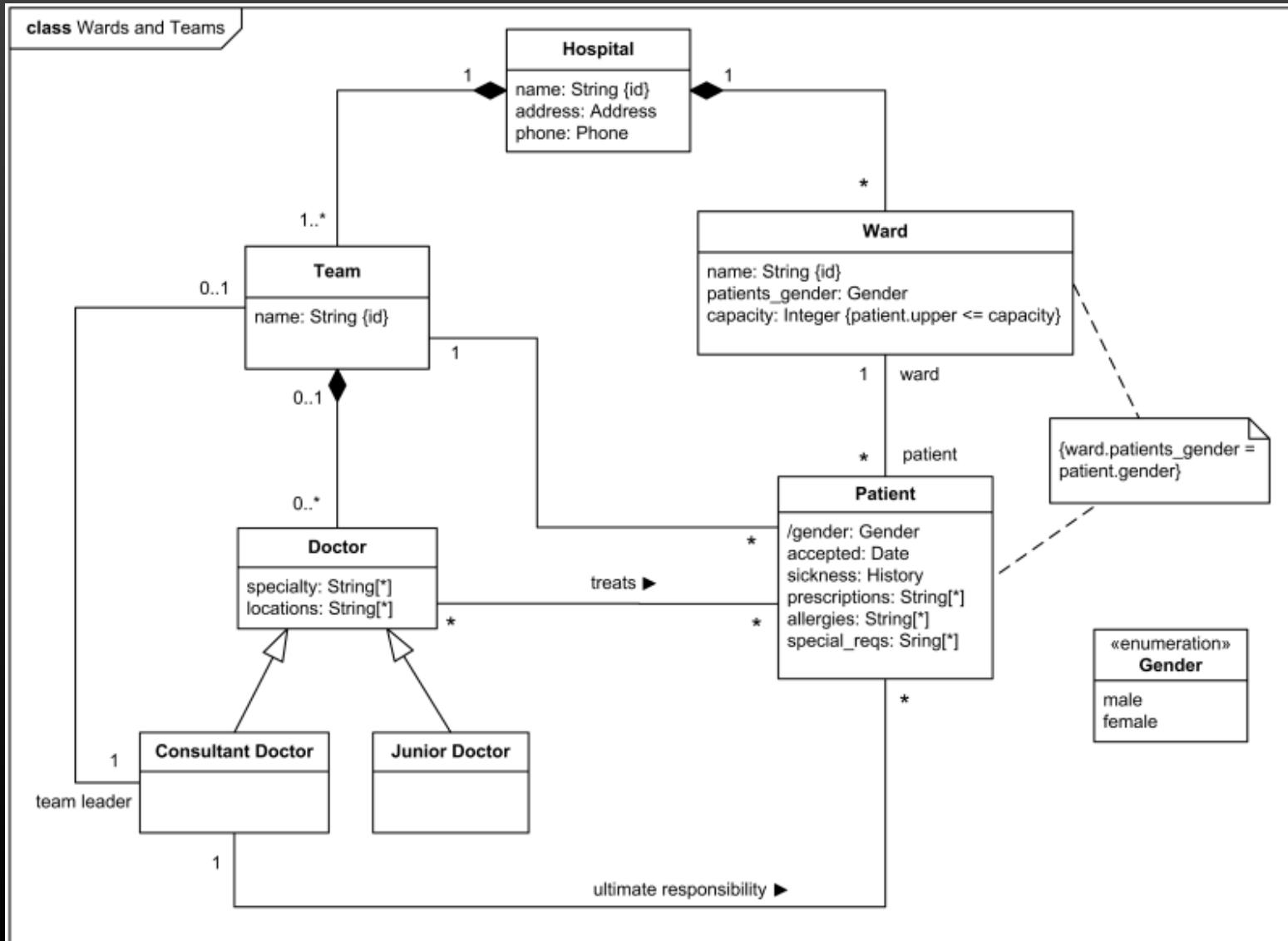
Diversos diagramas para abranger diferentes perspetivas de análise



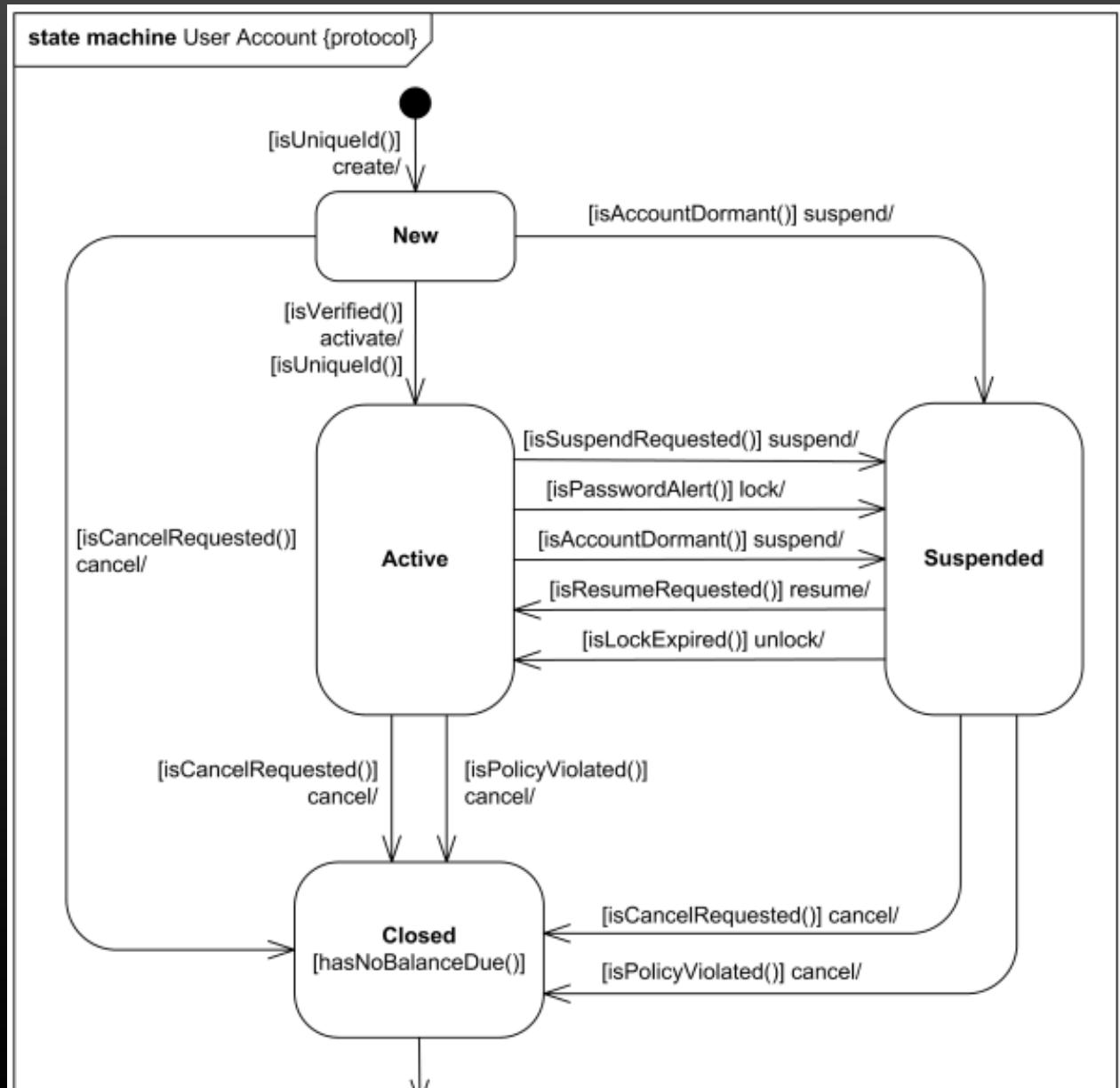
Diagramas de atividades para explicar procedimentos do domínio



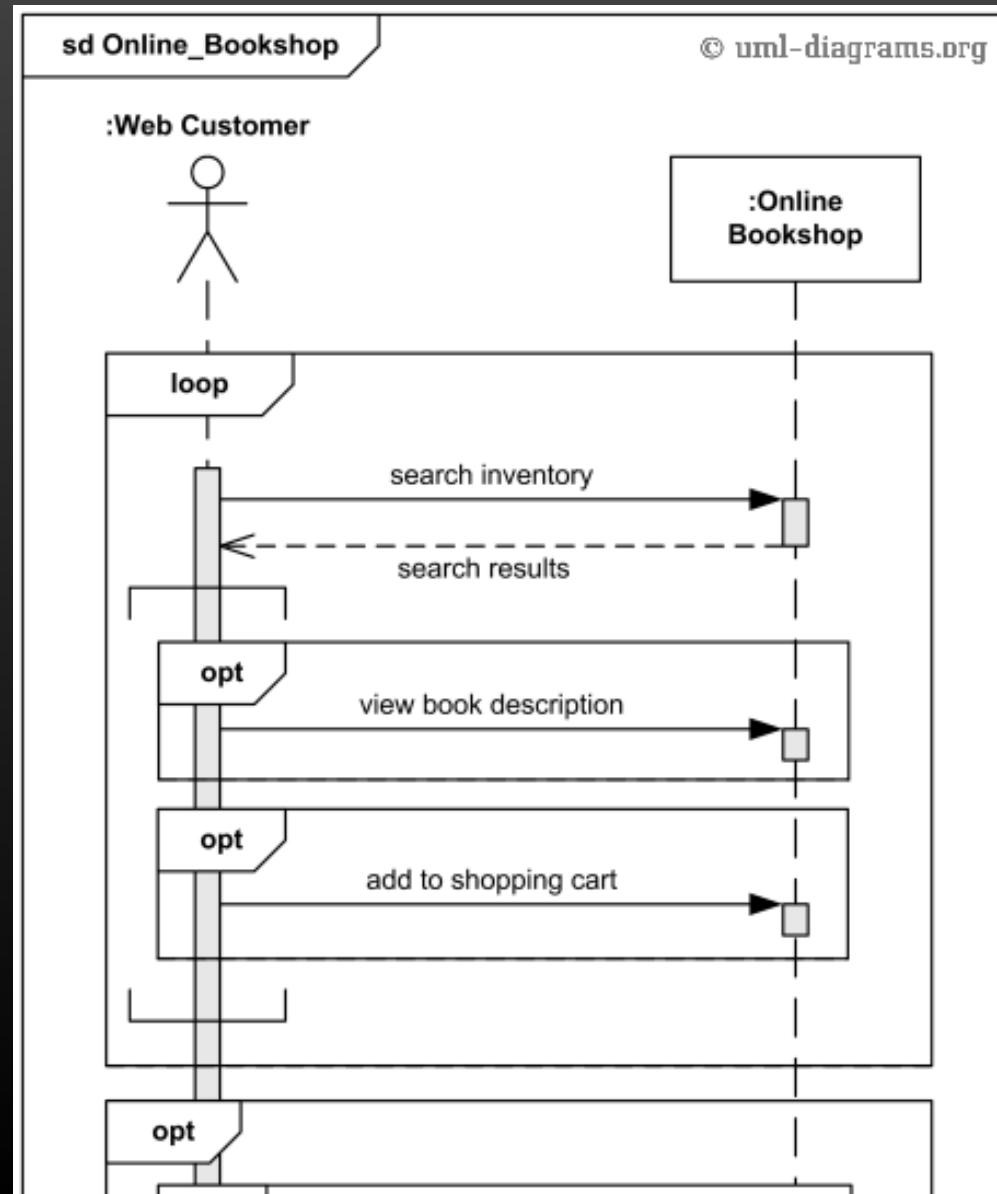
Classes para representar os conceitos da área do problema (modelo do domínio)



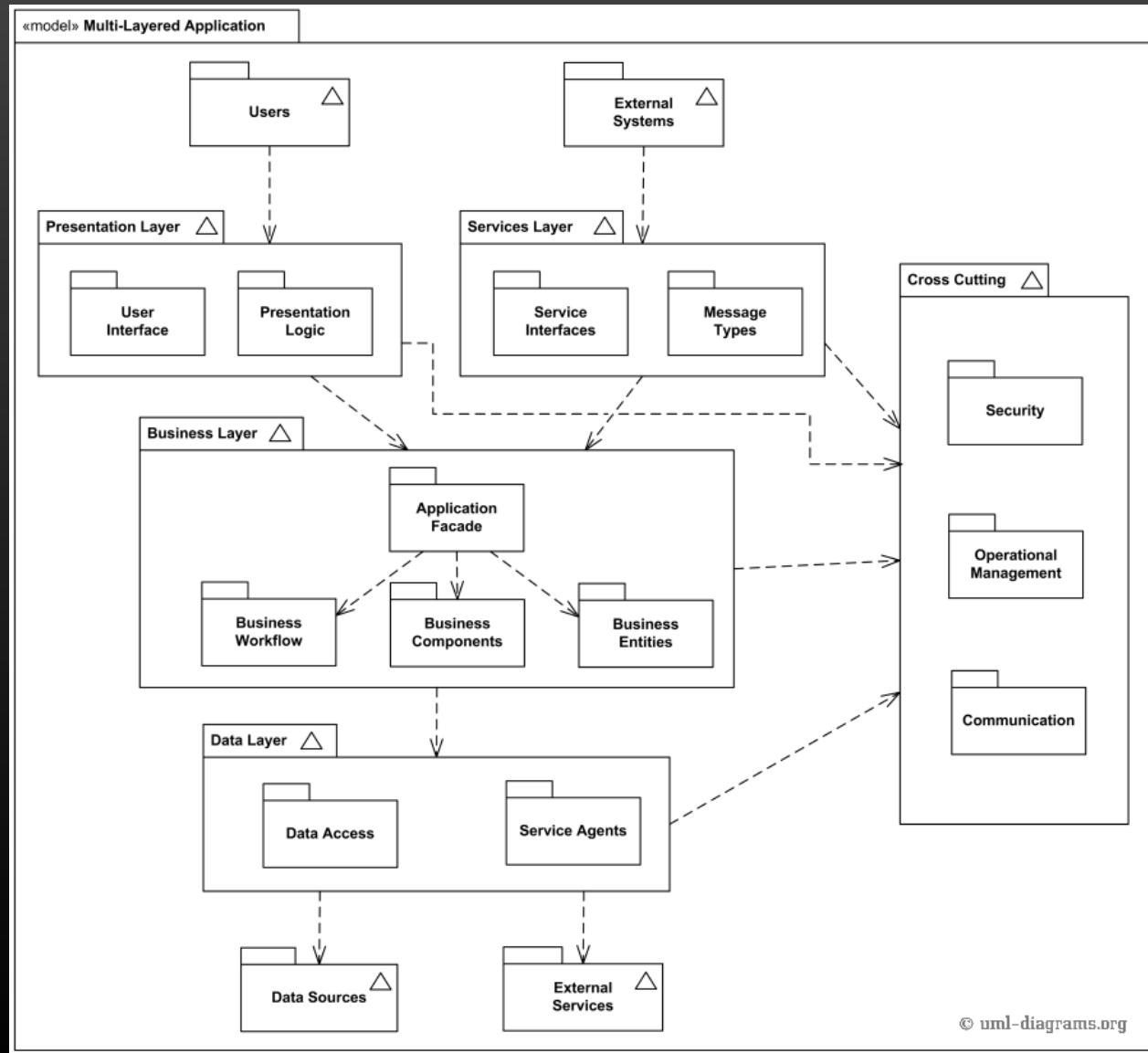
Máquina de estados de entidades/objetos



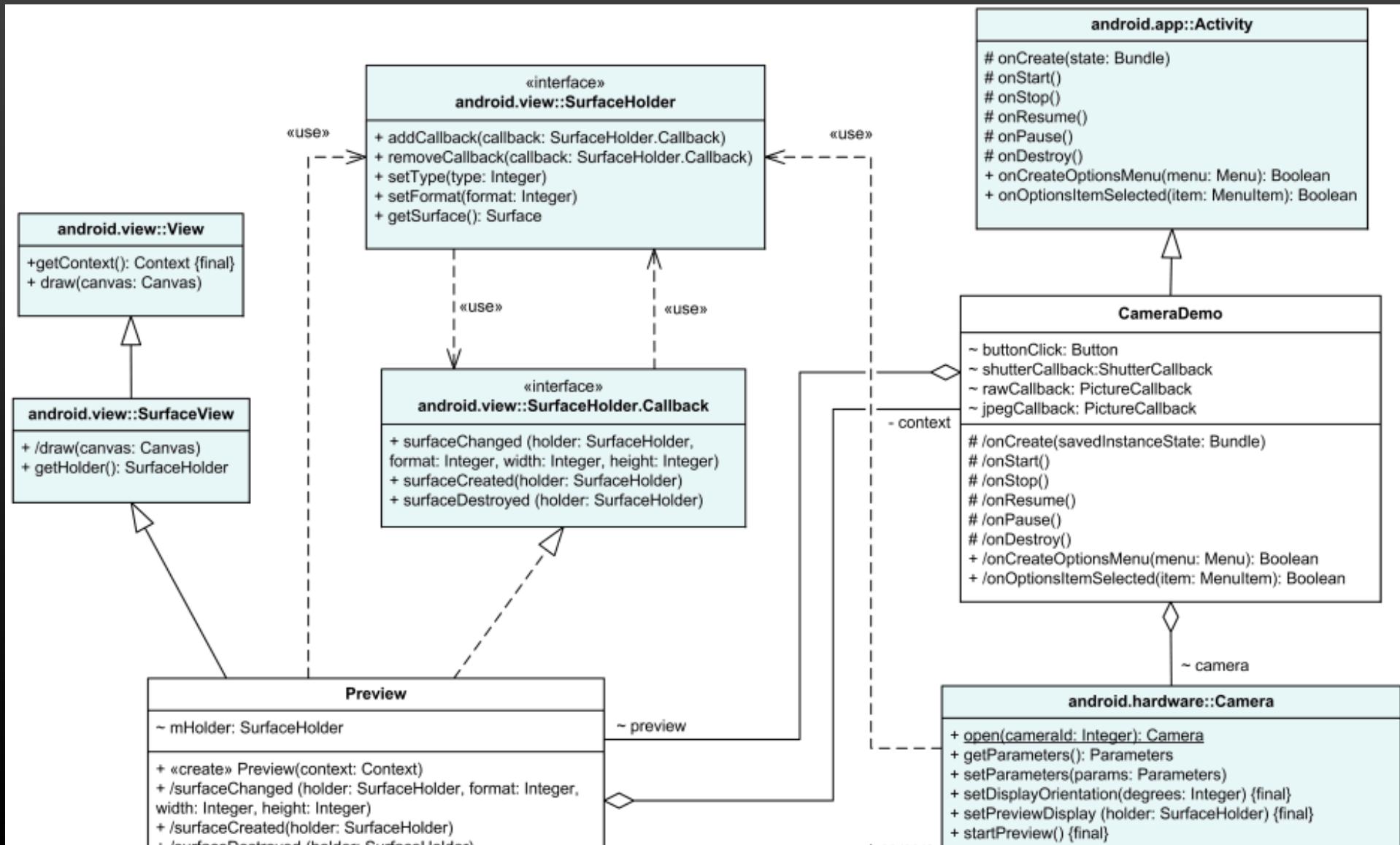
Interação entre atores/objetos



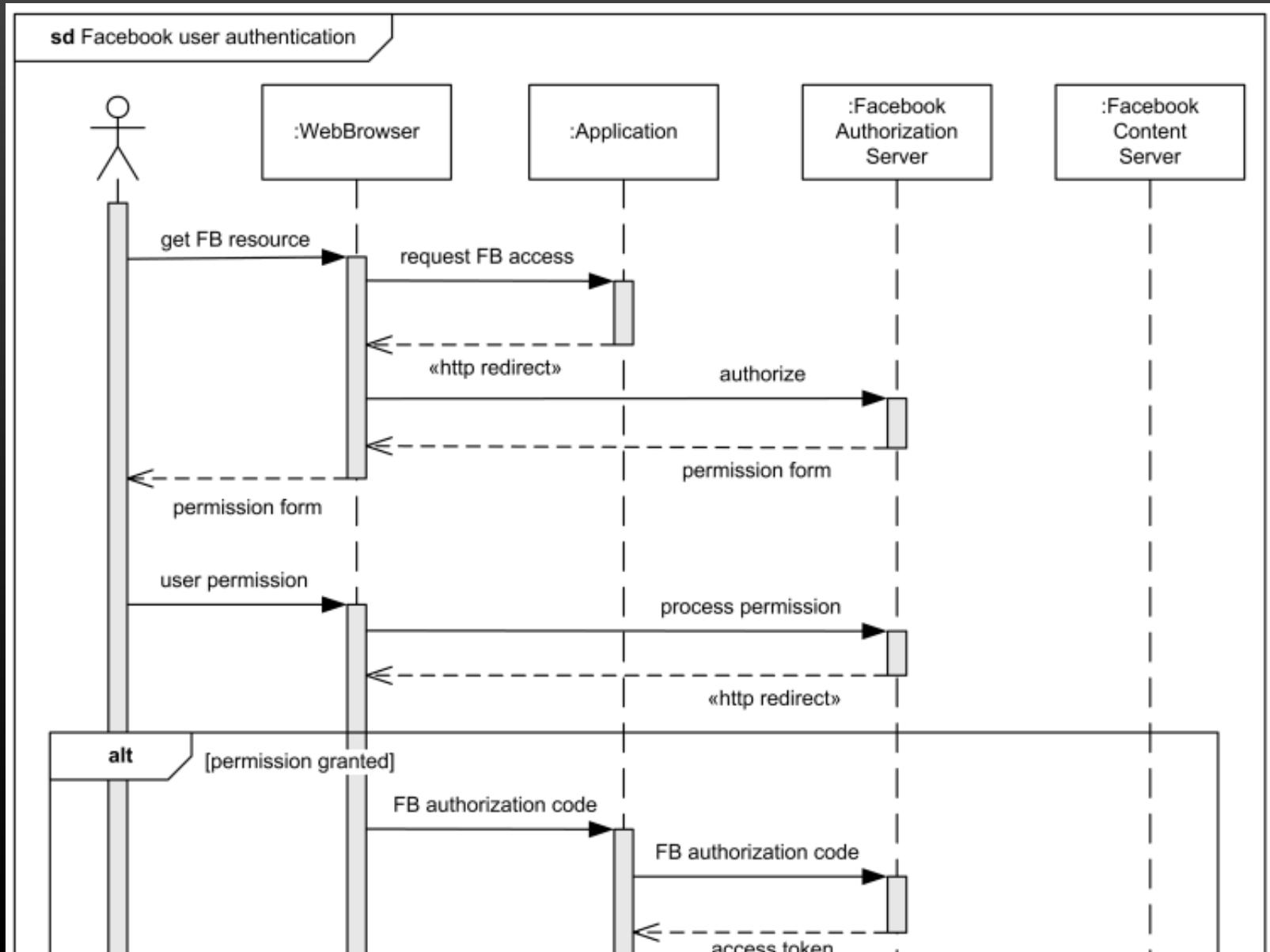
Visualizar a arquitetura lógica com D. Pacotes



Classes para visualizar objetos de um linguagem de programação

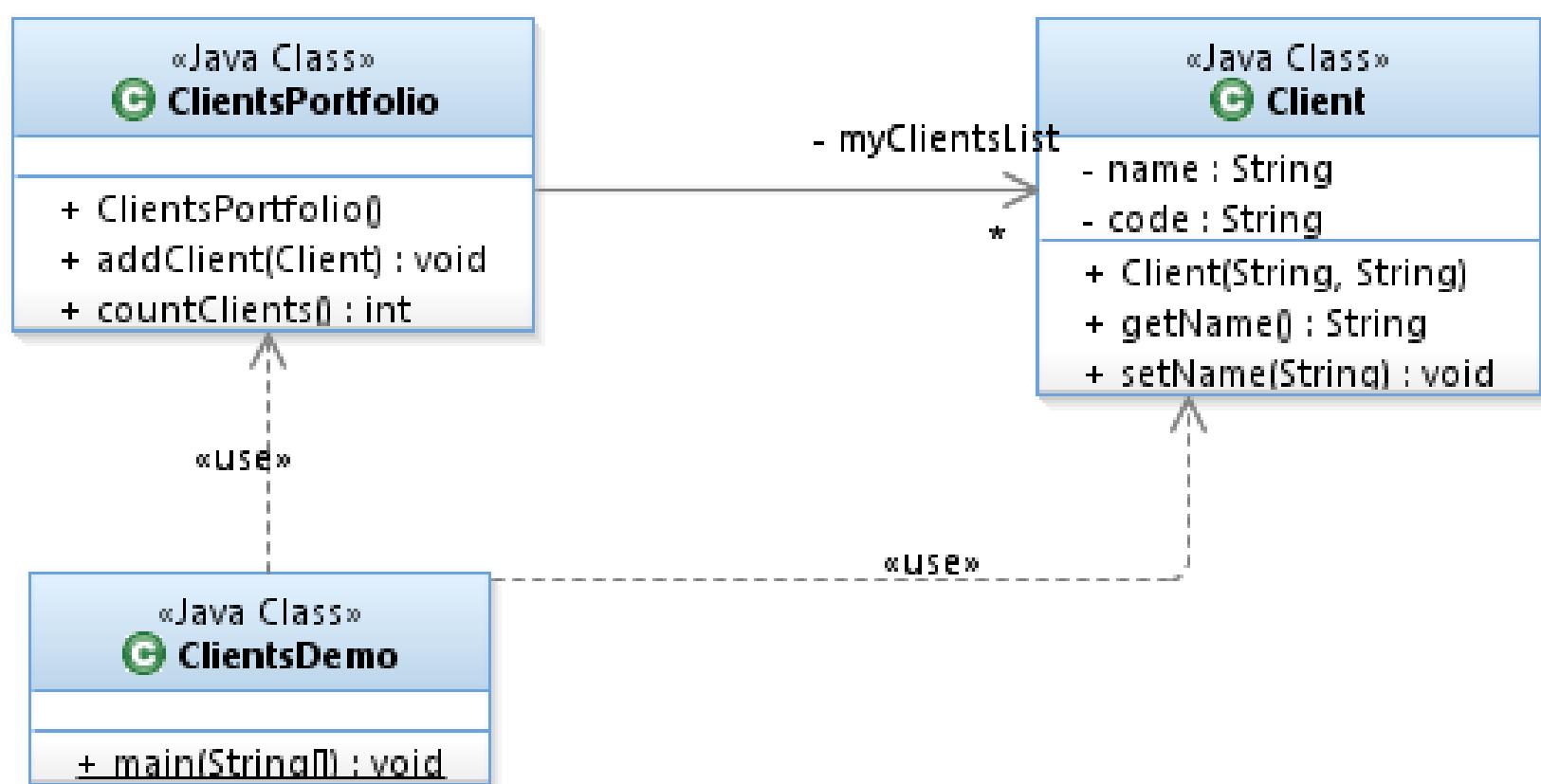


Interações entre componentes do software

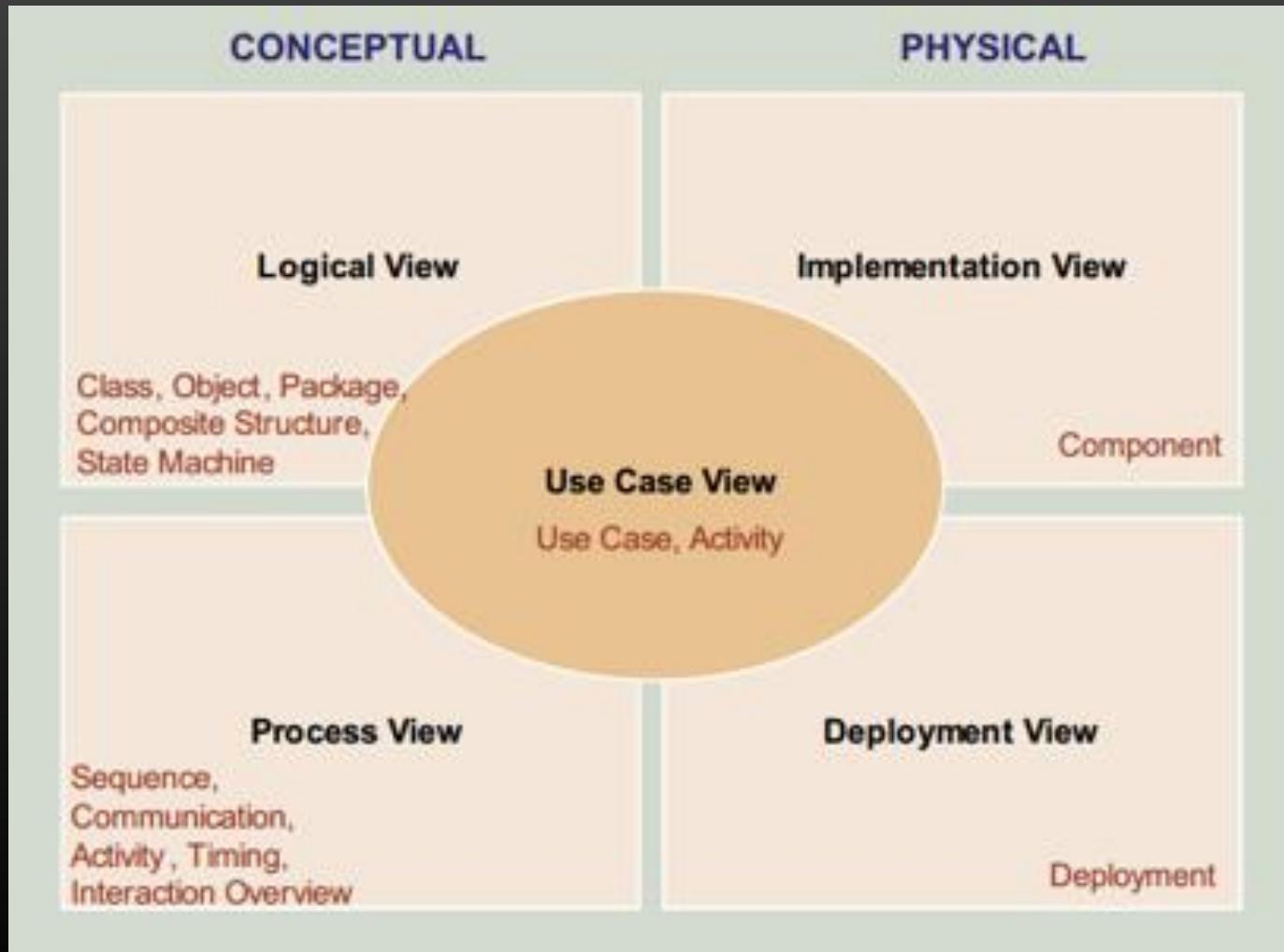


Objetos em código

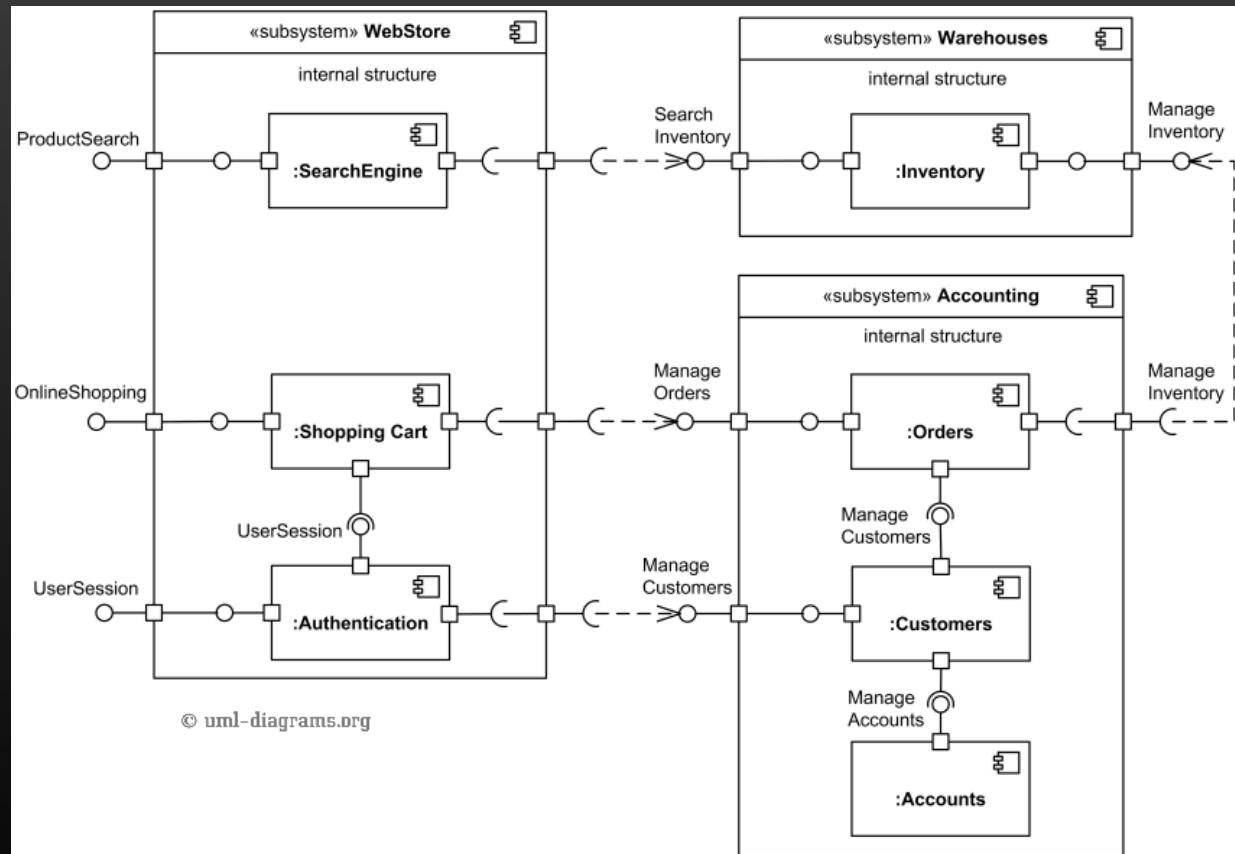
```
public class ClientsPortfolio {  
    private ArrayList<Client> myClientsList;  
  
    public ClientsPortfolio() {  
        myClientsList = new ArrayList<>();  
    }  
    public void addClient(Client newClient) {  
        this.myClientsList.add(newClient);  
    }  
    public int countClients() {  
        return this.myClientsList.size();  
    }  
}
```



Diversos diagramas para abranger diferentes perspetivas de análise



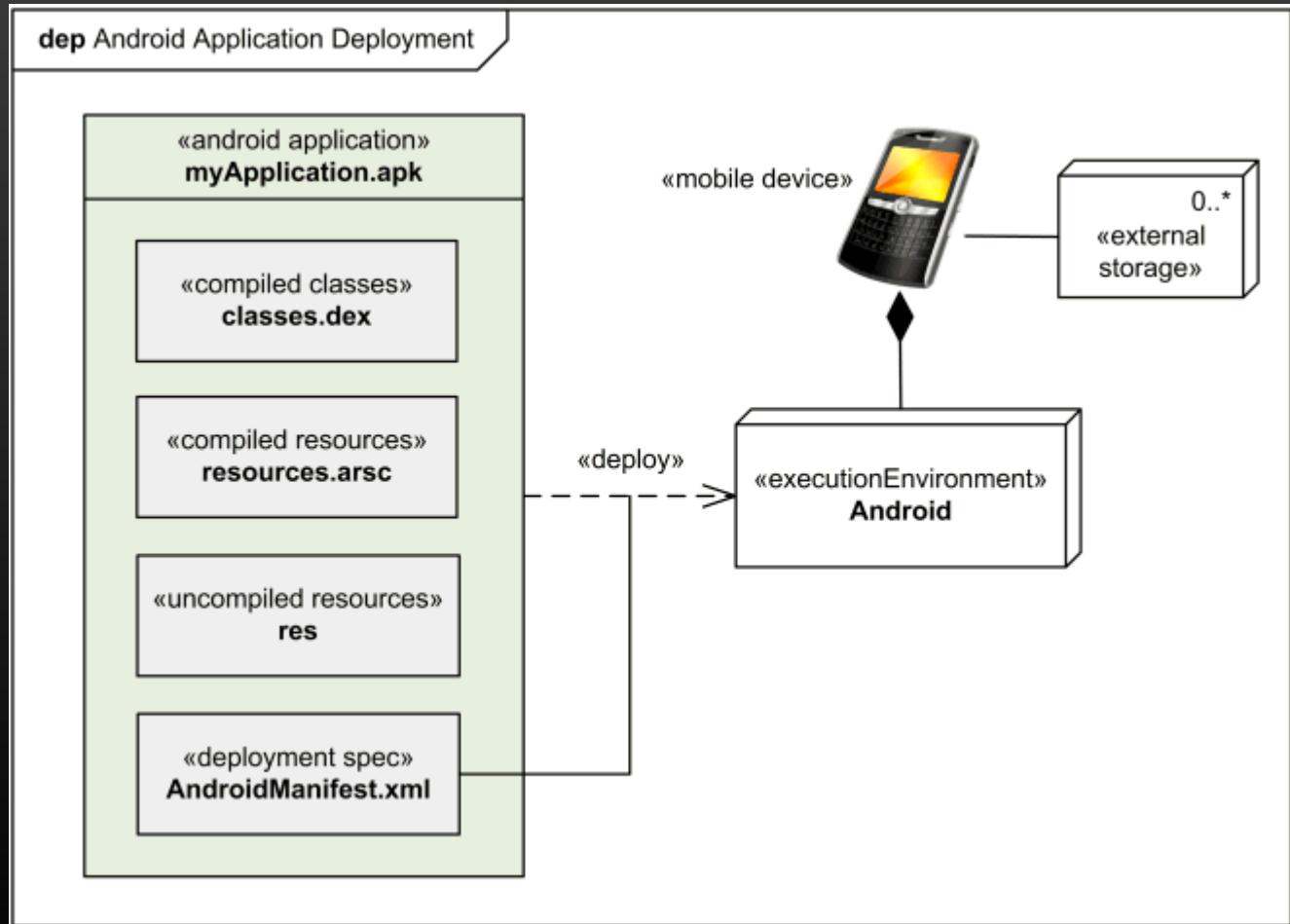
Módulos (executáveis) da solução captados em componentes



Os componentes têm correspondência em artefactos concretos



D. Instalação: mostrar o *setup* para produção



Ferramentas CASE



Readings & references

Core readings	Suggested readings
<ul style="list-style-type: none">• [Dennis15] – Chap. 1	[LAR'12] Larman, C. (2012). <i>Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Iterative Development</i> . Pearson Education. → chap. 10, chap. 15.

47006- ANÁLISE E MODELAÇÃO DE SISTEMAS
1st Semester, 2018/19

Modeling Activities in UML

Ilídio Oliveira | 2020-10-09

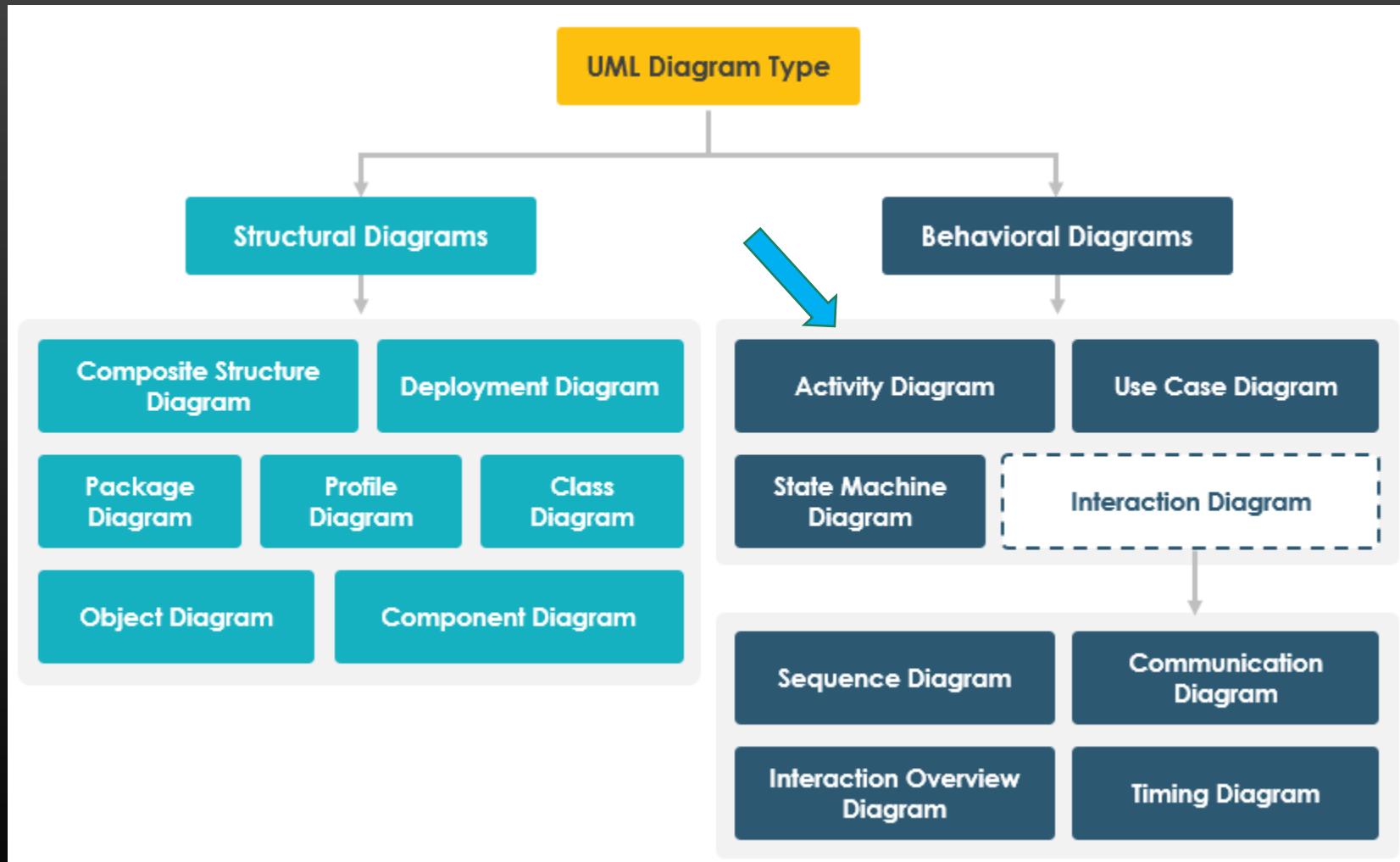
Learning objectives for this lecture

Read and create Activity Diagrams

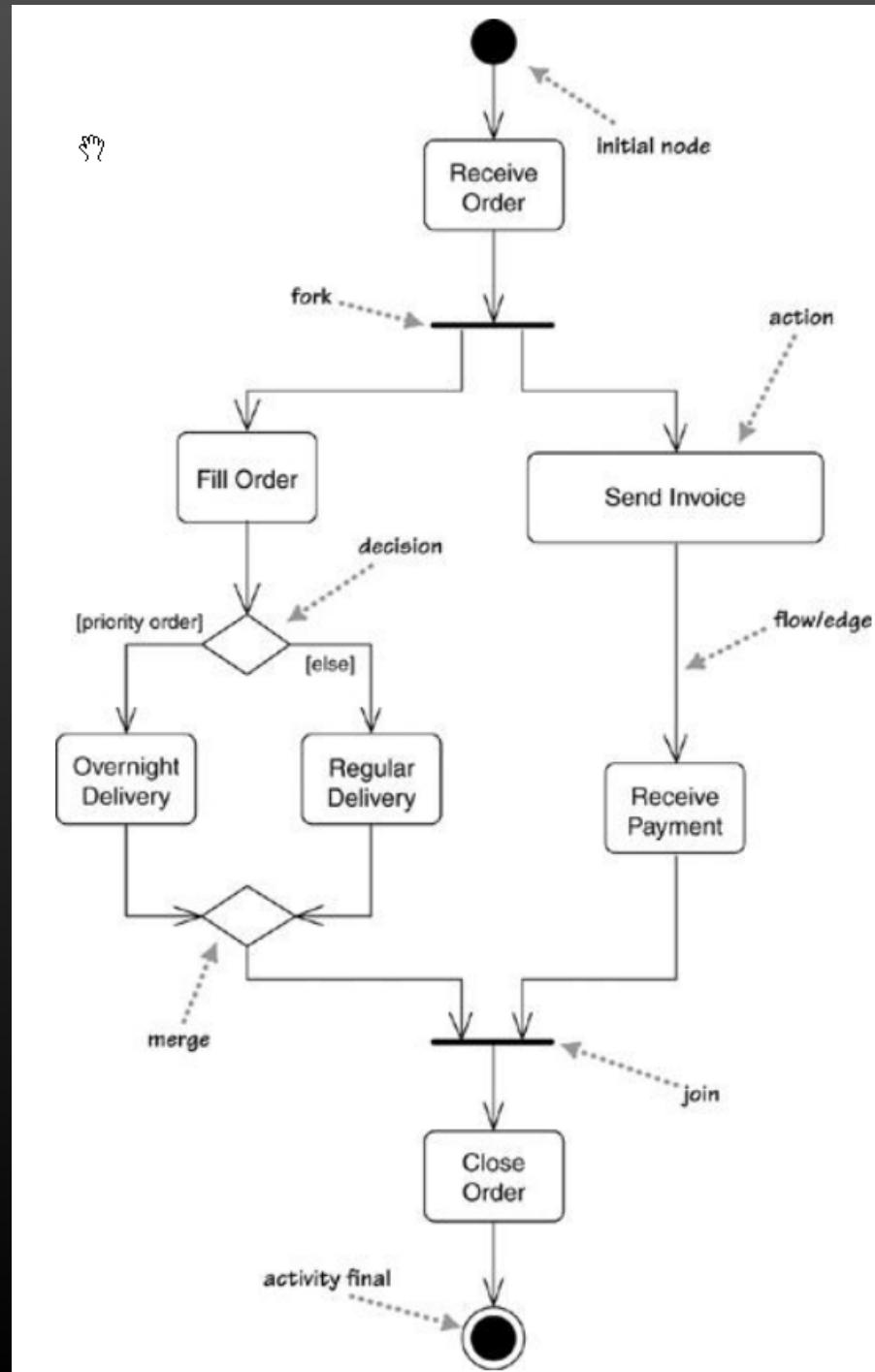
Distinguish structured activities from actions; control flows from data flows

Identify when to use Activity modeling

Diagramas da UML 2.x



Elementos do diagrama de atividades

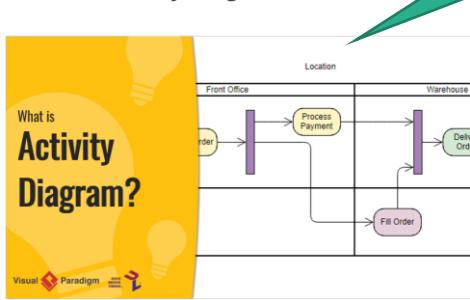


Os diagramas de atividade mostram o fluxo de ações (e de dados)

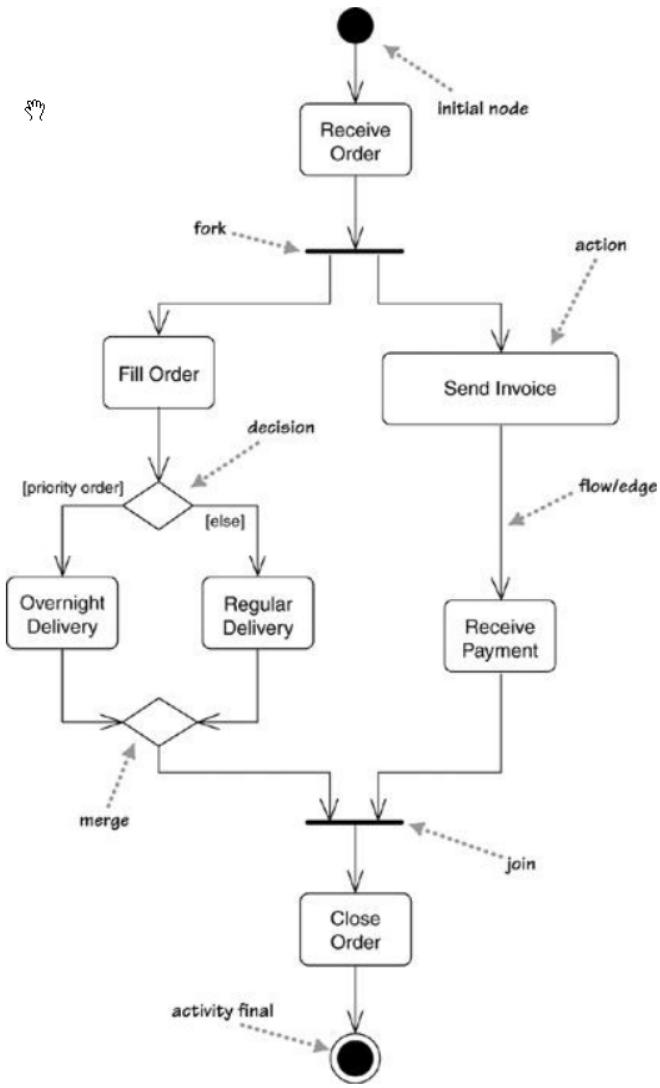
Quando aplicar?

- Modelar fluxos de trabalho/processos
- Modelar processos computacionais: e complexo

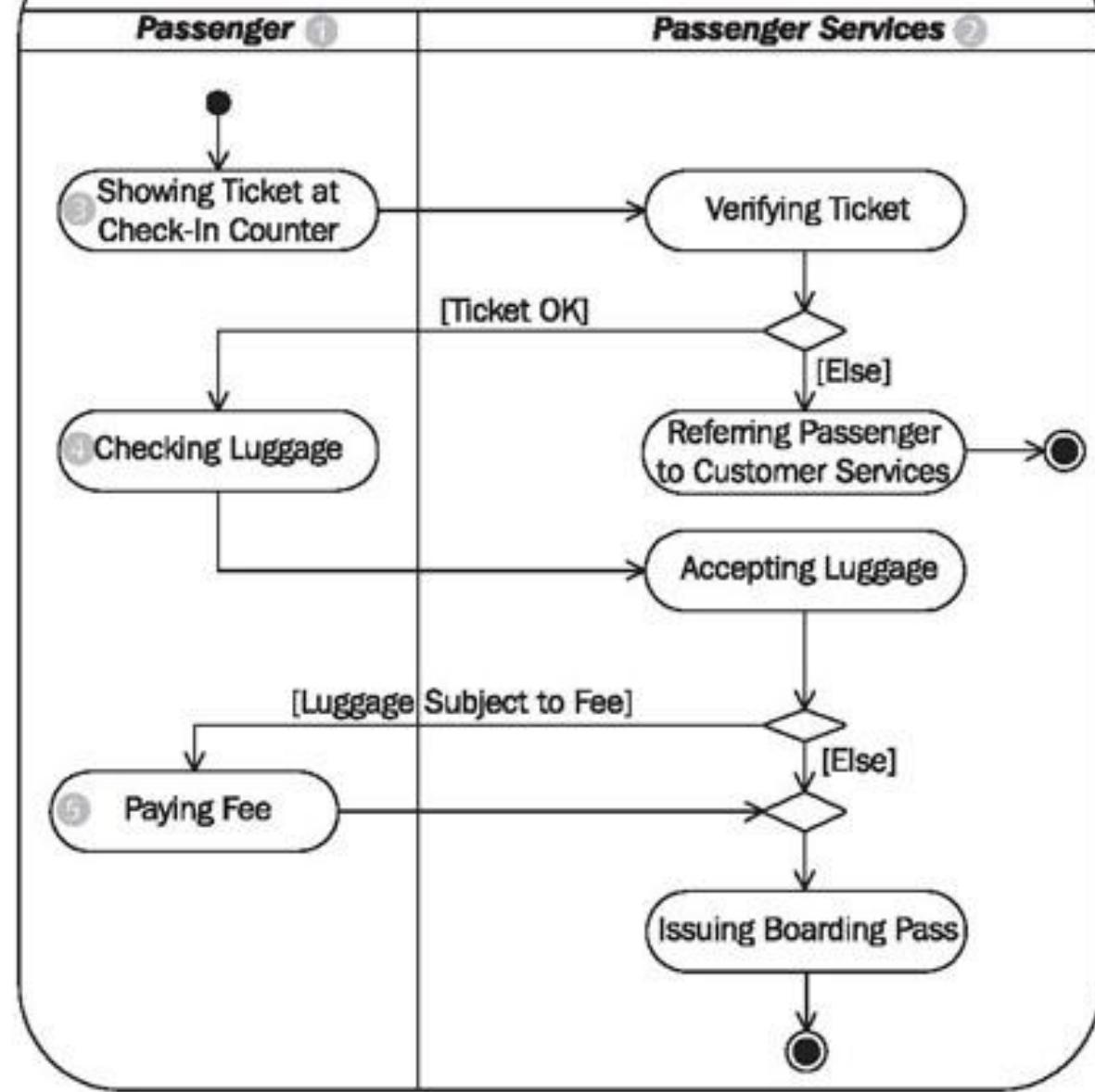
What is Activity Diagram?



<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/>



Passenger checks in



Os diagramas de atividade mostram o fluxo de ações (e de dados)

Quando aplicar?

Modelar fluxos de trabalho/processos de negócio

Descrever um algoritmo complexo

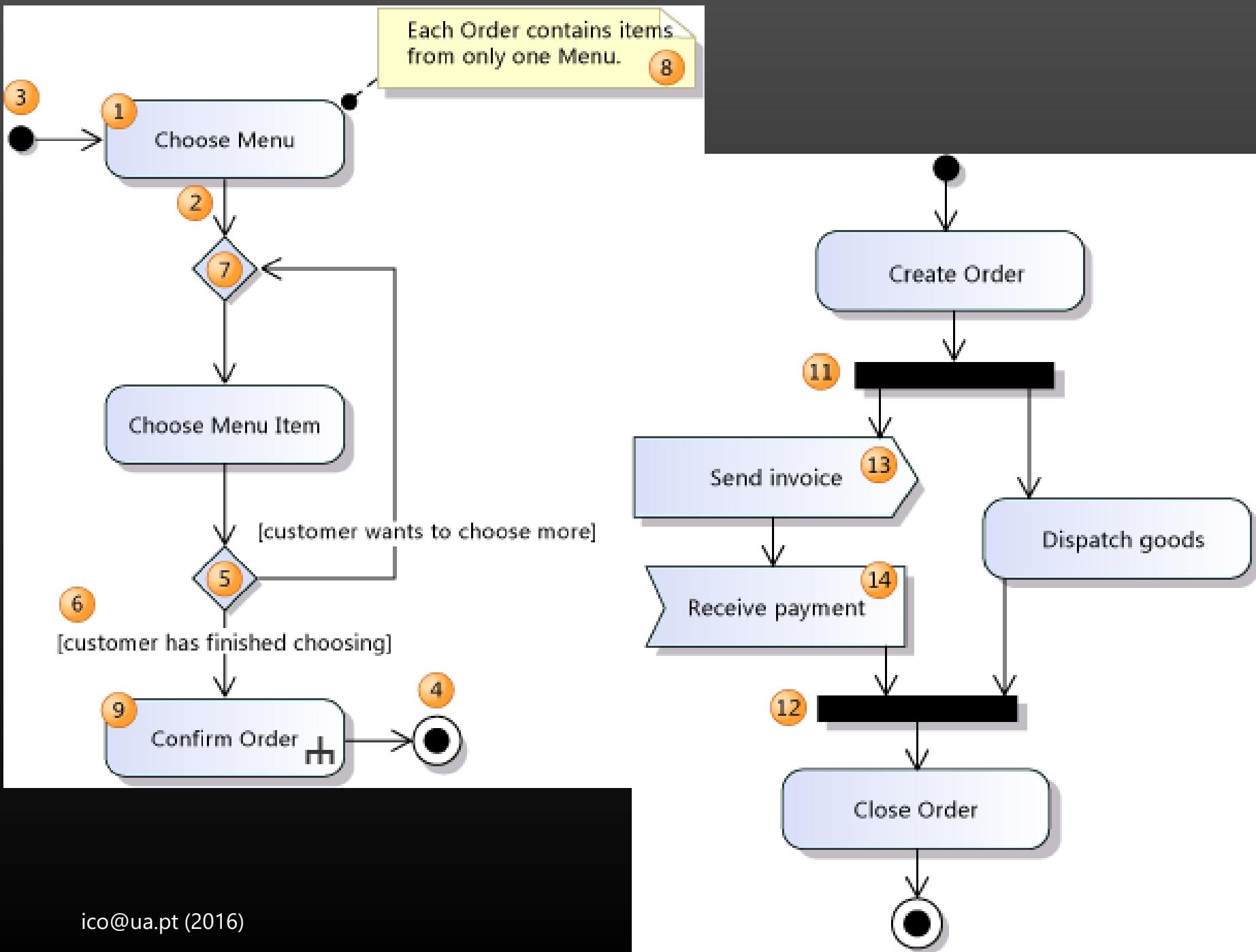
Descrever a sequência de interações entre atores e o sistema sob especificação, num caso de utilização

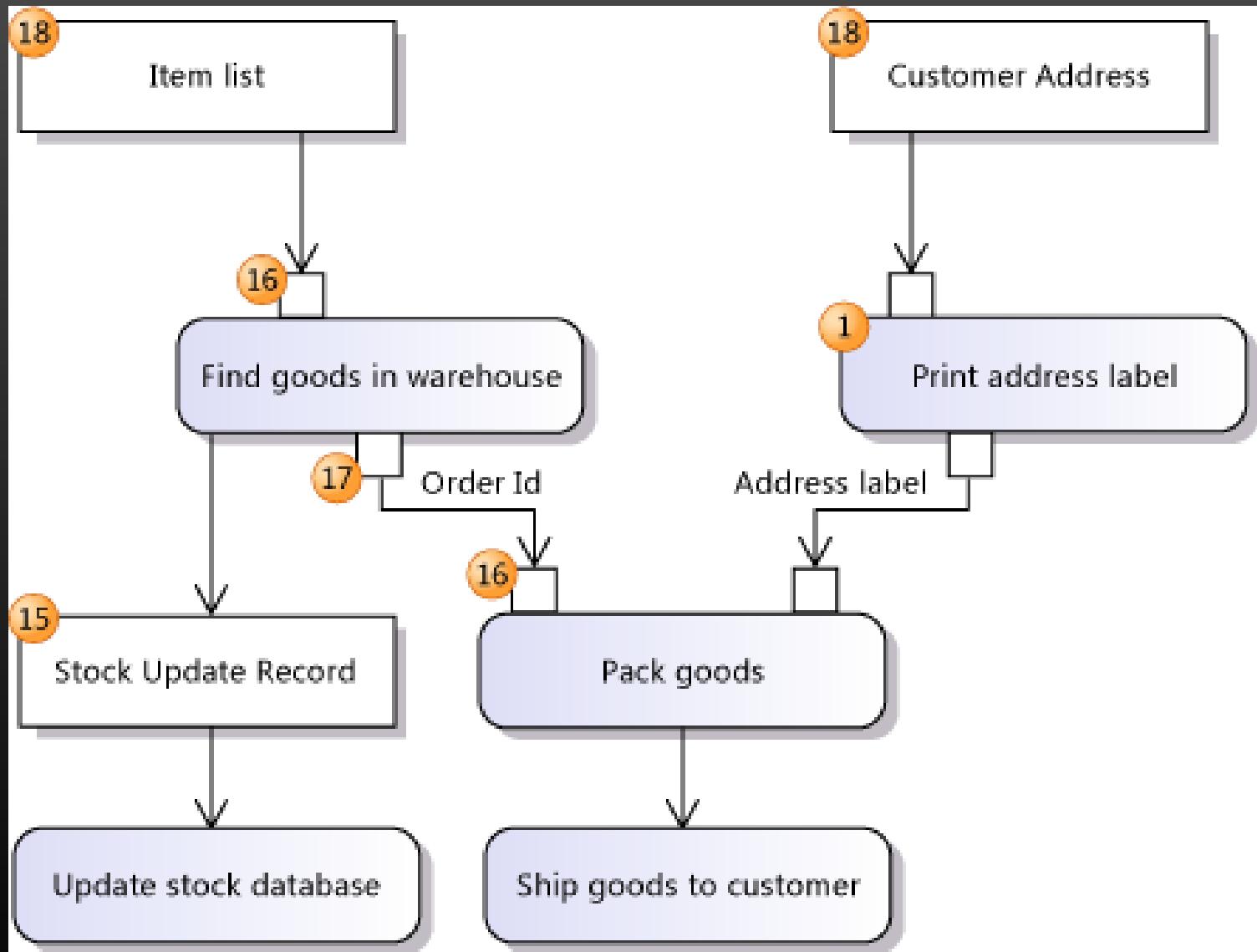
Pode ser usado para descrever os processos organizacionais existentes / novos

Neutro em relação à programação

Bom a captar papéis

Pode captar o fluxo de dados também





Diagramas para representar workflow

Definido na UML.

Diagram	Description	Advantages	Disadvantages
System flowchart	Earliest form for depicting sequencing of activities.	<ul style="list-style-type: none">■ Intuitive. Each type of input and output is clearly marked with its own symbol.■ Includes logic symbols.	<ul style="list-style-type: none">■ Not compliant with UML.■ Can be hard to learn (many symbols).
Activity diagram	UML tool for describing logic. Used to describe entire system, a use case, or an activity within a use case. Has two versions: <ul style="list-style-type: none">■ Activity diagram without partitions (swimlanes): Does not show who does what.■ Activity diagram with partitions: shows who does what.	<ul style="list-style-type: none">■ Part of UML standard.■ Can handle many situations in one diagram.■ Simple diagramming conventions.■ Encourages thinking about opportunities for parallel activities (more than one activity going on at the same time).	<ul style="list-style-type: none">■ Ability to handle many situations can lead to a diagram that is too complex to follow.
Business process diagram (BPD)	Business process modeling notation (BPMN) tool for describing workflow	<ul style="list-style-type: none">■ Part of BMN standard, managed by the OMG■ Rich symbol set can model complex and subtle workflow requirements better than activity diagrams.	<ul style="list-style-type: none">■ Not UML-compliant■ Difficult to understand without prior training

Readings & references

Core readings	Suggested readings
<ul style="list-style-type: none">• [Dennis15] – Chap. 4	<ul style="list-style-type: none">• Vídeo tutorial com uma explicação dos Diagramas de Atividades• Visual Paradigm tutorials: what is the Activity Diagram• MSDN, “Developing models for software design”

47006- ANÁLISE E MODELAÇÃO DE SISTEMAS

Software Processes: systematic approaches to software development

Ilídio Oliveira | 2020-10-14

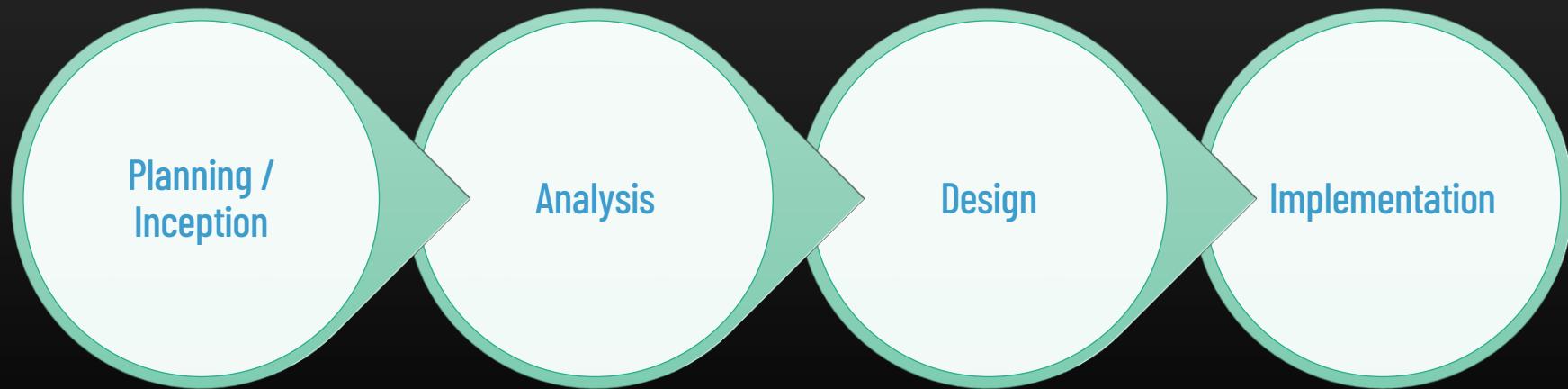
Objetivos de aprendizagem

- 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

Software Development lifecycle

Quatro fases fundamentais: planeamento/conceito, análise, desenho e implementação. Diferentes projetos podem enfatizar diferentes partes do SDLC ou realizar as fases SDLC de diferentes formas, 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.



Fases fundamentais: planeamento, análise, desenho e implementação

A fase de planeamento é o processo fundamental de compreensão do porquê de um sistema de informação ser construído e determinar como a equipa do projeto irá construí-lo.

Atividades-chave:

1. Arranque do projeto (obter o “OK”)

O valor que o sistema gerará para a organização é identificado.

O pedido do sistema e a análise de viabilidade são apresentados a uma comissão para decidir se o projeto deve ser realizado.

2. Gestão do projeto

O gestor do projeto cria um plano de trabalho, equipa o projeto, e coloca técnicas para a equipa controlar e dirigir o projeto através de todo o SDLC.

Fases fundamentais: planeamento, análise, desenho e implementação

A fase de análise responde às questões de quem irá usar o sistema, o que o sistema vai fazer, e onde e quando será utilizado.

Durante esta fase, a equipa do projeto investiga qualquer sistema atual, identifica oportunidades de melhoria e desenvolve um conceito para o novo sistema.

Atividades-chave:

1. Análise dos sistemas existentes,
2. Recolha de requisitos (necessidades da organização)
3. Conceito de solução (proposta do sistema)

Fases fundamentais: planeamento, análise, desenho e implementação

A fase de desenho (=projeto técnico) decide como o sistema funcionará, em termos de hardware, software e infraestrutura de rede; a interface, formulários e relatórios do utilizador; e os programas específicos, bases de dados e ficheiros que serão necessários.

Atividades-chave:

1. Estratégia de desenvolvimento (interna ou contratar?)
2. Desenho da arquitetura do sistema
3. Desenho do modelo de dados
4. Desenho dos programas (classes, etc.)

Fases fundamentais: planeamento, análise, desenho e implementação

Na fase de implementação, o sistema é efetivamente construído (ou adquirido, no caso de integração de pacotes existentes).

Inclui também a transição para o ambiente de produção.

Atividades-chave:

1. Implementação de sistemas (construção e garantia de qualidade)
2. Instalação e transição
3. Plano de suporte (revisão pós-instalação e gestão de alterações)

Towards an engineering process

The SDLC is realized using a systematic software process.

Why do we need a formal process?

Failures occur (too) often

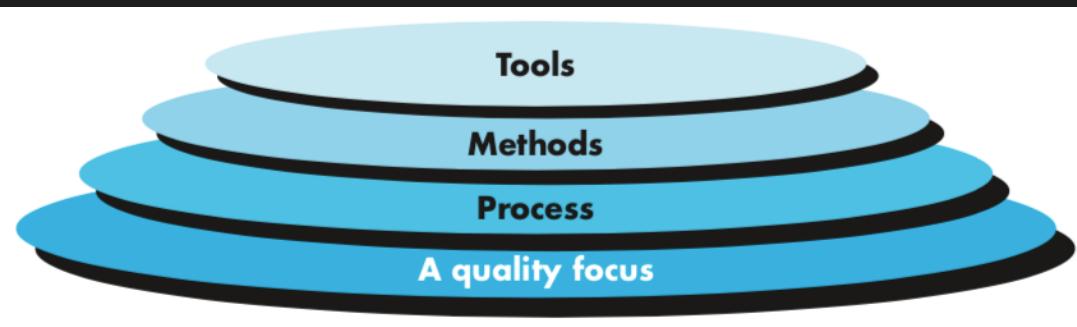
Creating systems is not intuitive.

Projects are late, over budget or delivered with fewer features than planned

Software Process

A software process is a framework for the activities, actions, and tasks that are required to build high-quality software.

It establishes the technical and management framework for applying methods, tools, and people to the development task.

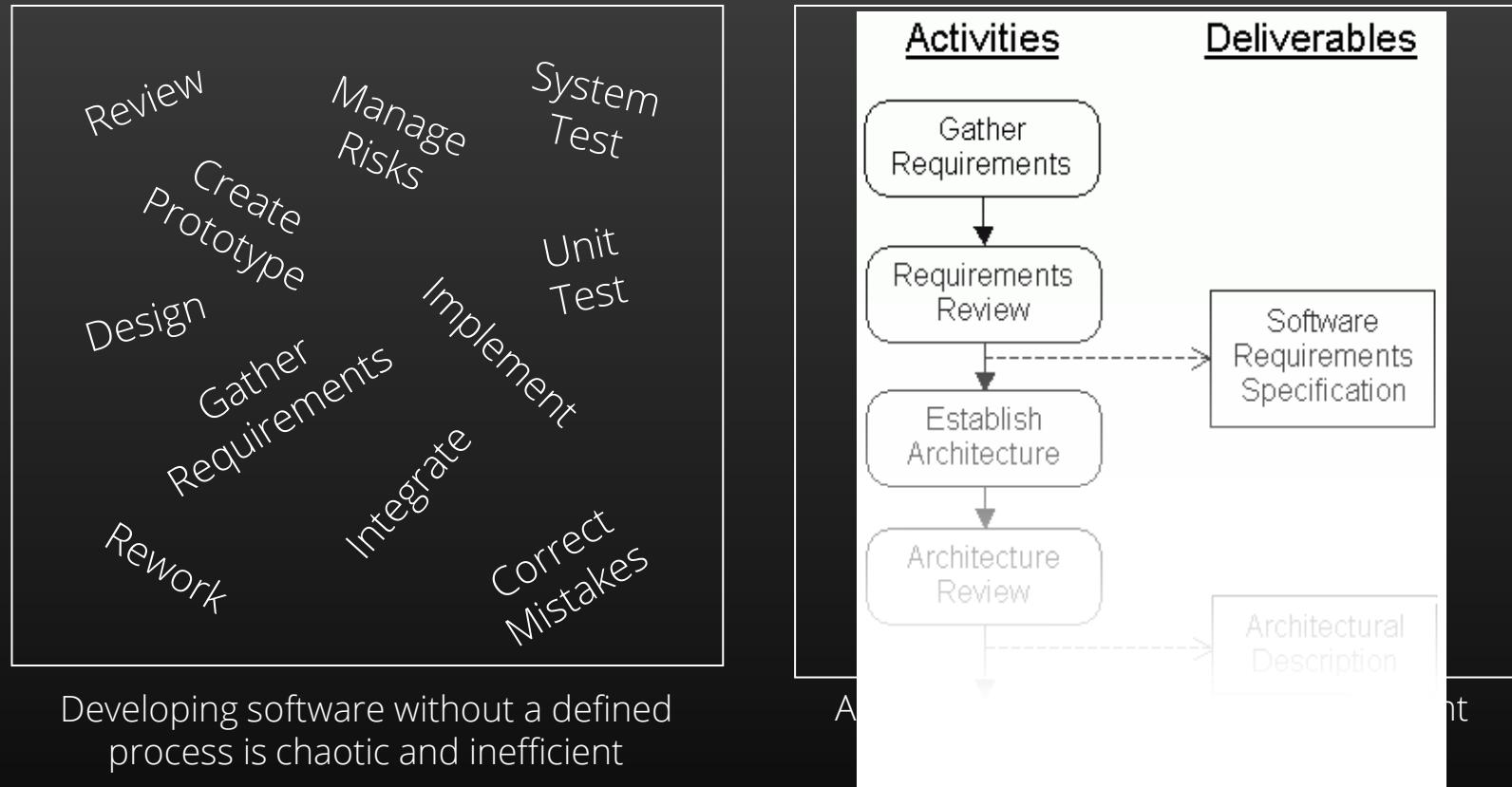


Is “process” synonymous with “software engineering”? Yes and no.

A software process defines the approach that is taken as software is engineered.

But software engineering also encompasses technologies that populate the process—technical methods and automated tools.

Why Software Process?



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

Software process description

When we describe and discuss processes, we usually talk about ...

the **activities** in these processes such as specifying a data model, designing a user interface, etc. and the **sequence** of these activities (process flow).

Process descriptions may also include:

Products, which are the outcomes of a process activity;

Roles, which reflect the responsibilities of the people involved in the process;

Pre- and post-conditions (dependencies), which are statements that are true before and after a process activity has been enacted or a product produced.

Software process

A process specifies:

What?

Who?

How?

When?

A process includes:

Roles

Workflows

Procedures

Standards

Templates

There is no single “best process”

Organizations should select (or customize) their process.

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

Plan-driven or evolutionary processes?

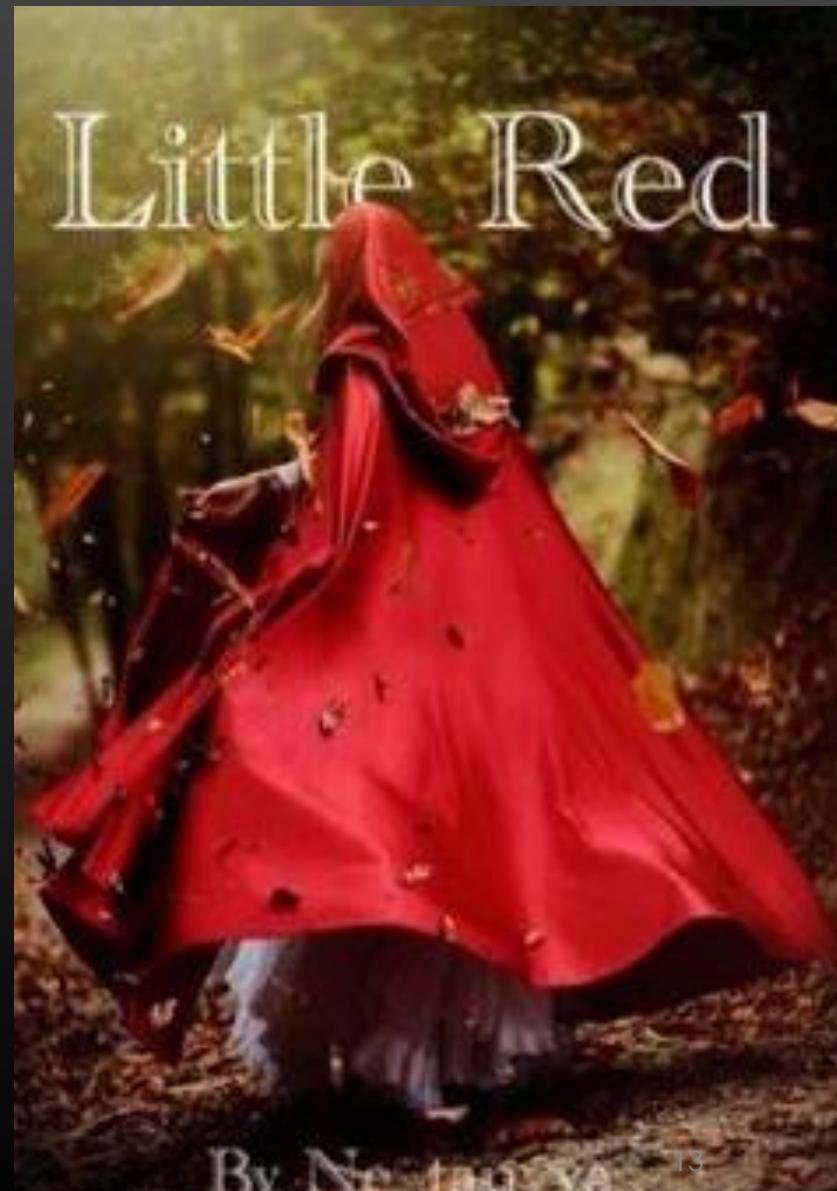
Plan-driven/prescriptive processes

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

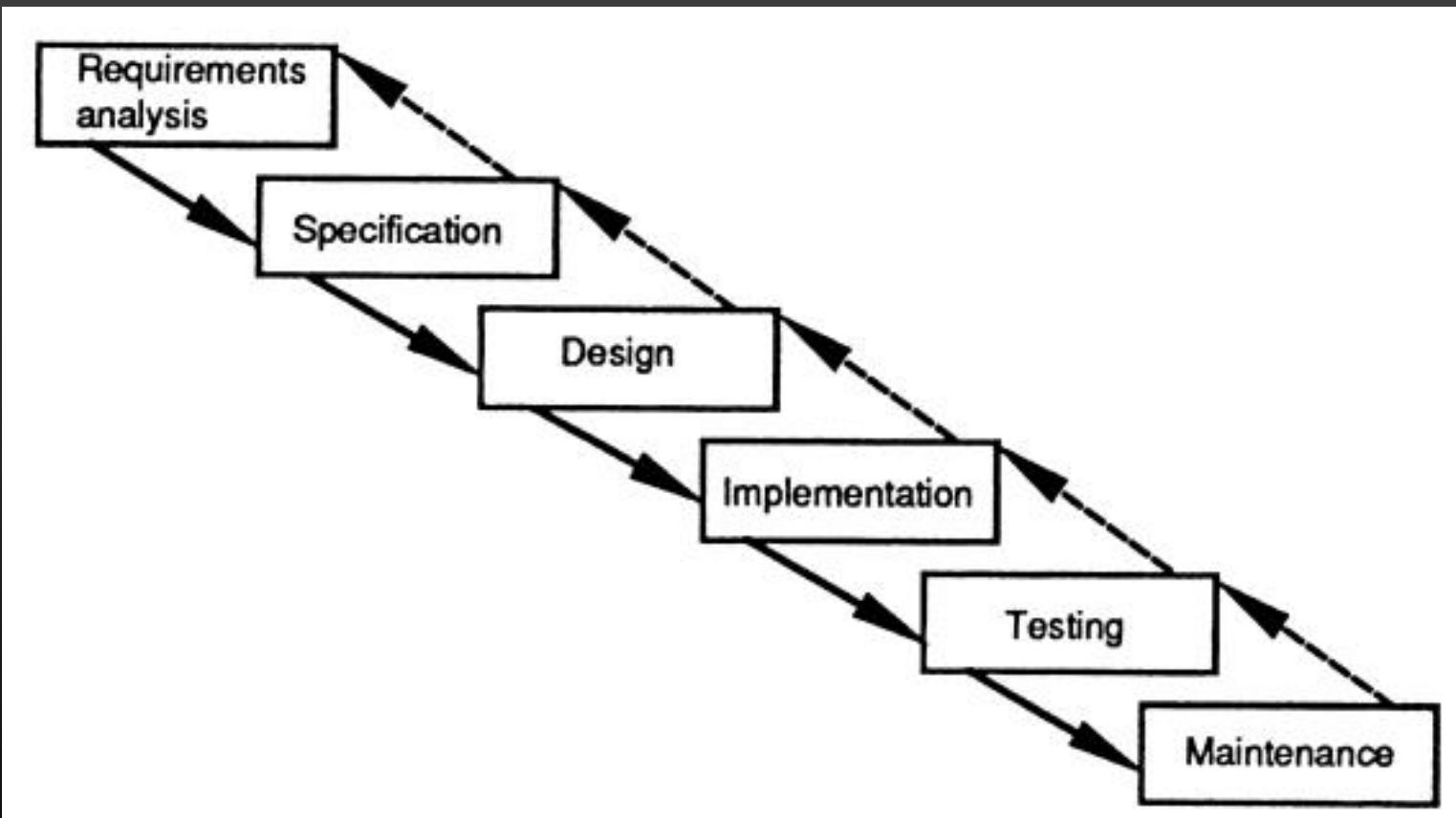
Evolutionary processes

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

<http://zle.9n.xsl.pt>



“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.

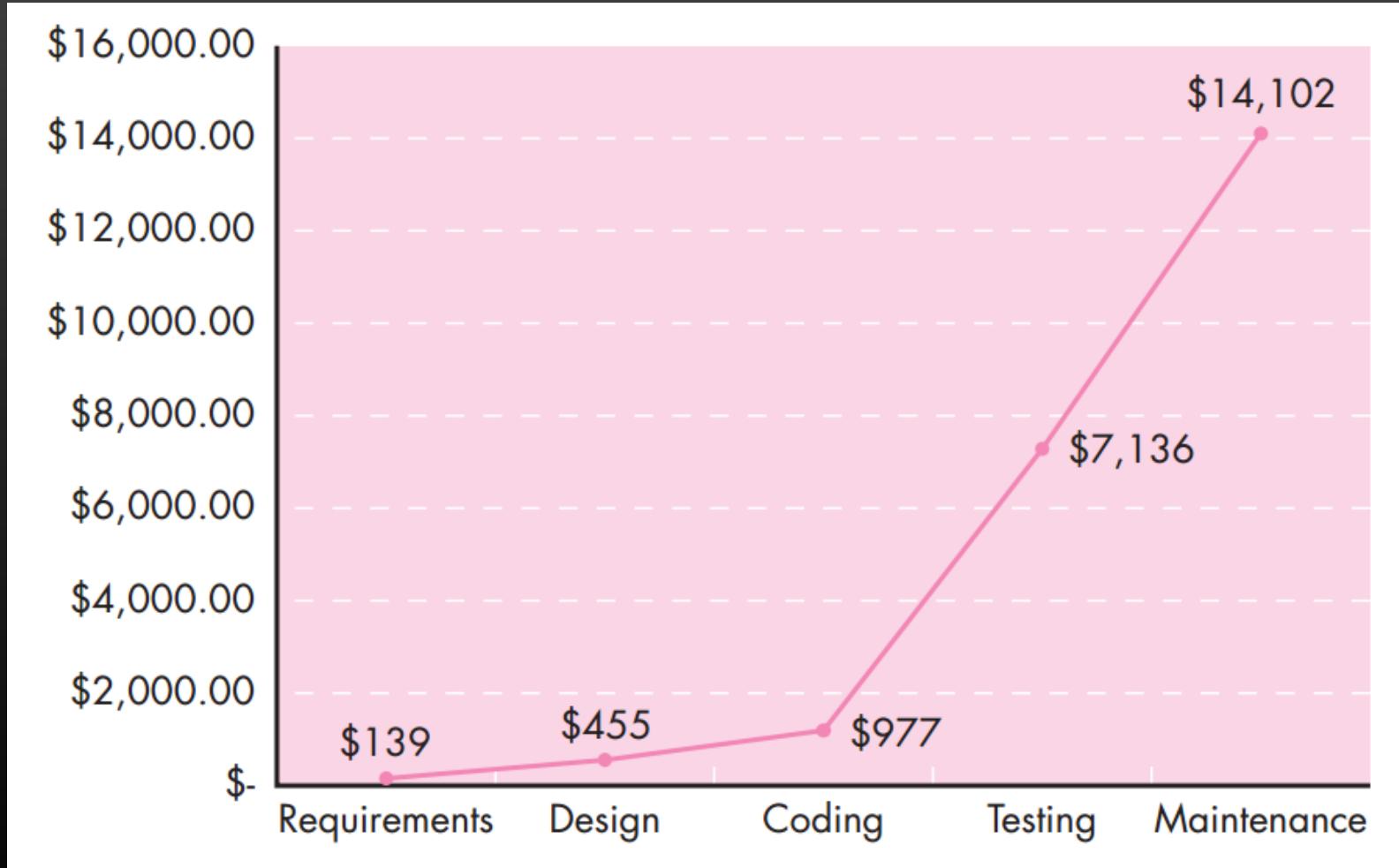
Why it may fail?

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

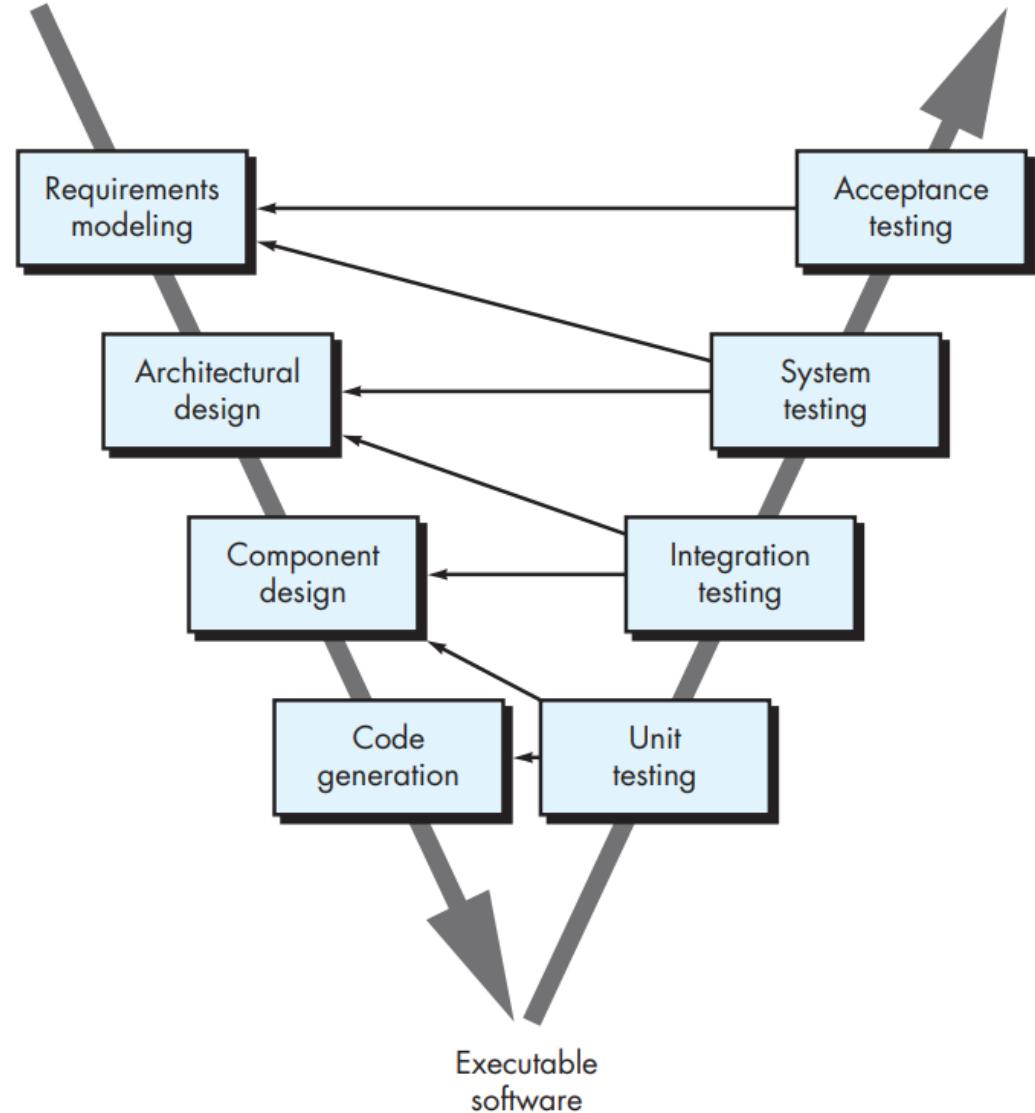


Boehm, B., and V. Basili, "Software Defect Reduction Top 10 List," IEEE Computer,
vol. 34, no. 1, January 2001, pp. 135-137. <http://doi.ieeecomputersociety.org/10.1109/2.962984>¹⁷

Waterfall variation: V-Model

FIGURE 4.2

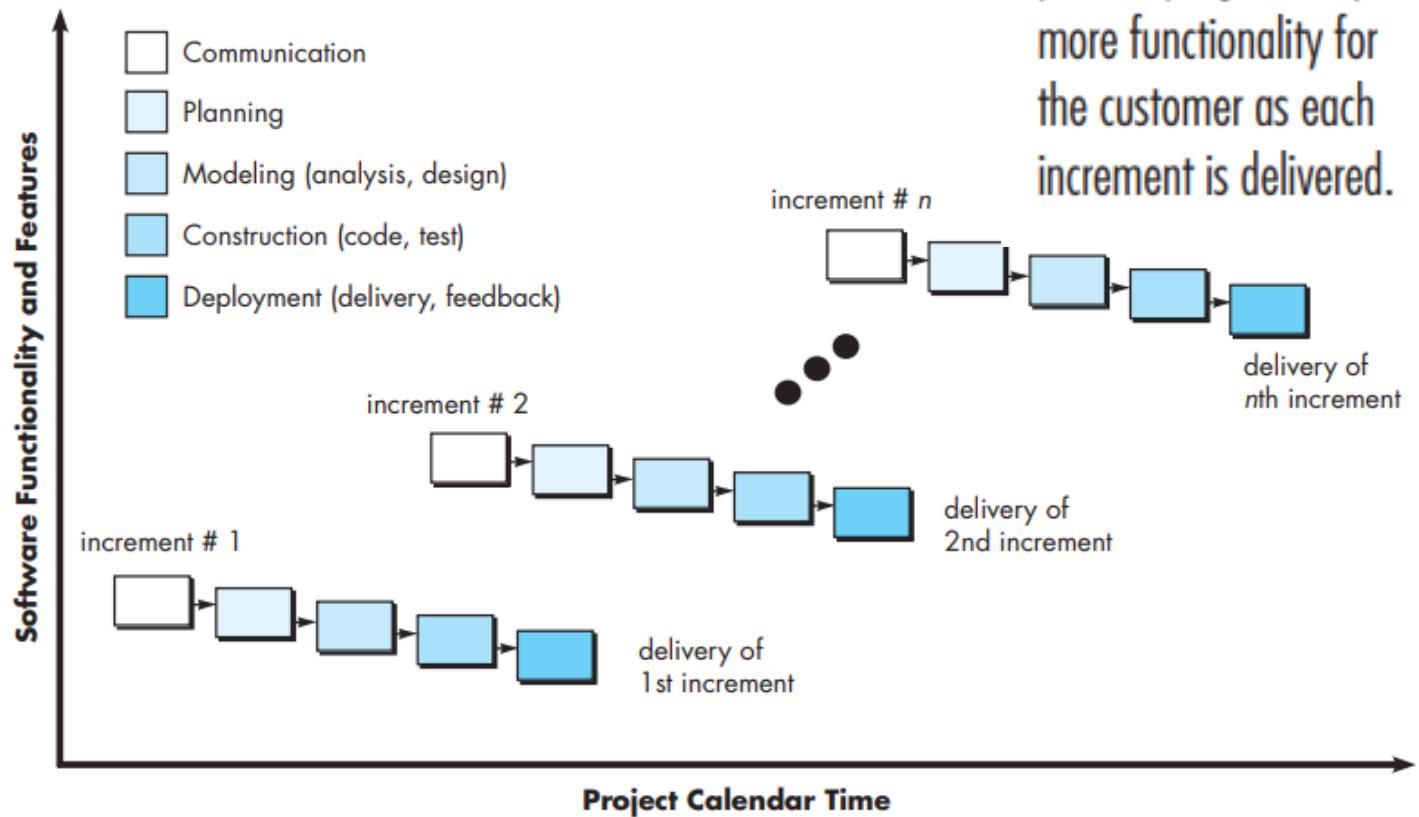
The V-model



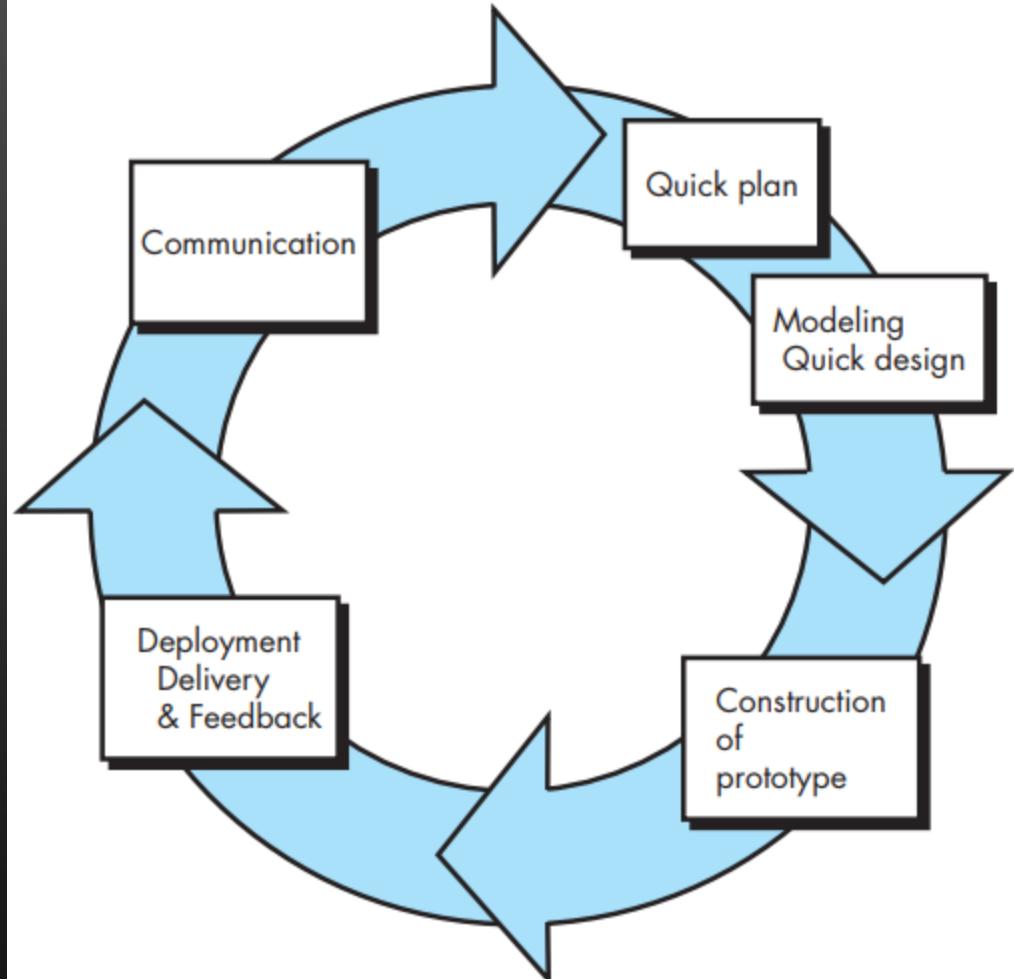
The incremental model delivers a series of releases, called increments, that provide progressively more functionality for the customer as each increment is delivered.

FIGURE 4.3

The incremental model

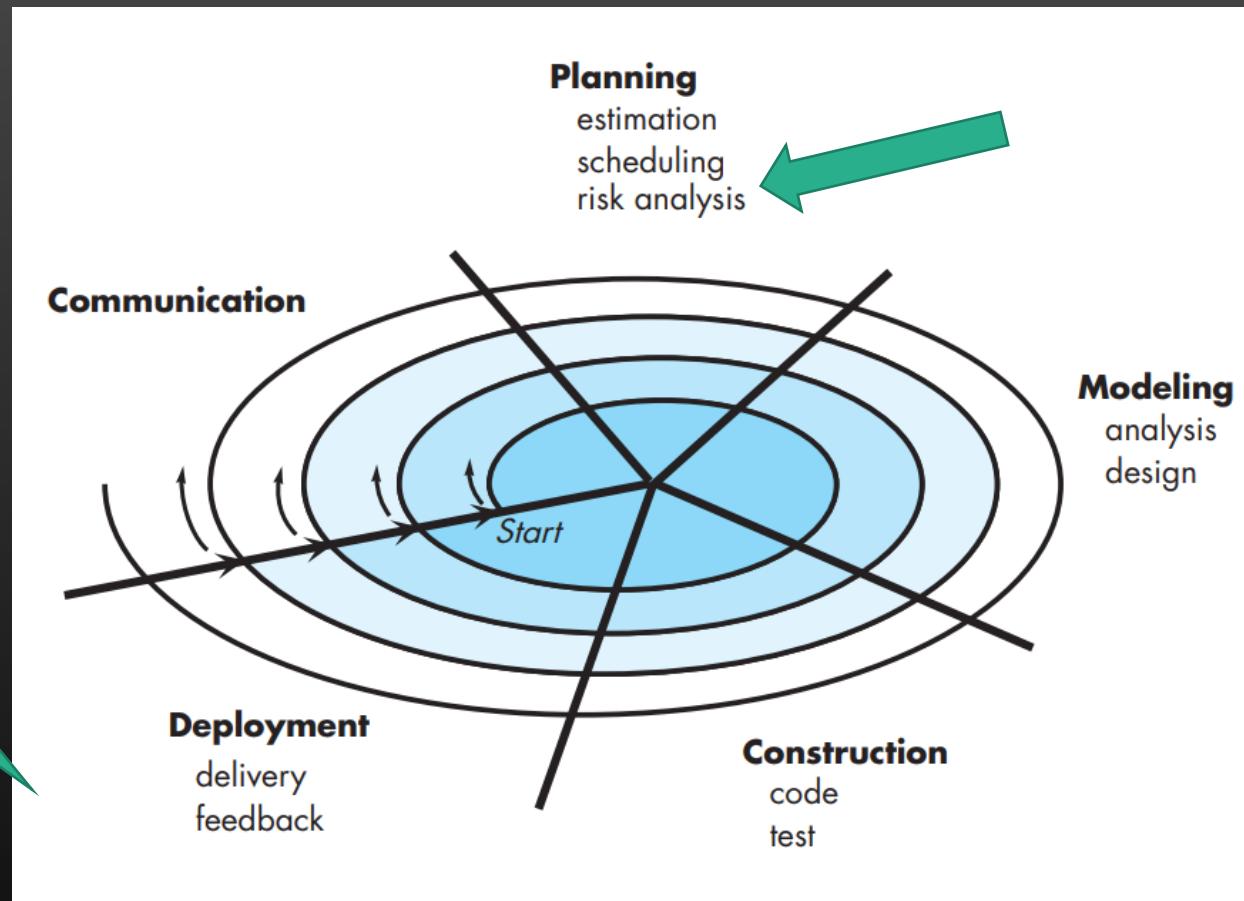


Evolutionary: prototyping



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



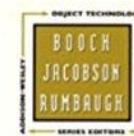
Using the spiral model, software is developed in a series of evolutionary releases. During early iterations, the release might be a model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.

THE UNIFIED SOFTWARE DEVELOPMENT PROCESS

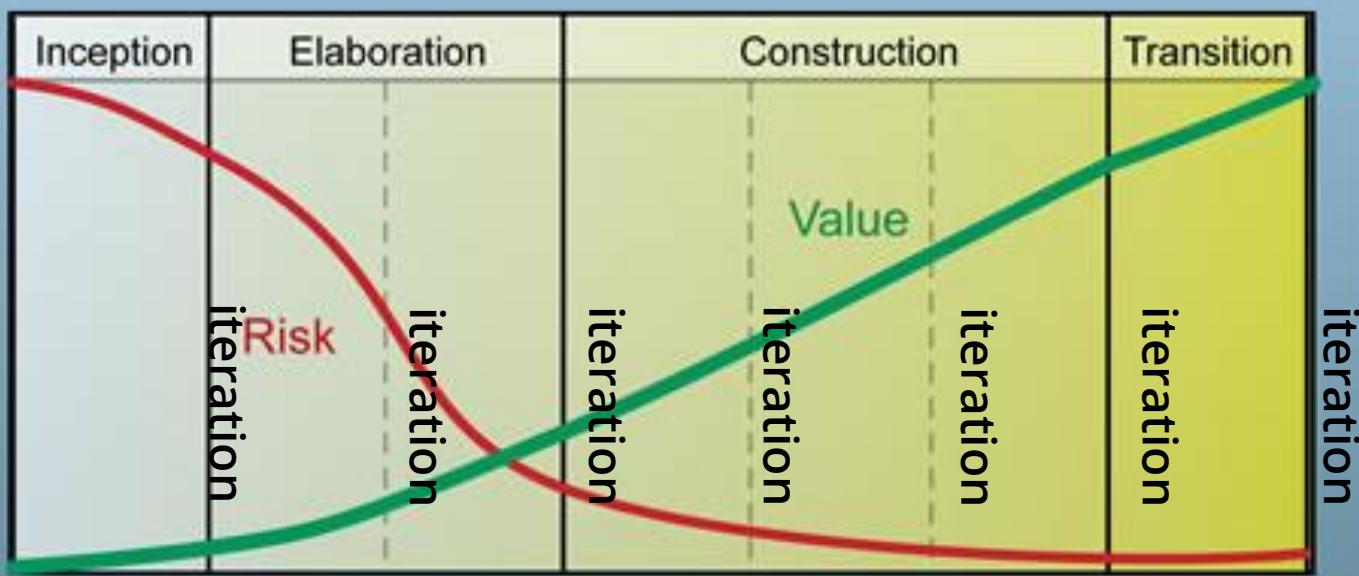
IVAR JACOBSON
GRADY BOOCHE
JAMES RUMBAUGH



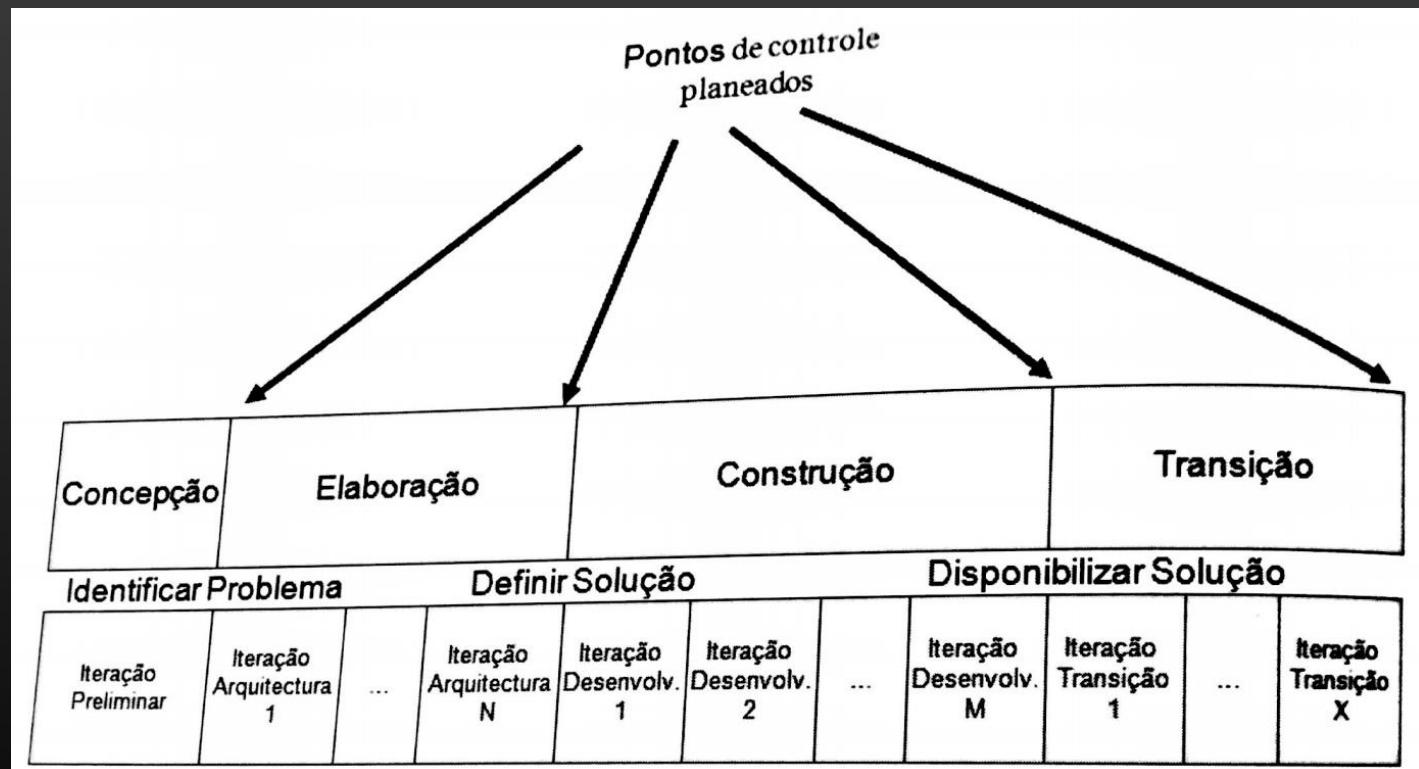
The complete guide
to the Unified
Process from the
original designers



Project Lifecycle



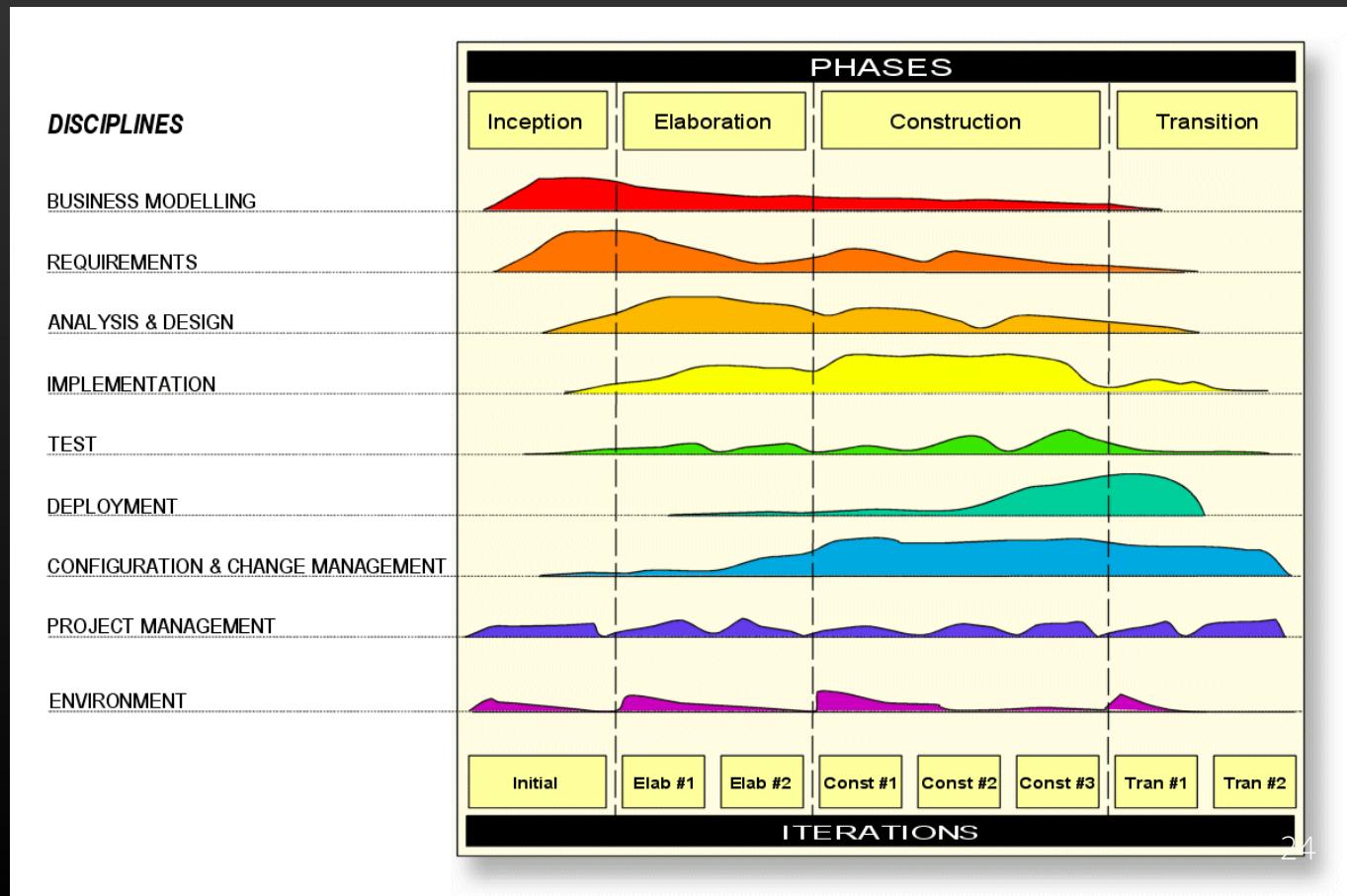
PT: Fases, iterações e pontos de controlo



OpenUP/Unified Process activities

The UP offers an approach to the SDLC visualized as a matrix, crossing different **technical disciplines** with evolving **iterations** in the project. (Note: UP phases ≠ SDLC phases)

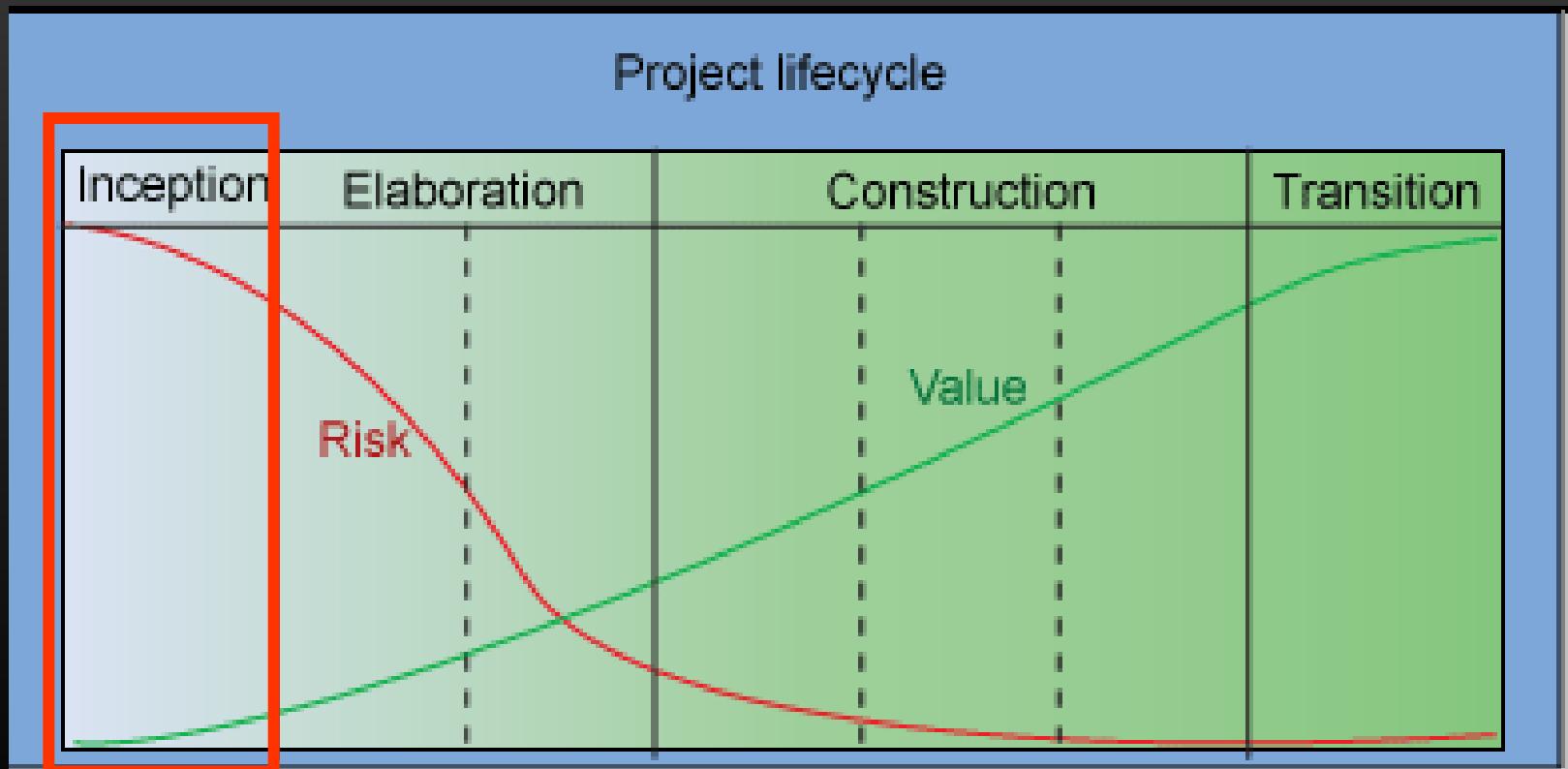
Requirements analysis is mainly performed at the beginning of the project (requirements baseline) but also during the iterations (evolutionary requirements).

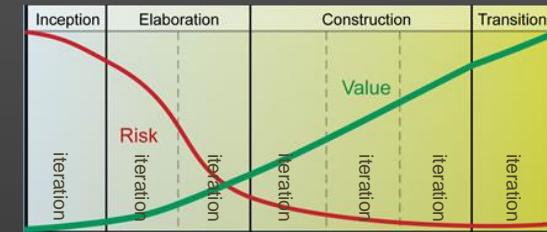


The phases: Inception

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'

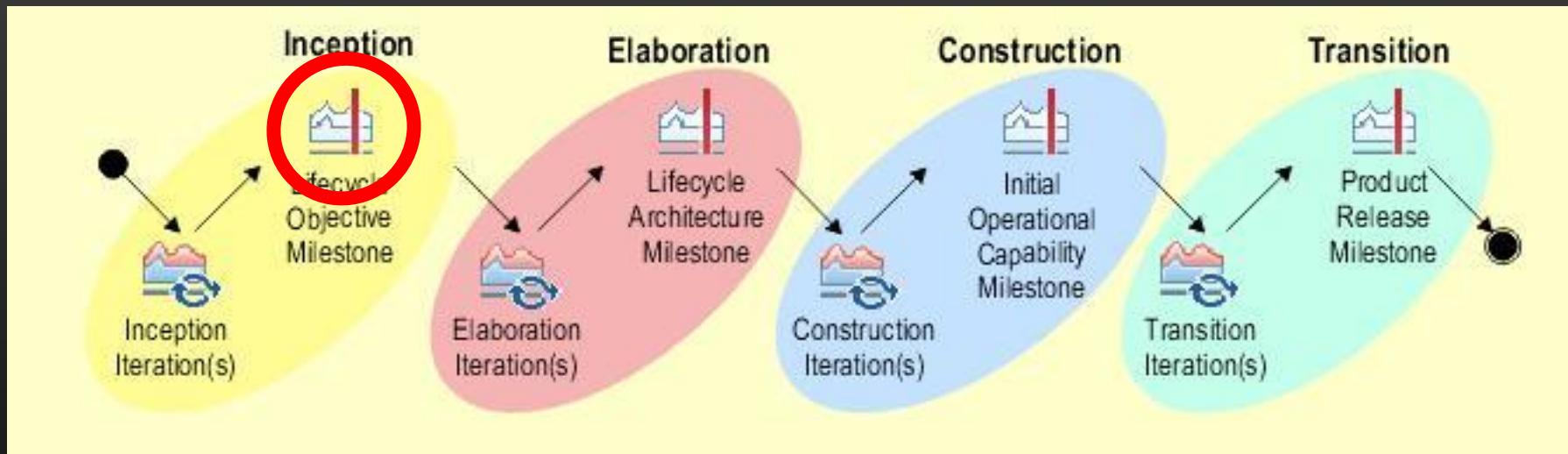
Manage project scope

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.

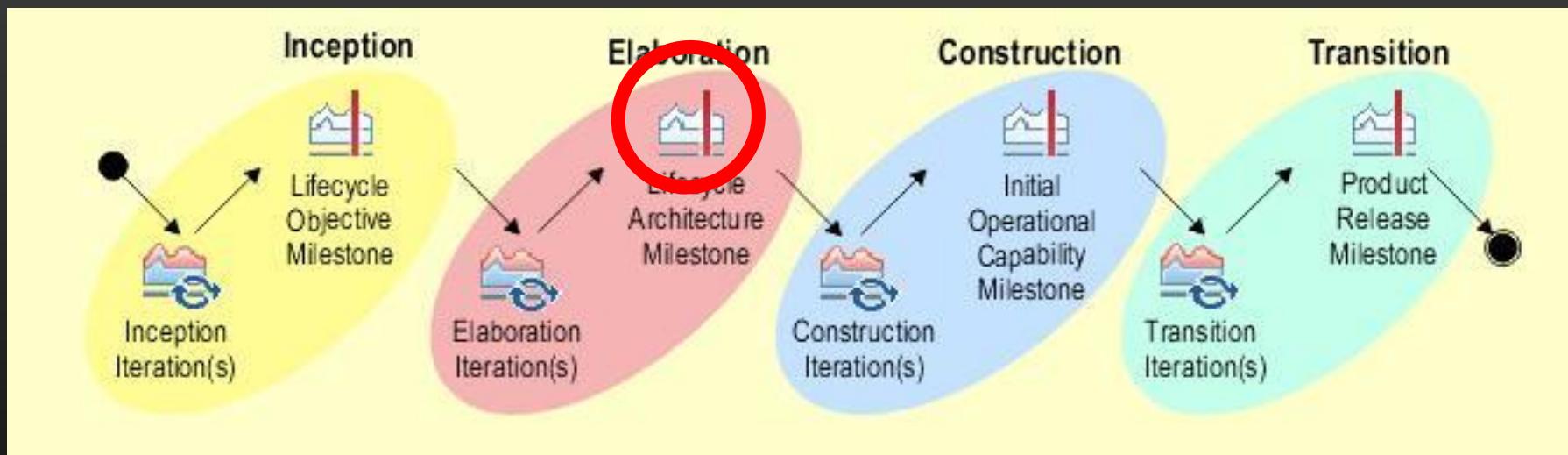
Roughly 10% of code is implemented.

Drive architecture with key use cases

20% of use cases drive 80% of the architecture

Credit: Per Kroll (IBM)

Milestones: Elaboration

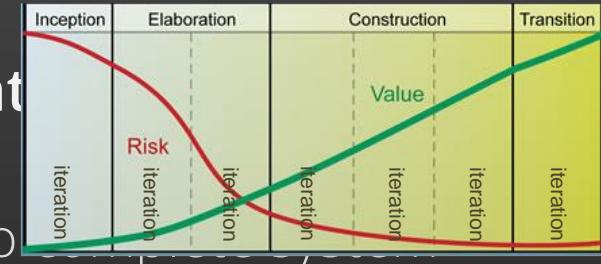


Lifecycle Architecture Milestone. At this point, a baseline of requirements is agreed to, you examine the detailed system objectives and scope, the choice of architecture, and the resolution of the major risks. The milestone is achieved when the architecture has been validated.

Construction: Build The Product

Incrementally define, design, implement and more scenarios

Incrementally evolve executable architecture to
Evolve architecture as you go along



Frequent demonstrations and partial deployment

Partial deployment strategy depends greatly on what system you build

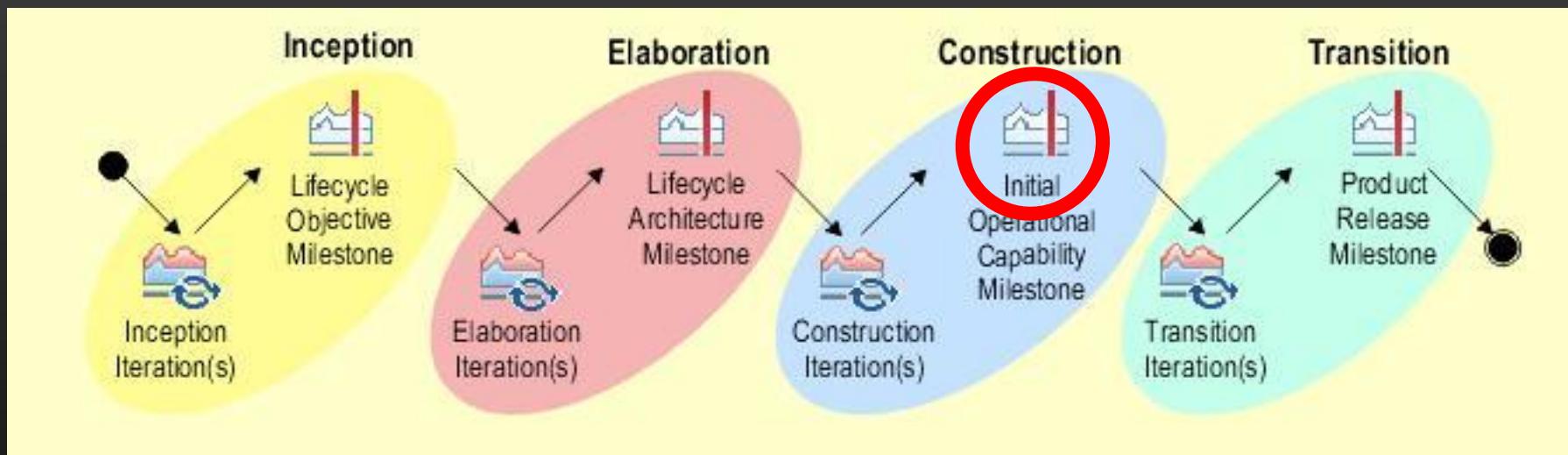
Daily build with automated build process

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

Complex test environments
Safety or mission critical systems

Credit: Per Kroll (IBM)

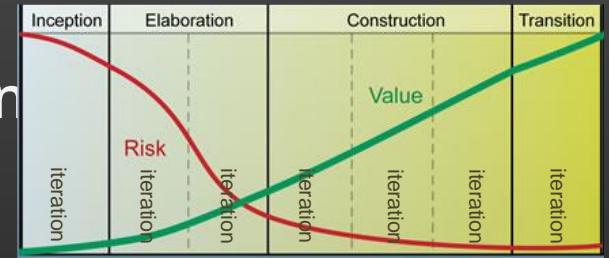
Milestones: Construction



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
to stabilizing and tuning



Produce incremental 'bug-fix' releases

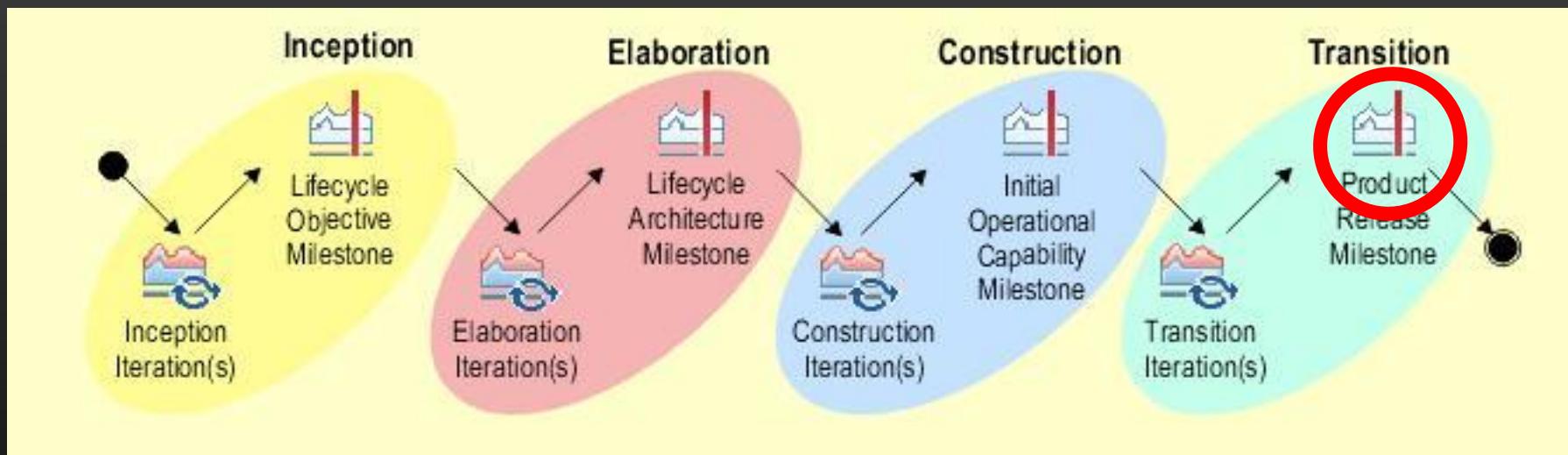
Update user manuals and deployment
documentation

Execute cut-over

Conduct "post-mortem" project analysis

Credit: Per Kroll (IBM)

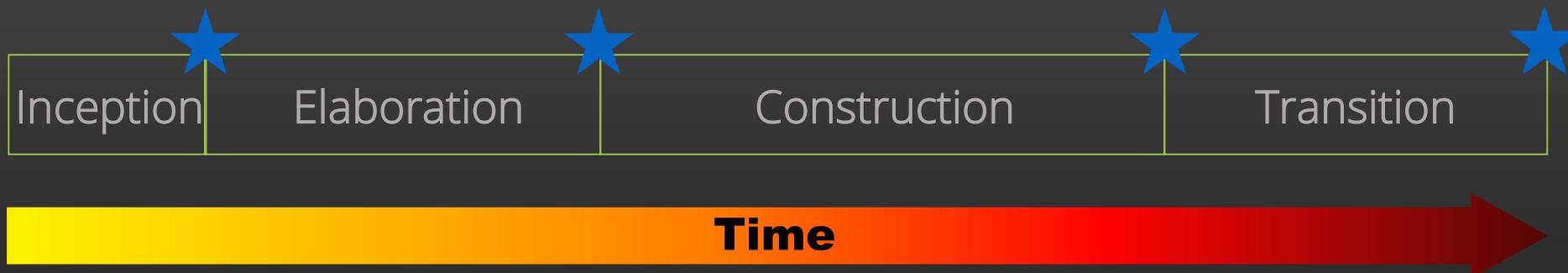
Milestones: Transition



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 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

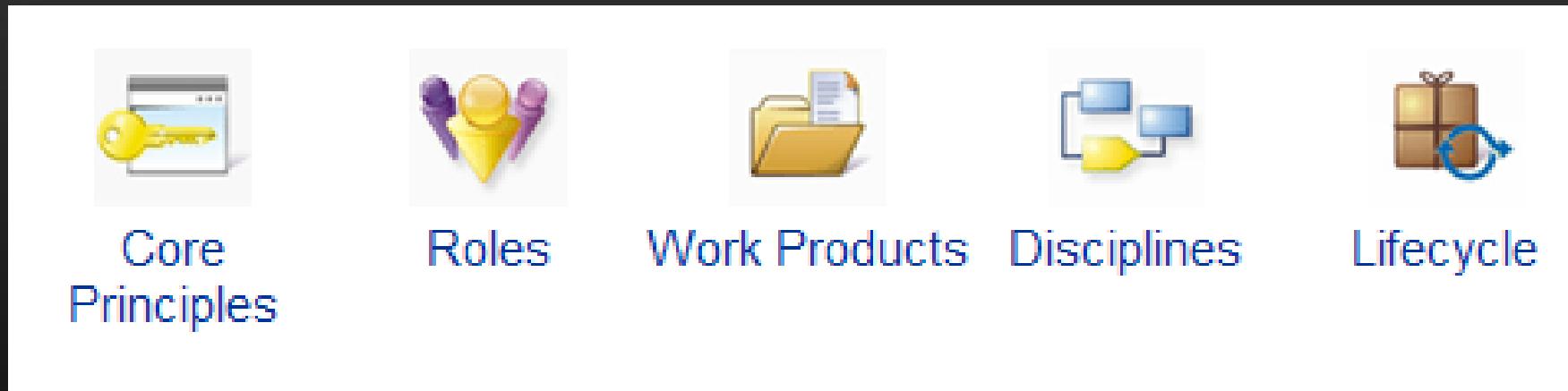
Transition: **Validate and deploy solution**

Stakeholder acceptance, cutover to production

O SDLC é concretizado em metodologias de desenvolvimento

Adotar um processo de engenharia testado & (a)provado

O que é que inclui um processo?



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

What is “Agility” in software development?

Effective (rapid and adaptive) response to change

Effective communication among all stakeholders

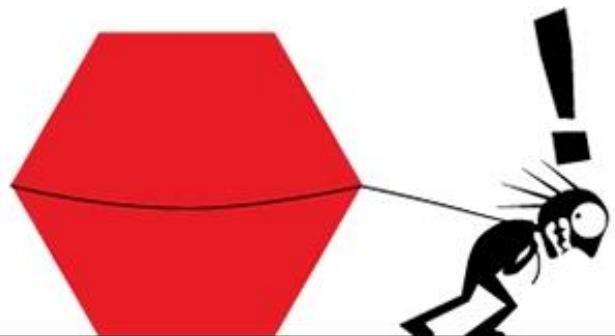
Drawing the customer onto the team

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

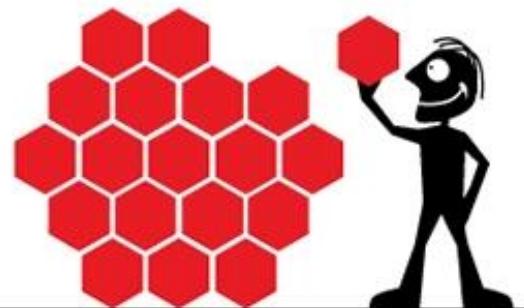
Yielding...

Rapid, incremental delivery of software

One project? Micro-projects?



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



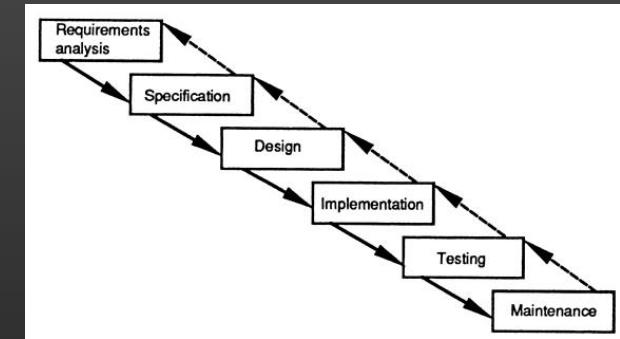
*'It's so much better delivering this
project in bite-sized sections'*

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

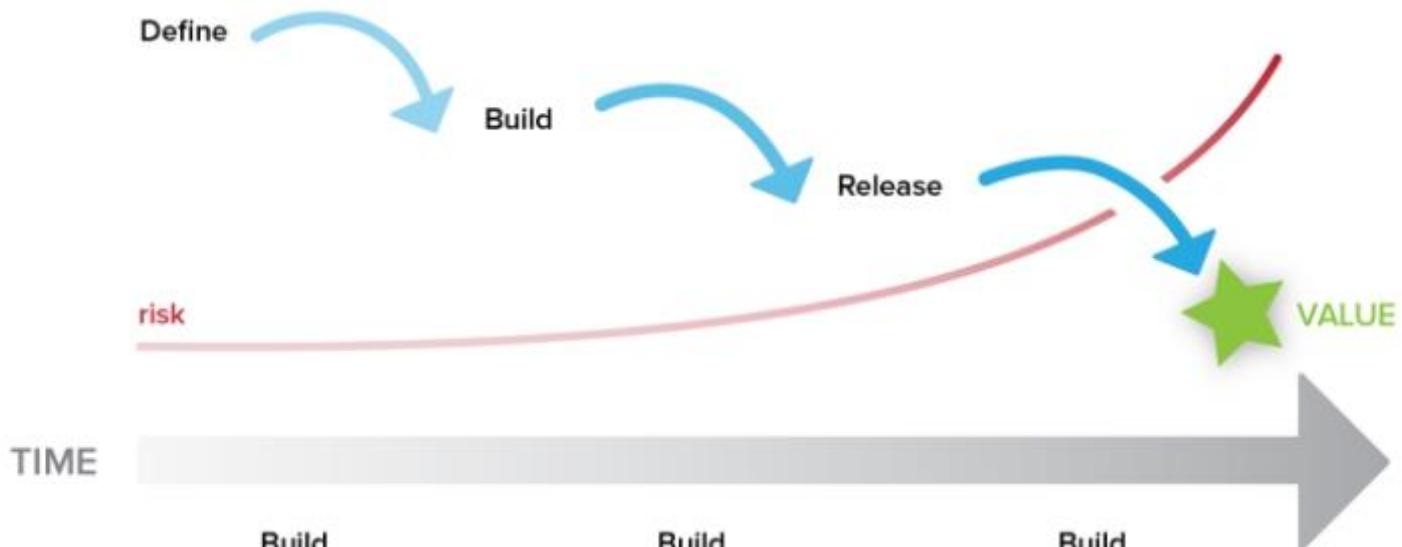
Iterative development focuses on short and value-oriented cycles → Agile



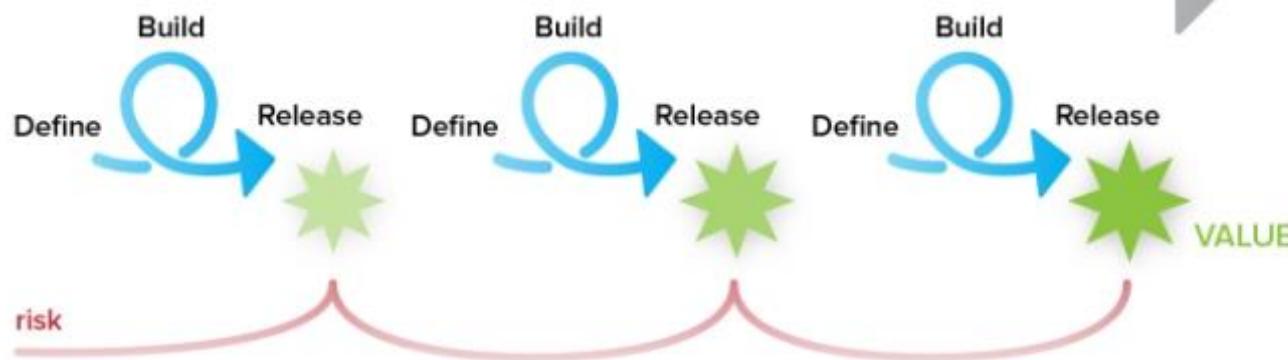
Each iteration augments the solution by integrating some executable result



WATERFALL



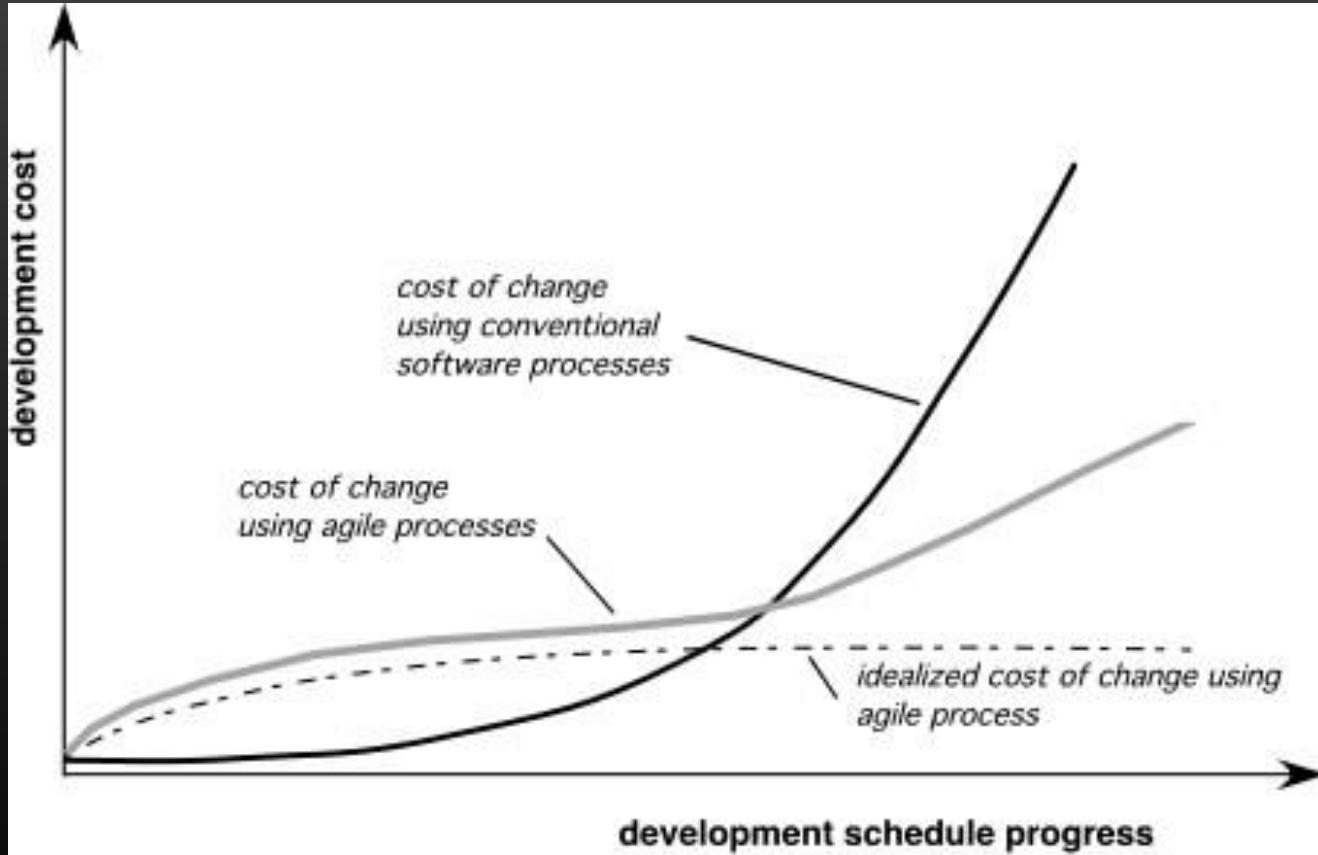
TIME



AGILE

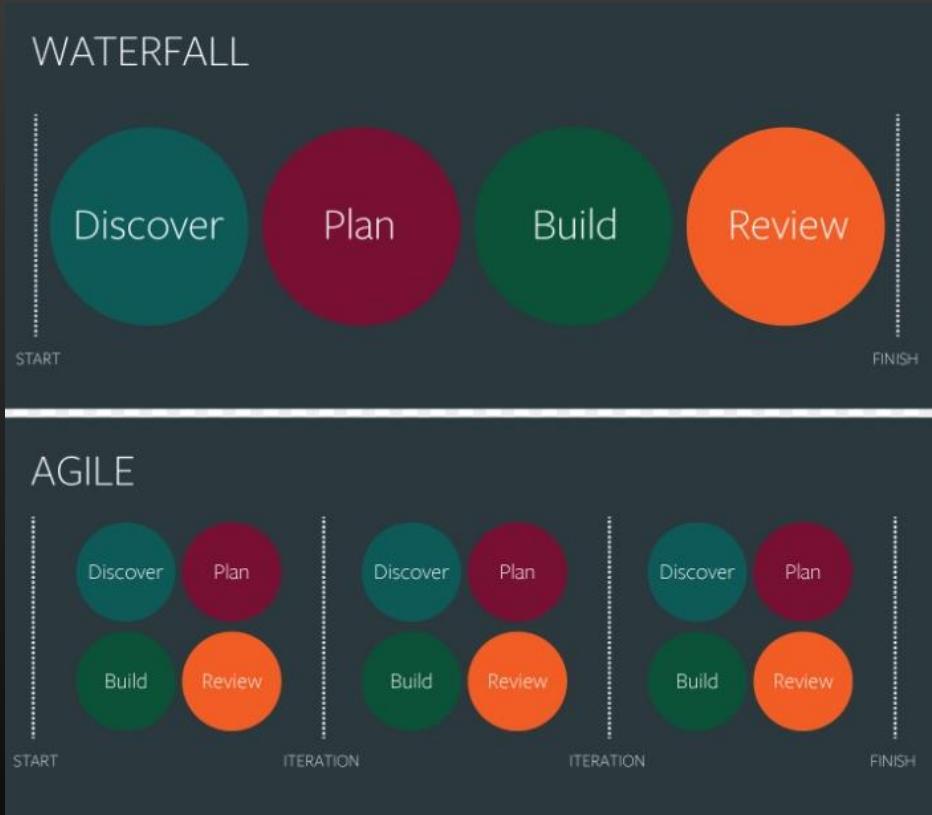
<https://blog.ganttpro.com/en/waterfall-vs-agile-with-advantages-and-disadvantages/>

Agility and the Cost of Change



The cost of change increases nonlinearly as the project progresses

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?

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

Is OpenUP an agile process?

Readings & references

Core readings	Suggested readings
<ul style="list-style-type: none">• [Pressman'15] – Chap. 4, 5	<ul style="list-style-type: none">• [Dennis'15] – Chap 1.

47006- ANÁLISE E MODELAÇÃO DE SISTEMAS

Functional modeling with use cases

Ilídio Oliveira

v2020-10-16 | TP-04

Learning objectives for this lecture

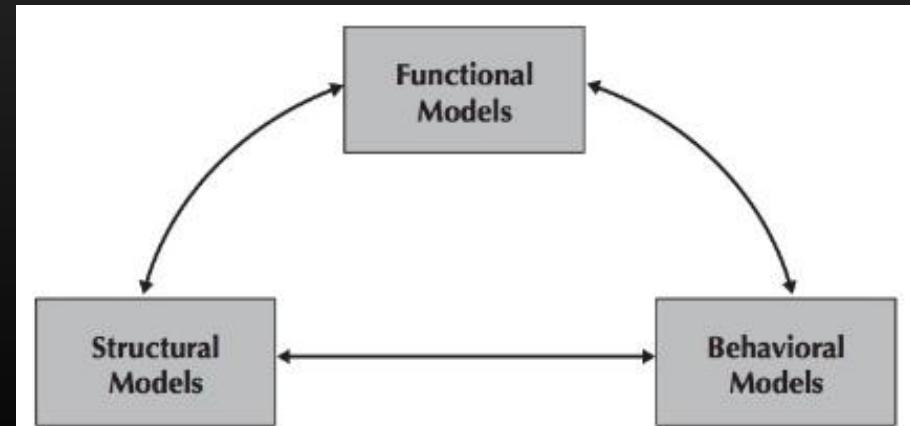
- Explain the activities related to use case modeling in the SDLC and those that rely on UC as inputs
- Explain the process used to discover use cases
- Read and create Use Case diagrams
- Explain the components of a use case description and the associated templates
- Create functional models of business processes using use-case diagrams, activity diagrams, and use case descriptions.

3 complementary views on a system

Functional. What the system does? The external behavior of the system (black-box).

Structural. What are the parts ("things") the system is made off?

Behavior. How does the system do the operations? View of the system interactions along time.



Functional models

Develop use-cases from the requirements

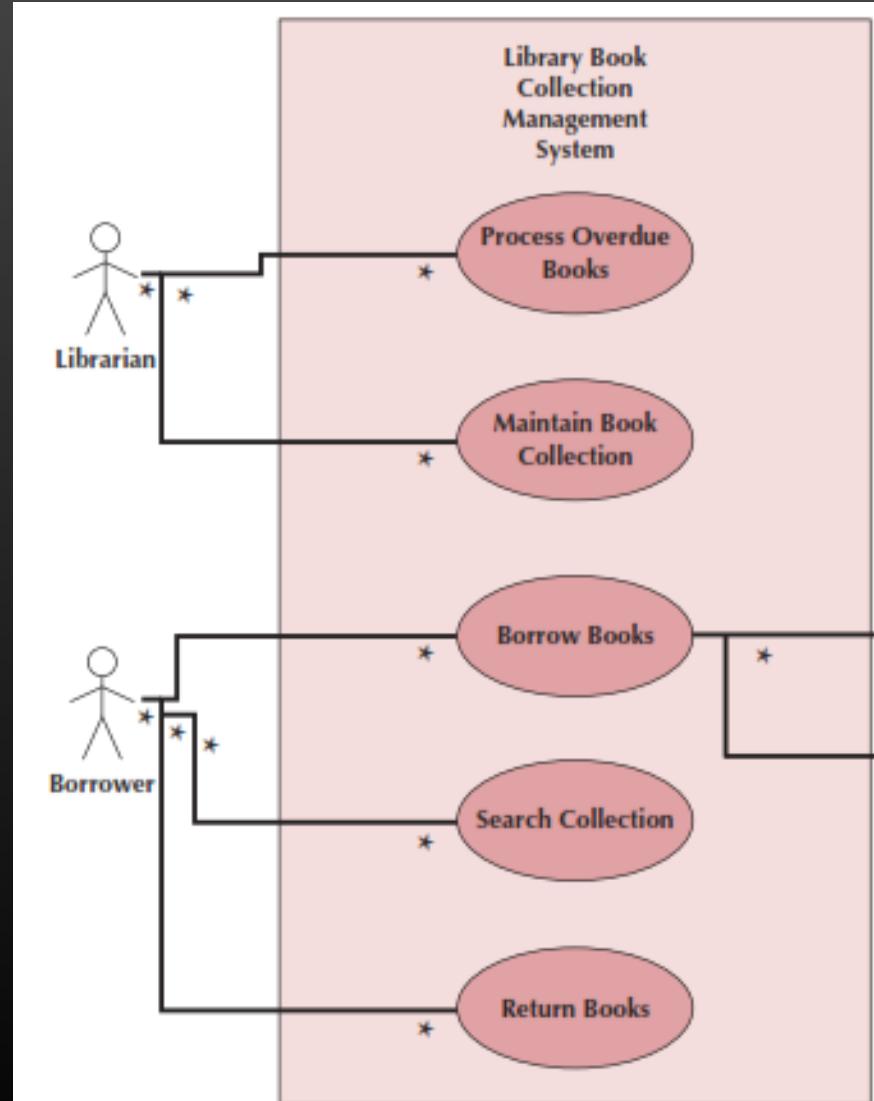
Use-case: how a business system interacts with its environment

Includes a diagram and a description to depict the discrete activities that the users perform

Develop activity diagrams from the use-cases

These model the business processes or how a business operates

Used to illustrate the movement of objects (data) between activities

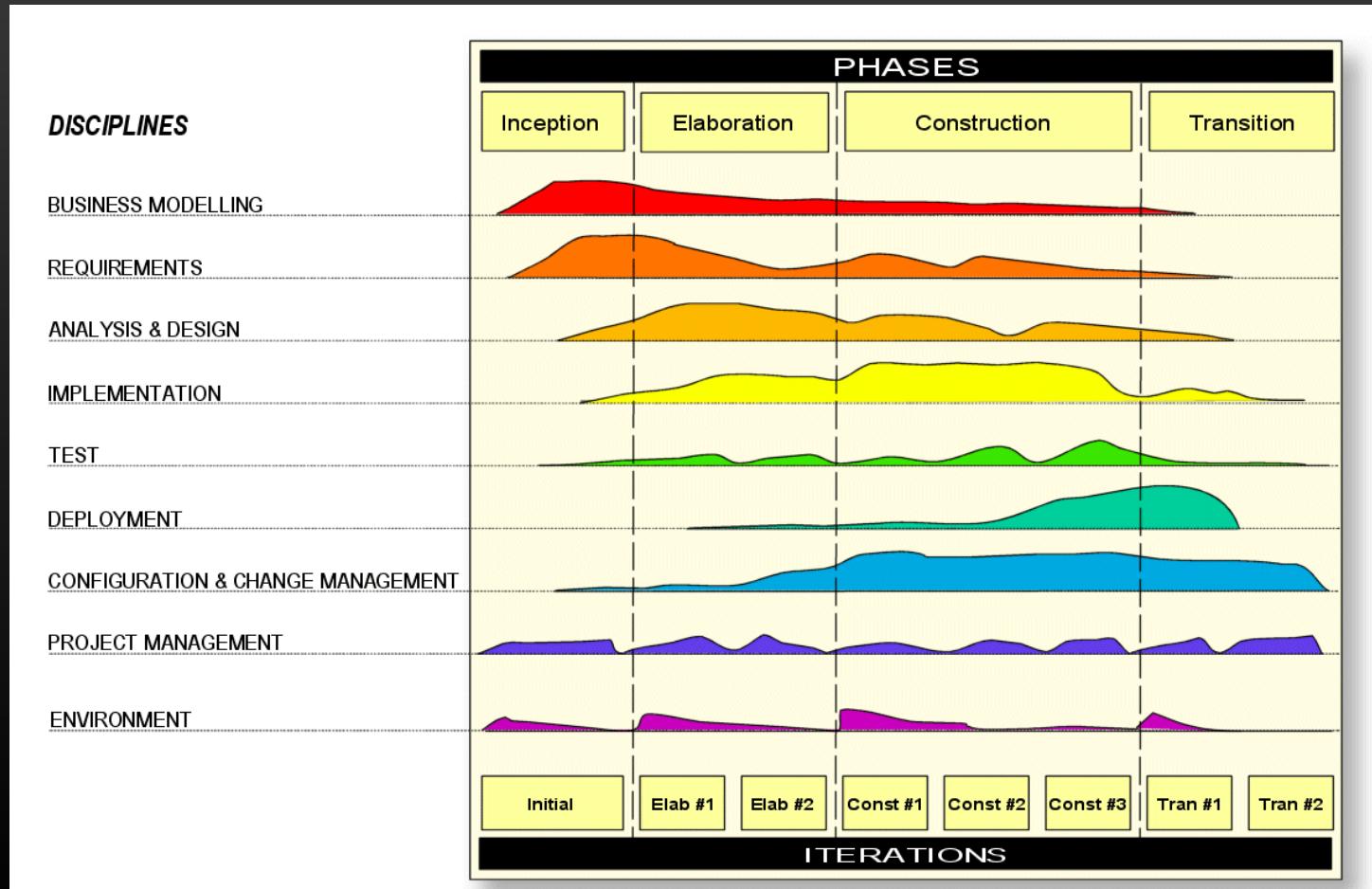


Unified Process is Use-case centric

OpenUP/Unified Process activities

The UP offers an approach to the SDLC visualized as a **matrix**, crossing different **technical disciplines** with evolving **iterations** in the project. (Note: UP phases ≠ SDLC phases)

Requirements analysis is mainly performed at the beginning of the project (requirements baseline) but also during the iterations (evolutionary requirements).



Which main elicitation approaches exist?

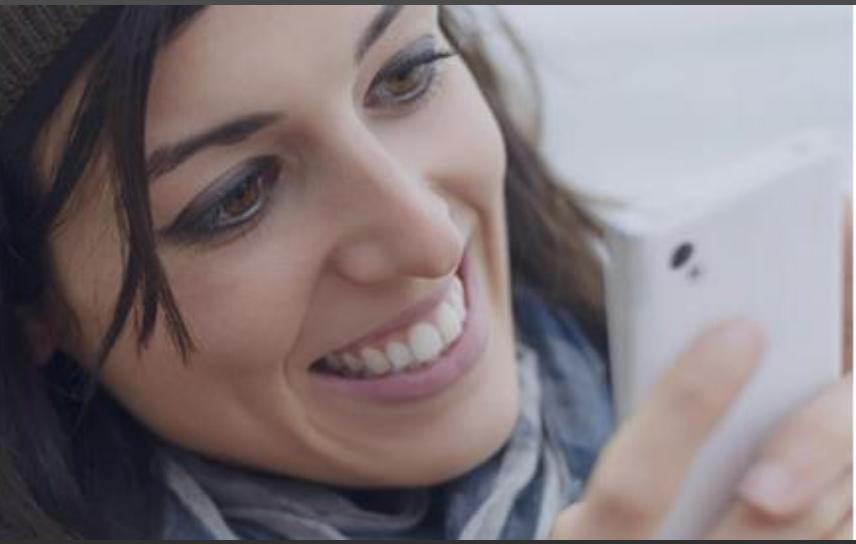
What is the goal the user wants to achieve? vs.

What capability should the system process?



Usage-centric or product-centric?

Requirements elicitation typically takes either a usage-centric or a product-centric approach, although other strategies also are possible. The usage-centric strategy emphasizes understanding and exploring user goals to derive the necessary system functionality. The product-centric approach focuses on defining features that you expect will lead to marketplace or business success. A risk with product-centric strategies is that you might implement features that don't get used much, even if they seemed like a good idea at the time. We recommend understanding business objectives and user goals first, then using that insight to determine the appropriate product features and characteristics.



- Consulta de Saldos e Movimentos de Contas e Cartões de Crédito;
- Consulta de Posição Integrada;
- Transferências para beneficiários, contas BPI ou contas de outros Bancos (zona SEPA);
- Pagamentos de Serviços, Estado e Telemóveis;
- Criação e gestão de beneficiários de transferências e de pagamentos predefinidos;
- Constituição, reforço e mobilização de contas poupança objetivo;
- Cartões: pedido de alteração de Limites de Crédito, alteração de opção de pagamento e pagamento de Saldo ou Reforço;
- Consulta de catálogo e aquisição de Produtos Prestígio;
- Acesso a contactos, localização e serviços de Balcões, Centros de Investimento e Centros de Empresas;
- Login com código de 4 dígitos ou com impressão digital.

 Contas Cartões Crédito Poupança e Investimento Imóveis Seguros À sua Medida

Eu quero...



Ser cliente
da Caixa



Comprar
uma casa



Comprar
um Carro



Viajar



Preparar o Futuro
dos meus Filhos



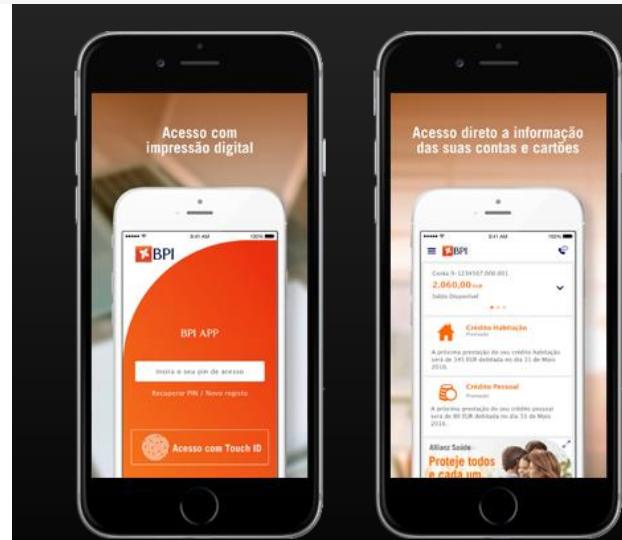
Poupar para
o Futuro



Preparar a
minha Reforma



Proteger a minha
Família

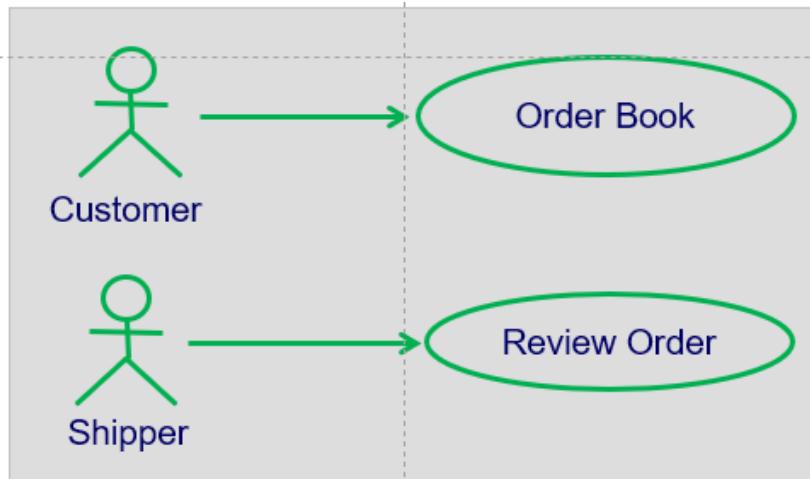


OpenUP recommended practices

Use Case Driven Development

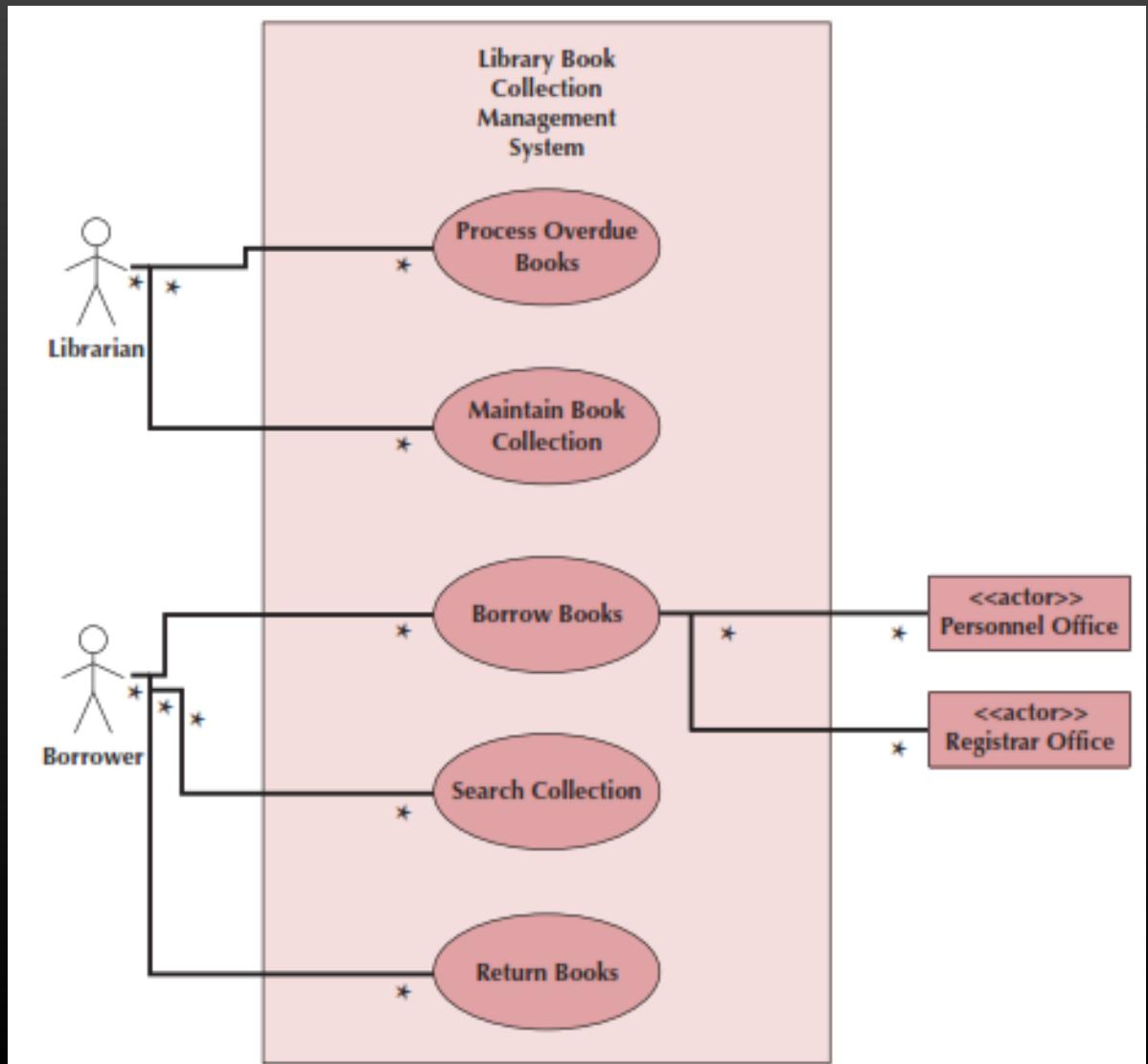


- This practice describes how to capture requirements with a combination of use cases and system-wide requirements, and then drive development and testing from those use cases.



Example Use-Case

Library Book Collection Management - System Use Case Diagram



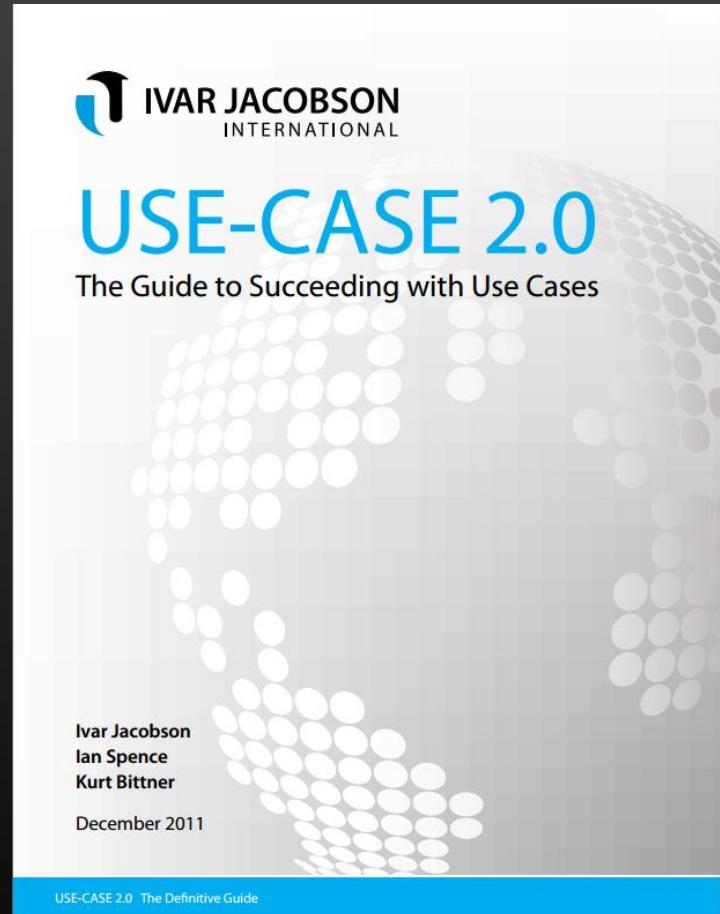
Credit: Dennis et al, "Systems Analysis and Design: An Object Oriented Approach with UML", 5th ed Oliveira

Use case (Caso de Utilização: CaU)

A motivation from an actor to use the system and a sequence of actions the system performs to produce a result with value to that actor.

Requires:

- The focus is on the user and the episodes of usage
- Focus on understanding what the users perceive as a value (motivations to use a system)



[https://www.ivarjacobson.com/
publications/white-papers/use-
case-ebook](https://www.ivarjacobson.com/publications/white-papers/use-case-ebook)

Who (the actor) does what (interaction) on the system, whith a goal in mind (motivation)

UC is a flow of actions that produces a result with values to a particular actor (including the variations in the flow, related to the same goal)

The required yet sufficient amount of activity (interaction) that produces a result of interest for an actor

UC provides a context to a related set of requirements.
(Favors a more coherent division of the system)

Use-cases discovery (in-class...)

→ ...



<https://www.menti.com/9ysw7pk2ge>



**The use case model includes an overview
(diagrams) + description of scenarios**

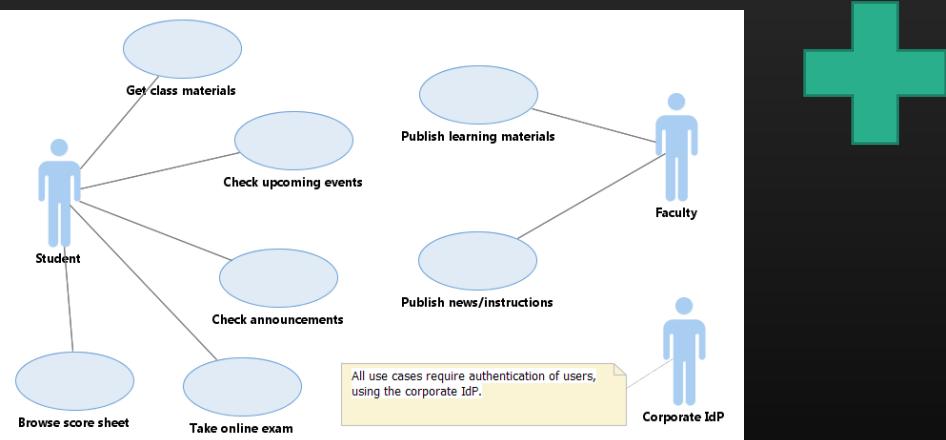
Use cases documentation (use cases modeling)

Visual overview

Use-cases Diagram (UML)

Description of scenarios

Structured narratives (text)
Optional: activity models.



ID and Name:	UC-4 Request a Chemical
Created By:	Lori
Date Created:	8/22/13
Primary Actor:	Requester
Secondary Actors:	Buyer, Chemical Stockroom, Training Database
Description:	The Requester specifies the desired chemical to request by entering its name or chemical ID number or by importing its structure from a chemical drawing tool. The system either offers the Requester a container of the chemical from the chemical stockroom or lets the Requester order one from a vendor.
Trigger:	Requester indicates that he wants to request a chemical.
Preconditions:	PRE-1. User's identity has been authenticated. PRE-2. User is authorized to request chemicals. PRE-3. Chemical inventory database is online.
Postconditions:	POST-1. Request is stored in the CTS. POST-2. Request was sent to the Chemical Stockroom or to a Buyer.
Normal Flow:	4.0 Request a Chemical from the Chemical Stockroom 1. Requester specifies the desired chemical. 2. System lists containers of the desired chemical that are in the chemical stockroom, if any. 3. System gives Requester the option to View Container History for any container. 4. Requester selects a specific container or asks to place a vendor order (see 4.1). 5. Requester enters other information to complete the request. 6. System stores the request and notifies the Chemical Stockroom.
Alternative Flows:	4.1 Request a Chemical from a Vendor 1. Requester searches vendor catalogs for the chemical (see 4.1.E1). 2. System displays a list of vendors for the chemical with available container sizes, grades, and prices. 3. Requester selects a vendor, container size, grade, and number of containers. 4. Requester enters other information to complete the request. 5. System stores the request and notifies the Buyer.
Exceptions:	4.1.E1 Chemical Is Not Commercially Available 1. System displays message: No vendors for that chemical. 2. System asks Requester if he wants to request another chemical (3a) or to exit (4a). 3a. Requester asks to request another chemical. 3b. System starts normal flow over. 4a. Requester asks to exit. 4b. System terminates use case.
Priority:	High
Frequency of Use:	Approximately 5 times per week by each chemist, 200 times per week by chemical

Types of Use Cases

Purpose	Amount of information	
	Overview/High-level	Detail
Essential	High-level overview of issues essential to understanding required functionality	Detailed description of issues essential to understanding required functionality
Real	High-level overview of a specific set of steps performed on the real system once implemented	Detailed description of a specific set of steps performed on the real system once implemented

Use case model

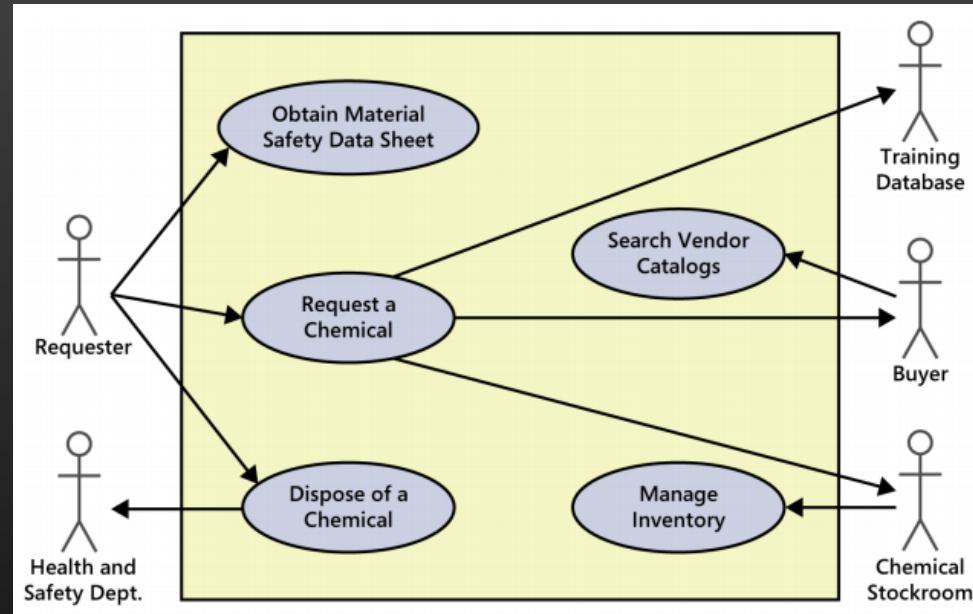
The **users (or systems)** who interact with the system with some objective in mind are modeled as **actors**.

The ways/episodes in which the system will be used to achieve these goals are modeled as **use cases**.

A **use case diagram** is a model of the useful ways to use a system. This allows you to quickly grasp the scope of the system – what is included and what is not – and give the team a global view of what the system will do.

The **intentional high-level perspective** offers a birds-eye view, without losing ourselves in the details of the internal parts of the system.

→ A context/instrument to **discuss and discover** the system requirements!



Elements of the use case model

Actor

Any entity (the role of someone, another system,...) external to the system under specification, which interacts with it

Scenario

A particular situation/history of use of the system, i.e., a possible path in the execution of a case of use

E.g.: payment of the purchase with cash or card; failure in payment due to lack of card authorization;...

Use case

Set of scenarios related to the same goal
An episode of using + variants.

Associations

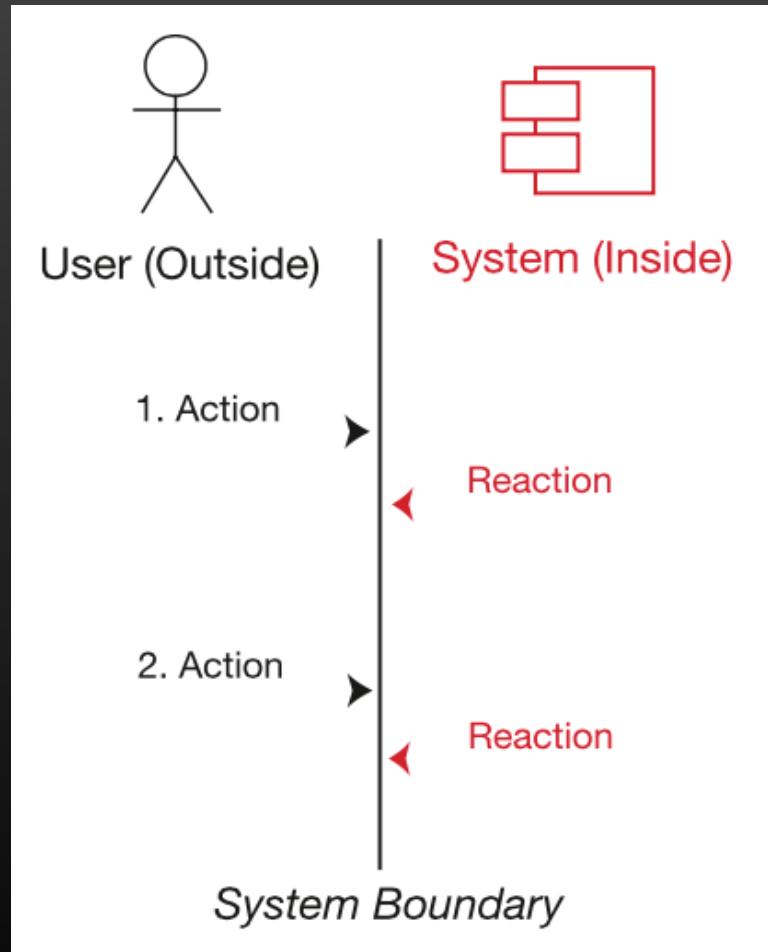
Relationships of interest between actors/CaU, CaU/CaU, actors/actors
Most relevant:
Actor *participates in* a use case.

Use cases hold/encapsulate interaction scenarios

The use case captures a dialog between the actor(s) and the system

CaU: Pay at checkout

1. Customer arrives at POS checkout with goods to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price is calculated from a set of price rules.
Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any)



The Use Case has several flows

The typical flow

The “normal script” for the actor/system collaboration

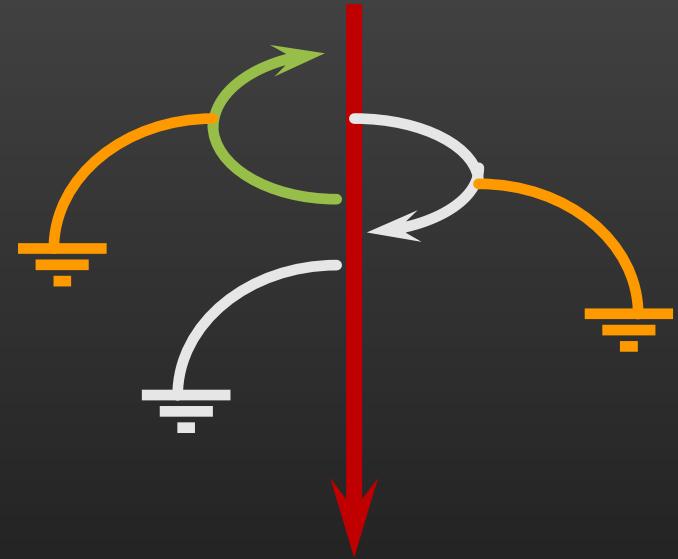
Several alternative flows

Variations due to options from the users

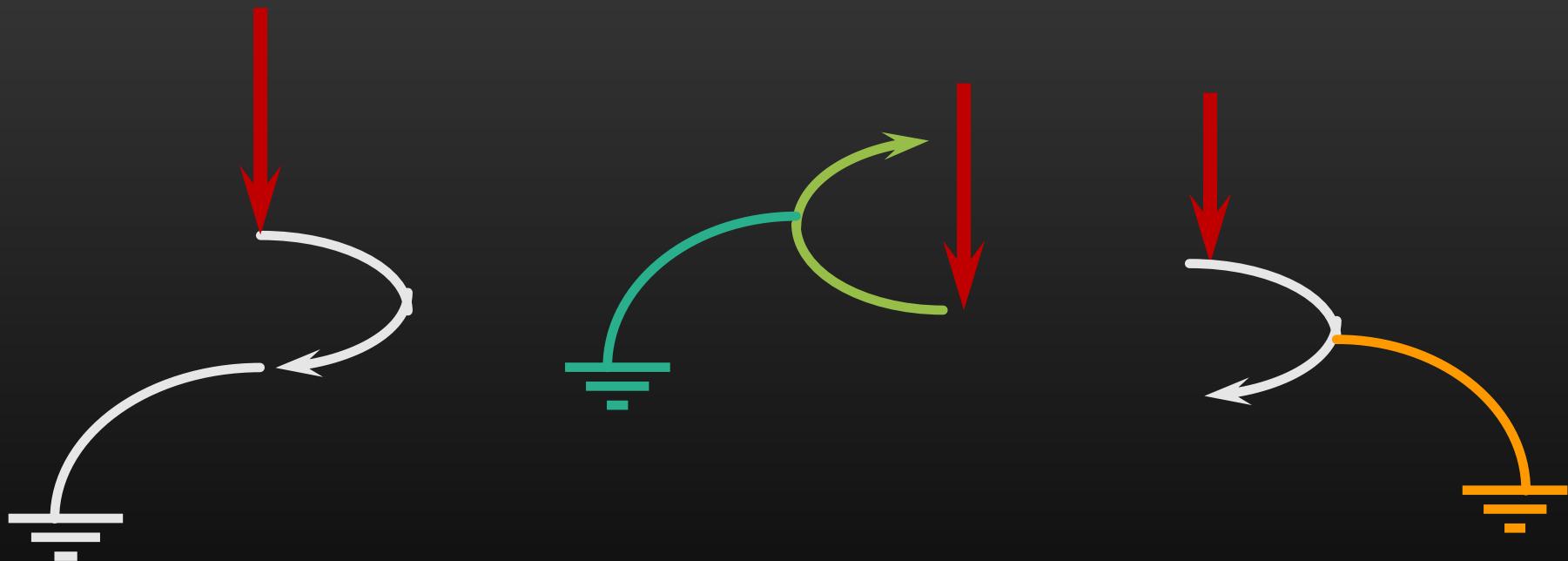
Uncommon/special cases

Exception conditions and errors (and what should be done)

e.g.: how to enter the product description (in the sale scenario) when the bar code can not be read?

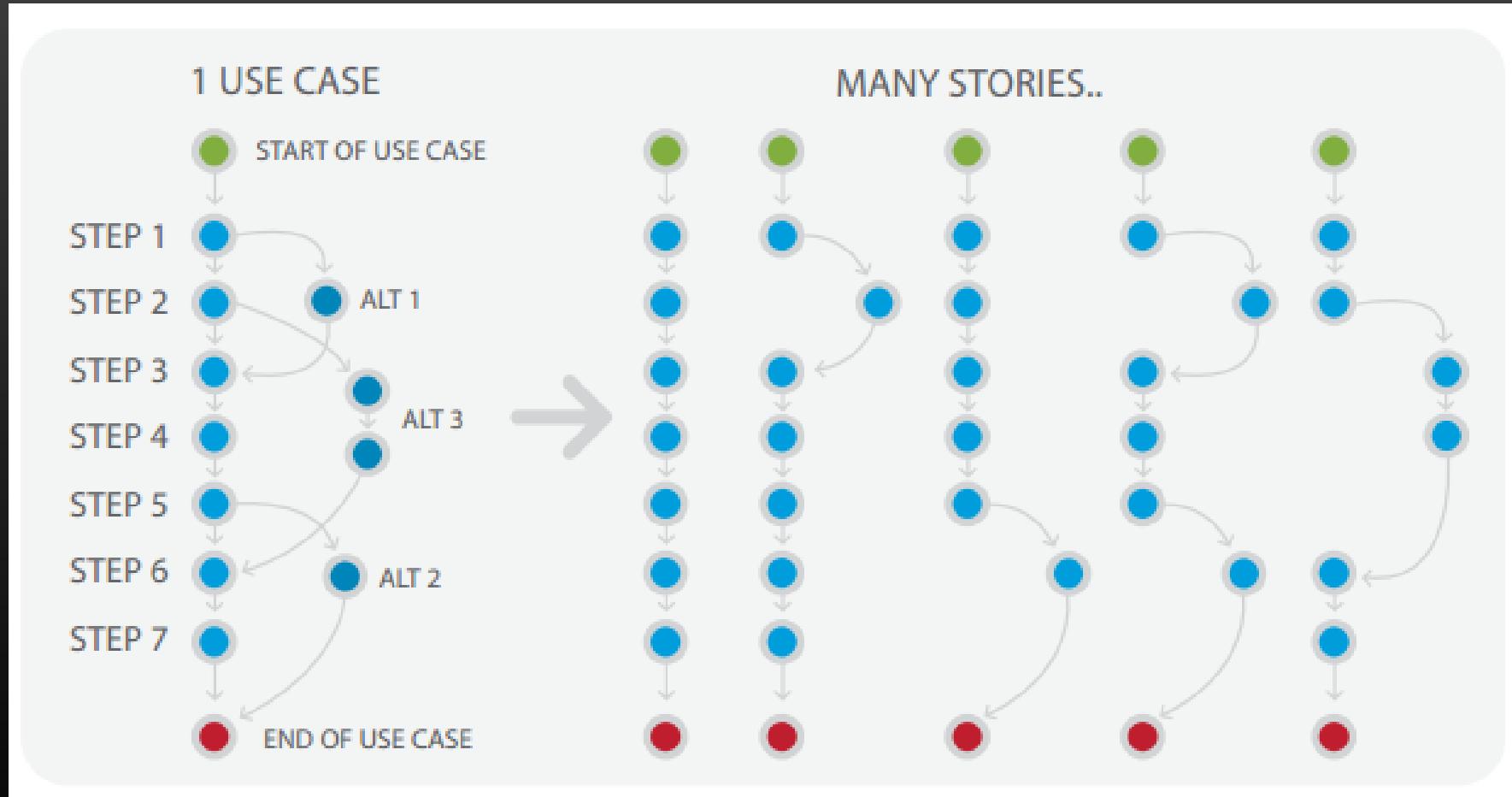


A scenario is an instance of a use case. It is one flow through a use case.



One use case has different flows

Still, all flows are related to the same goal in-mind.

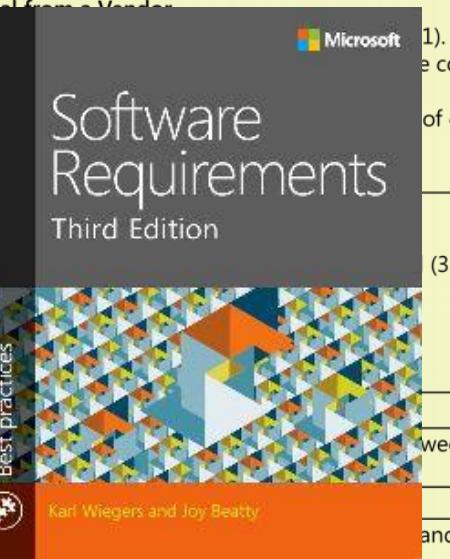


Essential elements of a use case specification

- ✓ A unique identifier and a succinct name that states the user goal
- ✓ A brief textual description that describes the purpose of the use case
- ✓ A trigger condition that initiates execution of the use case
- ✓ Zero or more preconditions that must be satisfied before the use case can begin
- ✓ One or more postconditions that describe the state of the system after the use case is successfully completed
- ✓ A numbered list of steps that shows the sequence of interactions between the actor and the system – a dialog – that leads from the preconditions to the postconditions

→ Credit: Wiegers 2013

ID and Name:	UC-4 Request a Chemical
Created By:	Lori
Date Created:	8/22/13
Primary Actor:	Requester
Secondary Actors:	Buyer, Chemical Stockroom, T
Description:	The Requester specifies the desired chemical to request by entering number or by importing its structure from a chemical drawing tool. the Requester a container of the chemical from the chemical stock order one from a vendor.
Trigger:	Requester indicates that he wants to request a chemical.
Preconditions:	PRE-1. User's identity has been authenticated. PRE-2. User is authorized to request chemicals. PRE-3. Chemical inventory database is online.
Postconditions:	POST-1. Request is stored in the CTS. POST-2. Request was sent to the Chemical Stockroom or to a Buyer.
Normal Flow:	4.0 Request a Chemical from the Chemical Stockroom 1. Requester specifies the desired chemical. 2. System lists containers of the desired chemical that are in the che 3. System gives Requester the option to View Container History for 4. Requester selects a specific container or asks to place a vendor or 5. Requester enters other information to complete the request. 6. System stores the request and notifies the Chemical Stockroom.
Alternative Flows:	4.1 Request a Chemical from a Vendor 1. Requester search 2. System displays a and prices. 3. Requester selects 4. Requester enters 5. System stores the
Exceptions:	4.1.E1 Chemical Is I 1. System displays n 2. System asks Requ 3a. Requester asks t 3b. System starts no 4a. Requester asks t 4b. System terminat
Priority:	High
Frequency of Use:	Approximately 5 tim stockroom staff
Business Rules:	BR-28, BR-31
Other Information:	The system must be any of the supports



"Request a Chemical" use case specification

ID and Name:	UC-4 Request a Chemical	
Created By:	Lori	Date Created: 8/22/13
Primary Actor:	Requester	Secondary Actors: Buyer, Chemical Stockroom, Training Database
Description:	The Requester specifies the desired chemical to request by entering its name or chemical ID number or by importing its structure from a chemical drawing tool. The system either offers the Requester a container of the chemical from the chemical stockroom or lets the Requester order one from a vendor.	
Trigger:	Requester indicates that he wants to request a chemical.	
Preconditions:	PRE-1. User's identity has been authenticated. PRE-2. User is authorized to request chemicals. PRE-3. Chemical inventory database is online.	
Postconditions:	POST-1. Request is stored in the CTS. POST-2. Request was sent to the Chemical Stockroom or to a Buyer.	
Normal Flow:	4.0 Request a Chemical from the Chemical Stockroom <ol style="list-style-type: none">1. Requester specifies the desired chemical.2. System lists containers of the desired chemical that are in the chemical stockroom, if any.3. System gives Requester the option to View Container History for any container.4. Requester selects a specific container or asks to place a vendor order (see 4.1).5. Requester enters other information to complete the request.6. System stores the request and notifies the Chemical Stockroom.	
Alternative Flows:	4.1 Request a Chemical from a Vendor <ol style="list-style-type: none">1. Requester searches vendor catalogs for the chemical (see 4.1.E1).2. System displays a list of vendors for the chemical with available container sizes, grades, and prices.3. Requester selects a vendor, container size, grade, and number of containers.4. Requester enters other information to complete the request.5. System stores the request and notifies the Buyer.	
Exceptions:	4.1.E1 Chemical Is Not Commercially Available <ol style="list-style-type: none">1. System displays message: No vendors for that chemical.2. System asks Requester if he wants to request another chemical (3a) or to exit (4a).	

Get the story details: use case specification

The purpose of a CaU narrative is to tell the story how the system and its actors work together to achieve a particular goal.

The narratives:

Outline the stories used to explore requirements and identify scenarios.

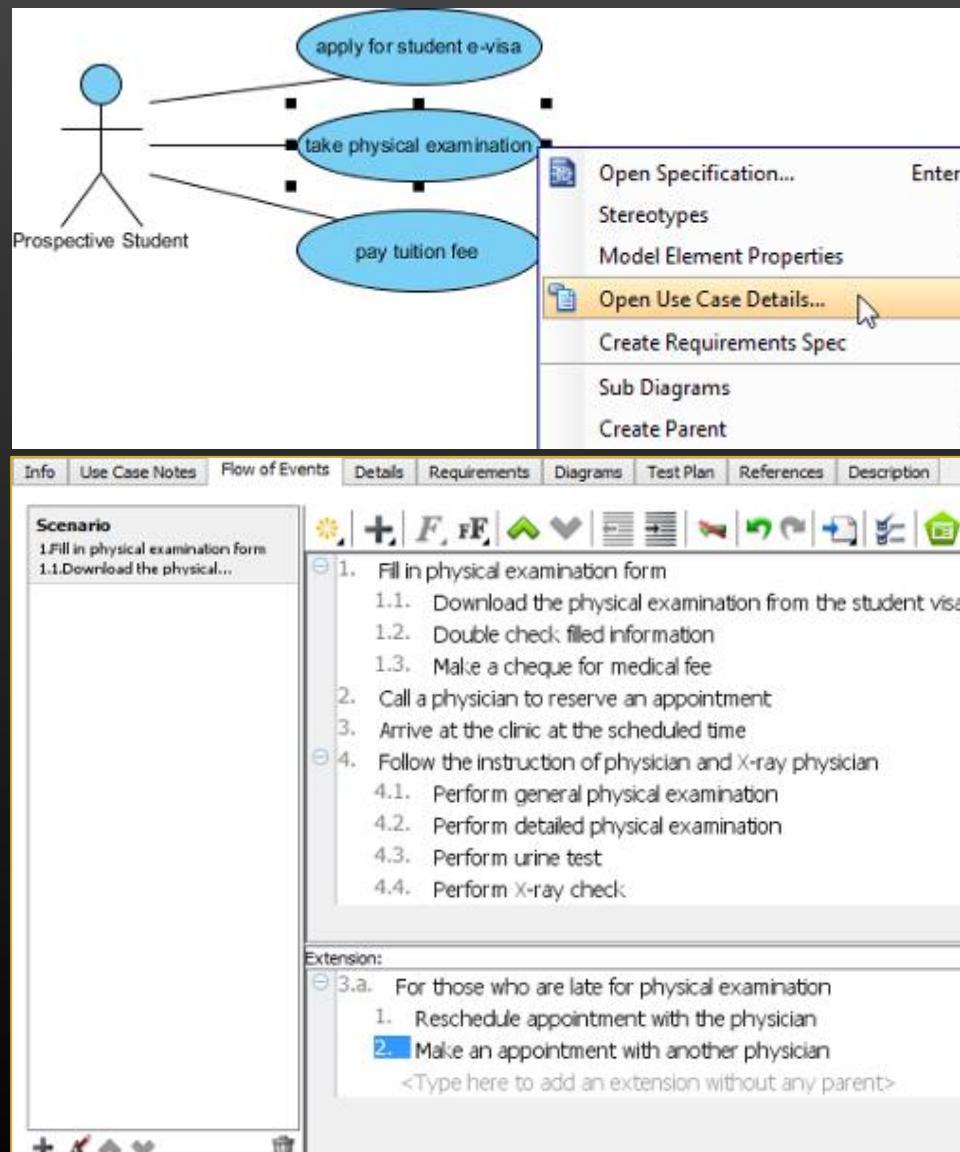
They describe a sequence of actions, including variants, that actors and a system can perform to achieve a goal

They are presented with a set of flows that describe how an actor uses a system to achieve a goal and what the system does (responsibilities)

Captures/organizes information on the requirements needed to support development activities.

The narratives can be presented in many ways

Wiki, reports in MS Word, embedded directly in modeling tools (e.g: VisualParadigm).



The use case details describe an interaction

HOW TO WRITE A USE CASE: THE THREE MAGIC QUESTIONS

Well, OK, this whole chapter describes how to write a use case. But when writing use cases, you need to keep asking the following three fundamental questions:¹

1. What happens?

(This gets your “sunny-day scenario” started.)

2. And then what happens?

(Keep asking this question until your “sunny-day scenario” is complete.)

3. What else might happen?

(Keep asking this one until you’ve identified all the “rainy-day scenarios” you can think of, and described the related behavior.)

BASIC COURSE:

The Customer clicks the Write Review button for the book currently being viewed, and the system shows the Write Review screen. The Customer types in a Book Review, gives it a Book Rating out of five stars, and clicks the Send button. The system ensures that the Book Review isn't too long or short, and that the Book Rating is within one and five stars. The system then displays a confirmation screen, and the review is sent to a Moderator, ready to be added.

ALTERNATE COURSES:

User not logged in: The user is first taken to the Login screen and then to the Write Review screen once he is logged in.

The user enters a review that is too long (text > 1MB): The system rejects the review and responds with a message explaining why the review was rejected.

The review is too short (< 10 characters): The system rejects the review.

Use case:	Brief description:
Create new assignment	<p>The Teaching Staff creates a new Activity of type Assignment, directly inserting it in the page layout. The assignment must define a title and a time period for submissions and can be configured to work with individual or group submissions. The assignment is listed in the student view and on the specified date (or immediately, if none is given) accepts submissions from registered students.</p>
Use case: <u>Add new assignment</u> Brief description: <p>The Faculty creates assignments for students, directly inserting it in the course page. The assignment defines a time period for submissions and can be configured to work with individual or group submissions. The assignment is listed in the student view and on the specified date (or immediately, if none is given) accepts submissions from students.</p> Basic flow: <ol style="list-style-type: none"> 1. Log-in using corporate IdP. 2. Select desired course. 3. Turn editing mode on. 4. Add Assignment activity in the page layout. 5. Configure Assignment activity. 6. Commit changes. Alternative flows: <p>Step 1: IdP unavailable.</p> <p>Step 4/5: Instead of a new, empty assignment, the user may reuse an existing one.</p> Open issues: <p>Step 3/4. The course is closed. Are changes allowed to past courses?</p> <p>Step 5. The browser does not accept the rich text editor. Default to plain text?</p>	

Use Case Writing Guidelines

1. Write in the form of subject-verb-direct object
2. Make sure it is clear who the initiator of the step is
3. Write from independent observer's perspective
4. Write at about the same level of abstraction
5. Ensure the use case has a sensible set of steps
6. Apply the KISS principle liberally.
7. Write repeating instructions after the set of steps to be repeated

How to discover the Use Cases?

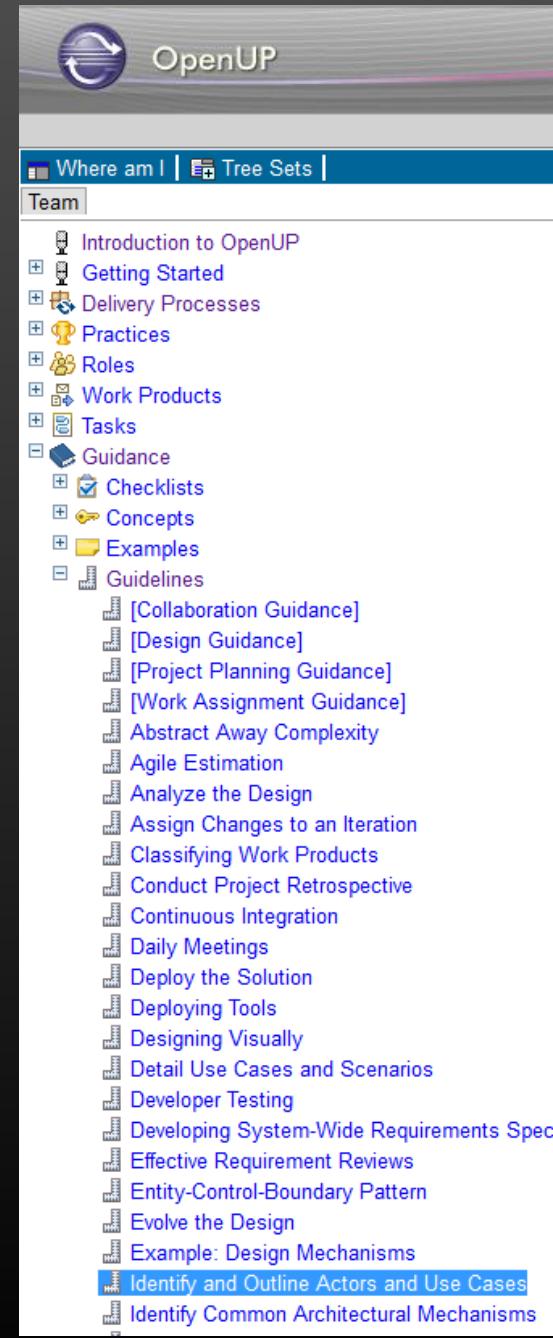
Identify the system boundary

Identify the actors who somehow interact with the system

For each actor, identify the objectives/motivations to use the system

Define CaU that satisfies the objectives of the actors

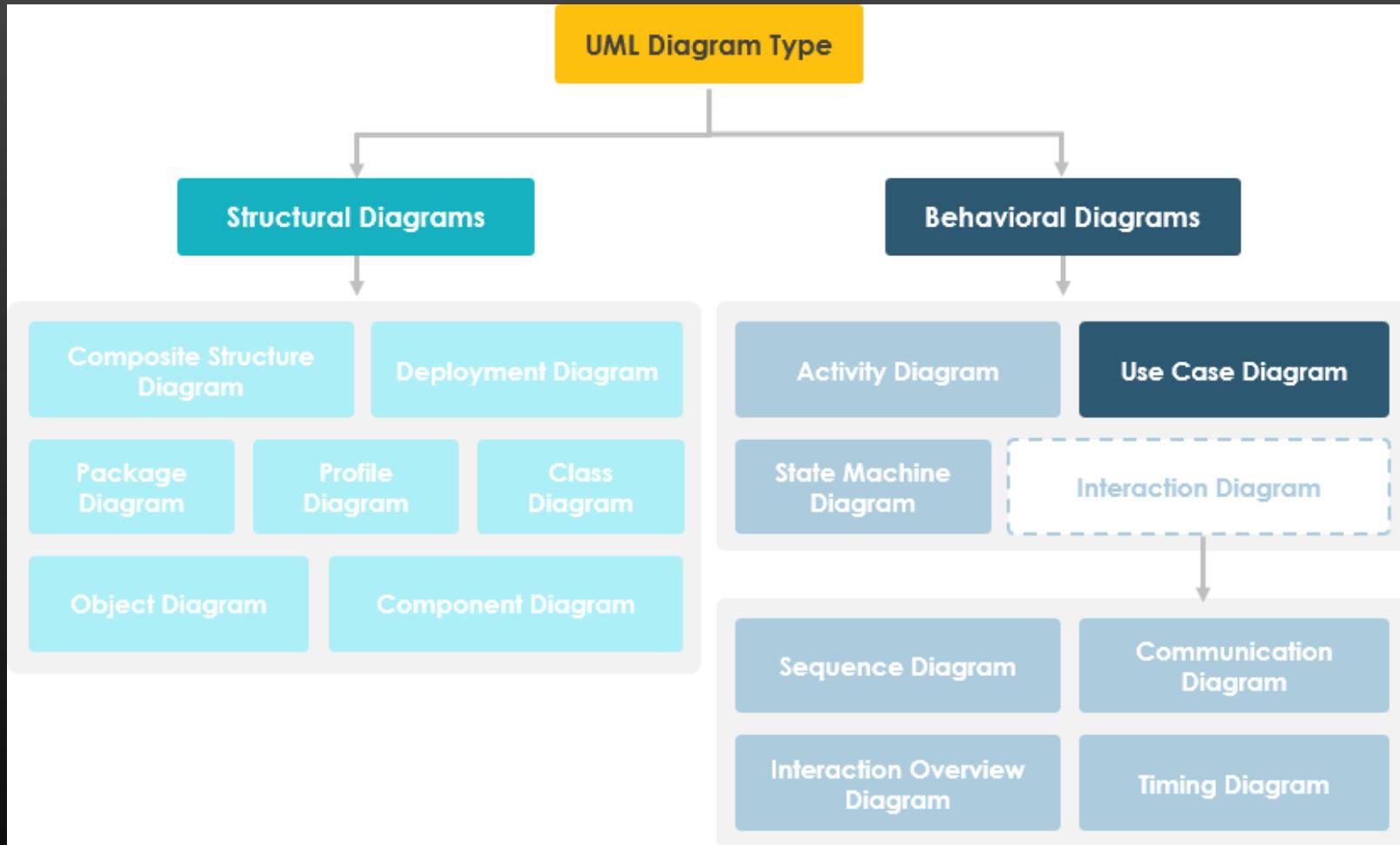
Give names that reflect the motivation of the actor



The screenshot shows the OpenUP interface with a navigation tree on the right. The tree is organized under a 'Team' category, which includes sections like Introduction to OpenUP, Getting Started, Delivery Processes, Practices, Roles, Work Products, Tasks, Guidance, Checklists, Concepts, Examples, and Guidelines. Under 'Guidelines', there are many sub-items such as [Collaboration Guidance], [Design Guidance], [Project Planning Guidance], [Work Assignment Guidance], Abstract Away Complexity, Agile Estimation, Analyze the Design, Assign Changes to an Iteration, Classifying Work Products, Conduct Project Retrospective, Continuous Integration, Daily Meetings, Deploy the Solution, Deploying Tools, Designing Visually, Detail Use Cases and Scenarios, Developer Testing, Developing System-Wide Requirements Spec, Effective Requirement Reviews, Entity-Control-Boundary Pattern, Evolve the Design, Example: Design Mechanisms, Identify and Outline Actors and Use Cases (which is highlighted in blue), and Identify Common Architectural Mechanisms.

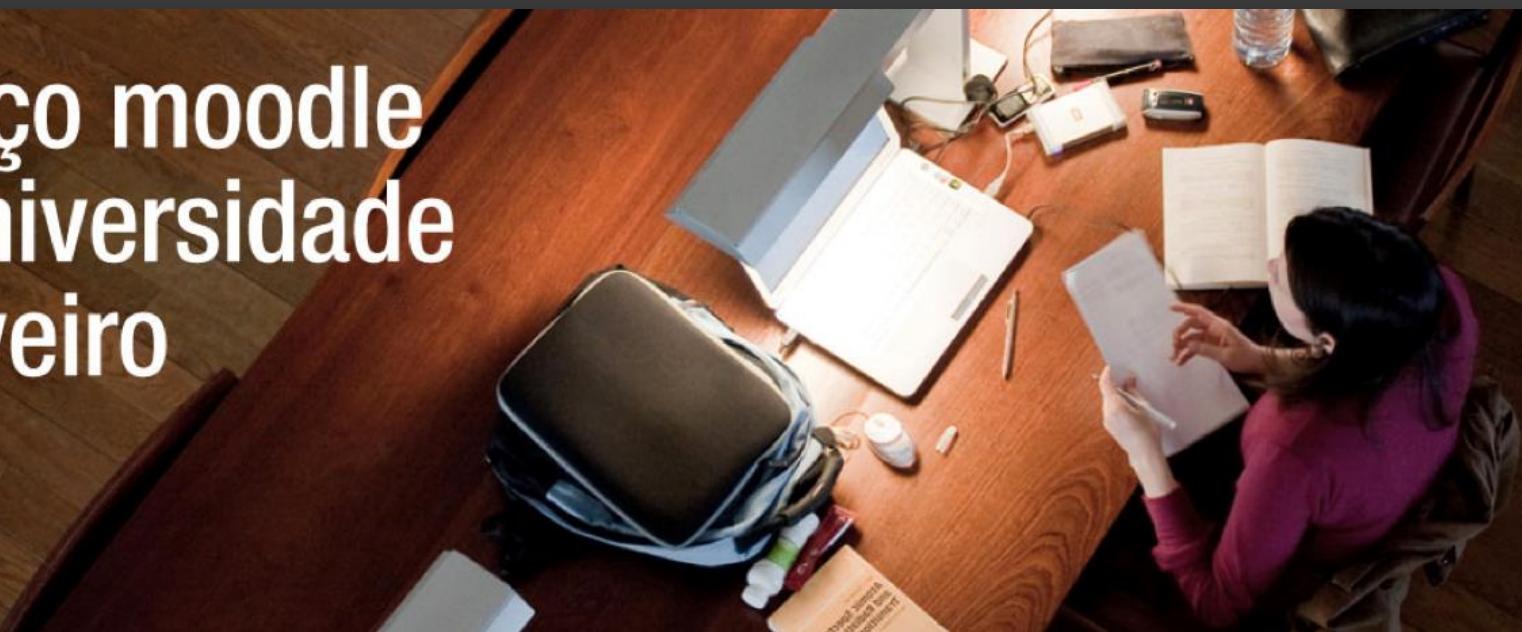
Guideline: Identify and Outline Actors and Use Cases

UML support

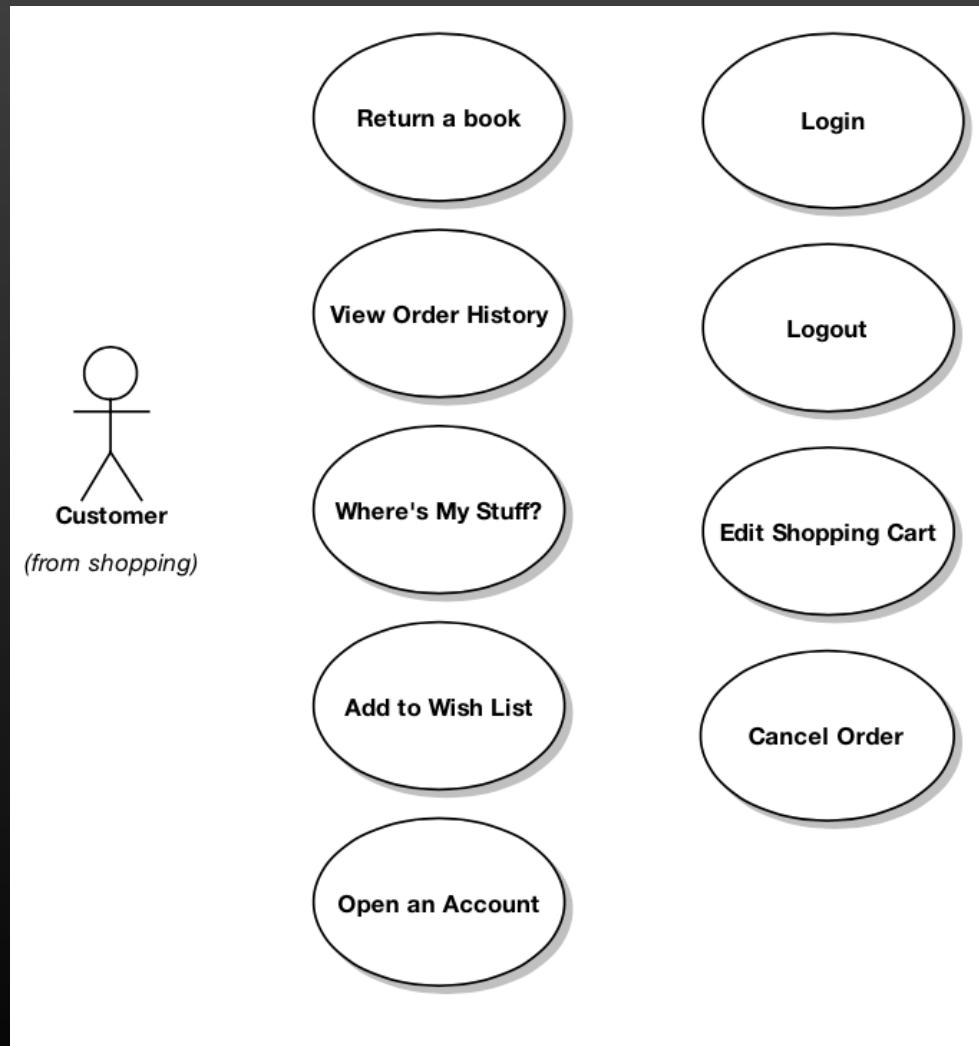


<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/>

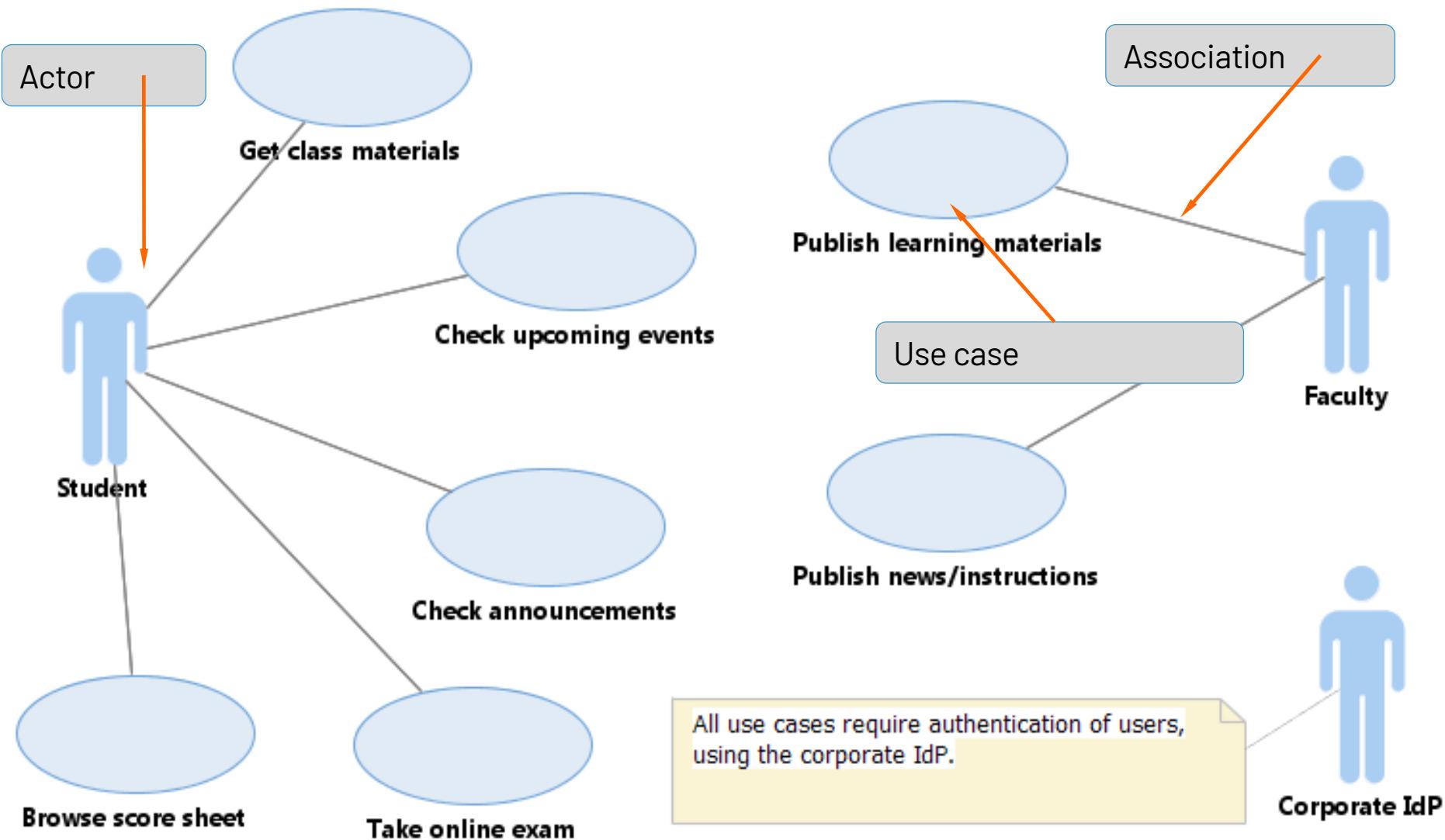
espaço moodle da universidade de aveiro



UML for use cases modeling



Elements of the UC diagram



Reusing behavior with **include**

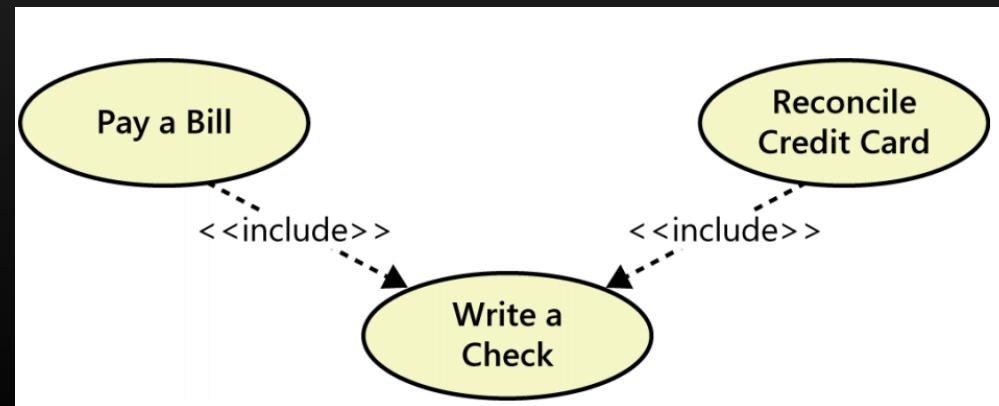
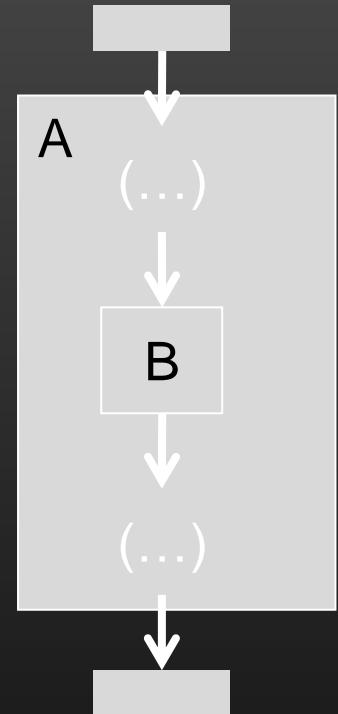
A-story includes the B-story

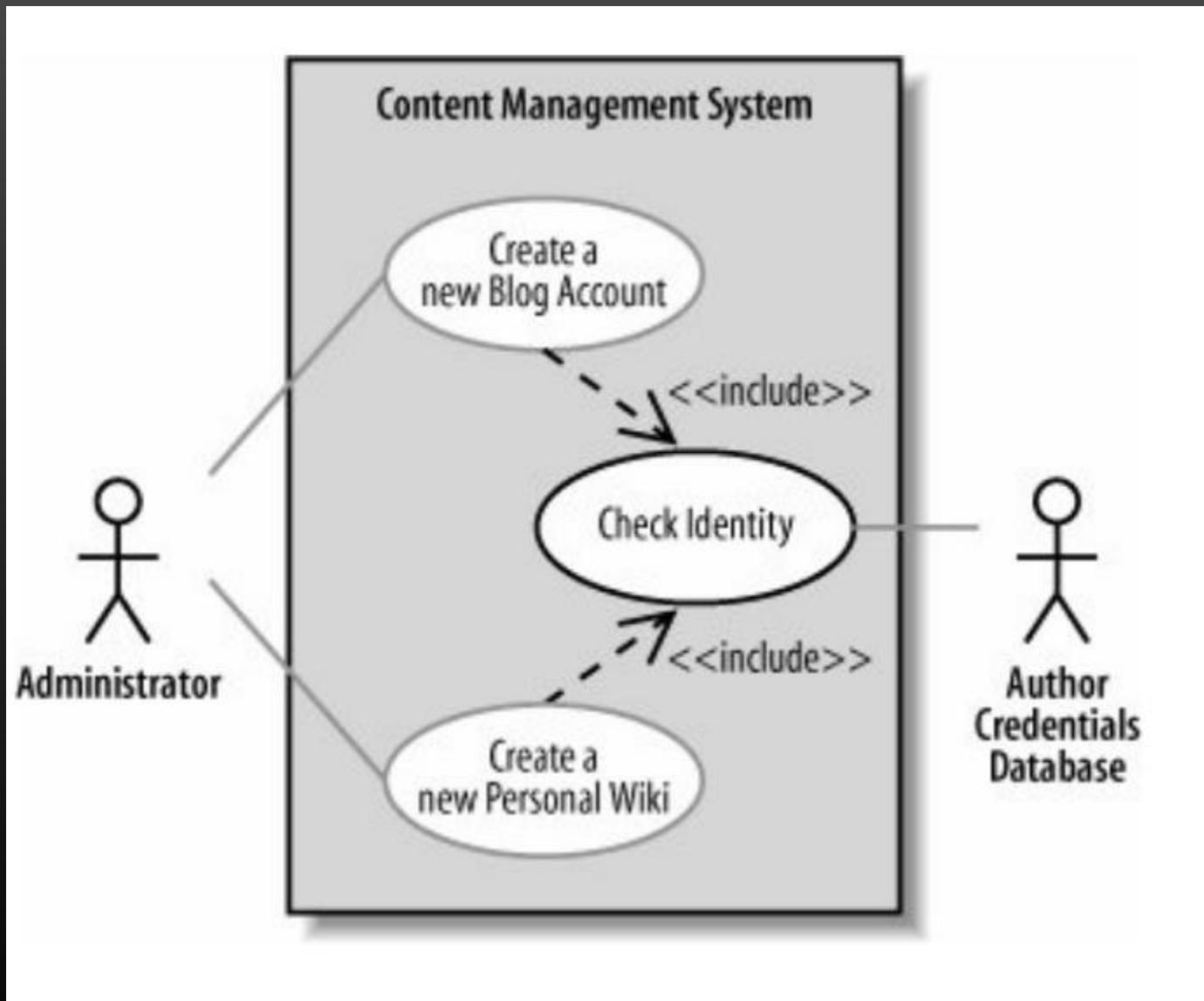
The behavior in A always includes the behavior modeled in B

Can help with “factoring out” common behavior

Include is the stereotype of the dependency relationship

- Not a verb here



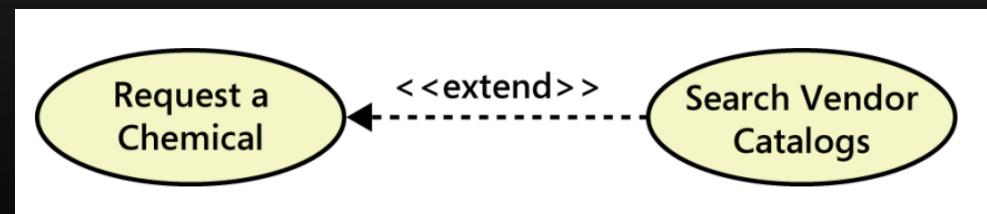
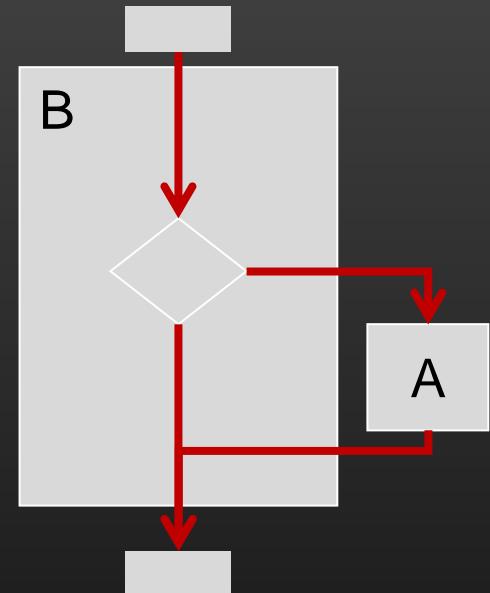


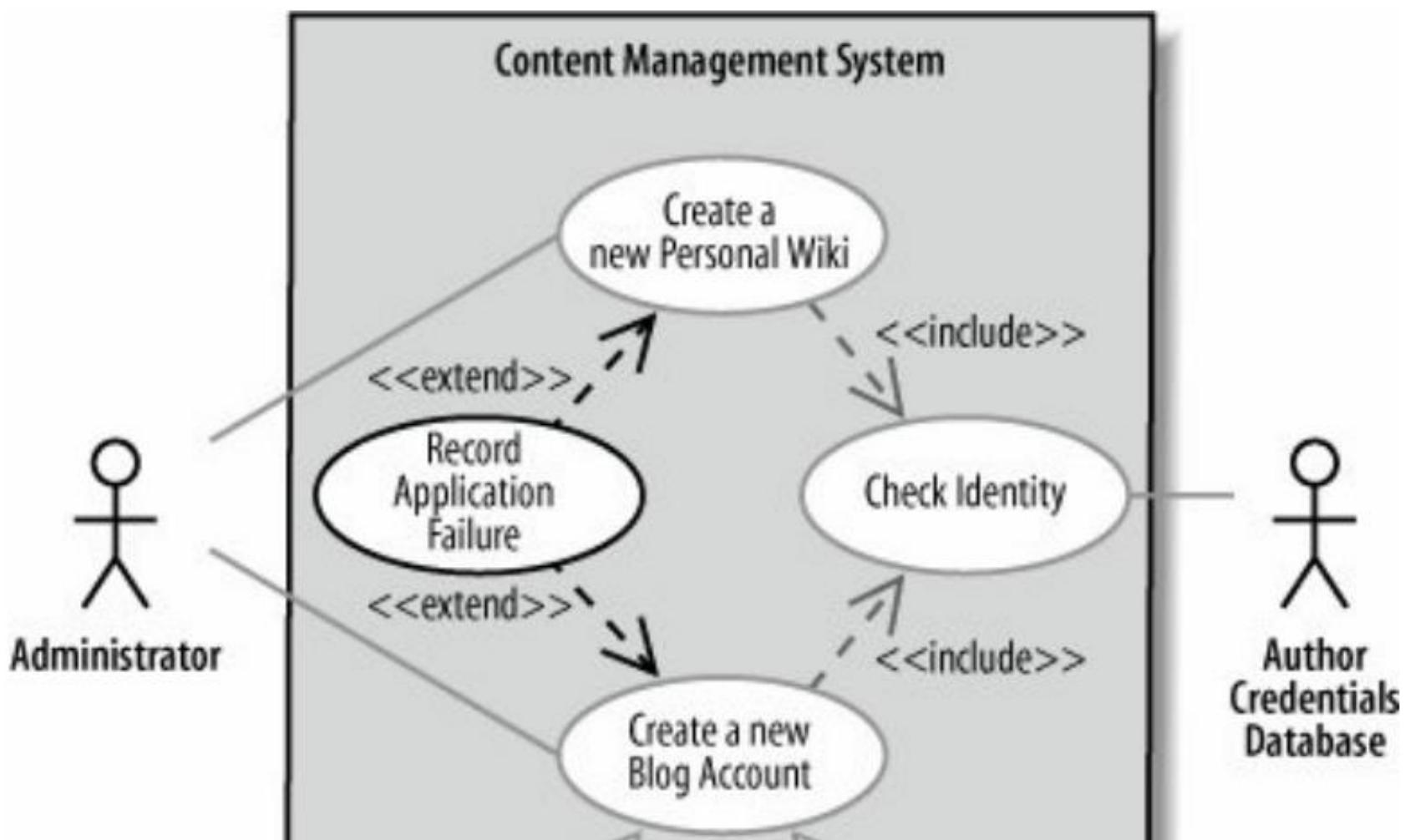
Optional behavior activation with **extend**

A-story may extend B-story

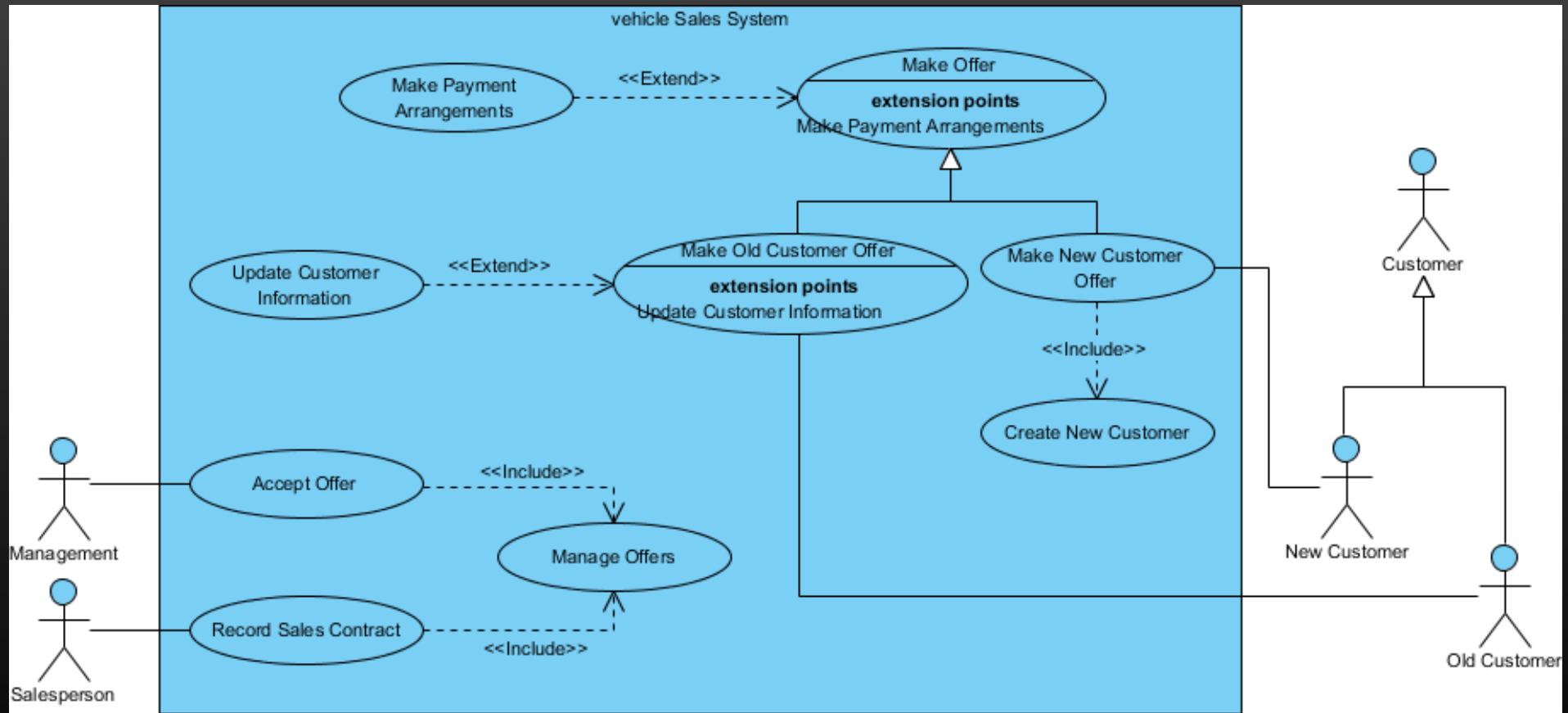
The behavior of B can incorporate the behavior of A, depending on the verification of an "extension condition" (*extension point*)

Unlike the include relationship,
extend models
optional/conditional behavior





Sample (full syntax)



The use cases may be organized in coherent functional groups → using packages

general

- + Add to Wish List
- + Cancel Order
- + Edit Shopping Cart
- + Login
- + Logout
- + Open an Account
- + Return a book
- + View Order History
- + Where's My Stuff?

admin

- + Customer Service
- + Seller
- + Shipping Clerk
- + Webmaster
- + Add Books to Catalog
- + Add Editorial Review
- + Add External Books to Catalog
- + Dispatch Order
- + Moderate Customer Reviews
- + Monitor Stock Levels
- + Order Books from Publisher
- + Process Refund
- + Remove Books from Catalog
- + Remove External Books from Catalog
- + Respond to Enquiry
- + Unlock Locked Account

shopping

- + Customer
- + Add Item to Shopping Cart
- + Checkout
- + Edit Shopping Cart
- + Enter Address
- + Pay by Card
- + Pay by Check
- + Pay by Purchase Order
- + Remove Item From Shopping Cart

searching

- + Advanced Search
- + Search by Author
- + Search by Category

Readings & references

Core readings	Suggested readings
<ul style="list-style-type: none">• [Dennis15] – Chap. 4• <u>How to write effective use cases</u>, VisualParadigm documentation• Jacobson, I., Spence, I., & Kerr, B. (2016). <u>Use-case 2.0</u>. Communications of the ACM, 59(5), 61–69.	

47006- ANÁLISE E MODELAÇÃO DE SISTEMAS

Requirements determination: finding what to build

Ilídio Oliveira

v2020/10/21, TP

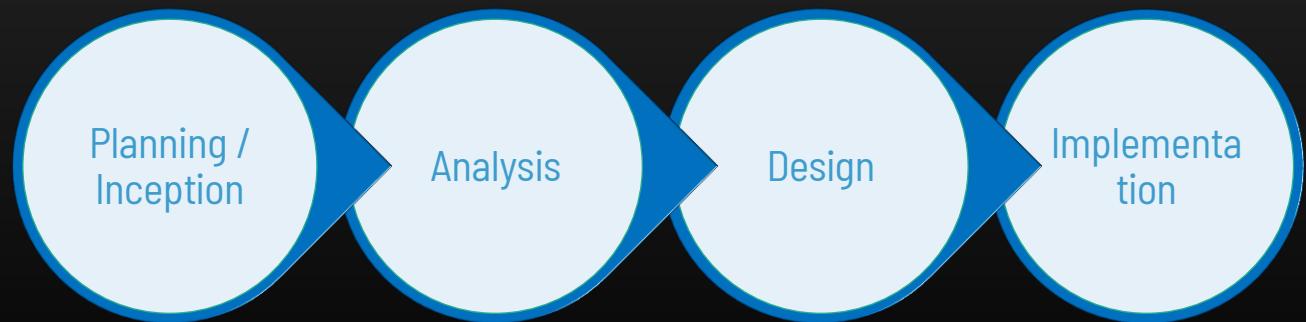
Learning objectives for this lecture

- Distinguish functional and non-functional requirements
- Enumerate requirements gathering techniques and argument when to use each one
- Describe requirements documentation techniques
- Understand the relation between requirements and use cases
- Identify the requirements related disciplines and activities in the OpenUP

Requirements determination

The single most critical step of the entire SDLC

- Changes can be made easily in this stage
- Most (>50%) system failures are due to problems with requirements



What is the impact of correcting “errors” in the software?

FIGURE 19.2

Relative cost of correcting errors and defects
Source: Adapted from [Boe01b].



Why conduct “requirements determination” activities?

Purpose

To convert high level business requests into detailed requirements that can be used as inputs for creating models

What is a requirement?

A statement of what the system must do or a characteristic it must have.

Will later evolve into a technical description of how the system will be implemented.

Project “mission”

Develop a system that conforms to the established requirements (capabilities & operating conditions)

Sample (functional) requirements

Ordering multiple meals and multiple food items

The COS shall permit the user to order multiple identical meals, up to the fewest available units of any menu item in the order.

Delivery or pickup

The Patron shall specify whether the order is to be picked up or delivered.

Examples from
Wieger's book. COS:
Cafeteria Ordering
System

Product specification includes quality attributes



The same functional capabilities, very different quality attribs



What are the types of requirements?

Types of requirements

Functional: relates to a process or data (*things the system can perform*)

Non-functional: relates to a system quality (*how well must the system do the operation*)

"sample" requirements

- The system should search patients by name, using at least two words that should occur in sequence in the name.
- The system will retrieve the doctor's daily appointments in <1 sec.

Function or non-functional: how to tell?

Functional requirements

Capture the intended behavior

- Services and functions that the system must perform

Captured in use cases descriptions (usage scenarios)

Can be supplemented with behavior diagrams: activities, sequence,...

Non-functional requirements

Global restrictions/operating constraints in the software

- E.g.: user friendliness, portability, robustness,...

a.k.a. quality attributes

- they refer to the expected degree of a certain desired quality

Usually, they are not limited to a function/module, rather cross-cutting

Sample requirements definition

Nonfunctional Requirements

1. Operational Requirements

- 1.1. The system will operate in Windows environment.
- 1.2. The system should be able to connect to printers wirelessly.
- 1.3. The system should automatically back up at the end of each day.

2. Performance Requirements

- 2.1. The system will store a new appointment in 2 seconds or less.
- 2.2. The system will retrieve the daily appointment schedule in 2 seconds or less.

3. Security Requirements

- 3.1. Only doctors can set their availability.
- 3.2. Only a manager can produce a schedule.

4. Cultural and Political Requirements

- 4.1. No special cultural and political requirements are anticipated.

Functional Requirements

1. Manage Appointments

- 1.1. Patient makes new appointment.
- 1.2. Patient changes appointment.
- 1.3. Patient cancels appointment.

2. Produce Schedule

- 2.1. Office Manager checks daily schedule.
- 2.2. Office Manager prints daily schedule.

3. Record Doctor Availability

- 3.1. Doctor updates schedule

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

Wieger's software quality attributes (*-ies)

TABLE 14-1 Some software quality attributes

External quality	Brief description
Availability	The extent to which the system's services are available when and where they are needed
Installability	How easy it is to correctly install, uninstall, and reinstall the application
Integrity	The extent to which the system protects against data inaccuracy and loss
Interoperability	How easily the system can interconnect and exchange data with other systems or components
Performance	How quickly and predictably the system responds to user inputs or other events
Reliability	How long the system runs before experiencing a failure
Robustness	How well the system responds to unexpected operating conditions
Safety	How well the system protects against injury or damage
Security	How well the system protects against unauthorized access to the application and its data
Usability	How easy it is for people to learn, remember, and use the system
Internal quality	Brief description
Efficiency	How efficiently the system uses computer resources
Modifiability	How easy it is to maintain, change, enhance, and restructure the system
Portability	How easily the system can be made to work in other operating environments
Reusability	To what extent components can be used in other systems
Scalability	How easily the system can grow to handle more users, transactions, servers, or other extensions
Verifiability	How readily developers and testers can confirm that the software was implemented correctly

Credit: Wiegers '13

FURPS categories of quality attributes

Functionality

- assessed by evaluating the feature set and capabilities of the program, security (functions)

Usability

- assessed by considering human factors, overall aesthetics, consistency, and documentation.

Reliability

- evaluated by measuring the frequency and severity of failure, the accuracy of output results, the mean-time-to-failure (MTTF), the ability to recover from failure

Performance

- measured using processing speed, response time, resource consumption, throughput, and efficiency.

Supportability

- combines extensibility, adaptability, and serviceability (i.e., **maintainability**) and in addition, testability, compatibility, configurability, the ease with which a system can be installed, and the ease with which problems can be localized

What can be involved in “performance” (as a quality attribute)?

TABLE 14-2 Some aspects of performance

Performance dimension	Example
Response time	Number of seconds to display a webpage
Throughput	Credit card transactions processed per second
Data capacity	Maximum number of records stored in a database
Dynamic capacity	Maximum number of concurrent users of a social media website
Predictability in real-time systems	Hard timing requirements for an airplane's flight-control system
Latency	Time delays in music recording and production software
Behavior in degraded modes or overloaded conditions	A natural disaster leads to a massive number of emergency telephone system calls

Requirements

STRQ1: Want to be able to transfer funds from other accounts (not necessarily held with this firm) to a trading account.

STRQ2: State and federal regulations require monthly reports of account activity. Refer to specification RUFS-1234 for details of the information required.

STRQ3: The system should allow the use of any browser.

STRQ4: Customers want to manage their retirement funds.

STRQ5: Must be able to upgrade the system without taking it offline.

STRQ6: The system should allow traders to trade in multiple markets across the world.

STRQ7: Must be able to provide convenient answers to customer's most common questions.

STRQ8: The system must provide a secure environment that prohibits fraudulent access.

STRQ9: Need a way to train customers in the use of the system quickly and conveniently.

STRQ10: The system must operate on hardware that falls under the company's current maintenance contracts.

STRQ11: Need to be able to maintain the system with our current IT hardware and skills. Refer to enterprise architecture document EA-1234 for details.

STRQ12: Need account activity statements for tax reporting.

STRQ13: The system must provide all the basic capabilities of a normal stock broking firm.

STRQ14: Need to be able to perform research on any given stock.

STRQ15: The system must allow traders to obtain up-to-date news and alerts on nominated stock.

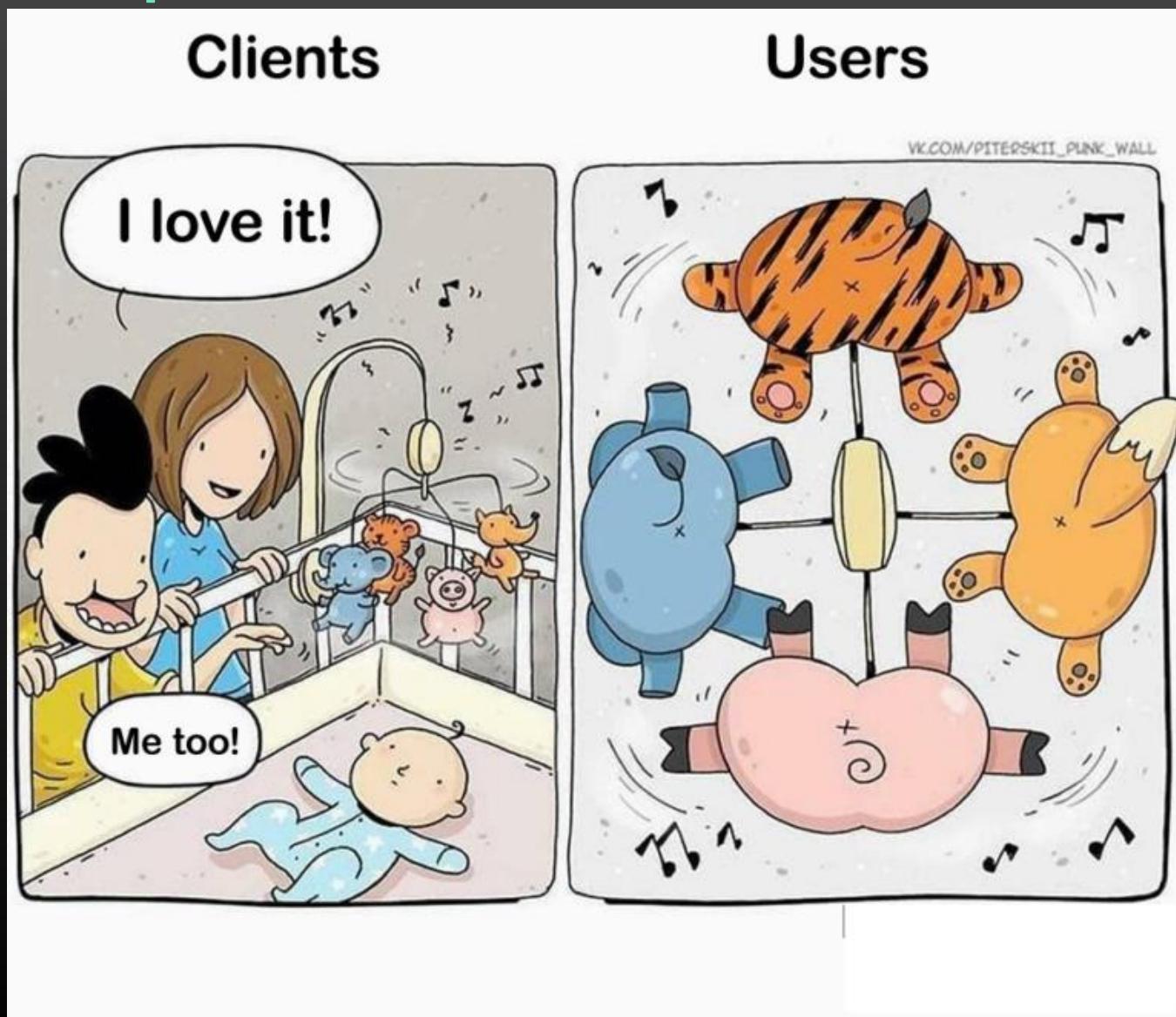
STRQ16: The system must provide current and historical information on Trading Accounts. Such as number of shares held, current price, total Trading Account value

STRQ17: The system shall provide the following types of trades: Market Trades (buy and sell), Limit Trades (buy and sell), and transfers between mutual funds.

Software designers tend to focus on the problem to be solved. Just don't forget that the FURPS attributes are always part of the problem. They must be considered.

Gathering requirements

Customer/promoter vs users



Requirements Gathering Techniques

Process used to:

Uncover all requirements (those uncovered late in the process are more difficult to incorporate)

Build support and trust among users/stakeholders

Which technique(s) to use?

1. Interviews
2. Joint Application Development (JAD)
3. Questionnaires
4. Document analysis
5. Observation

OpenUP requirements practices

OpenUP practices

[OpenUP](#) > Practices >
Technical Practices >
Shared Vision >
Requirements Gathering
Techniques

Details available on the
OpenUp documentation



OpenUP

Requirements Gathering Techniques

After you have identified these sources, there are a number of techniques that follow will describe the various techniques, followed by a brief discussion of

To get the requirements down on paper, you can to do one or more of the following:

- Conduct a brainstorming session
- Interview users
- Send questionnaires
- Work in the target environment
- Study analogous systems
- Examine suggestions and problem reports
- Talk to support teams
- Study improvements made by users
- Look at unintended uses
- Conduct workshops
- Demonstrate prototypes to stakeholders

The best idea is to get the requirements down quickly and then to encourage them in those corrections, and repeat the cycle. Do it now, keep it small, and correct what you can devise, but expect to keep on correcting it throughout the process. Sure correct it immediately.

Requirements-Gathering Techniques Compared

A combination of techniques may be used

Document analysis & observation require little training; JAD sessions can be very challenging

	Interviews	Joint Application Design	Questionnaires	Document Analysis	Observation
Type of information	As-is, improvements, to-be	As-is, improvements, to-be	As-is, improvements	As-is	As-is
Depth of information	High	High	Medium	Low	Low
Breadth of information	Low	Medium	High	High	Low
Integration of information	Low	High	Low	Low	Low
User involvement	Medium	High	Low	Low	Low
Cost	Medium	Low–Medium	Low	Low	Low to Medium

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

Alternative Techniques

Domain analysis

Study the characteristics of the domain

Learn from others

Increase the readiness for scaling to more customers

Concept Maps

Represent meaningful relationships between concepts

Focus individuals on a small number of key ideas

User Stories, Story Cards & Task Lists

Associated with agile development methods

Very low tech, high touch, easily updatable, and very portable

Captured using story cards (index cards)

Capture both functional and nonfunctional requirements.

"house wish-list..."



Requirements elicitation

System to build

Requirements documented in analysis

Inconsistent

Not feasible
(technology)

superfluous

Missing requirements

Initial scope

What are the requirements elicitation activities?

The heart of requirements development is **elicitation**

- the process of identifying the needs and constraints of the various stakeholders for a software system.

Elicitation is not the same as “gathering requirements.”

- Nor is it a simple matter of transcribing exactly what users say.

Elicitation is a collaborative and analytical process that includes activities to collect, discover, extract, and define requirements. Elicitation is used to discover business, user, functional, and nonfunctional requirements.

Requirements elicitation is perhaps the most challenging, critical, error-prone, and communication-intensive aspect of software development.

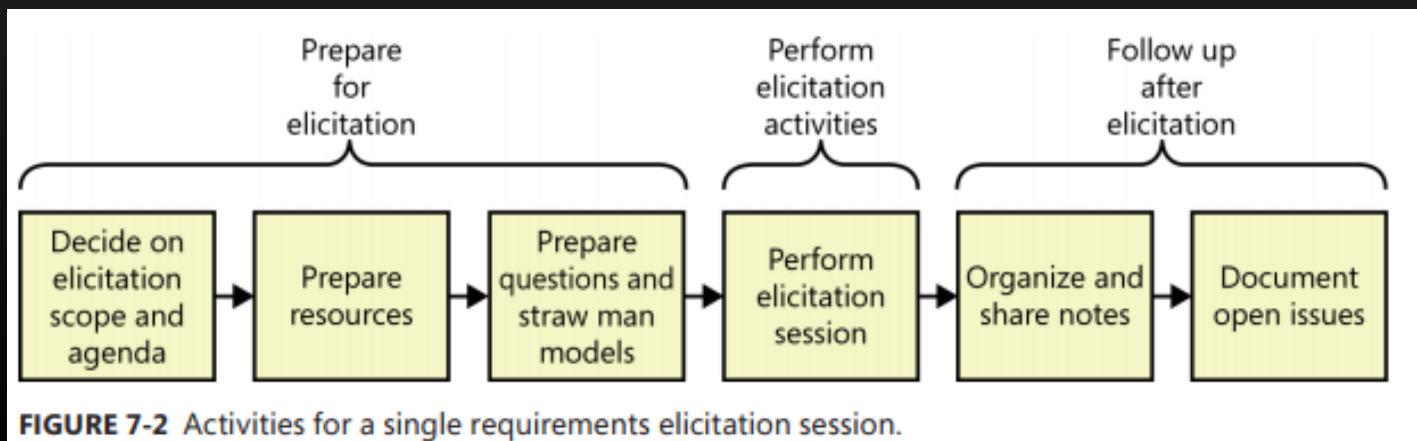


FIGURE 7-2 Activities for a single requirements elicitation session.

Figure 7-3 suggests the elicitation techniques that are most likely to be useful for various types of projects. Select the row or rows that represent characteristics of your project and read to the right to see which elicitation techniques are most likely to be helpful (marked with an X). For instance, if you're developing a new application, you're likely to get the best results with a combination of stakeholder interviews, workshops, and system interface analysis. Most projects can make use of interviews and workshops. Focus groups are more appropriate than workshops for mass-market software because you have a large external user base but limited access to representatives. These suggestions for elicitation techniques are just that—suggestions. For instance, you might conclude that you do want to apply user interface analysis on mass-market software projects.

	Interviews	Workshops	Focus groups	Observations	Questionnaires	System interface analysis	User interface analysis	Document analysis
Mass-market software	X		X		X			
Internal corporate software	X	X	X	X		X		X
Replacing existing system	X	X		X		X	X	X
Enhancing existing system	X	X				X	X	X
New application	X	X				X		
Packaged software implementation	X	X		X		X		X
Embedded systems	X	X				X		X
Geographically distributed stakeholders	X	X			X			

FIGURE 7-3 Suggested elicitation techniques by project characteristic.

Requirements documentation

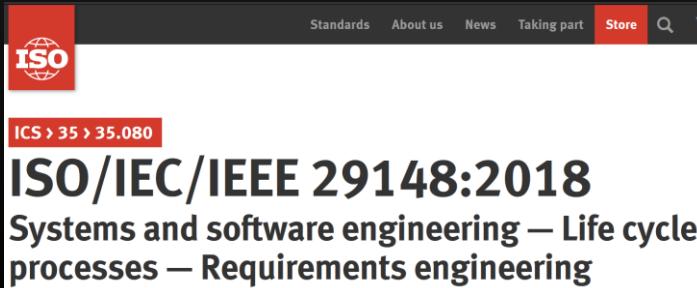
Requisitos são muitas vezes elencados em listas “o sistema deve...”

#	Requisito
RF.1	O sistema deve permitir a um profissional criar um novo pedido de adesão , em auto-serviço, na web.
RF.2	O sistema deve enviar credenciais temporárias para os pedidos de adesão e enviá-las, por email, aos solicitantes.
RF.3	O sistema deve permitir a pesquisa de cheques-dentista (emitidos) por número de utente do SNS.
RNF.1	As pesquisas de cheques-dentista têm de retornar resultados em <5 segundos ou um evento de tempo expirado.
...	

What are well-formed requirements? (ISO-IEEE 29148)

A statement that:

- can be verified,
- has to be met or possessed by a system to solve a stakeholder problem or to achieve a stakeholder objective,
- is qualified by measurable conditions and bounded by constraints, and
- defines the performance of the system when used by a specific stakeholder or the corresponding capability of the system, but not a capability of the user, operator, or other stakeholder.



The image shows a screenshot of the ISO website. At the top, there is a navigation bar with links for 'Standards', 'About us', 'News', 'Taking part', 'Store' (which is highlighted in red), and a search icon. Below the navigation bar, there is a red banner with the text 'ICS > 35 > 35.080'. The main content area features the title 'ISO/IEC/IEEE 29148:2018' in large bold letters, followed by the subtitle 'Systems and software engineering — Life cycle processes — Requirements engineering'.

Elements of style

A requirement is a statement which expresses a need and its associated constraints and conditions.

If expressed in the form of a natural language, the statement should comprise a subject, a verb and a complement. A requirement shall state the subject of the requirement (e.g., the system, the software, etc.) and what shall be done (e.g., operate at a power level, provide a field for).

E.g: The Invoice System [Subject], shall display pending customer invoices [Action] in ascending order [Value] in which invoices are to be paid.

S. M. A. R. T. (Specific, Measurable, Attainable, Relevant and time-sensitive)

Step 5: Specify well-structured quality requirements

Simplistic quality requirements such as "The system shall be user-friendly" or "The system shall be available 24x7" aren't useful. The former is far too subjective and vague; the latter is rarely realistic or necessary. Neither is measurable. Such requirements provide little guidance to developers. So the final step is to craft specific and verifiable requirements from the information that was elicited regarding each quality attribute. When writing quality requirements, keep in mind the useful SMART mnemonic—make them *Specific, Measurable, Attainable, Relevant, and Time-sensitive*.

"The system shall be 100% reliable and 100% available".

"The system shall have a minimum response to a query of 1 second irrespective of system load".

AVL-1. The system shall be at least 95 percent available on weekdays between 6:00 A.M. and midnight Eastern Time, and at least 99 percent available on weekdays between 3:00 P.M. and 5:00 P.M. Eastern Time.

IOP-1. The Chemical Tracking System shall be able to import any valid chemical structure from the ChemDraw (version 13.0 or earlier) and MarvinSketch (version 5.0 or earlier) tools.

PER-1. Authorization of an ATM withdrawal request shall take no more than 2.0 seconds.

PER-2. The anti-lock braking system speed sensors shall report wheel speeds every 2 milliseconds with a variation not to exceed 0.1 millisecond.

PER-3. Webpages shall fully download in an average of 3 seconds or less over a 30 megabits/second Internet connection.

PER-4. At least 98 percent of the time, the trading system shall update the transaction status display within 1 second after the completion of each trade.

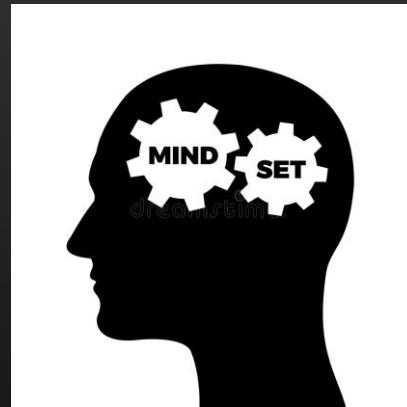
Requirements and use cases

Which main elicitation approaches exist?

What is the goal the **user**
wants to achieve?

vs.

What capability should the
product/system possess?

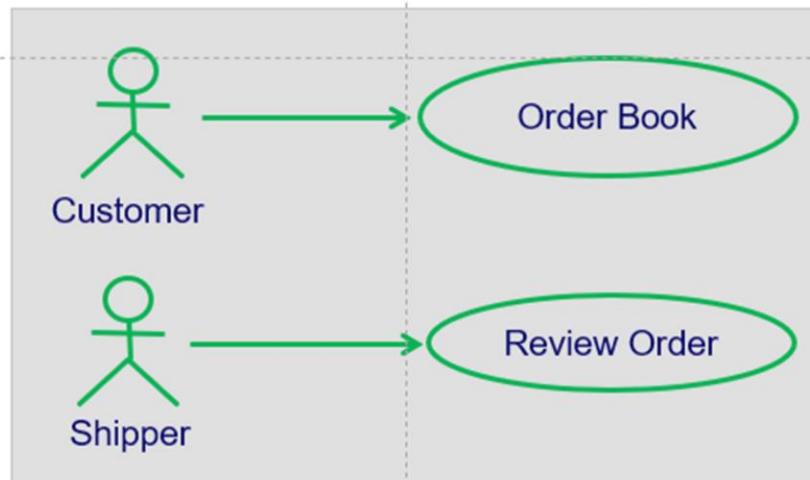


OpenUP recommended practices

Use Case Driven Development



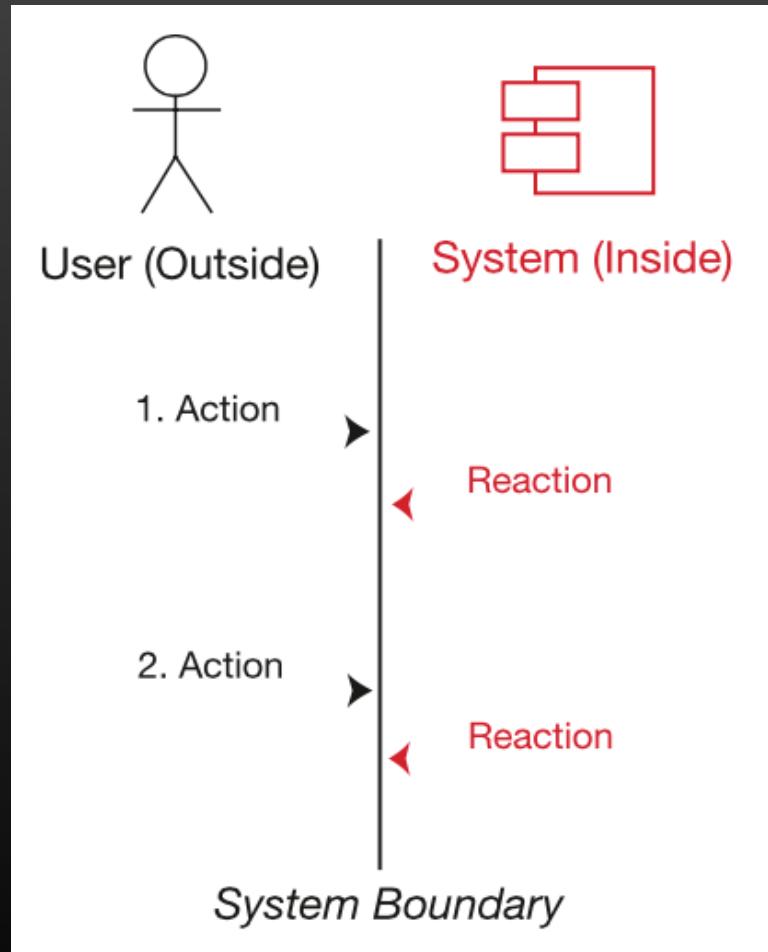
- This practice describes how to capture requirements with a combination of use cases and system-wide requirements, and then drive development and testing from those use cases.



The use case captures a dialog between the actor(s) and the system

CaU: Pay at checkout

1. Customer arrives at POS checkout with goods to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price is calculated from a set of price rules.
Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any)



The use case details describe an interaction

HOW TO WRITE A USE CASE: THE THREE MAGIC QUESTIONS

Well, OK, this whole chapter describes how to write a use case. But when writing use cases, you need to keep asking the following three fundamental questions:¹

1. What happens?

(This gets your “sunny-day scenario” started.)

2. And then what happens?

(Keep asking this question until your “sunny-day scenario” is complete.)

3. What else might happen?

(Keep asking this one until you’ve identified all the “rainy-day scenarios” you can think of, and described the related behavior.)

"Request a Chemical" use case specification

ID and Name:	UC-4 Request a Chemical	
Created By:	Lori	Date Created: 8/22/13
Primary Actor:	Requester	Secondary Actors: Buyer, Chemical Stockroom, Training Database
Description:	The Requester specifies the desired chemical to request by entering its name or chemical ID number or by importing its structure from a chemical drawing tool. The system either offers the Requester a container of the chemical from the chemical stockroom or lets the Requester order one from a vendor.	
Trigger:	Requester indicates that he wants to request a chemical.	
Preconditions:	PRE-1. User's identity has been authenticated. PRE-2. User is authorized to request chemicals. PRE-3. Chemical inventory database is online.	
Postconditions:	POST-1. Request is stored in the CTS. POST-2. Request was sent to the Chemical Stockroom or to a Buyer.	
Normal Flow:	4.0 Request a Chemical from the Chemical Stockroom 1. Requester specifies the desired chemical. 2. System lists containers of the desired chemical that are in the chemical stockroom, if any. 3. System gives Requester the option to View Container History for any container. 4. Requester selects a specific container or asks to place a vendor order (see 4.1). 5. Requester enters other information to complete the request. 6. System stores the request and notifies the Chemical Stockroom.	
Alternative Flows:	4.1 Request a Chemical from a Vendor 1. Requester searches vendor catalogs for the chemical (see 4.1.E1). 2. System displays a list of vendors for the chemical with available container sizes, grades, and prices. 3. Requester selects a vendor, container size, grade, and number of containers. 4. Requester enters other information to complete the request. 5. System stores the request and notifies the Buyer.	
Exceptions:	4.1.E1 Chemical Is Not Commercially Available 1. System displays message: No vendors for that chemical. 2. System asks Requester if he wants to request another chemical (3a) or to exit (4a).	

Use case:	Brief description:
Create new assignment	<p>The Teaching Staff creates a new Activity of type Assignment, directly inserting it in the page layout. The assignment must define a title and a time period for submissions and can be configured to work with individual or group submissions. The assignment is listed in the student view and on the specified date (or immediately, if none is given) accepts submissions from registered students.</p>
Use case: <u>Add new assignment</u> Brief description: <p>The Faculty creates assignments for students, directly inserting it in the course page. The assignment defines a time period for submissions and can be configured to work with individual or group submissions. The assignment is listed in the student view and on the specified date (or immediately, if none is given) accepts submissions from students.</p> Basic flow: <ol style="list-style-type: none"> 1. Log-in using corporate IdP. 2. Select desired course. 3. Turn editing mode on. 4. Add Assignment activity in the page layout. 5. Configure Assignment activity. 6. Commit changes. Alternative flows: <p>Step 1: IdP unavailable.</p> <p>Step 4/5: Instead of a new, empty assignment, the user may reuse an existing one.</p> Open issues: <p>Step 3/4. The course is closed. Are changes allowed to past courses?</p> <p>Step 5. The browser does not accept the rich text editor. Default to plain text?</p>	

Putting it together

The screenshot shows a web browser displaying the OpenUP website. The header includes the OpenUP logo, navigation links for Glossary, Feedback, and About, and a Print button. The left sidebar has a 'Where am I' link, a 'Tree Sets' link, and a 'Team' section with a tree diagram. The main content area shows the breadcrumb path: Practices > Technical Practices > Use Case Driven Development > Tasks > Identify and Outline Requirements. The title of the current page is 'Task: Identify and Outline Requirements'. A yellow arrow icon points to the text: 'This task describes how to identify and outline the requirements for the system so that the scope of work can be determined.' Below this, under 'Disciplines: Requirements', are 'Expand All Sections' and 'Collapse All Sections' buttons. A 'Purpose' section contains the text: 'The purpose of this task is to identify and capture functional and non-functional requirements for the system. These requirements form the basis of communication and agreement between the stakeholders and the development team on what the system must do to satisfy stakeholder needs. The goal is to understand the requirements at a high-level so that the initial scope of work can be determined. Further analysis will be performed to detail these requirements prior to implementation.' At the bottom right is a 'Back to top' link.

Where am I | Tree Sets | Team

Practices > Technical Practices > Use Case Driven Development > Tasks > Identify and Outline Requirements

Task: Identify and Outline Requirements

This task describes how to identify and outline the requirements for the system so that the scope of work can be determined.

Disciplines: Requirements

[Expand All Sections](#) [Collapse All Sections](#)

Purpose

The purpose of this task is to identify and capture functional and non-functional requirements for the system. These requirements form the basis of communication and agreement between the stakeholders and the development team on what the system must do to satisfy stakeholder needs. The goal is to understand the requirements at a high-level so that the initial scope of work can be determined. Further analysis will be performed to detail these requirements prior to implementation.

[Back to top](#)



And second, those of us in the software domain tend to be enamored with technical and process solutions to our challenges. We sometimes fail to appreciate that requirements elicitation—and much of software and systems project work in general—is primarily a human interaction challenge. No magical new techniques have come along to automate that, although various tools are available to help geographically separated people collaborate effectively.

In: Wiegers, "Software Requirements"

Readings & references

Core readings	Suggested readings
<ul style="list-style-type: none">• [Dennis15] - Chap. 3 – Requirements Determination	<ul style="list-style-type: none">• [Pressman15] - Chap. 8 – Understanding Requirements• [Wiegers13] - Chap. 1-3

47006- ANÁLISE E MODELAÇÃO DE SISTEMAS

OpenUP and the AMS project assignment

Ilídio Oliveira

v2020-10-23 | TP-06

Learning objectives for this lecture

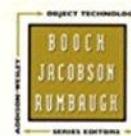
- Describe the structure of the Unified Process (phases, milestone objectives, iterations)
- Identify the key activities required in the project assignment
- Map technical disciplines to the OpenUP phases

THE UNIFIED SOFTWARE DEVELOPMENT PROCESS

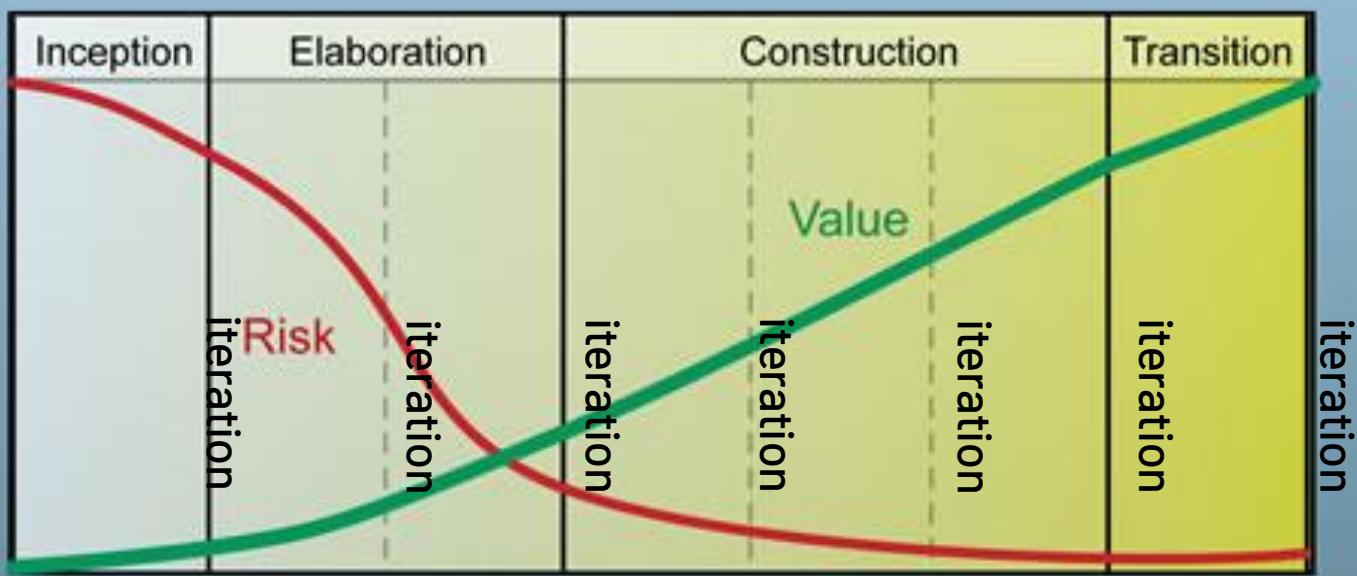
IVAR JACOBSON
GRADY BOOCHE
JAMES RUMBAUGH



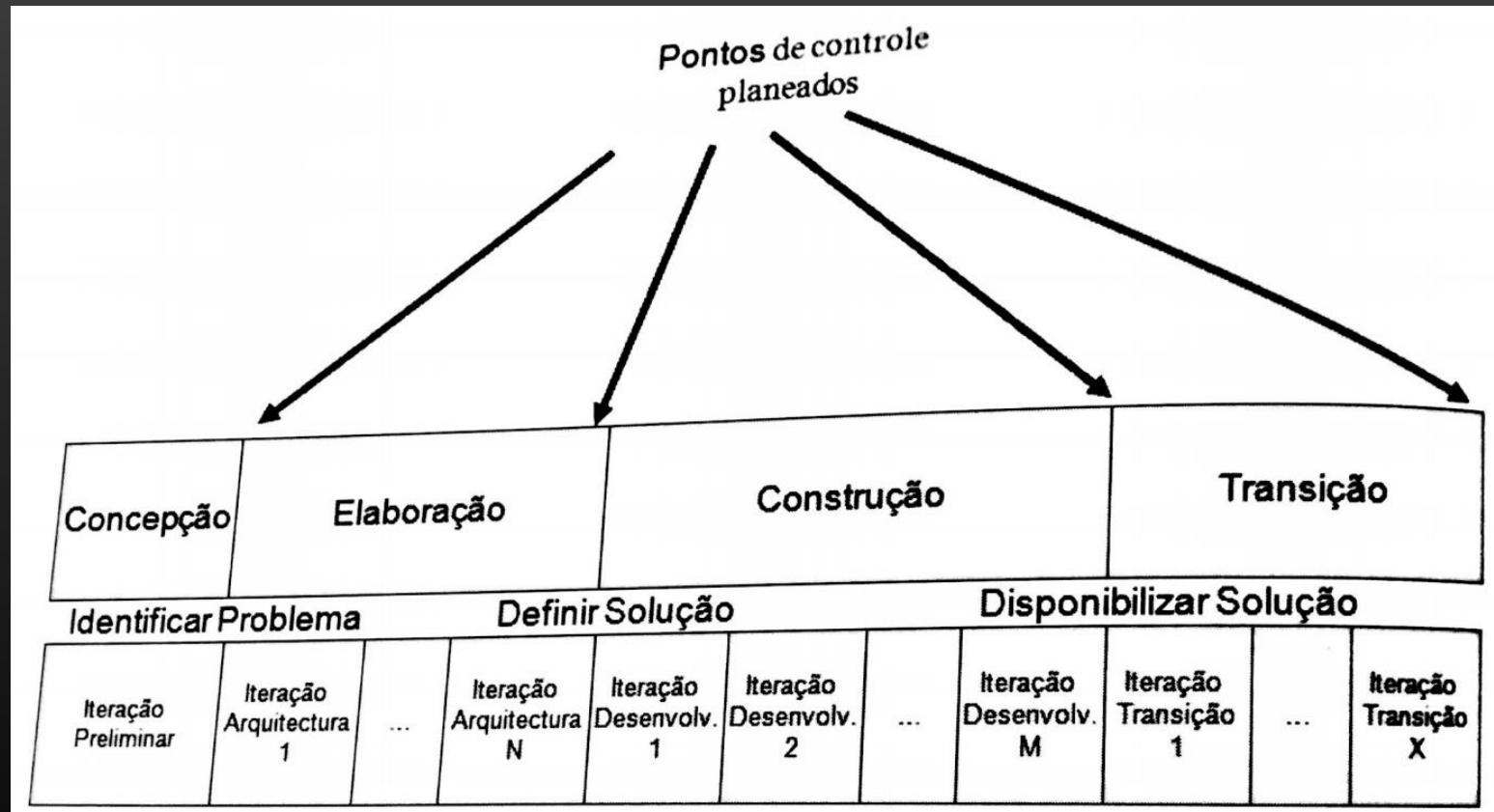
The complete guide
to the Unified
Process from the
original designers



Project Lifecycle



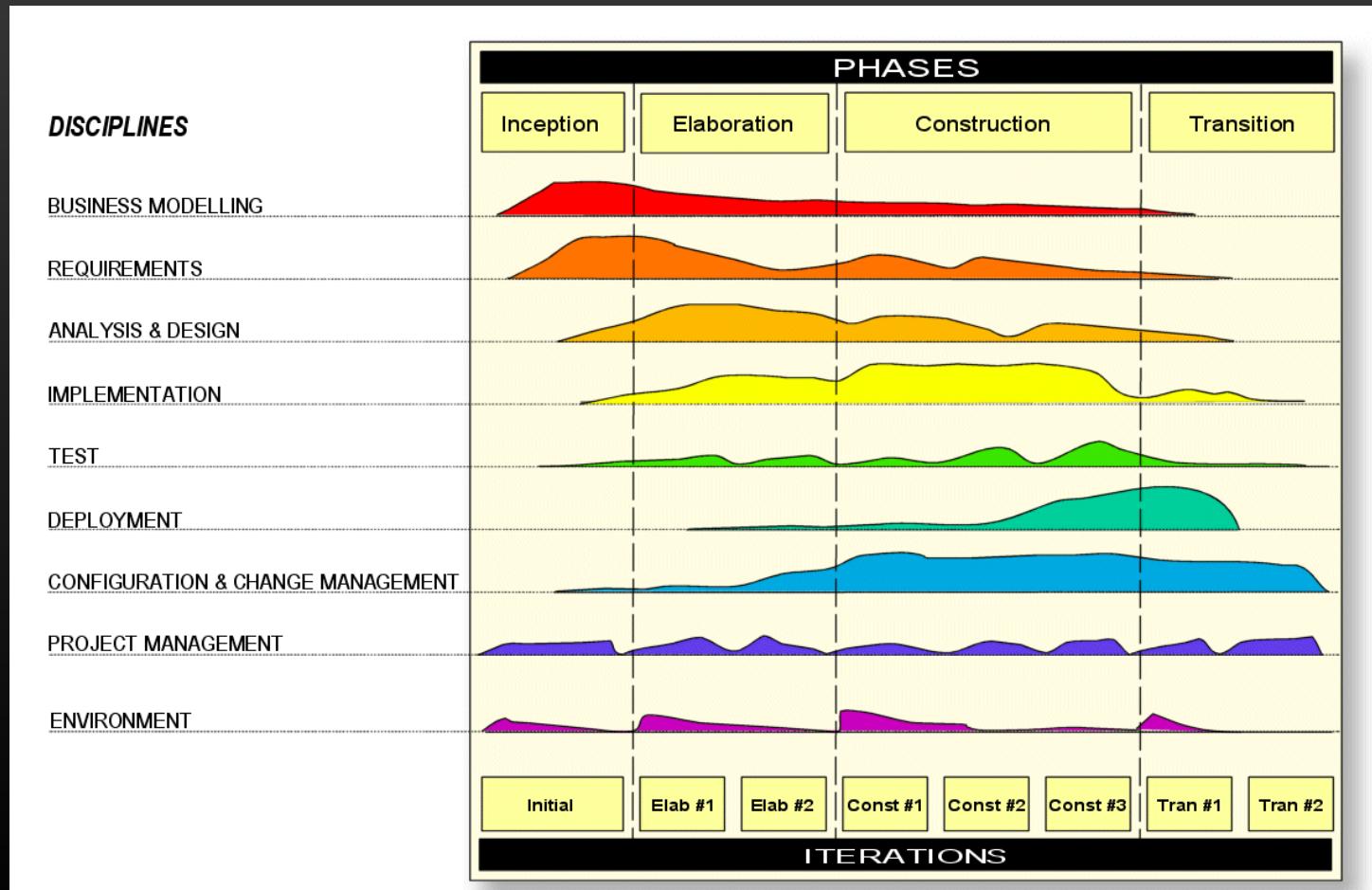
PT: Fases, iterações e pontos de controlo



OpenUP/Unified Process activities

The UP offers an approach to the SDLC visualized as a **matrix**, crossing different **technical disciplines** with evolving **iterations** in the project. (Note: UP phases ≠ SDLC phases)

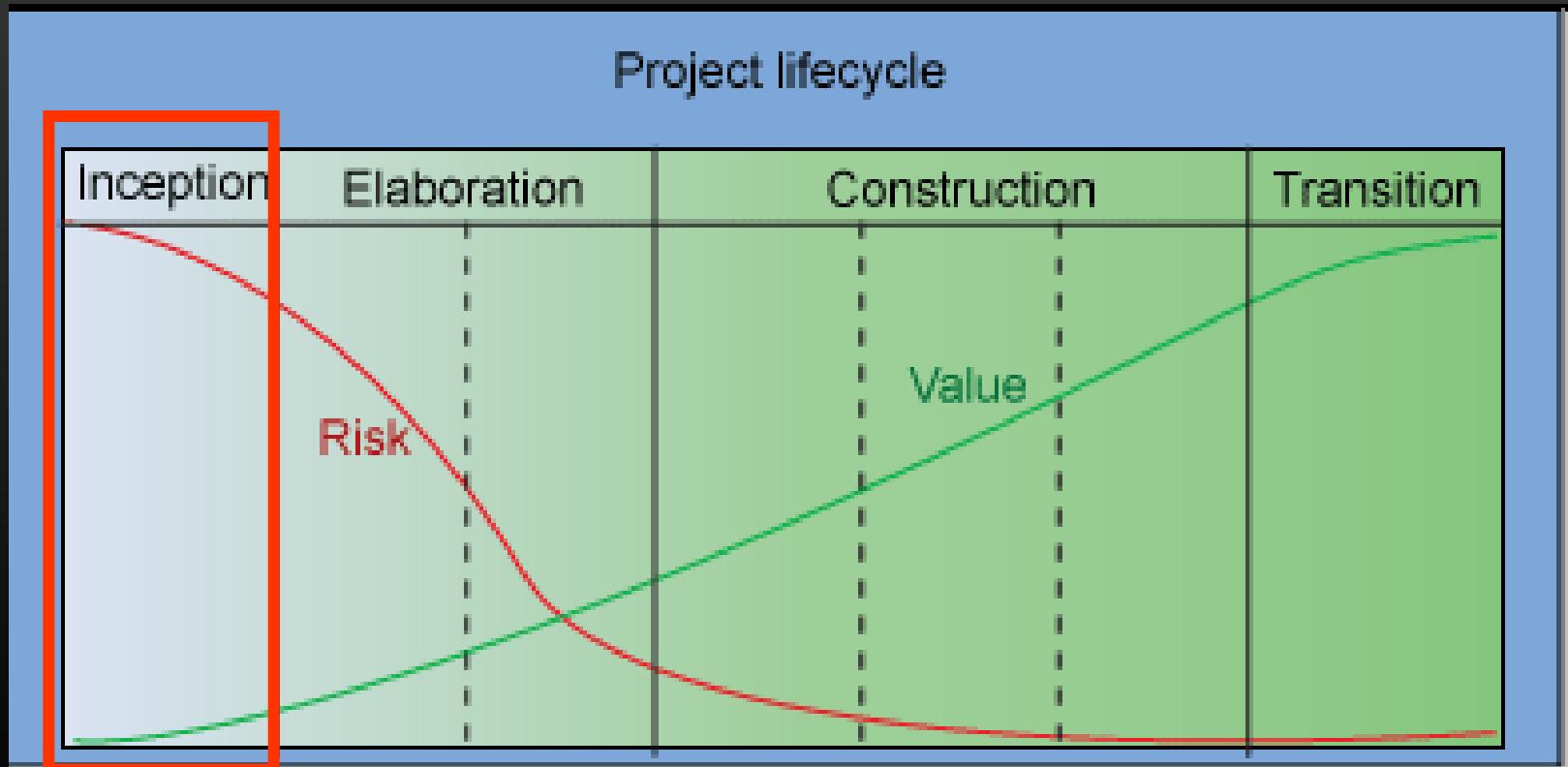
Requirements analysis is mainly performed at the beginning of the project (requirements baseline) but also during the iterations (evolutionary requirements).



The phases: Inception

Figura Project lifecycle

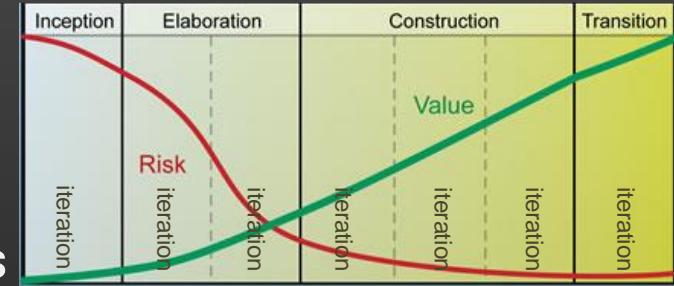
Do we agree on project scope and objectives, and whether or not the project should proceed?



Inception: Know What to Build

Typically one short iteration

Produce vision document and initial business



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

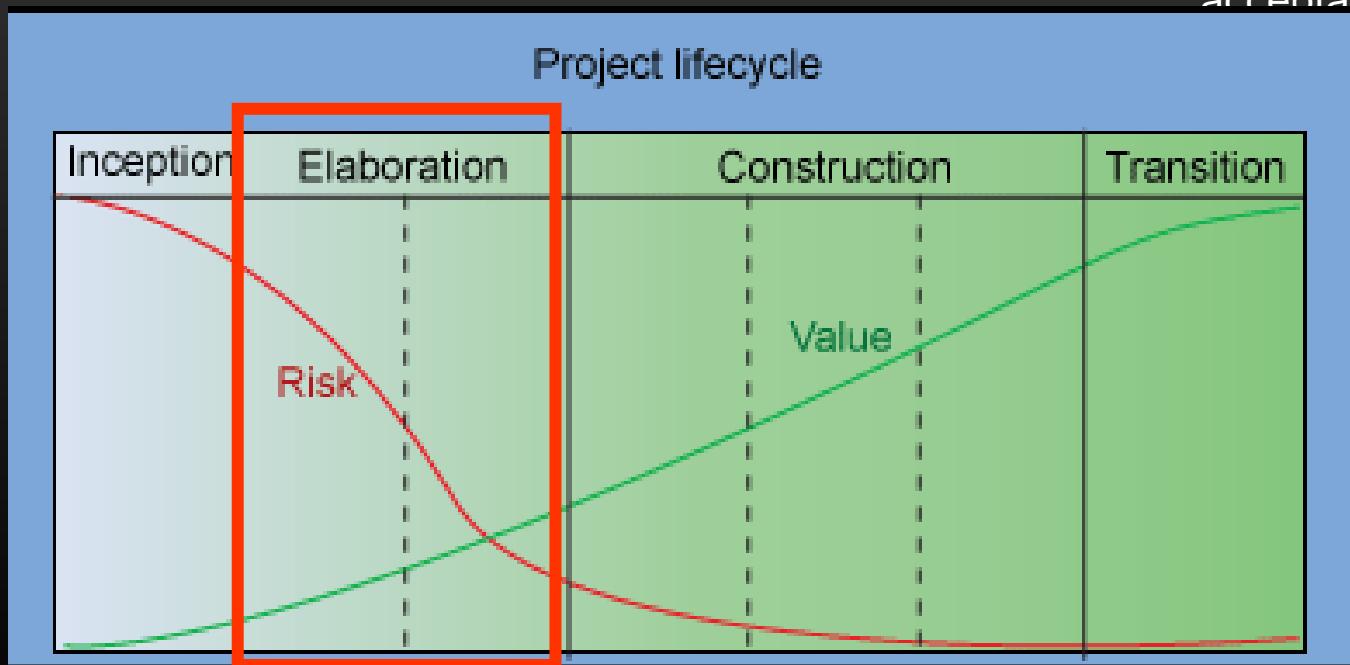
Produce conceptual prototypes as needed

Credit: Per Kroll (IBM)

The phases: elaboration

Figura

Do we agree on the executable architecture to be used for developing the application and do we find that the value delivered so far and the remaining risk is acceptable?



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

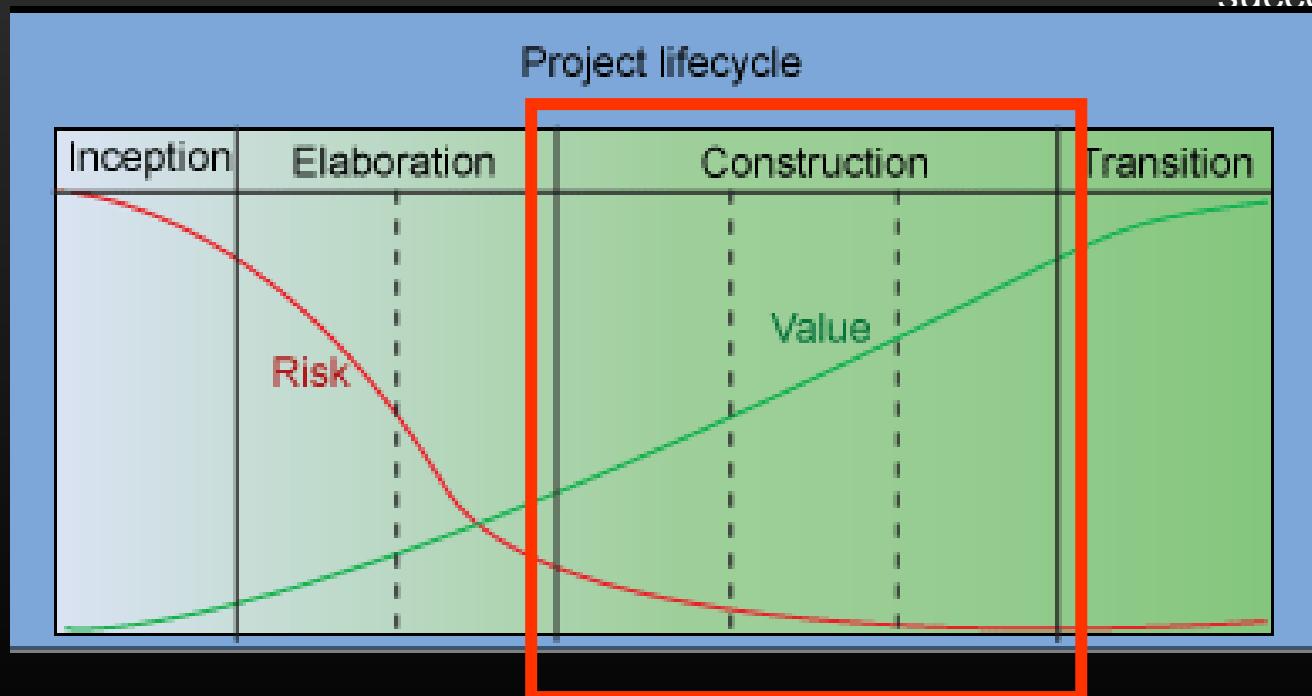
20% of use cases drive 80% of the architecture

Credit: Per Kroll (IBM)

The phases: Construction

Figura

Do we find that we have an application that is sufficiently close to being released that we should switch the primary focus of the team to tuning, polishing and ensuring successful deployment?

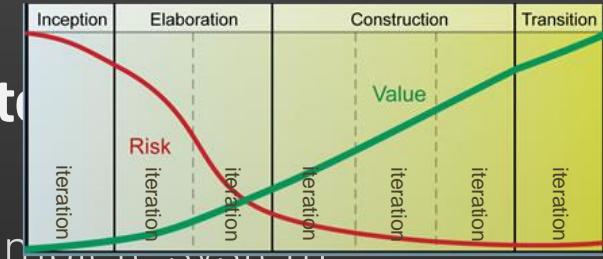


Construction: Build The Product

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

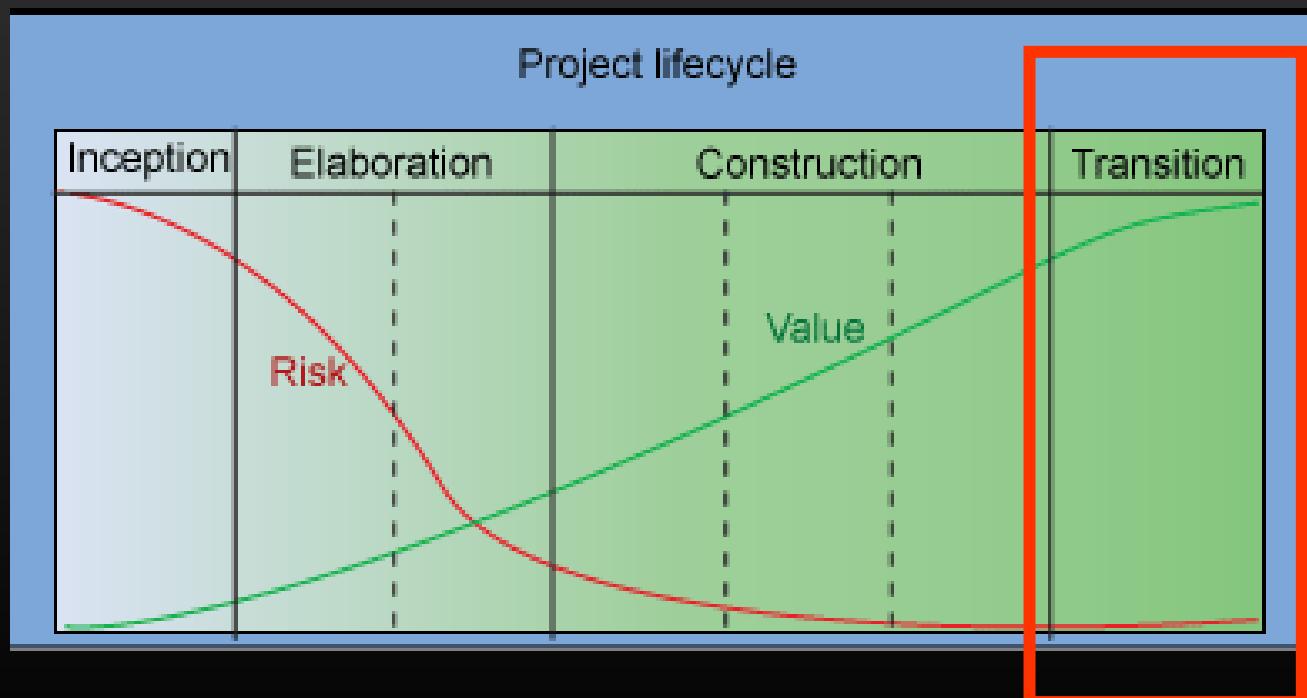
Safety or mission critical systems

Credit: Per Kroll (IBM)

The phases: Transition

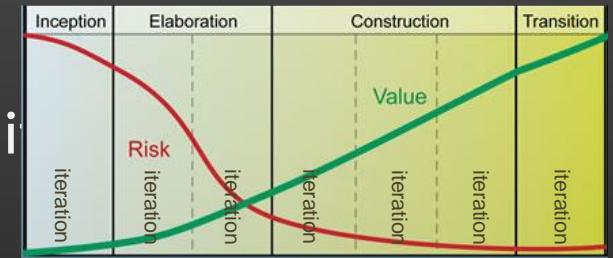
Figura

Is the application ready to release?



Transition: Stabilize and Deploy

Project moves from focusing on new capabilities and tuning



Produce incremental 'bug-fix' releases

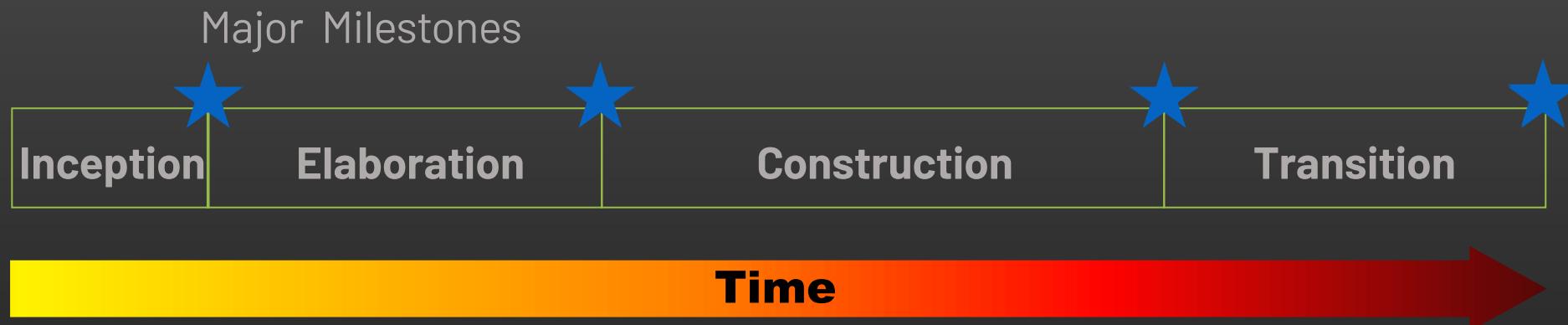
Update user manuals and deployment documentation

Execute cut-over

Conduct "post-mortem" project analysis

Credit: Per Kroll (IBM)

Recap main control points (lifecycle objective 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

Project assignment

"Partial" OpenUP

Iteration #1

- Covers Inceptions activities
- Develop the concept

Iteration #2

- Elaboration for a small set of core use cases
- Focus on detailing the use cases and prototype

Iteration #3

- Elaboration for more use cases
- Architecture validation by building some

Iteration #4

- Implement core use cases

The screenshot shows a web-based project management system for OpenUP. The top navigation bar includes links for 'Glossary', 'Feedback', 'Home', and 'Manage'. On the left, a sidebar titled 'Team' lists various sections: 'Where am I', 'Tree Sets', and a tree view under 'Team' with nodes like 'Introduction to OpenUP', 'Getting Started', 'Delivery Processes', 'OpenUP Lifecycle', 'Practices', 'Roles', 'Work Products', 'Tasks', 'Guidance', 'Tools', and 'Release Info'. The main content area displays the 'Introduction to OpenUP' page, which is currently being modified by Michael Yang. Below the title, there's a 'Main Description' section with icons for 'Getting Started', 'Core Principles', 'Roles', 'Work Products', 'Disciplines', and 'Lifecycle'. A sidebar on the right provides 'Expand All Sections' and 'Collapse' options.

Key reference:
→ [Project execution plan](#) doc

UP: Conceção

Objetivo	Atividades	Produtos
<ul style="list-style-type: none">• Atingir um consenso entre os diversos stakeholders acerca dos objetivos e âmbito do projeto.• Garantir que as condições necessárias à viabilidade do projeto estão reunidos.	<ul style="list-style-type: none">• Elaborar modelo de requisitos de alto nível.• Identificar interações com entidades externas.• Casos de utilização levantados (os de maior risco podem ser detalhados).• Planeamento das fases subsequentes e pontos de decisão.	<ul style="list-style-type: none">• Visão geral do problema• Modelo de Casos de Utilização (alto nível, parcial)• Avaliação de risco inicial• Justificação da viabilidade do projeto• Plano de projeto• Protótipos iniciais (para mitigação de risco, se necessário).

UP: Elaboração (*Elaboration*)

Objetivo	Atividades	Produtos
<ul style="list-style-type: none">• Validar os cenários• Definir a arquitetura	<ul style="list-style-type: none">• Detalhar o modelo de casos de utilização• Analisar domínio• Definir arquitetura candidata• Validar arquitetura com implementação exploratória	<ul style="list-style-type: none">• Modelo de Casos de Utilização (especificação abrangente)• Requisitos (incluindo não-funcionais)• Descrição da arquitetura do software• Protótipos (mitigação de risco).• Implementação exploratória (validar arquitetura).• Plano de projeto revisto• Medidas para mitigação do risco

UP: Construção

Objetivo	Atividades	Produtos
<ul style="list-style-type: none">Entregar valor (cenários nucleares)	<ul style="list-style-type: none">Definir realização dos casos de utilização (interações entre objetos)Desenvolver UI/UXImplementar códigoValidar incremento (garantia de qualidade)	<ul style="list-style-type: none">Implementação parcial (cenários core).Garantia de qualidade: demonstração de testes funcionais

47006- ANÁLISE E MODELAÇÃO DE SISTEMAS

Requirements management: tracking and evolve

Ilídio Oliveira

V2020/10/30 | TP-08

Learning objectives for this lecture

- Distinguish user-centric and product-centric requirements specification
- Describe requirements documentation techniques
- Understand the relation between requirements and use cases
- Identify the requirements related disciplines and activities in the OpenUP
- Explain the meaning of evolutionary requirements concept
- Describe the types of information the Analyst collects in the requirements engineering activities
- Explain how to implement requirements tracking activities

Requirements engineering activities

Discover/develop

Requirements gathering and elicitation

Critical aspect in the overall lifecycle

The approach to requirements development should be adapted to the kind of project in hands.

Manage

Document, track and evolve

Apply a change process (assess the risk,...)

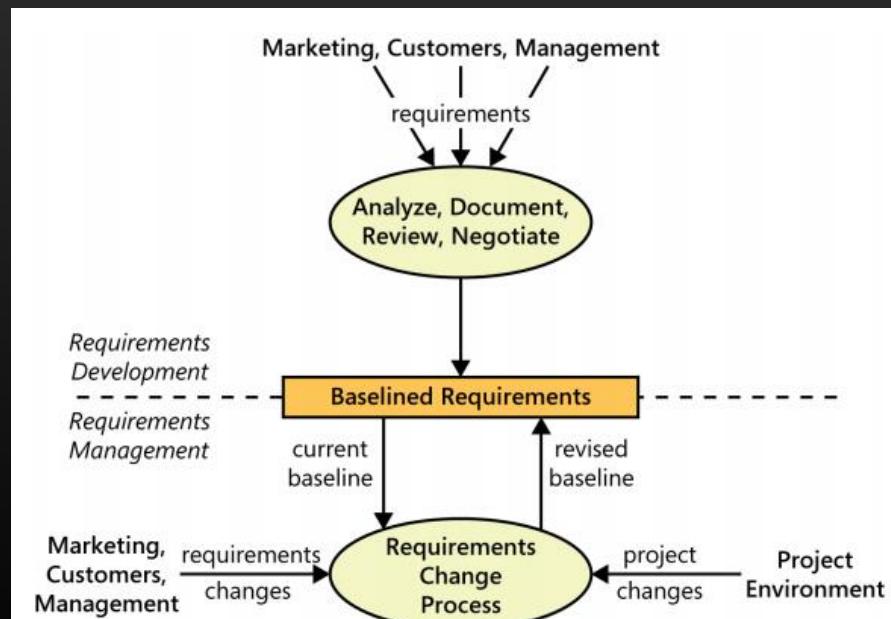


FIGURE 1-5 The boundary between requirements development and requirements management.

Requirements and/with use cases

Which main elicitation approaches exist?

What is the goal the user wants to achieve?.

What capability should the system process?

Usage-centric or product-centric?

Requirements elicitation typically takes either a usage-centric or a product-centric approach, although other strategies also are possible. The usage-centric strategy emphasizes understanding and exploring user goals to derive the necessary system functionality. The product-centric approach focuses on defining features that you expect will lead to marketplace or business success. A risk with product-centric strategies is that you might implement features that don't get used much, even if they seemed like a good idea at the time. We recommend understanding business objectives and user goals first, then using that insight to determine the appropriate product features and characteristics.

Product centric: “The system shall....”

#	Requisito
RF.1	O sistema deve permitir a um profissional criar um novo pedido de adesão , em auto-serviço, na web.
RF.2	O sistema deve enviar credenciais temporárias para os pedidos de adesão e enviá-las, por email, aos solicitantes.
RF.3	O sistema deve permitir a pesquisa de cheques-dentista (emitidos) por número de utente do SNS.
RNF.1	As pesquisas de cheques-dentista têm de retornar resultados em <5 segundos ou um evento de tempo expirado.
...	

What are well-formed requirements? (ISO-IEEE 29148)

A statement that:

- has to be met or possessed by a system to solve a stakeholder problem or to achieve a stakeholder objective,
- can be verified,
- is qualified by measurable conditions and bounded by constraints, and
- defines the performance of the system when used by a specific stakeholder or the corresponding capability of the system, but not a capability of the user, operator, or other stakeholder.

The image shows a screenshot of the ISO website. At the top, there is a red header bar with the ISO logo. Below it, a white navigation bar contains links for "Standards", "About us", "News", "Taking part", "Store" (which is highlighted in red), and a search icon. The main content area features the title "ISO/IEC/IEEE 29148:2018" in large bold letters, followed by the subtitle "Systems and software engineering — Life cycle processes — Requirements engineering". A red banner at the bottom left of the content area reads "ICS > 35 > 35.080".

Elements of style

A requirement is a statement which expresses a need and its associated constraints and conditions.

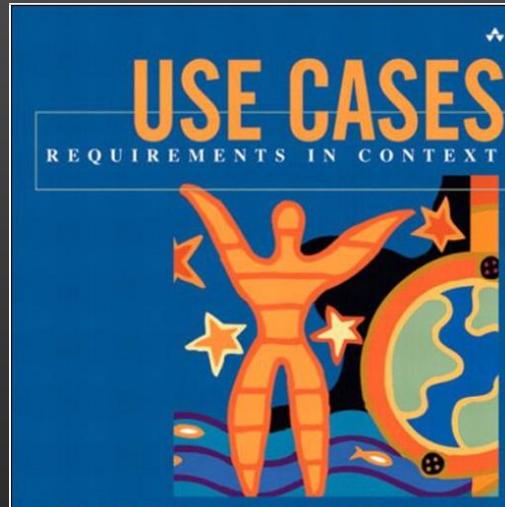
If expressed in the form of a natural language, the statement should comprise a subject, a verb and a complement. A requirement shall state the subject of the requirement (e.g., the system, the software, etc.) and what shall be done (e.g., operate at a power level, provide a field for).

E.g: The Invoice System [Subject], shall display pending customer invoices [Action] in ascending order [Value] in which invoices are to be paid.

User/usage centric: capture requirements by telling stories

A) Use cases

Focus on the actor/system interaction scenario
UML supported



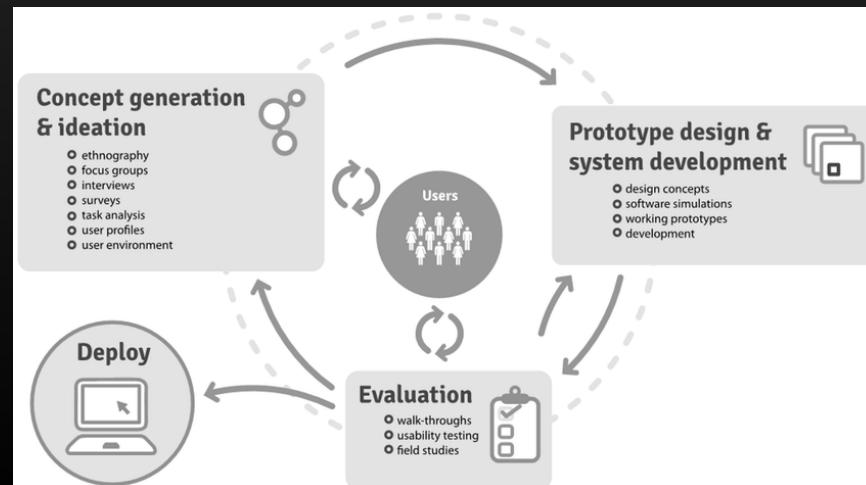
The screenshot shows a digital interface for creating a customer journey map. It includes a profile picture of Daniel Ferreira, a scenario (Entrada na UAI), objectives (Poder Amigos, Deixar os pais orgulhosos, Ter um curso superior), and a timeline of steps and associated emotions.

B) Customer Journey maps

Focus on the user comprehensive experience
Look for pain-points & opportunities

C) User-centered design

Depart from personas & scenarios elaboration
Evolve through prototyping & validation cycles



CaU contam histórias que mostram os requisitos funcionais em contexto

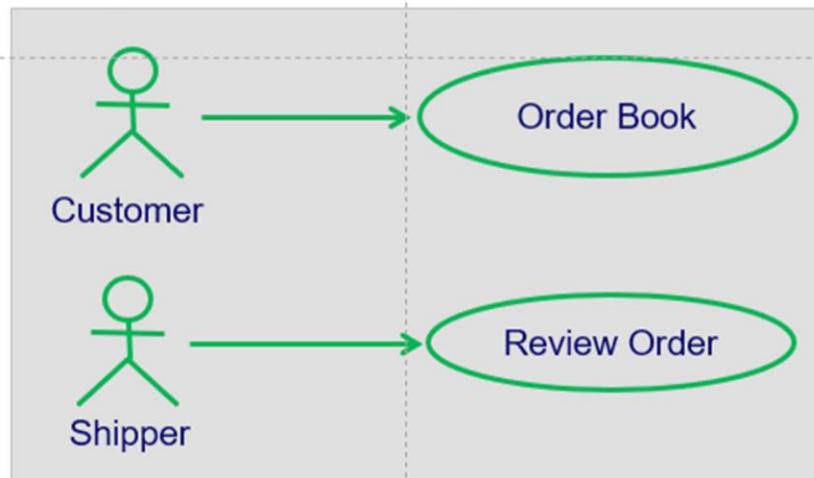
Caso Utiliz.:	Consultar informação clínica (referenciação)
Motivação:	O Médico Dentista (MD) acede ao sistema para consulta informação clínica inserida pelo médico assistente.
Sequência típica:	<ol style="list-style-type: none">1. Inicia-se quando o Médico Dentista recebe um Utente portador de Cheque-Dentista para consulta.2. O MD acede à opção de pesquisa na sua página de entrada.3. O sistema apresenta o formulário de pesquisa, com as hipóteses de pesquisa por nr utente, nome, género e data de Nascimento.4. O MD insere o número de utente do SNS e confirma a pesquisa.5. O sistema pesquisa os cheques-dentista existentes para aquele utente e apresenta uma listagem ordenada do mais recente para o mais antigo.6. O MD seleciona uma entrada na lista para abrir a informação de detalhe.7. O sistema apresenta para esse CD o cabeçalho com a identificação do cheque e utente, e uma seção com a informação clínica disponível.
Sequências	...
RF.3	O sistema deve permitir a pesquisa de cheques-dentista (emitidos) por número de utente do SNS.

OpenUP recommended practices

Use Case Driven Development 🏆



- This practice describes how to capture requirements with a combination of use cases and system-wide requirements, and then drive development and testing from those use cases.



Guidelines to apply use cases

You don't need to apply
OpenUP to develop use cases

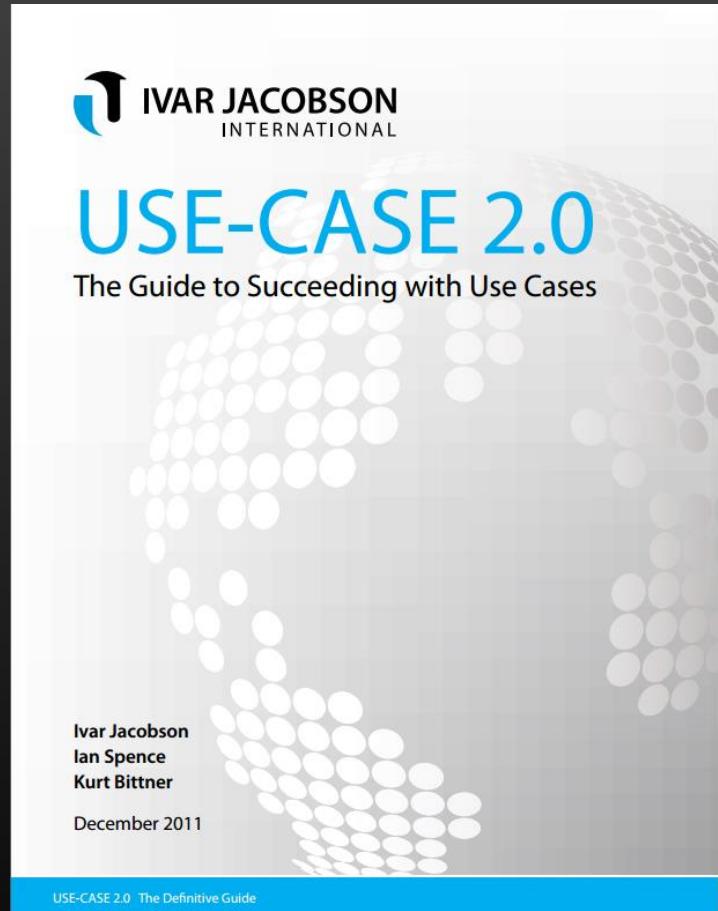
The technique works for other
types of methods

Jabson offers updated views

Adapt an "old" technique to
modern agile approaches

The white paper is well worth
reading!

Conceito apresentado originalmente em:
Jacobson, I., 1993. *Object-oriented software engineering: a use case driven approach*. Pearson Education.

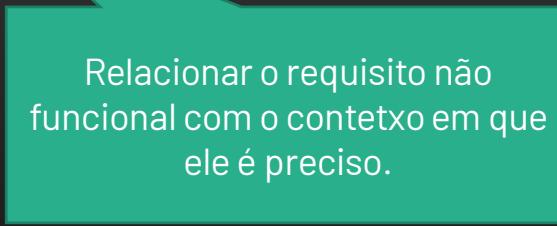


[https://www.ivarjacobson.com/
publications/white-papers/use-
case-ebook](https://www.ivarjacobson.com/publications/white-papers/use-case-ebook)

Can use cases address non-functional requirements?

Use-Case 2.0: Handling all types of requirement

Although they are one of the most popular techniques for describing systems' functionality, use cases are also used to explore their non-functional characteristics. The simplest way of doing this is to capture them as part of the use cases themselves. For example, relate performance requirements to the time taken between specific steps of a use case or list the expected service levels for a use case as part of the use case itself.



Relacionar o requisito não funcional com o contexto em que ele é preciso.

Take note of NFR in the use case description

→ Special requirements section in this [model](#)

Documenting requirements

Documenting requirements: SRS report

A special report to document the requirements specification

- “System Proposal” [Dennis’15]
- “Software requirements specification” is an industry-standard term (ISO/IEC/IEEE 2011).

1. Introduction
1.1 Purpose
1.2 Scope
1.3 Product overview
1.3.1 Product perspective
1.3.2 Product functions
1.3.3 User characteristics
1.3.4 Limitations
1.4 Definitions
2. References
3. Specific requirements
3.1 External interfaces
3.2 Functions
3.3 Usability Requirements
3.4 Performance requirements
3.5 Logical database requirements
3.6 Design constraints
3.7 Software system attributes
3.8 Supporting information
4. Verification
(parallel to subsections in Section 3)
5. Appendices
5.1 Assumptions and dependencies
5.2 Acronyms and abbreviations

Who relies on the SRS?

Customers, the marketing department, and sales staff need to know what product they can expect to be delivered.

Project managers base their estimates of schedule, effort, and resources on the requirements.

Software development teams need to know what to build.

Testers use it to develop requirements-based tests, test plans, and test procedures.

Documentation writers base user manuals and help screens on the SRS and the user interface design.

Training personnel use the SRS and user documentation to develop educational materials.

Legal staff ensures that the requirements comply with applicable laws and regulations.

Subcontractors base their work on—and can be legally held to—the specified requirements.

Software requirements reports

[ISO/IEC/IEEE 29148:2011](#): Systems and software engineering -- Life cycle processes -- Requirements engineering

1. Introduction

- 1.1 Purpose
- 1.2 Scope
- 1.3 Product overview
 - 1.3.1 Product perspective
 - 1.3.2 Product functions
 - 1.3.3 User characteristics
 - 1.3.4 Limitations
- 1.4 Definitions

2. References

3. Specific requirements

- 3.1 External interfaces
- 3.2 Functions
- 3.3 Usability Requirements
- 3.4 Performance requirements
- 3.5 Logical database requirements
- 3.6 Design constraints
- 3.7 Software system attributes
- 3.8 Supporting information

4. Verification

(parallel to subsections in Section 3)

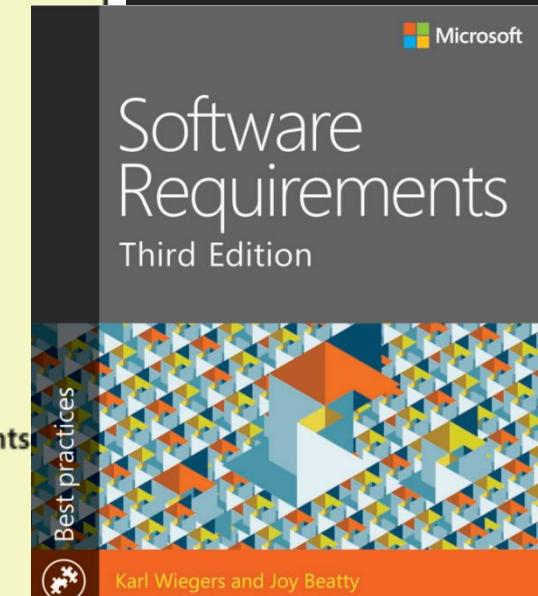
5. Appendices

- 5.1 Assumptions and dependencies
- 5.2 Acronyms and abbreviations

Figure 8 — Example SRS Outline

Wieger's proposal

- 1. Introduction**
 - 1.1 Purpose
 - 1.2 Document conventions
 - 1.3 Project scope
 - 1.4 References
 - 2. Overall description**
 - 2.1 Product perspective
 - 2.2 User classes and characteristics
 - 2.3 Operating environment
 - 2.4 Design and implementation constraints
 - 2.5 Assumptions and dependencies
 - 3. System features**
 - 3.x System feature X
 - 3.x.1 Description
 - 3.x.2 Functional requirements
 - 4. Data requirements**
 - 4.1 Logical data model
 - 4.2 Data dictionary
 - 4.3 Reports
 - 4.4 Data acquisition, integrity, retention, and disposal
 - 5. External interface requirements**
 - 5.1 User interfaces
 - 5.2 Software interfaces
 - 5.3 Hardware interfaces
 - 5.4 Communications interfaces
 - 6. Quality attributes**
 - 6.1 Usability
 - 6.2 Performance
 - 6.3 Security
 - 6.4 Safety
 - 6.x [others]
 - 7. Internationalization and localization requirements**
 - 8. Other requirements**
- Appendix A: Glossary**
- Appendix B: Analysis models**



Requirements-related artifacts in (Open)UP?

Use-Case Model

A set of typical scenarios of using a system. There are primarily for **functional** (behavioral) requirements.

Supplementary Specification (System-wide requirements)

Basically, everything not in the use cases. This artifact is primarily for all **non-functional** requirements, such as performance or licensing. It is also the place to record functional features not expressed (or expressible) as use cases (for example, a report generation).

Glossary

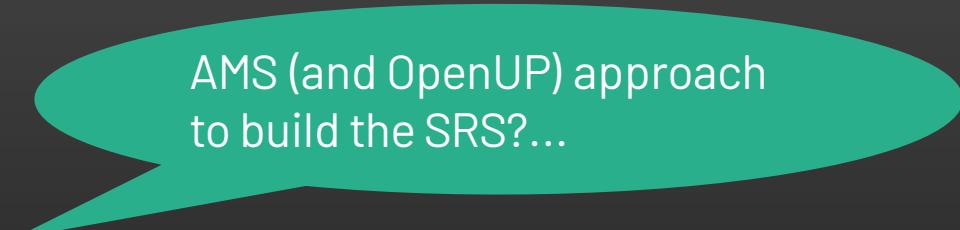
In its simplest form, the Glossary defines noteworthy terms. It also encompasses the concept of the data dictionary, which records **requirements related to data**, such as validation rules, acceptable values, and so forth. The Glossary can detail any element: an attribute of an object, a parameter of an operation call, a report layout, and so forth.

Business Rules

Business rules (also called Domain Rules) typically describe **requirements or policies that transcend one software project** they are required in the domain or business, and many applications may need to conform to them. An good example is government tax laws.

Examples of SRS reports

- SRS example from Wiegers
- System Proposal from Dennis (chap. 3)
- OpenUP: use cases + system-wide requirements



AMS (and OpenUP) approach
to build the SRS?...

Evolutionary requirements

Evolutionary requirements

Plan-driven, “waterfall” approach

Attempting to fully define and stabilize the requirements in the first phase of a project before programming

Requirements defined up-front for the entire systems

Agile, evolutionary approach

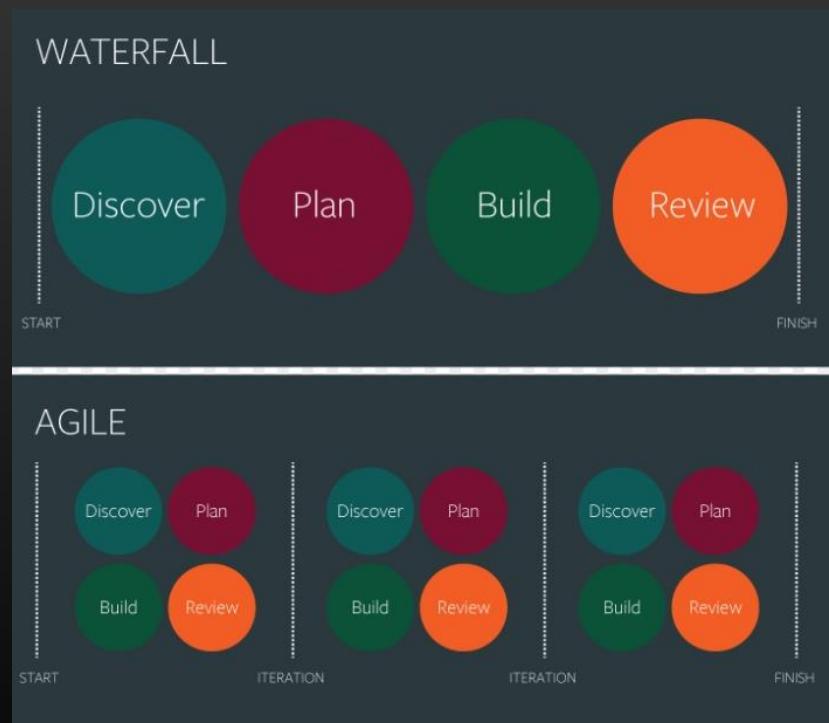
Accept and prepare for the inevitably changing and unclear stakeholder's wishes/needs

Some sort of systematic approach to finding, documenting, organizing, and tracking the *changing* requirements of a system

High-level, soon; detailed, closer to the build process

Wieggers:

You don't have to write the SRS for the entire product before beginning development, but you should capture the requirements for each increment before building that increment.



Does OpenUP propose evolutionary requirements?

Practices > Technical Practices > Use Case Driven Development > Tasks > Identify and Outline Requirements

Task: Identify and Outline Requirements



This task describes how to identify and outline the requirements for the system so that the scope of work can be determined.

Disciplines: Requirements



Expand All Sections



Collapse All Sections

Purpose

The purpose of this task is to identify and capture functional and non-functional requirements for the system. These requirements form the basis of communication and agreement between the stakeholders and the development team on what the system must do to satisfy stakeholder needs. The goal is to understand the requirements at a high-level so that the initial scope of work can be determined. Further analysis will be performed to detail these requirements prior to implementation.

→ Back | Home

Requirements engineering practices (Wiegers)

Requirements development activities

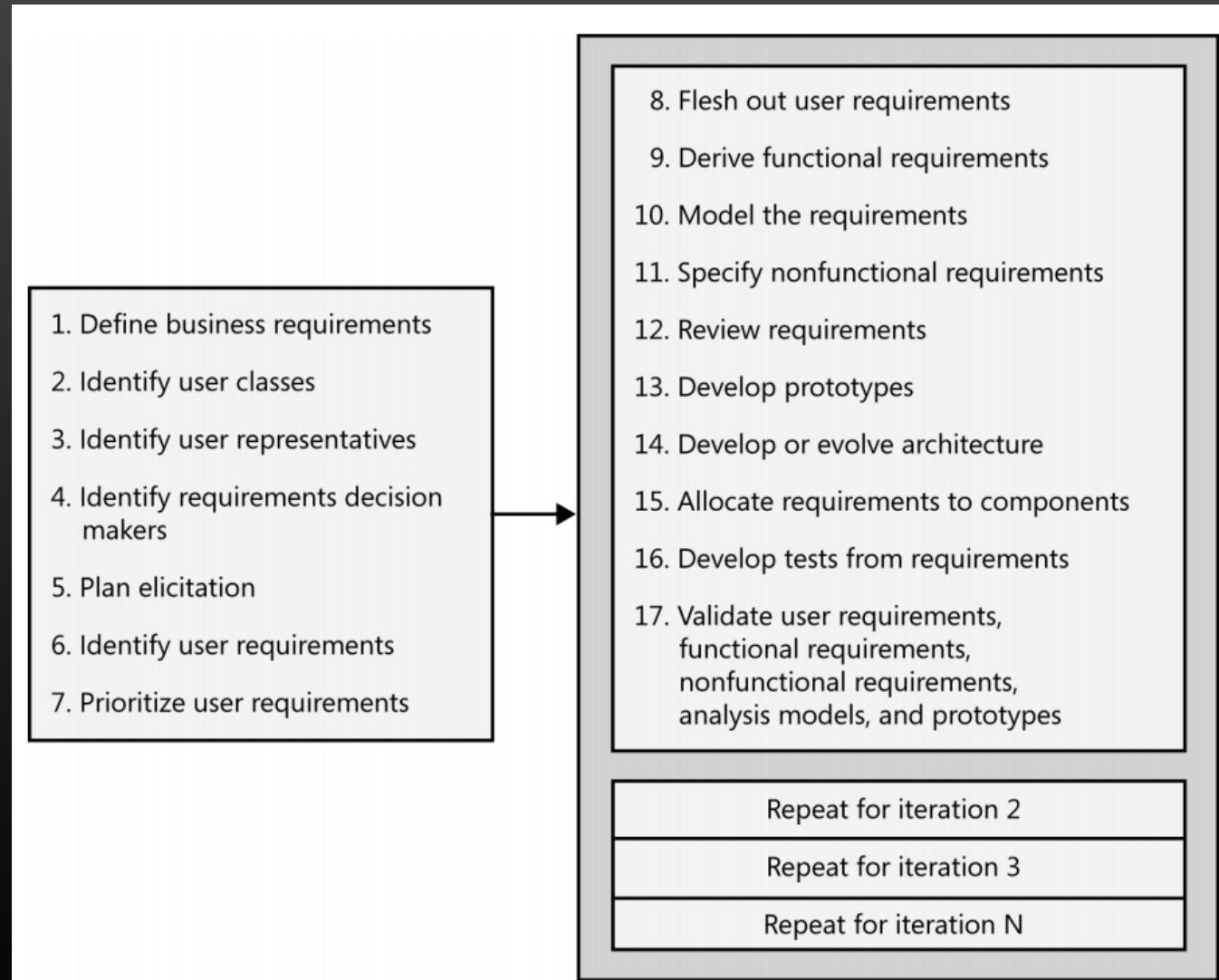


TABLE 3-1 Requirements engineering good practices

Elicitation	Analysis	Specification	Validation
<ul style="list-style-type: none">■ Define vision and scope■ Identify user classes■ Select product champions■ Conduct focus groups■ Identify user requirements■ Identify system events and responses■ Hold elicitation interviews■ Hold facilitated elicitation workshops■ Observe users performing their jobs■ Distribute questionnaires■ Perform document analysis■ Examine problem reports■ Reuse existing requirements	<ul style="list-style-type: none">■ Model the application environment■ Create prototypes■ Analyze feasibility■ Prioritize requirements■ Create a data dictionary■ Model the requirements■ Analyze interfaces■ Allocate requirements to subsystems	<ul style="list-style-type: none">■ Adopt requirement document templates■ Identify requirement origins■ Uniquely label each requirement■ Record business rules■ Specify nonfunctional requirements	<ul style="list-style-type: none">■ Review the requirements■ Test the requirements■ Define acceptance criteria■ Simulate the requirements
Requirements management		Knowledge	Project management
<ul style="list-style-type: none">■ Establish a change control process■ Perform change impact analysis■ Establish baselines and control versions of requirements sets■ Maintain change history■ Track requirements status■ Track requirements issues■ Maintain a requirements traceability matrix■ Use a requirements management tool		<ul style="list-style-type: none">■ Train business analysts■ Educate stakeholders about requirements■ Educate developers about application domain■ Define a requirements engineering process■ Create a glossary	<ul style="list-style-type: none">■ Select an appropriate life cycle■ Plan requirements approach■ Estimate requirements effort■ Base plans on requirements■ Identify requirements decision makers■ Renegotiate commitments■ Manage requirements risks■ Track requirements effort■ Review past lessons learned

Prototyping as a requirements validation practice

TABLE 15-1 Typical applications of software prototypes

	Throwaway	Evolutionary
Mock-up	<ul style="list-style-type: none">■ Clarify and refine user and functional requirements.■ Identify missing functionality.■ Explore user interface approaches.	<ul style="list-style-type: none">■ Implement core user requirements.■ Implement additional user requirements based on priority.■ Implement and refine websites.■ Adapt system to rapidly changing business needs.
Proof of concept	<ul style="list-style-type: none">■ Demonstrate technical feasibility.■ Evaluate performance.■ Acquire knowledge to improve estimates for construction.	<ul style="list-style-type: none">■ Implement and grow core multi-tier functionality and communication layers.■ Implement and optimize core algorithms.■ Test and tune performance.

Evolving requirements: life after the baseline

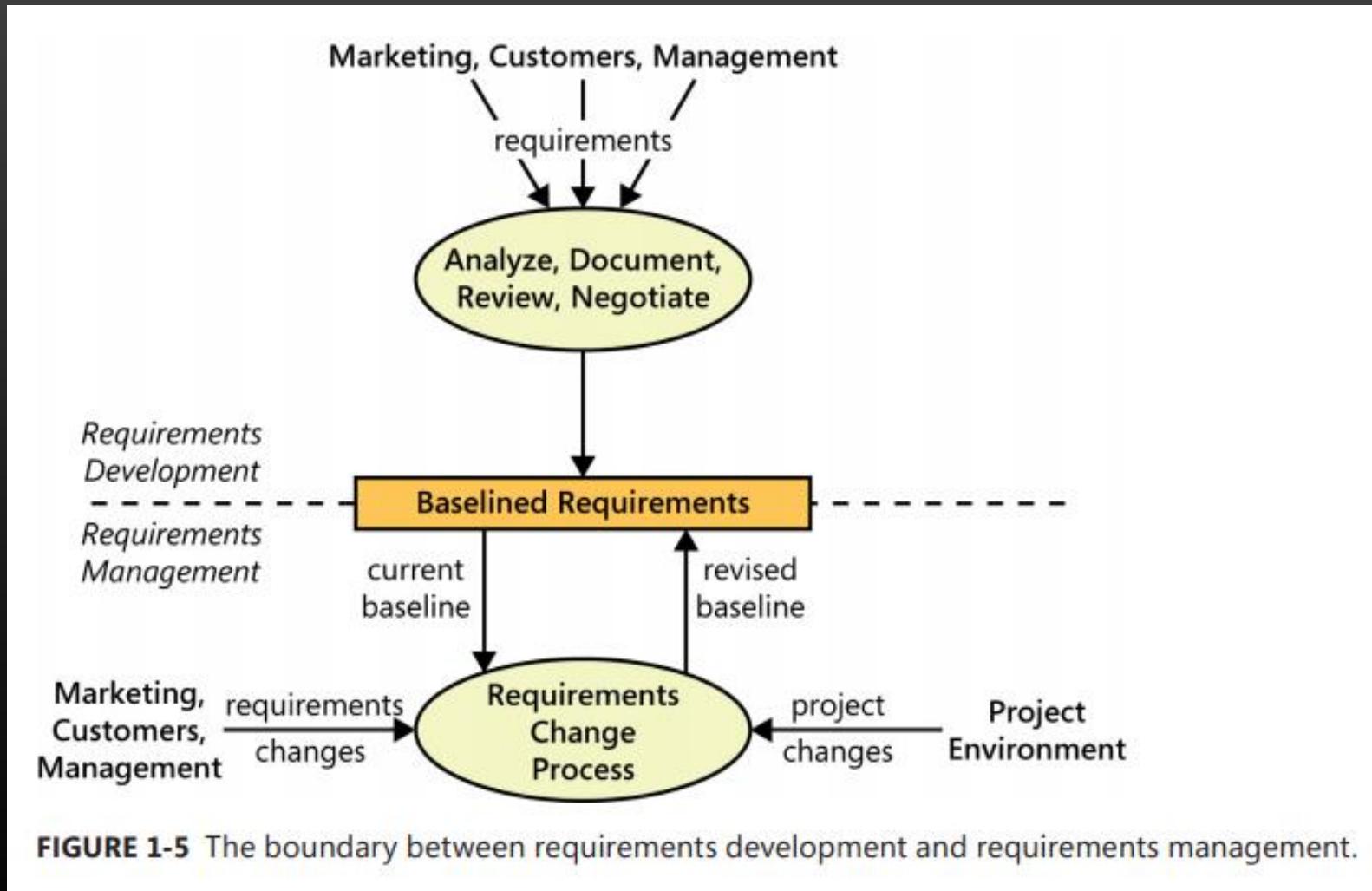
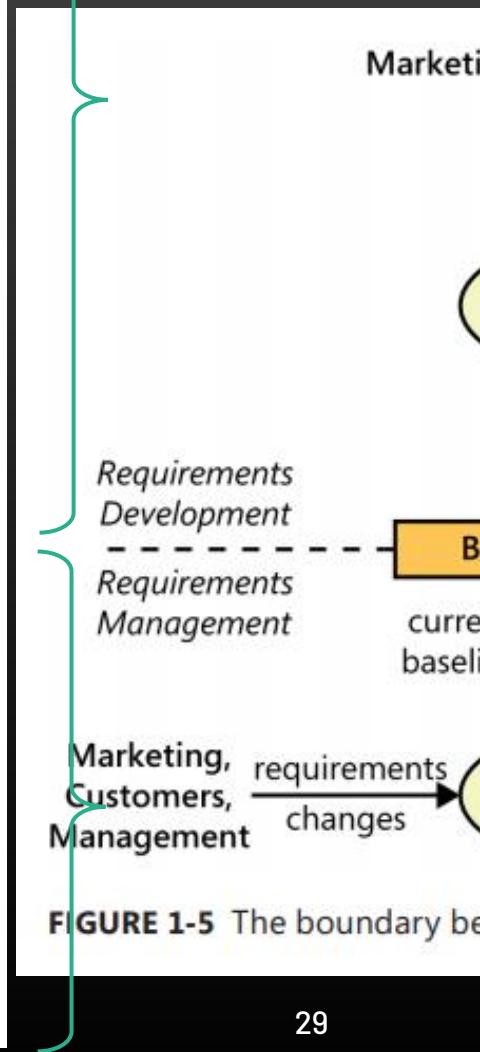


TABLE 3-1 Requirements engineering good practices

Elicitation	Analysis	Specification	Validation
<ul style="list-style-type: none">■ Define vision and scope■ Identify user classes■ Select product champions■ Conduct focus groups■ Identify user requirements■ Identify system events and responses■ Hold elicitation interviews■ Hold facilitated elicitation workshops■ Observe users performing their jobs■ Distribute questionnaires■ Perform document analysis■ Examine problem reports■ Reuse existing requirements	<ul style="list-style-type: none">■ Model the application environment■ Create prototypes■ Analyze feasibility■ Prioritize requirements■ Create a data dictionary■ Model the requirements■ Analyze interfaces■ Allocate requirements to subsystems	<ul style="list-style-type: none">■ Adopt requirement document templates■ Identify requirement origins■ Uniquely label each requirement■ Record business rules■ Specify nonfunctional requirements	<ul style="list-style-type: none">■ Review the requirements■ Test the requirements■ Define acceptance criteria■ Simulate the requirements
Requirements management	Knowledge	Project management	<pre>graph LR; RD[Requirements Development] --- RM[Requirements Management]; subgraph RM [Requirements Management]; end; subgraph MC [Marketing, Customers, Management]; end; MC --> PM[Project management]; RM --> PM;</pre>



Types of information collected in requirements development

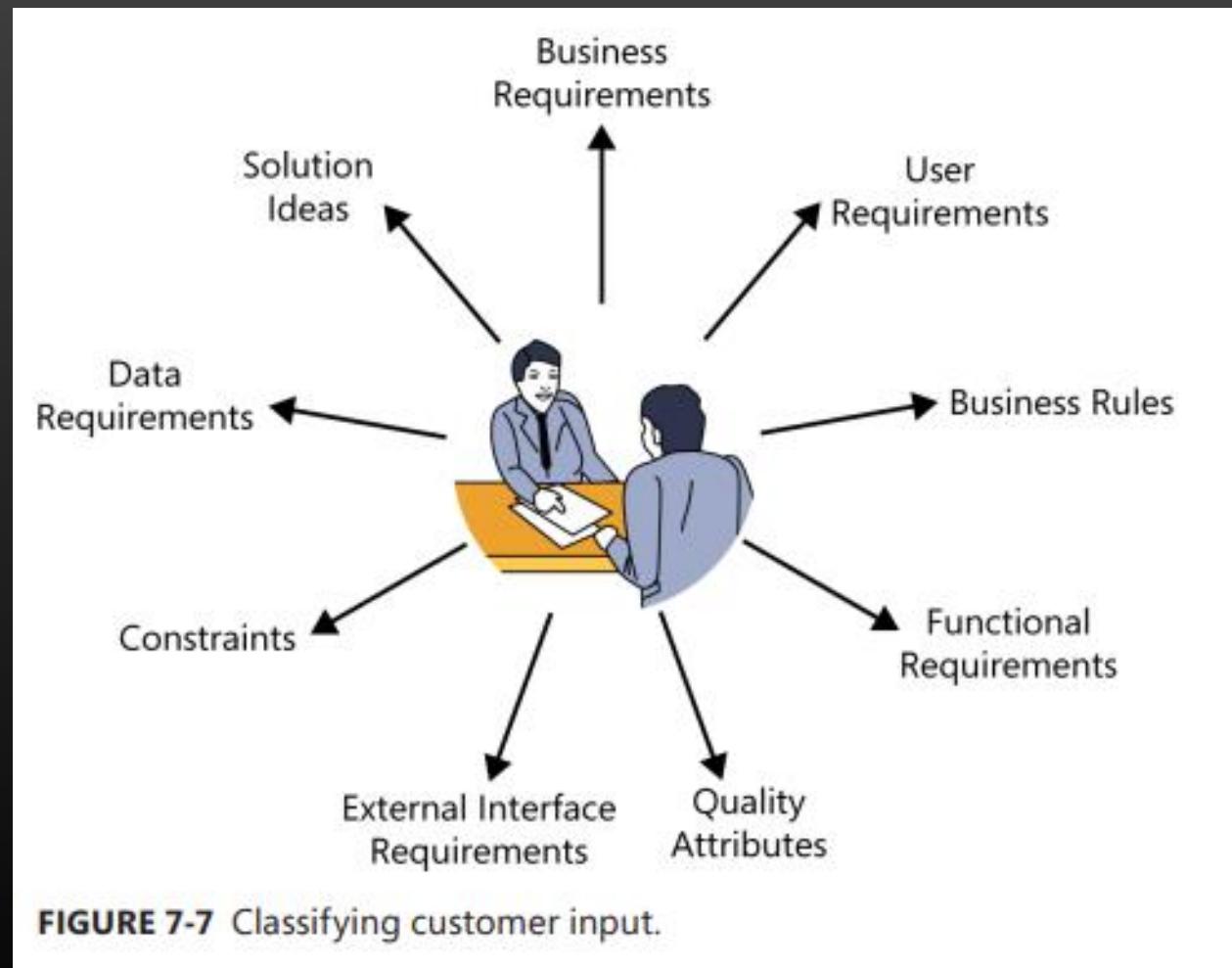


FIGURE 7-7 Classifying customer input.

TABLE 1-1 Some types of requirements information

Term	Definition
Business requirement	A high-level business objective of the organization that builds a product or of a customer who procures it.
Business rule	A policy, guideline, standard, or regulation that defines or constrains some aspect of the business. Not a software requirement in itself, but the origin of several types of software requirements.
Constraint	A restriction that is imposed on the choices available to the developer for the design and construction of a product.
External interface requirement	A description of a connection between a software system and a user, another software system, or a hardware device.
Feature	One or more logically related system capabilities that provide value to a user and are described by a set of functional requirements.
Functional requirement	A description of a behavior that a system will exhibit under specific conditions.
Nonfunctional requirement	A description of a property or characteristic that a system must exhibit or a constraint that it must respect.
Quality attribute	A kind of nonfunctional requirement that describes a service or performance characteristic of a product.
System requirement	A top-level requirement for a product that contains multiple subsystems, which could be all software or software and hardware.
User requirement	A goal or task that specific classes of users must be able to perform with a system, or a desired product attribute.

Types of information connected to requirements 1/2

Business rule

Business rule → a policy, guideline, standard, or regulation that defines or constrains some aspect of the business. Not a software requirement in itself (because they have an existence beyond the boundaries of any specific software application), but the origin of several types of software requirements.

Phrases such as "Must comply with . . .," "If <some condition is true>, then <something happens>," or "Must be calculated according to . . ." suggest that the user is describing a business rule. E.g.:

→ "A new client must pay 30 percent of the estimated consulting fee and travel expenses in advance."

→ "Time-off approvals must comply with the company's HR vacation policy."

External interface requirements

Requirements in this category describe the connections between your system and the rest of the universe.

Phrases such as "Must read signals from . . .," "Must send messages to . . .," "Must be able to read files in <format>," and "User interface elements must conform to <a standard>" indicate that the customer is describing an external interface requirement. Following are some examples:

→ "The manufacturing execution system must control the wafer sorter."

→ "The mobile app should send the check image to the bank after I photograph the check I'm depositing."

Types of information connected to requirements 2/2

Functional requirements

Functional requirements describe the observable behaviors the system will exhibit under certain conditions and the actions the system will let users take.

Here are some examples of functional requirements as you might hear them from users:

- "If the pressure exceeds 40.0 psi, the high-pressure warning light should come on."
- "The user must be able to sort the project list in forward and reverse alphabetical order."

These statements illustrate how users typically present functional requirements, but they don't represent good ways to write functional requirements. **The BA will need to craft these into more precise specifications.**

Quality attributes

Statements that describe how well the system does something are quality attributes. Listen for words that describe **desirable system characteristics**: fast, easy, user-friendly, reliable, secure.

You'll need to work with the users to understand just what they mean by these ambiguous and subjective terms so that you can write clear, variable quality goals.

The following examples suggest what quality attributes might sound like when described by users:

- "The mobile software must respond quickly to touch commands."
- "The shopping cart mechanism has to be simple to use so my new customers don't abandon the purchase."

Relationship between types of information

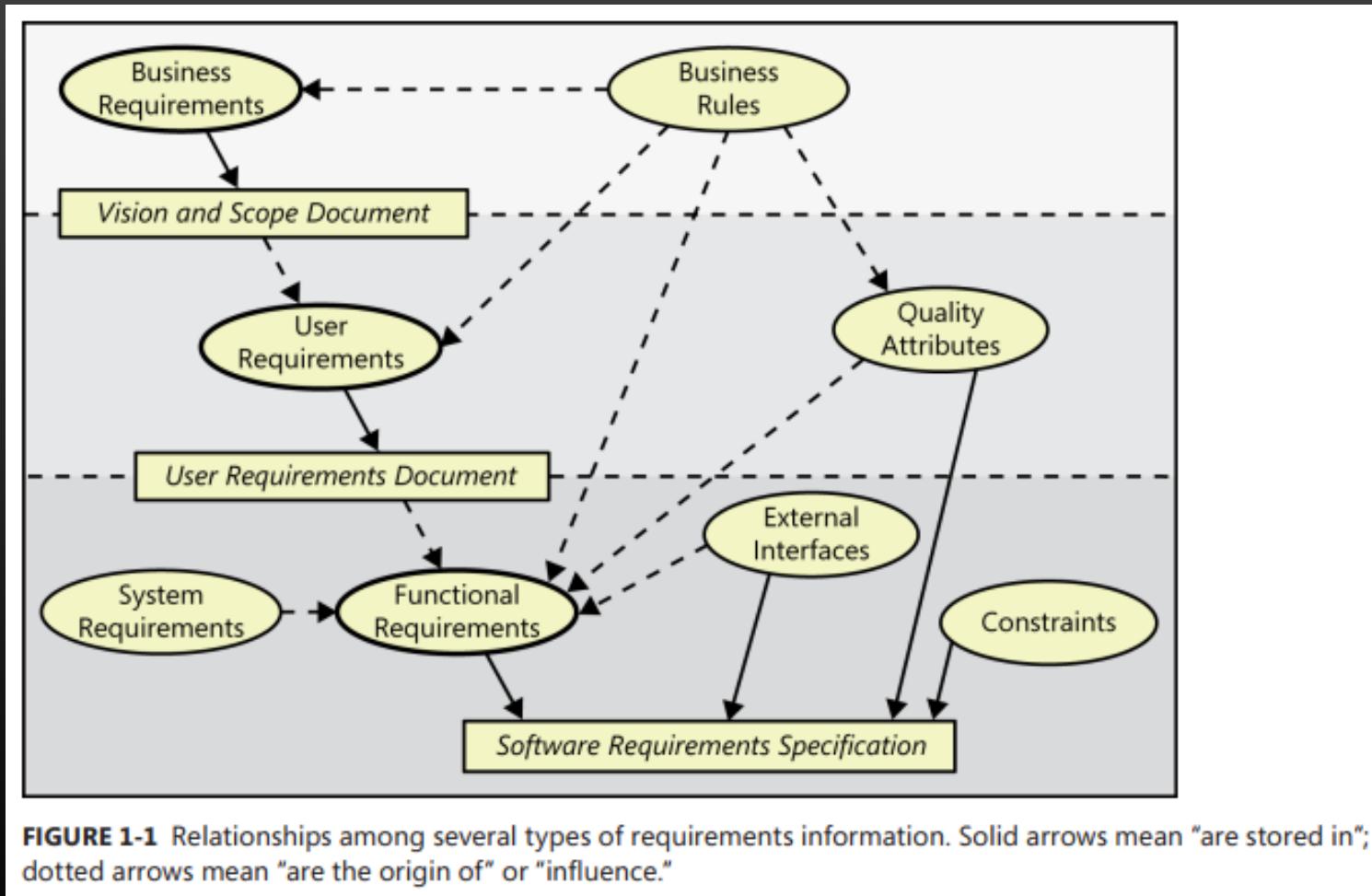
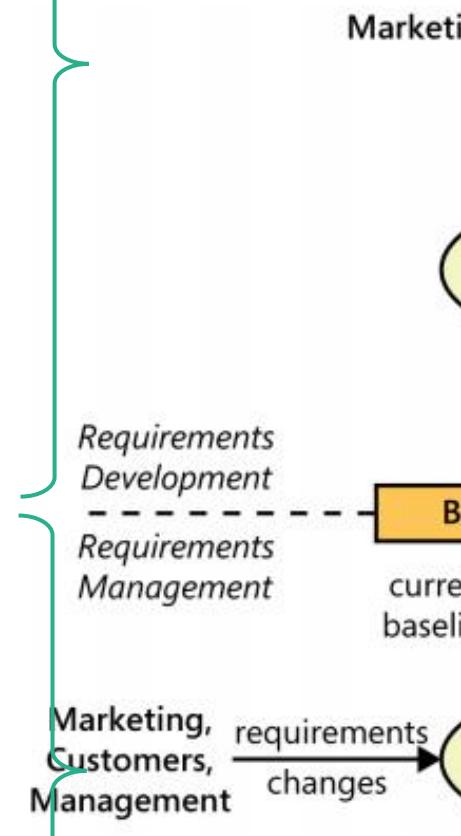


FIGURE 1-1 Relationships among several types of requirements information. Solid arrows mean “are stored in”; dotted arrows mean “are the origin of” or “influence.”

TABLE 3-1 Requirements engineering good practices

Elicitation	Analysis	Specification	Validation
<ul style="list-style-type: none">■ Define vision and scope■ Identify user classes■ Select product champions■ Conduct focus groups■ Identify user requirements■ Identify system events and responses■ Hold elicitation interviews■ Hold facilitated elicitation workshops■ Observe users performing their jobs■ Distribute questionnaires■ Perform document analysis■ Examine problem reports■ Reuse existing requirements	<ul style="list-style-type: none">■ Model the application environment■ Create prototypes■ Analyze feasibility■ Prioritize requirements■ Create a data dictionary■ Model the requirements■ Analyze interfaces■ Allocate requirements to subsystems	<ul style="list-style-type: none">■ Adopt requirement document templates■ Identify requirement origins■ Uniquely label each requirement■ Record business rules■ Specify nonfunctional requirements	<ul style="list-style-type: none">■ Review the requirements■ Test the requirements■ Define acceptance criteria■ Simulate the requirements
Requirements management <ul style="list-style-type: none">■ Establish a change control process■ Perform change impact analysis■ Establish baselines and control versions of requirements sets■ Maintain change history■ Track requirements status■ Track requirements issues■ Maintain a requirements traceability matrix■ Use a requirements management tool	Knowledge <ul style="list-style-type: none">■ Train business analysts■ Educate stakeholders about requirements■ Educate developers about application domain■ Define a requirements engineering process■ Create a glossary	Project management <ul style="list-style-type: none">■ Select an appropriate life cycle■ Plan requirements approach■ Estimate requirements effort■ Base plans on requirements■ Identify requirements decision makers■ Renegotiate commitments■ Manage requirements risks■ Track requirements effort■ Review past lessons learned	<p>The diagram illustrates the relationship between Requirements Development and Requirements Management. A vertical bracket on the left groups Elicitation, Analysis, Specification, and Validation under 'Requirements Development'. A horizontal dashed line separates this from 'Requirements Management'. To the right, a vertical bracket groups Knowledge and Project management under 'Requirements Management'. Above this, a large bracket covers all columns under 'Marketing, Customers, Management'. An arrow labeled 'changes' points from this bracket towards the 'Requirements Management' section.</p>

**FIGURE 1-5** The boundary be

What is a requirements traceability matrix?

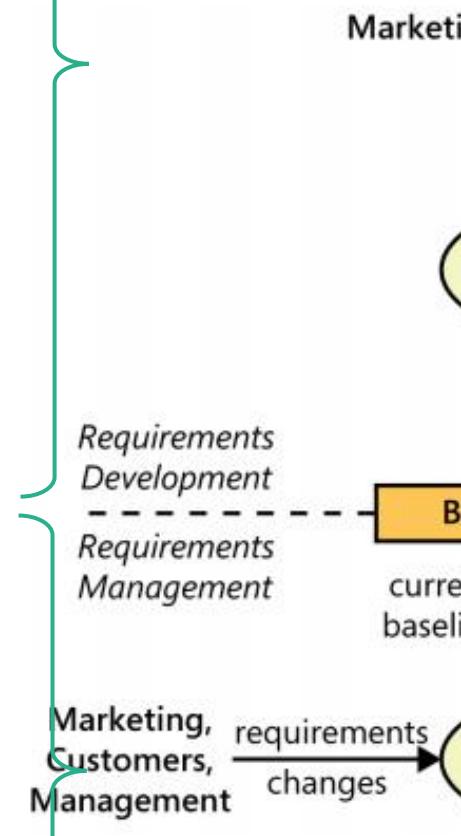
The most common way to represent the links between requirements and other system elements is in a requirements traceability matrix, also called a requirements trace matrix or a traceability table.

TABLE 29-2 Requirements traceability matrix showing links between use cases and functional requirements

Functional requirement	Use case			
	UC-1	UC-2	UC-3	UC-4
FR-1	↔			
FR-2	↔			
FR-3			↔	
FR-4			↔	
FR-5		↔		↔
FR-6			↔	

TABLE 3-1 Requirements engineering good practices

Elicitation	Analysis	Specification	Validation
<ul style="list-style-type: none">■ Define vision and scope■ Identify user classes■ Select product champions■ Conduct focus groups■ Identify user requirements■ Identify system events and responses■ Hold elicitation interviews■ Hold facilitated elicitation workshops■ Observe users performing their jobs■ Distribute questionnaires■ Perform document analysis■ Examine problem reports■ Reuse existing requirements	<ul style="list-style-type: none">■ Model the application environment■ Create prototypes■ Analyze feasibility■ Prioritize requirements■ Create a data dictionary■ Model the requirements■ Analyze interfaces■ Allocate requirements to subsystems	<ul style="list-style-type: none">■ Adopt requirement document templates■ Identify requirement origins■ Uniquely label each requirement■ Record business rules■ Specify nonfunctional requirements	<ul style="list-style-type: none">■ Review the requirements■ Test the requirements■ Define acceptance criteria■ Simulate the requirements
Requirements management <ul style="list-style-type: none">■ Establish a change control process■ Perform change impact analysis■ Establish baselines and control versions of requirements sets■ Maintain change history■ Track requirements status■ Track requirements issues■ Maintain a requirements traceability matrix■ Use a requirements management tool	Knowledge <ul style="list-style-type: none">■ Train business analysts■ Educate stakeholders about requirements■ Educate developers about application domain■ Define a requirements engineering process■ Create a glossary	Project management <ul style="list-style-type: none">■ Select an appropriate life cycle■ Plan requirements approach■ Estimate requirements effort■ Base plans on requirements■ Identify requirements decision makers■ Renegotiate commitments■ Manage requirements risks■ Track requirements effort■ Review past lessons learned	<p>The diagram illustrates the relationship between Requirements Development and Requirements Management. A vertical bracket on the left groups Elicitation, Analysis, Specification, and Validation under 'Requirements Development'. A horizontal dashed line separates Requirements Development from Requirements Management. To the right of Requirements Management, three boxes labeled 'Marketing', 'Customers', and 'Management' are connected by arrows pointing towards Requirements Management, indicating that these external factors influence 'changes' in requirements.</p>

**FIGURE 1-5** The boundary be

Requirement attributes to consider:

- Date the requirement was created
- Current version number of the requirement
- Author who wrote the requirement
- Priority
- Status
- Origin or source of the requirement
- Rationale behind the requirement
- Release number or iteration to which the requirement is allocated
- Stakeholder to contact with questions or to make decisions about proposed changes
- Validation method to be used or acceptance criteria

Requirements Management Tool

An RM tool provides a robust solution to the limitations of storing requirements in documents.

Small project teams can get away with just entering the requirements text and several attributes of each requirement.

Larger project teams will benefit from letting users import requirements from source documents, define attribute values, filter and display the database contents, export requirements in various formats, define traceability links, and connect requirements to items stored in other software development tools.

E.g.: [IBM Rational DOORS](#)

What are the features of a requirements management tool?

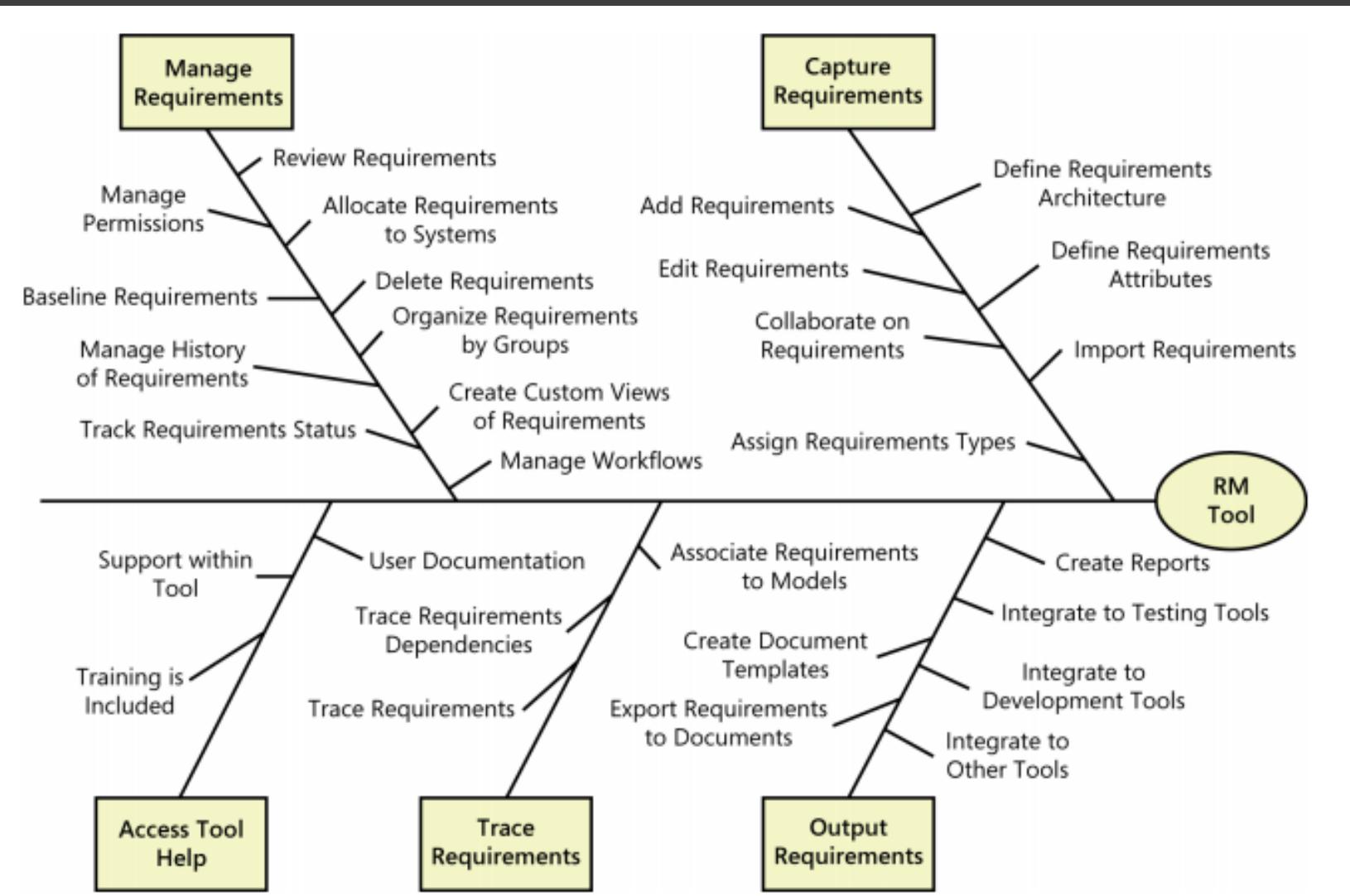


FIGURE 30-1 Common RM tool features.



And second, those of us in the software domain tend to be enamored with technical and process solutions to our challenges. We sometimes fail to appreciate that requirements elicitation—and much of software and systems project work in general—is primarily a human interaction challenge. No magical new techniques have come along to automate that, although various tools are available to help geographically separated people collaborate effectively.

In: Wiegers, "Software Requirements"

Readings & references

Core readings	Suggested readings
<ul style="list-style-type: none">• [Wiegers13] – Multiple chapters used.	<ul style="list-style-type: none">• [Dennis15] – Chap. 3 – Requirements Determination• [Pressman15] – Chap. 8 – Understanding Requirements

47006- ANÁLISE E MODELAÇÃO DE SISTEMAS

Modeling the domain concepts with classes

Ilídio Oliveira

v2020/11/04, TP09

Learning objectives for this lecture

- Justify the use of structural models in systems specification.
- Draw a simple class diagram to capture the concepts of a problem domain.
- Describe the types and roles of the different associations in the class diagram.
- Explain the relationship between class and objects; and between class and object diagrams.

Objetos: classificar as “coisas” do mundo

Desenvolvimento por objetos



abstração

Veiculo	
-matricula	
-cor	
-lotação	
-velocidade atual	
+travar()	
+acelerar()	

É uma estratégia para simplificar o espaço do problema, modularizando-o

Esquema mental comum à análise (requisitos) e programação

Classificar entidades similares

Facilita a reutilização de software

Orientado pelo “vocabulário” do domínio

3 mecanismos conceptuais para modelar/gerir a complexidade

Abstração



- Tratar os objetos que
são semelhantes como
um “tipo”/categoria

Encapsulamento



- O objeto é uma unidade
que esconde o seu
estado interno

- A interação com o
objeto é feita através de
“pontos de acesso”

Hierarquia



- Procurar
relações “é-um”

- Procurar
relações
“parte-de”

Mecanismo: abstração

abstração decorre do reconhecimento de semelhanças entre certos objetos, situações ou processos no mundo real, e a decisão de se concentrar nestas semelhanças → um *tipo de coisa*

Uma abstração denota as características essenciais de um objeto que o distingue de todos os outros tipos de objetos.

Abstraction: Temperature Sensor

Important Characteristics:

temperature
location

Responsibilities:

report current temperature
calibrate

Figure 2–6 Abstraction of a Temperature Sensor

Abstraction: Heater

Important Characteristics:

location
status

Responsibilities:

turn on
turn off
provide status

Related Candidate Abstractions: Heater Controller, Temperature Sensor

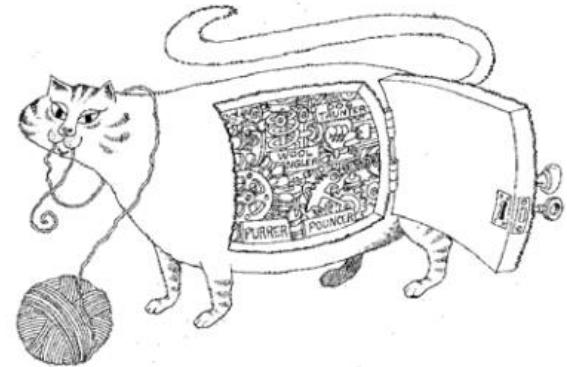
Figure 2–9 Abstraction of a Heater

Encapsulamento

Encapsulamento incentiva a ocultação da estrutura (interna) de um objeto (a informação que mantém), bem como a implementação dos seus métodos (como faz).

Nenhuma parte de um sistema complexo deve depender do conhecimento dos detalhes internos de qualquer outra parte.

O encapsulamento permite que as alterações de um programa sejam feitas de forma fiável, com repercussões limitadas (mexer num módulo não deve estragar os outros)



Encapsulation hides the details of the implementation of an object.

Como é que o encapsulamento contribui para a gerir a complexidade?

Encapsulamento ajuda a gerir a complexidade escondendo a dimensão “privada” (interna) das nossas abstrações.

Abstraction: Temperature Sensor

Important Characteristics:

temperature
location

Responsibilities:

report current temperature
calibrate

Figure 2–6 Abstraction of a Temperature Sensor

Abstraction: Heater

Important Characteristics:

location
status

Responsibilities:

turn on
turn off
provide status

Related Candidate Abstractions: Heater Controller, Temperature Sensor

O que é que Heater deve conhecer de Temperature Sensor?

Figure 2–9 Abstraction of a Heater



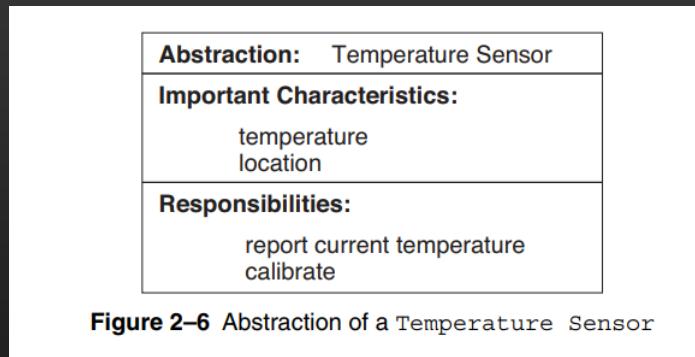
Hierarquia: a relação de herança

A herança define uma relação entre classes (tipos de coisas), em que uma classe partilha/especializa a estrutura ou o comportamento definido numa outra classe.

A herança denota uma relação semântica "é-um", e.g.:

um urso "é um" (tipo de) mamífero;
uma casa "é um" tipo de ativo tangível; uma caravana é um veículo.

A herança implica, assim, uma generalização ↔ especialização



Qual a relação semântica?
Um Sensor de temperatura é um (*is-a*) Sensor.

Hierarquia: a relação de agregação

A agregação define uma relação entre categorias de coisas (i.e. classes) do tipo “parte-de”

A agregação denota a relação semântica “parte-de”, e.g.:

Um País é parte de um Continente; o Estudante é parte da Turma; uma Obra é parte de uma Coleção.

A herança implica, assim, um agregador ←→ parte-de

Abstraction: Heater
Important Characteristics: location status
Responsibilities: turn on turn off provide status

Related Candidate Abstractions: Heater Controller, Temperature Sensor

Figure 2–9 Abstraction of a Heater

Abstraction: Temperature Sensor
Important Characteristics: temperature location
Responsibilities: report current temperature calibrate

Figure 2–6 Abstraction of a Temperature Sensor

O Temperature Sensor é parte de um Heater.

Um Estudante é parte de uma Turma.

3 mecanismos conceptuais para modelar/gerir a complexidade

Abstração



- Tratar os objetos que
são semelhantes como
um “tipo”/categoria

Encapsulamento



- O objeto é uma unidade
que esconde o seu
estado interno

- A interação com o
objeto é feita através de
“pontos de acesso”

Hierarquia



- Procurar
relações “é-um”

- Procurar
relações
“parte-de”

Classes and objects

In object-oriented modeling

Object-oriented development

It is a strategy to simplify the problem space by working with smaller, "real-world-like" entities

The same mental constructions for analysis (requirements) and coding

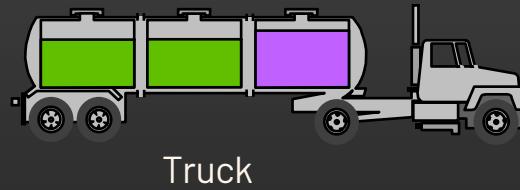
Classify similar entities

Facilitates software reuse

Objects (in OO) model real-world/problem space entities

Observable in the physical world

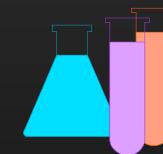
e.g.: Student, airplane



Truck

Concepts

e.g.: sale, booking



Chemical Process

Software abstractions

e.g.: LinkedList, Vector



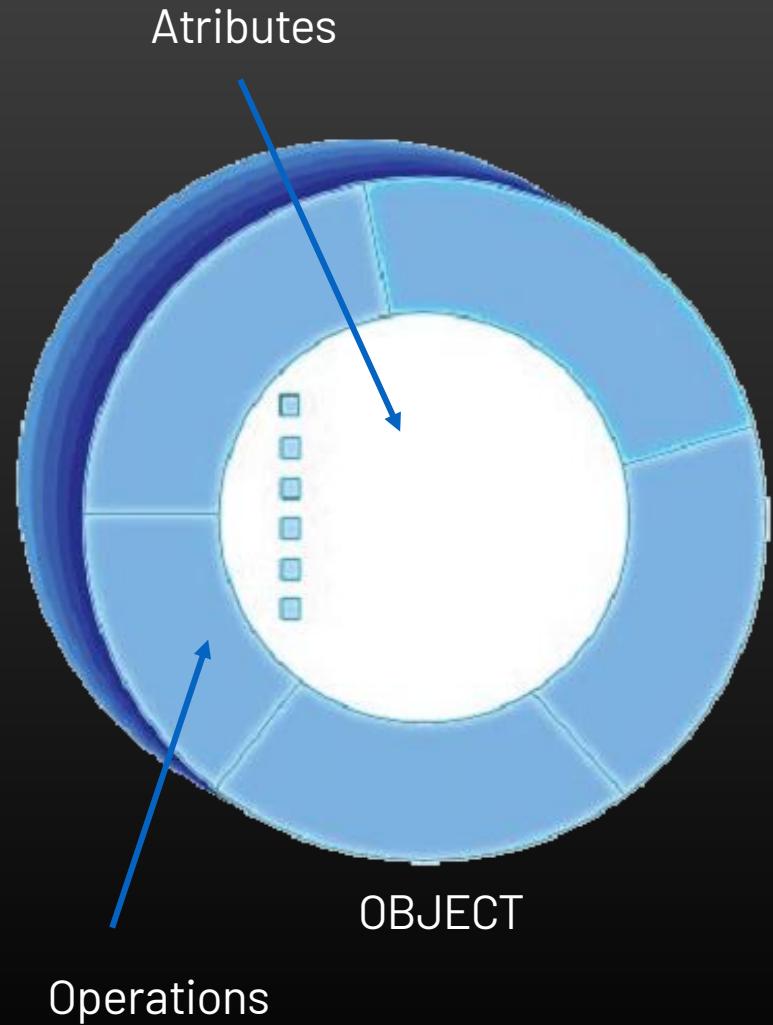
Object = purpose + state + behavior

An object has a well-defined purpose

It is an entity with a boundary that encapsulates state and behavior.

The state is represented through attributes and relationships.

The behaviorally is represented through operations.



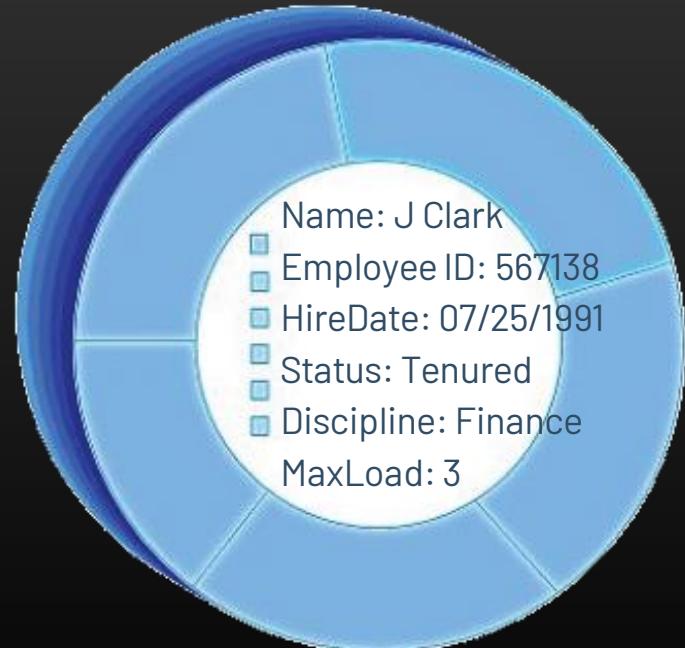
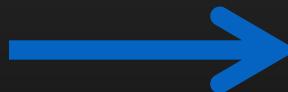
An object has a (internal) state

The state of an object matches one of the conditions/settings is that it is possible for the object to present itself.

It is normal for the object state to change over time.



Name: J Clark
Employee ID: 567138
Date Hired: July 25, 1991
Status: Tenured
Discipline: Finance
Maximum Course Load: 3 classes

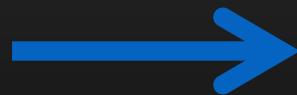


Professor Clark

An object has behavior (functionality)

The behavior defines how the object acts/reacts

The visible/exposed behavior is modeled by the set of messages that it responds to (operations)



Professor Clark

Professor Clark's behavior
Submit Final Grades
Accept Course Offering
Take Sabbatical
Maximum Course Load: 3 classes

Object: State + operations

You know something

Its attributes

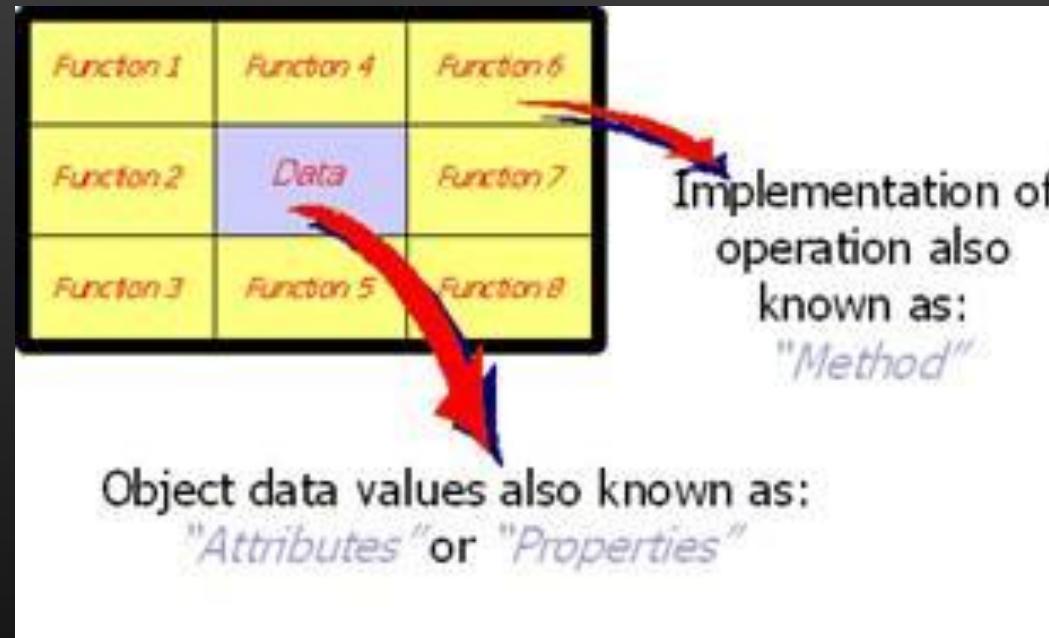
Your relationships

Do you know how to do something

Its behaviour, activated through operations

Capsule

The others (objects) do not need to know what he knows...



An object has its own identity

Each object has its own unique identity, even if its state is equal to that of another object.



Professor "J Clark"
teaches Biology



Professor "J Clark"
teaches Biology

What's a class?

It is a category of similar objects, which share the same attributes, operations, relationships, and semantics.

The object is an instance (occurrence) of a class

A class is an abstraction

Categorizes similar objects

Emphasizes the characteristics of interest (and suppresses others)



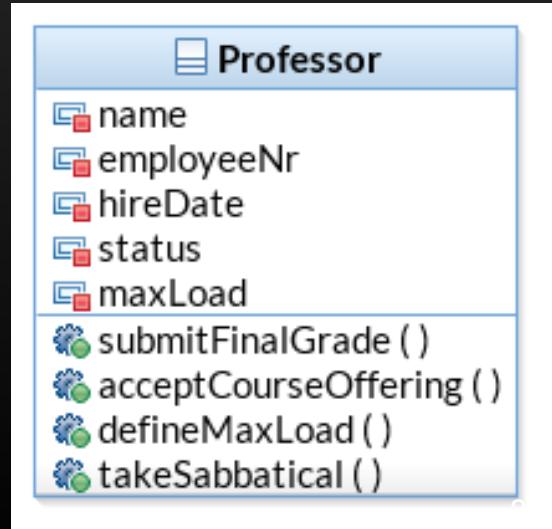
Professor Torpie



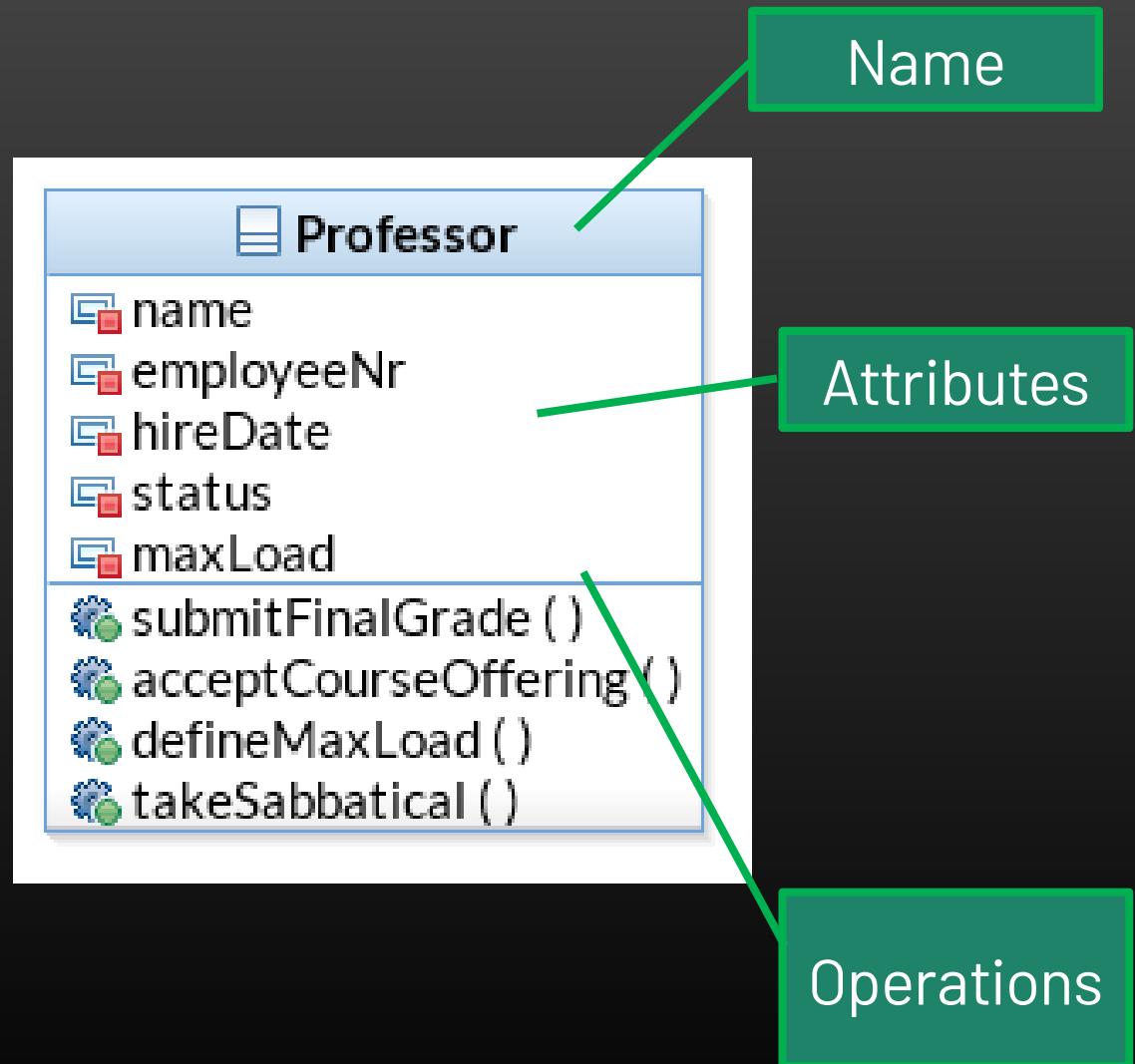
Professor Meijer



Professor Allen



UML class representation



The relationship between classes and objects



A class is an abstract definition of an object

Defines the structure and behavior of each object in that class/category.

Works as a template for creating objects.

Classes are not collections of objects

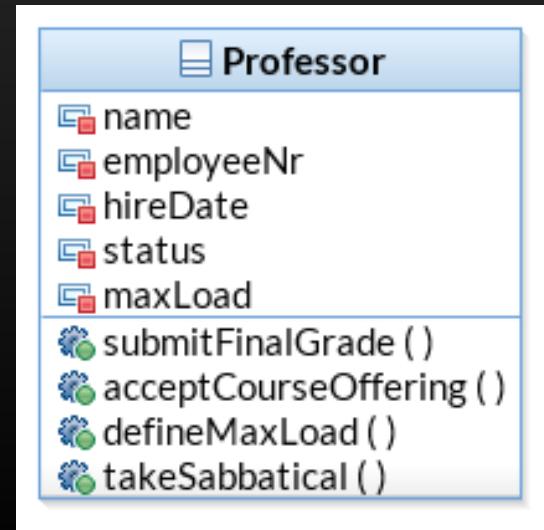


Professor Meijer

Professor Torpie



Professor Allen

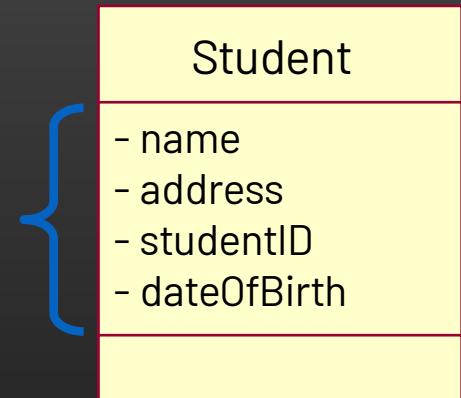


What is an attribute?

Is a property of a class
that describes the sort
of values that instances
can hold

A class can have multiple
or no attributes

Attributos



What's an operation?

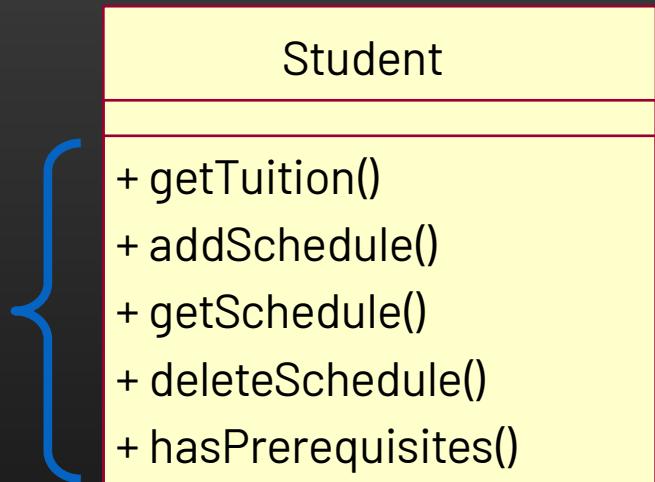
Is the implementation of a service/action that can be requested from any object in a class

What the class can do

The class can have multiple or no operations

Common: commands and interrogations (lookups)

Operações



Arrivées

13:02

Compagnie	Vol	Provenance	Heure	Hall	Remarque
AIR ALGERIE	AH 5017	Ouagadougou	05:40	2	Retardé
QATAR	QR 1379	Doha	12:05	1	Arrivé 12:05
IBERIA	IB 3308	Madrid	12:15	2	Arrivé 12:25
AIR ALGERIE	AH 1023	Marseille	12:25	2	Arrivé 12:50
AIGLE AZUR	ZI 707	Marseille	13:05	2	Retardé 13:30
AIR ALGERIE	AH 1037	Lyon	13:10	2	Retardé 13:20
Emirates	EK 757	Dubai	13:10	1	Atterri 13:00
AIGLE AZUR	ZI 223	Paris-Orly	13:10	1	Retardé 13:20
AIR ALGERIE	AH 1003	Paris-CDG	13:15	2	Retardé 14:40

Os objetos de um domínio estão ligados numa “rede de conhecimento”

Um Projeto é coordenado por um determinado Departamento.

O Projeto tem uma equipa de vários Funcionário(s) atribuída.

Cada Projeto define várias Tarefas que, por sua vez, podem estar organizadas em sub-tarefas.

etc.



Como transportar o conhecimento do domínio pra um modelo?
Há “coisas” de interesse e “relacionamentos” entre elas

O que é a “associação”?

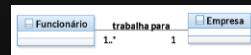
A relação semântica que se estabelece entre duas ou mais classes que descreve as ligações existentes entre as respetivas instâncias.

Mostra que objetos de um tipo estão ligados a objetos de outro tipo.

O tipo de “ligação” deve ser anotado com uma explicação do significado.

A classe Funcionário e Empresa estão associadas. A descrição da associação é “trabalha para”.

Portanto:
“Funcionário” → “trabalha em” → “Empresa”.



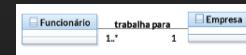
Com que “frequência” os objetos de F e E estão associados?



O que é a multiplicidade (de uma associação)?

Nr de instâncias de uma classe que se relacionam com uma instância da outra.

“O Funcionário só trabalha numa empresa; a Empresa tem pelo menos um Funcionário”



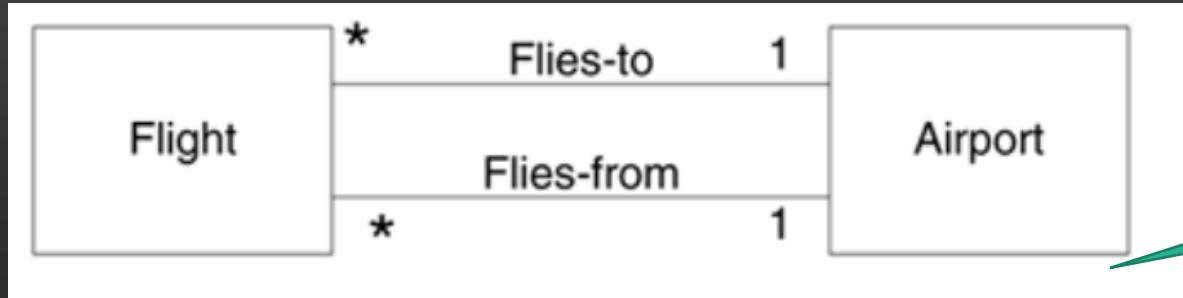
“O Funcionário só trabalha numa empresa; a Empresa só tem um Funcionário”



Indicação de multiplicidade

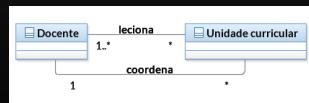
Exactly one	1	<pre> classDiagram class Department class Boss Department "1" --> "1" Boss </pre>	A department has one and only one boss.
Zero or more	0..*	<pre> classDiagram class Employee class Child Employee "0..*" --> "*" Child </pre>	An employee has zero to many children.
One or more	1..*	<pre> classDiagram class Boss class Employee Boss "1..*" --> "1..*" Employee </pre>	A boss is responsible for one or more employees.
Zero or one	0..1	<pre> classDiagram class Employee class Spouse Employee "0..1" --> "0..1" Spouse </pre>	An employee can be married to zero or one spouse.
Specified range	2..4	<pre> classDiagram class Employee class Vacation Employee "2..4" --> "2..4" Vacation </pre>	An employee can take from two to four vacations each year.
Multiple, disjoint ranges	1..3,5	<pre> classDiagram class Employee class Committee Employee "1..3,5" --> "1..3,5" Committee </pre>	An employee is a member of one to three or five committees.

Múltiplas associações entre duas classes



Um voo: parte de um Aeroporto; chega a um Aeroporto.
O conhecimento sobre um Voo associa 2 objetos Aeroporto, por razões diferentes

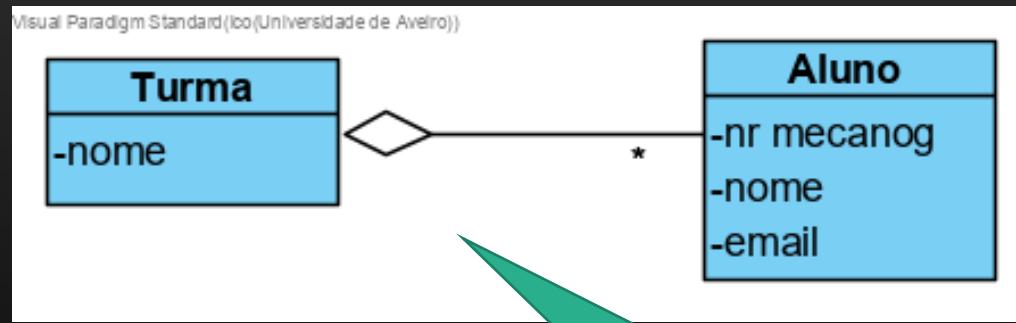
Um docente: leciona várias UC; coordena várias UC.



O que é uma agregação?

É uma forma especial de associação que modela uma relação de todo-parte, entre o agregador (“contentor”) e as suas partes constituintes

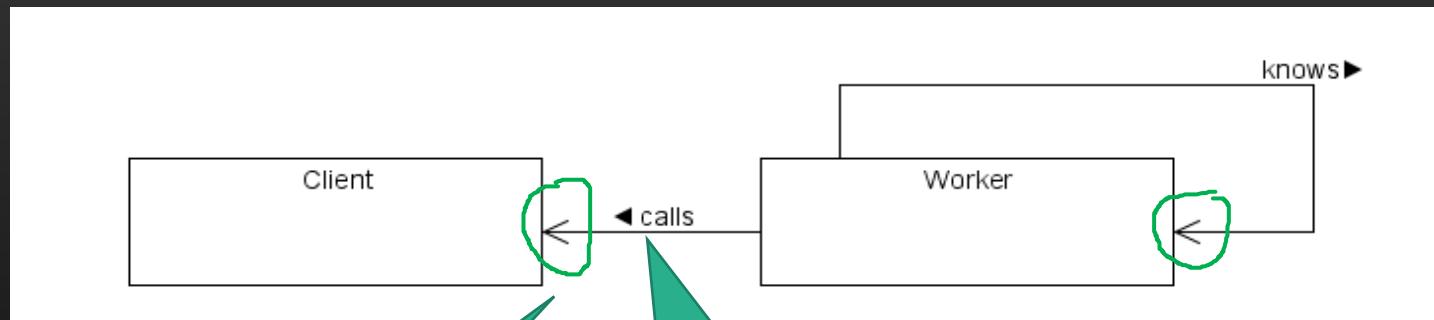
Deve ser natural ler-se “É parte de...” (sentido parte → todo); se for “forçado”, usar a associação normal



Um objeto *Turma* agrupa
vários objetos *Aluno*.

O que é a navegabilidade?

Indica a possibilidade de navegar de uma classe de partida para uma classe de chegada, usando a associação



Em que sentido é que a associação é navegável/aplicável?
(Se omissos → ambos)

Notação auxiliar: em que sentido se deve ler a etiqueta. (Worker > calls > Client.
(Se omissos: cima p/ baixo; esquerda p/ direita)

Os dois primeiros casos são as situações mais comuns.



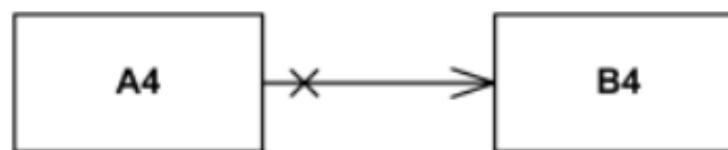
Both ends of association have unspecified navigability.



A2 has unspecified navigability while B2 is navigable from A2.



A3 is not navigable from B3 while B3 has unspecified navigability.



A4 is not navigable from B4 while B4 is navigable from A4.

O que é a generalização (=herança)

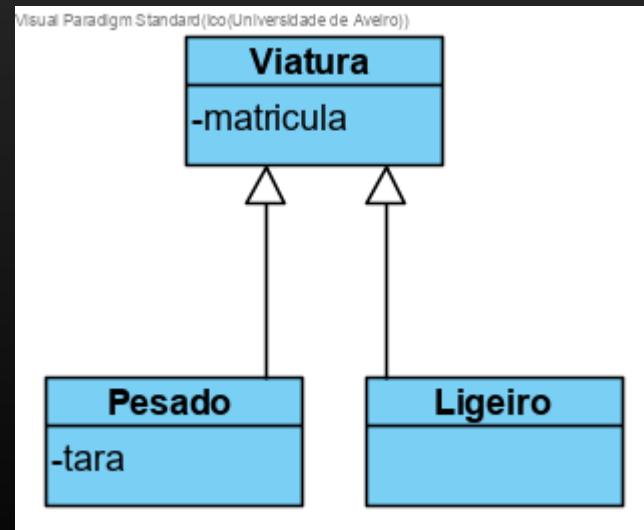
É relação entre classes em que uma especializa a estrutura e/ou comportamento de outra, partilhando todas as características

Define uma hierarquia em que a subclass herda das características da superclasse

A subclass pode sempre ser usada onde a superclasse é usada, mas não ao contrário.

Pode ler-se “é um tipo de”

(Um *Pesado* é um tipo de *Viatura*, com matrícula e tara.)



O que é passado à subclasse?

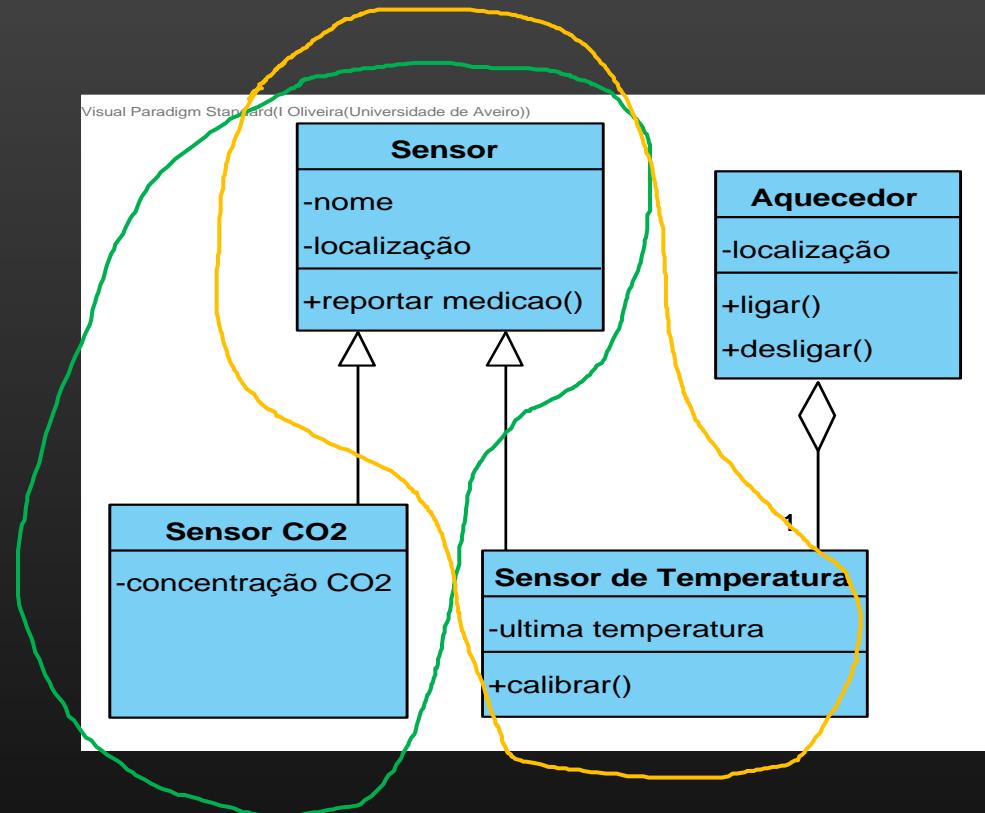
A subclasse herda os atributos, operações e relacionamentos da superclasse

A subclasse pode:

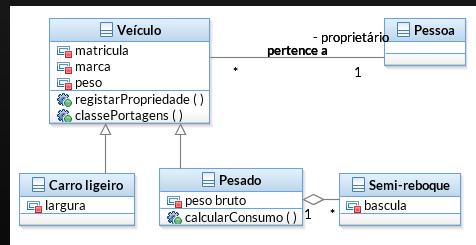
Adicionar mais atributos, operações e relacionamentos à base herdada

Redefinir as operações da superclasse

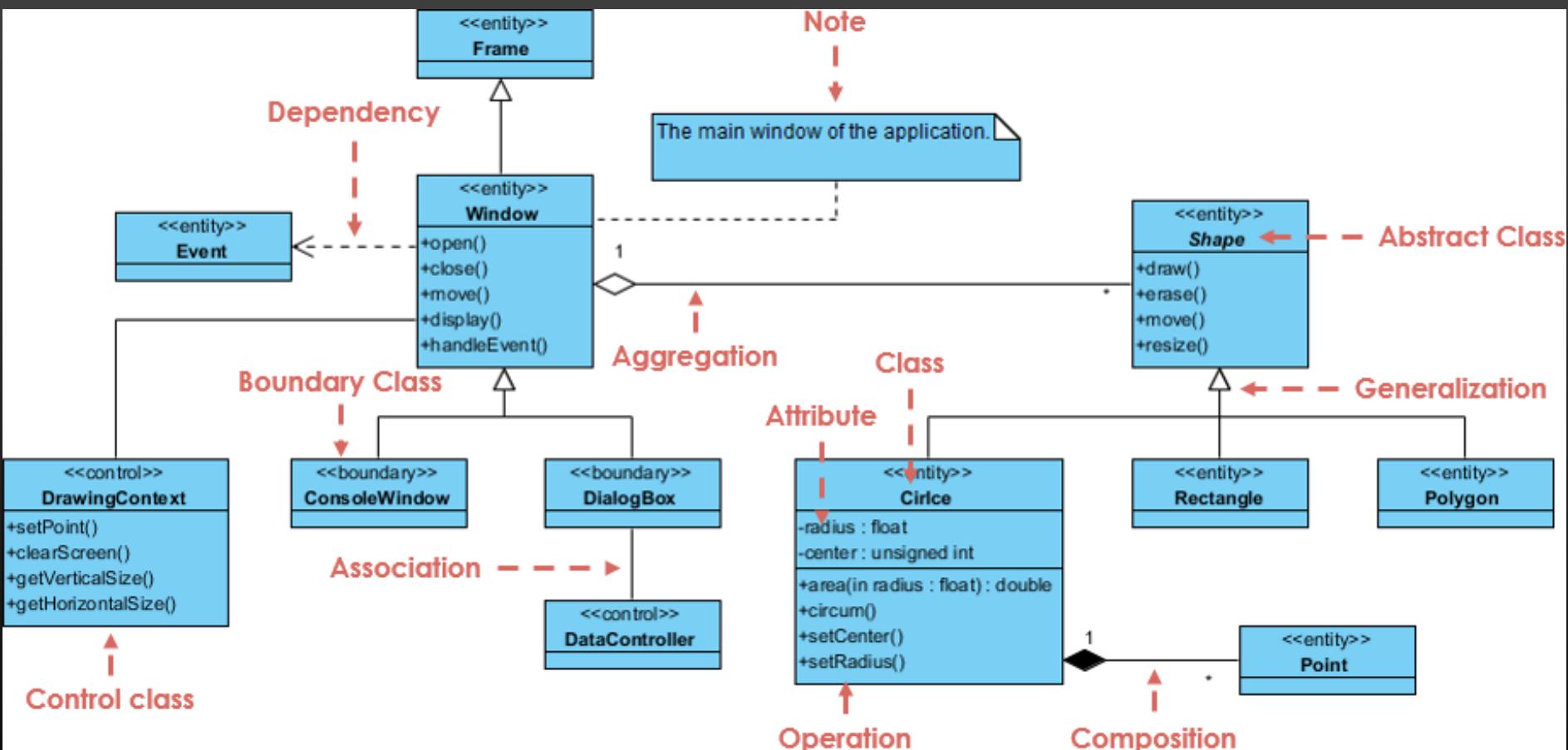
A herança põe em evidencia as características comuns entre classes



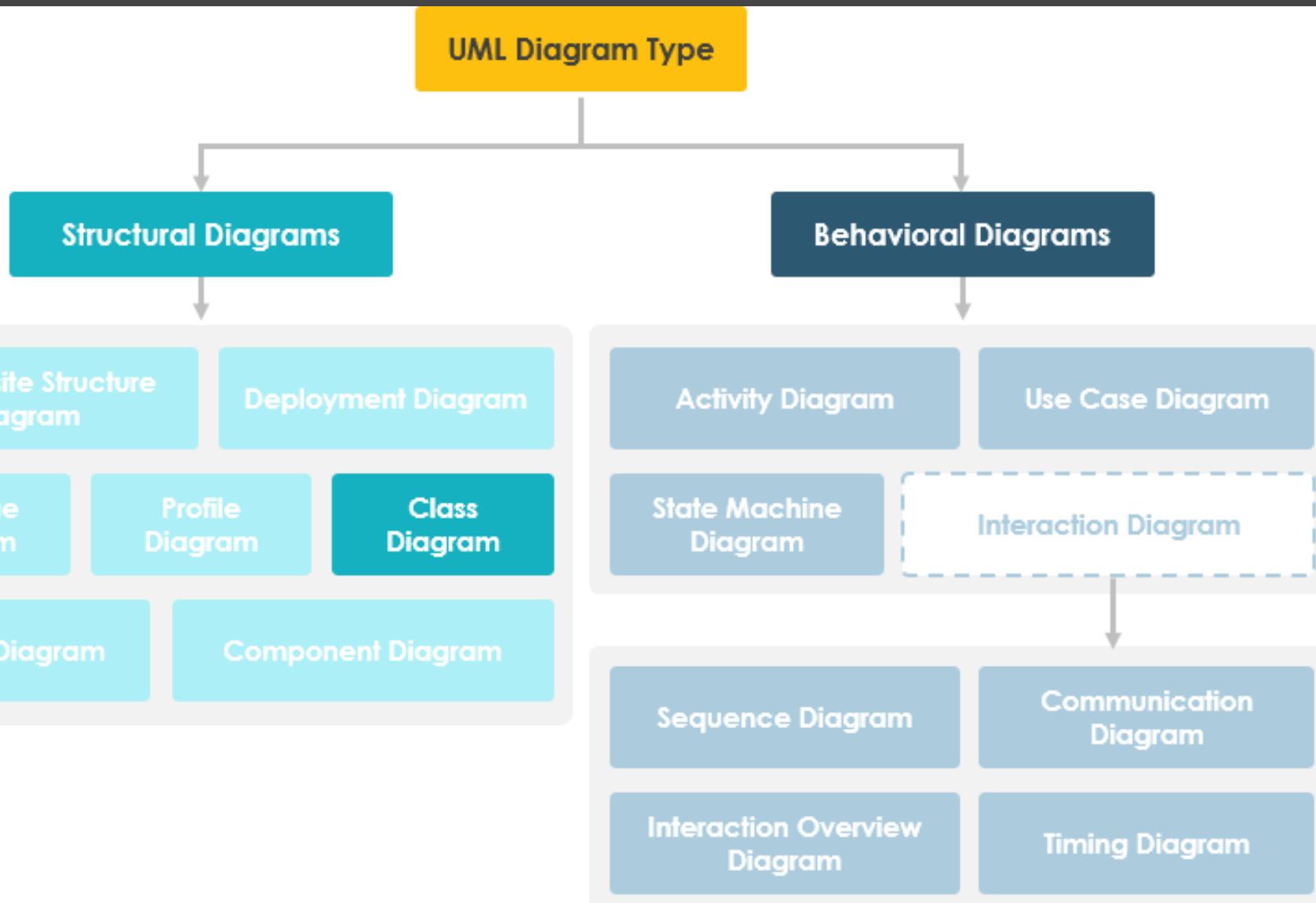
What is "inherited"?



Class Diagram notation overview



<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/>



Exercício: domínio da gestão de projetos

Object-oriented “decomposition”

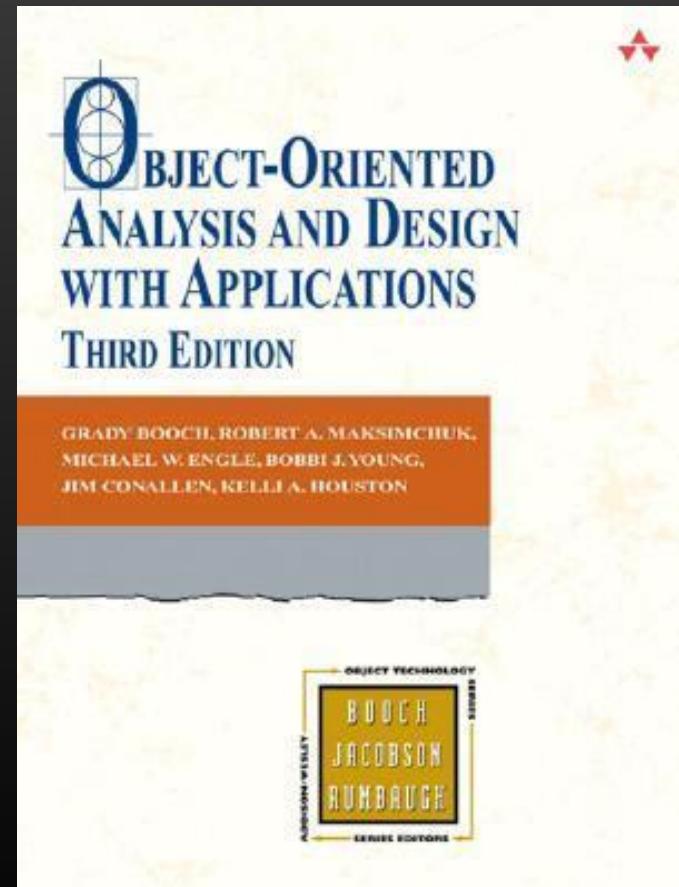
“Although both designs solve the same problem, they do so in quite different ways.

In this second decomposition, we view the world as a set of autonomous entities that collaborate to perform some higher-level behavior. *Get formatted update* thus does not exist as an independent algorithm; rather, it is an **operation associated with the object File of Updates**.

Calling this operation creates another object, *Update to Card*. In this manner, **each object in our solution embodies its own unique behavior, and each one models some object in the real world**.

Objects do things, and we **ask them to perform what they do by sending them messages**.

Because our decomposition is based upon objects and not algorithms, we call this an *object-oriented decomposition*.



Readings & references

Core readings	Suggested readings
<ul style="list-style-type: none">• [Dennis15] - Chap. 3	<ul style="list-style-type: none">• [Larman'12]- Chap. 5• [Pressman'10] – Chap. 5

47006- ANÁLISE E MODELAÇÃO DE SISTEMAS

Domain model and additional Class diagrams notation

Ilídio Oliveira

v2020/11/11, TP11

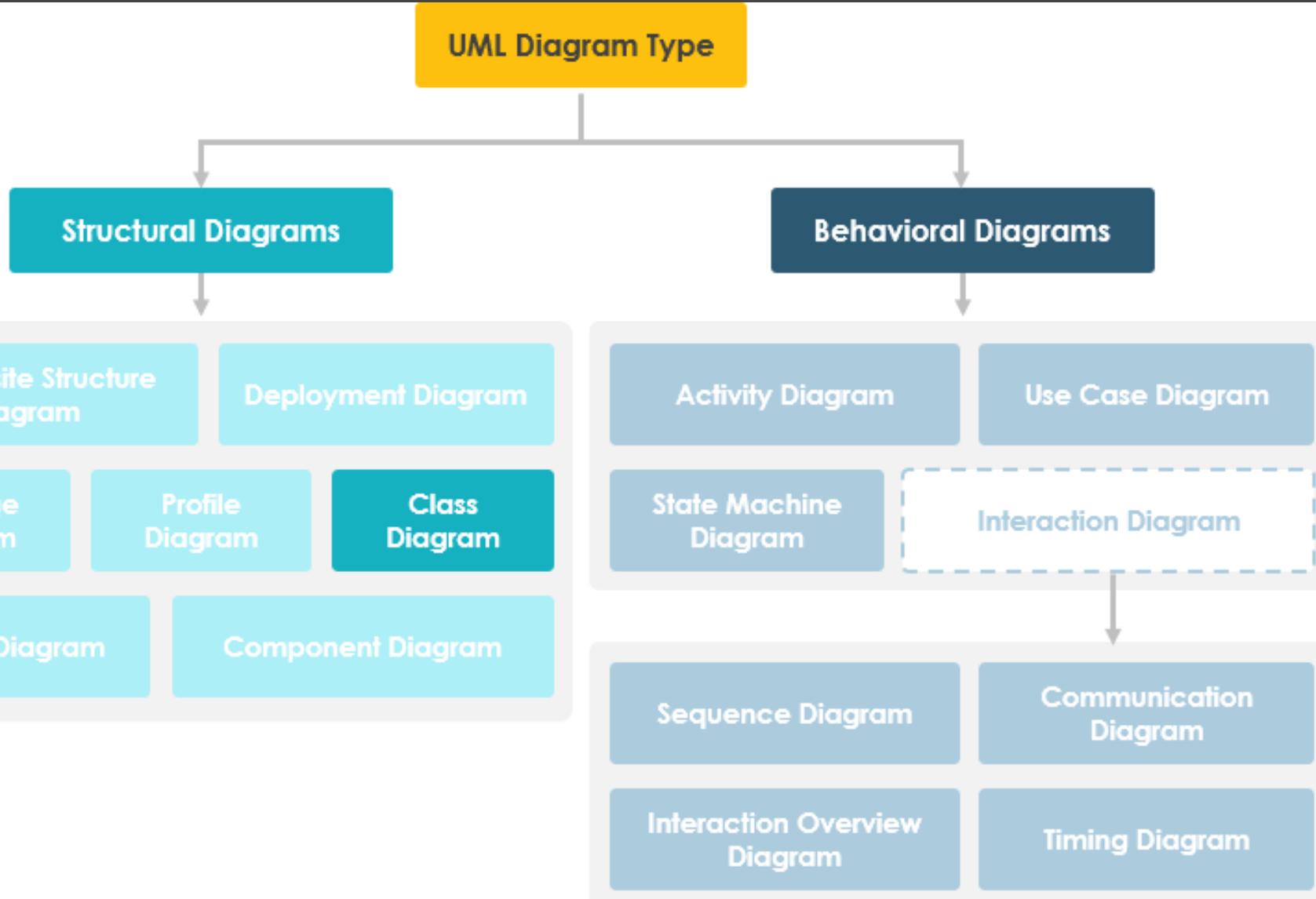
Learning objectives for this lecture

Draw a simple class diagram to capture the concepts of a problem domain.

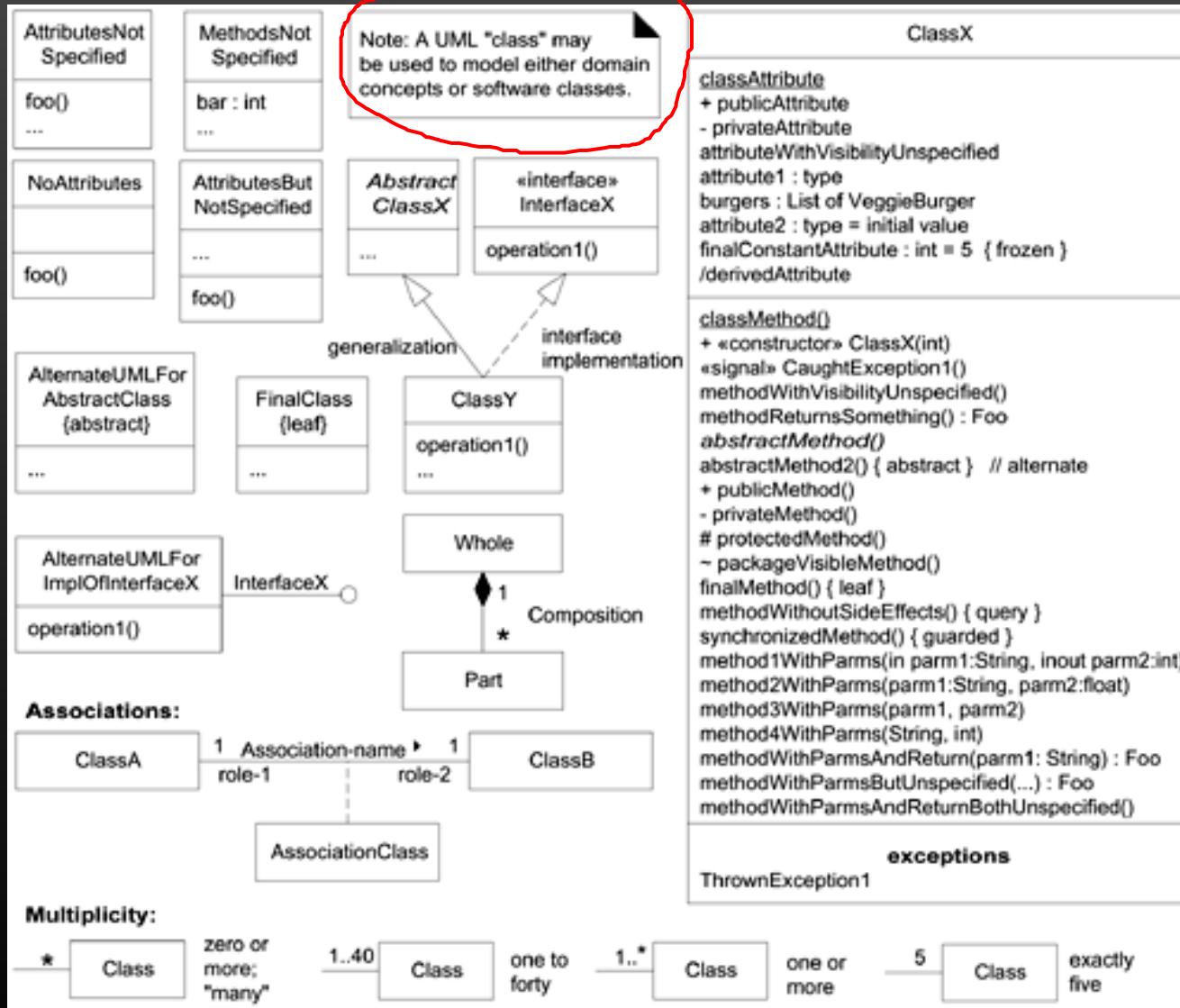
Describe the types and roles of the different associations in the class diagram.

Critically review existing models to capture the domain concepts.

Spot implementation-specific constructs that may pollute the domain model.



Use of the UML classes diagram



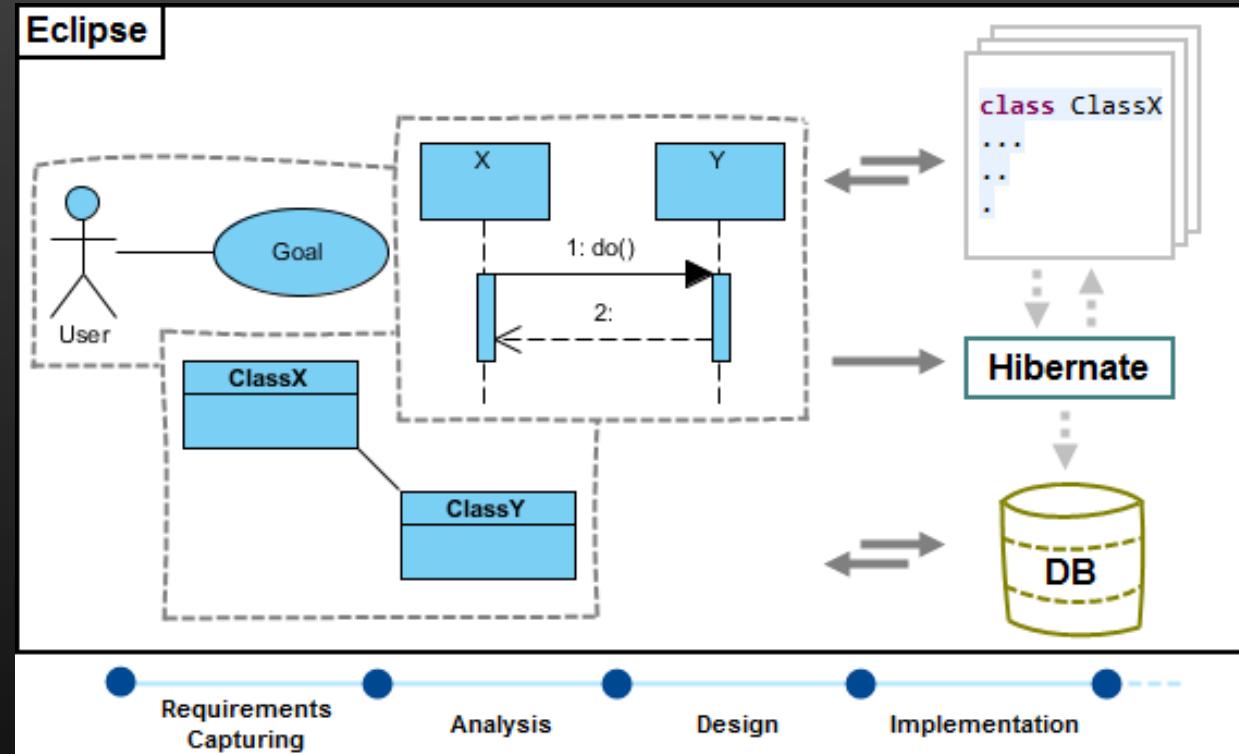
Analysis

→ Domain concepts

Design and implementation

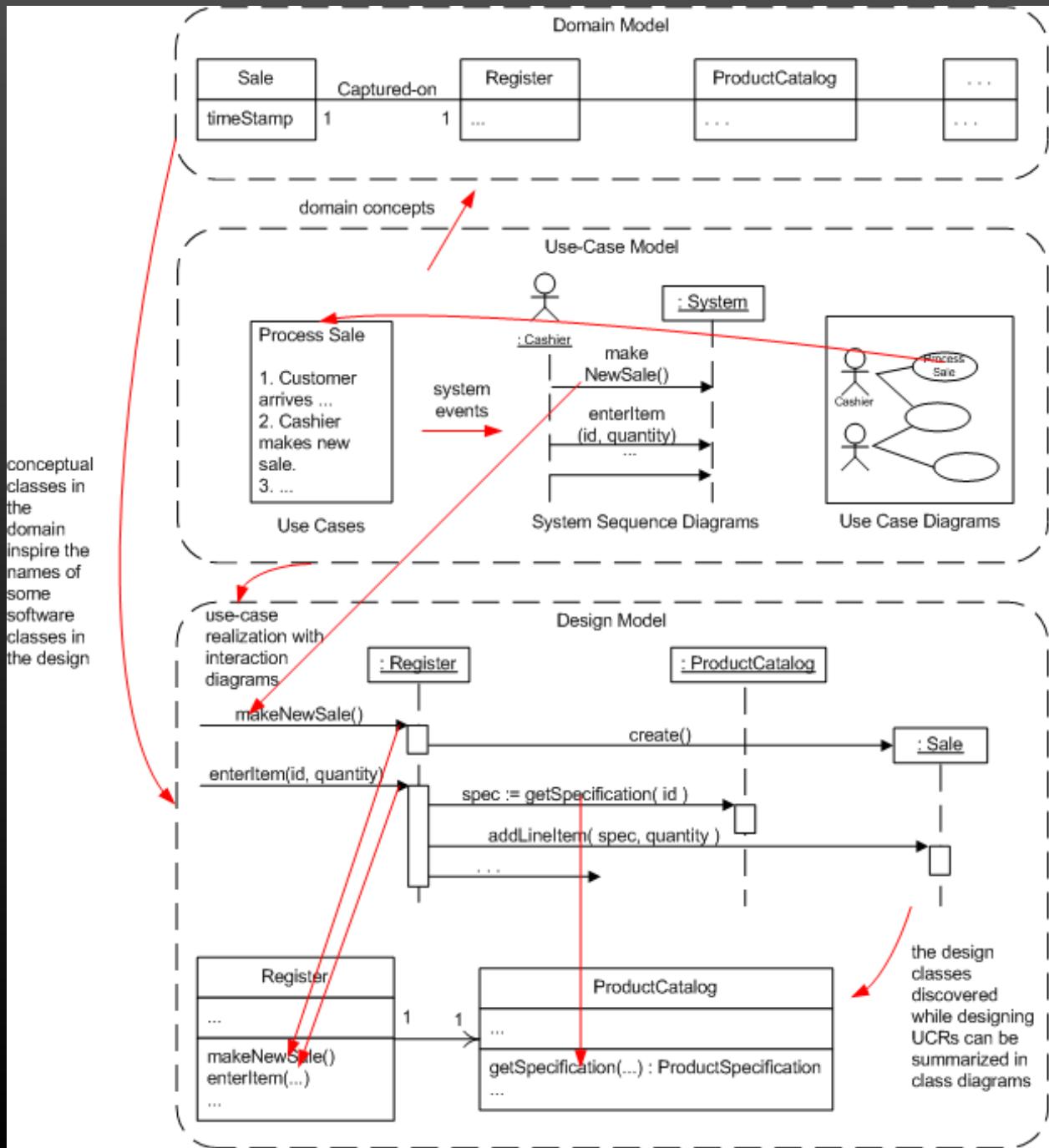
→ Programming entities (classes, enumerations,...)

Classes are used in the domain model and in the design model



Classes are used in the domain model and in the design model

conceptual classes in the domain inspire the names of some software classes in the design



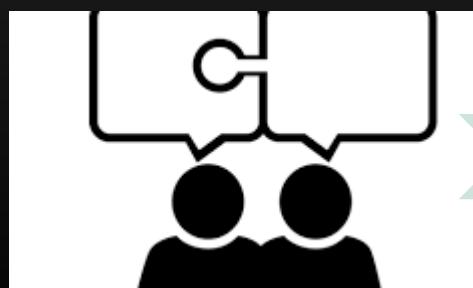
O modelo do domínio não tem a implementação

Classes de análise: perspetiva conceitual

Resultado da análise dos requisitos (**Analista**)

Neutro em relação à implementação

Não fornece diretamente o modelo de dados nem as classes de programação



Análise

Especificação

Classes em Java: perspetiva de implementação

Resultado do desenho/implementação (**Programador**)

Escritas numa linguagem concreta (OO)

Assunto de P-3 (e, em parte, de MAS)



Implementação

Domain model: focus on capturing the information and rules of the problem

O modelo do domínio é um *mapa dos conceitos* de uma área de aplicação

Mostra os conceitos
de um problema

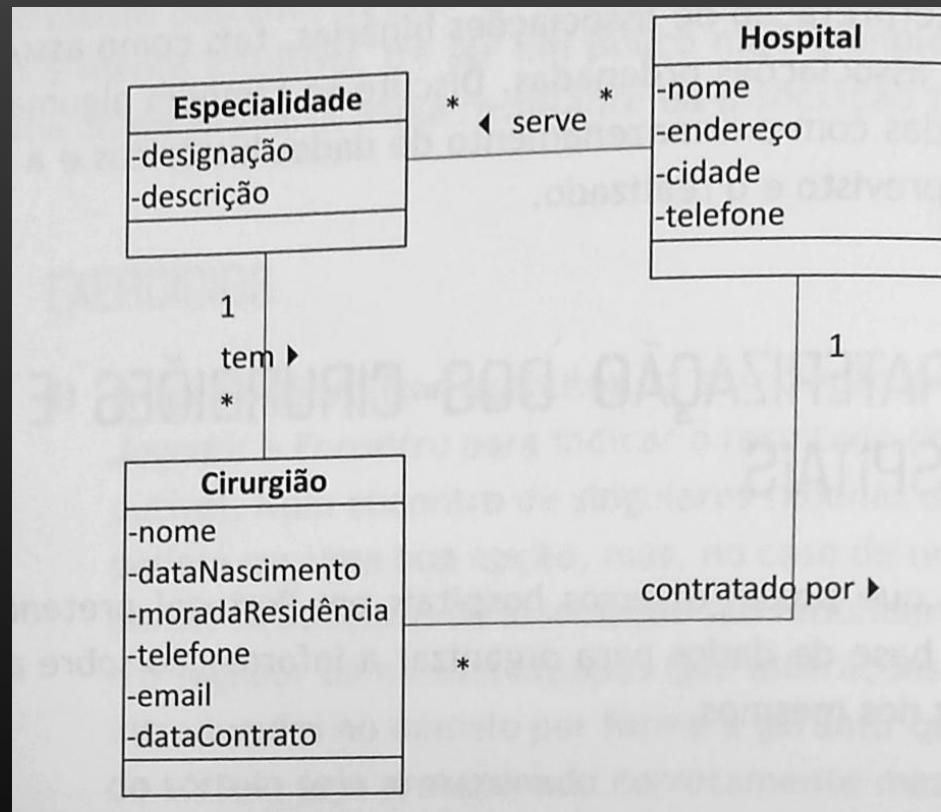
O vocabulário de uma área
de aplicação

Não é software!

É um resultado da análise,
na investigação do
problema.

É visualizado num
diagrama de classes

É uma vista estrutural.



Como encontrar as classes?

Procurar numa lista de situações comuns

→ categorias de classes

Análise documental

Explorar documentos/relatórios existentes na área do problema

Análise de nomes

→ explorar descrições do problema à procurar dos substantivos

Situações comuns para encontrar classes candidatas

Conceptual Class Category	Examples
product or service related to a transaction or transaction line item <i>Guideline:</i> Transactions are for something (a product or service). Consider these next.	<i>Item</i> <i>Flight, Seat, Meal</i>
where is the transaction recorded? <i>Guideline:</i> Important.	<i>Register, Ledger</i> <i>FlightManifest</i>
roles of people or organizations related to the transaction; actors in the use case <i>Guideline:</i> We usually need to know about the parties involved in a transaction.	<i>Cashier, Customer, Store</i> <i>MonopolyPlayer Passenger, Airline</i>
place of transaction; place of service	<i>Store</i> <i>Airport, Plane, Seat</i>
noteworthy events, often with a time or place we need to remember	<i>Sale, Payment</i> <i>MonopolyGame</i> <i>Flight</i>
physical objects <i>Guideline:</i> This is especially relevant when creating device-control software, or simulations.	<i>Die</i> [Larman 2004], Cap. 9.

Análise de nomes num texto

Nomes podem revelar conceitos (classes) ou atributos

Nem todos os nomes têm consequências diretas no modelo...

Os projetos têm uma duração variável, com data de início e de fim bem definidas.

Cada projeto é organizado em várias tarefas (ou atividades), com uma duração bem definida.

Um projeto vai ter uma equipa de funcionários atribuída.

Nem todos os funcionários participam em todas as tarefas, mas é importante saber que trabalhou numa tarefa.

O projeto tem um *Project Leader* atribuído (que é um funcionário)

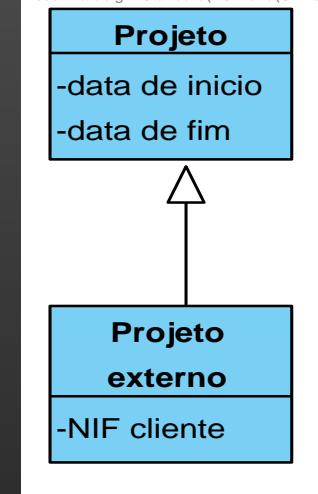
Os projetos externos (ao contrário dos projetos internos), têm um cliente. O NIF do contribuinte é sempre necessário.

Domínio: clínica veterinária

- A clínica veterinária Faísca faz consultas a vários tipos de animais domésticos, e está em expansão na região de Aveiro.
- Em cada consulta só é visto um animal, por um veterinário da clínica.
- A clínica mantém a informação do dono do animal, incluindo nome, morada, telefone e NIF.
- Cada animal é tratado com elevados padrões de higiene, e tem uma ficha própria, em que se caracteriza o nome, género, espécie e data de nascimento.
- Os animais podem ser vacinados, de acordo com as vacinas indicadas para essa espécie.
- O Sr. Joaquim tem três cães, Tejo, Tamisa, Danúbio, que são pastores-alemães.

- A clínica veterinária Faísca faz consultas a vários tipos de animais domésticos, e está em expansão na região de Aveiro.
- Em cada consulta só é visto um animal, por um veterinário da clínica.
- A clínica mantém a informação do dono do animal, incluindo nome, morada, telefone e NIF.
- Cada animal é tratado com elevados padrões de higiene, e tem uma ficha própria, em que se caracteriza o nome, género, espécie e data de nascimento.
- Os animais podem ser vacinados, de acordo com as vacinas indicadas para essa espécie.
- O Sr. Joaquim tem três cães, Tejo, Tamisa, Danúbio, que são pastores-alemães.

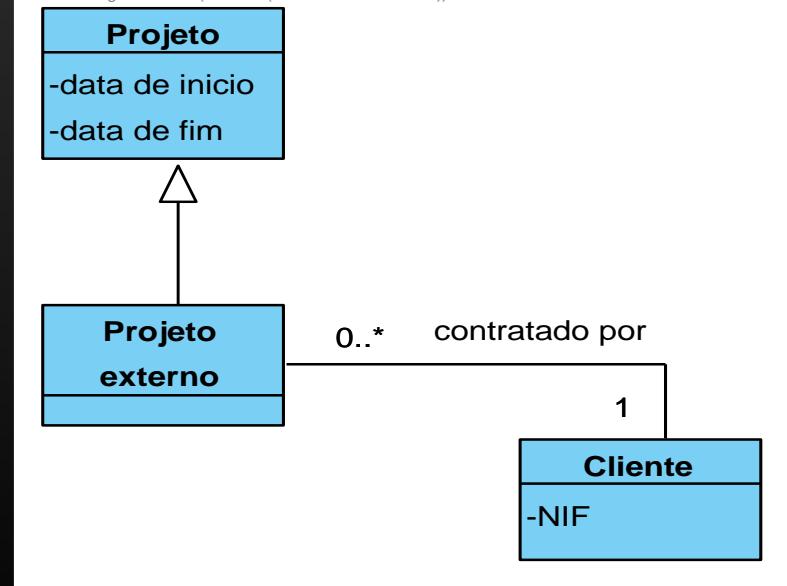
Dá origem a: Classe Atributo A ponderar



Classe ou atributo?

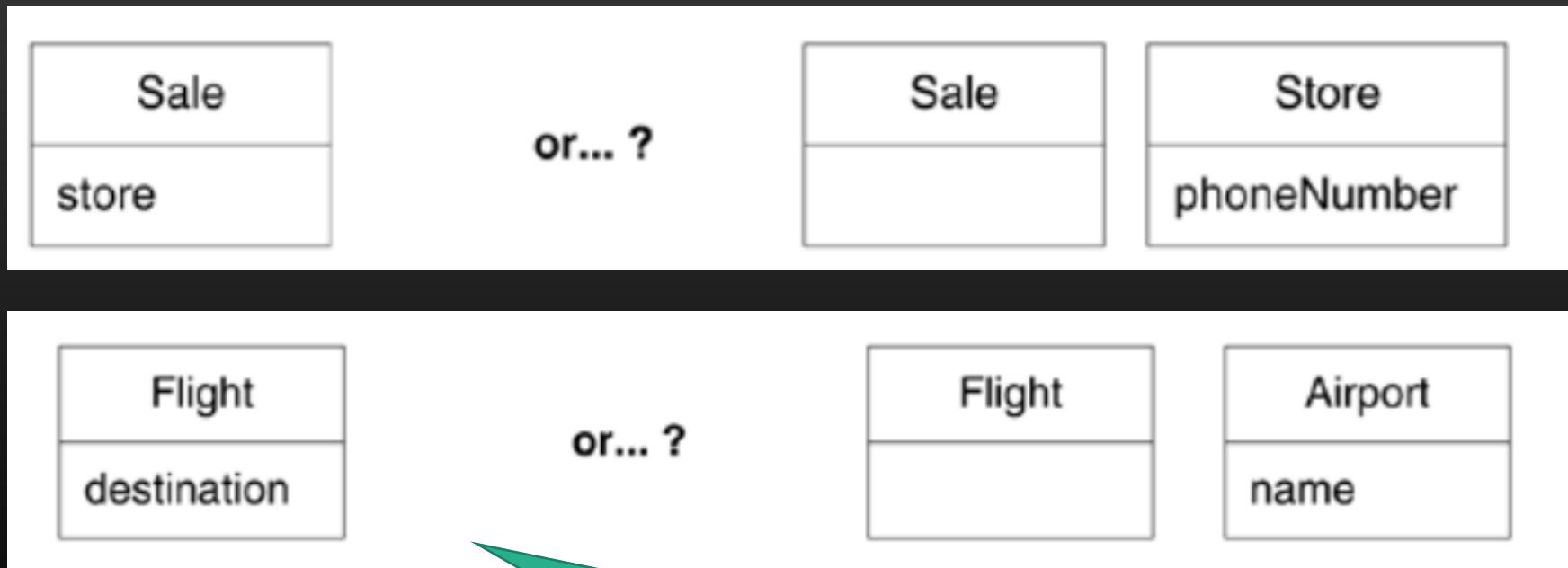
É natural pensar num dado atributo como um “tipo primitivo” ou como um conceito (tipo de coisas)?

“tipos primitivos”: números, datas, uma String,...



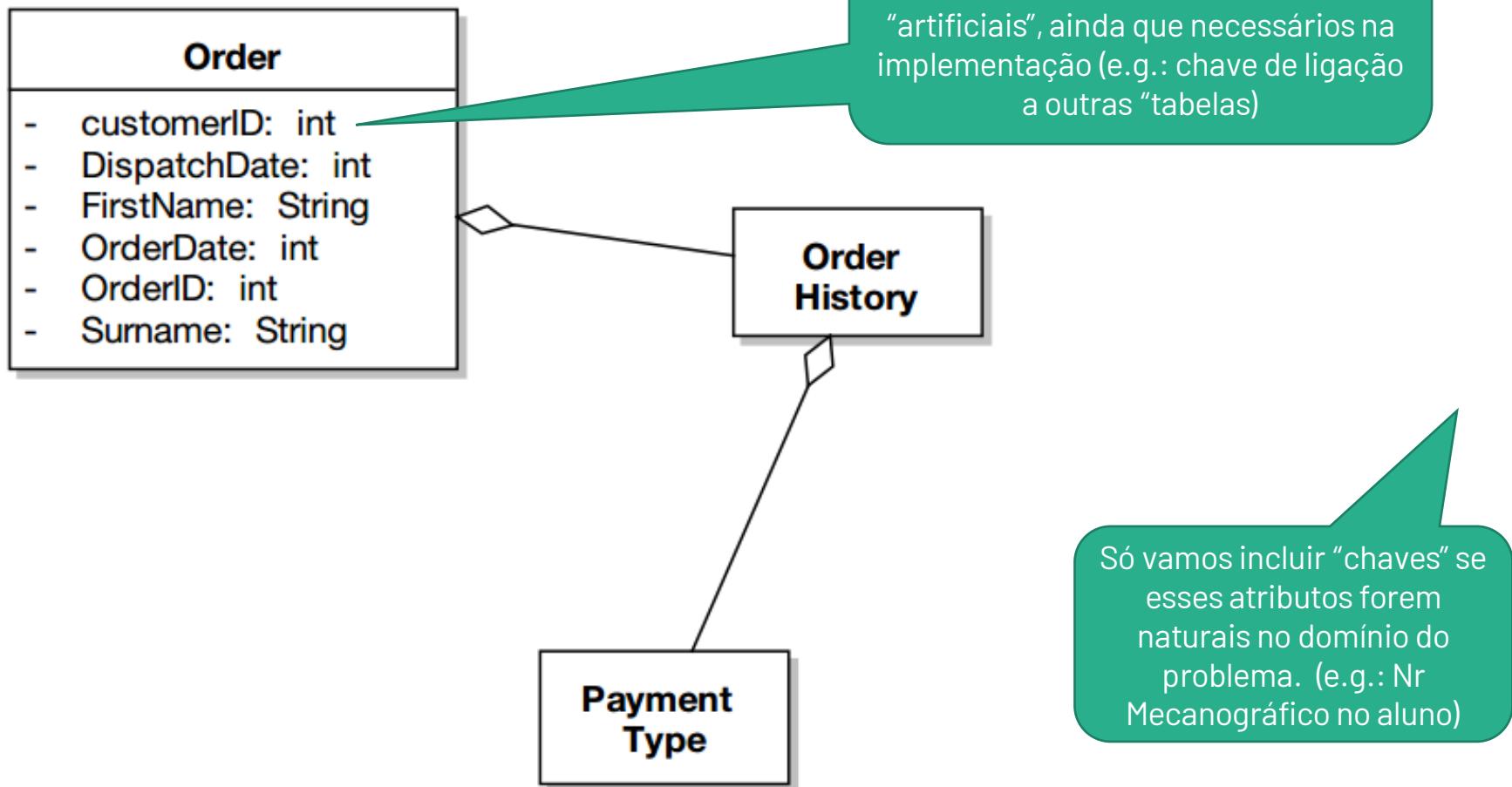
Conceito ou atributo?

Se não é natural pensar em X como um número ou texto (um tipo “primitivo”), então X provavelmente é uma classe e não um atributo



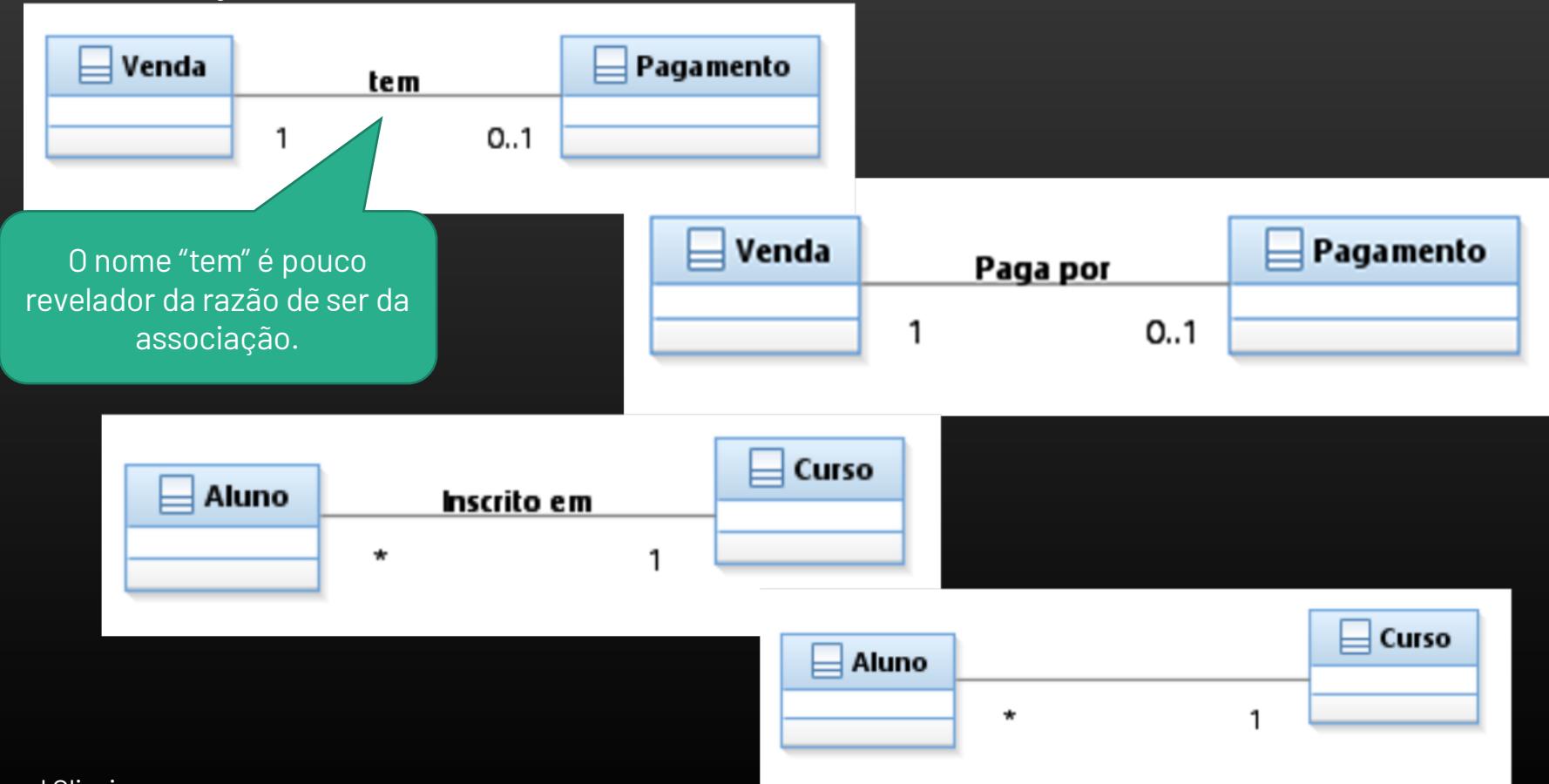
O destino de um voo deve ser um atributo próprio ou resultar de uma associação (com Aeroporto)? **Na dúvida, preferir a associação.**

Figure 2-9 shows a domain model diagram with attributes on the Order class. What database-related problem does the diagram suggest?



Nomear as associações com Classe → Expressão verbal → Classe

A sequência deve ser legível e revelar o significado da associação



Boas práticas

Uma classe representa um tipo de coisas. O nome é no singular.

O nome da classe é um substantivo (representa um conceito, não uma ação).



Roles ≠ name of the association

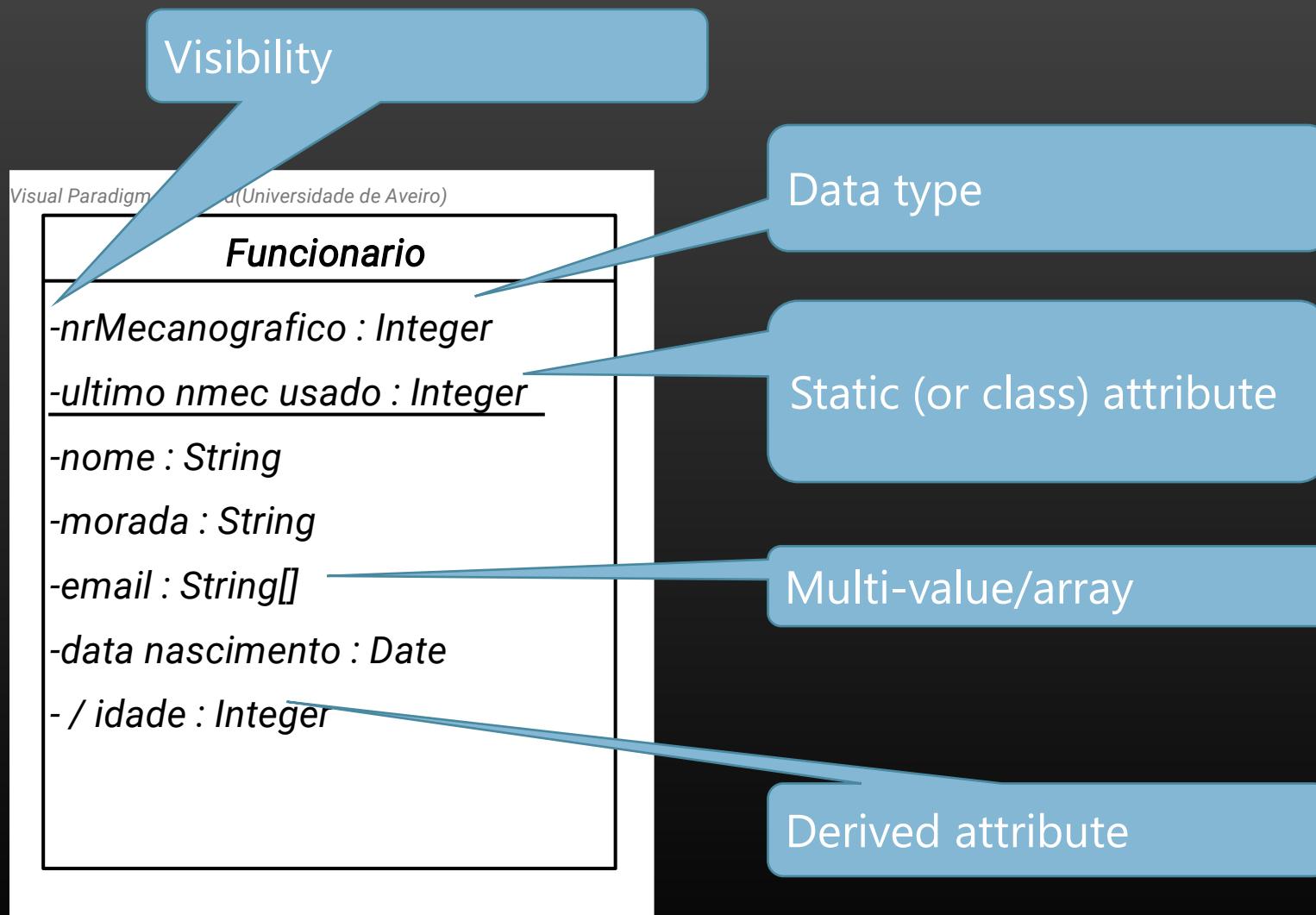
The name should clarify the association semantics

The role characterizes the way instances of a type participate in the association.



Clas diagrams: more syntax

Characterization of attributes



Class attributes are shared by all instances

Funcionário

Ultimo nmec utilizador = 161

f1: Funcionario

Nmec=145

Nome='João Joaquim'

...

Idade=35

f2: Funcionario

Nmec=15

Nome='António Joaquim'

...

Idade=25

f3: Funcionario

Nmec=40

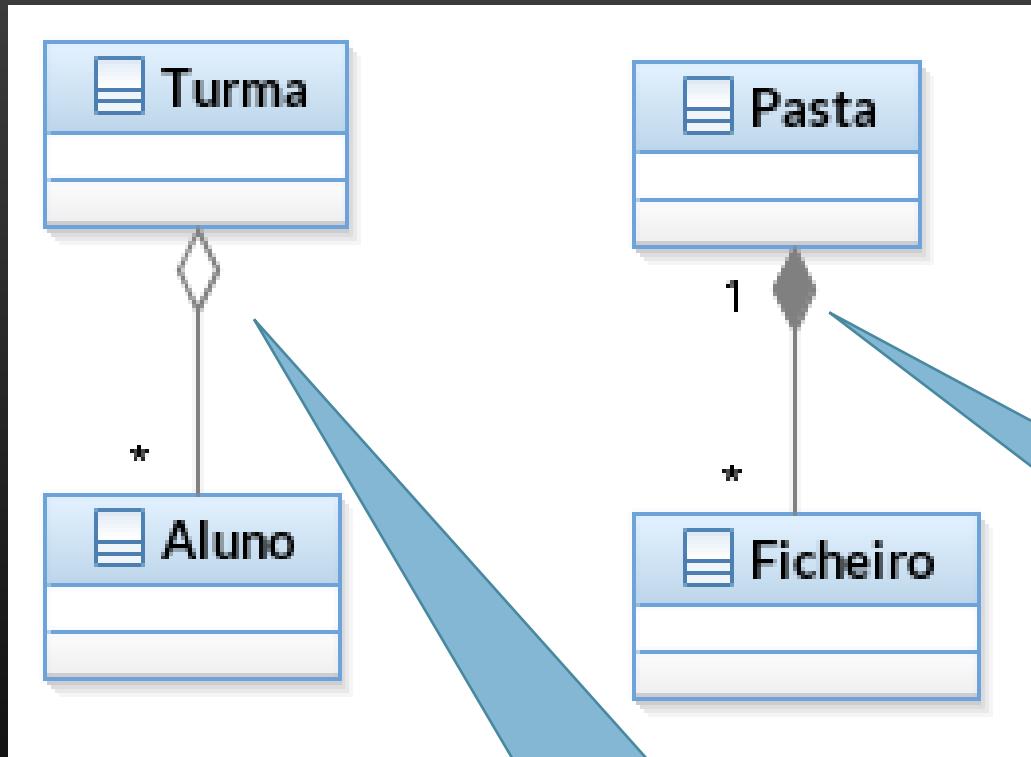
Nome='Joana Joaquim'

...

Idade=31

Aggregation vs. composition

Figure Shared or exclusive instances



Aggregation: A owns parts B
in a non-exclusive way

→ Shareable B Instances

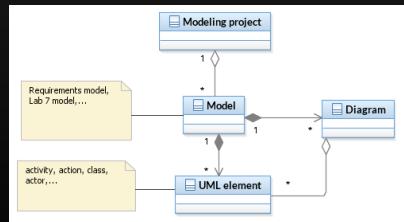
Composition: The B are
constituent parts of A

→ Non-shareable B instances

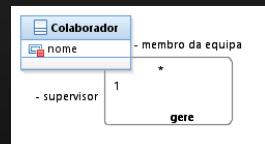
Composition

Agregation

Structure of a modeling project in a UML tool

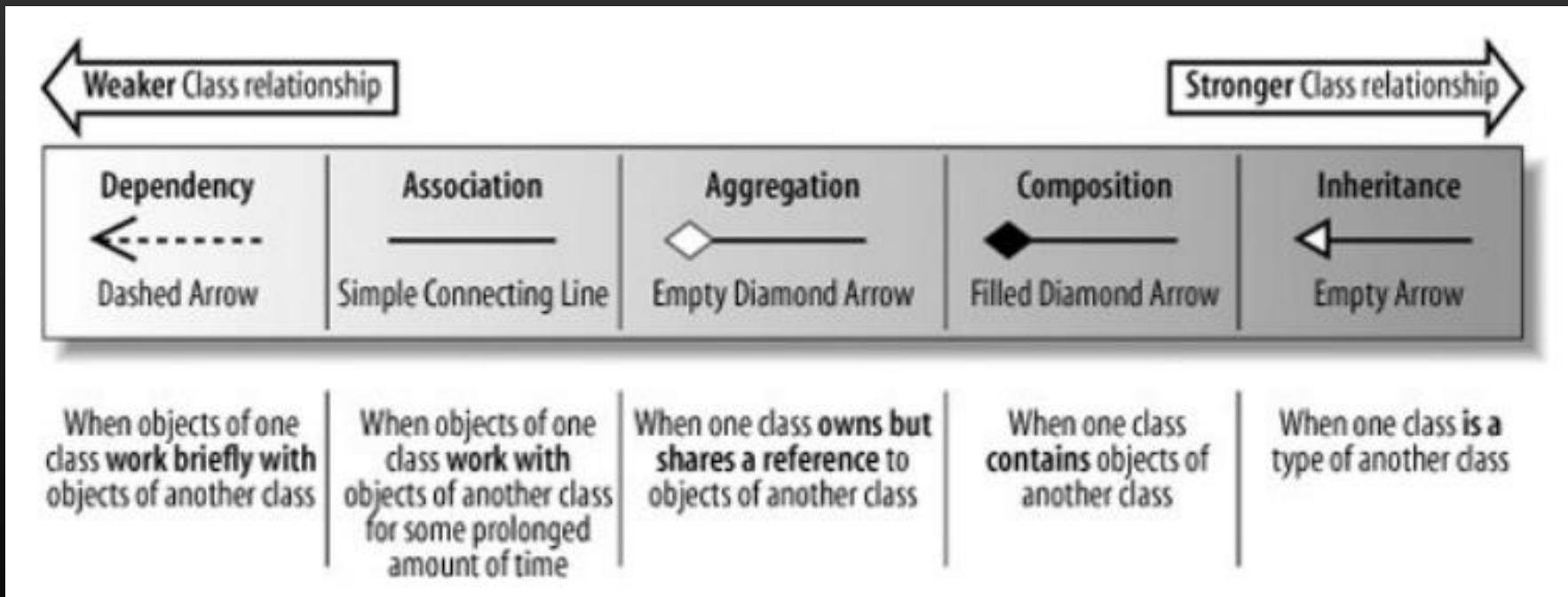


Reflexive associations relate instances of the same class



Structural bond strength between classes

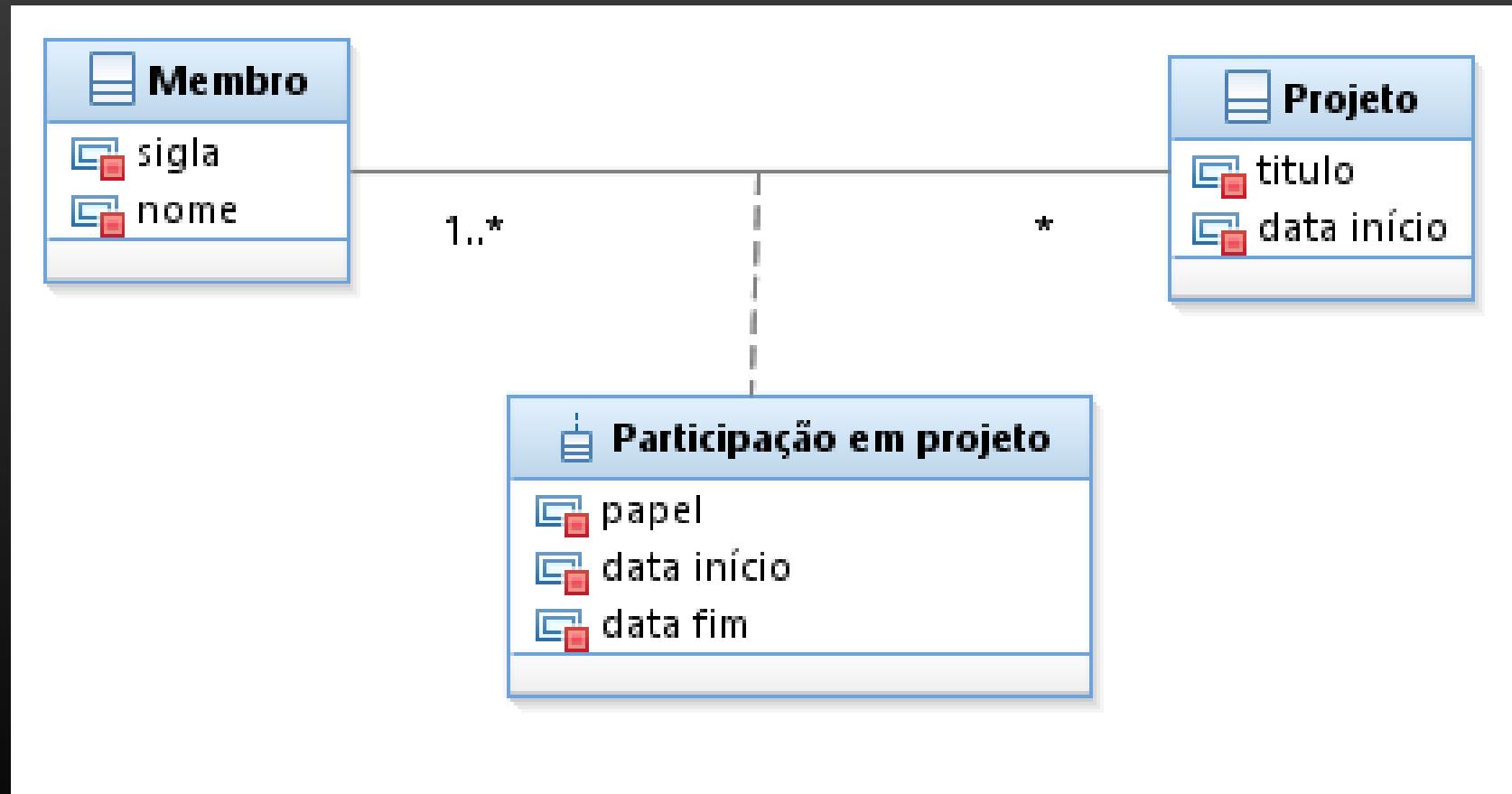
Five different types of relationship between classes, with different binding levels



Modeling problema....

A given Task (in a project) can be contributed by several employees, during different periods. Anna worked in the PayPal integration task , as a quality assurance engineer, for the first half of it.

Association-classes captures information describing the relationship

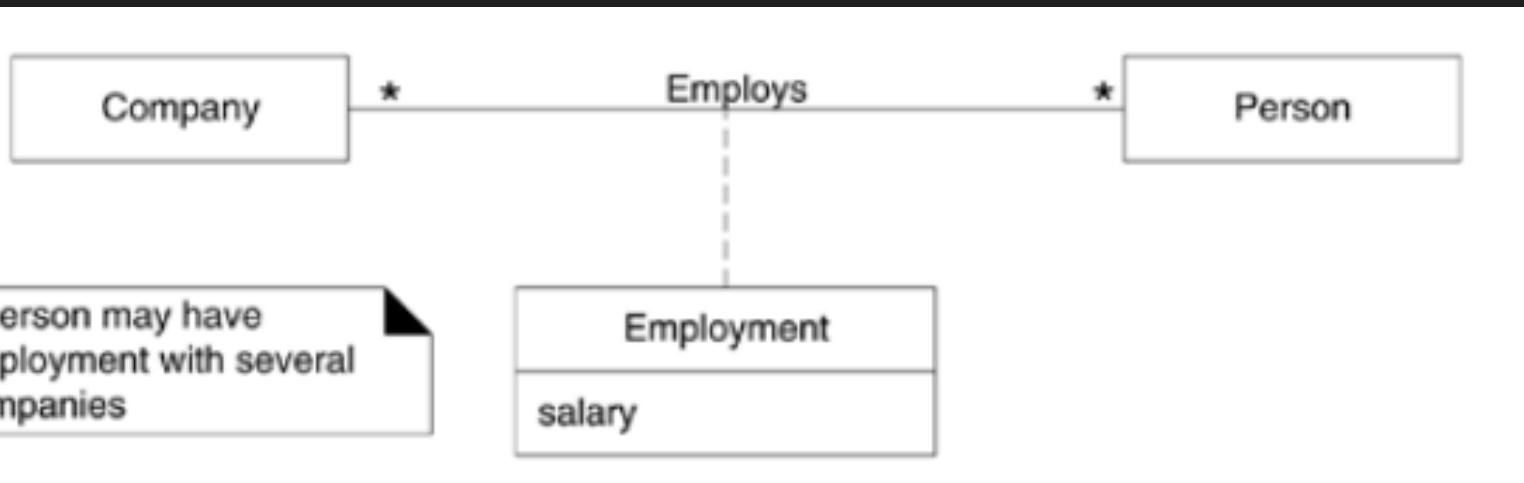


Indications for the use of a class-association in the domain model

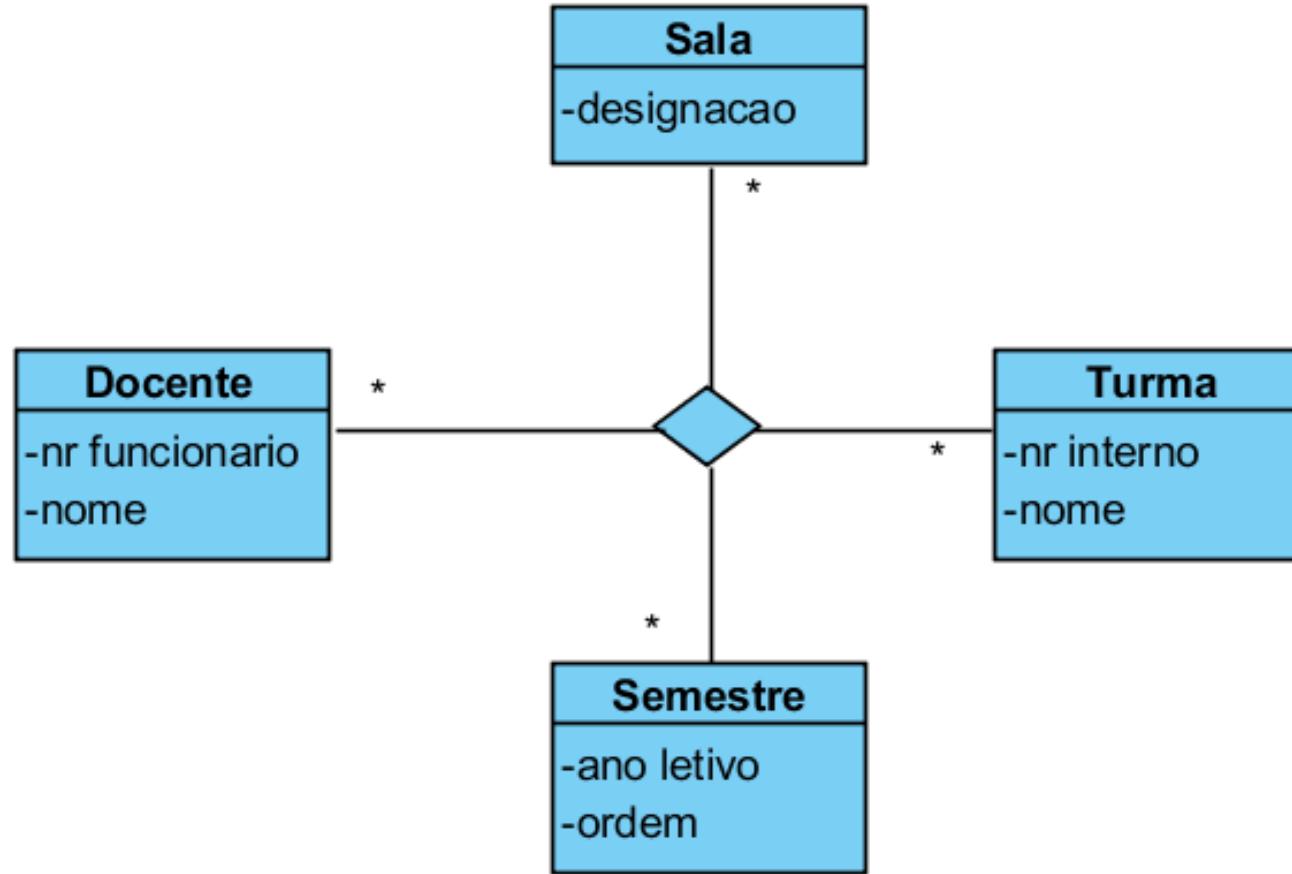
An attribute is related to (the occurrence of) an association.

Instances of the association-class have a lifetime dependent on the association

There is a N:M relationship between two concepts and information that characterizes the association itself



N-ary Associations



Modeling problema....

In our business, every payment (from sales) has to be one out of 3 possibilities: in cash, with card and in cheque. In the case of a cheque, it is important to keep the cheque number for follow-up.

Abstract Classes

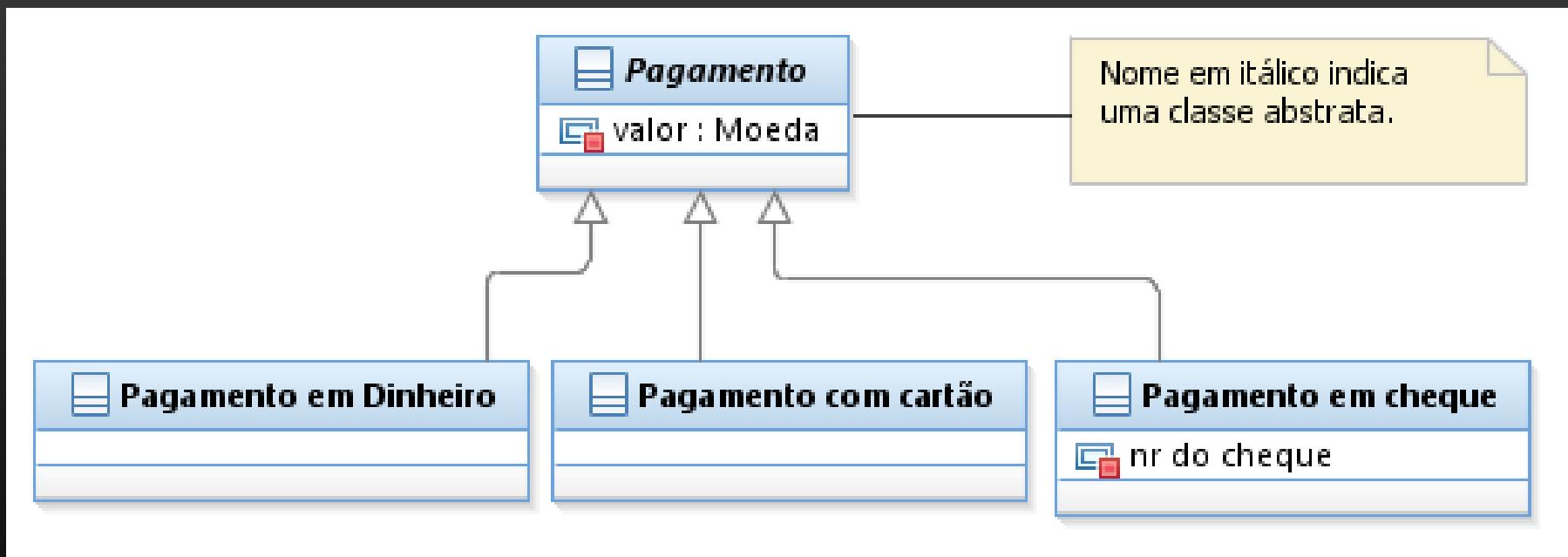


Pode haver instâncias de Pagamento que não sejam em Dinheiro, Cartão ou Cheque? Se sim, Pagamento não deve ser uma classe abstrata.

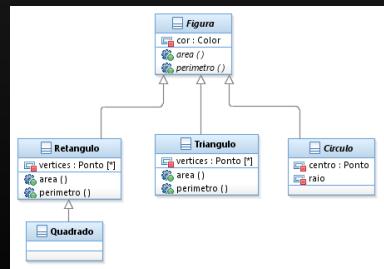


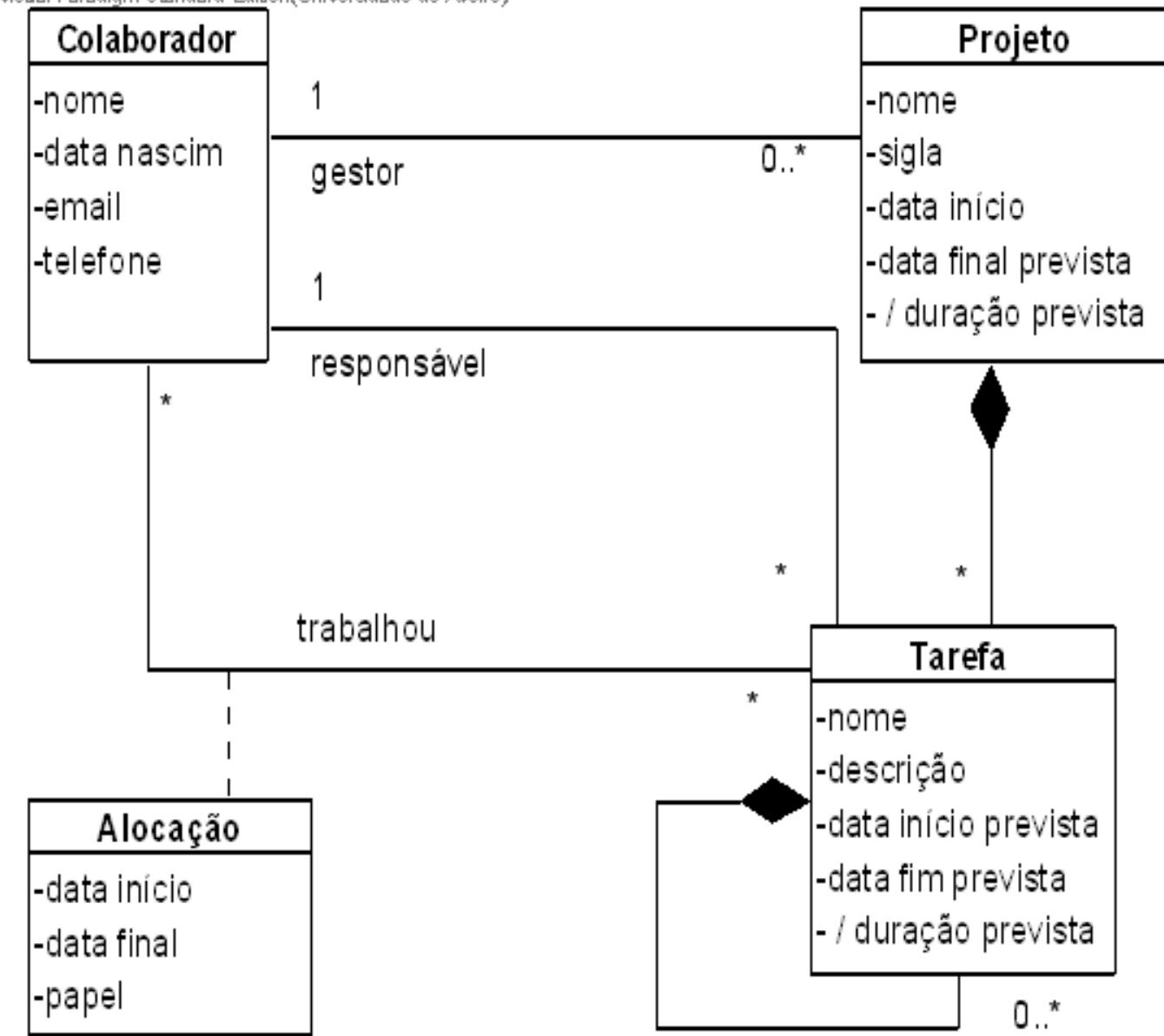
Um Pagamento tem de ser instanciado numa das formas indicadas pelas subclasses: em Dinheiro, Cartão ou Cheque. Pagamento deve ser uma classe abstrata.

An abstract class is not instantiated directly



Abstract Classes facilitate partial implementations (common parts in the superclass)





Perguntas de exame

16.

Considere o Diagrama 3:

- a) Um Projeto pode ou não ter um gestor.
- b) Um Projeto pode agregar sub-projetos.
- c) Uma Tarefa pode agregar sub-tarefas.
- d) Uma Tarefa pode ser realizada em diferentes projetos.
- e) Cada Colaborador é responsável por uma Tarefa.

17.

O Diagrama 3 utiliza uma classe de associação, para caracterizar a alocação a projetos.

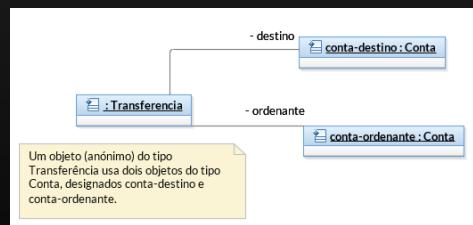
- a) A classe de associação é uma classe cujas características descrevem uma associação, e não um objeto "normal".
- b) A classe de associação facilita a visualização da interdependência entre duas classes.
- c) A classe de associação pode ser suprimida, desde que se mova os respetivos atributos para uma das classes associadas.
- d) A classe de associação deve ser usada sempre que há uma multiplicidade de muitos para muitos (entre as classes base associadas).
- e) Um bom modelo deve evitar a utilização de classes de associação.

18.

Relativamente ao Diagrama 3, Tarefa e Projeto indicam durações previstas.

- a) Há um erro de notação no atributo associado à duração prevista.
- b) A duração prevista pode ser determinada à custa de outros atributos, não deve ser representada na classe.
- c) A duração prevista pode ser determinada à custa de outros atributos, é um atributo derivado.
- d) A duração prevista deveria ser definida apenas na Tarefa.
- e) A duração prevista é obtida por um método e é errado apresentar como um atributo.

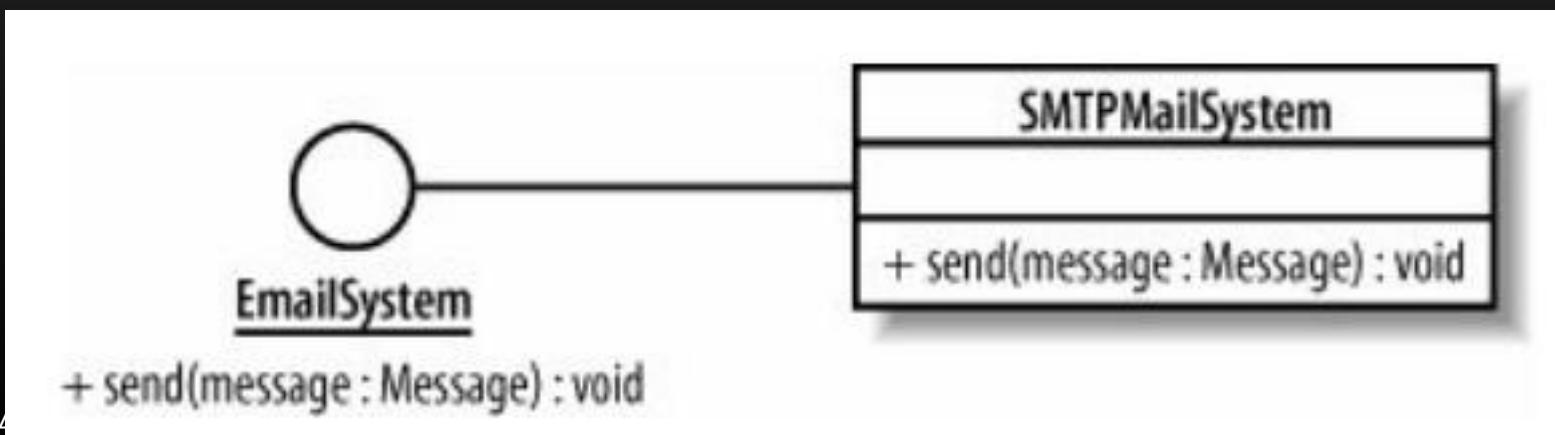
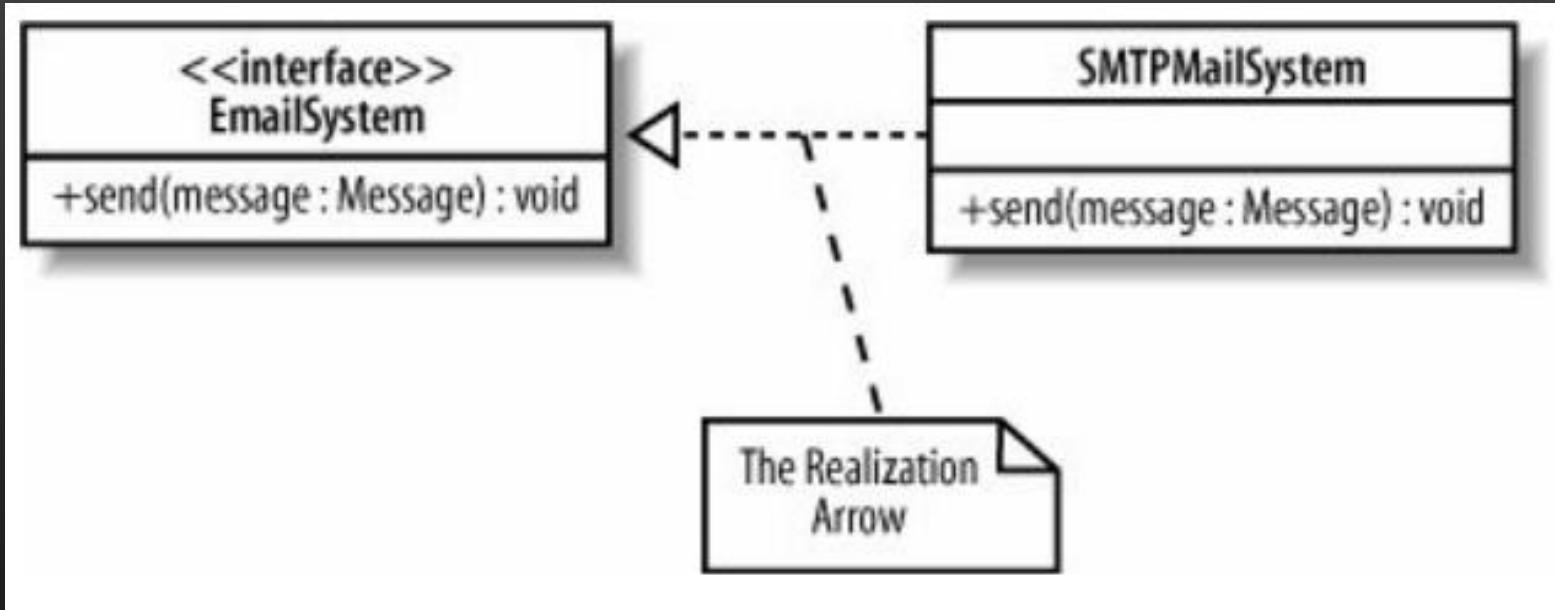
Object diagrams



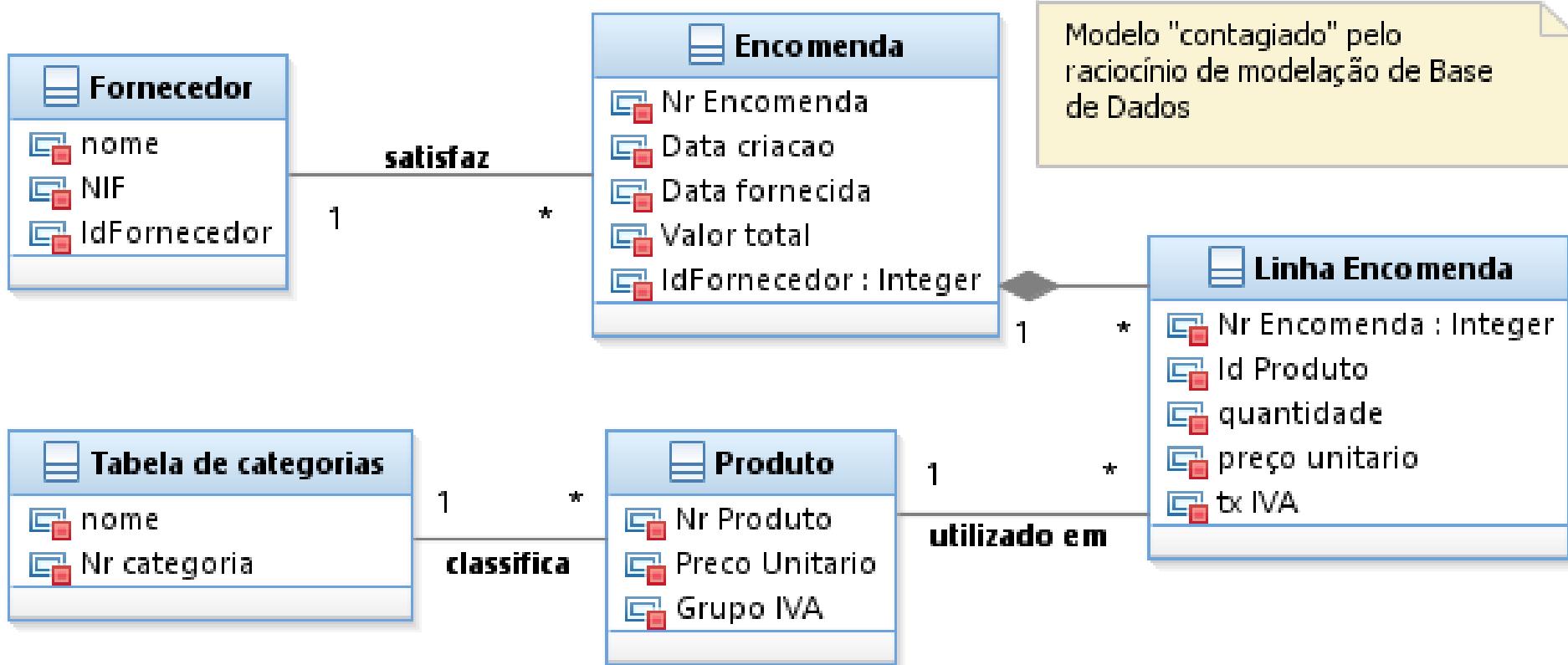
Example

O João Joaquim autorizou a requisição de material #75, que pedia tinteiros CLI-520.

Interfaces are unimplemented and stateless contracts (programming concept)



The domain model is not the model of a database



Readings & references

Core readings	Suggested readings
<ul style="list-style-type: none">• [Dennis15] - Chap. 5	<ul style="list-style-type: none">• [Larman12] -Chap. 9• VisualParadigm: <u>what is the class diagram?</u>

47006- ANÁLISE E MODELAÇÃO DE SISTEMAS

Behavioral models (sequence, state)

Ilídio Oliveira

v2020/11/13, TP11

Learning objectives for this lecture

Understand the role of behavior modeling in the SDLC

Understand the rules and style guidelines for sequence,
communication and state diagrams

Understand the complementarity between sequence and
communication diagrams

Map sequence diagrams in code and reverse

Explain the relationship between function, structural and
behavior models

The target of behavioral modeling

Behavioral models describe the dynamic aspects of an information system.

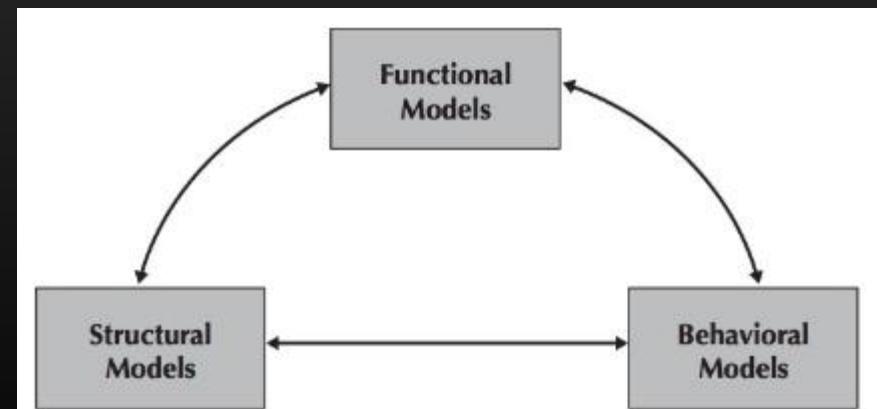
During analysis: behavioral models describe what the **internal logic of the processes** is without specifying how the processes are to be implemented. (in the design and implementation phases, the detailed design is fully specified)

Behavioral modeling is also **use-case driven**. You specify use cases realizations.

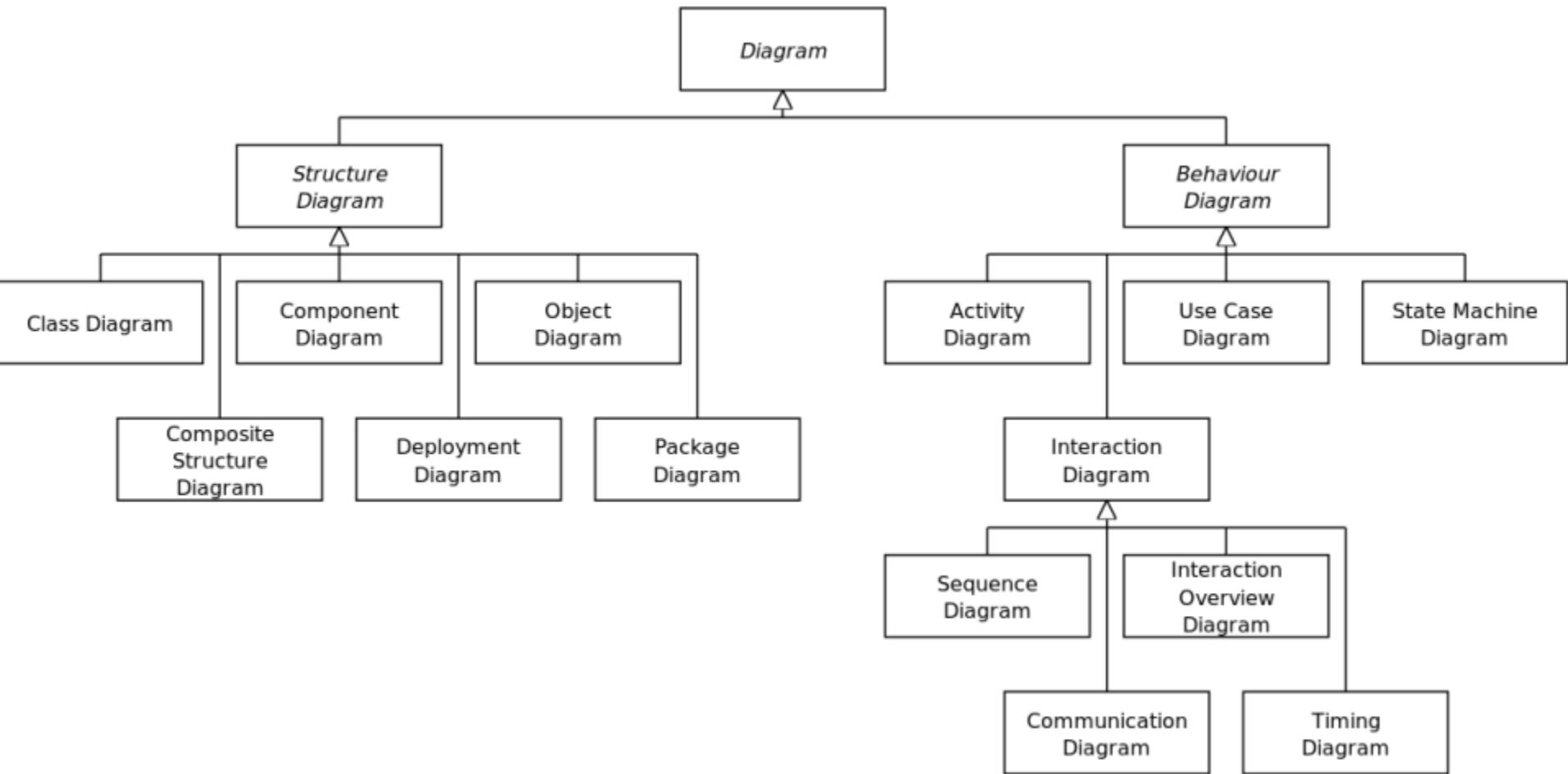
One of the primary purposes is to **show how the underlying objects in a problem domain will work together to form a collaboration** to support each of the use cases' scenarios.

Structural models → the objects and the relationships

Behavioral models → internal view of the process that a use case describes.



Diagramas da UML 2.x



Behavioral model types

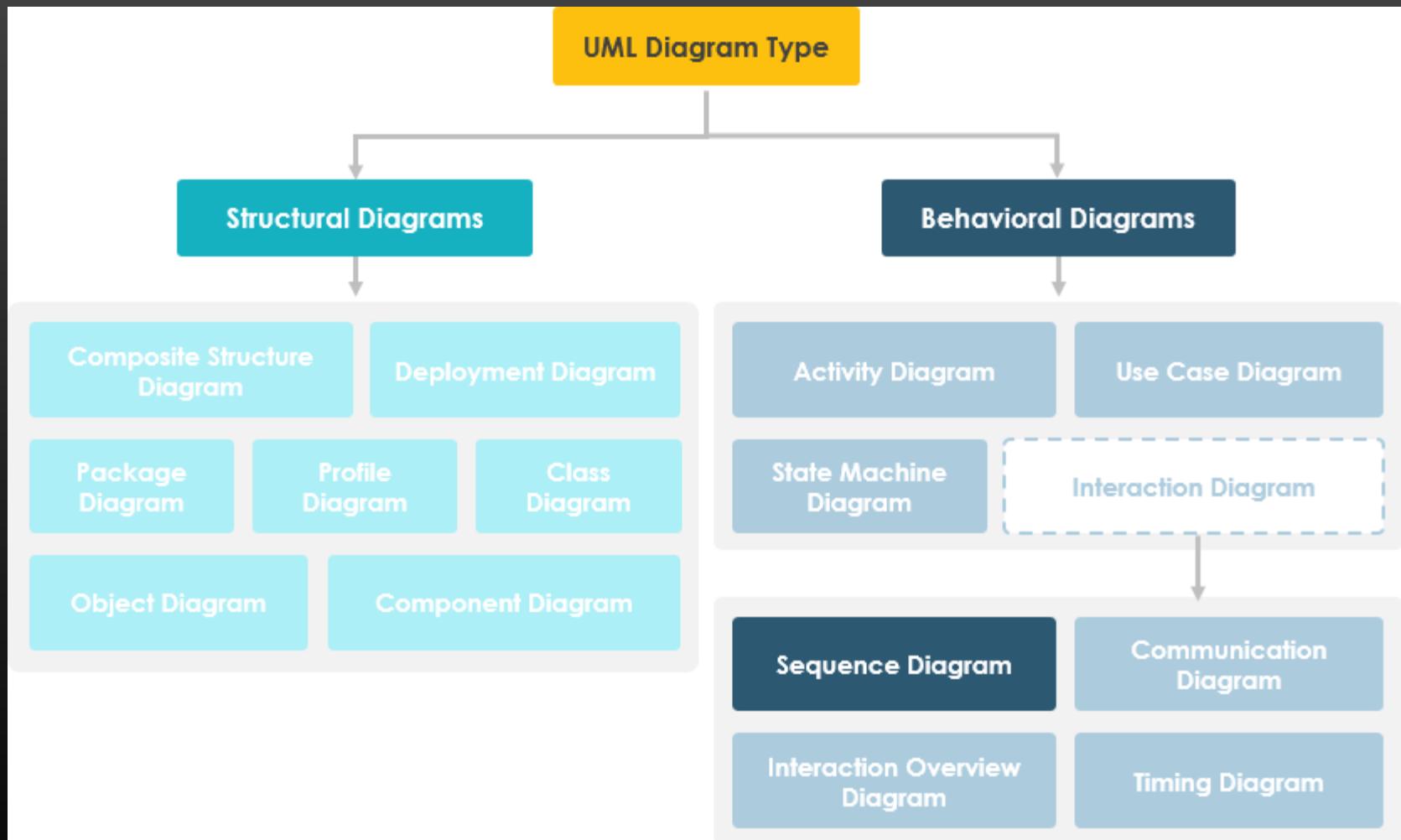
Representation of the details of a process

- Interaction diagrams (Sequence & Communication)
- Shows how objects collaborate to provide the functionality defined in the use cases.

Representations of changes in the data (state)

- Behavioral state machines

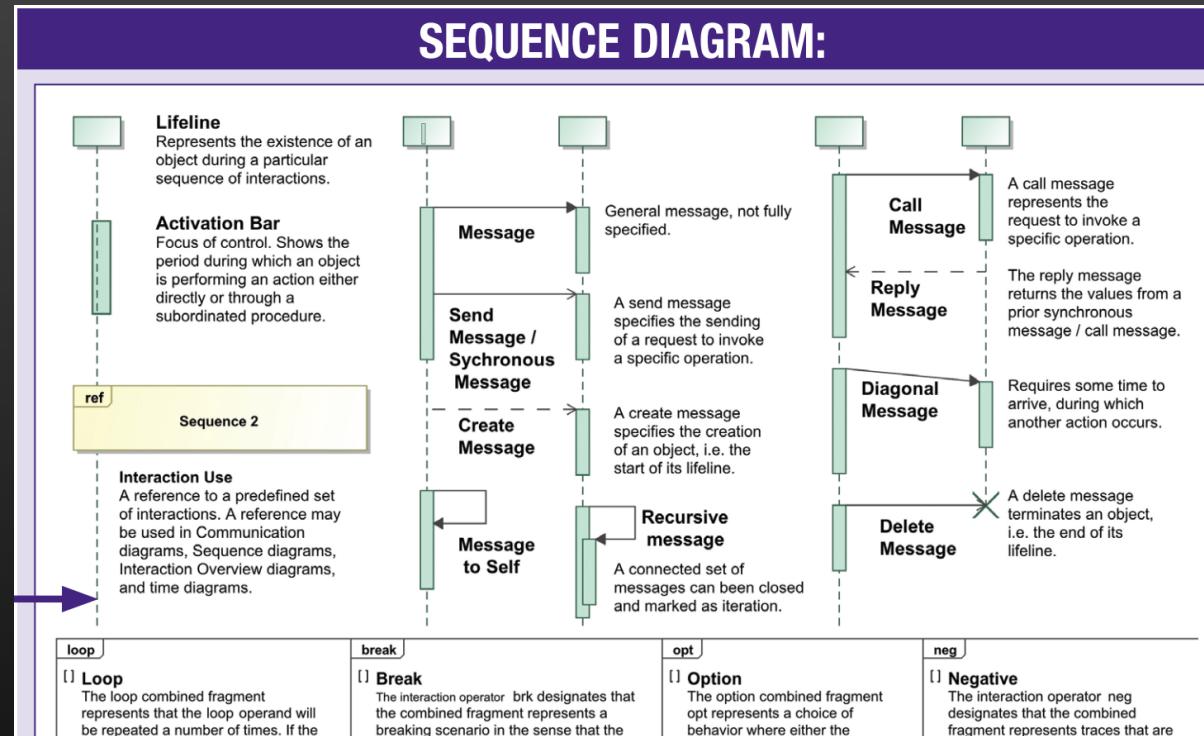
UML diagrams



Sequence diagram

Illustrate the objects in a collaboration and the messages that pass between them over time.

A sequence diagram is a dynamic model that shows the explicit sequence of messages that are passed between objects in a defined interaction.



NoMagic's [UML Reference card](#)

Focus on **object-level collaboration**

Class diagrams

The modeling focus of class diagrams is at the class level.

Each object has attributes that describe information (state) about the object.

Interaction diagrams

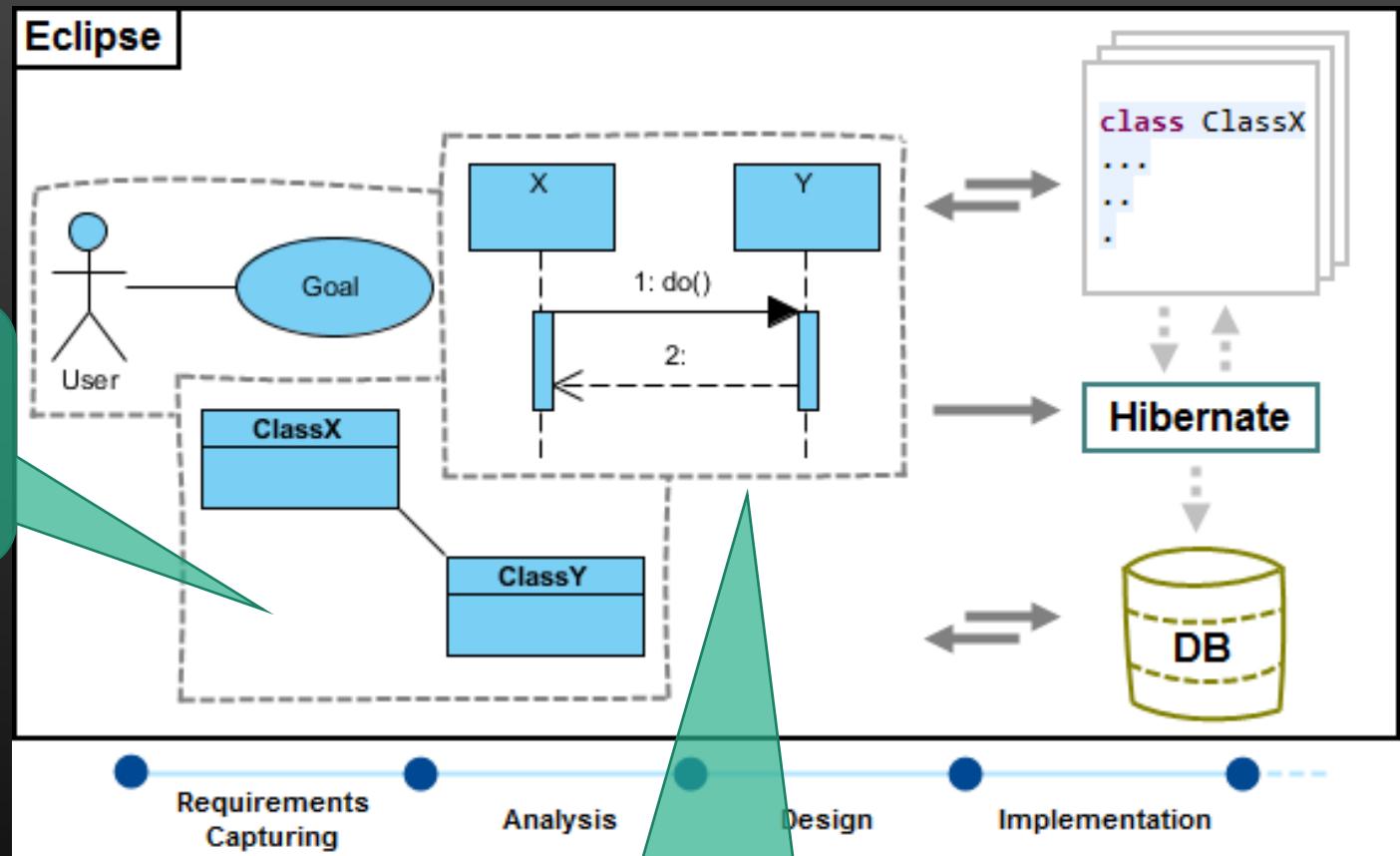
The interaction diagrams focus on the **object level**

Each object also has behaviors. The behaviors are described by operations. An operation is an action that an object can perform.

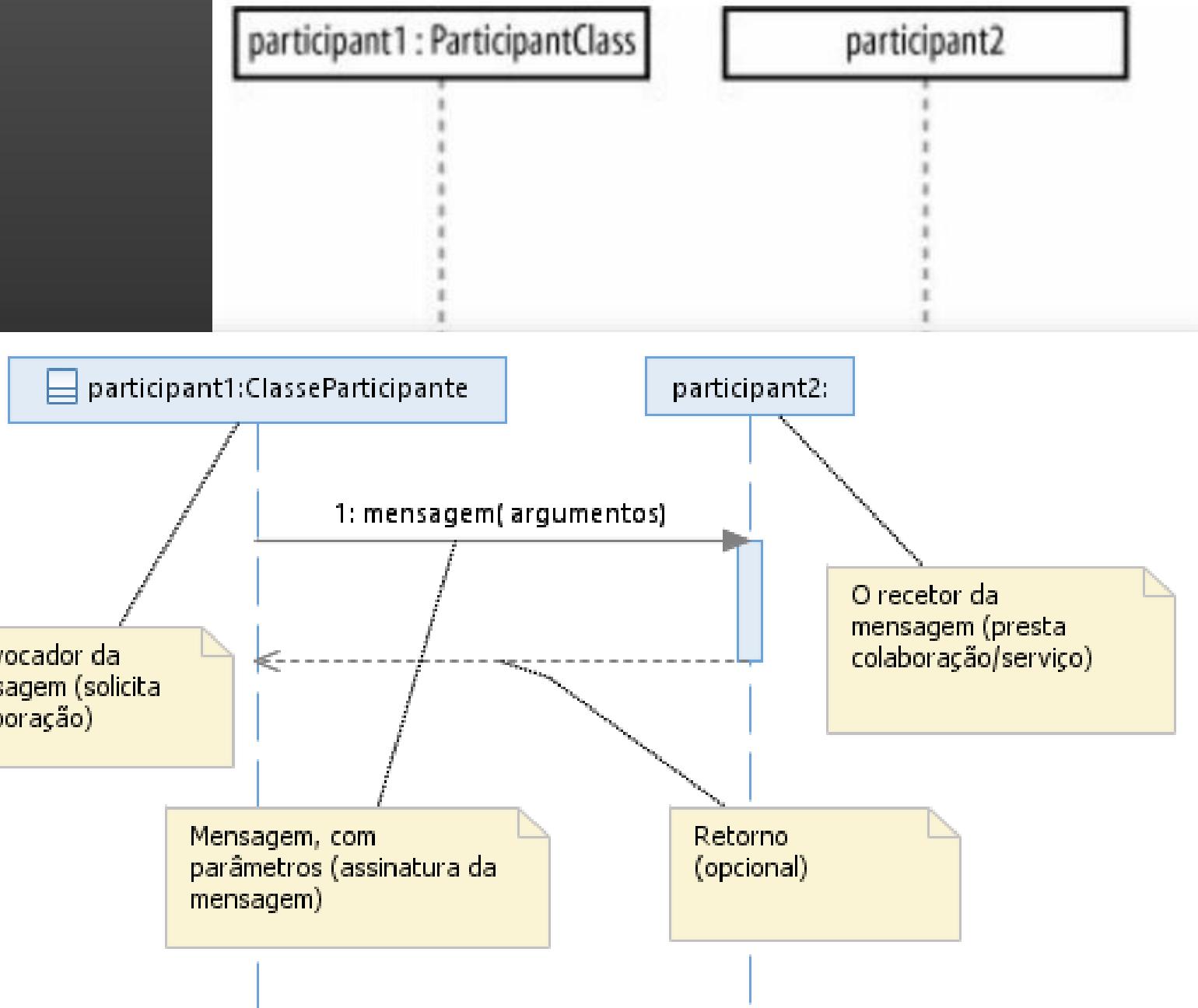
Each object also can **send and receive messages.** Messages are information sent to objects to tell an object to execute one of its behaviors. Essentially, a message is a function or a *call* from one object to another object.

Nível de abstração: análise ou implementação?

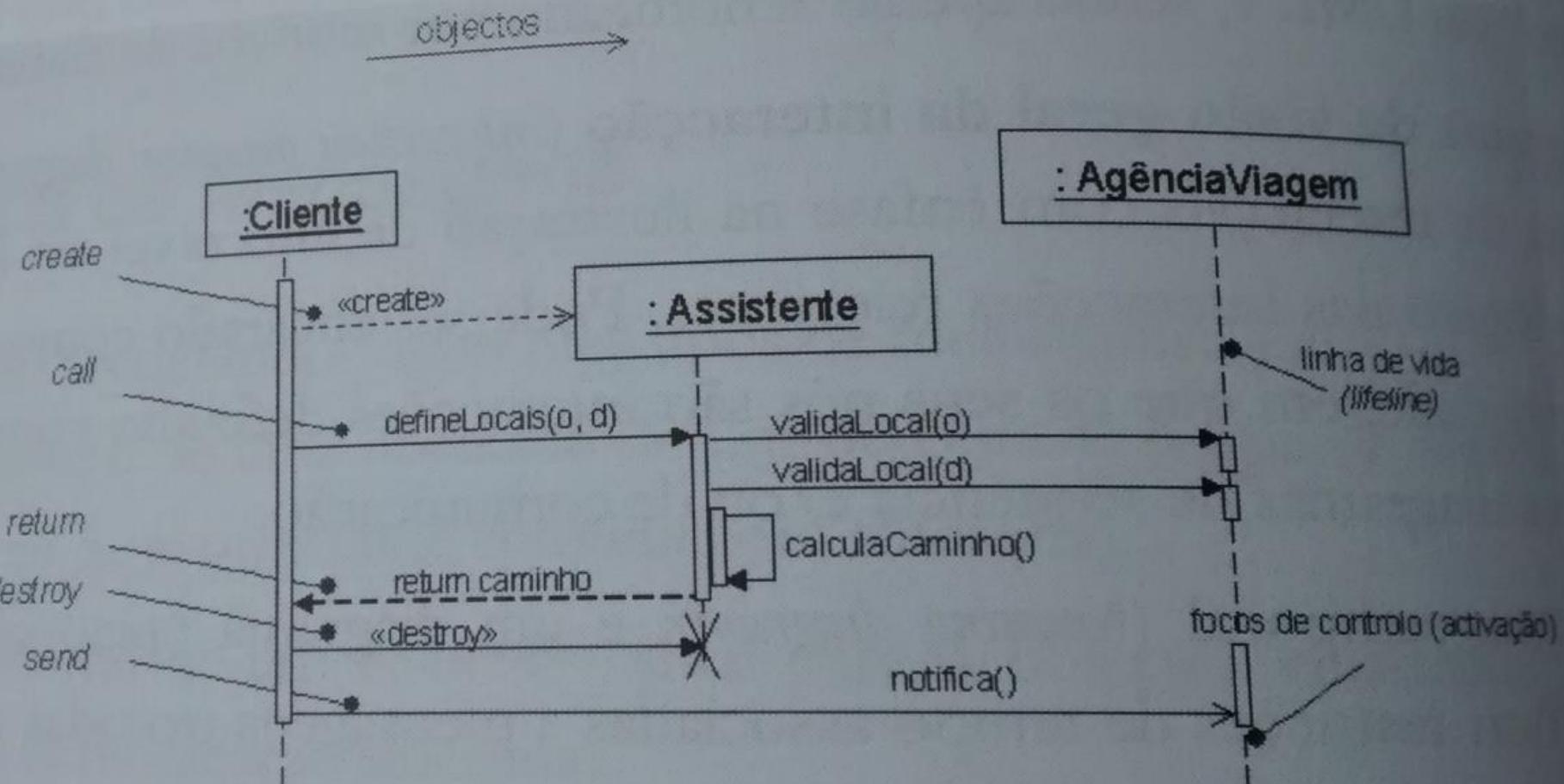
As Classes definem características dos objetos de um tipo (*molde*).
As classes podem representar entidades do domínio ou da implementação.



O D. de Sequência mostra como é que os objetos (que seguem uma Classe) colaboram com outros, para realizar uma atividade. Também aqui, essa atividade pode ser no nível do “domínio” ou no nível do código.



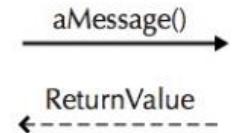
Colaboração entre objetos por mensagens (síncronas)



Term and Definition	Symbol
<p>An actor:</p> <ul style="list-style-type: none"> ■ Is a person or system that derives benefit from and is external to the system. ■ Participates in a sequence by sending and/or receiving messages. ■ Is placed across the top of the diagram. ■ Is depicted either as a stick figure (default) or, if a nonhuman actor is involved, as a rectangle with <><> in it (alternative). 	 <p>anActor</p> <div data-bbox="1416 626 1699 727" style="border: 1px solid black; padding: 5px; width: fit-content;"> <<actor>> anActor </div>
<p>An object:</p> <ul style="list-style-type: none"> ■ Participates in a sequence by sending and/or receiving messages. ■ Is placed across the top of the diagram. 	<div data-bbox="1416 856 1699 928" style="border: 1px solid black; padding: 5px; width: fit-content;"> anObject : aClass </div>
<p>A lifeline:</p> <ul style="list-style-type: none"> ■ Denotes the life of an object during a sequence. ■ Contains an X at the point at which the class no longer interacts. 	

A message:

- Conveys information from one object to another one.
- A operation call is labeled with the message being sent and a solid arrow, whereas a return is labeled with the value being returned and shown as a dashed arrow.



A guard condition:

- Represents a test that must be met for the message to be sent.



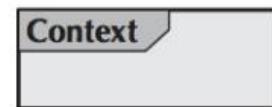
For object destruction:

- An X is placed at the end of an object's lifeline to show that it is going out of existence.

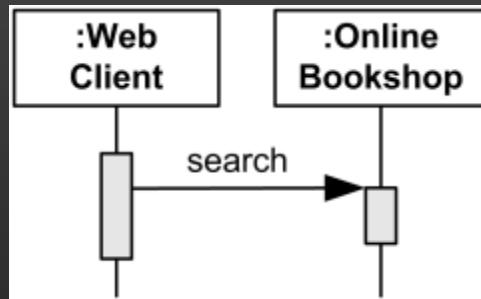
X

A frame:

- Indicates the context of the sequence diagram.

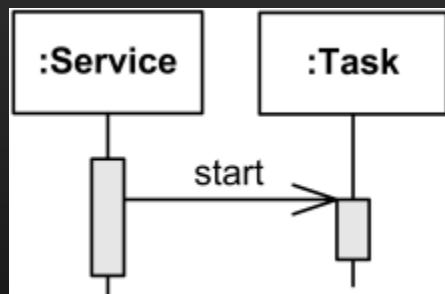


Semântica da invocação



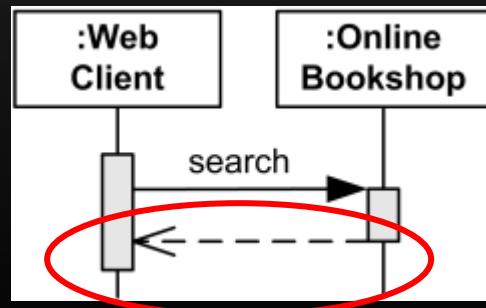
síncrona

...
result = B.search();
...



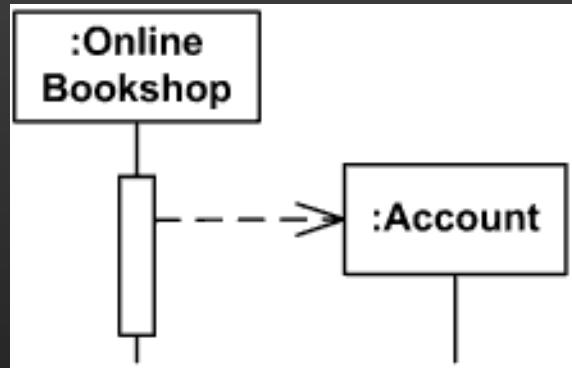
assíncrona

...



retorno

Modelar a criação/destruição do participante

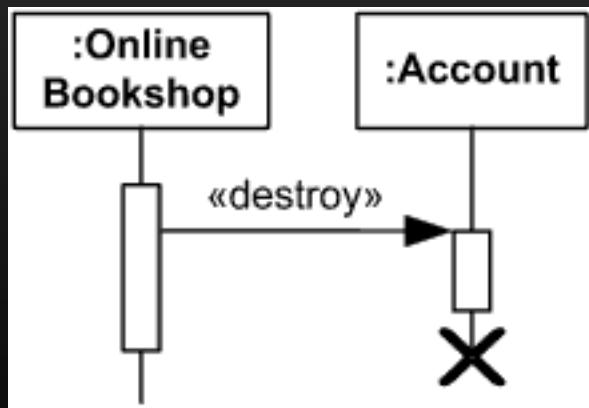


create

...

```
Account a= new Account()
```

...



destroy

...

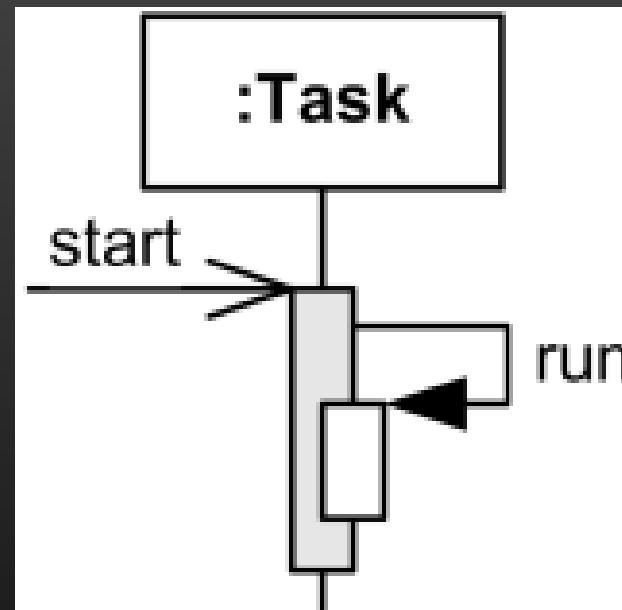
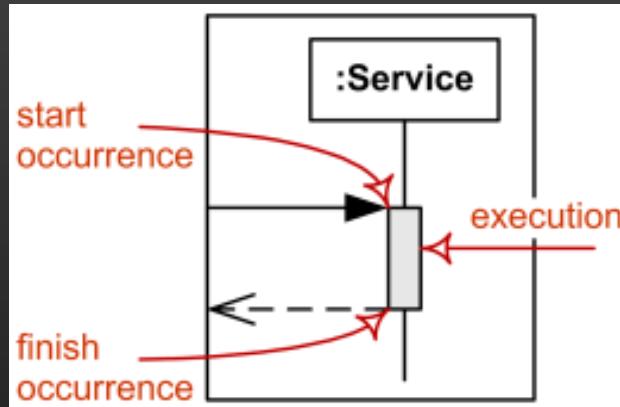
```
a=null; // in java
```

```
[a release] // objective C
```

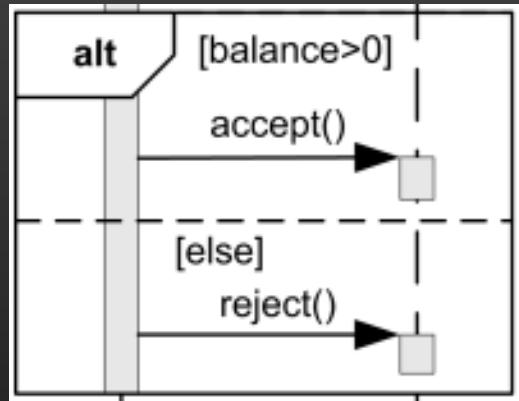
```
free a; // C
```

From <http://www.uml-diagrams.org/sequence-diagrams.html>

Ativação



Fragments para mostrar alternativas

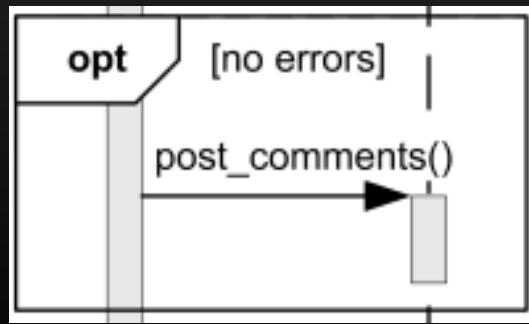


If (balance>0)

otherObj.Accept()

Else

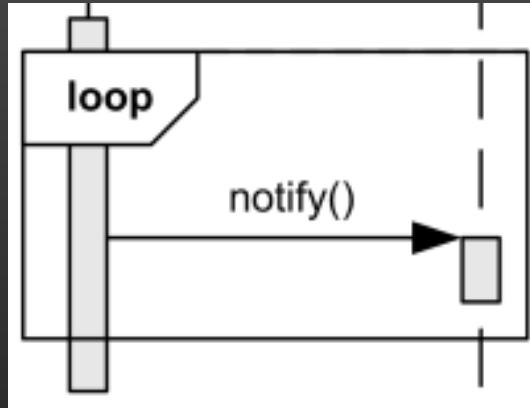
otherObj.Reject()



If (no errors)

otherObj.Post_comments()

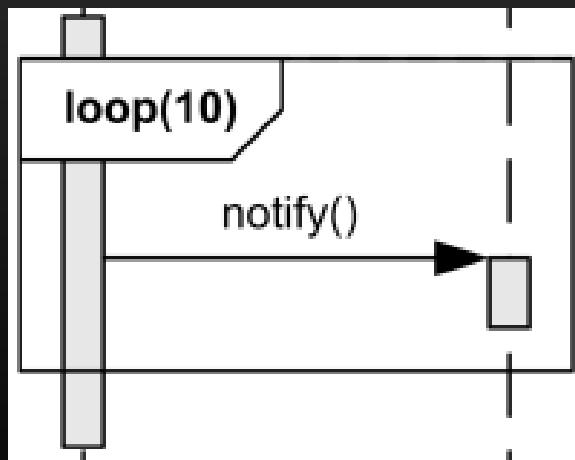
Fragments para mostrar loops



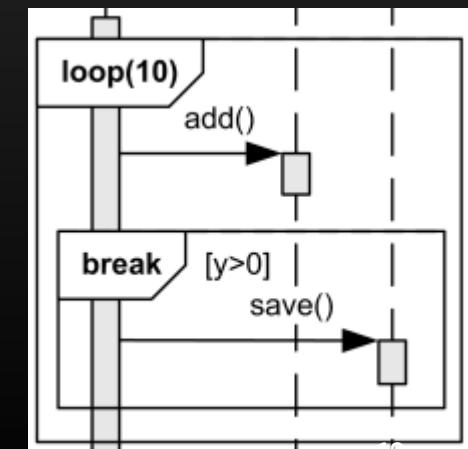
```
i=0;  
While( i<10 )
```

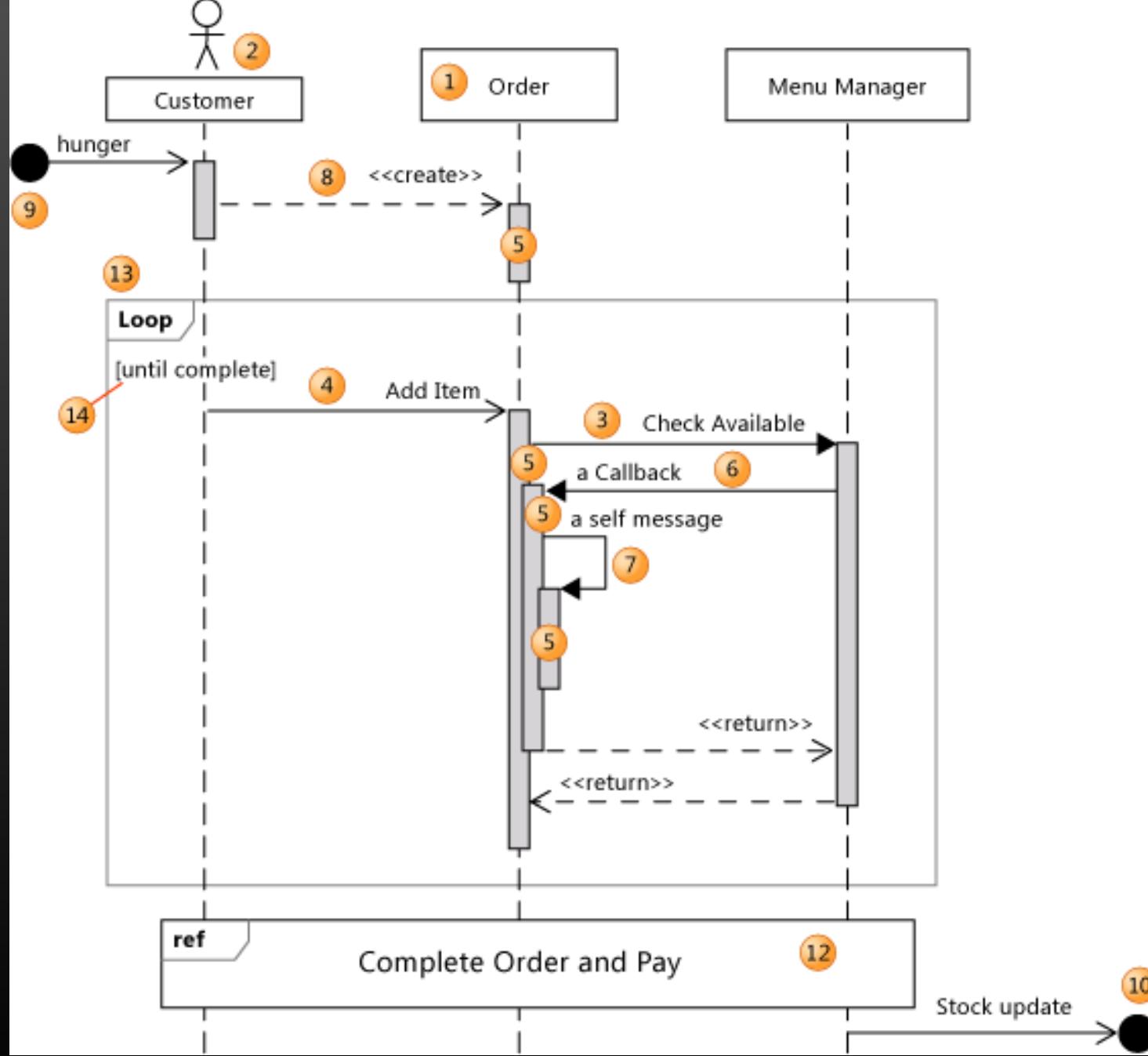
```
{
```

```
otherObj.Add()
```



```
if ( y>0 ) break;  
i++  
}
```





Building Sequence Diagrams

1. Set the context
2. Identify actors and objects that interact in the use-case scenario
3. Set the lifeline for each object
4. **Add messages by drawing arrows**
 - Shows how they are passed from one object to another
 - Include any parameters in parentheses
 - Obvious return values are excluded
5. Add execution occurrence to each object's lifeline
6. Validate the sequence diagram
7. Ensures that it depicts all of the steps in the process

Quatro tipos de diagramas de iteração disponíveis

D. Sequência



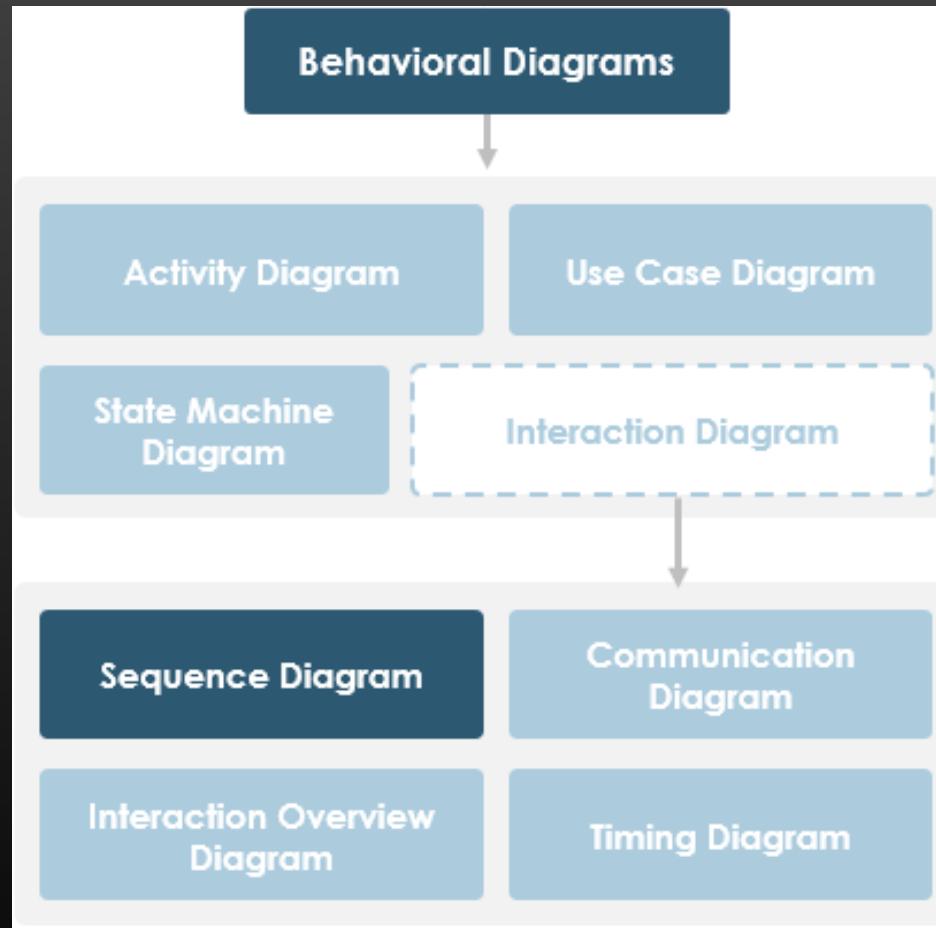
Formato alinhado

D. Comunicação

Formato grafo

Diagrama temporal (*timing*)

Diagrama de visão geral da interação (*interaction overview*)



Quatro tipos de diagramas de iteração disponíveis

D. Sequência

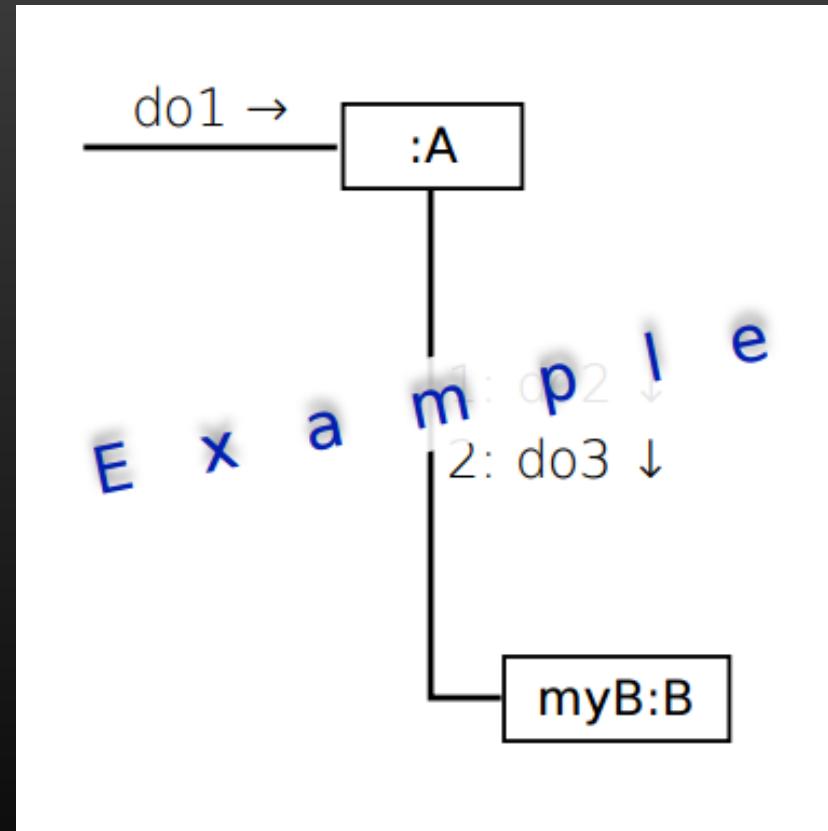
Formato alinhado

→ D. Comunicação

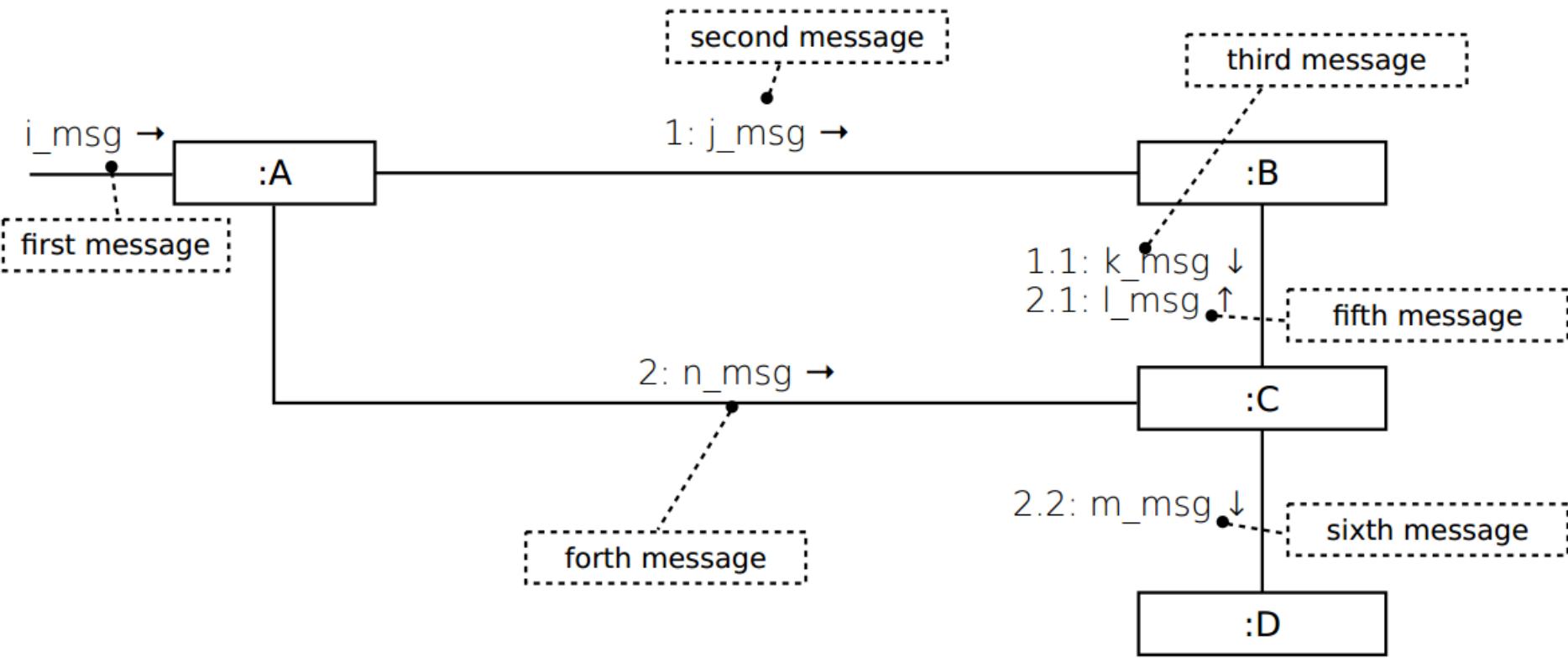
Formato grafo

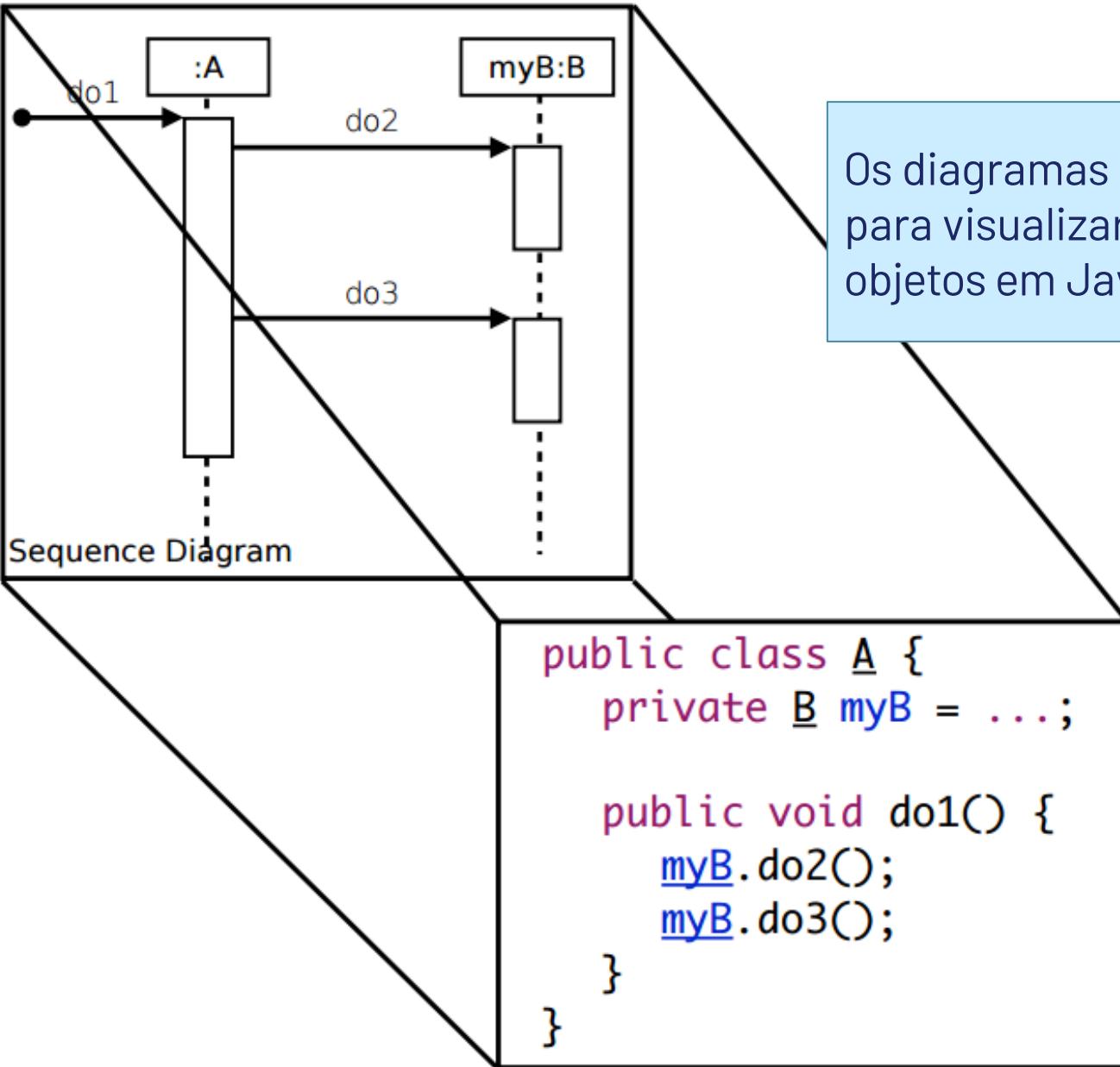
Diagrama temporal (*timming*)

Diagrama de visão geral da interação (*interaction overview*)



Diagramas de comunicação





Os diagramas de podem ser usados para visualizar a colaboração entre objetos em Java.

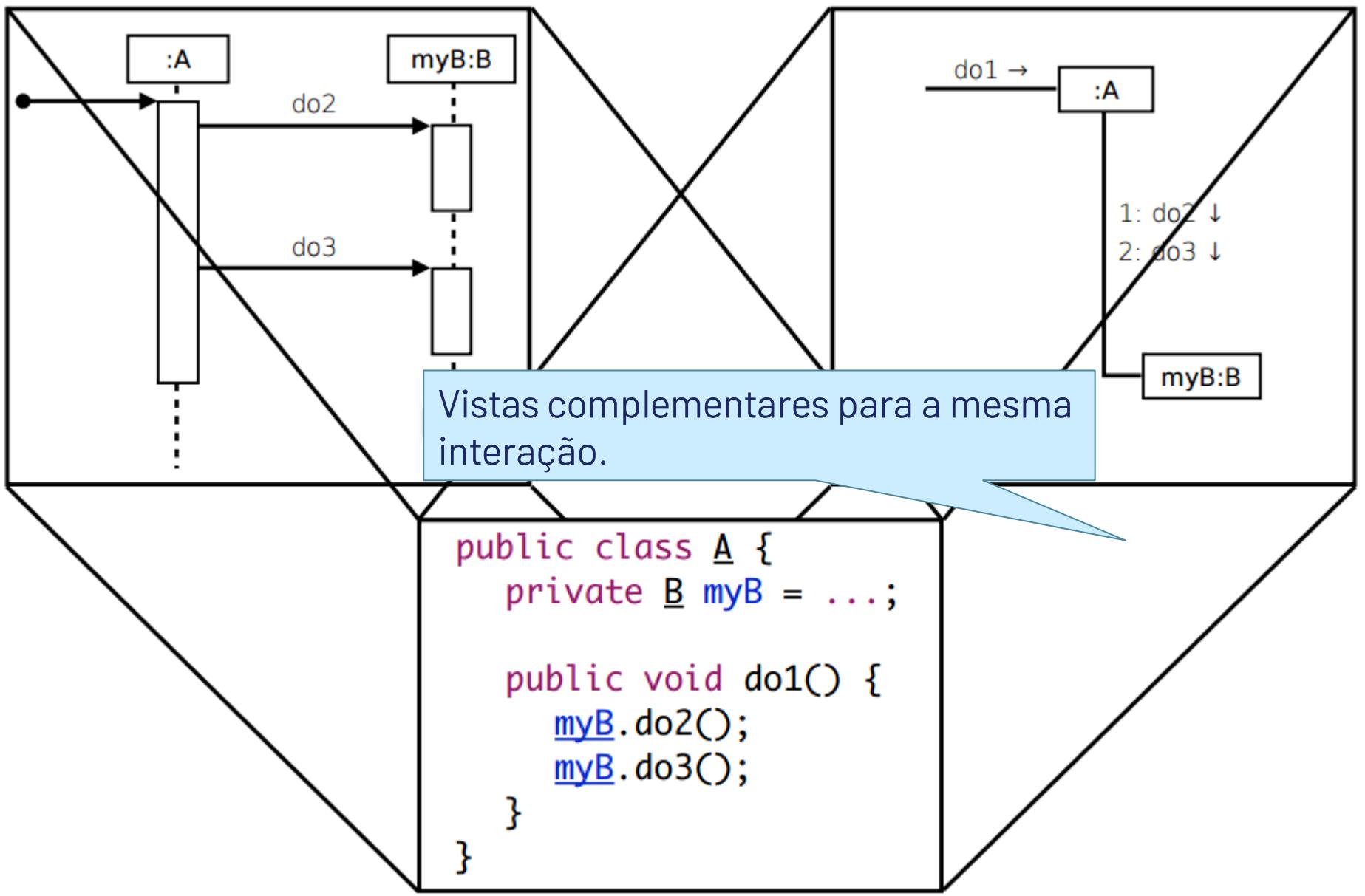


Figure Sequence Diagram for a Scenario of the Make Patient Appt Use Case

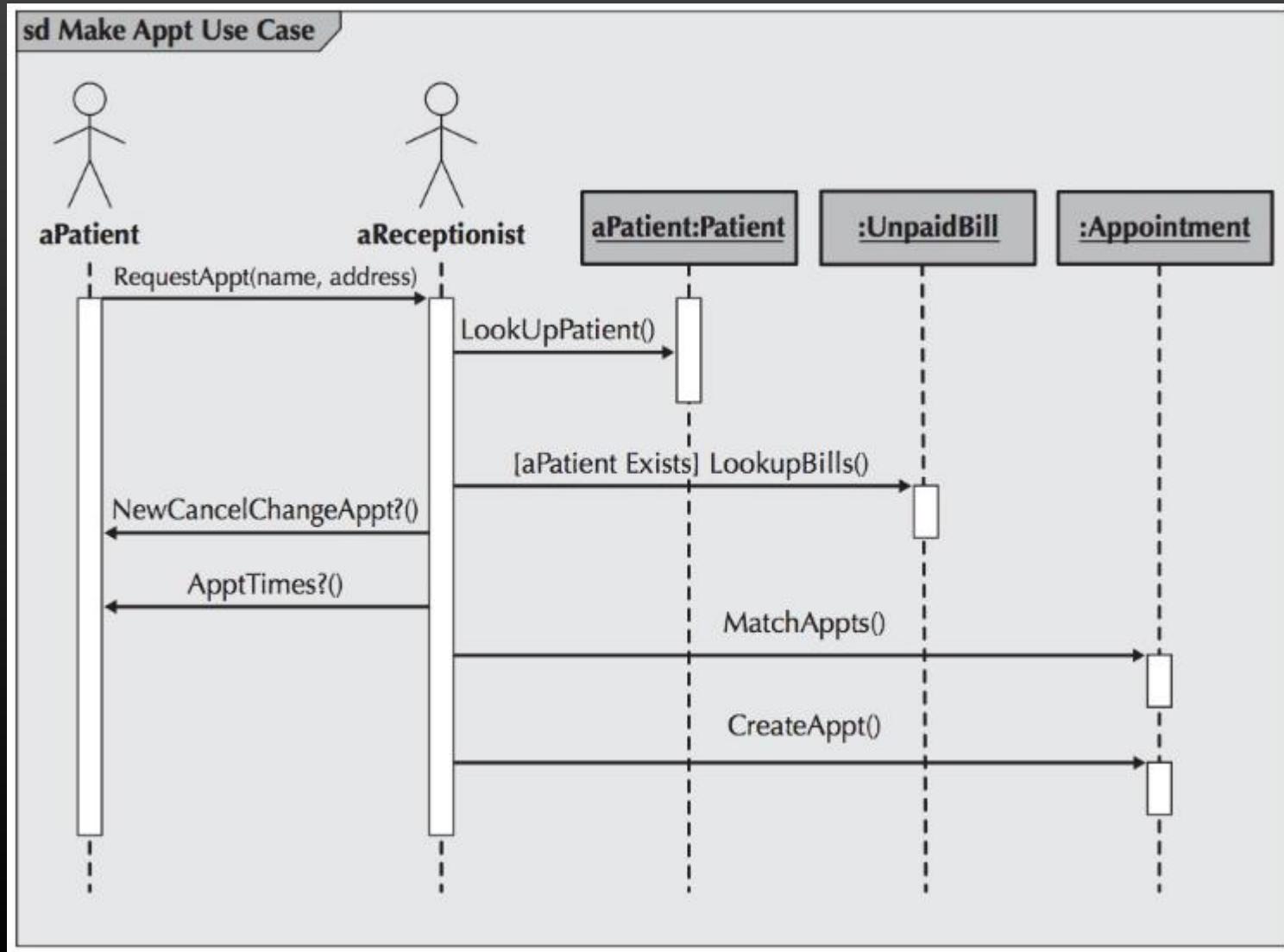
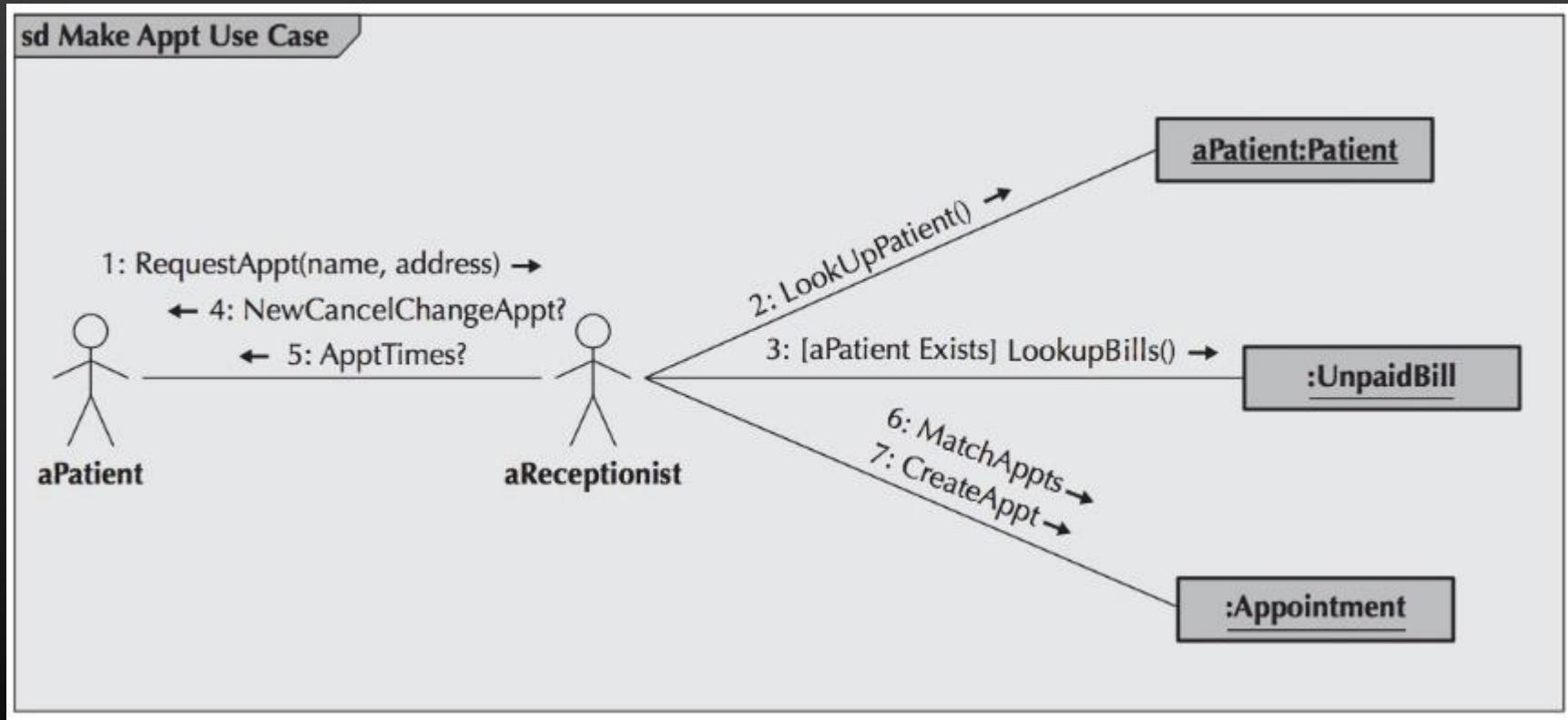


Figure Communication Diagram for a Scenario of the Make Patient Appt Use Case



Diagramas de sequência *vs* diagramas de comunicação

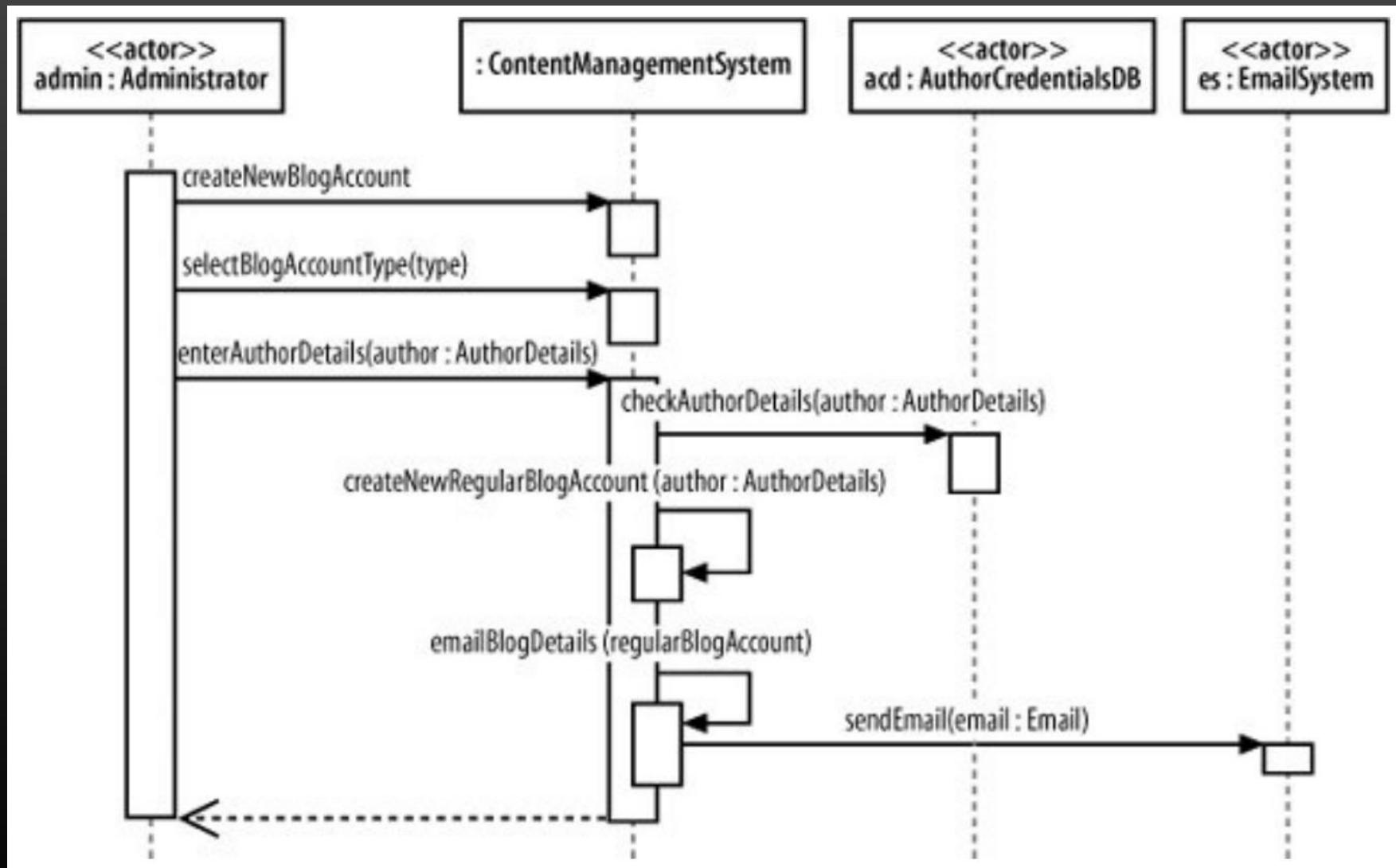
	Benefícios	Limitações
Diagrama de sequência	<ul style="list-style-type: none">• Mostra claramente a sequência/ordem temporal das mensagens• Possibilidades de notação alargadas	<ul style="list-style-type: none">• Cresce para a direita à medida que se acrescentam objetos
Diagrama de comunicação	<ul style="list-style-type: none">• Mais fácil de desenhar (objetos podem ser adicionado em qq parte)	<ul style="list-style-type: none">• Menos expressivo (ordem temporal)• Menos opções de notação• Pouco suportado nas ferramentas UML

Um caso de utilização é **realizado pela colaboração entre atores e sistema**

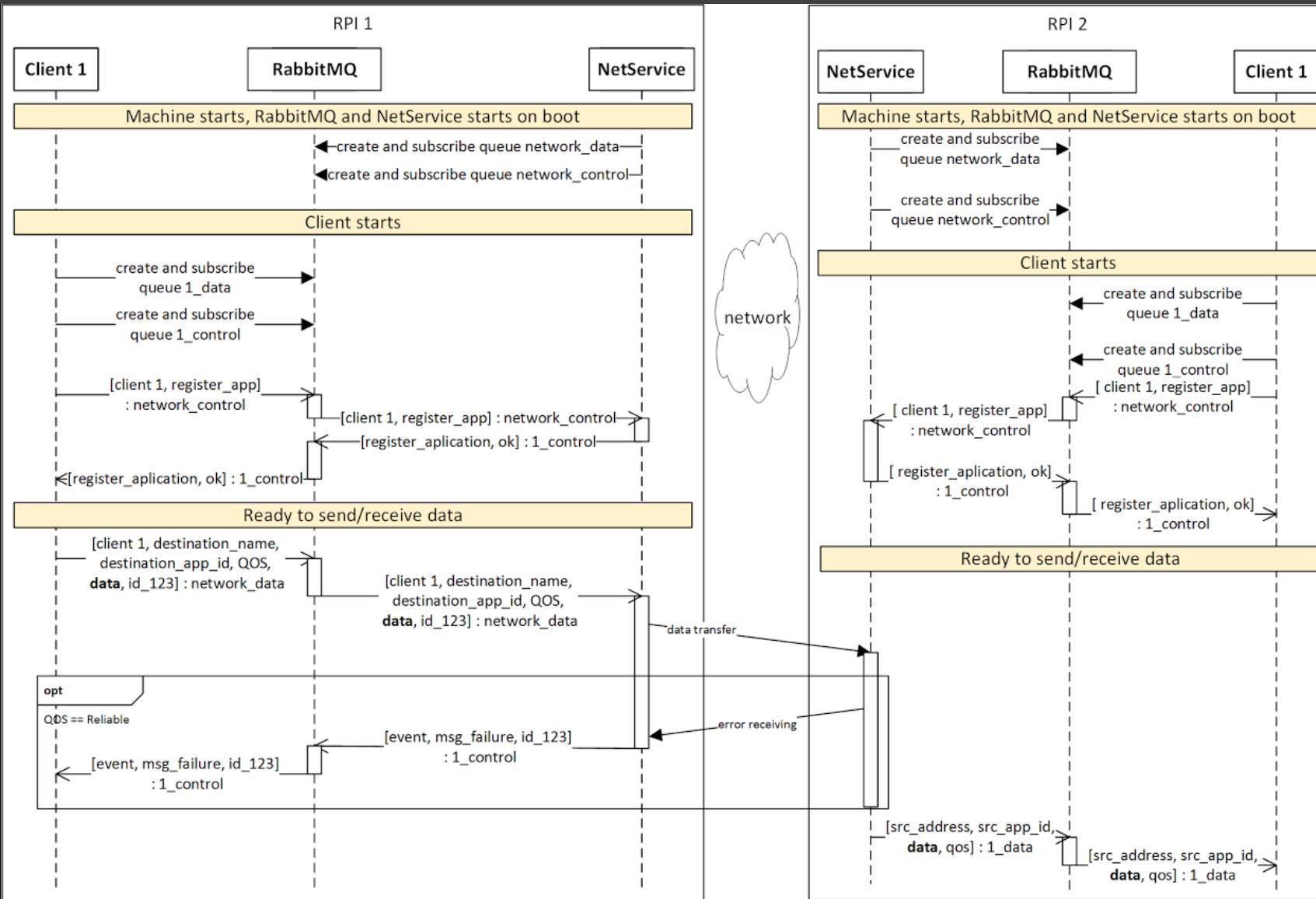
FLUXO TÍPICO

1. O Administrador pede ao sistema (CMS) para criar uma nova conta de *blogger*
2. O Administrador escolhe o tipo Normal
3. O Administrador fornece os detalhes do autor do blog
4. O CMS verifica os detalhes (se já existe) na base de Dados de Autores
5. O CMS cria a nova conta normal.
6. Um sumário dos detalhes da nova conta são enviados por email ao autor.





Outro exemplo de aplicação: documentar um protocolo



Behavioral State Machines

Objects may change state in response to an event

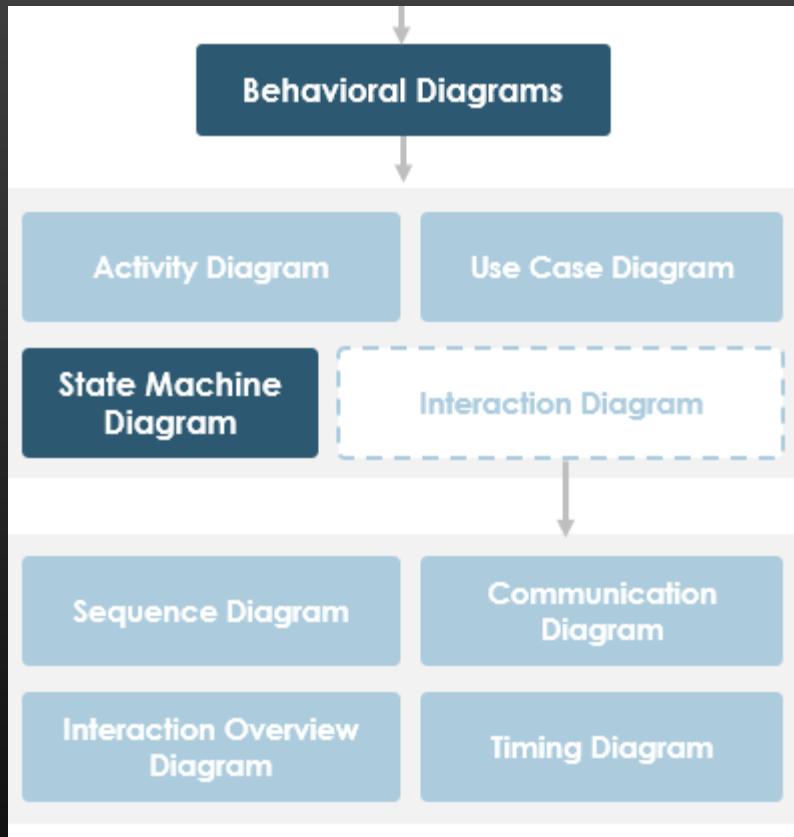
Different states are captured in this model

- Shows the different states through which a single object passes during its life
- May include the object's responses and actions

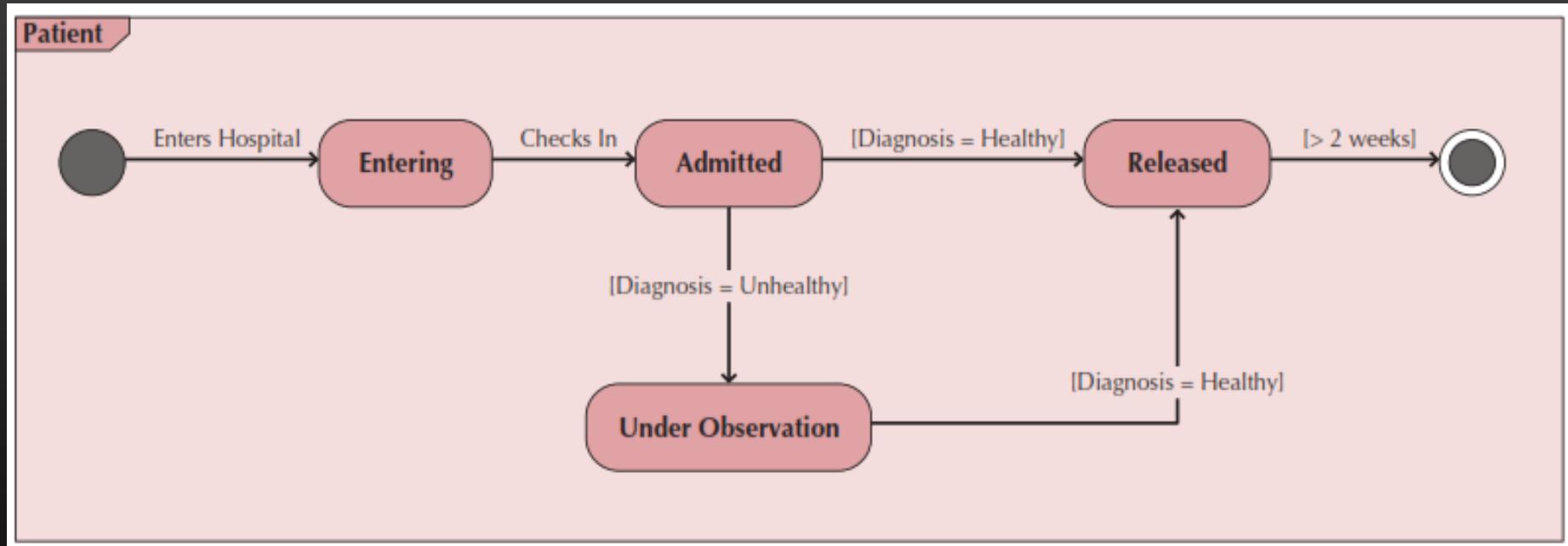
Example: patient states

- New patient—has not yet been seen
- Current patient—is now receiving treatment
- Former patient—no longer being seen or treated

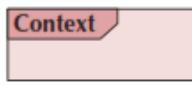
Typically used only for complex objects

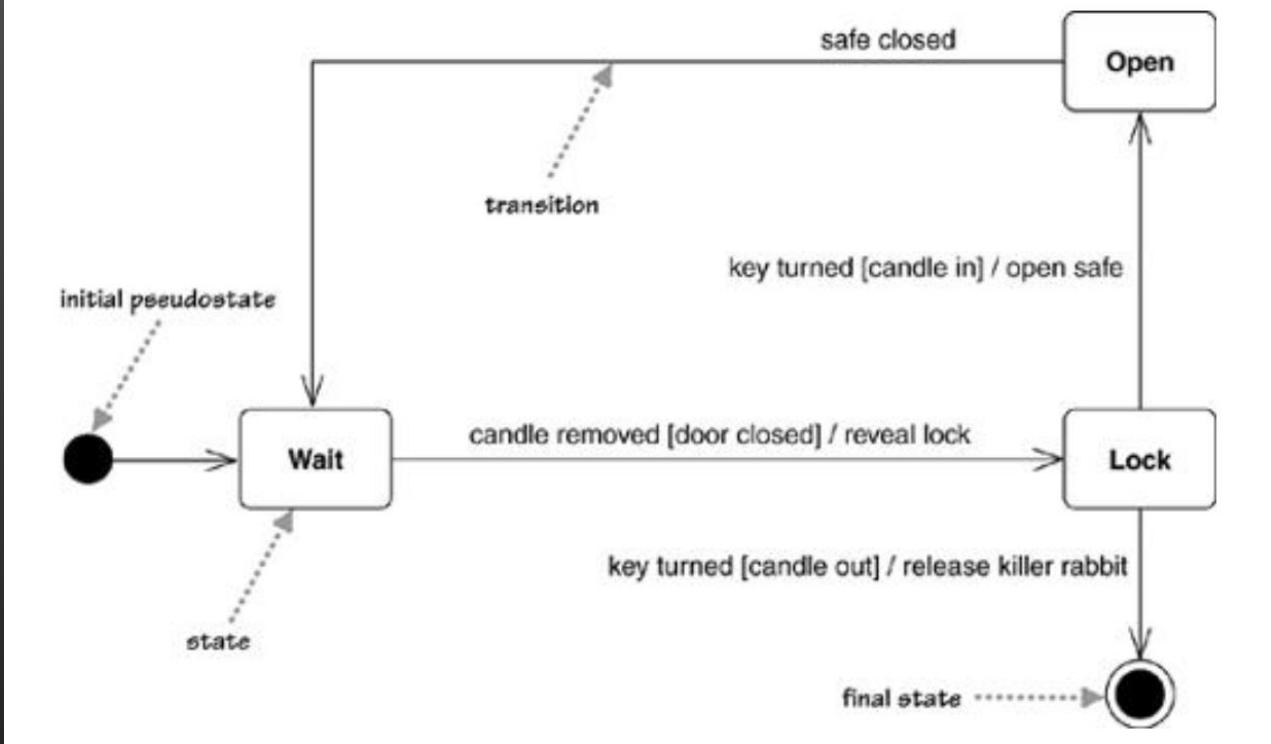


Sample State Machine



State Machine Syntax

Term and Definition	Symbol
<p>A state:</p> <ul style="list-style-type: none">■ Is shown as a rectangle with rounded corners.■ Has a name that represents the state of an object.	
<p>An initial state:</p> <ul style="list-style-type: none">■ Is shown as a small, filled-in circle.■ Represents the point at which an object begins to exist.	
<p>A final state:</p> <ul style="list-style-type: none">■ Is shown as a circle surrounding a small, filled-in circle (bull's-eye).■ Represents the completion of activity.	
<p>An event:</p> <ul style="list-style-type: none">■ Is a noteworthy occurrence that triggers a change in state.■ Can be a designated condition becoming true, the receipt of an explicit signal from one object to another, or the passage of a designated period of time.■ Is used to label a transition.	anEvent
<p>A transition:</p> <ul style="list-style-type: none">■ Indicates that an object in the first state will enter the second state.■ Is triggered by the occurrence of the event labeling the transition.■ Is shown as a solid arrow from one state to another, labeled by the event name.	
<p>A frame:</p> <ul style="list-style-type: none">■ Indicates the context of the behavioral state machine.	



The transition indicates a movement from one state to another.

Each transition has a label that comes in three parts: trigger-signature [guard]/activity. All the parts are optional.

The trigger-signature is usually a single event that triggers a potential change of state. The guard, if present, is a Boolean condition that must be true for the transition to be taken. The activity is some behavior that's executed during the transition.

Notação básica dos D. Estado

Estados → caixas

Condição em que se encontra o objeto

Transições → setas

Evolução de um estado para outro

Triggers → etiquetas

Acontecimentos relevantes que causam transições de estado

Condições de acesso

Marcador início/fim

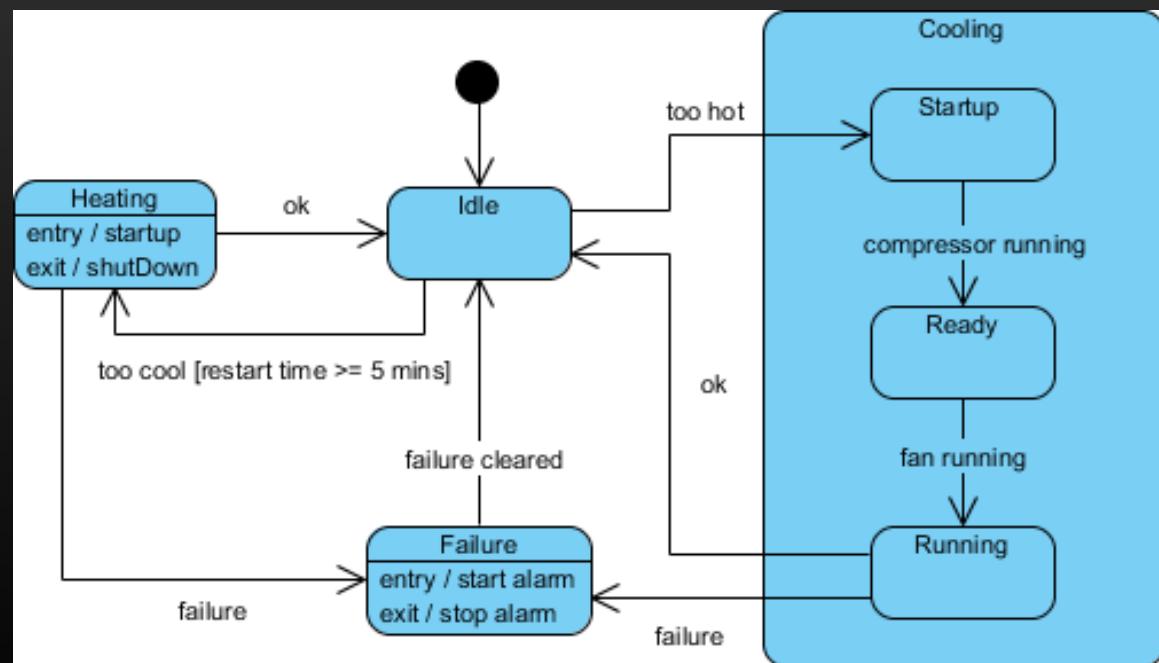
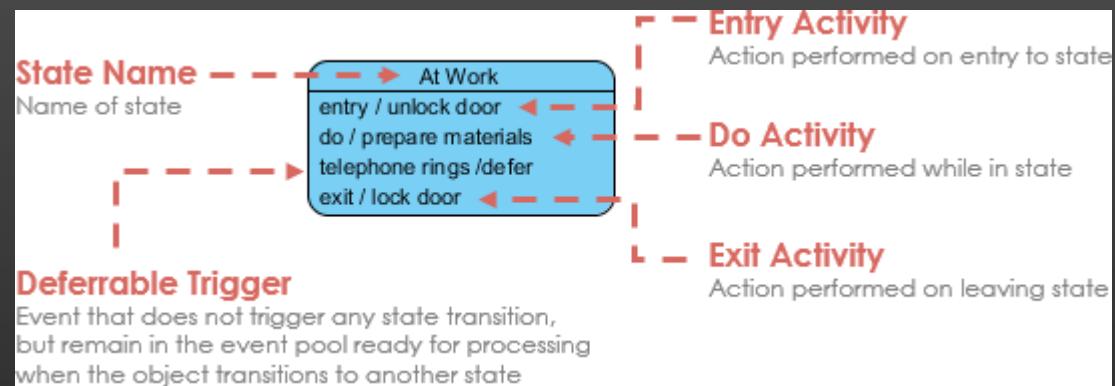
"An **event** is a significant or noteworthy occurrence e.g. a telephone receiver is taken off the hook.

A **state** is the condition of an object at a moment in time e.g. a telephone is in the state of being "idle" after the receiver is placed on the hook and until it is taken off the hook.

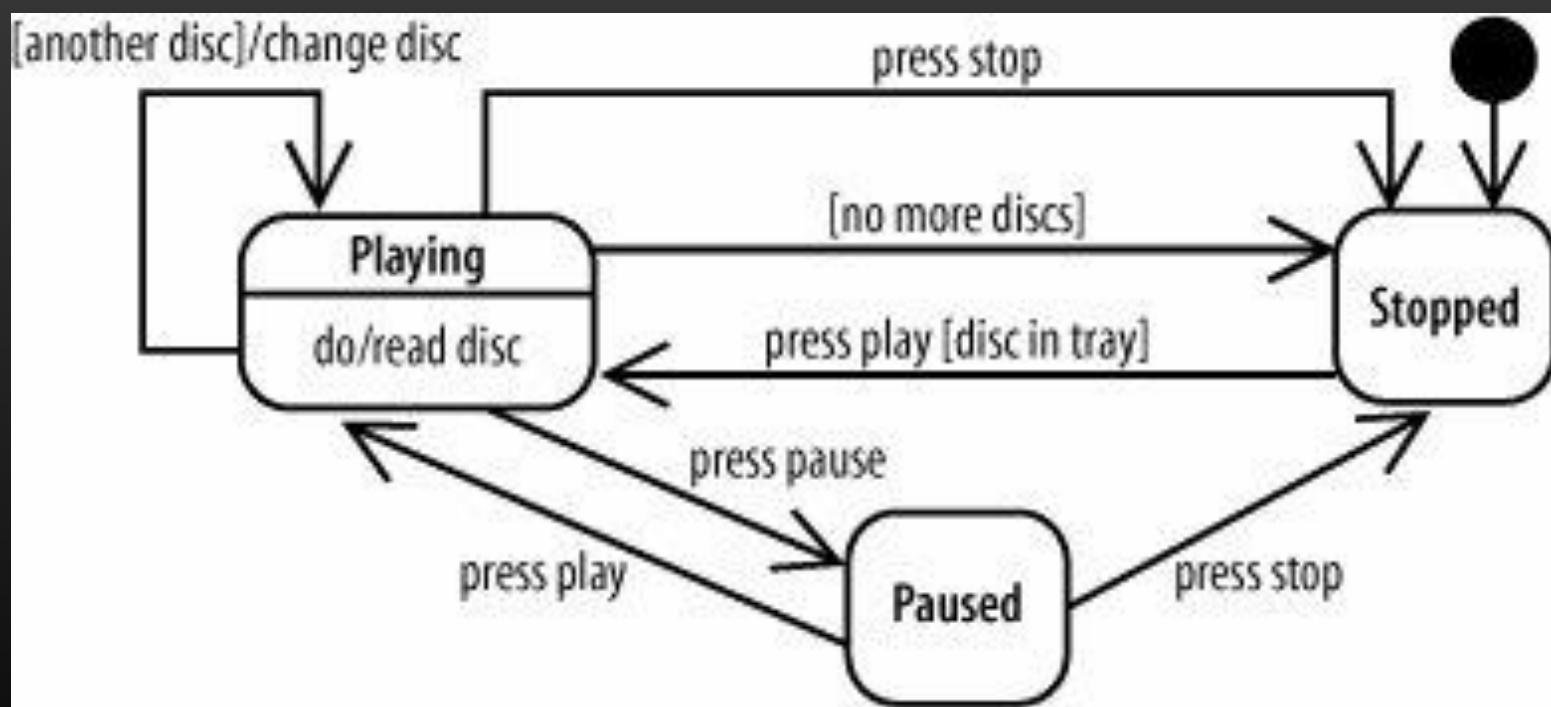
A **transition** is a relationship between two states that indicates when an event occurs e.g. when the event "off hook" occurs, transition the telephone from "idle" to "active" state.

Internal Activities

States can react to events without transition, using internal activities: putting the event, guard, and activity inside the state box itself



Exemplo



Quando usar?

Objetos com comportamento dependente do estado

Exemplos:

Controlador de um dispositivo físico (hw)

Lógica de objetos de negócio
(Venda, Reserva,...)

Protocolos de comunicação

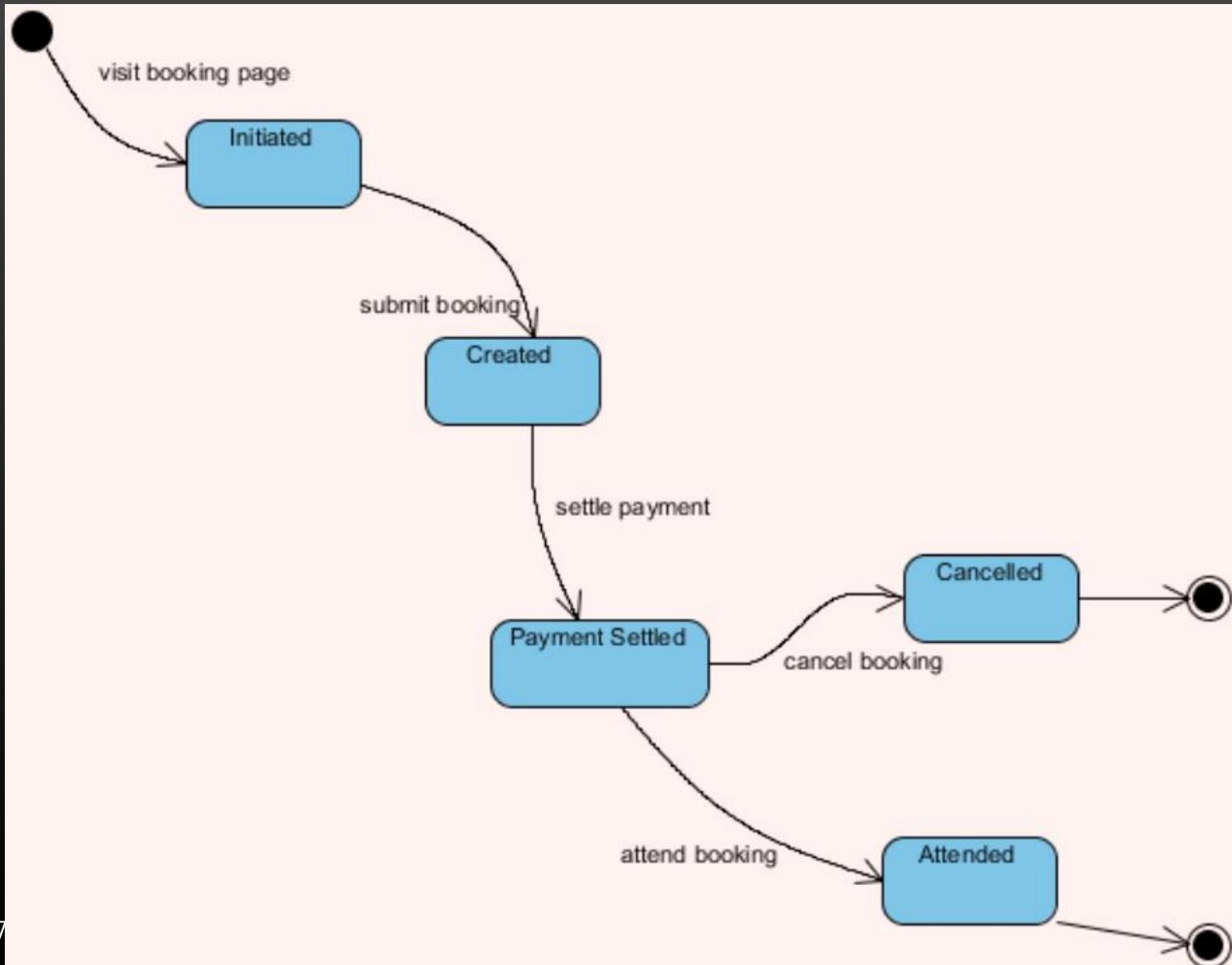
Coordenadores do fluxo da interface gráfica

No modelo do domínio:

Caracterizar os estados de uma classe complexa

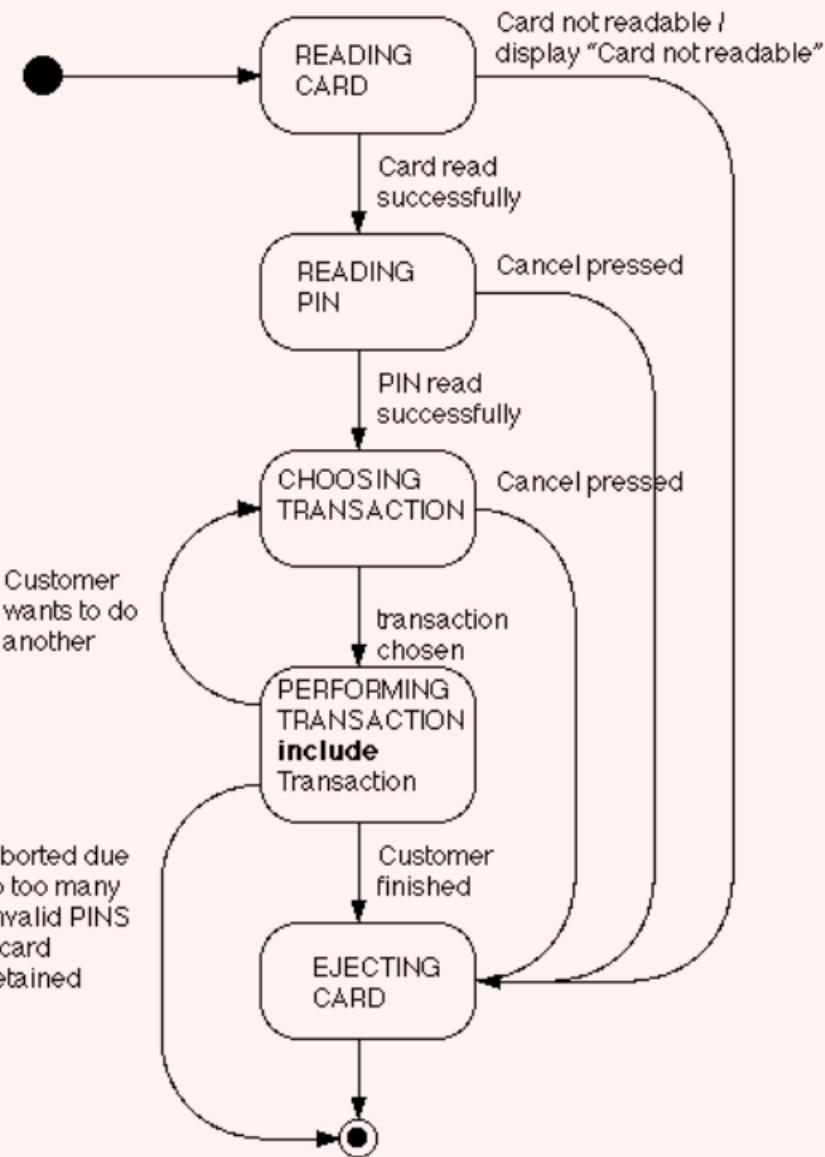
e.g.: reserva, inscrição,...

Modeling the states of an appointment



ATM controller

State-Chart for One Session



Readings & references

Core readings	Suggested readings
<ul style="list-style-type: none">• [Dennis15] - Chap. 6• <u>What is Sequence Diagram?</u>, VisualParadigm docs• <u>What is Communication Diagram?</u>, VisualParadigm docs• <u>What is State Machine Diagram?</u>, VisualParadigm docs	<ul style="list-style-type: none">• [Larman04] - Chap. 10, Chap. 15.

47006- ANÁLISE E MODELAÇÃO DE SISTEMAS

Desenho por objetos: UML na visualização do código

Ilídio Oliveira

v2020/11/20, TP14a

Objetivos de aprendizagem

Interpretar diagramas de classes (de Código)

Representar construções de código (em Java) nos modelos da UML

Interpretar diagramas de sequência que modelam colaboração entre objetos)

Visualização do código Java com a UML

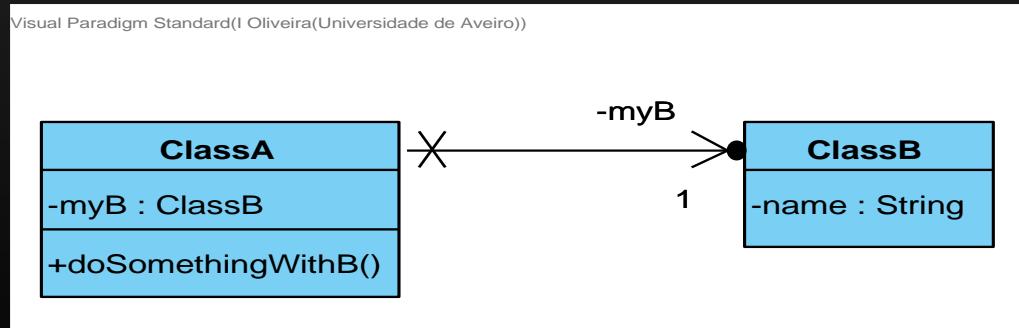
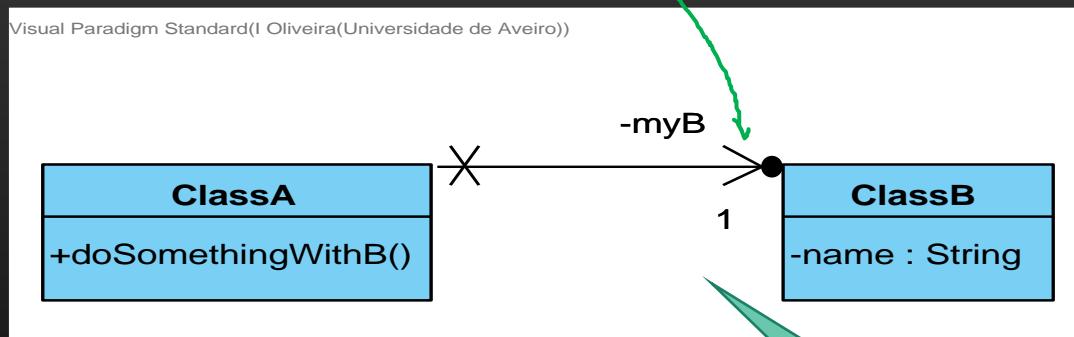
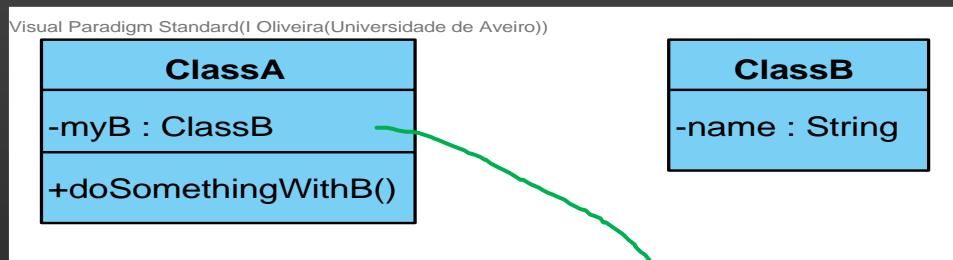
Visualização de Código com classes

```
public class ClassA {  
  
    private ClassB myB;  
  
    public void doSomethingWithB() {  
        // todo;  
    }  
}
```

Visual Paradigm Standard(I Oliveira(Universidade



Visualização do código com classes



Modelos semanticamente equivalentes.
Mostrar os atributos como associações evidencia os relacionamentos.

```

public class ClientsPortfolio {
    private ArrayList<Client> myClientsList;

    public ClientsPortfolio() {
        myClientsList = new ArrayList<>();
    }

    public void addClient(Client newClient) {
        this.myClientsList.add(newClient);
    }

    public int countClients() {
        return this.myClientsList.size();
    }
}

```

Classe

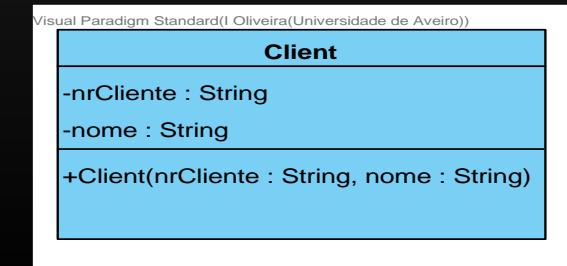
Atributo (neste caso, é uma lista de objetos do tipo Client)

Operações (que podem requerer parâmetros e produzir um valor de retorno)



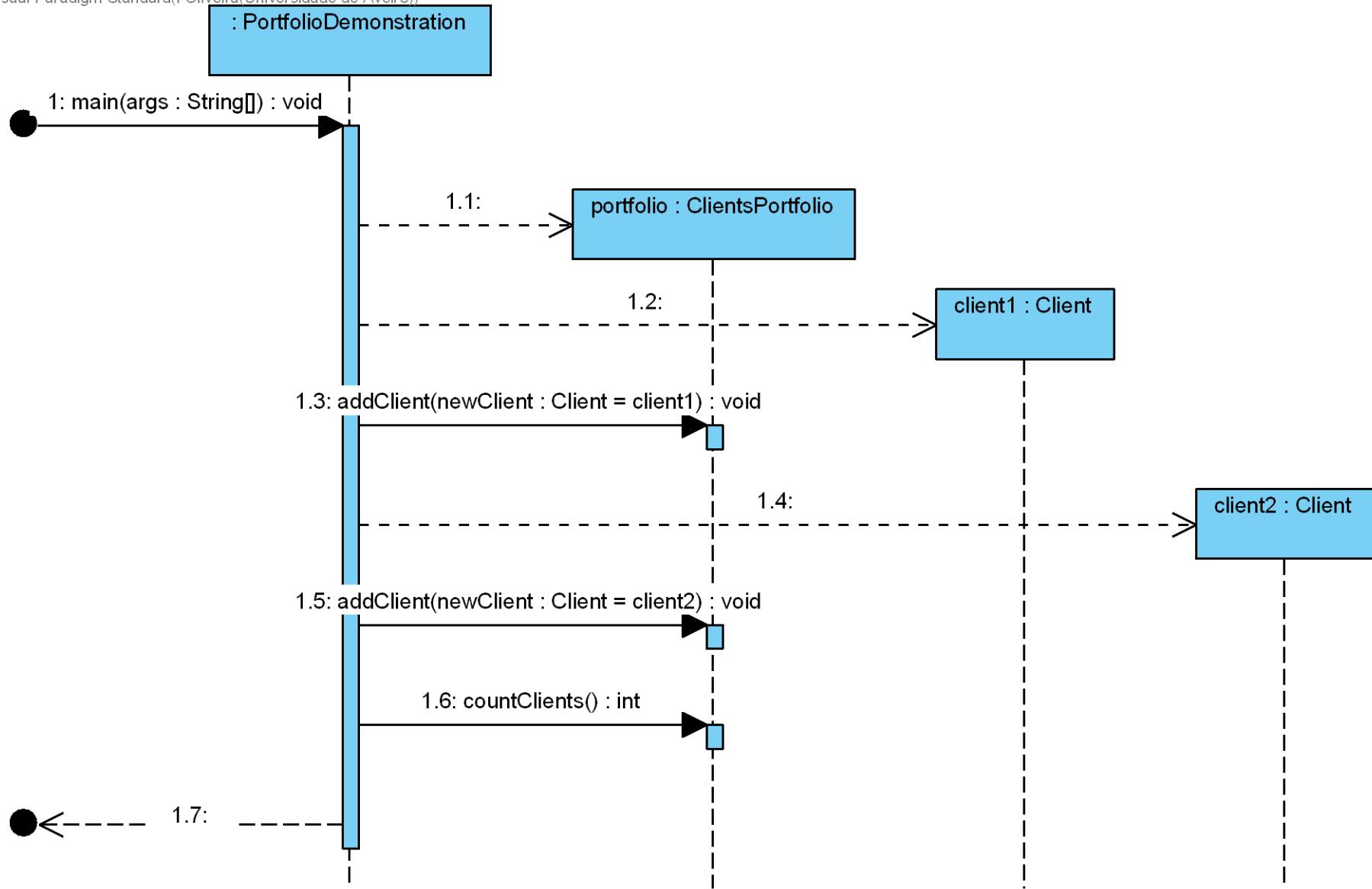
Objetos enviam mensagens

```
public class PortfolioDemonstration {  
    public static void main(String[] args) {  
        // obter um novo objeto da classe ClientsPortfolio  
        ClientsPortfolio portfolio = new ClientsPortfolio();  
  
        // obter um novo objeto da classe Cliente e adicioná-lo ao porfolio  
        Client client1= new Client( "C103", "Logistica Tartaruga");  
        portfolio.addClient( client1 );  
          
        Client client2 = new Client( "C104", "Jose, Maria & Jesus Lda");  
        portfolio.addClient( client2 );  
          
        System.out.println( "Clients count: " + portfolio.countClients() );  
    }  
}
```

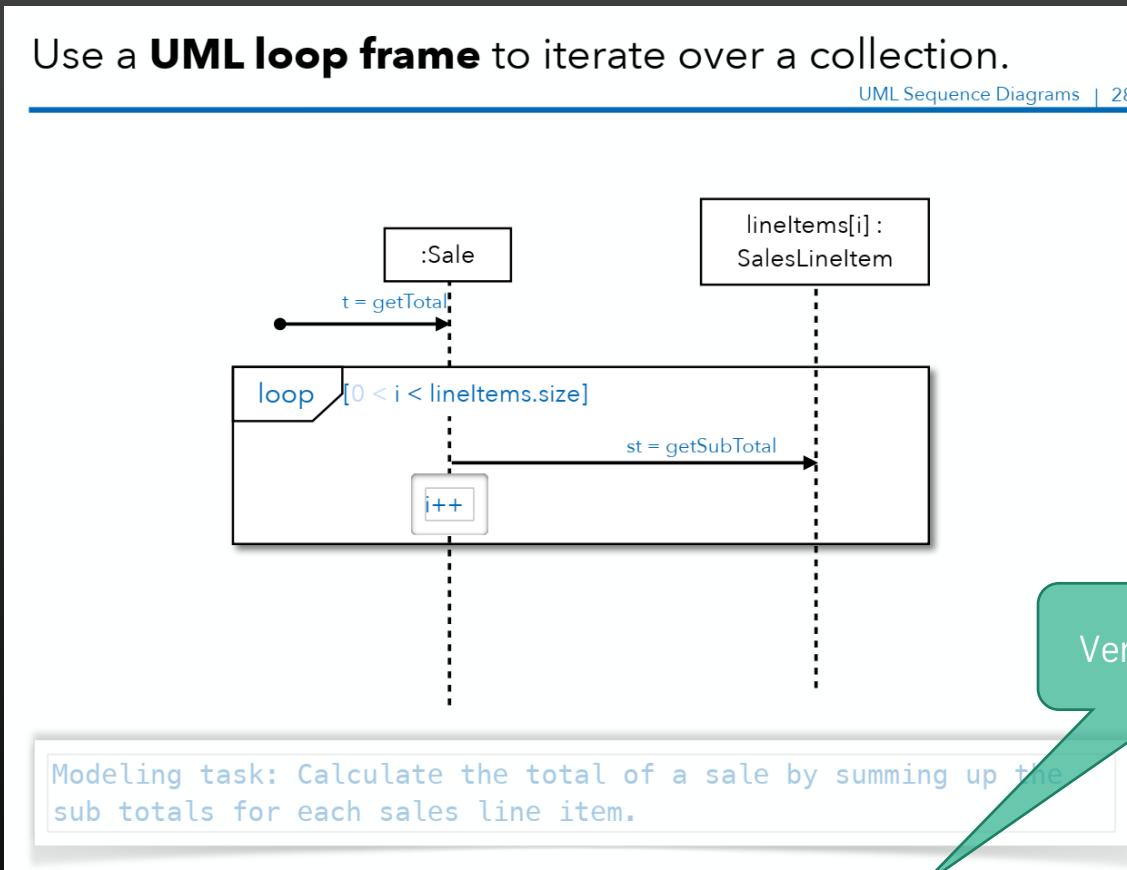


...que podem ser vistas num modelo dinâmico

Visual Paradigm Standard(I Oliveira(Universidade de Aveiro))



Alguns exemplos adicionais



http://stg-tud.github.io/eise/WS18-SE-08-Modeling-dynamic_Part.pdf

UML para “visualizar” o código: estrutura e interação

O objetos Java colaboram para realizar objetivos

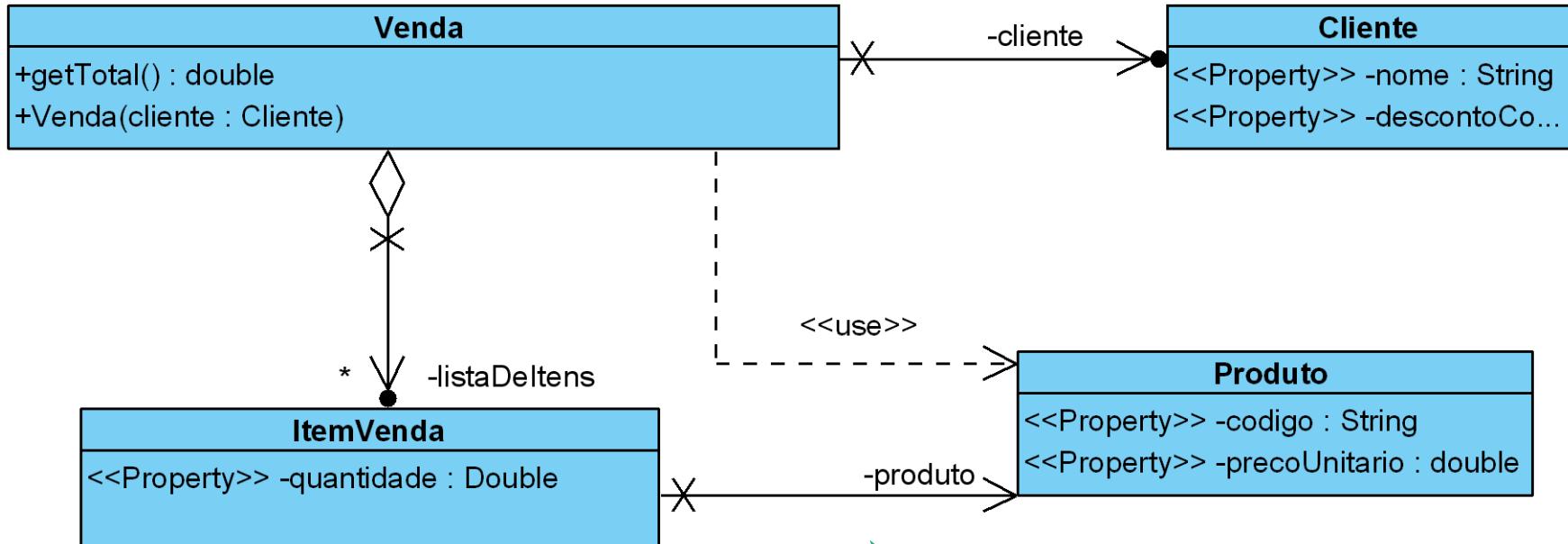
```
public class Encomenda {  
  
    private Cliente cliente;  
    private Collection<ItemDaEncomenda> listaDeItens = new ArrayList<>();  
  
    public double getTotal() {  
  
        double total= 0.0;  
  
        Produto produto;  
        for (ItemDaEncomenda item : listaDeItens) {  
            produto = item.getProduto();  
            total += produto.getPrecoUnitario() * item.getQuantidade();  
        }  
        total = total * (1- this.cliente.getDescontoComercial());  
        return total;  
    }  
  
    public Encomenda(Cliente cliente) {  
        super();  
        this.cliente = cliente;  
    }  
}
```

| Oliveira

Quais são as classes envolvidas?
O que podemos descobrir sobre o seu
“esqueleto” (operações e assinaturas,
atributos)?

Vista estrutural (definição das classes)

Visual Paradigm Standard(I Oliveira(Universidade de Aveiro))

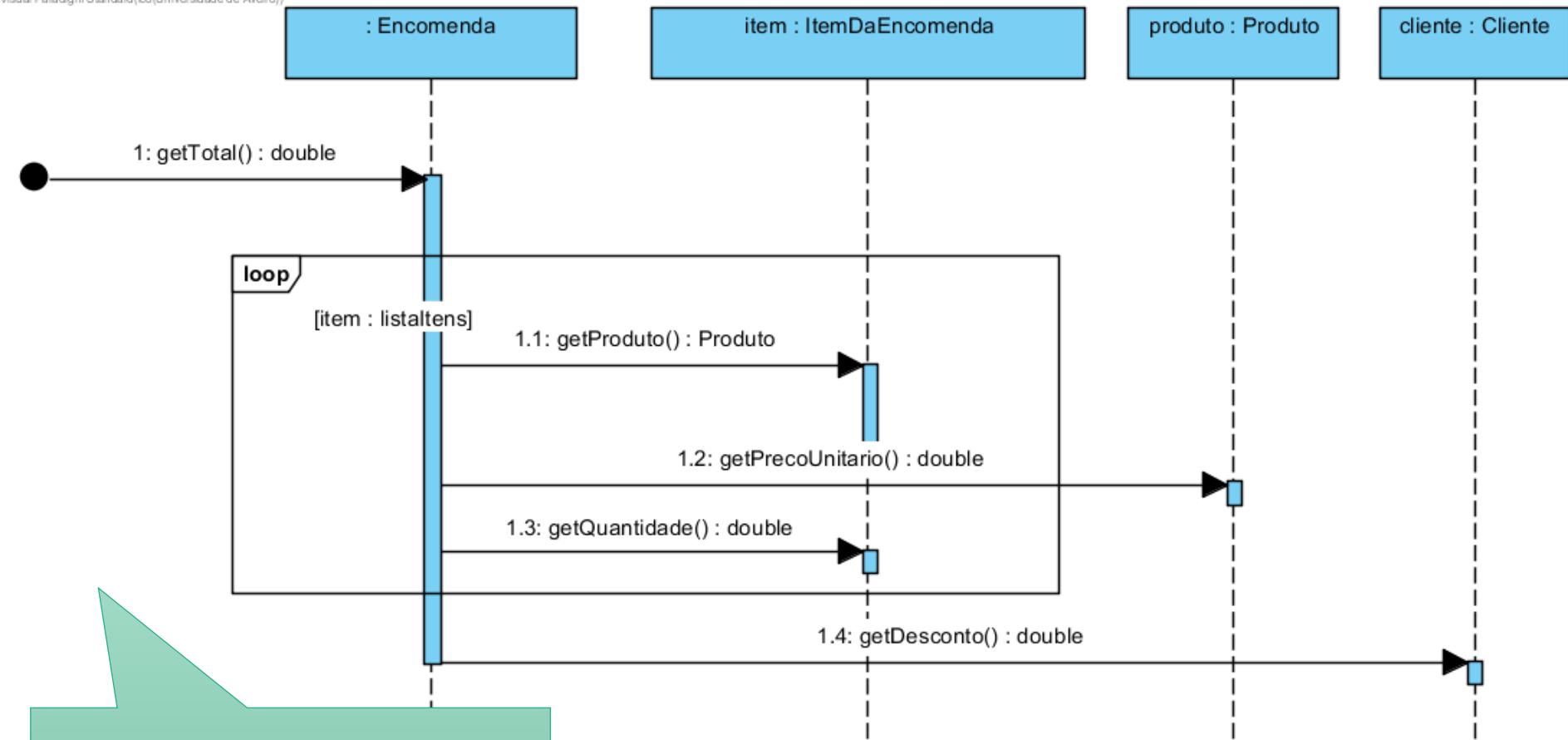


Os atributos que implicam um relacionamento entre classes estão representados como associações.

O esteriótipo `<<Property>>` marca atributos que têm `getter` e `setter`

Vista dinâmica (interações entre objetos)

Visual Paradigm Standard (co/Universidade de Aveiro)



Qual a colaboração entre objetos necessária para implementar Encomenda#getTotal()?

Referências

Core readings	Suggested readings
<ul style="list-style-type: none">• [Dennis15] - Chap. 8	<ul style="list-style-type: none">• [Larman04] - Chap. 17 and 18• Slides by M. Eichberg : SSD and OO-Design

47006- ANÁLISE E MODELAÇÃO DE SISTEMAS

Desenho por objetos: dos resultados da análise para o código

Ilídio Oliveira

v2020/11/20, TP14b

Objetivos de aprendizagem

Explicar como os casos de utilização podem ser usados para orientar as atividades de desenho

Explicar os princípios do baixo acoplamento e alta coesão em OO

Cenário: modelar a colaboração que ocorre num restaurante

Cliente entra no restaurante e chama o Empregado (de mesa).

Cliente pede informações sobre as opções do dia.

Empregado anota o novo pedido, com os pratos pedidos.

Empregado avisa Cozinha (informa a nova comanda)

Cozinheiro confecciona o pedido, usando os ingredientes necessários.

Cozinheiro disponibiliza os pratos confeccionados, quando pronto.

Empregado entrega pedido ao cliente.

Vista estrutural: que “tipos de coisas” (i.e.: classes)?

Papéis de pessoas?

Lugares e pontos de serviço?

Transações de bens/serviços?

Itens numa transação?

...

Vista dinâmica: como é que os objetos (instâncias) colaboram?

Quais os objetos que participam?

O que é que cada objeto solicita de outro?



“Tipos de coisas”: alguns candidatos

Papéis de pessoas?

Cliente, Empregado, Cozinheiro

Lugares e pontos de serviço?

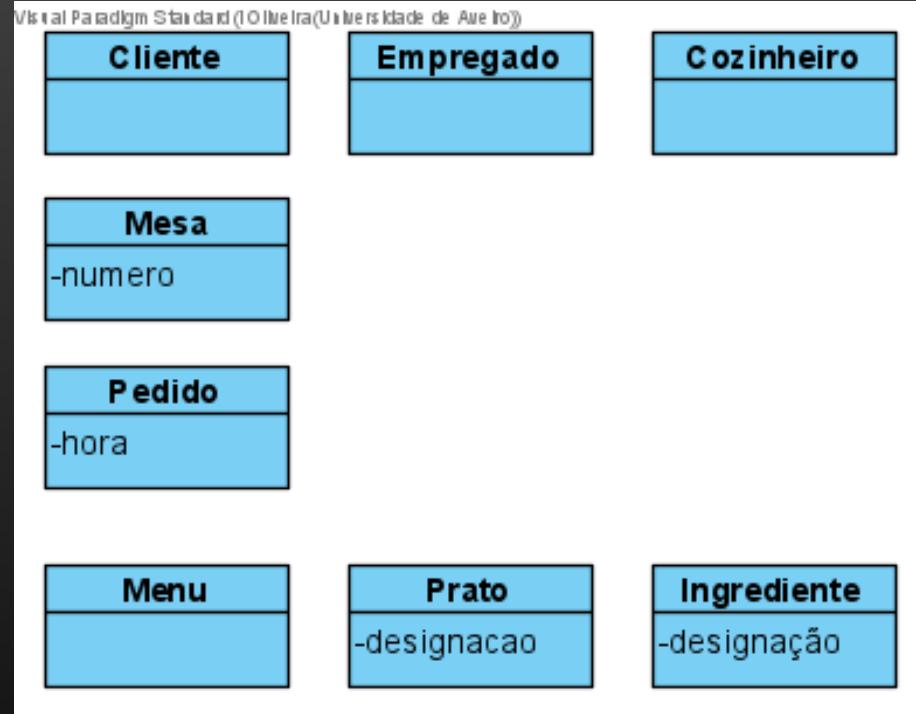
Mesa? Restaurante? Sala?

Transações de bens/serviços?

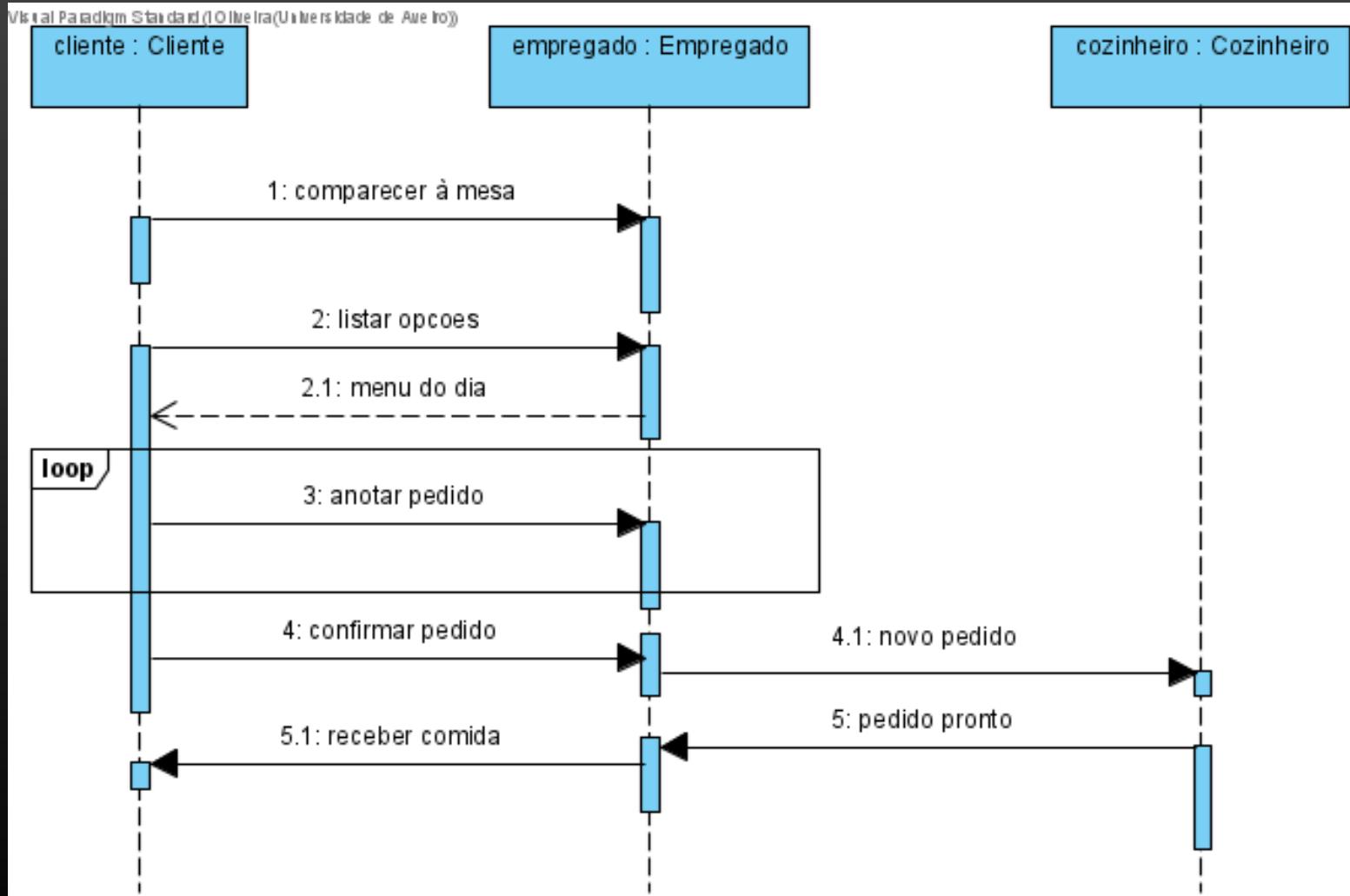
Pedido; comanda/talão?

Itens numa transação?

Prato/Opção; Menu; Ingredientes?



Interação entre “participantes”

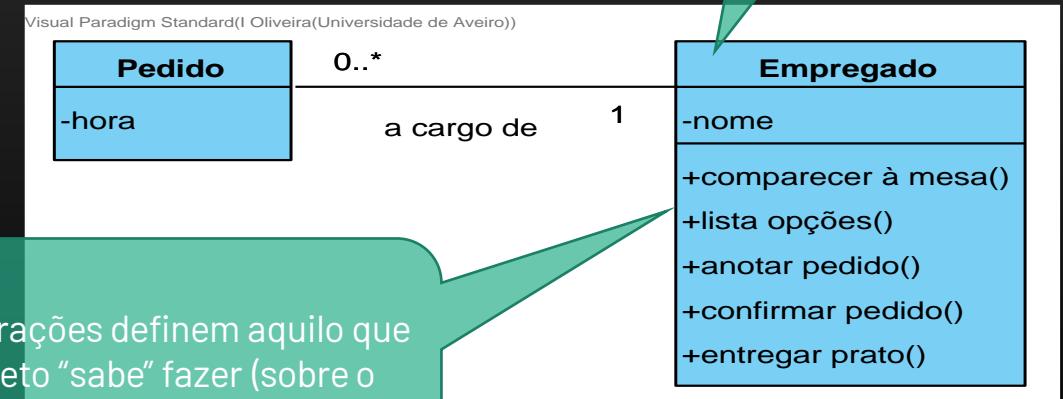


Os diagramas de classes e os de interação “distribuem” responsabilidades

Análise por classes define
dois grupos de
responsabilidades:

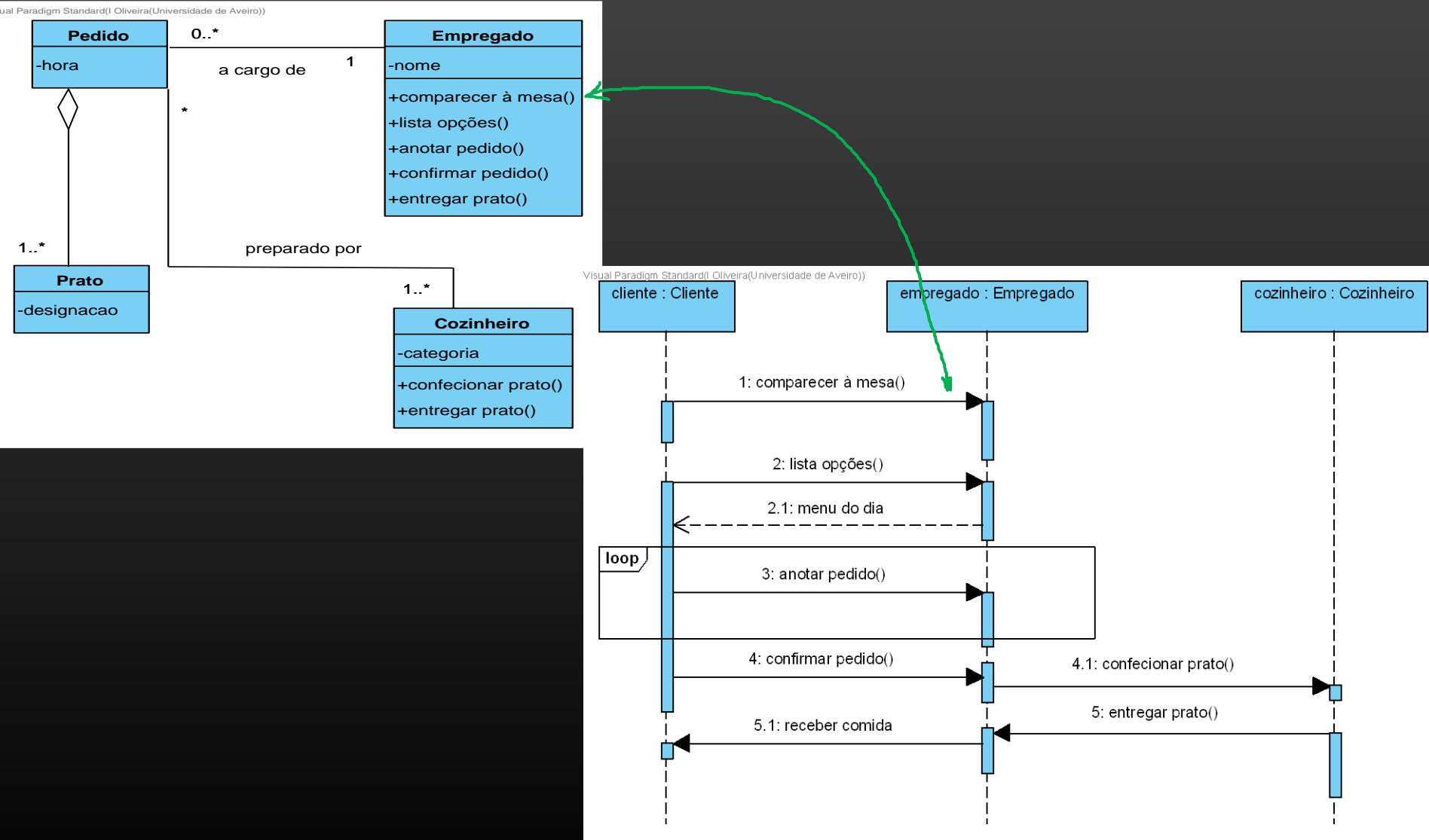
- O que é que cada tipo é responsável por conhecer/guardar
- O que é que cada tipo é responsável por fazer

Atributos e objetos associados definem o âmbito do que o objeto guarda.



As operações definem aquilo que o objeto “sabe” fazer (sobre o conhecimento que guarda)

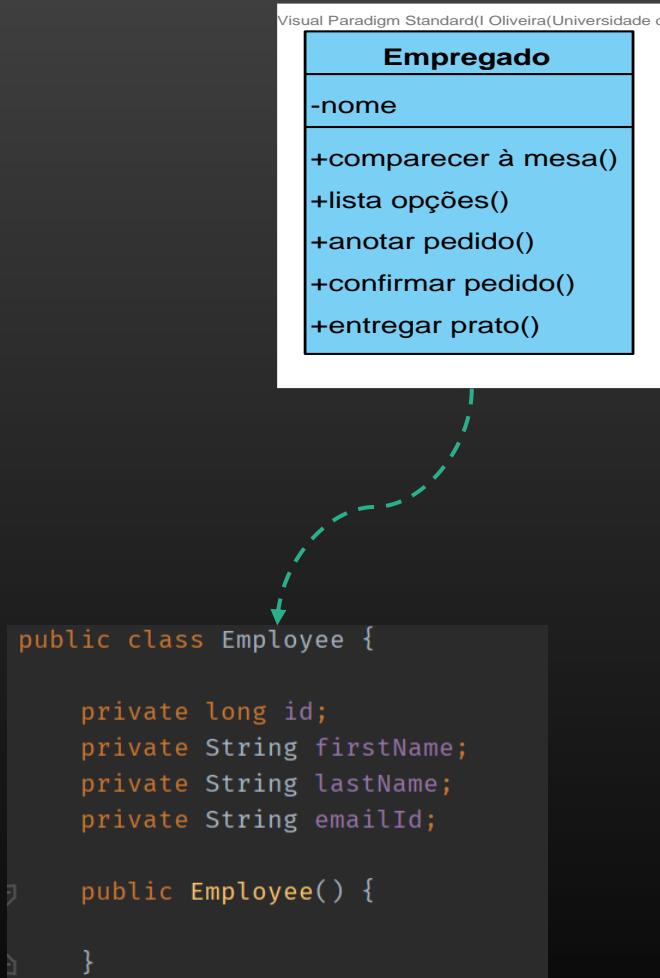
Vista complementares



Este raciocínio, no domínio do problema, pode ser aplicado para o código?

Aproveitar o modelo do domínio para “inspirar” a implementação do código!

- Modelo do domínio explora o vocabulário do problema
- Modelo do domínio explica os relacionamentos relevantes e algumas regras (formas de associar objetos)
- A implementação não usa diretamente as representações do domínio do problema...



Em código....

A classe passa a representar uma entidade do software

- Pode ser o “mesmo conceito” do domínio
- Mas pode ser outro tipo de entidade, com significado apenas para o software

Mesmo mecanismo mental

- classificar em tipos (=classes)
- a classe funciona como uma unidade modular, especializada, com conhecimento e operações limitadas
- os objetos são instâncias de classes
- os objetos colaboram “em rede”!

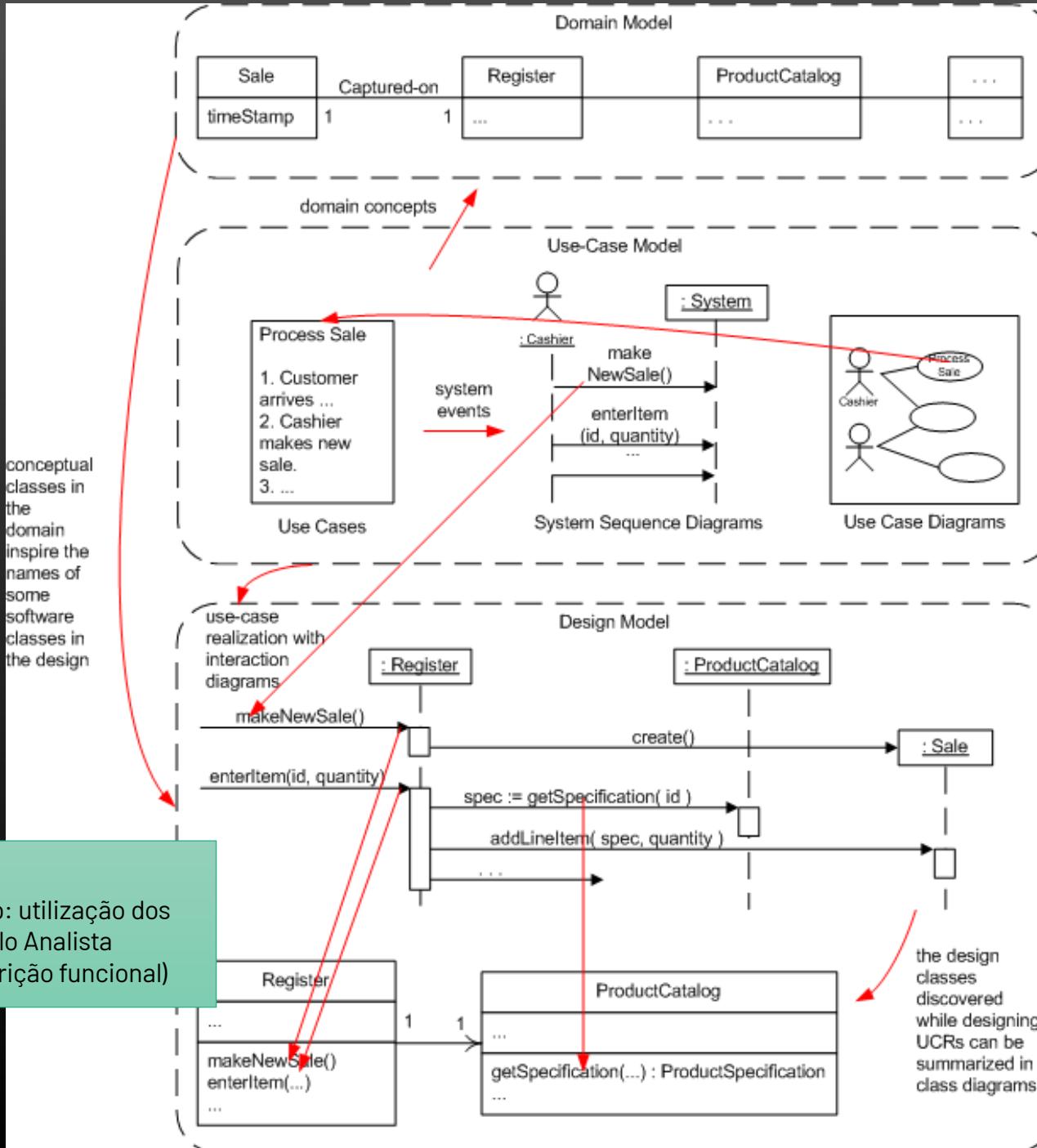
Não é um conceito do domínio, mas uma entidade/unidade que faz sentido no programa.

Atributos e objetos associados definem o âmbito do que o objeto guarda.

	• EmployeeController	
	repository	EmployeeRepository
	• all() CollectionModel<EntityModel<Employee>>	
	• newEmployee(Employee)	Employee
	• one(Long)	EntityModel<Employee>
	• replaceEmployee(Employee, Long)	Employee
	• deleteEmployee(Long)	void

As operações definem aquilo que o objeto “sabe” fazer (sobre o conhecimento que guarda/tem acesso)

Como evoluir os resultados da análise para o desenho?



@Larman:

Da análise para o desenho: utilização dos resultados preparados pelo Analista
(modelo do domínio, descrição funcional)

From the use cases into software design

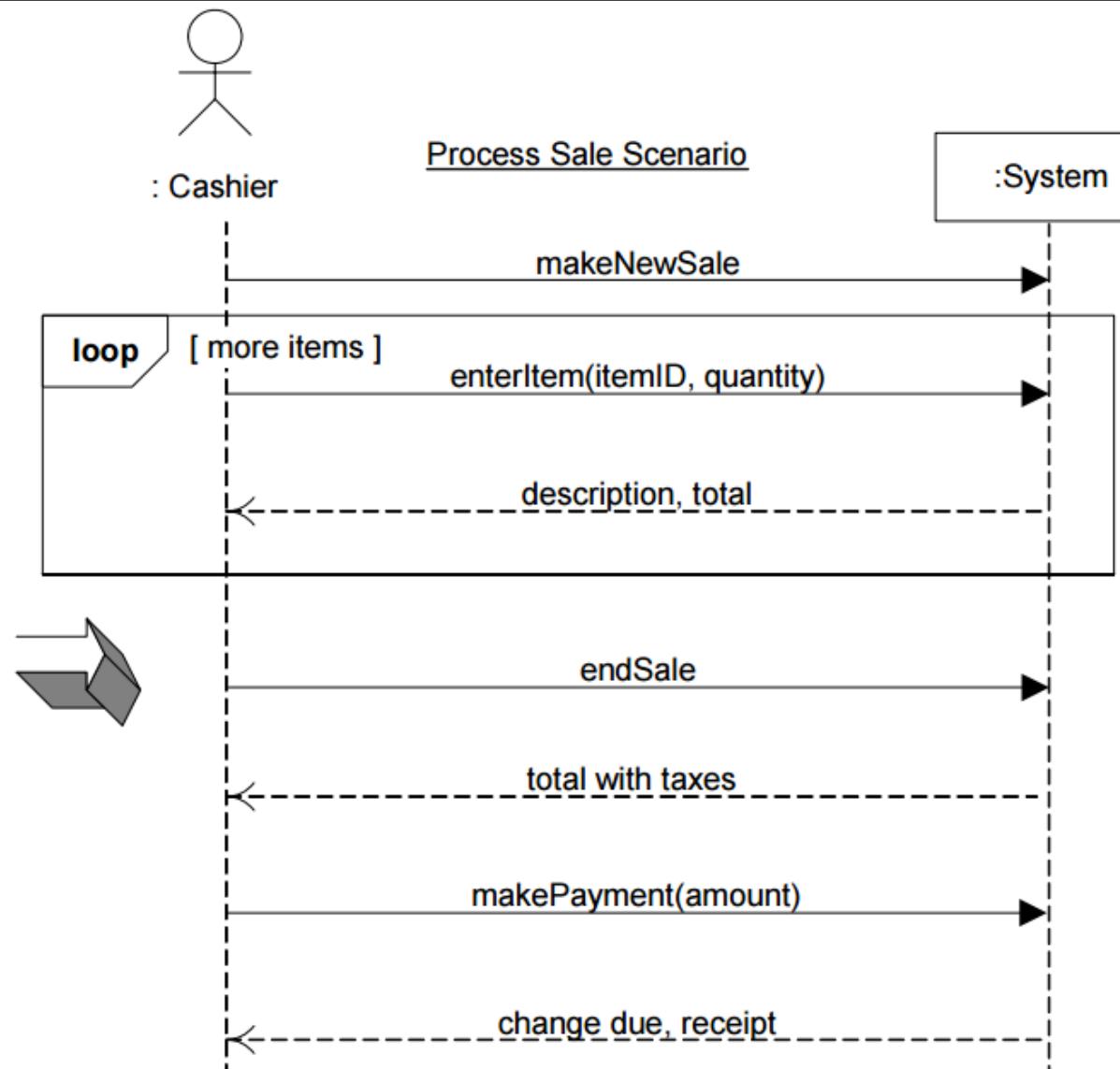
Simple cash-only Process Sale scenario:

1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.

Cashier repeats steps 3-4 until indicates done.

5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.

...



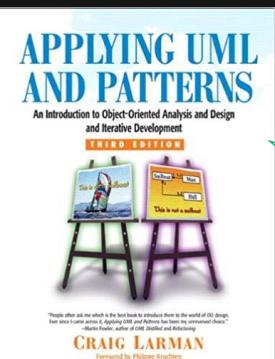
DSS

A new system sequence diagram (SSD) for each CaU

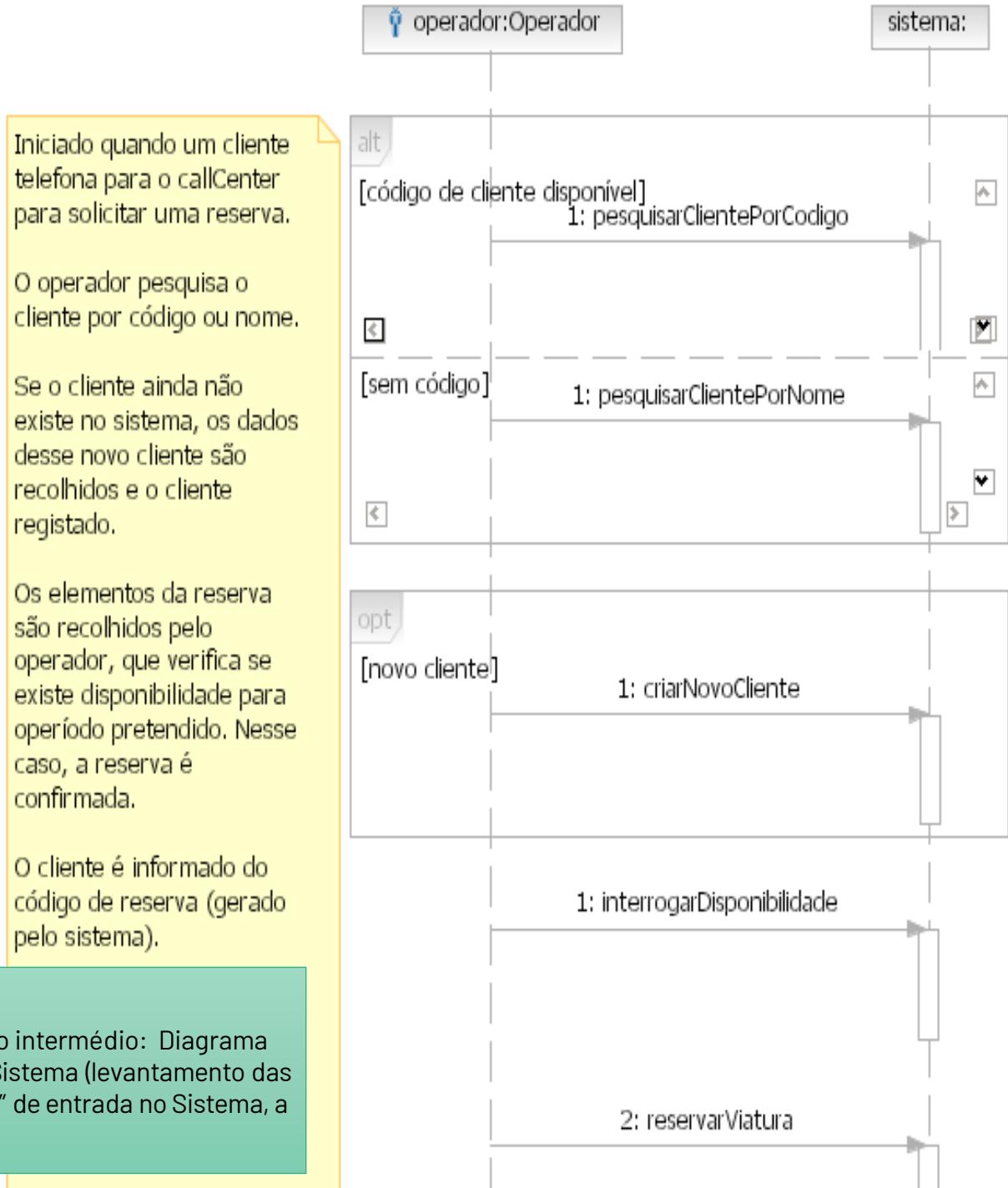
A lifeline for the primary actor (or actors) and another for the system.

The system is modeled by a class that represents it globally

Optionally, you can include the text of the CaU description



In Larman:
Passo de transição intermédio: Diagrama de Sequência de Sistema (levantamento das funções “externas” de entrada no Sistema, a partir do CaU)



In Larman:

Passo de transição intermédio: Diagrama de Sequência de Sistema (levantamento das funções “externas” de entrada no Sistema, a partir do CaU)

APPLYING UML AND PATTERNS

An Introduction to Object-Oriented Analysis and Design
and Iterative Development

THIRD EDITION



"People often ask me which is the best book to introduce them to the world of OO design. Ever since I came across it, Applying UML and Patterns has been my unreserved choice."

-Martin Fowler, author of UML Distilled and Refactoring

CRAIG LARMAN

Foreword by Philippe Kruchten

I Oliveira

Iniciado quando um cliente telefona para o callCenter para solicitar uma reserva.

O operador pesquisa o cliente por código ou nome.

Se o cliente ainda não existe no sistema, os dados desse novo cliente são recolhidos e o cliente registado.

Os elementos da reserva são recolhidos pelo operador, que verifica se existe disponibilidade para período pretendido. Nesse caso, a reserva é confirmada.

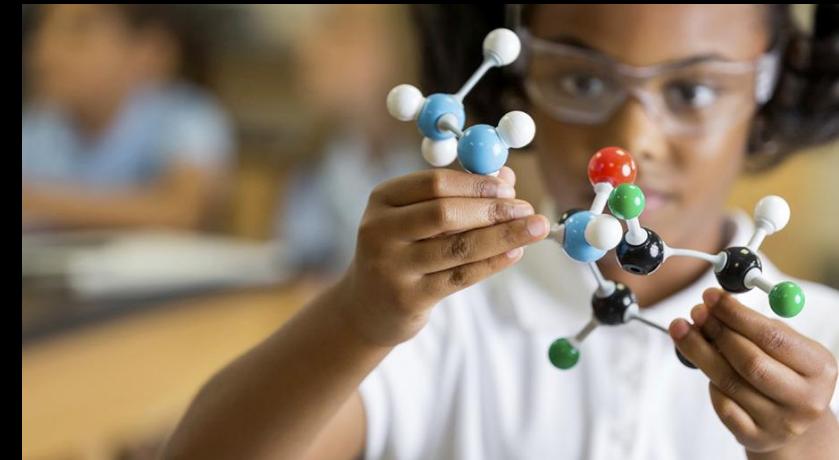
O cliente é informado do código de reserva (gerado pelo sistema).



Operação de sistema: pesquisarClientePorCodigo

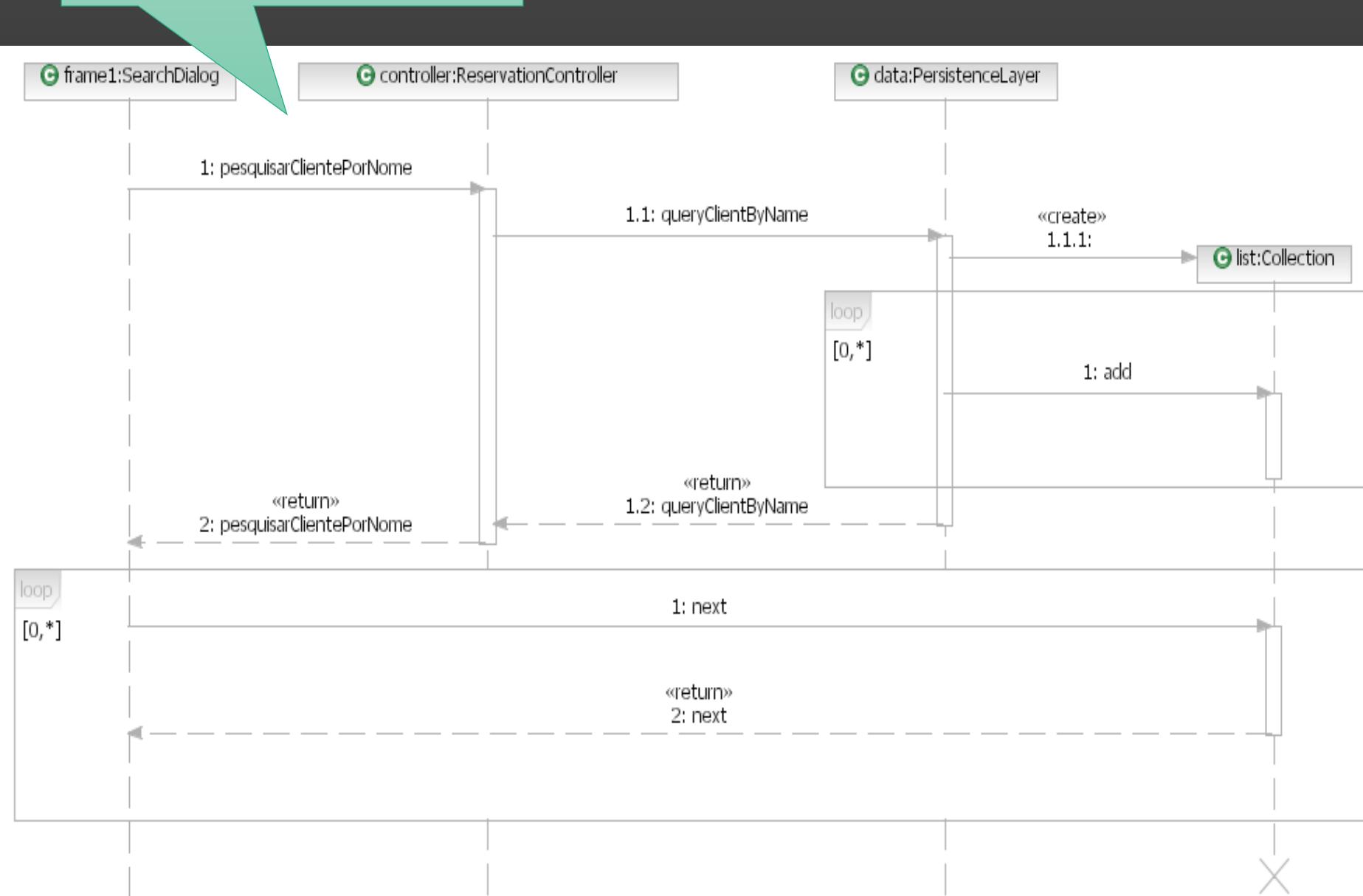
Quem deve controlar a interação? E a visualização?

Que outros objetos são necessários? Com que métodos?

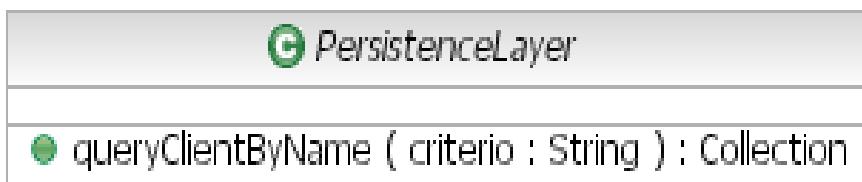
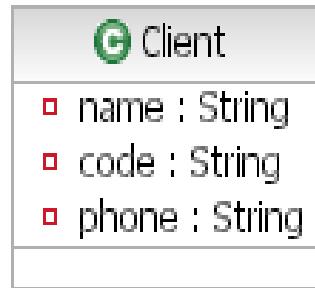


Qual é a ordem das mensagens entre objetos?

Expansão de cada operação de sistema:
qual a colaboração concreta de objetos que
a realiza? Processo de descoberta.



Os diagramas de interacção ajudam a distribuir responsabilidades → encontrar os métodos

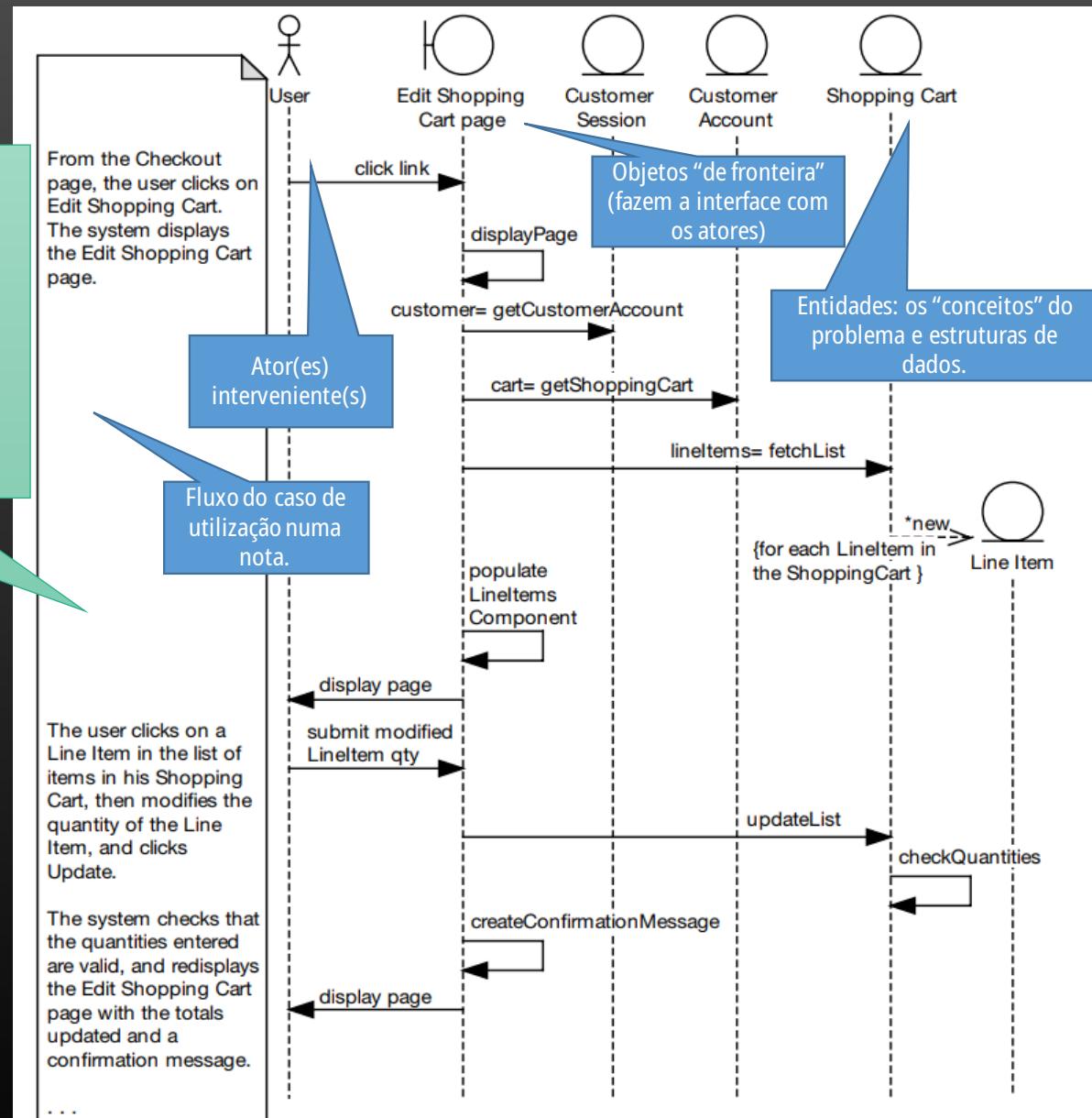
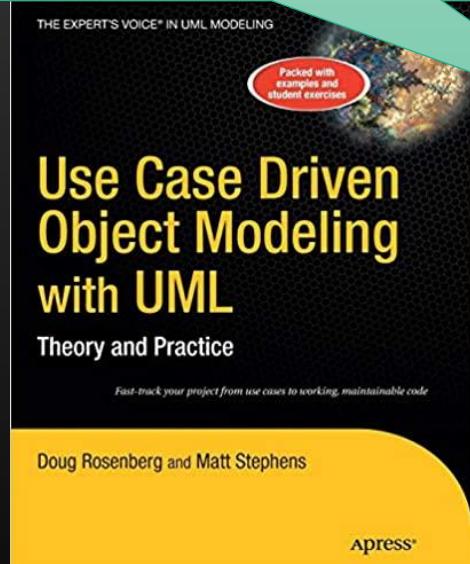


In Rosenbeg:

Da análise para o desenho: utilização dos resultados preparados pelo Analista para desenvolver o "modelo de robustez"

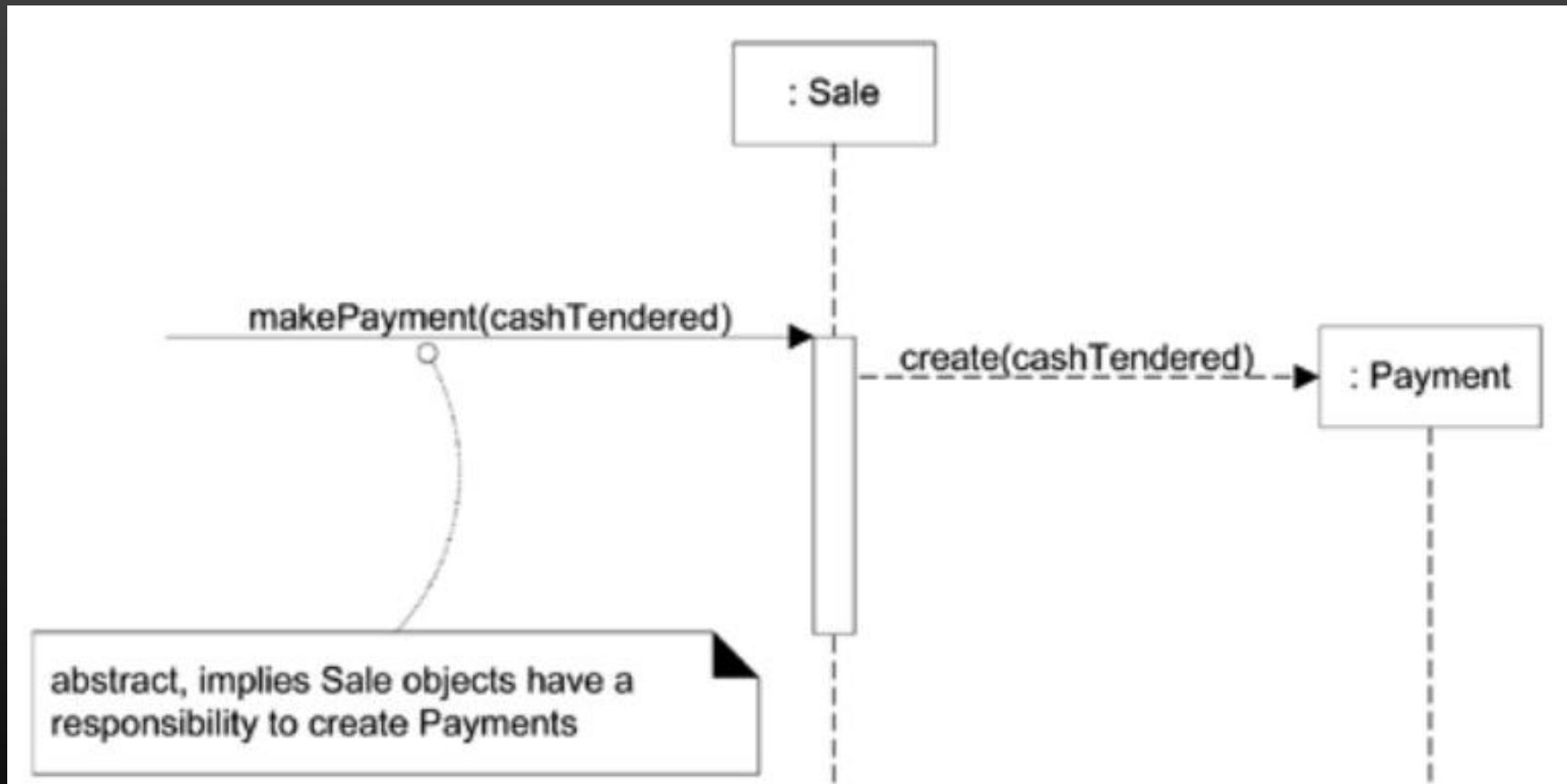
Três categorias de classes:

- Fronteiras
- Controladores
- Entidades



**“Pensar por objetos” é aplicar princípios
para “distribuir” as responsabilidades
pelas classes**

Ao desenhar um diagrama de interação, estamos a atribuir responsabilidades



Como atribuir responsabilidades aos objetos?

Não é uma ciência exata

Por isso temos...

Bom e mau desenho

Desenho eficiente e ineficiente

Desenho elegante e tenebroso...

Implicações na facilidade de manter e evoluir uma solução



“Desenho”, no ciclo de engenharia do software, significa o processo de planear/idealizar o código. A pessoa que lidera o desenho é o “arquiteto de software”.

Sempre que, mesmo num problema simples, começamos por nos interrogar: quais as classes? Como é que elas vão estar interdependentes?, estamos a “desenhar” o o código (fazendo escolhas).

Responsabilidades de um objeto

Fazer

Fazer alguma coisa sobre o seu estado, como calcular alguma coisa, criar objetos,...

Iniciar uma ação em outros objetos

Coordenar/controlar as ações em outros objetos

Saber

Conhecer o seu estado interno ("escondido")

Conhecer os objetos relacionados

Critérios para o desenho

- Um conjunto de métricas para avaliar o desenho
- Acoplamento (*coupling*): refere-se ao grau de proximidade/interdependência da relação entre classes
- Coesão (*cohesion*): refere-se ao grau com que os atributos e métodos de uma classe estão relacionados internamente.

Uma classe que tem muitos atributos que são objetos de outro tipo, tem um *coupling* elevado: **depende de** outras classes.

Uma classe que mantém, internamente, detalhes das Vendas e dos Produtos vendidos, não é coesa: em vez de ter um **foco único**, está a assumir várias responsabilidades.

Coupling

Mede a força/intensidade da dependência de uma classe de outras

A classe C1 está emparelhada com C2 se precisa de C2, direta ou indiretamente.

Uma classe que depende de outras 2 tem um “coupling” mais baixo que uma que dependa de 8.

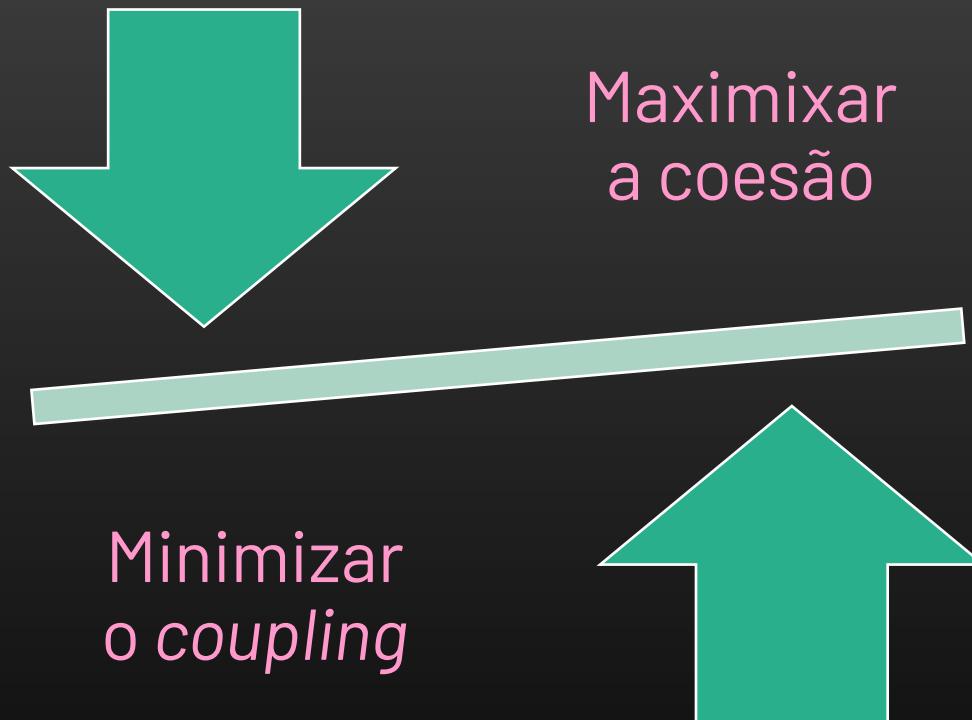
Coesão

Mede a força/intensidade do relacionamento dos elementos de uma classe entre si.

Todas as operações e dados de uma classe devem estar natural e diretamente relacionados com o conceito que a classe modela

Uma classe deve ter um foco único (vs. responsabilidades desgarradas)

Critérios gerais para um melhor desenho



Common Forms of Coupling in Java

- Type X has an attribute that refers to a type Y instance or type Y itself

```
class X{ private Y y = ...}  
class X{ private Object o = new Y(); }
```

- A type X object calls methods of a type Y object

```
class Y{f();}  
class X{ X(){new Y.f();}}
```

- Type X has a method that references an instance of type Y (E.g. by means of a parameter, local variable, return type,...)

```
class Y{}  
class X{ X(Y){...}}  
class X{ Y f(...)}  
class X{ void f(){Object y = new Y();}}
```

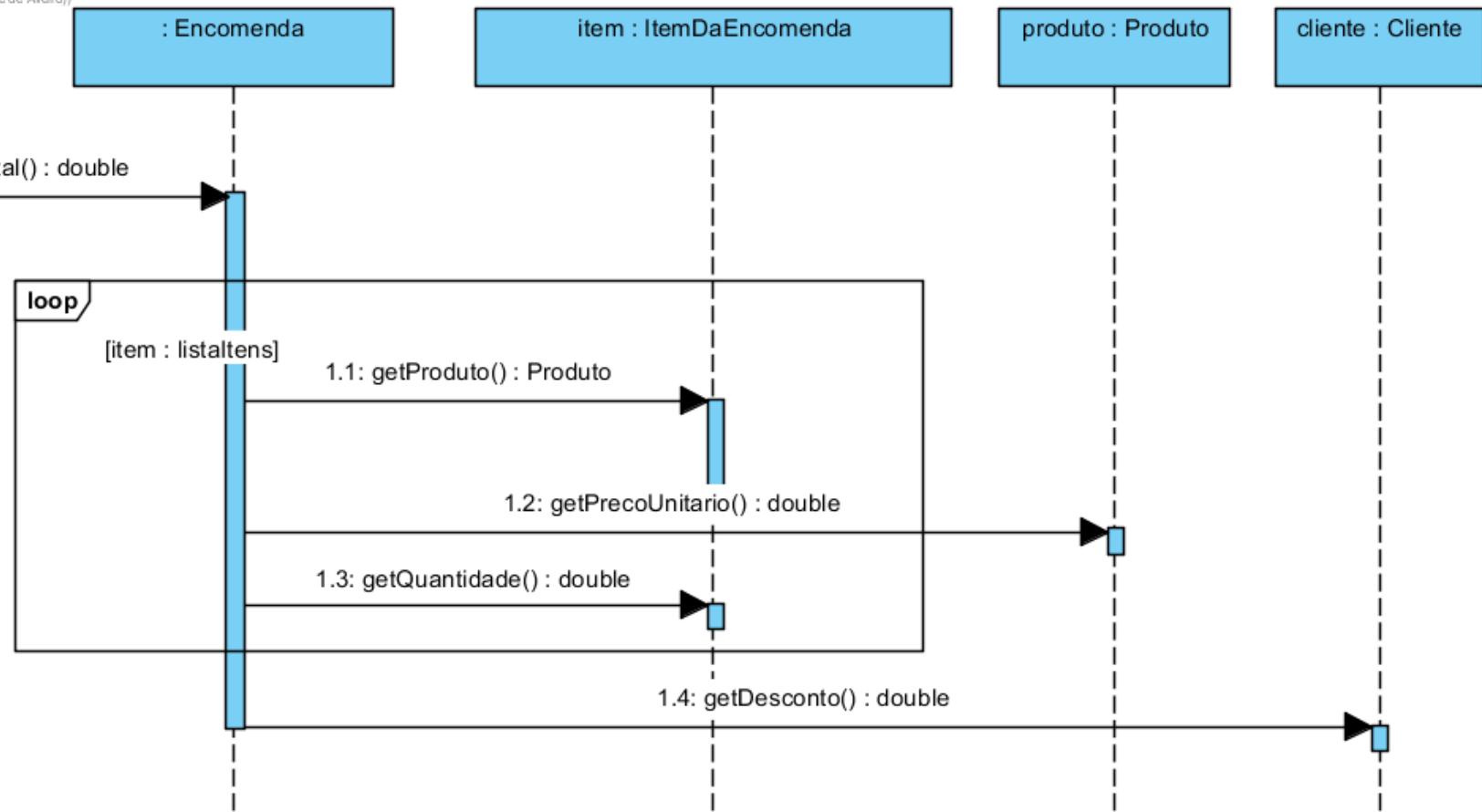
- Type X is a subtype of type Y

```
class Y{}  
class X extends Y{}
```

- ...

Coupling de interação

Visual Paradigm Standard (co/Universidade de Aveiro)



Coupling...

Coupling	Features	Desirability
Data	A & B communicate by simple data only	High (use parameter passing & only pass necessary info)
Stamp	A & B use a common type of data	Okay (but should they be grouped in a data abstraction?)
Control (activating)	A transfers control to B by procedure call	Necessary
Control (switching)	A passes a flag to B to tell it how to behave	Undesirable (why should A interfere like this?)
Common environment	A & B make use of a shared data area (global variables)	Undesirable (if you change the shared data, you have to change both A and B)
Content	A changes B's data, or passes control to the middle of B	Extremely Foolish (almost impossible to debug!)

Types of Interaction Coupling

Level	Type	Description
Good	No Direct Coupling	The methods do not relate to one another; that is, they do not call one another.
	Data	The calling method passes a variable to the called method. If the variable is composite (i.e., an object), the entire object is used by the called method to perform its function.
	Stamp	The calling method passes a composite variable (i.e., an object) to the called method, but the called method only uses a portion of the object to perform its function.
	Control	The calling method passes a control variable whose value will control the execution of the called method.
	Common or Global	The methods refer to a “global data area” that is outside the individual objects.
Bad	Content or Pathological	A method of one object refers to the inside (hidden parts) of another object. This violates the principles of encapsulation and information hiding. However, C++ allows this to take place through the use of “friends.”

Source: These types were adapted from Meilir Page-Jones, *The Practical Guide to Structured Systems Design*, 2nd ed. (Englewood Cliffs, NJ: Yarden Press, 1988); and Glenford Myers, *Composite/Structured Design* (New York: Van Nostrand Reinhold, 1978).

Coesão



Qual é a hipótese que oferece maior coesão?

Qual é o que é mais fácil de avariar/dar problemas?

De que é que precisamos 80% das vezes?....

Coesão

Uma classe, objeto ou método coesos têm um único "foco"

Coesão a nível dos métodos

O método executa mais do que um propósito/operação?

Realizar mais do que uma operação é mais difícil de entender e implementar

Coesão a nível da classe

Os atributos e métodos representam um único objeto?

As classes não devem misturar papéis, domínios ou objetos

Coesão na especialização/generalização

As classes numa hierarquia devem mostrar uma relação "tipo-de"

Cohesion (methods)

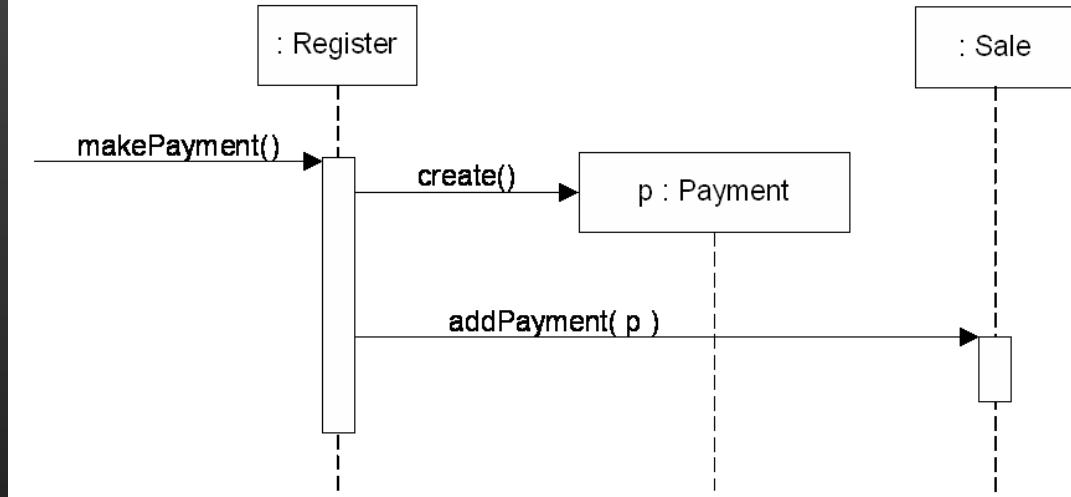
Cohesion	Features	Desirability
Data	all part of a well defined data abstraction	Very High
Functional	all part of a single problem solving task	High
Sequential	outputs of one part form inputs to the next	Okay
Communicational	operations that use the same input or output data	Moderate
Procedural	a set of operations that must be executed in a particular order	Low
Temporal	elements must be active around the same time (e.g. at startup)	Low
Logical	elements perform logically similar operations (e.g. printing things)	No way!!
Coincidental	elements have no conceptual link other than repeated code	No way!!

Types of Method Cohesion

Level	Type	Description
Good	Functional	A method performs a single problem-related task (e.g., calculate current GPA).
	Sequential	The method combines two functions in which the output from the first one is used as the input to the second one (e.g., format and validate current GPA).
	Communicational	The method combines two functions that use the same attributes to execute (e.g., calculate current and cumulative GPA).
	Procedural	The method supports multiple weakly related functions. For example, the method could calculate student GPA, print student record, calculate cumulative GPA, and print cumulative GPA.
	Temporal or Classical	The method supports multiple related functions in time (e.g., initialize all attributes).
	Logical	The method supports multiple related functions, but the choice of the specific function is chosen based on a control variable that is passed into the method. For example, the called method could open a checking account, open a savings account, or calculate a loan, depending on the message that is send by its calling method.
Bad	Coincidental	The purpose of the method cannot be defined or it performs multiple functions that are unrelated to one another. For example, the method could update customer records, calculate loan payments, print exception reports, and analyze competitor pricing structure.

Source: These types were adapted from Page-Jones, *The Practical Guide to Structured Systems*, and Myers, *Composite/Structured Design*.

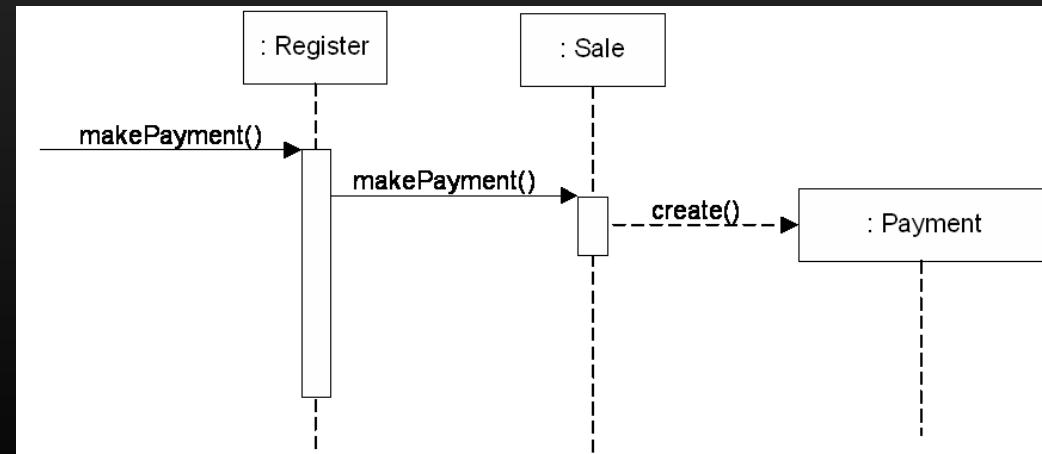
Exemplos



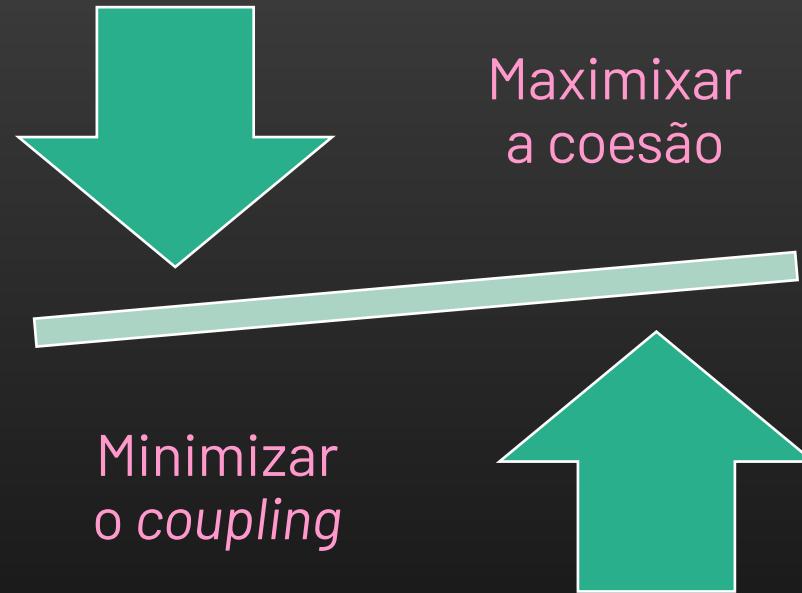
Qual é a hipótese que oferece maior coesão?

No primeiro caso, `: Register` conhece informação de pagamentos e de vendas.

No segundo caso, `: Register` apenas se relaciona com Venda (e não precisa de representar a lógica dos pagamentos)



É preciso balancear

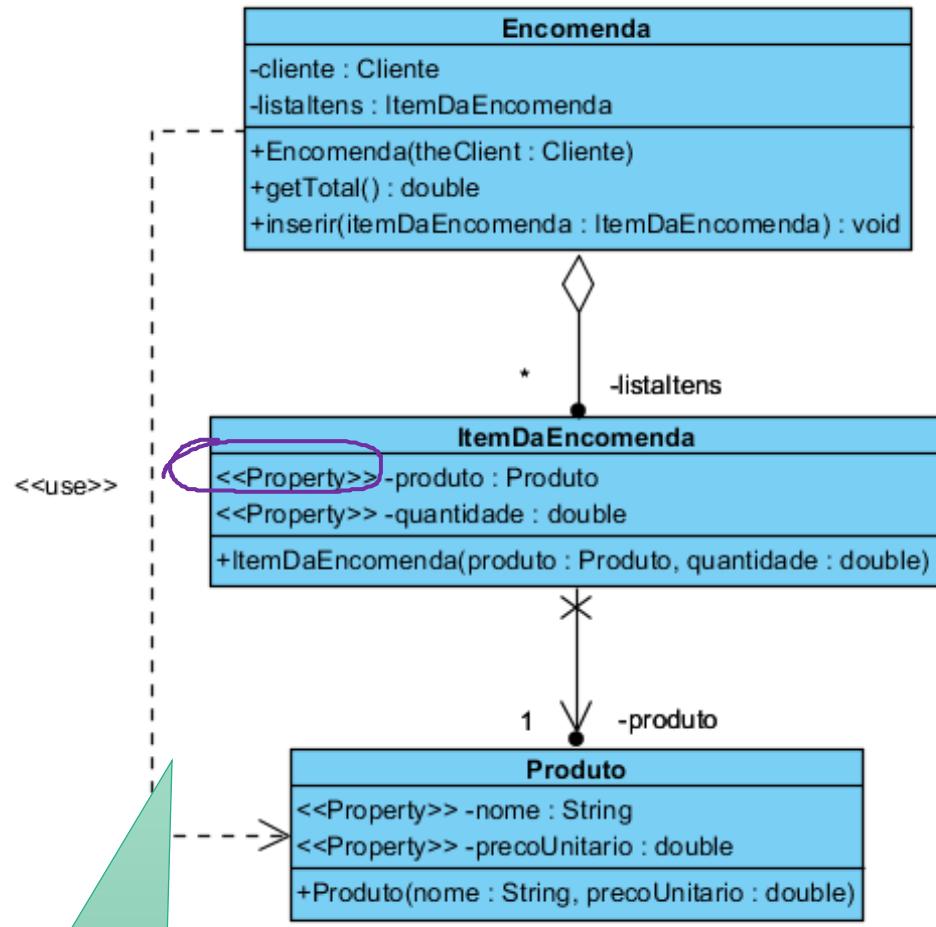


Por hipótese, a situação com melhor *coupling* (mais baixo possível) seria ter uma única classe na solução.

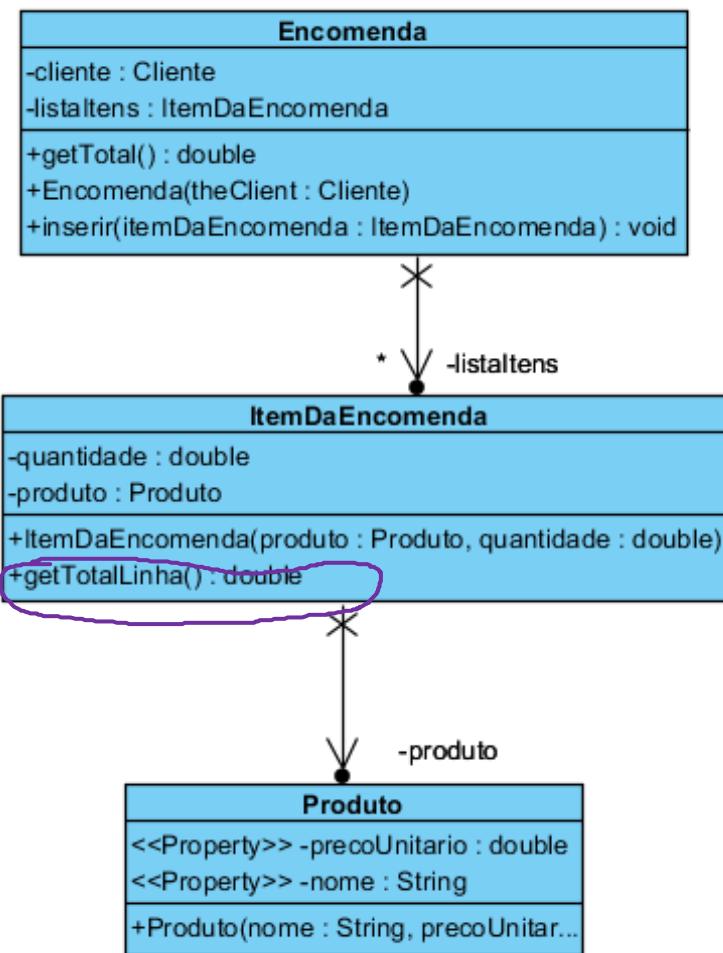
Mas essa seria a pior escolha do ponto de vista da coesão.

Avaliação de coupling/coesão: exemplo da encomenda

Visual Paradigm Standard (icq|Universidade de Aveiro)



getTotal() consulta o preço unitário definido em Produto



getTotal() pede ao “item da encomenda” para lhe dar o total da linha.

GRASP (Larman)

Generic Responsibility Assignment Principles

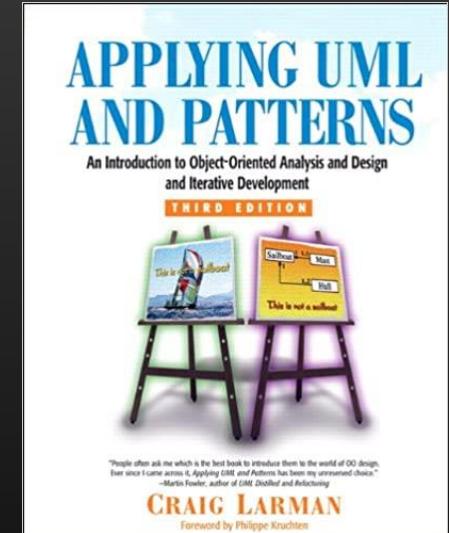
↓Coupling

↑Cohesion

Information Expert

Creator

Controller



Existem recomendações/princípio para orientar a distribuição de responsabilidades pelos objetos.
E.g.: GRASP



SOLID

Single responsibility
Open-closed
Liskov substitution
Interface segregation
Dependency inversion

SOLID

Software Development is not a Jenga game

Referências

Core readings	Suggested readings
<ul style="list-style-type: none">• [Dennis15] - Chap. 8	<ul style="list-style-type: none">• [Larman04] - Chap. 17 and 18• Slides by M. Eichberg : SSD and OO-Design

47006- ANÁLISE E MODELAÇÃO DE SISTEMAS

Agile methods

Ilídio Oliveira

v2020/12/16, TP16

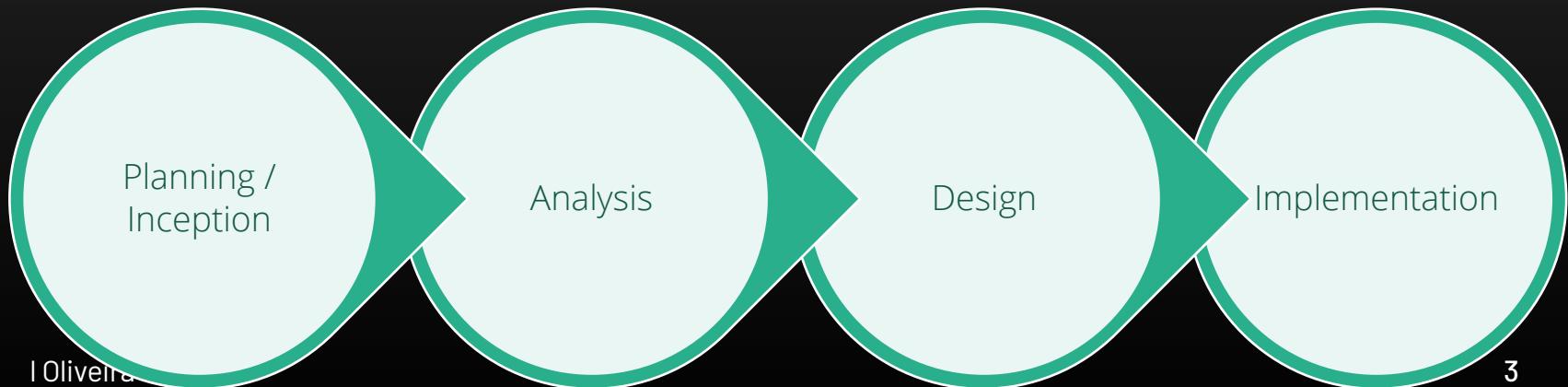
Learning objectives for this lecture

- Identify the distinctive characteristic of sequential processes, such as the waterfall approach.
- Identify the distinctive practices of Agile methods (what is new in the process model, comparing to the “traditional” approach?).
- Elaborate on the argument that “The waterfall approach tends to mask the real risks to a project until it is too late to do anything meaningful about them.”
- Identify advantages of structuring a project in iterations, producing increments.
- Characterize the principles of backlog management in Agile projects.
- Discuss the “principles” defined in the Agile Manifesto in your own words.

SDLC phases

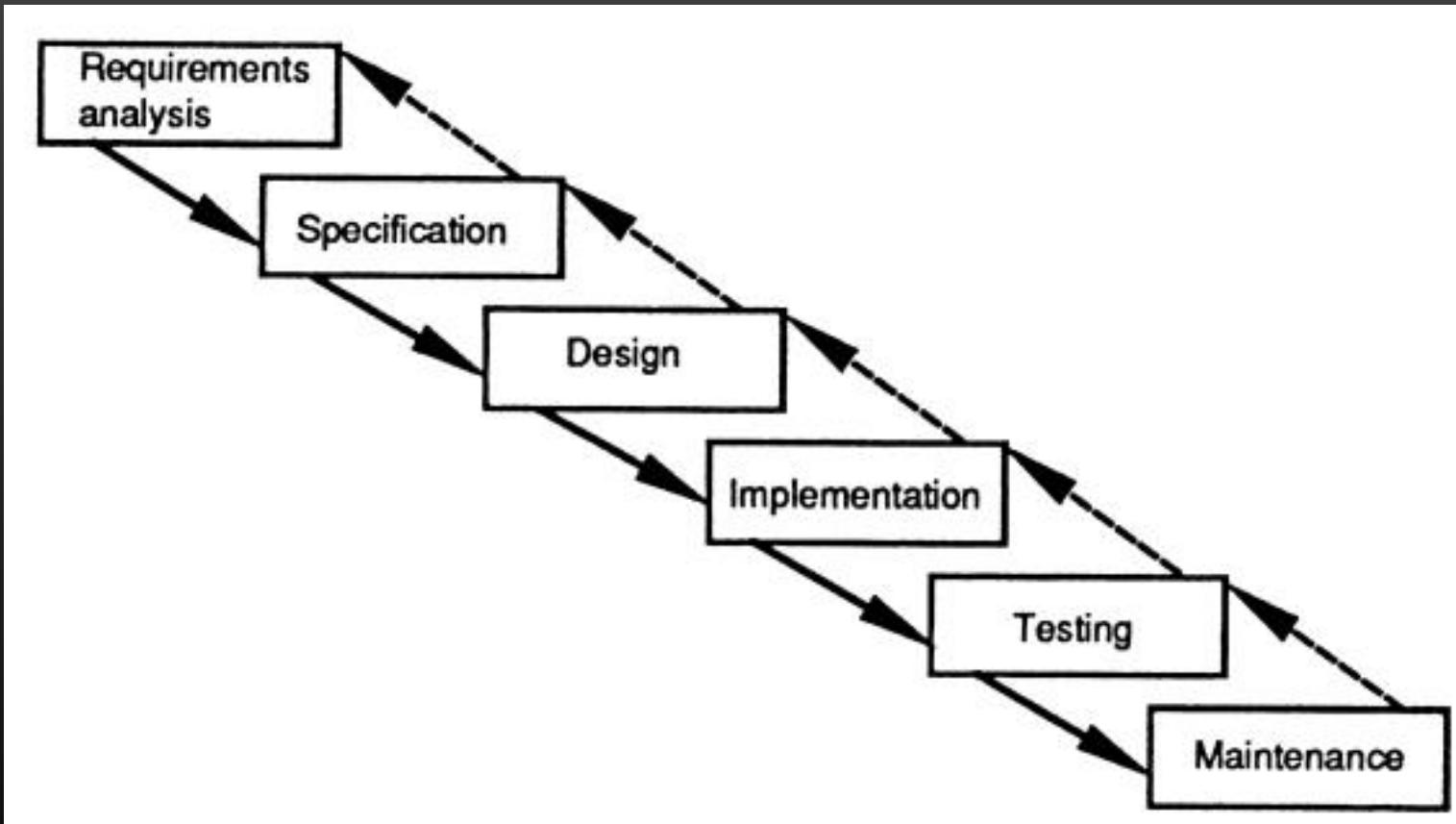
Four fundamental phases: planning, analysis, design, and implementation. Different projects might approach the SDLC phases in different ways, but all projects have elements of these four phases.

Each phase is itself composed of a series of steps, which rely upon techniques that produce deliverables.



Two visions on the development process

“Classical” engineering approach: **Waterfall model**



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

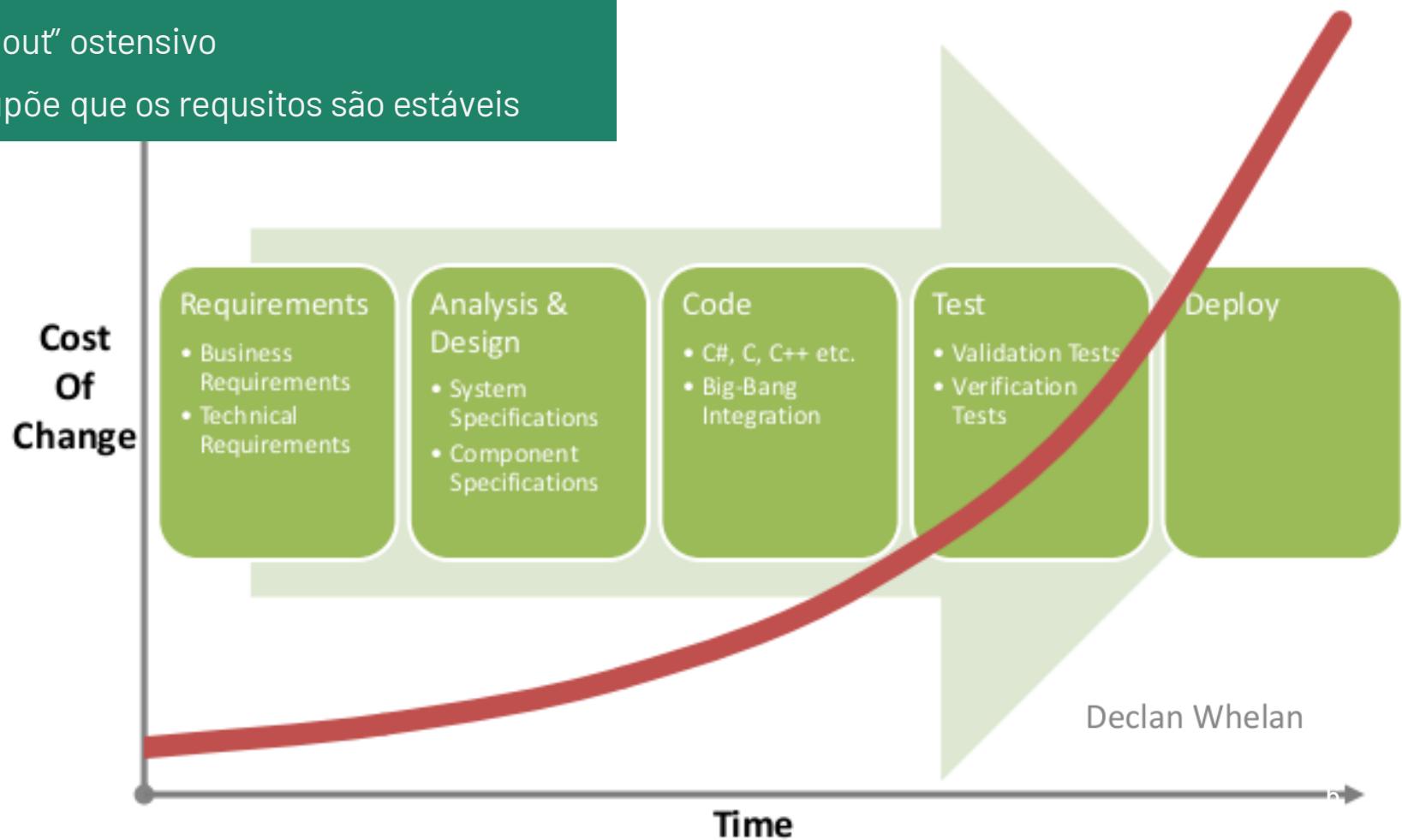
Problemas com a abordagem sequencial

Confirmação tardia de que os riscos estão controlados

Atividades de Integração e de Teste tardias

“Black out” ostensivo

Pressupõe que os requisitos são estáveis



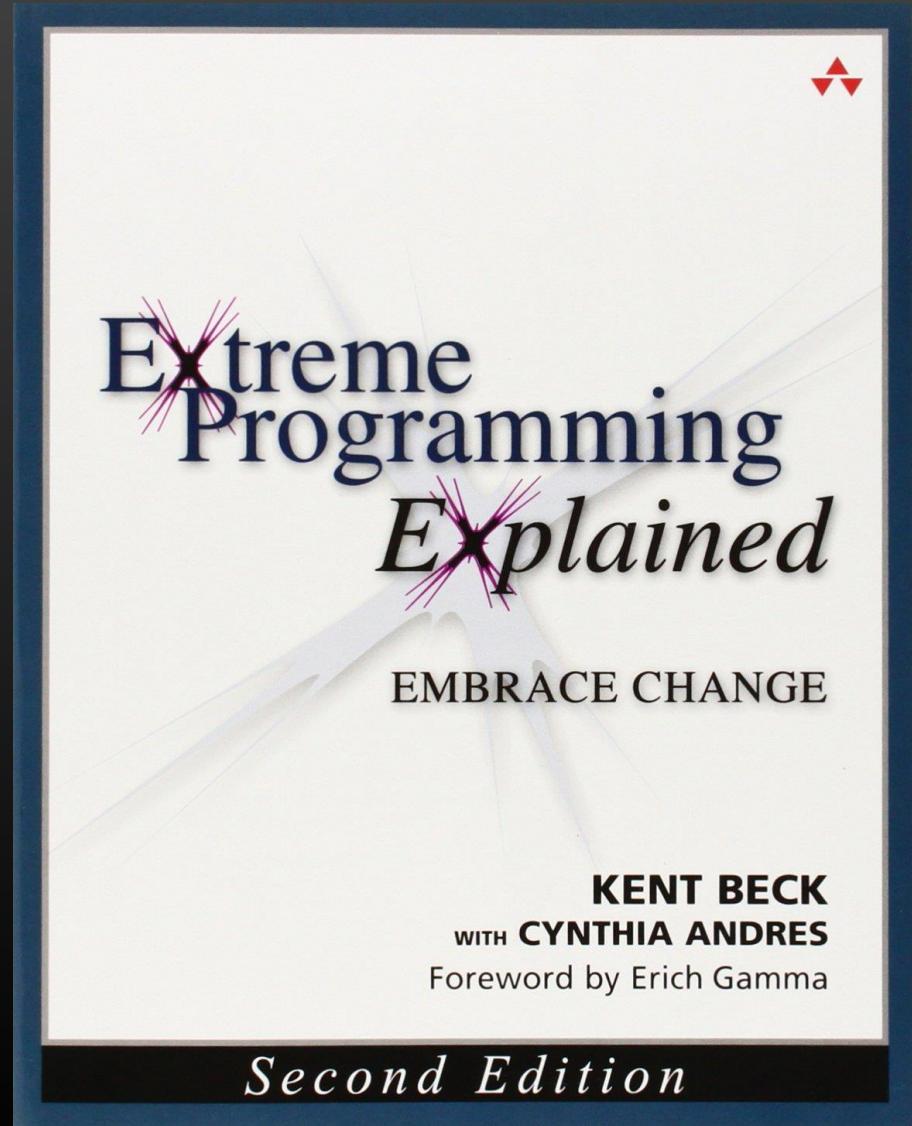
"embrace change"

Rather than:

- fighting the inevitable change that occurs in software development
- by trying (unsuccessfully) to fully and correctly specify, freeze, and "sign off" on a frozen requirement set and design before implementation

iterative and **evolutionary** development :

- is based on an attitude of embracing change and adaptation as unavoidable and indeed essential drivers.
- this is not to say that iterative development encourage an uncontrolled and reactive process.



Admitindo o objetivo...



Abordagem incremental



<http://jan-so.blogspot.com/2008/01/difference-between-iterative-and.html>

Abordagem iterativa



Abordagem incremental e iterativa



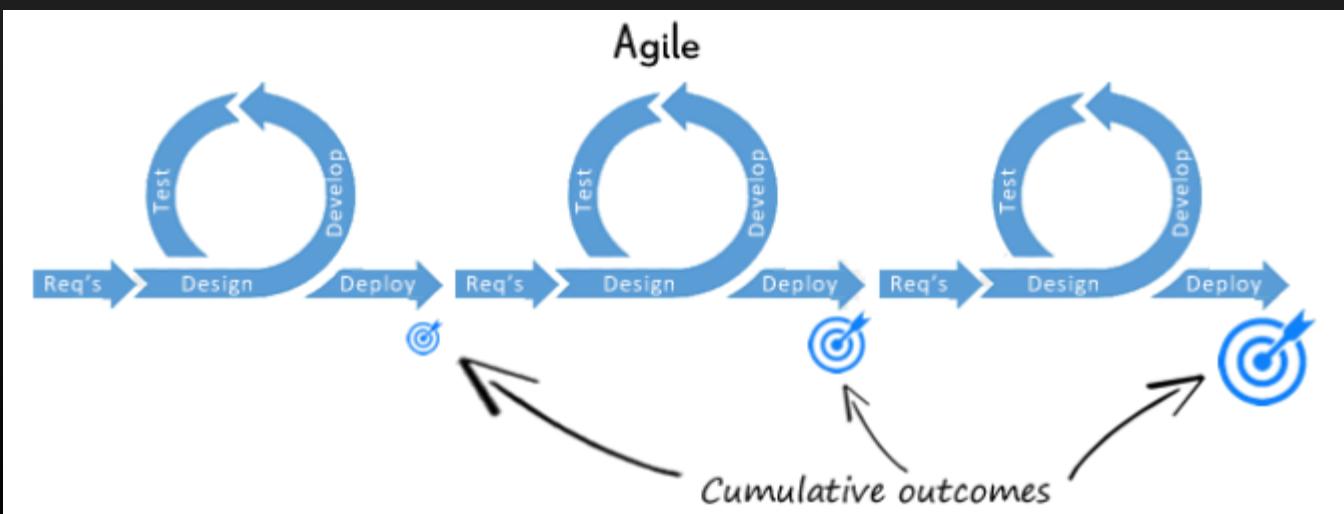
+



Iterative development

Each iteration involves choosing a small **subset of the requirements**, and quickly designing, implementing, and testing.

- Development in short cycles
- Each one is tested and integrated
- Each one gives an executable (partial) increment
- **Feedback** from each iteration leads to refinement and adaptation of the next.



Embrace change: agile processes and teams

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

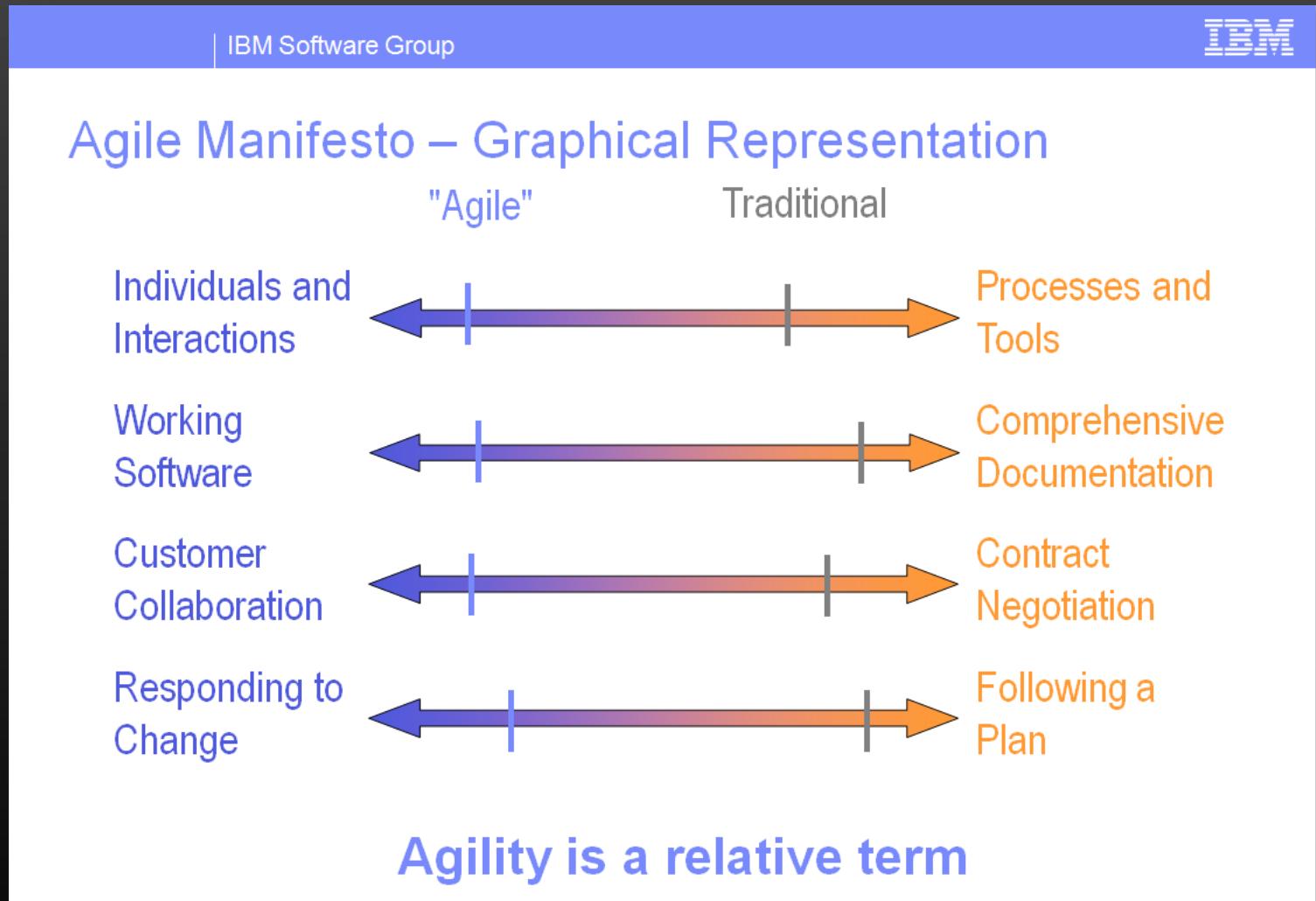
Software funcional mais do que documentação abrangente

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.

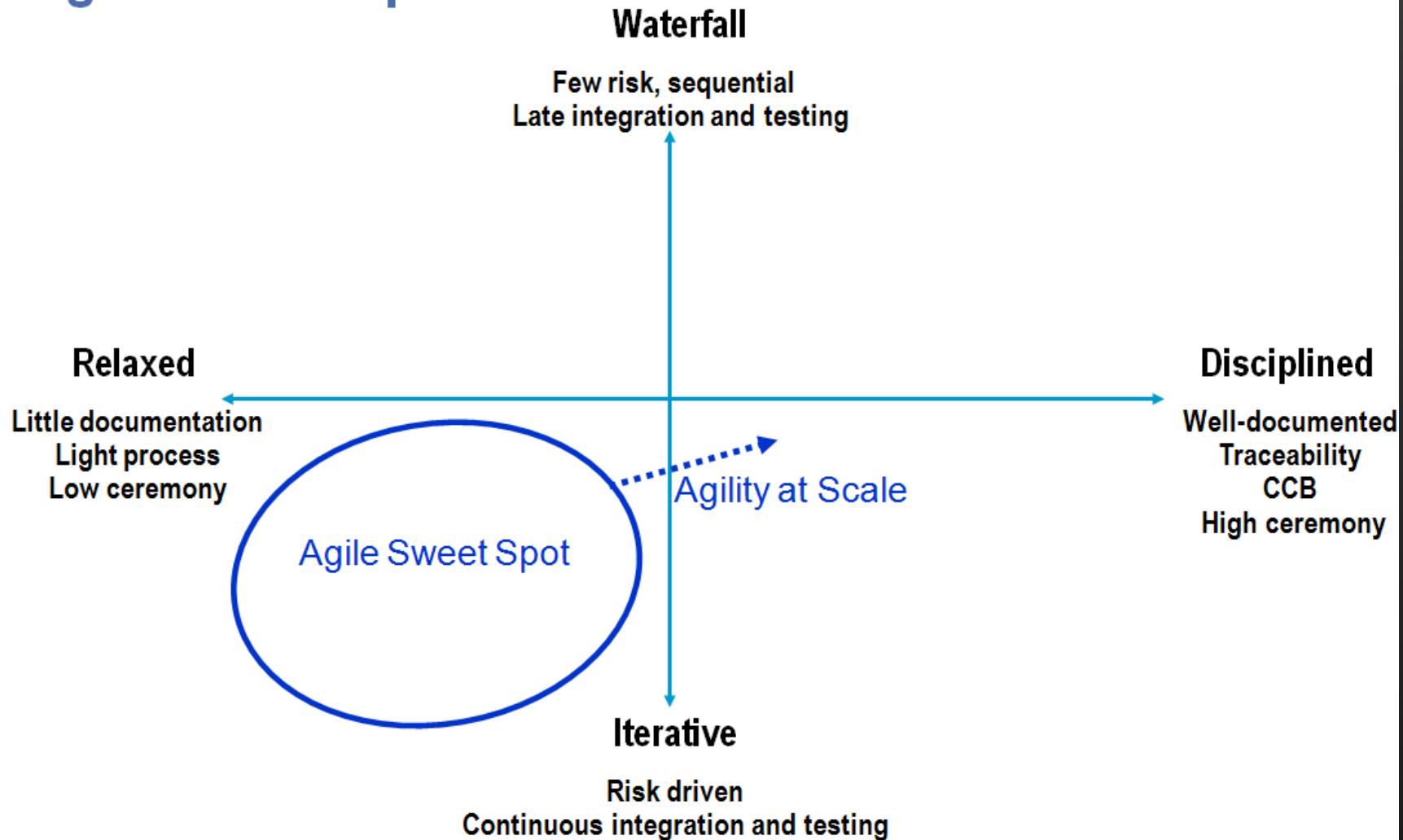
O desenvolvimento ágil de software



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 tardia 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 de interações face-a-face (conversas diretas).
I Oliveira
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.

Agile Sweet Spot



Credit: Per Kroll (IBM)

Marcas do *Agile*

É conduzido por descrições do cliente do que é necessário (cenários)

Reconhece que os planos são de curta duração

Desenvolve software iterativamente com uma forte ênfase nas atividades de construção

práticas e ferramentas devem ajudar ao desenvolvimento evolutivo

Fornece vários 'incrementos' do software

Adapta-se à medida que as mudanças ocorrem

How about the Unified Process?

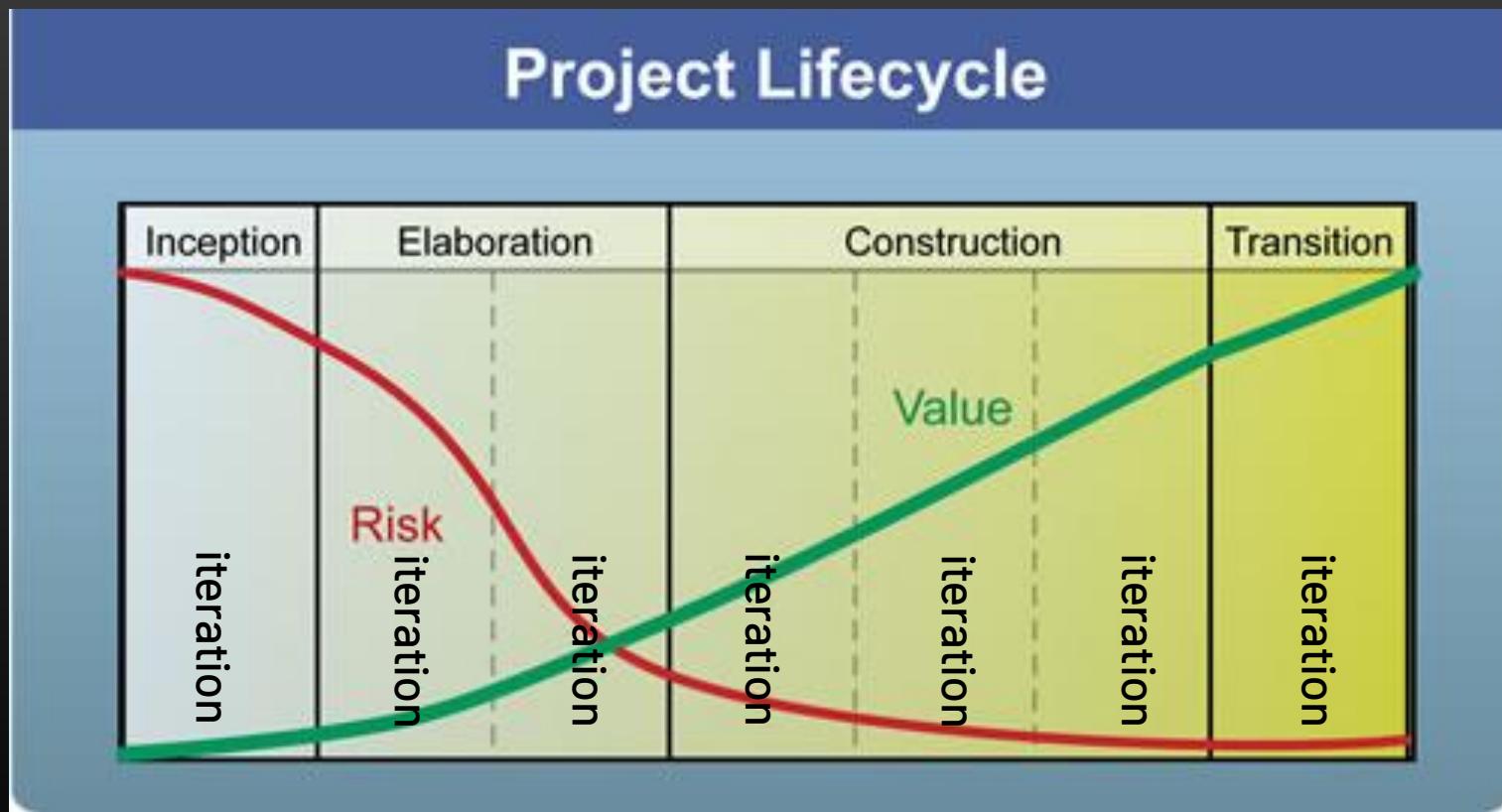
Unified Process approach

Iterative and Incremental

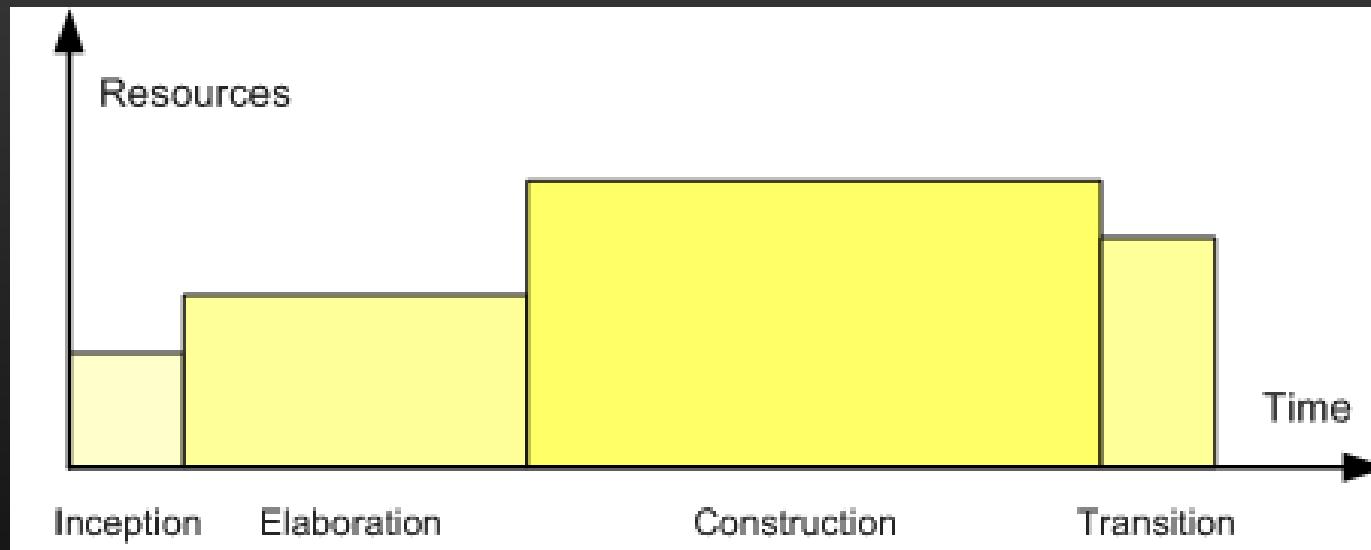
Use Case Driven

Architecture Centric

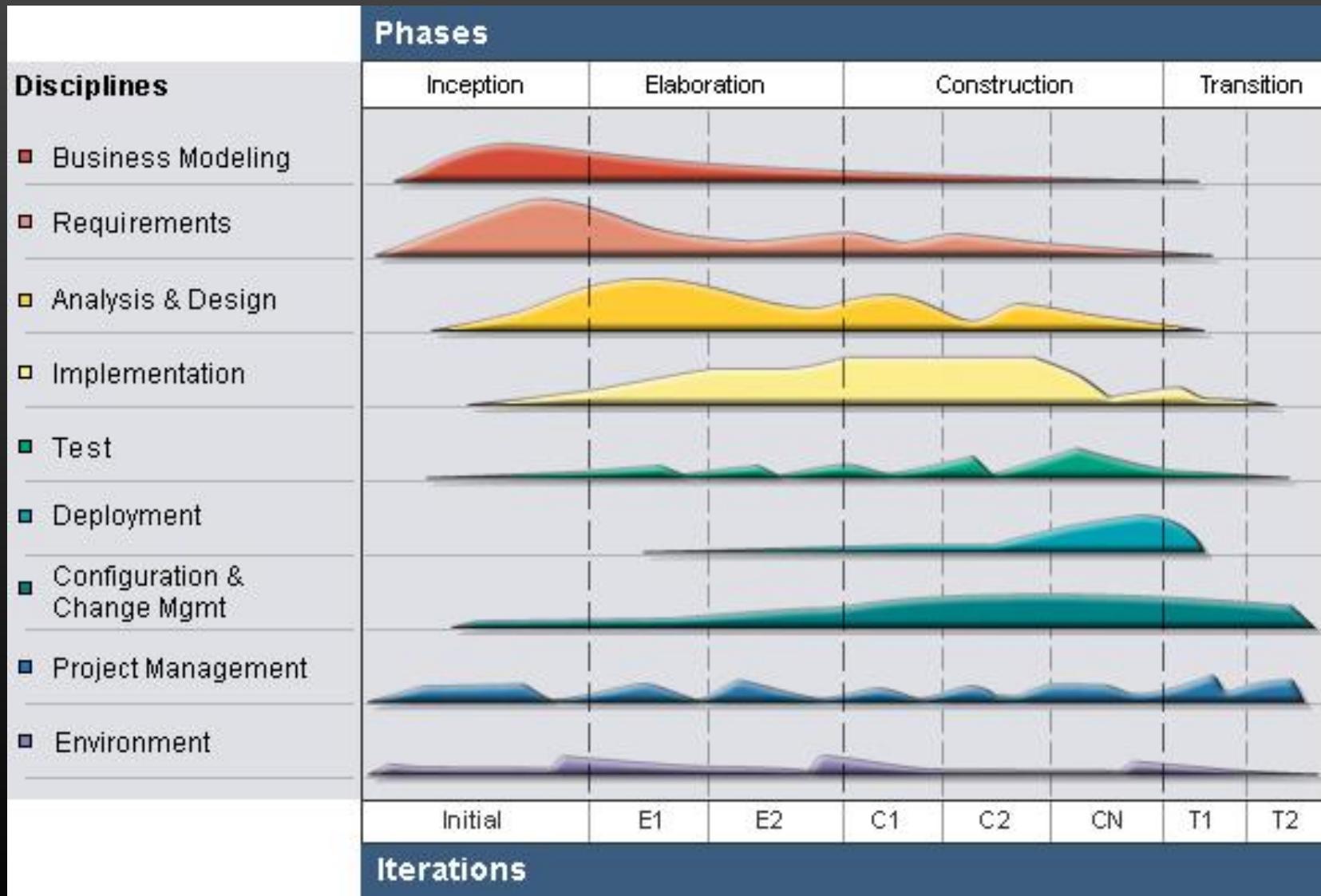
Estrutura do Unified Process



"typical" resource allocation profile



Ciclo de vida do Unified Process



Unified Process... adapted

Agile Unified Process (AUP), by
Scott W. Ambler

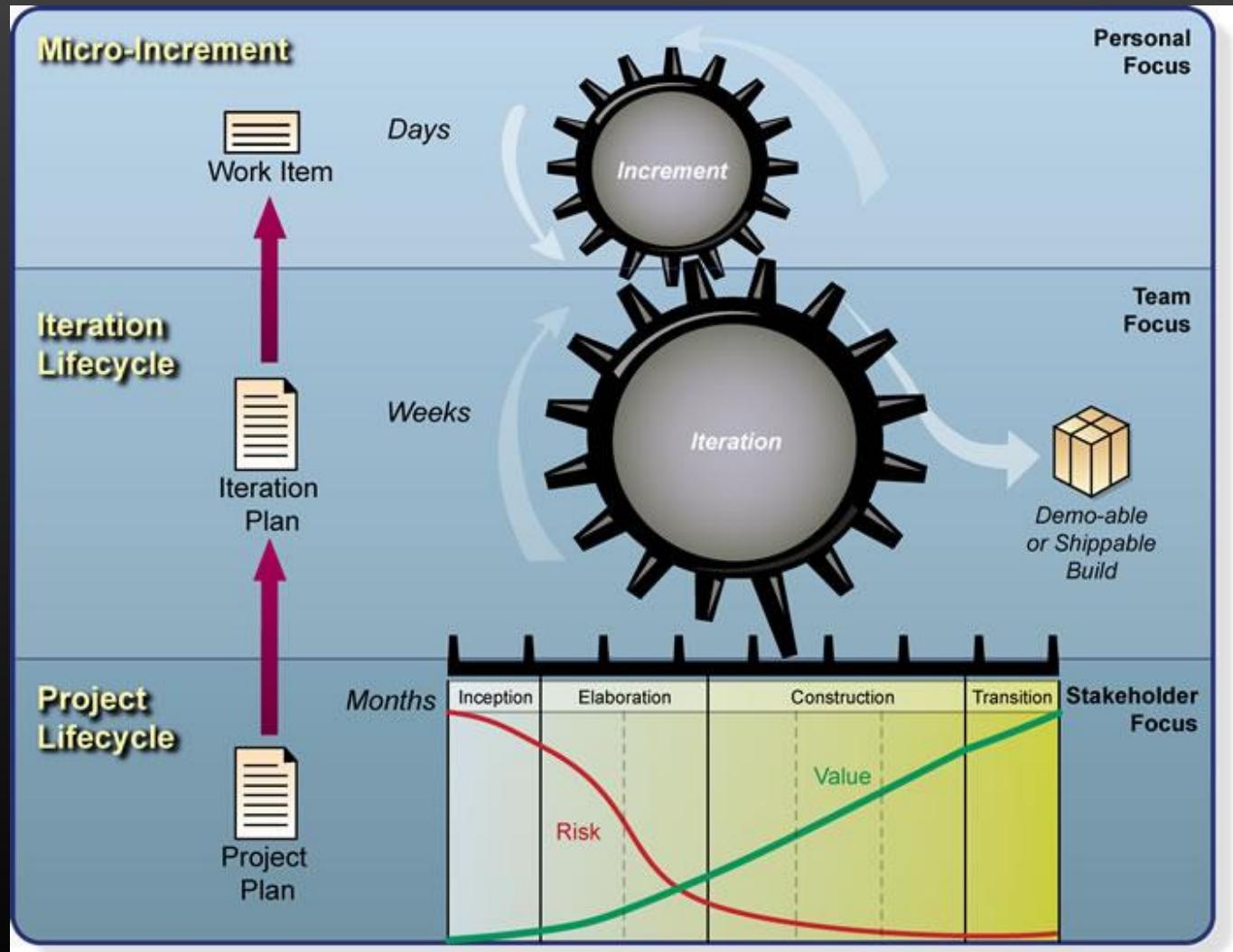
Open Unified Process (OpenUP),
by the Eclipse Process
Framework

Rational Unified Process (RUP), by
IBM / Rational Software

Oracle Unified Method (OUM), by
Oracle

... and more.

Agile example: Open Unified Process (OpenUP)



→ [OpenUP wiki](#)
(local mirror)

Evolving Applications through Micro-Increments

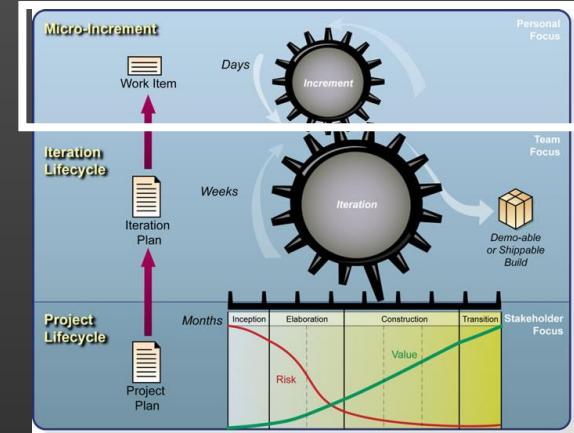
Each micro-increment

- corresponds to 0.5-3 days person days of work
- is added (and removed if necessary) to the build
- needs to be properly tested
- Needs to looked upon (requirements, design, implementation and testing) from a user-value perspective (use-case driven)

A month-long iteration with 20 developers would consists of ~200 micro-increments

Micro-increments provides the team with the ability to manage and demonstrate continuous progress

Micro-increments applies to any type of project activity



References

Core readings	Suggested readings
<ul style="list-style-type: none">• [Dennis15] - Chap. 1	[Pressman], Chap. 3 "Agile Development"

Introdução ao SCRUM

Ilídio Oliveira

v2020/12/16, TP16b

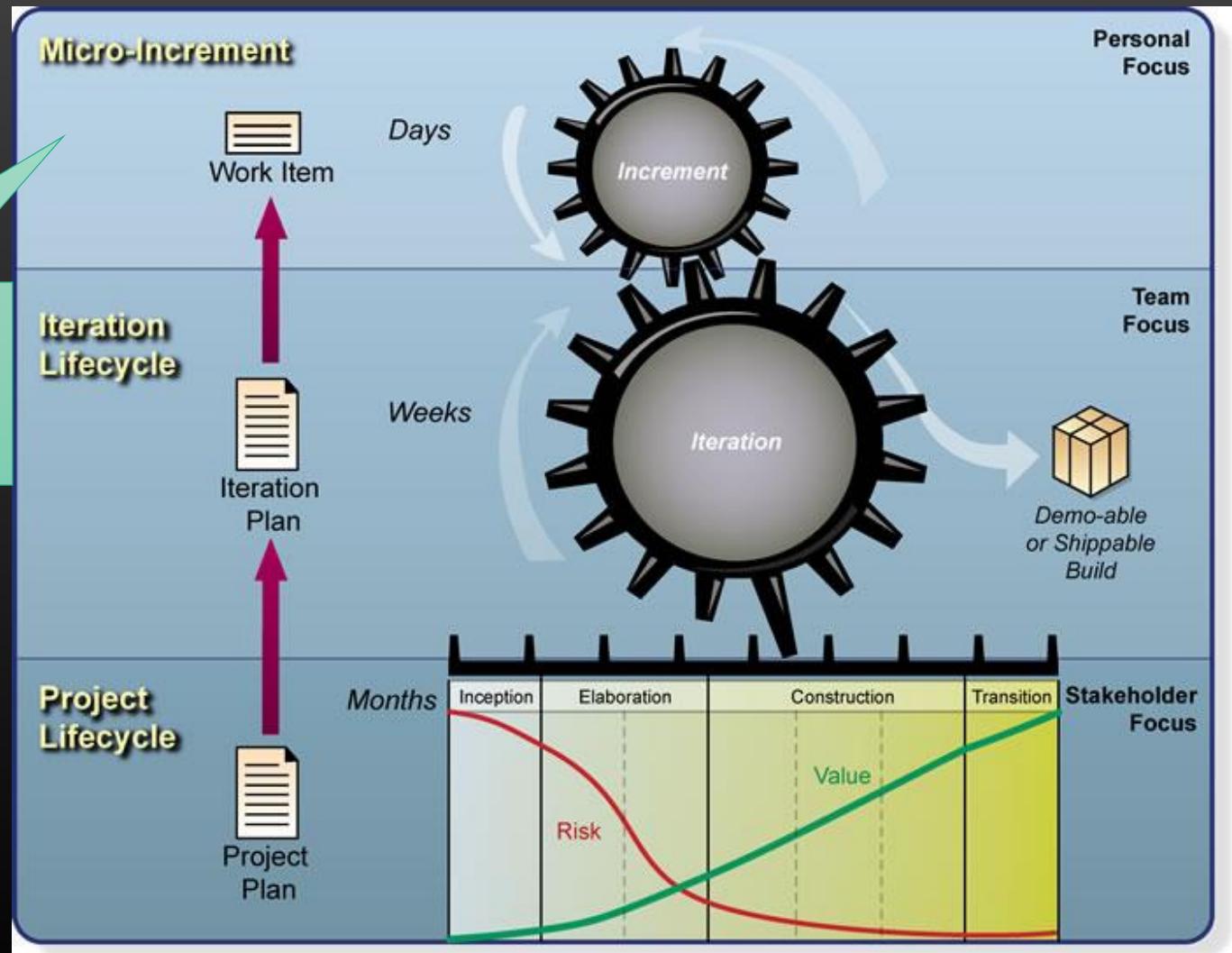
Objetivos de aprendizagem

- 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)

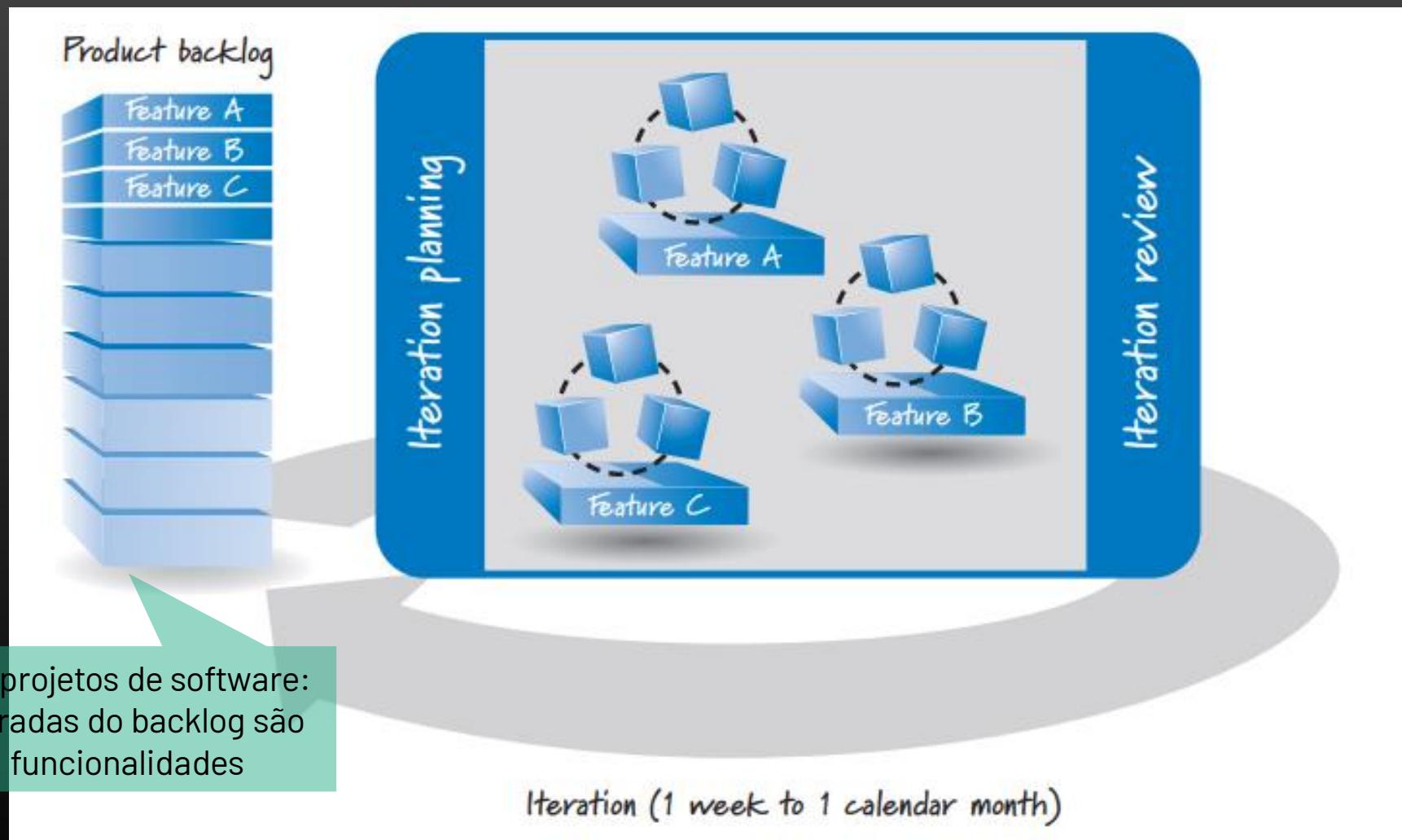
OpenUP prevê o trabalho por iterações

→ [OpenUP wiki](#)

Apesar de ser clara a abordagem incremental, como é que se organiza a equipa do dia-a-dia?



Planeamento do trabalho e métodos ágeis



A metáfora do Scrum...

"restarting play in rugby football that involves players packing closely together with their heads down and attempting to gain possession of the ball"

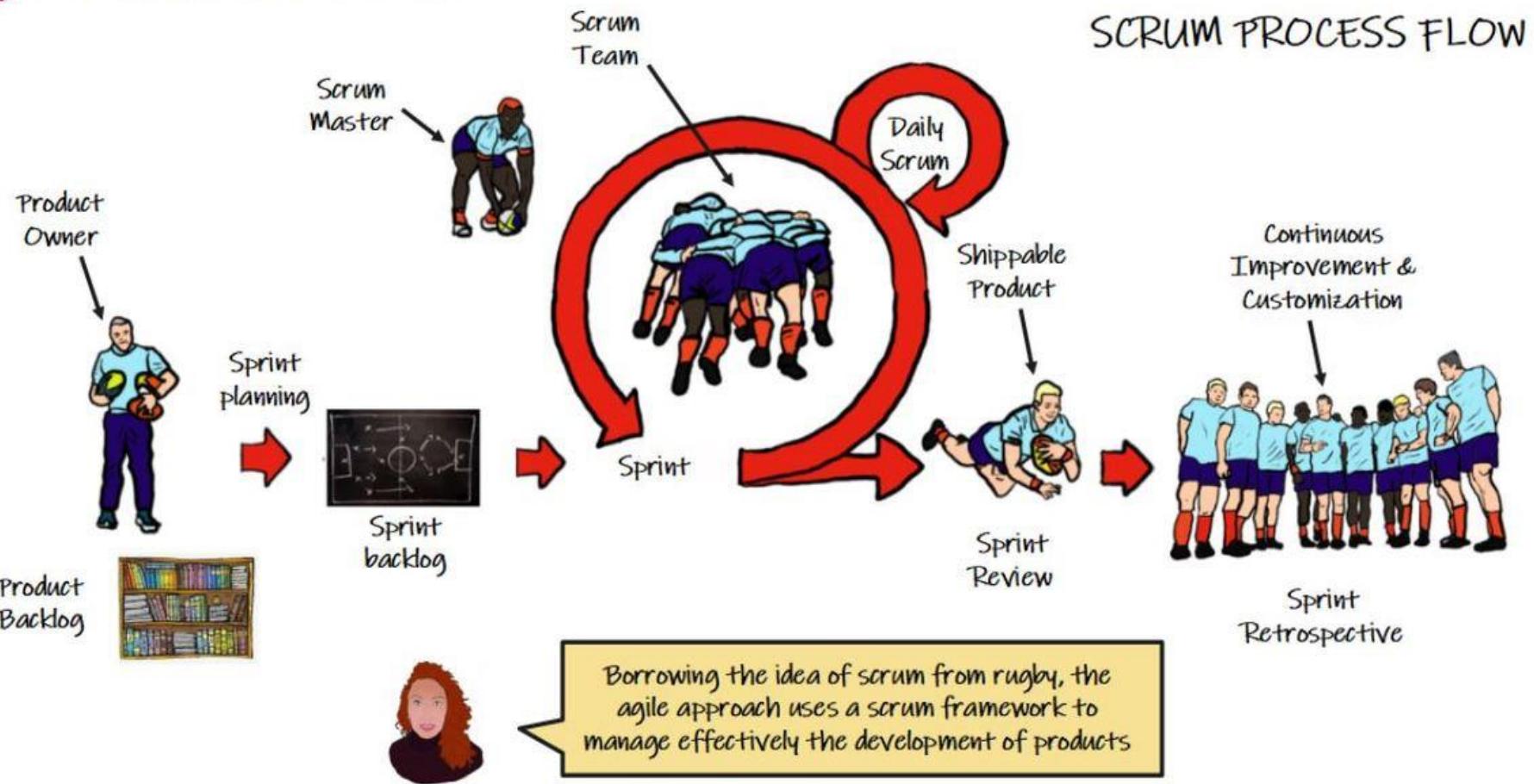
Scrum jobs?...

- [Indeed.pt](#)

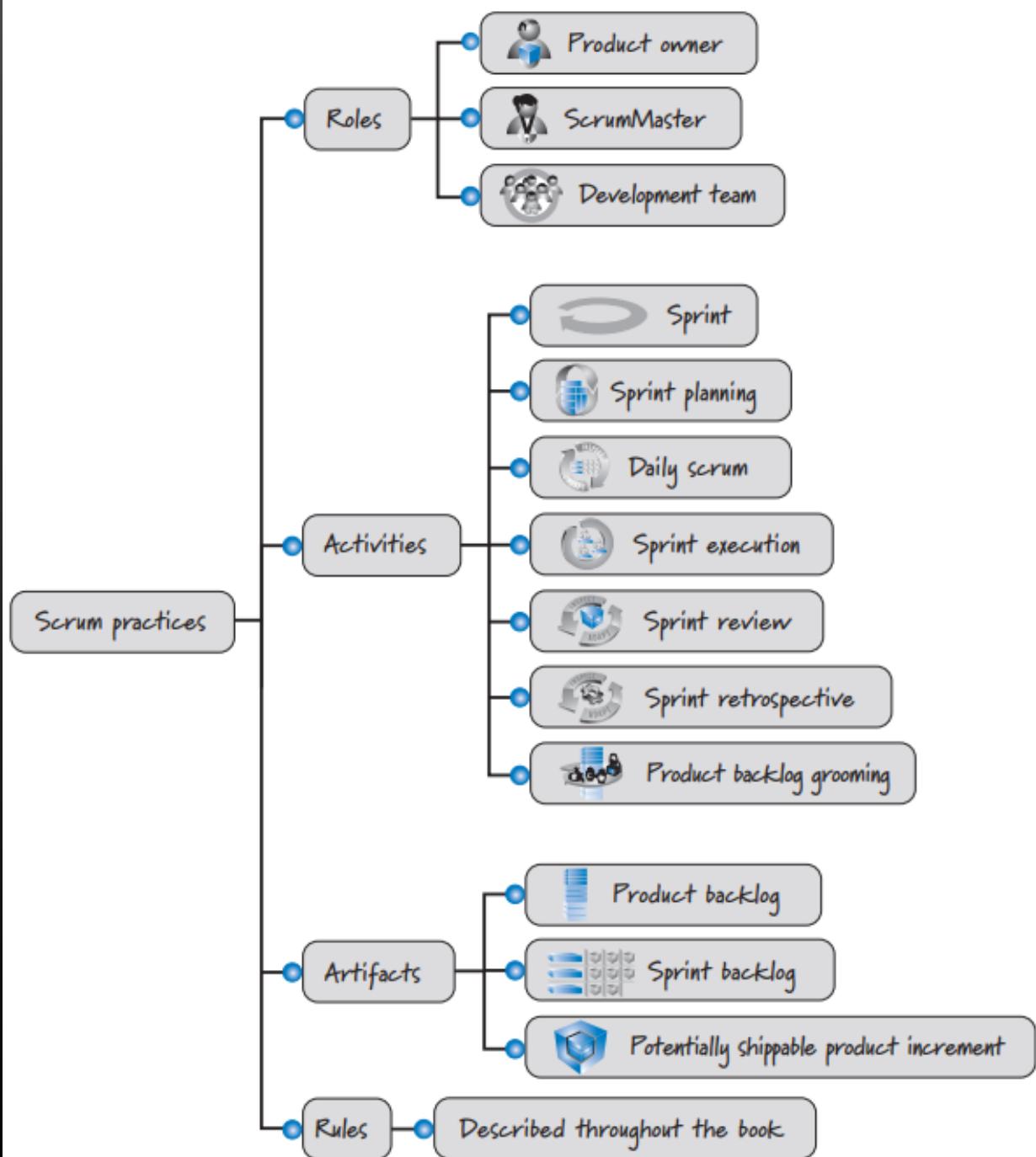


Metodologia de gestão de equipas SCRUM

SCRUM-INSTITUTE.ORG



Elementos do Scrum



"3355"

3
Roles



Product
Owner



Development
Team



Scrum Master

3
Artifacts



Product
Backlog



Sprint
Backlog



Product
Increment

5
Events



Sprint



Sprint Planning
Meeting



Daily Scrum
Meeting

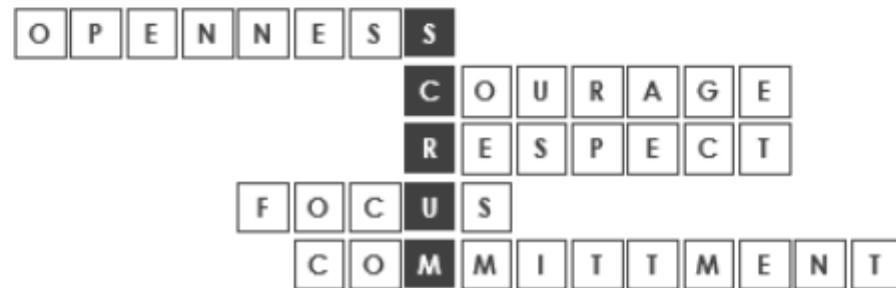


Scrum Review
Meeting



Scrum Retrospective
Meeting

5
Values



Scrum framework

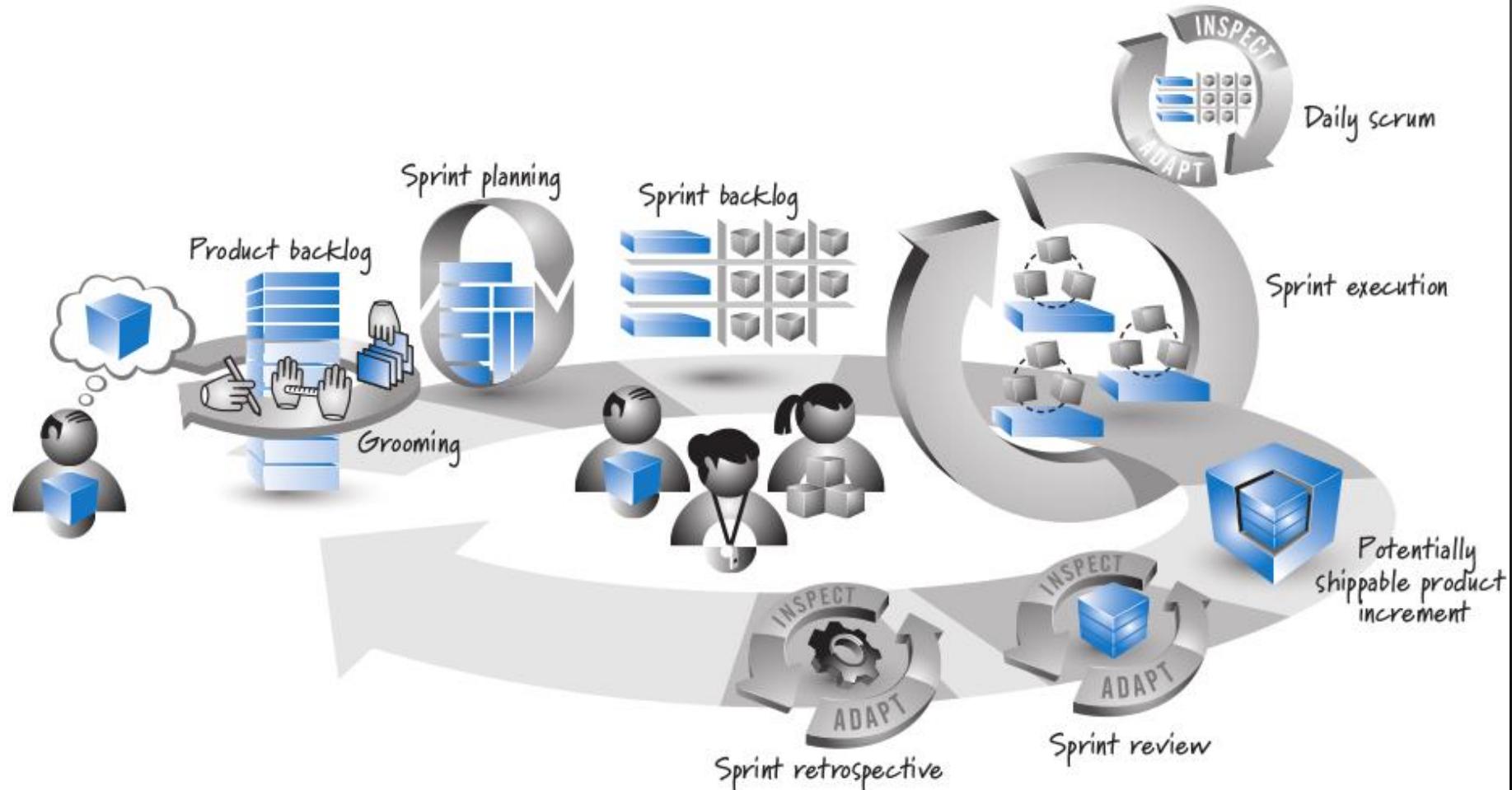
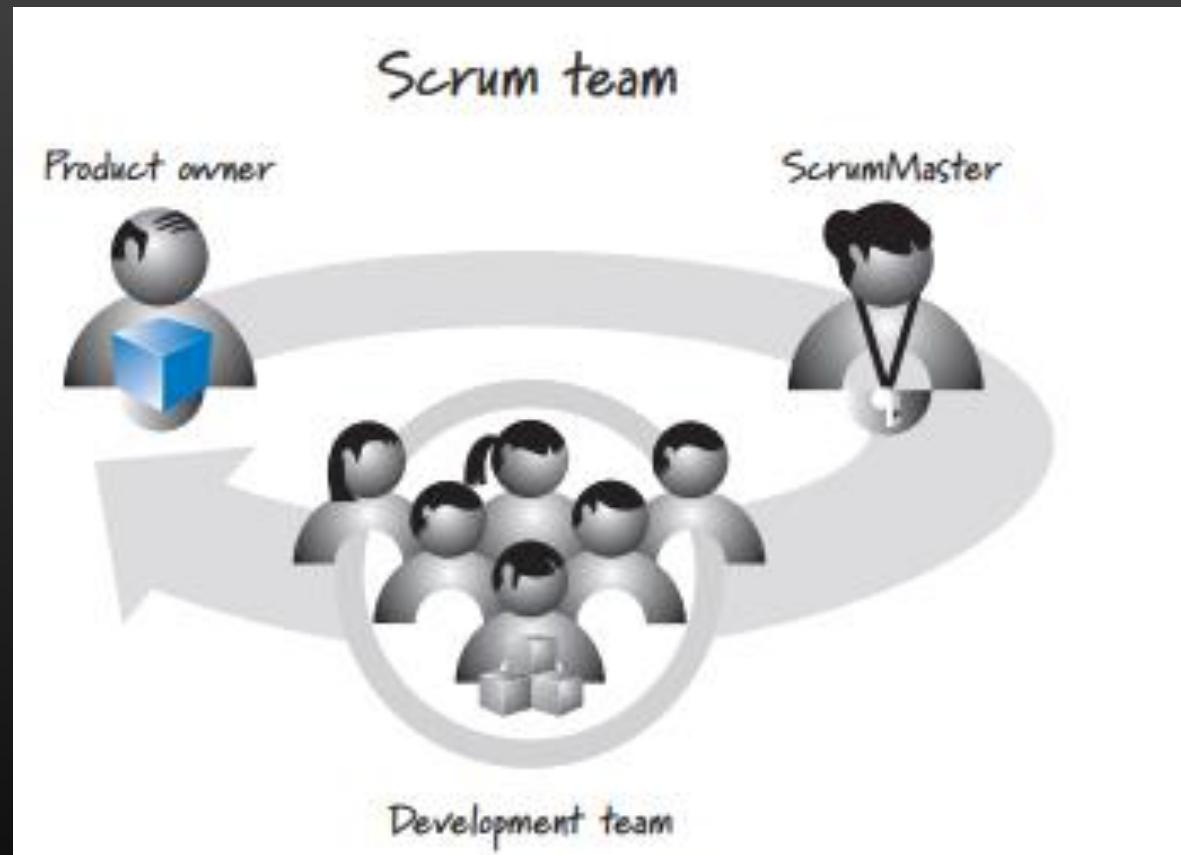


FIGURE 2.3 Scrum framework

Papéis previstos no Scrum



→ Guia do SCRUM (recursos da TP @Moodle)

Creating a Product: Scrum: Roles & Responsibilities



Product Owner

- Holds the vision for the product
- Determines what needs to be done
- Sets the priorities to deliver the highest value



Scrum Master

- Help the team best use Scrum to build the product
- Protecting the Scrum process
- Prevent distractions/impediments



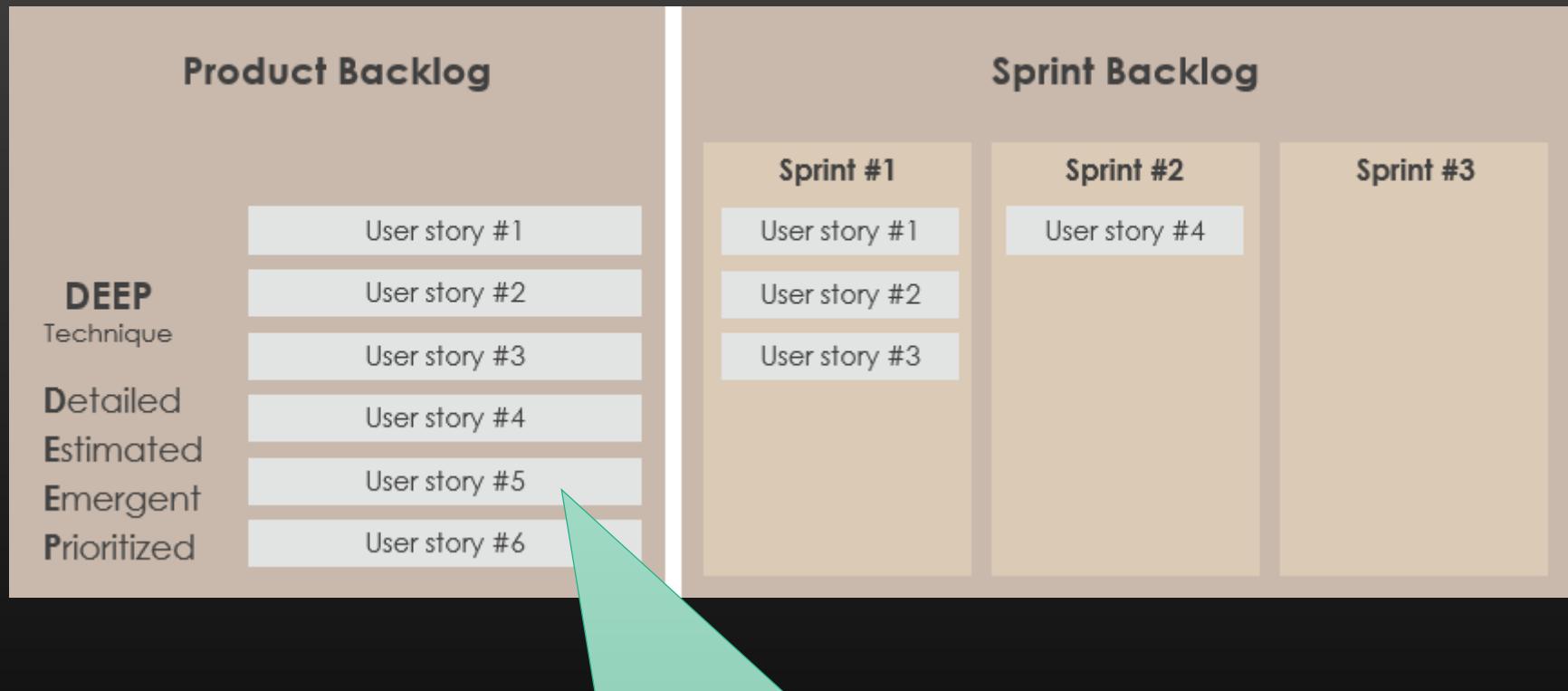
Development Team

- Builds the product
- Self-organizing group
- Takes on and determines how to deliver chunks of work in frequent increments

Credit: Nokia Networks.

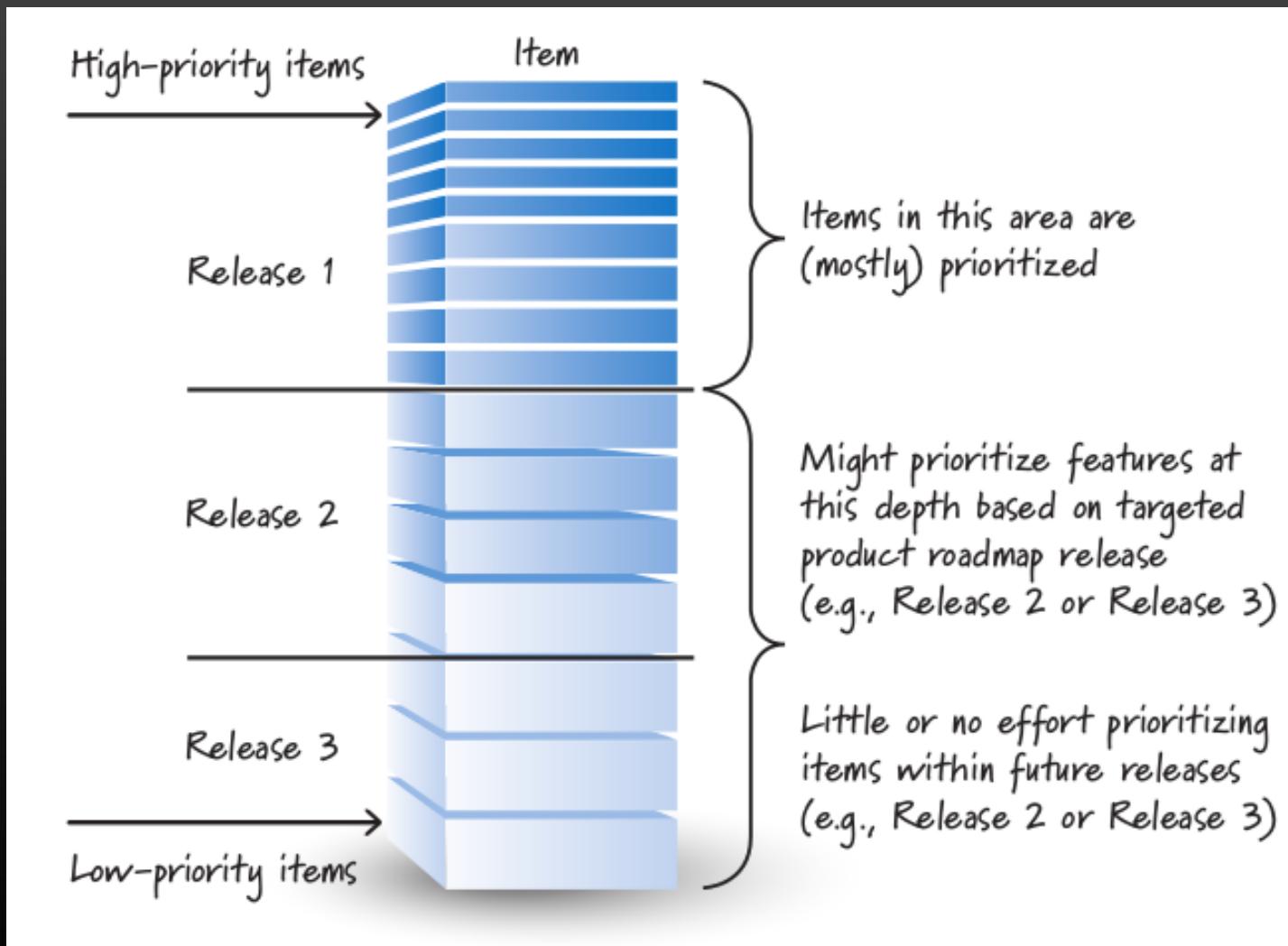
<https://forms.gle/9ZKdzBfY3fVLVSCWA>

Sprint planning



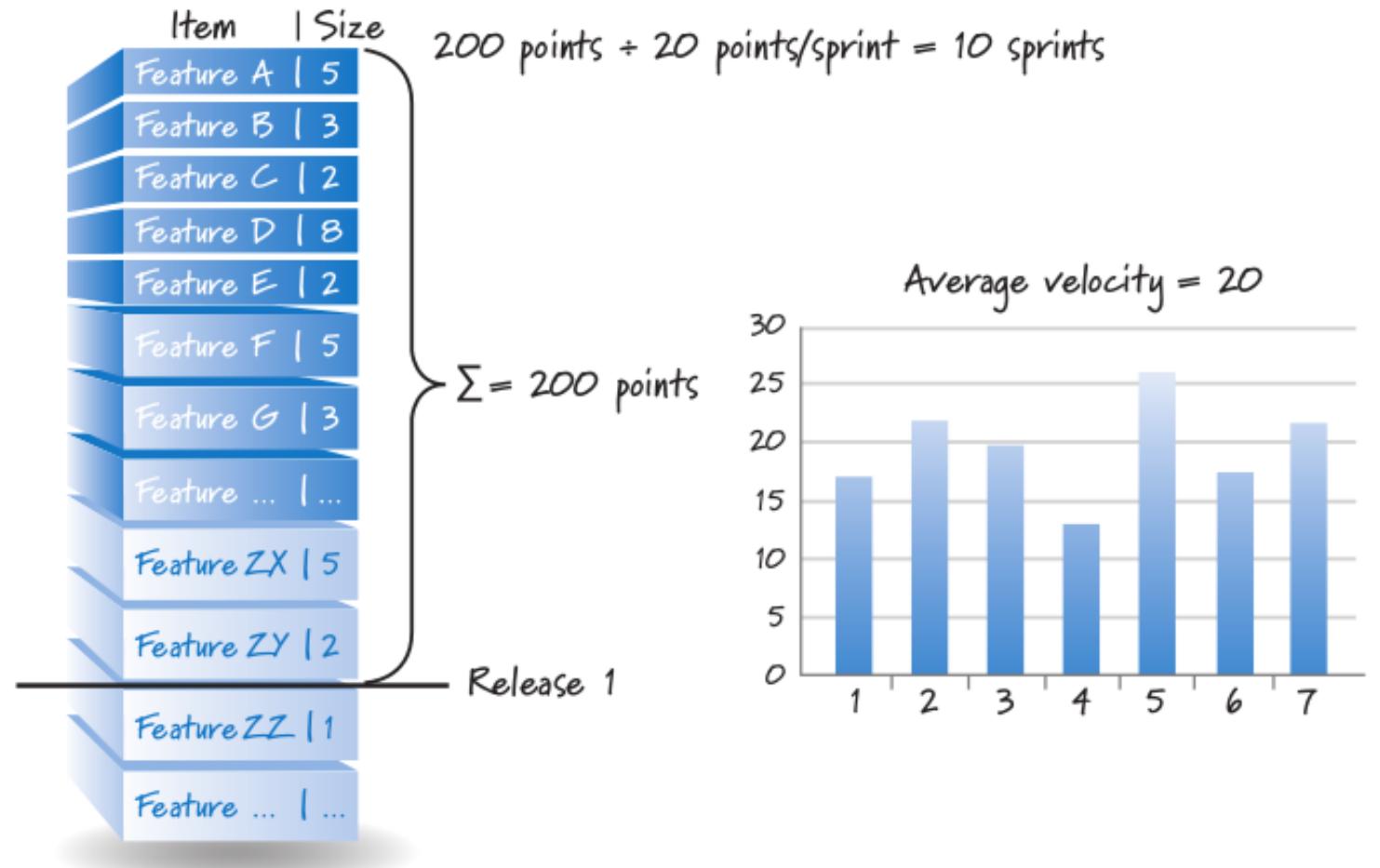
No desenvolvimento de sw, há um estilo para escrever as entradas do *backlog*, adotando o conceito de “user story”. A história é um exemplo de utilização, uma forma de percorrer um caso de utilização.

Scrum: backlog must be prioritized

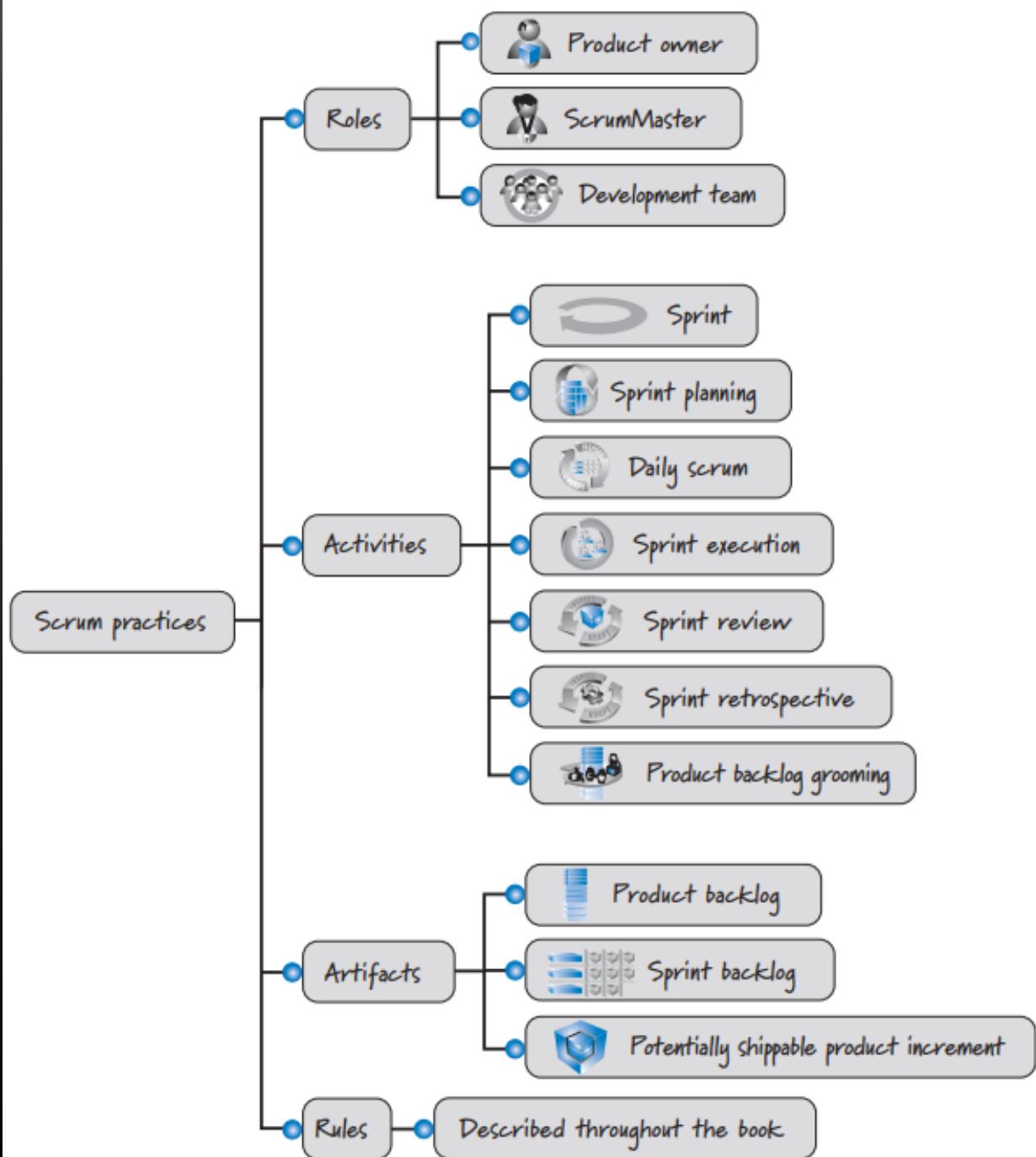


Scrum: Velocity

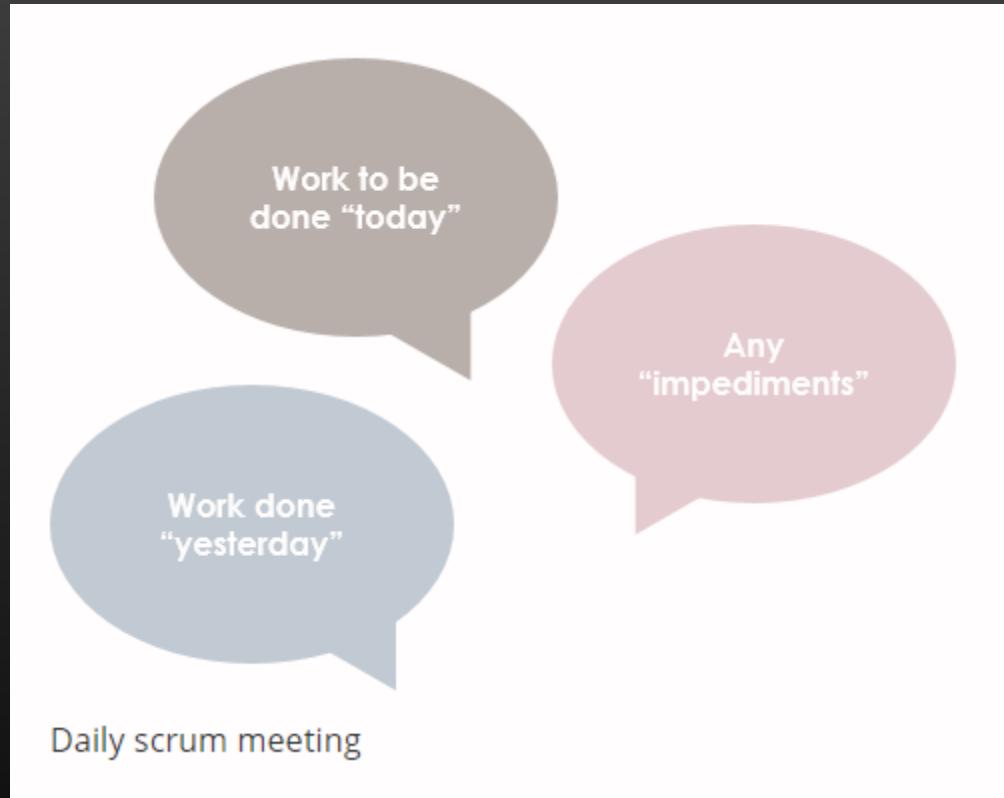
Estimated size ÷ measured velocity = (number of sprints)



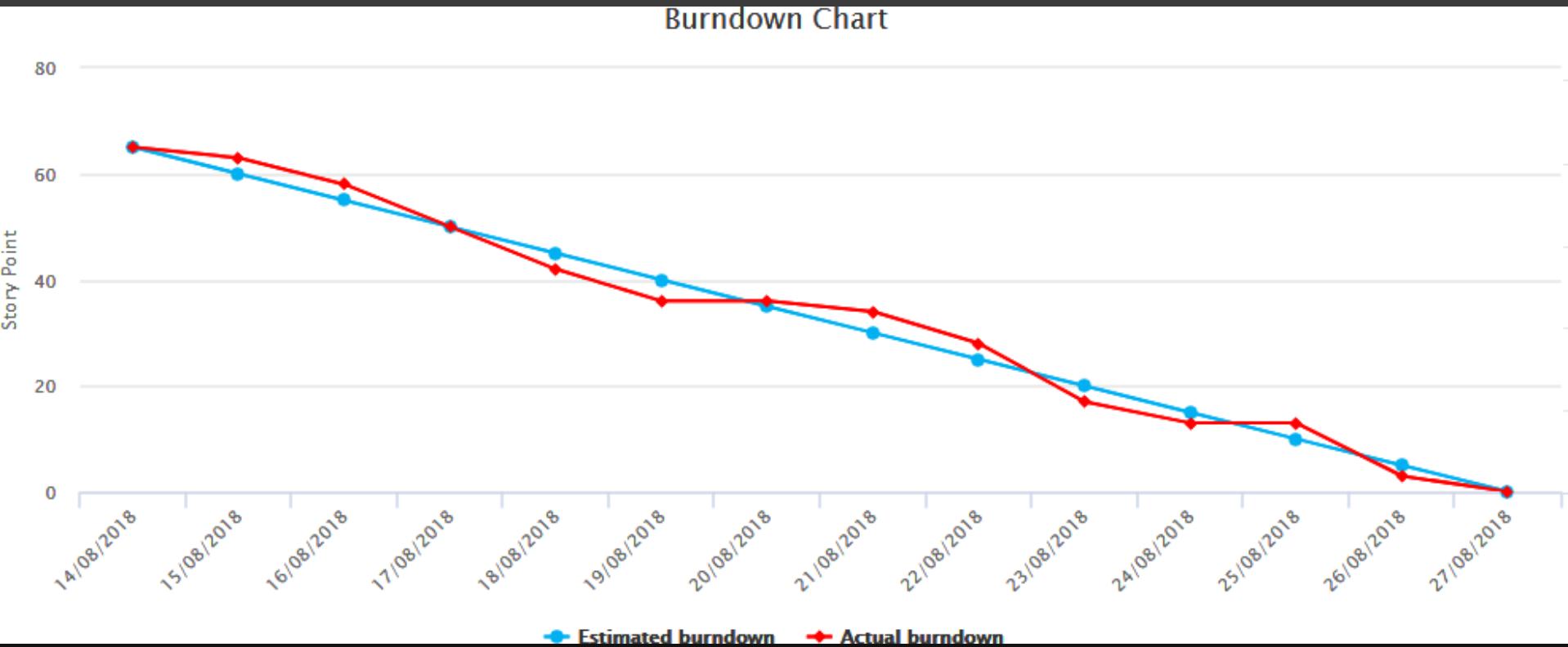
Elementos do Scrum



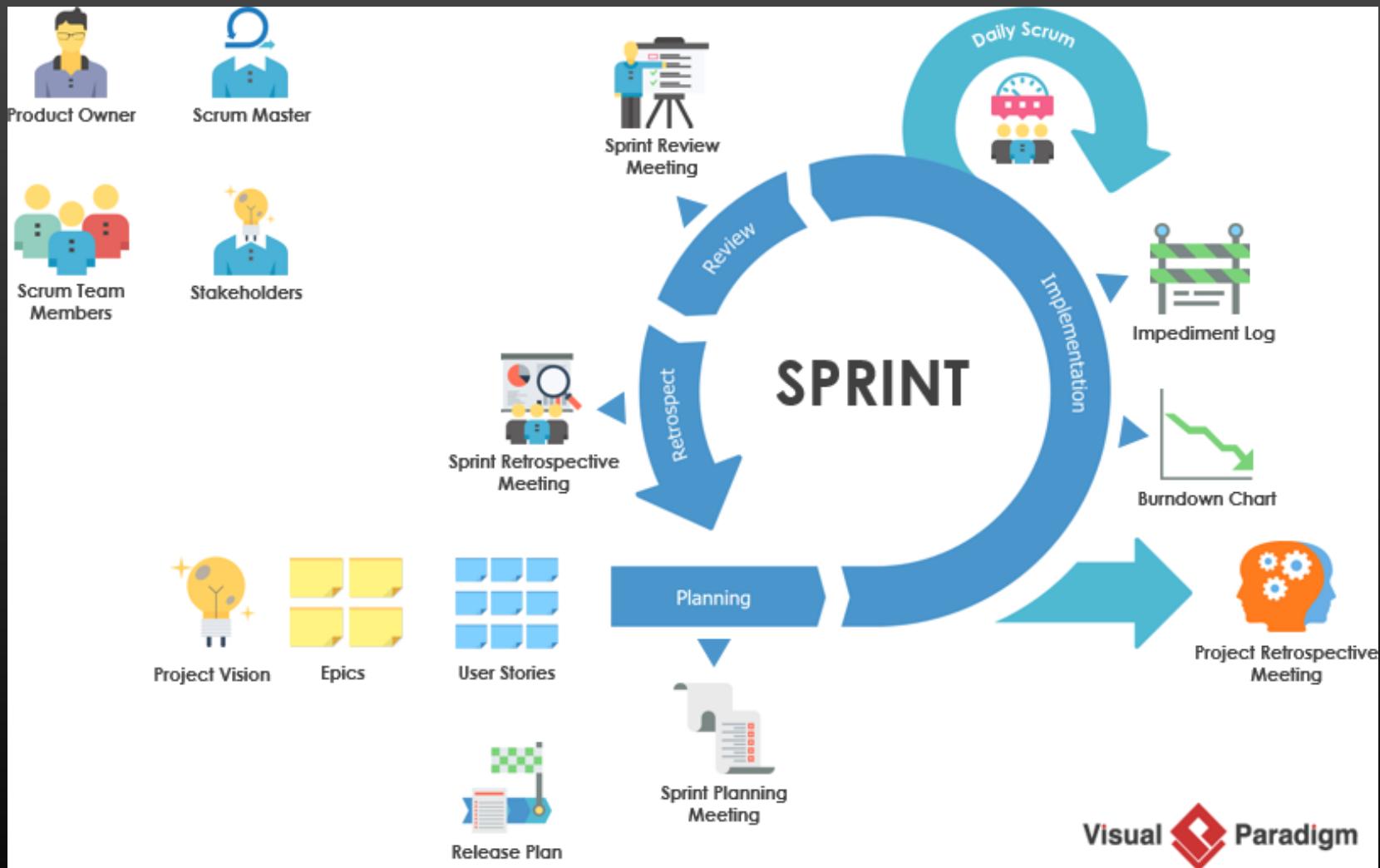
Daily Scrum



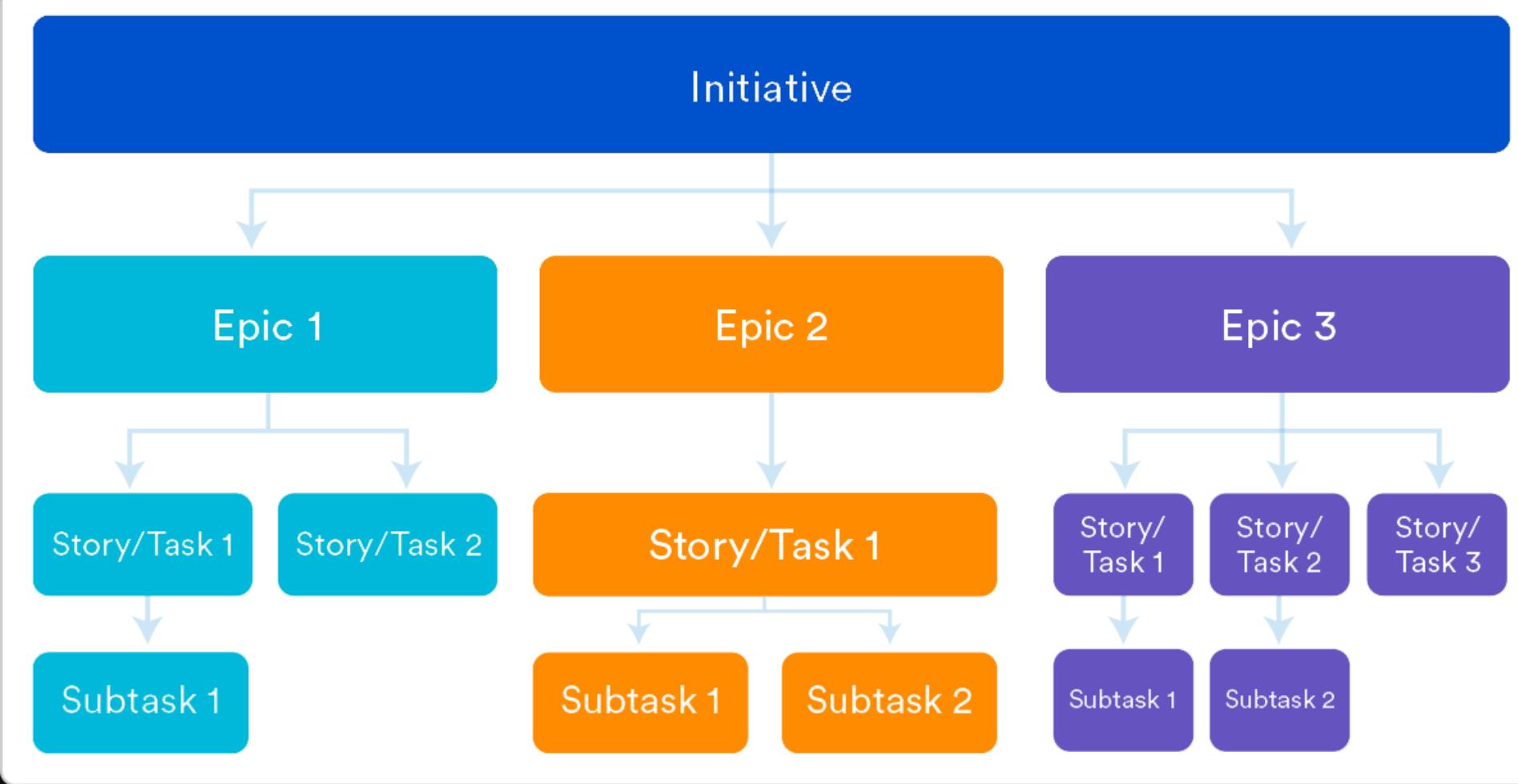
Monitorar o progresso com “burndown chart”



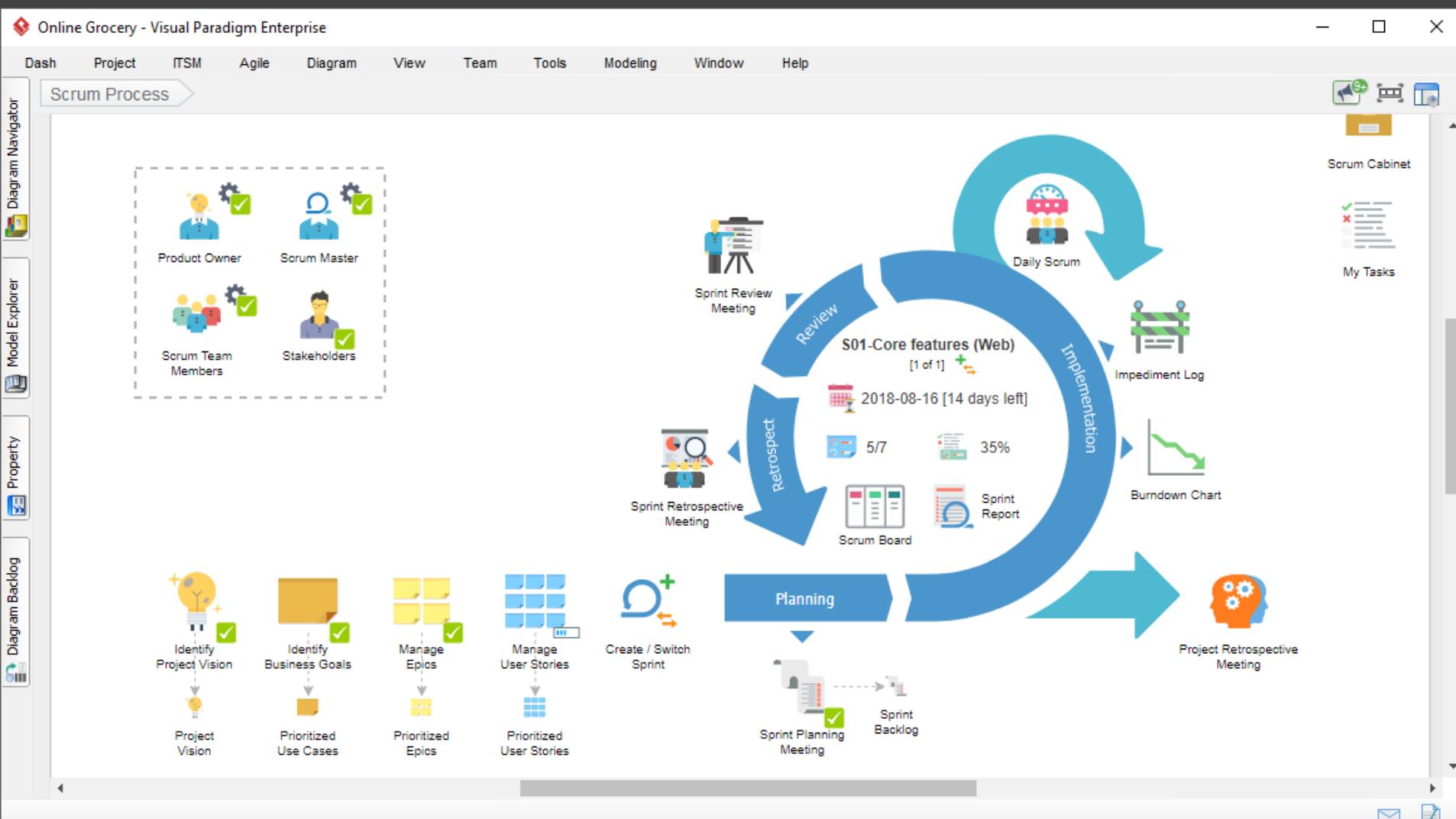
<https://www.visual-paradigm.com/cn/scrum/scrum-burndown-chart/>



Epic: coherent set of Stories that meets a relevant achievement



[Optional] Scrum tutorial by VisualParadigm



<https://www.visual-paradigm.com/tutorials/agile-tutorial/>
I Oliveira

Algumas ideias a reter

- Um processo de software explica o trabalho a desenvolver para construir o produto
- O processo não explica como organizar o dia-a-dia da equipa
- A Scrum oferece uma metodologia “leve” para gestão de equipas, a construir produtos complexos
- Mas... é desafiante dominar e aplicar a Scrum!

A Scrum é especialmente adequada para métodos ágeis de desenvolvimento de software

- Sprint (iteração)
- Equipa auto-organizadas e multifuncionais (comunicação)
- Foco no incremento (entrega frequente)
- Adaptação (“*embrance change*”)

References

Core readings	Suggested readings
<ul style="list-style-type: none">• Visual Paradigm, "What is Scrum?"• Ken Schwaber, Jeff Sutherland, "Scrum Guide".	

47006- ANÁLISE E MODELAÇÃO DE SISTEMAS

Agile methods: the role of user stories

Ilídio Oliveira

v2020/12/18, TP17a

Learning objectives for this lecture

Characterize the principles of backlog management in agile projects

Define and write stories for a given product.

Distinguish use story estimation and prioritization.

Write the acceptance criteria part of a user story.

Compare user stories and use cases with respect to commonalities and differences.

Describe the PivotalTracker story-based development workflow.

Scrum framework activities

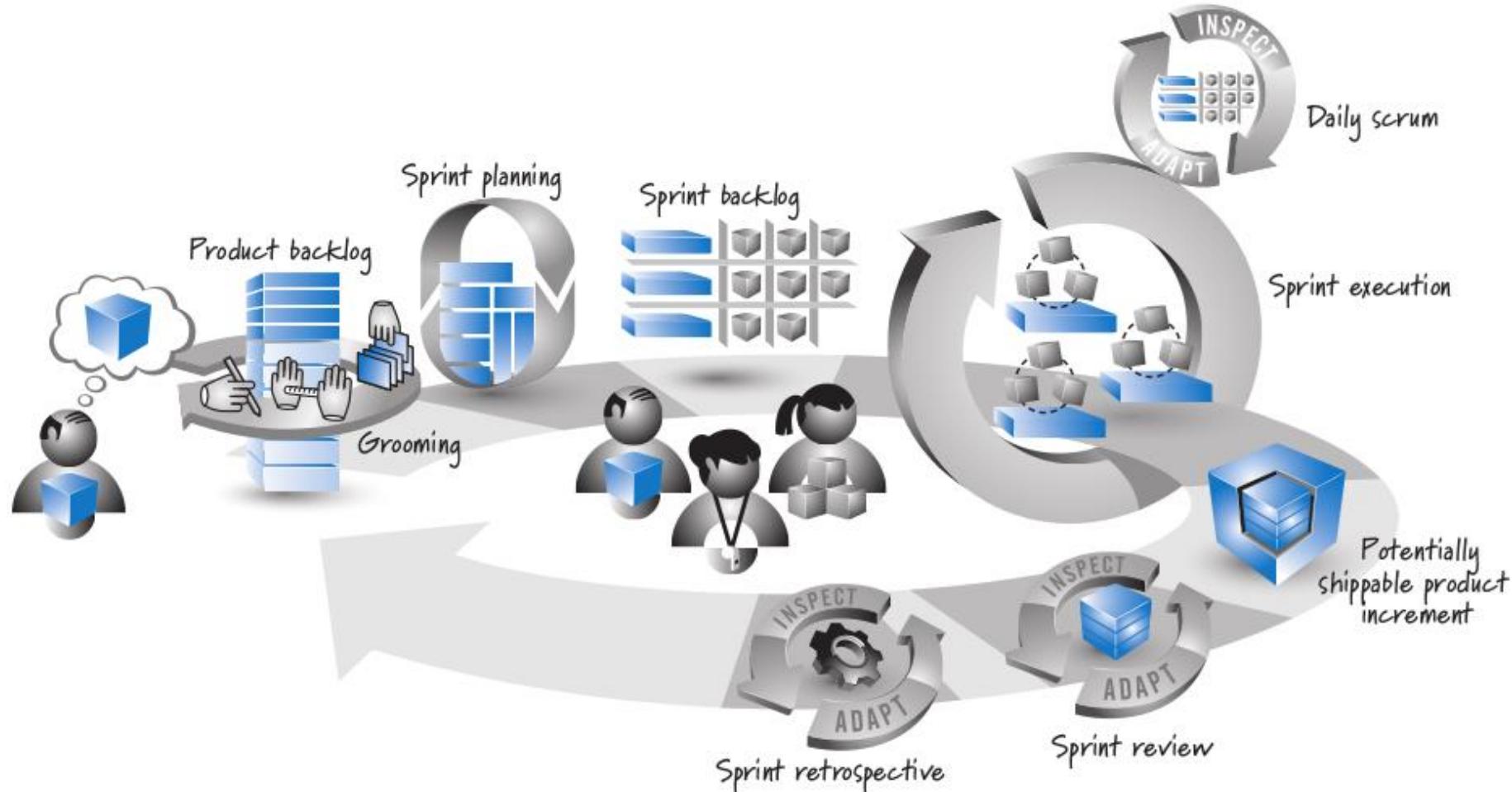
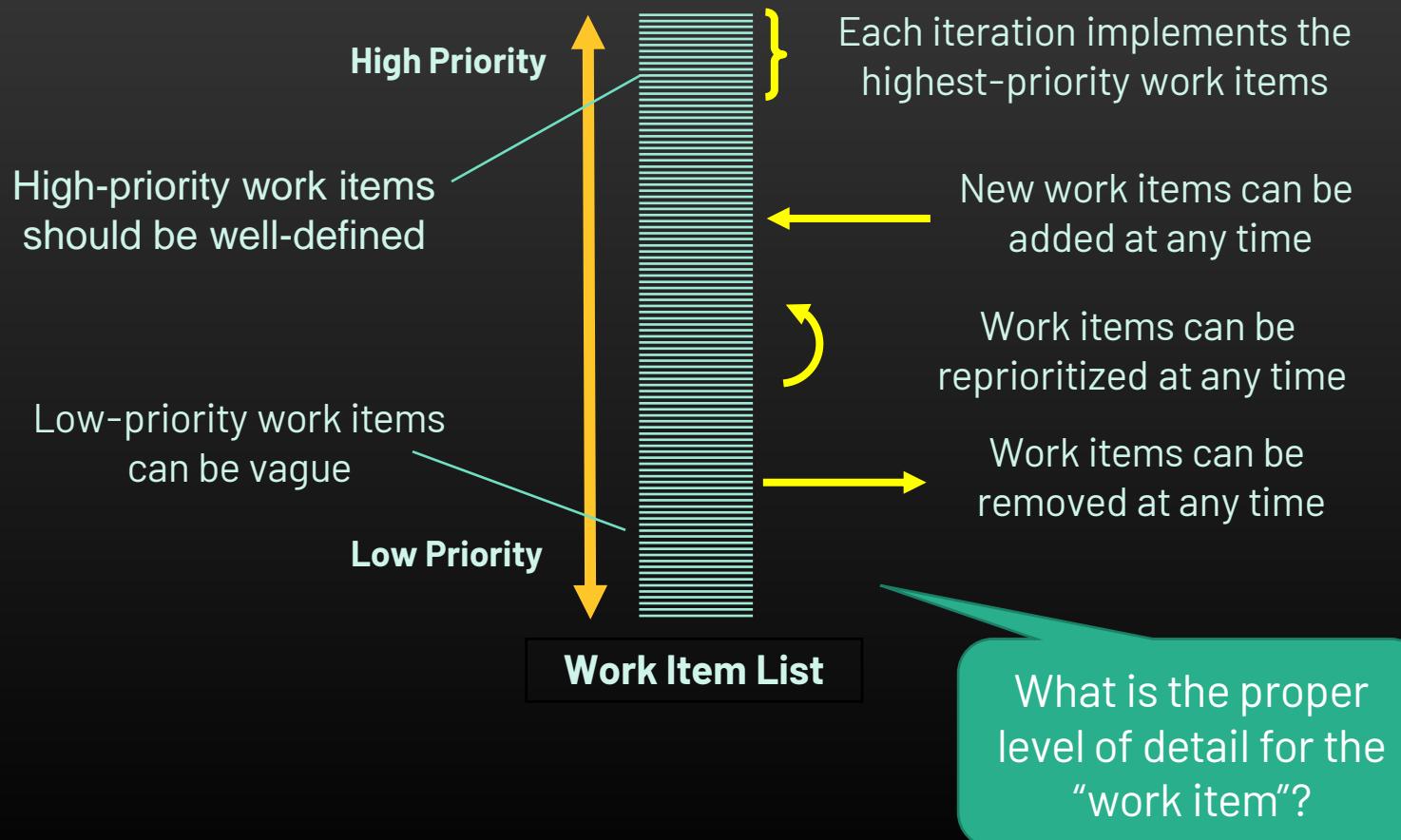
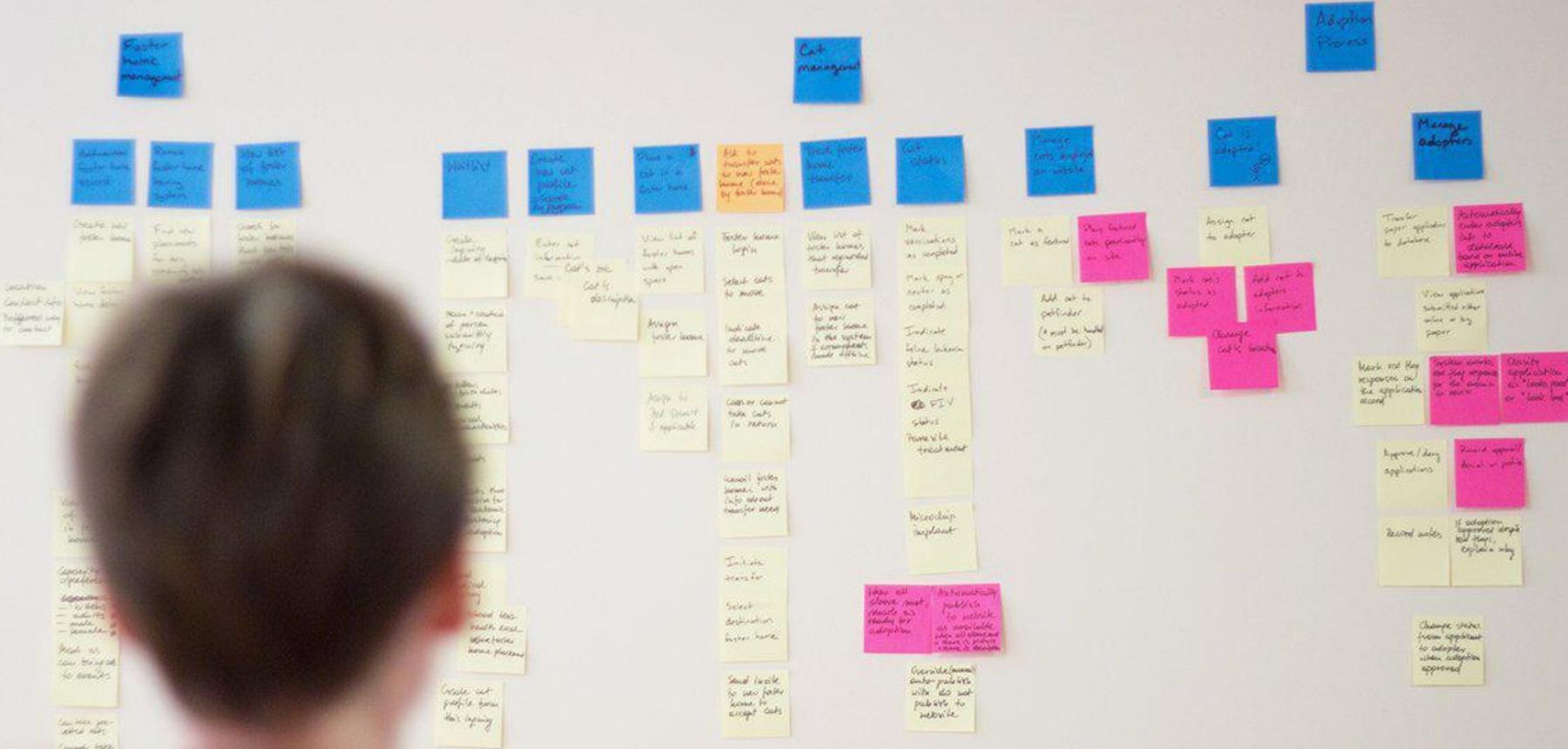


FIGURE 2.3 Scrum framework

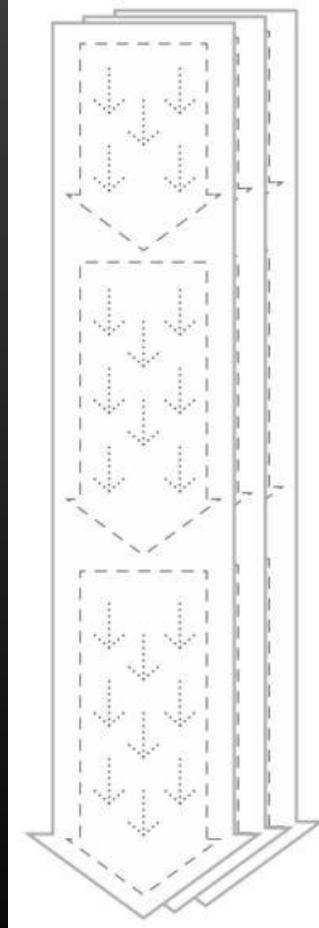
Managing the work items list (*backlog*)



Sticky note metaphor



The story should clarify how to check if it is working



"As a [persona],
I want to [do something]
so that I can [realize a reward]"

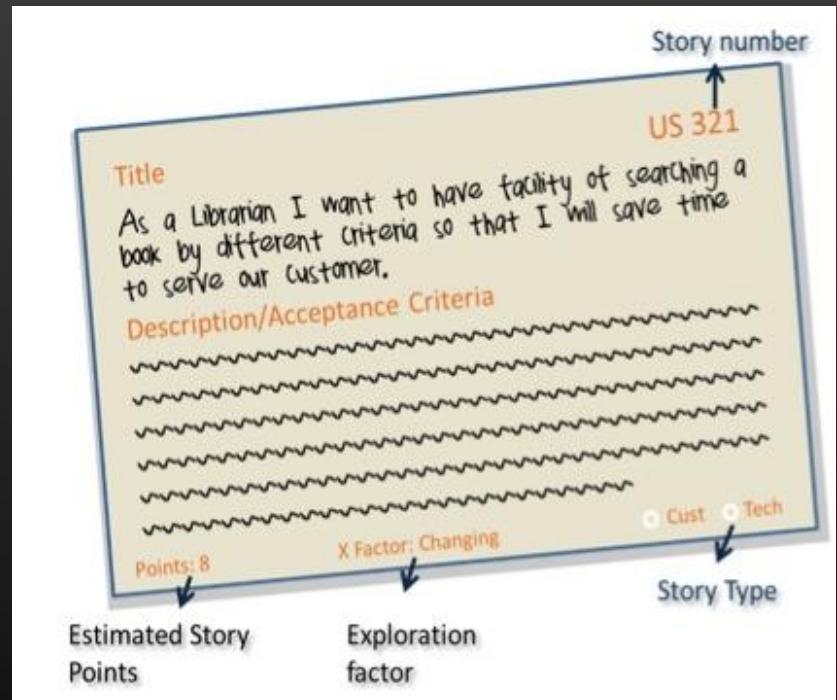
Who is this user?
What makes them tick?
Who's an example of such a person?

Why do they want to do this?
What's the benefit/reward?
How will we know of it's working?

User stories in agile methods

The *backlog* is the prioritized list of user stories –requirements– for the product and their allocation to upcoming iterations (called sprints in the agile development method called Scrum.)

User story: a “short, simple description of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system” (Cohn 2010)



User story != use case

→ See examples

Backlog granularity: **user stories**

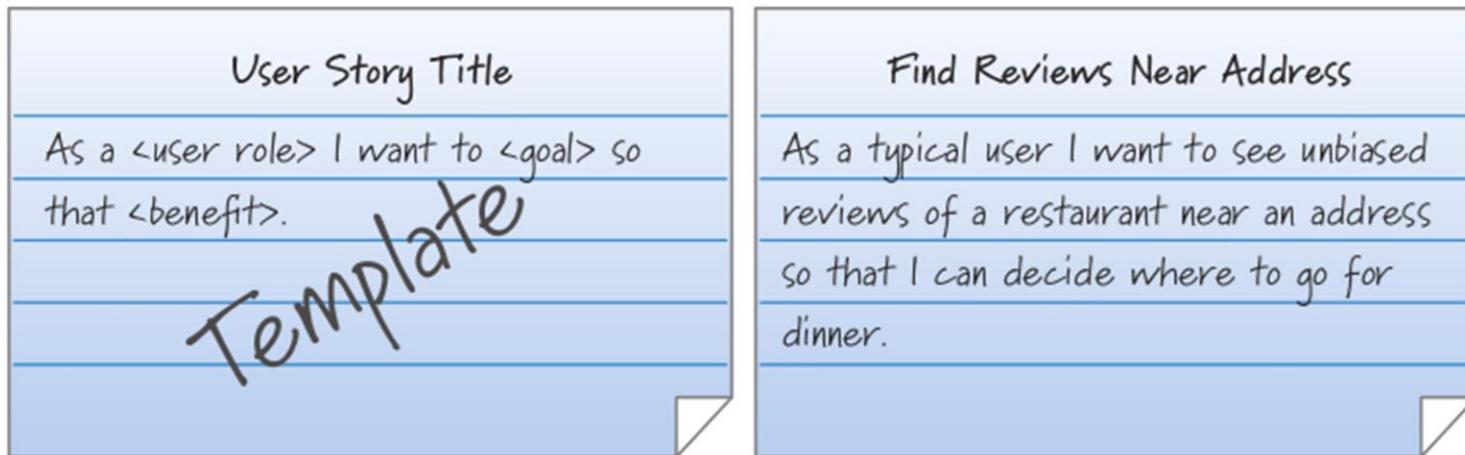
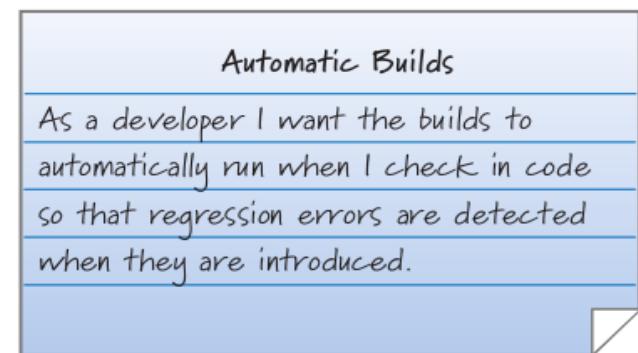


FIGURE 5.2 A user story template and card



Undesirable technical story

The user story

A “short, simple description of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system” (Cohn 2010).

User stories often are written according to the following structure (other styles also are used):

As a <type of user>, I want <some goal> so that <some reason>.

→ Advantages of the “As a user, I want” user story template.

As a customer, I want to add an item into shopping cart.

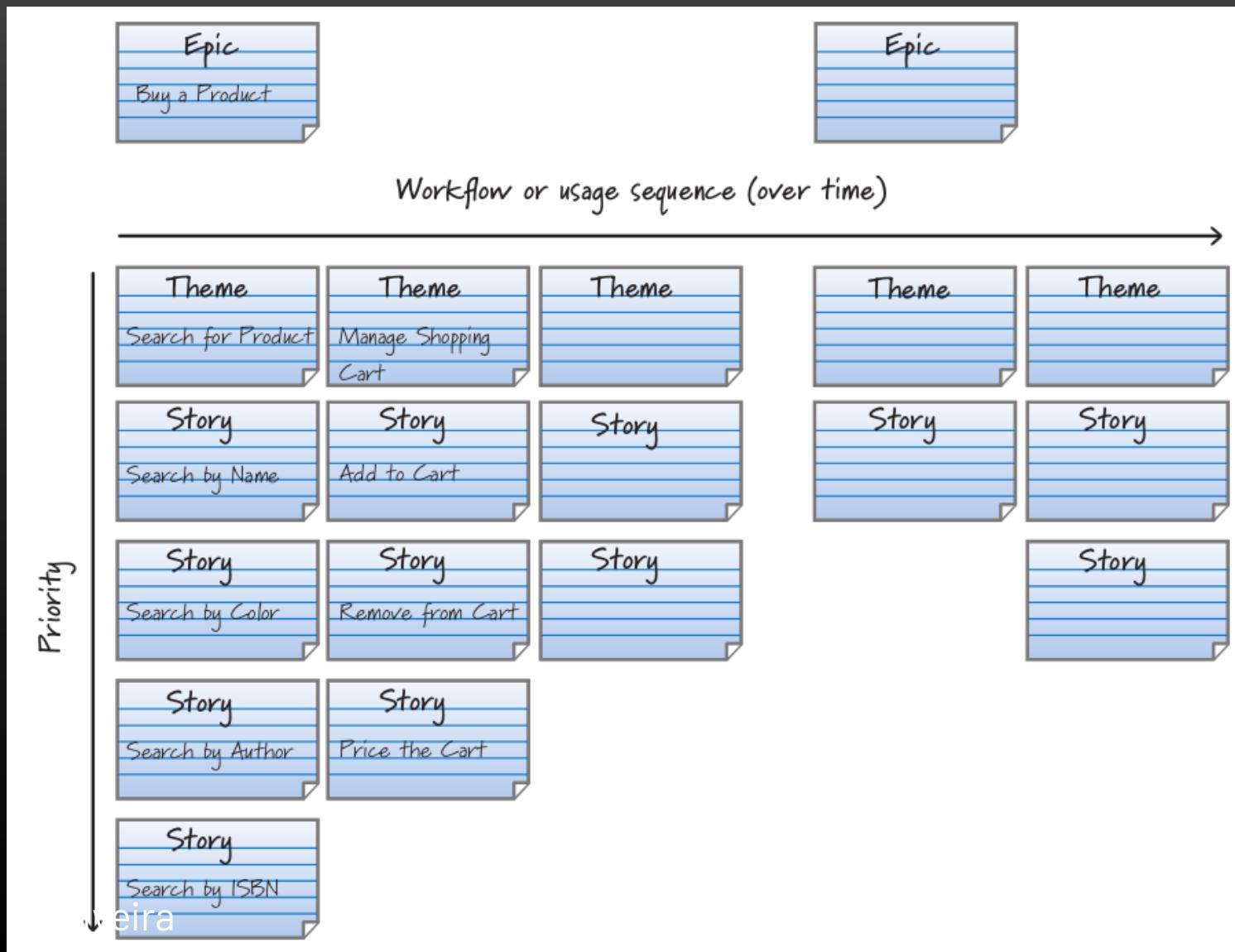
E.g. As a customer, I want to receive an SMS when the item is arrived so that I can go pick it up.

<role> represents the person, system, subsystem or any entity else who will interact with the system to be implemented to achieve a goal. He or she will gain values by interacting with the system.

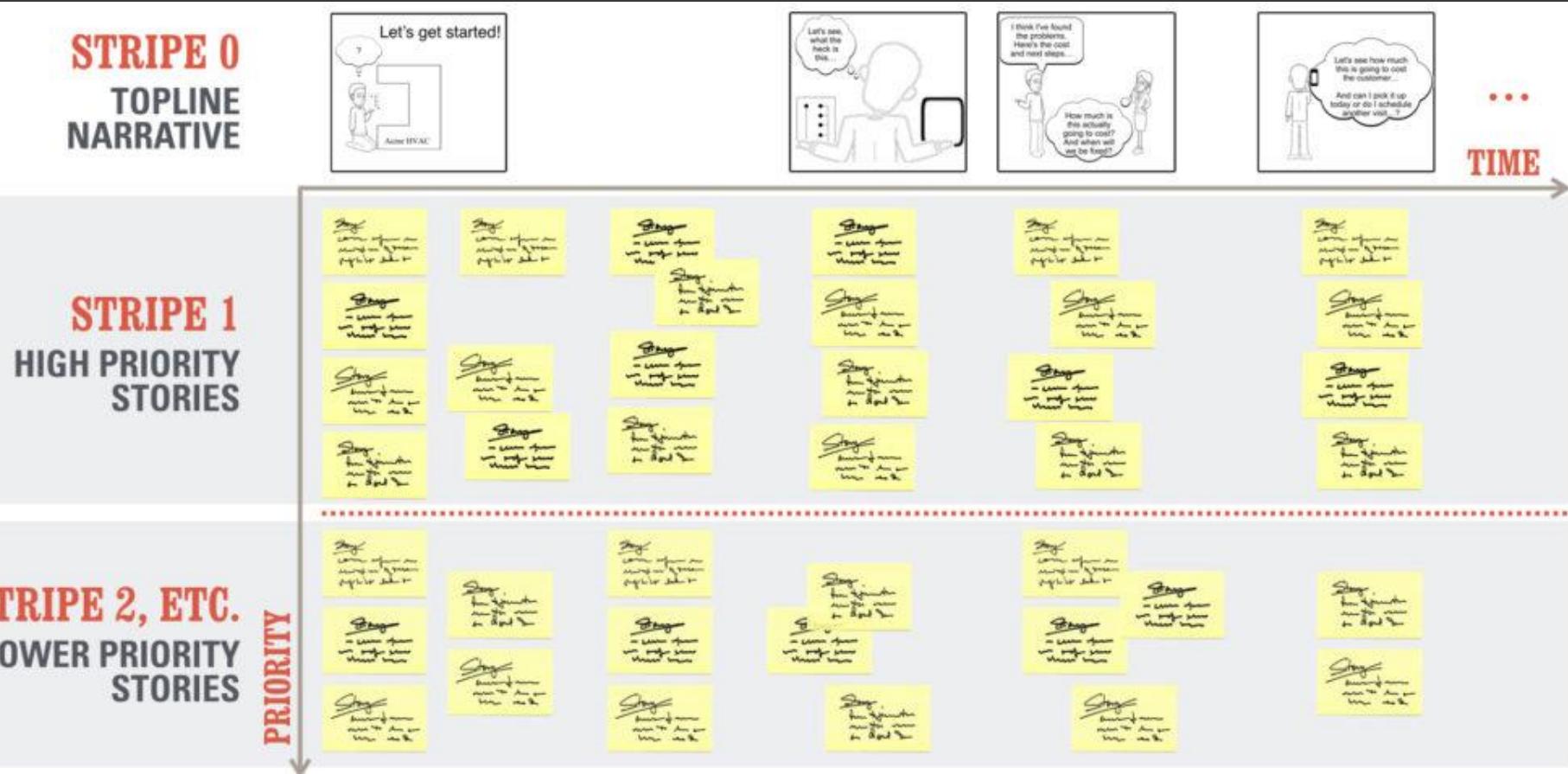
<business objective> represents a user's expectation that can be accomplished through interacting with the system.

<business value> represents the value behind the interaction with the system. May be omitted, if obvious from the business objective.

Finding good stories



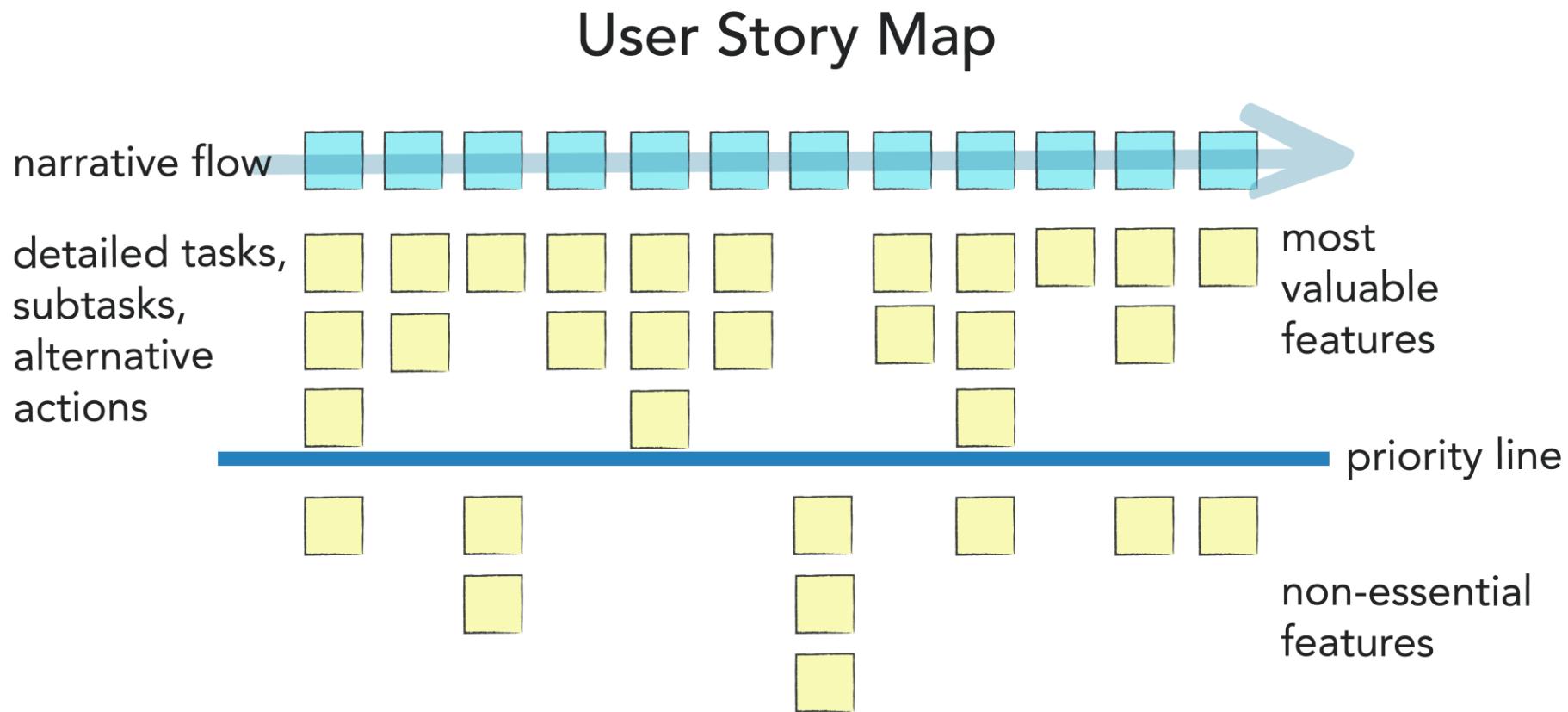
Organizing the stories in priority stripes



source: adapted from Jeff Patton's 'User Story Mapping'

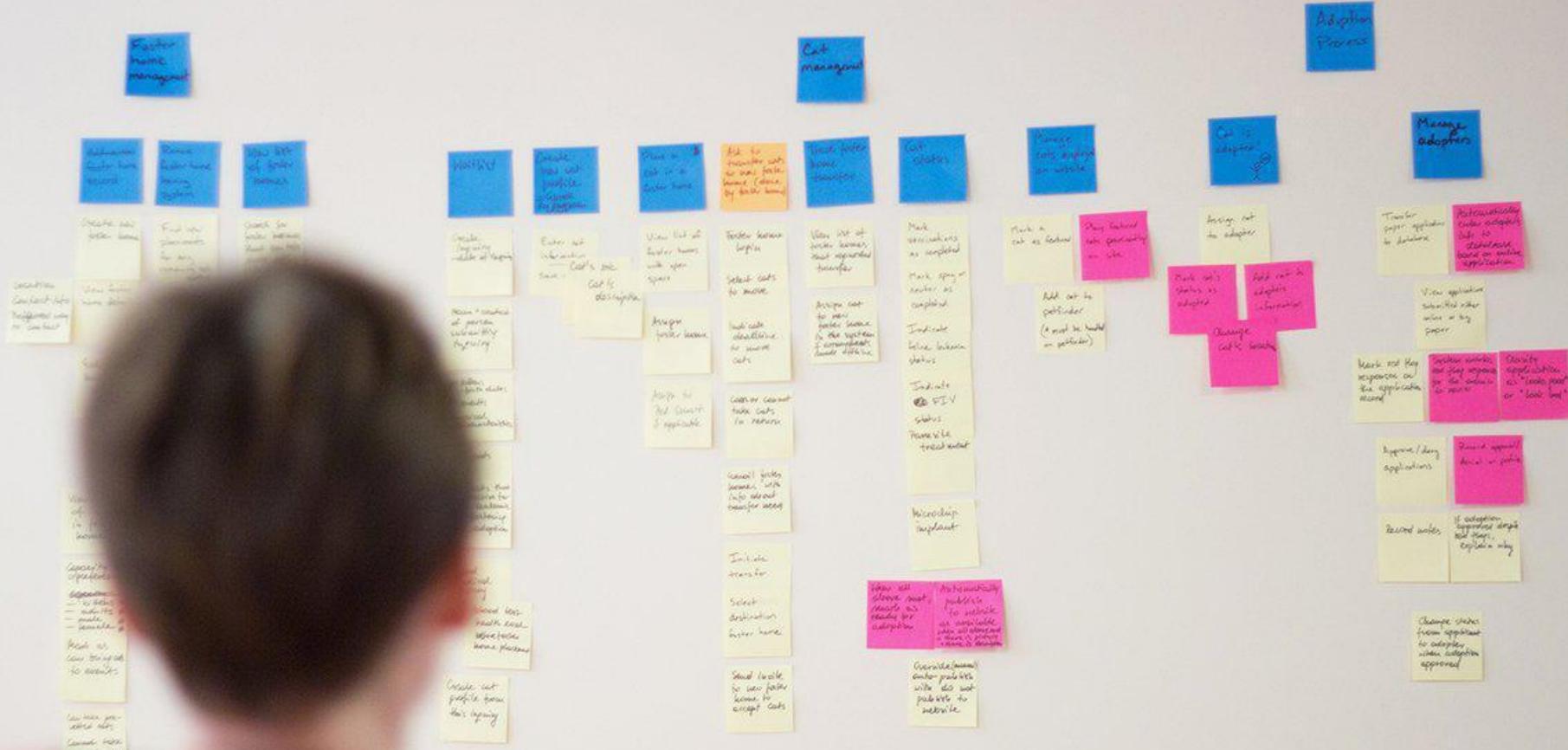
© 2015 COWAN+

The user story map



<https://www.caktusgroup.com/blog/2017/07/31/user-story-mapping-high-level-release-plan/>

Physical boards or in software tools...



Acceptance criteria

Rather than specifying functional requirements, agile teams typically elaborate a refined user story into a set of acceptance tests that collectively describe the story's "conditions of satisfaction."

The diagram illustrates a user story card and its associated conditions of satisfaction. On the left, a white rectangular card has 'Add Prospect' at the top, followed by a horizontal line, and a user story written in a cursive font: 'As a property manager I want to add a new prospect to the lead management system so I can track my interactions with the prospect.' To the right of this card is a large blue rectangular area containing two sections: 'Conditions of Satisfaction' at the top, followed by two bulleted items: 'Capture name, email, phone #, contact date, contact format, lease type, and move-in date' and 'Verify prospect is associated with an existing campaign.'

→ See examples

A structured language for writing the acceptance criteria

GIVEN [necessary context] WHEN [action] THEN [reaction].

→ Gerkin DSL.

→ Used to create AC that are meant to be automatize (feed the tests)

```
Title (one line describing the story)

Narrative:
As a [role]
I want [feature]
So that [benefit]

Acceptance Criteria: (presented as Scenarios)

Scenario 1: Title
Given [context]
  And [some more context]...
When [event]
Then [outcome]
  And [another outcome]...

Scenario 2: ...
```

User story as a collaboration context

Frank Can Add Another Person as a Friend

ID #115218319 Close

STORY TYPE	★ Feature
POINTS	0 Unestimated
STATE	Start Unscheduled
REQUESTER	RJ Ryan Jones
OWNERS	<none> +
FOLLOW THIS STORY	(1 follower) <input checked="" type="checkbox"/>

Updated: less than a minute ago

DESCRIPTION (edit)

As Frank I want to add a friend I searched for to my friend network so that I can see their posts, they can see my posts and I can direct message them

GIVEN I have searched for a friend's name
WHEN I select "Add Friend" next to my friend's name
THEN my friend's name should appear in my friend list on my homepage

Dev Notes: The added friend needs to be added to the Frank's friends in database
Design Notes: Attached are mocks for the button and placement

LABELS

add friend |x| individual user |x|

scenario: Wilson posts to his own blog
Given I am logged in as Wilson
When I try to post to "Expensive Therapy"
Then I should see "Your article was published."

scenario: Wilson fails to post to somebody else's blog
Given I am logged in as Wilson
When I try to post to "Greg's anti-tax rants"
Then I should see "Hey! That's not your blog!"

scenario: Greg posts to a client's blog
Given I am logged in as Greg
When I try to post to "Expensive Therapy"
Then I should see "Your article was published."

Stories define your project

Every project starts with a story, no matter what you're building. Tracker helps your team better develop and keep track of them while they progress from start to delivered.

Start with a good story

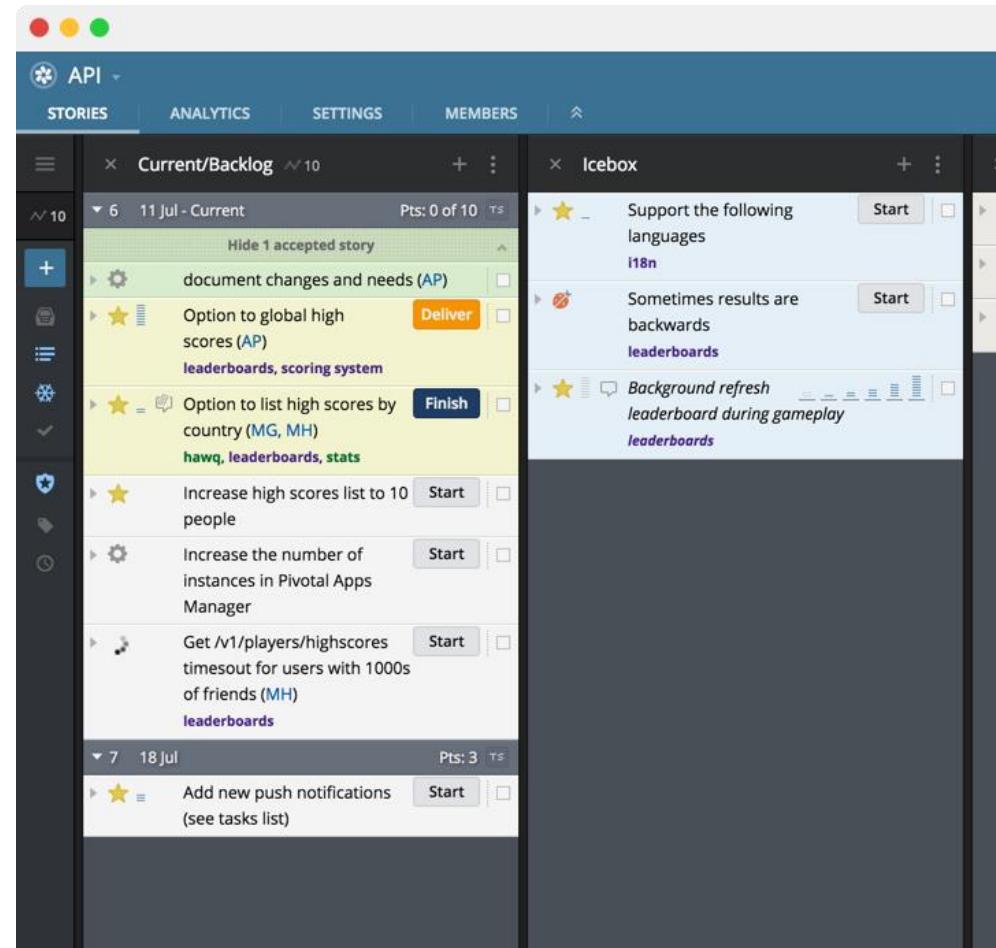
A story is a small, actionable bit of work that's either a placeholder for a future conversation or a reflection of one that already happened. Outlining what a user needs helps you focus on the what, not the how.

Define the story

Select among features, bugs, and chores to strike a healthy balance between building new features, staying ahead of technical debt, and keeping the bugs from piling up.

Estimate, then prioritize

Writing the story is just the beginning—now you get to rap about it. Estimate as a team to uncover the story's complexity. Choose among several point scales, then drag-and-drop to prioritize by iteration.



Pivotal Tracker style

Tracker lists stories in a project's Current and Backlog [panels](#) in priority order.

Dragging a story to the top of your Backlog makes it the top priority

Tracker is a "pull" scheduling system – it pulls in stories to fill the team's WIP limit, based on [velocity](#).

Tracker automatically moves stories from the top of your project's Backlog into the current iteration according to the current [velocity](#). (depending on the individual [story estimates](#)).

Tracker projects use a linear point scale (i.e., 0, 1, 2, 3). You can change

The screenshot shows the Pivotal Tracker interface with two main panels: 'Current' and 'Backlog'. The 'Current' panel on the left displays stories for the '20 Jun - Current' iteration, with a WIP limit of 8. Stories include: 'Initial demo to investors' (Priority 3), 'Admin can review all order questions and send responses to shoppers (MR)' (Priority 2, blocked by 'admin'), 'Signed in shopper should be able to post product reviews (DST)' (Priority 2), 'Product browsing pagination not working in IE6 (MR)' (Priority 2, blocked by 'admin'), 'Some product photos not scaled properly when browsing products (DST)' (Priority 2, blocked by 'shopping'), 'Request higher number of production slices, for scaling (DST)' (Priority 2, deployment), 'We may experience some slight turbulence and then...explode. (DST)' (Priority 2), 'Integrate with payment gateway (MR)' (Priority 1, blocked by 'checkout, shopping'), 'Shopper should be able to reset forgotten password (MR)' (Priority 1, blocked by 'shopping, signup / signin'), 'Cart manipulation should be AJAXy (MR)' (Priority 1, blocked by 'cart, shopping'), 'Set up Engine Yard production environment' (Priority 1, deployment, shopping), 'Beta launch' (Priority 1, blocked by 'Finish'), 'State Test' (Priority 1, blocked by 'Start'), 'Provide feedback to designer about look/feel of site' (Priority 1, blocked by 'design'), and 'Signed in shopper should be able to rate product, by choosing 1-5 stars' (Priority 1, user generated content). The 'Backlog' panel on the right lists stories for future iterations: 'When shopper is browsing products, show average product rating and number of reviews next to each product (DST)' (Priority 8, user generated content), 'Shopper should be able to read reviews for a product' (Priority 8, user generated content), 'Admin should be able to mark a product as featured' (Priority 8, admin, featured products, needs design, shopping), 'Featured products should appear on the site landing page' (Priority 8, featured products, needs design), 'When checking out, shopper should have the option to sign in to their account' (Priority 8, shopping, signup / signin), 'If authorization is successful, show order number and confirmation message to shopper' (Priority 8, checkout, needs discussion, shopping), 'Signed in shopper should be able to review order history' (Priority 8, shopper accounts), 'Signed in shopper should be able to save product to favorites' (Priority 8, shopper accounts), 'Signed in shopper should be able to review and remove product from favorites' (Priority 8, shopper accounts), 'Apply styling to all shopper facing parts of the site, based on assets from designer' (Priority 8, design), 'Admin should be able to create and edit blog articles' (Priority 8, admin, blog, needs design, shopping), 'Admin should be able to save blog articles in draft mode' (Priority 8, admin, blog), 'Signed in shopper should be able to save credit card and address information used in checkout (MR)' (Priority 8, shopper accounts), and 'Published blog articles should appear on the site' (Priority 8, blog, needs design). Stories in the backlog are color-coded by priority: gold (Priority 1), silver (Priority 2), and bronze (Priority 3).

Story Points

Funcionalidades (encomenda de comida online):

- F1: Inicio de sessão do utilizador (login)
- F2: Registo de novo utilizador na plataforma
- F3: Listar promoções em destaque do dia.
- F4: Colocar a encomenda (inclui pagamento)

Escala:

1pt: muito fácil. Direto de se implementar e âmbito reduzido.

2pts: acessível; não oferece grande dificuldade.

4pts: complexo; tem várias interdependências (de outros módulos/serviços) ou um fluxo elaborado

8pts: muito complexo; requer integrações, tecnologias ou conhecimentos que não são completamente dominados

→ <http://bit.ly/2IUrnMn>

PivotalTracker panels

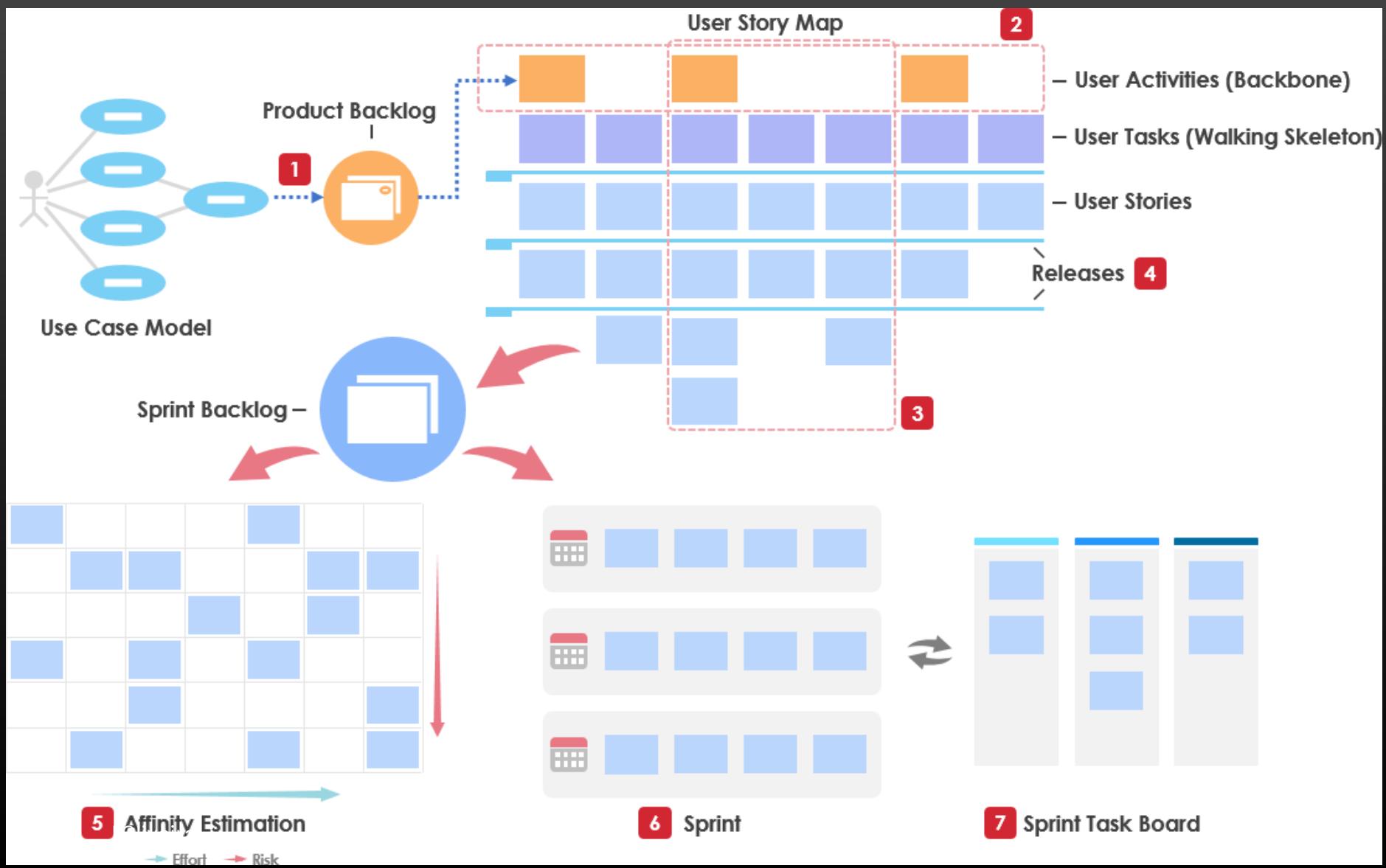
The screenshot shows the PivotalTracker interface with five main panels:

- Done:** Contains 8 stories:
 - message to shopper (checkout, needs discussion, shopping)
 - Admin can review all order questions and send responses to shoppers (admin, blocked)
 - Some product photos not scaled properly when browsing products (shopping)
 - Product browsing pagination not working in IE6 (admin, ied)
 - Signed in shopper should be able to post product reviews (user generated content)Estimated Points: 2.
- Current:** Contains 3 stories:
 - When checking out, shopper should have the option to sign in to their account (MR) (shopping, signup / signin)
 - Integrate with (Accept, Reject) (REASON FOR REJECTION (OPTIONAL): Integration fails using Alliance credits)
 - Initial demo to investors (Finish)Estimated Points: 3.
- Backlog:** Contains 3 stories:
 - Set up Cloud production environment (deployment, shopping)
 - Signed in shopper should be able to review order history (shopper accounts)
 - Beta launch (Finish)Estimated Points: 3.
- Icebox:** Contains 1 story:
 - Signed in shopper should be able to review order detailsEstimated Points: 1.

A detailed view of a story in the Current panel is shown on the right, with numbered callouts pointing to specific UI elements:

- Number 8 is inside a circle above the Done panel.
- Number 1 is inside a circle above the Icebox panel.
- Number 2 is inside a circle near the "Signed in shopper should be able to review order history" story in the Backlog panel.
- Number 3 is inside a circle near the "Apply styling to all shopper facing parts of the site" story in the Backlog panel.
- Number 4 is inside a circle near the "Set up Cloud production environment" story in the Backlog panel.
- Number 5 is inside a circle near the "Cart manipulation should be AJAXy (MR)" story in the Current panel.
- Number 6 is inside a circle near the "Shopper should be able to reset forgotten password (MR)" story in the Current panel.
- Number 7 is inside a circle near the "Accept" button in the Current panel's story card.

Agile in Visual Paradigm



Requirements elicitation by exploring user-centered scenarios

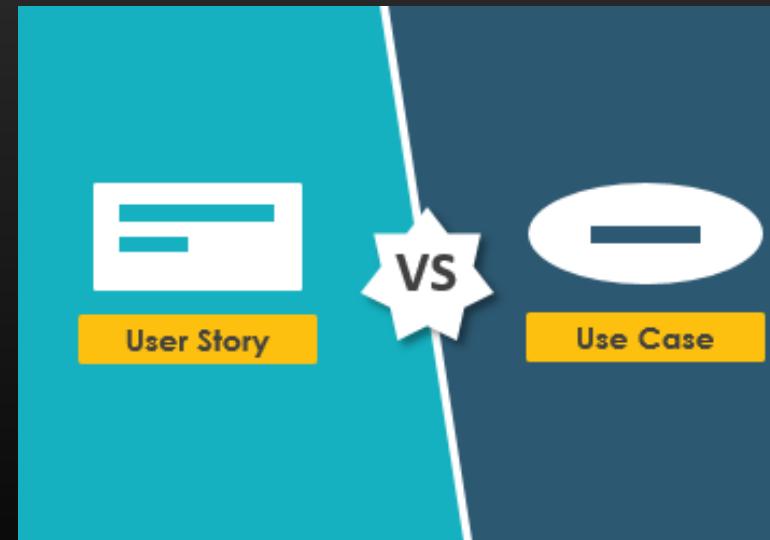
- A) Use cases
- B) User stories
- C) User-centered design (UCD)
- D) Customer Journey Map (Experience maps)

Recall: use cases and related assets

A *use case* describes a sequence of interactions between a system and an external actor that results in the actor being able to achieve some outcome of value.

The names of use cases are always written in the form of a verb followed by an object.

The use case is supplemented with a detailed description (following a template)



ID and Name:	UC-4 Request a Chemical	
Created By:	Lori	Date Created: 8/22/13
Primary Actor:	Requester	Secondary Actors: Buyer, Chemical Stockroom, Training Database
Description:	The Requester specifies the desired chemical to request by entering its name or chemical ID number or by importing its structure from a chemical drawing tool. The system either offers the Requester a container of the chemical from the chemical stockroom or lets the Requester order one from a vendor.	
Trigger:	Requester indicates that he wants to request a chemical.	
Preconditions:	PRE-1. User's identity has been authenticated. PRE-2. User is authorized to request chemicals. PRE-3. Chemical inventory database is online.	
Postconditions:	POST-1. Request is stored in the CTS. POST-2. Request was sent to the Chemical Stockroom or to a Buyer.	
Normal Flow:	4.0 Request a Chemical from the Chemical Stockroom <ol style="list-style-type: none"> Requester specifies the desired chemical. System lists containers of the desired chemical that are in the chemical stockroom, if any. System gives Requester the option to View Container History for any container. Requester selects a specific container or asks to place a vendor order (see 4.1). Requester enters other information to complete the request. System stores the request and notifies the Chemical Stockroom. 	
Alternative Flows:	4.1 Request a Chemical from a Vendor <ol style="list-style-type: none"> Requester searches vendor catalogs for the chemical (see 4.1.E1). System displays a list of vendors for the chemical with available container sizes, grades, and prices. Requester selects a vendor, container size, grade, and number of containers. Requester enters other information to complete the request. System stores the request and notifies the Buyer. 	
Exceptions:	4.1.E1 Chemical Is Not Commercially Available <ol style="list-style-type: none"> System displays message: No vendors for that chemical. System asks Requester if he wants to request another chemical (3a) or to exit (4a). a. Requester asks to request another chemical. b. System starts normal flow over. a. Requester asks to exit. b. System terminates use case. 	
Priority:	High	
Frequency of Use:	Approximately 5 times per week by each chemist, 200 times per week by chemical stockroom staff	

Use cases and similar user stories

TABLE 8-2 Some sample use cases and corresponding user stories

Application	Sample use case	Corresponding user story
Chemical tracking system	Request a Chemical	As a chemist, I want to request a chemical so that I can perform experiments.
Airport check-in kiosk	Check in for a Flight	As a traveler, I want to check in for a flight so that I can fly to my destination.
Accounting system	Create an Invoice	As a small business owner, I want to create an invoice so that I can bill a customer.
Online bookstore	Update Customer Profile	As a customer, I want to update my customer profile so that future purchases are billed to a new credit card number.

More often: user story ⊂ use case

Recall that user stories are concise statements of user needs, in contrast to the richer description that a use case provides. In the agile world, a user story sometimes covers the same scope as an entire use case, but in other cases a user story represents just a single scenario or alternative flow. If an agile development team were discussing requirements for the CTS, they might come up with user stories such as the following:

As a chemist, I want to request a chemical so that I can perform experiments.

As a chemist, I want to request a chemical from the Chemical Stockroom so that I can use it immediately.

As a chemist, I want to request a chemical from a vendor because I don't trust the purity of any of the samples available in the Chemical Stockroom.

The first of these three stories corresponds to the use case as a whole. The second and third user stories represent the normal flow of the use case and the first alternative flow, from Figure 8-3.

At this level, use cases look much like user stories. Both are focused on understanding what different types of users need to accomplish through interactions with a software system. However, the two processes move in different directions from these similar starting points, as illustrated in Figure 8-1. Both approaches can also produce other deliverables, such as visual analysis models, but Figure 8-1 illustrates the core distinction.

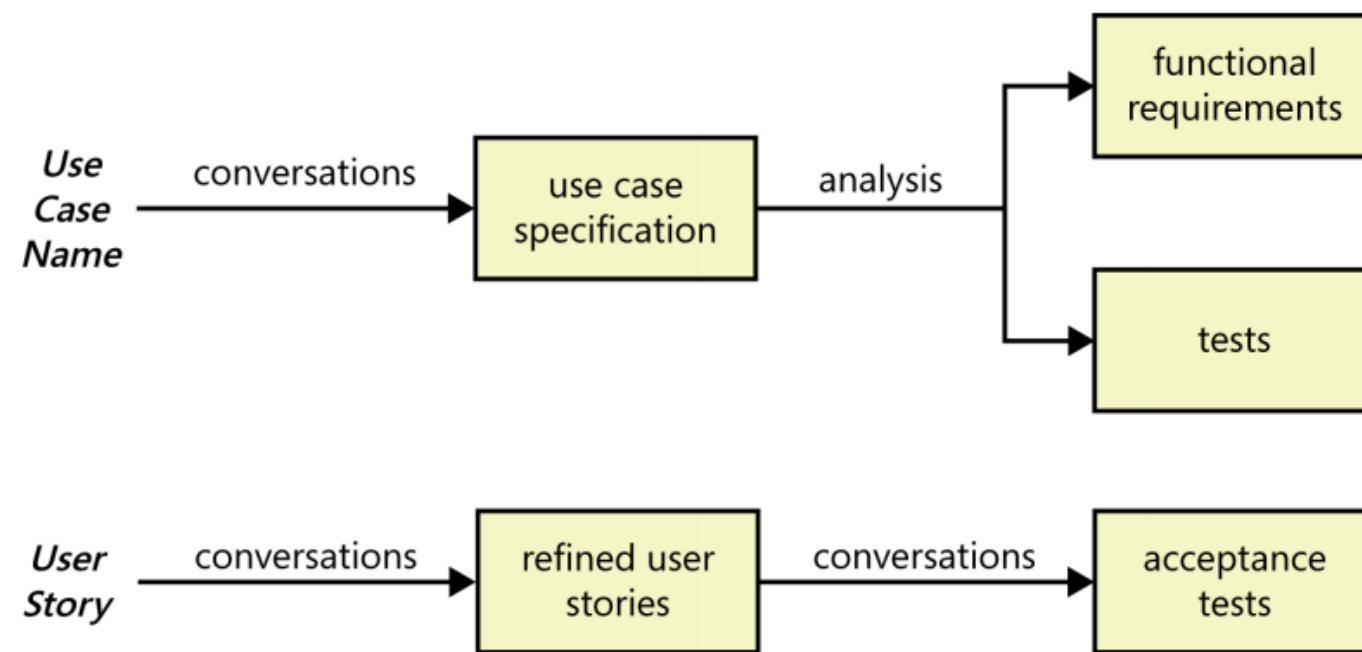


FIGURE 8-1 How user requirements lead to functional requirements and tests with the use case approach and the user story approach.

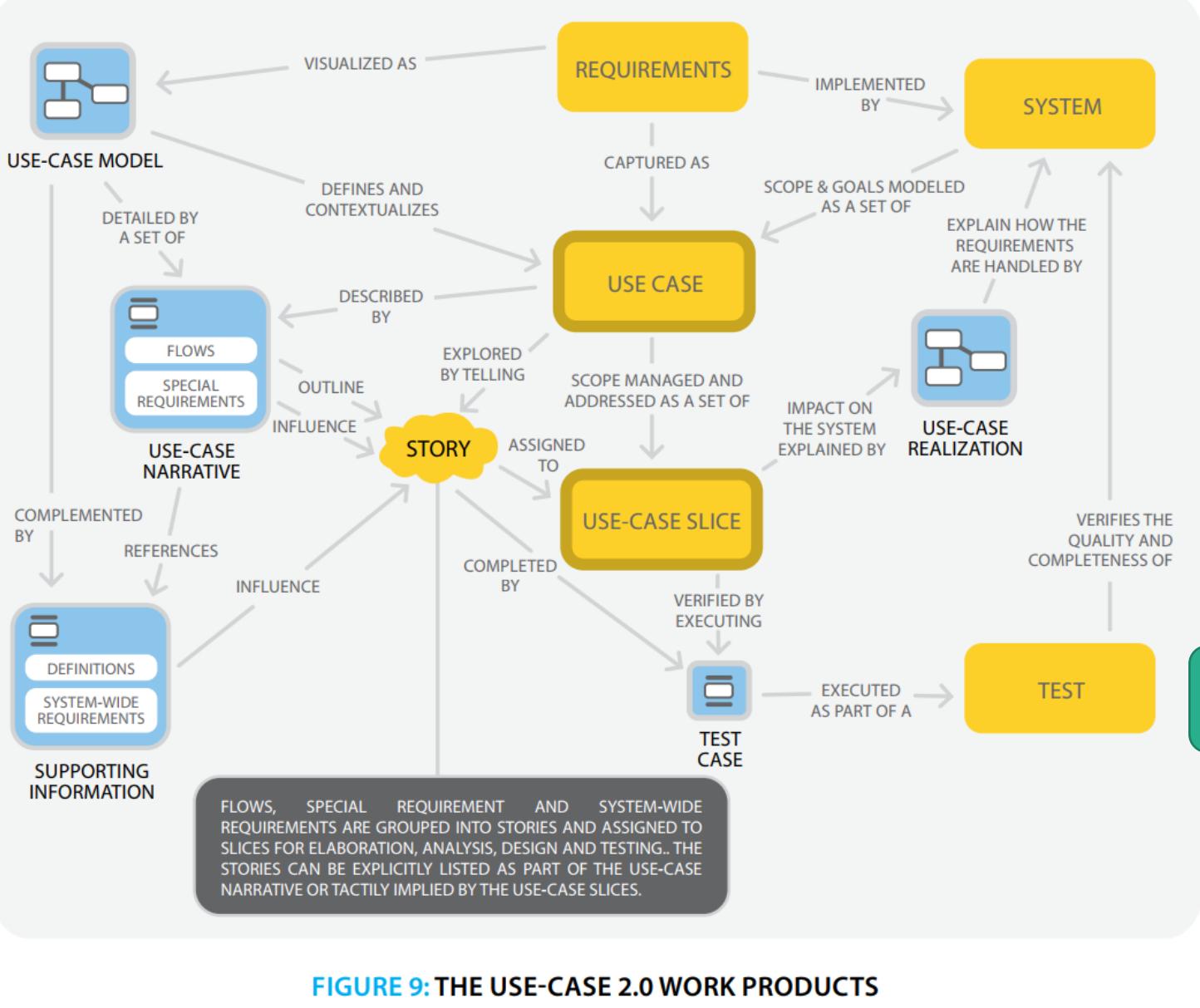
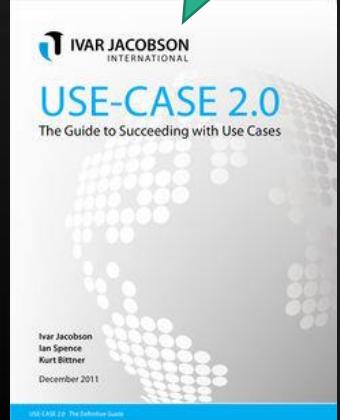


FIGURE 9: THE USE-CASE 2.0 WORK PRODUCTS

Copy in Moodle (TP-08 Resources)



Jacobson: flows in a use case match stories

A story is described by part of the use-case narrative, one or more flows and special requirements, and one or more test cases. The key to finding effective stories is to understand the structure of the use-case narrative. The network of flows can be thought of as a map that summarizes all the stories needed to describe the use case. **Figure 8** illustrates the relationship between the flows of a use-case narrative and the stories it describes.

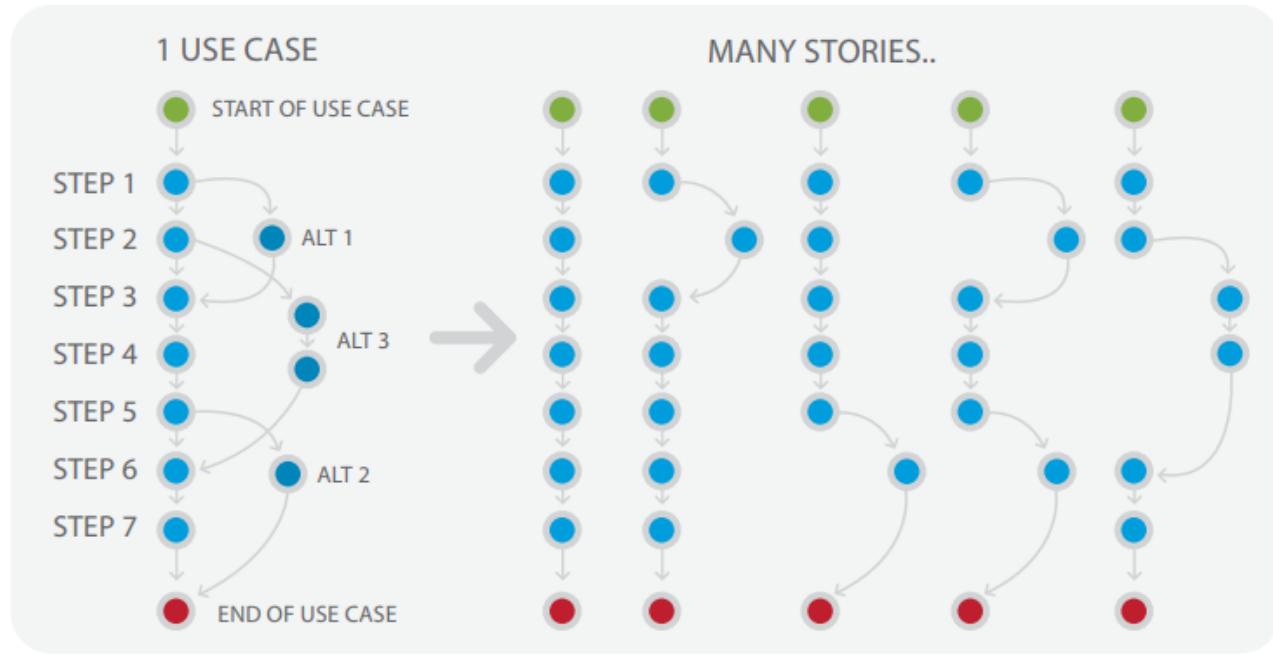


FIGURE 8:
THE RELATIONSHIP BETWEEN THE FLOWS AND THE STORIES

Figure 4. Use cases, use-case slices, increments, and releases.

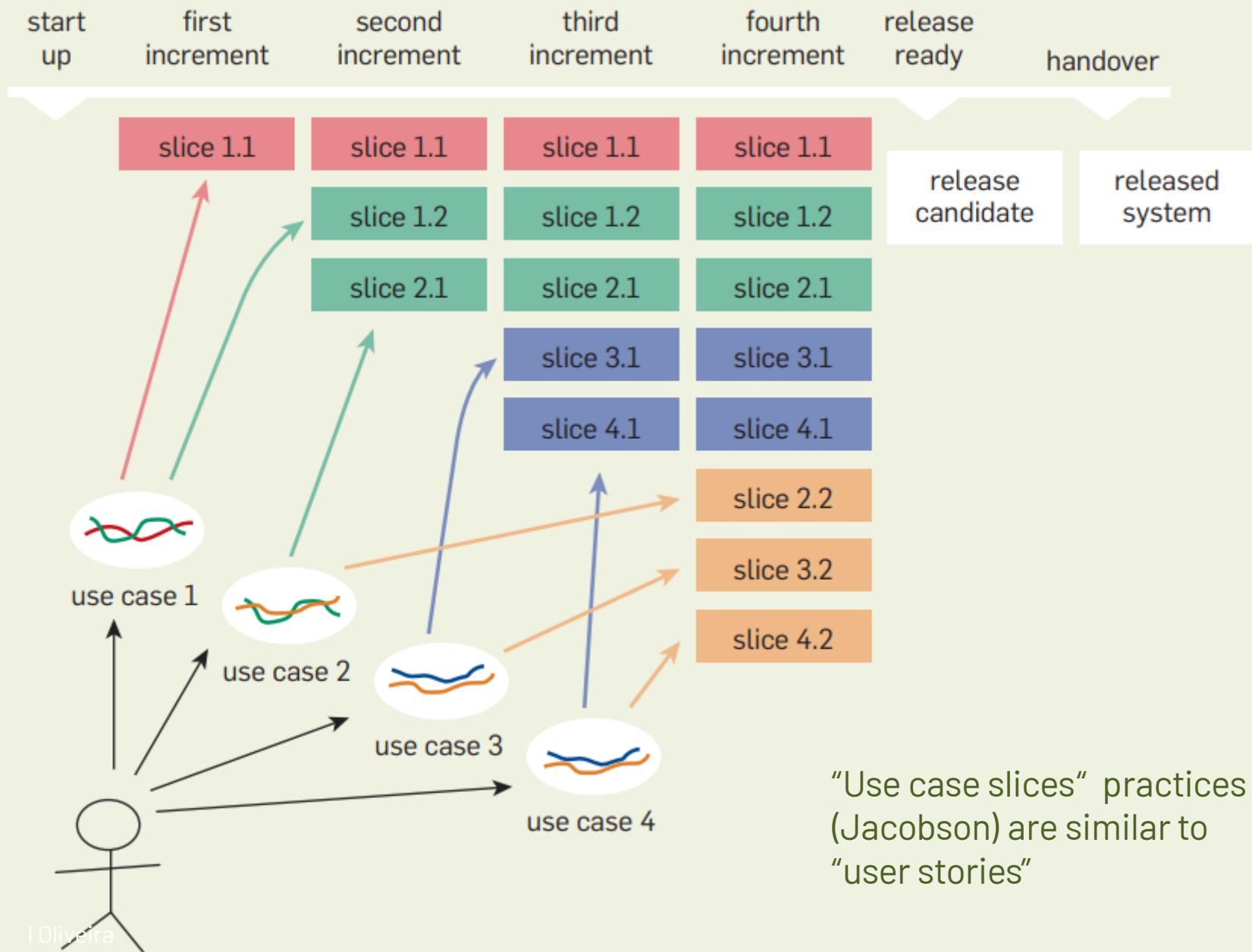


Figure 5. Capturing the properties of a use case and its slices using sticky notes.



a use case and its properties captured on a sticky note

7.1 select and buy 1 product

flows: BF
test: 1 product, default payment, valid details

5

7.2 select and buy 100 products

flows: BF
test: 100 products, default payment, valid details

5

7.3 support systems unavailable

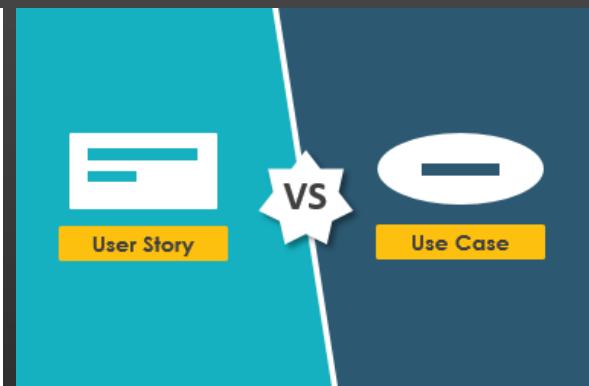
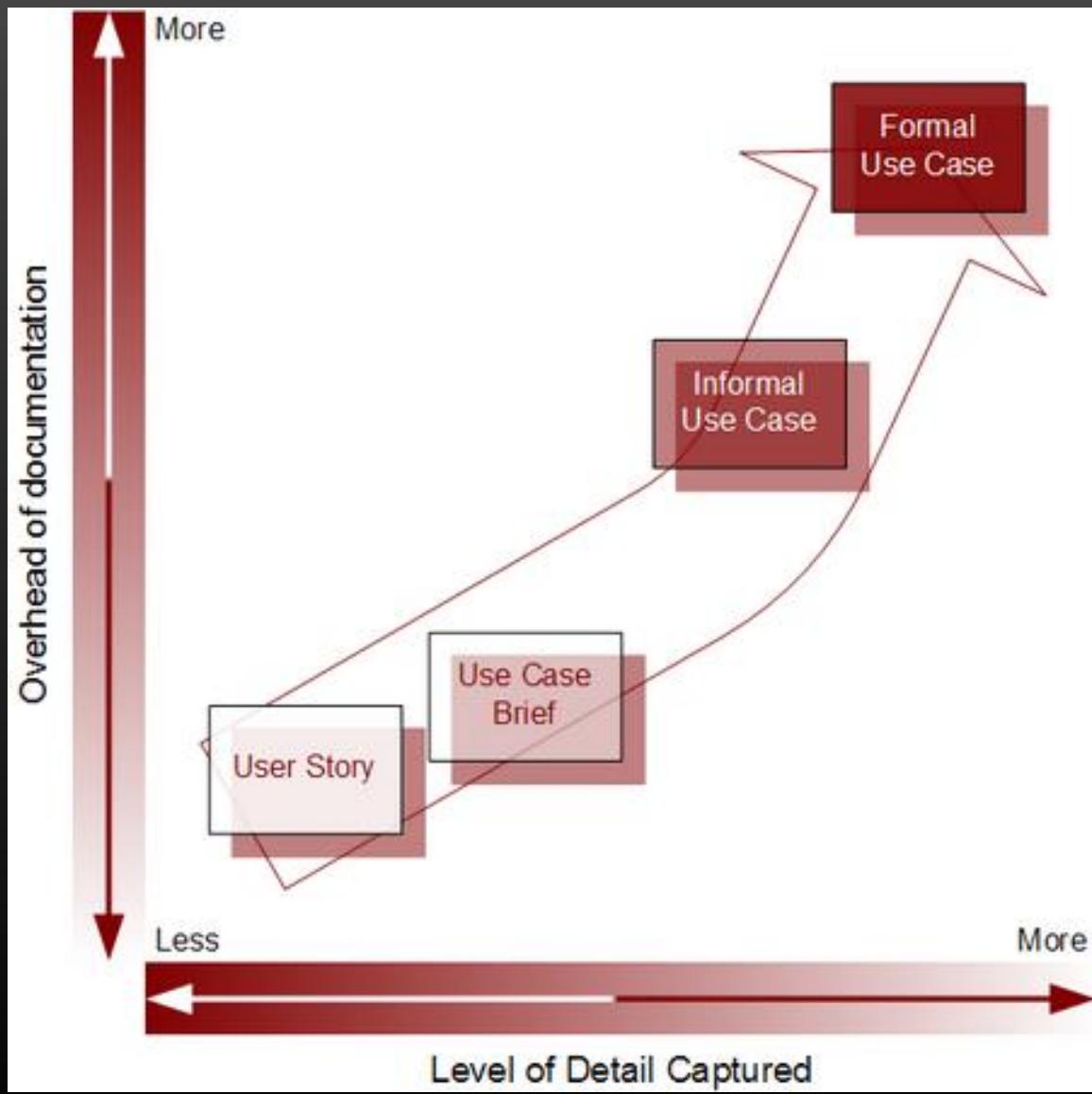
flows: BF, A9, A10, A1, A12
test: select product, provide information, disconnect each system in between

13

some slices from the use case captured on their own sticky notes

User stories and use cases: what is the difference?

Common	Use-Cases	User-story
<ul style="list-style-type: none">• Both take a usage-centric approach• Both are placeholders for a conversation• Both result in test cases that represent the acceptance criteria• Both can be estimated	<ul style="list-style-type: none">• Big picture to help people understand the extent of the system and its value• Dive further into describing how the user imagines interacting with the system to accomplish his objective.• Provide project participants with a structure and context that a collection of user stories lacks• You can examine each element of a use case (flows, preconditions, postconditions, and so on) to look for pertinent functional and nonfunctional requirements and to derive tests. This helps you avoid overlooking any requirements• Active scope management	<ul style="list-style-type: none">• Concise statement of a user's needs• Easy access to domain experts available (refine the story as needed)• More suited to act as a backlog item for daily activities (Scrum, Kanban, specification by example)• Explicit acceptance criteria



Requirements elicitation by exploring user-centered scenarios

- A) Use cases
- B) User stories
- C) User-centered design (UCD)
- D) Customer Journey Map (Experience maps)

Use cases way

ID and Name:	UC-4 Request a Chemical	
Created By:	Lori	Date Created: 8/22/13
Primary Actor:	Requester	Secondary Actors: Buyer, Chemical Stockroom, Training Database
Description:	The Requester specifies the desired chemical to request by entering its name or chemical ID number or by importing its structure from a chemical drawing tool. The system either offers the Requester a container of the chemical from the chemical stockroom or lets the Requester order one from a vendor.	
Trigger:	Requester indicates that he wants to request a chemical.	
Preconditions:	PRE-1. User's identity has been authenticated. PRE-2. User is authorized to request chemicals. PRE-3. Chemical inventory database is online.	
Postconditions:	POST-1. Request is stored in the CTS. POST-2. Request was sent to the Chemical Stockroom or to a Buyer.	
Normal Flow:	<p>4.0 Request a Chemical from the Chemical Stockroom</p> <ol style="list-style-type: none"> Requester specifies the desired chemical. System lists containers of the desired chemical that are in the chemical stockroom, if any. System gives Requester the option to View Container History for any container. Requester selects a specific container or asks to place a vendor order (see 4.1). Requester enters other information to complete the request. System stores the request and notifies the Chemical Stockroom. 	
Alternative Flows:	<p>4.1 Request a Chemical from a Vendor</p> <ol style="list-style-type: none"> Requester searches vendor catalogs for the chemical (see 4.1.E1). System displays a list of vendors for the chemical with available container sizes, grades, and prices. Requester selects a vendor, container size, grade, and number of containers. Requester enters other information to complete the request. 	

is the request and notifies the Buyer.

4.1.E1 Is Not Commercially Available

ays message: No vendors for that chemical.

Requester if he wants to request another chemical (3a) or to exit (4a).

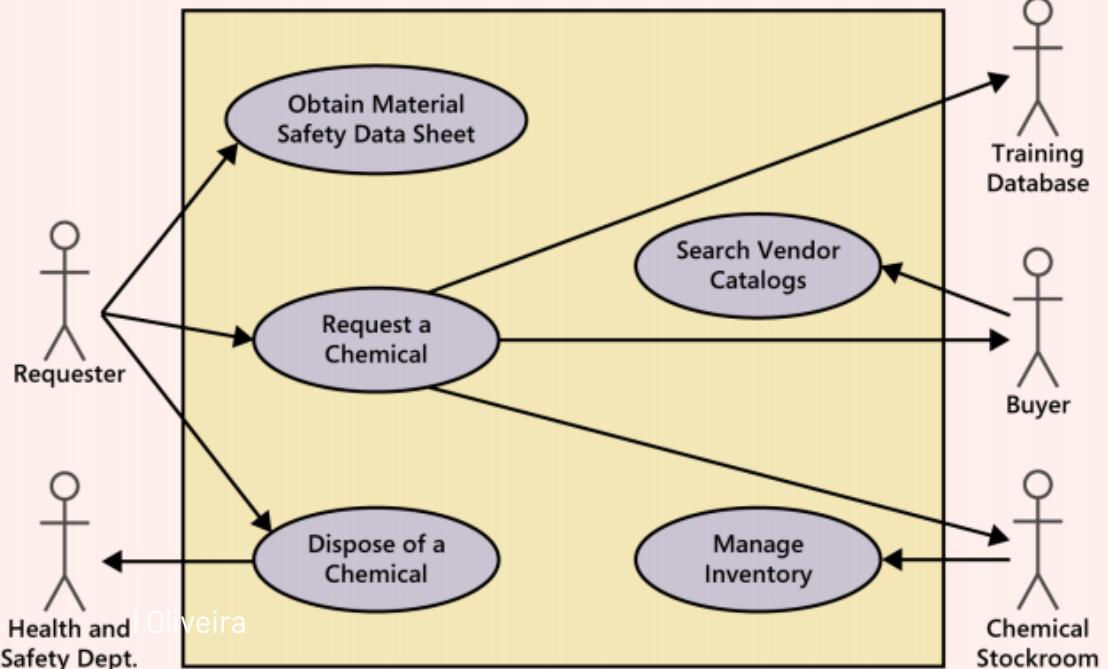
asks to request another chemical.

rts normal flow over.

asks to exit.

minates use case.

5 times per week by each chemist, 200 times per week by chemical
f



User stories

CIS board

Story Map by Easy Agile

+ Create Epic

Quick filters

Sprint swimlanes

...

?

Backlog

Navigation	CIS-1

Car Statistics	CIS-4

Phone Integration	CIS-3

Play Media	CIS-2

Fatigue Management	CIS-6

Sprint 1

The 'Young Professional' Driver / Install maps so that I can navigate to places easier	CIS-8
 2	

The 'Young Professional' Driver / Touch Screen to navigate easily	CIS-38
 1	

The 'Young Professional' Driver / Apple CarPlay Integration so that I can safely send and receive calls, texts and emails from my iOS device while driving	CIS-41
 1	

The 'Young Adult' Passenger / Allow Wifi Hotspot to support up to 5 devices	CIS-39
 1	

The 'Sunday' Driver / Enable 'Tourist Mode Assist' when travelling outside of standard travel radius	CIS-12
 2	

Sprint 2

The 'Sunday' Driver / Showcase local landmarks if travelling outside of standard travel radius	CIS-11
 3	

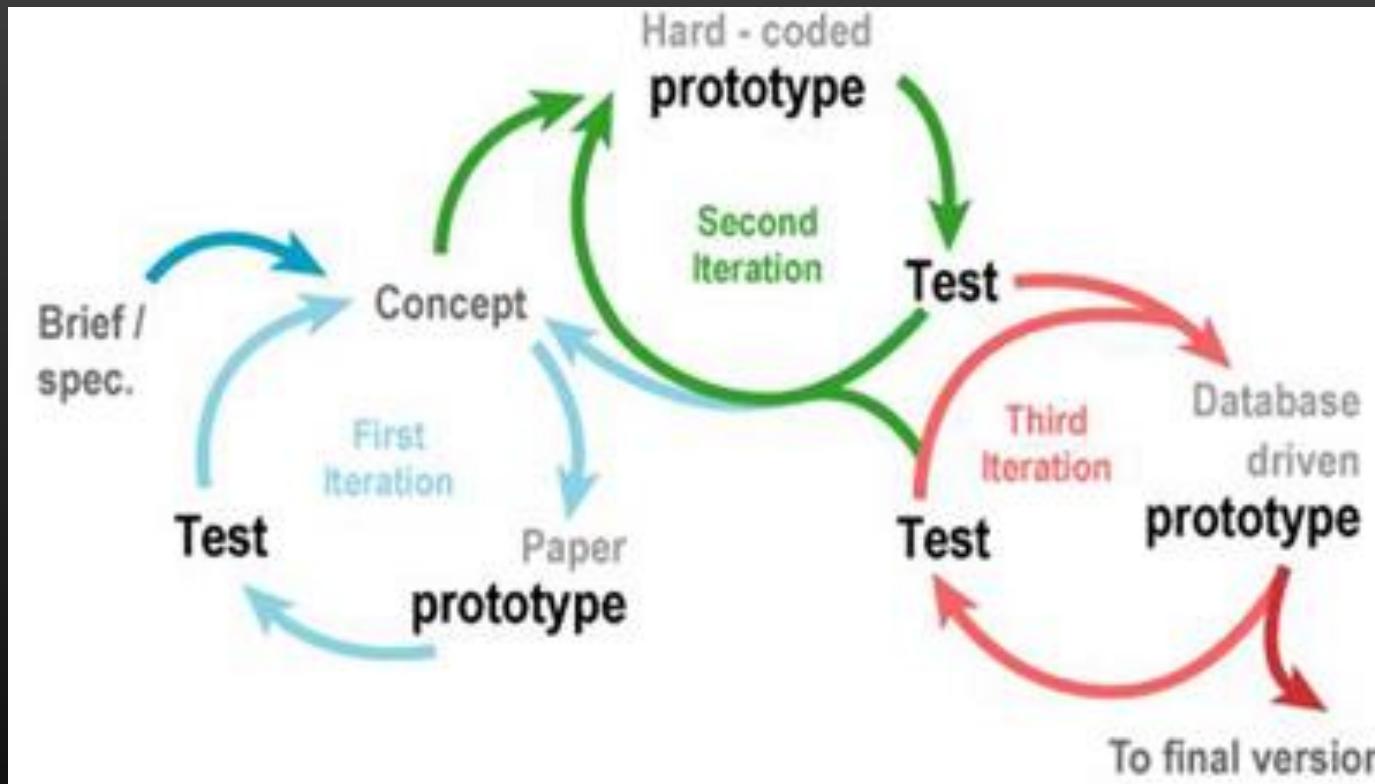
The 'Young Professional' Driver / Wear and Tear Report so that I can take preventative action to preserve the life of the car if needed	CIS-26
 5	

The 'Family' Driver / Microphone so that I can make phone calls safely while I'm driving	CIS-19
 4	

The 'Family' Driver / Graphical User Interface for easier use of media while driving	CIS-18
 3	

The 'Young Professional' Driver / Android Auto Integration so that I can safely send and receive calls, texts and emails while driving	CIS-42
 3	

UCD: prototyping & acceptance



<https://www.museumsandtheweb.com/mw2007/papers/brown/brown.html>

Rail Europe Experience Map

Guiding Principles

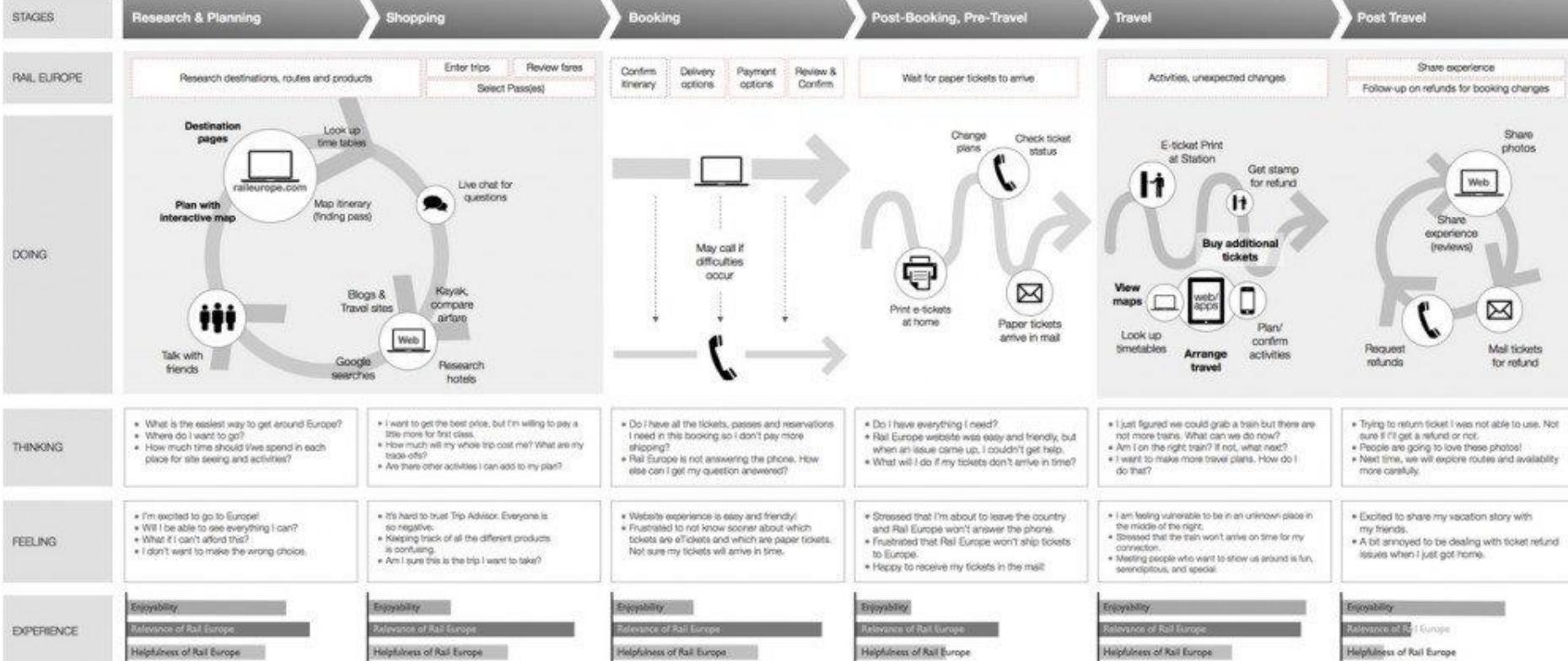
People choose rail travel because it is convenient, easy, and flexible.

Rail booking is only one part of people's larger travel process.

People build their travel plans over time.

People value service that is respectful, effective and personable.

Customer Journey



Opportunities

GLOBAL					
Communicate a clear value proposition.	Help people get the help they need.	Support people in creating their own solutions.	PLANNING, SHOPPING, BOOKING	POST-BOOK, TRAVEL, POST-TRAVEL	
STAGE: Info/Visit	STAGE: Global	STAGE: Global	Enable people to plan over time. STAGES: Planning, Shopping	Visualize the trip for planning and booking. STAGES: Planning, Shopping	Arm customers with information for making decisions. STAGES: Shopping, Booking
Make your customers into better, more savvy travelers. STAGES: Global	Engage in social media with explicit purposes. STAGES: Global		Connect planning, shopping and booking on the web. STAGES: Planning, Shopping, Booking	Aggregate shipping with a reasonable timeline. STAGE: Booking	Improve the paper ticket experience. STAGES: Post-Booking, Travel, Post-Travel
					Accommodate planning and booking in Europe too. STAGE: Traveling
					Proactively help people deal with change. STAGES: Post-Booking, Travel
					Communicate status clearly at all times. STAGES: Post-Booking, Post Travel

Information sources

Stakeholder interviews
Cognitive walkthroughs

Customer Experience Survey
Existing Rail Europe Documentation



Benefits of usage-centric requirements



The power of both use cases and user stories comes from their user-centric and usage-centric perspective. The users will have clearer expectations of what the new system will let them do than if you take a feature-centric approach. The customer representatives on several Internet development projects found that use cases clarified their notions of what visitors to their websites should be able to do. Use cases help BAs and developers understand the user's business. Thinking through the actor-system dialogs reveals ambiguity and vagueness early in the development process, as does generating tests from the use cases.

Overspecifying the requirements up front and trying to include every conceivable function can lead to implementing unnecessary requirements. The usage-centric approach leads to functionality that will allow the user to perform certain known tasks. This helps prevent "orphan functionality" that seems like a good idea but that no one uses because it doesn't relate directly to user goals.

Usage-centric approaches to requirements: benefits and limitations

Both use cases and user stories shift from the product-centric perspective of requirements elicitation to discussing what users need to accomplish, in contrast to asking users what they want the system to do.

The intent of this approach is to describe tasks that users will need to perform with the system, or user-system interactions that will result in a valuable outcome for some stakeholder.

Use cases and user stories work well for exploring the requirements for business applications, websites, kiosks, and systems that let a user control a piece of hardware.

However: they are inadequate for understanding the requirements of certain types of applications. Applications such as batch processes, computationally intensive systems, business analytics, and data warehousing might have just a few use cases. The complexity of these applications lies in the computations performed, the data found and compiled, or the reports generated, not in the user-system interactions.

Nor are use cases and user stories sufficient for specifying many embedded and other real-time systems.

Take away messages

- Agile projects (especially Scrum ones) use a product backlog, which is a prioritized list of the functionality to be developed.
- Product backlog items can be whatever the team desires, but user stories have emerged as the best and most popular form of product backlog items.
- Both use cases and user stories focus on conversations and system usage by people.
- Use cases provide more structure and a way to document details collected in analysis.
- User stories are refined as needed; focus on giving an example for a small feature.

References

Core readings	Suggested readings
<ul style="list-style-type: none">• Jacobson, I., Spence, I., & Kerr, B. (2016). <u>Use-case 2.0</u>. <i>Communications of the ACM</i>, 59(5), 61–69.• “<u>User Story vs Use Case for Agile Software Development</u>”, Visual Paradigm	<ul style="list-style-type: none">• Jacobson, I., Spence, I., & Bittner, K. (2011). <u>Use-Case 2.0</u> <i>The Guide to Succeeding with Use Cases</i>. [e-Book]• <u>User story</u> (VisualParadigm handbook)

40431: Modelação e Análise de Sistemas

Arquitetura Evolutiva

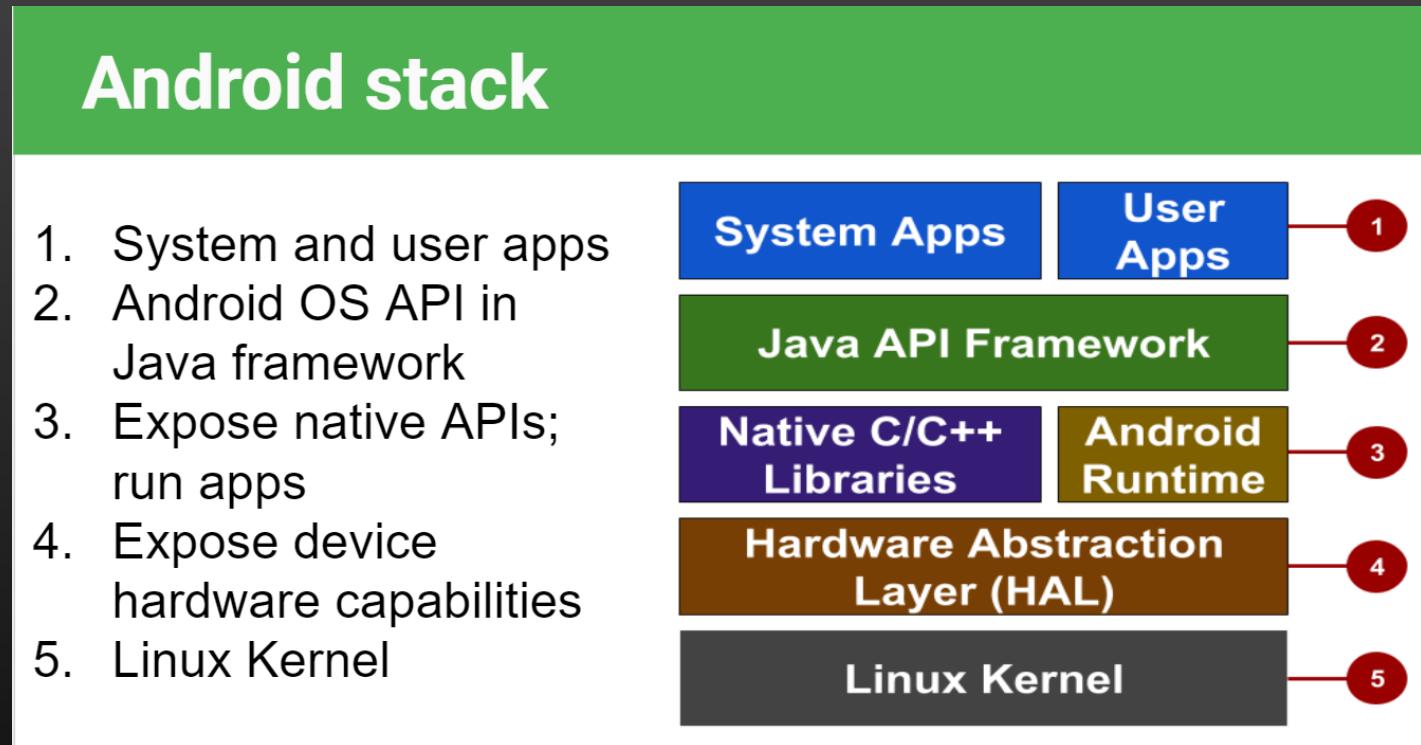
Ilídio Oliveira

v2020-12-18 | TP17b

Objetivos de aprendizagem

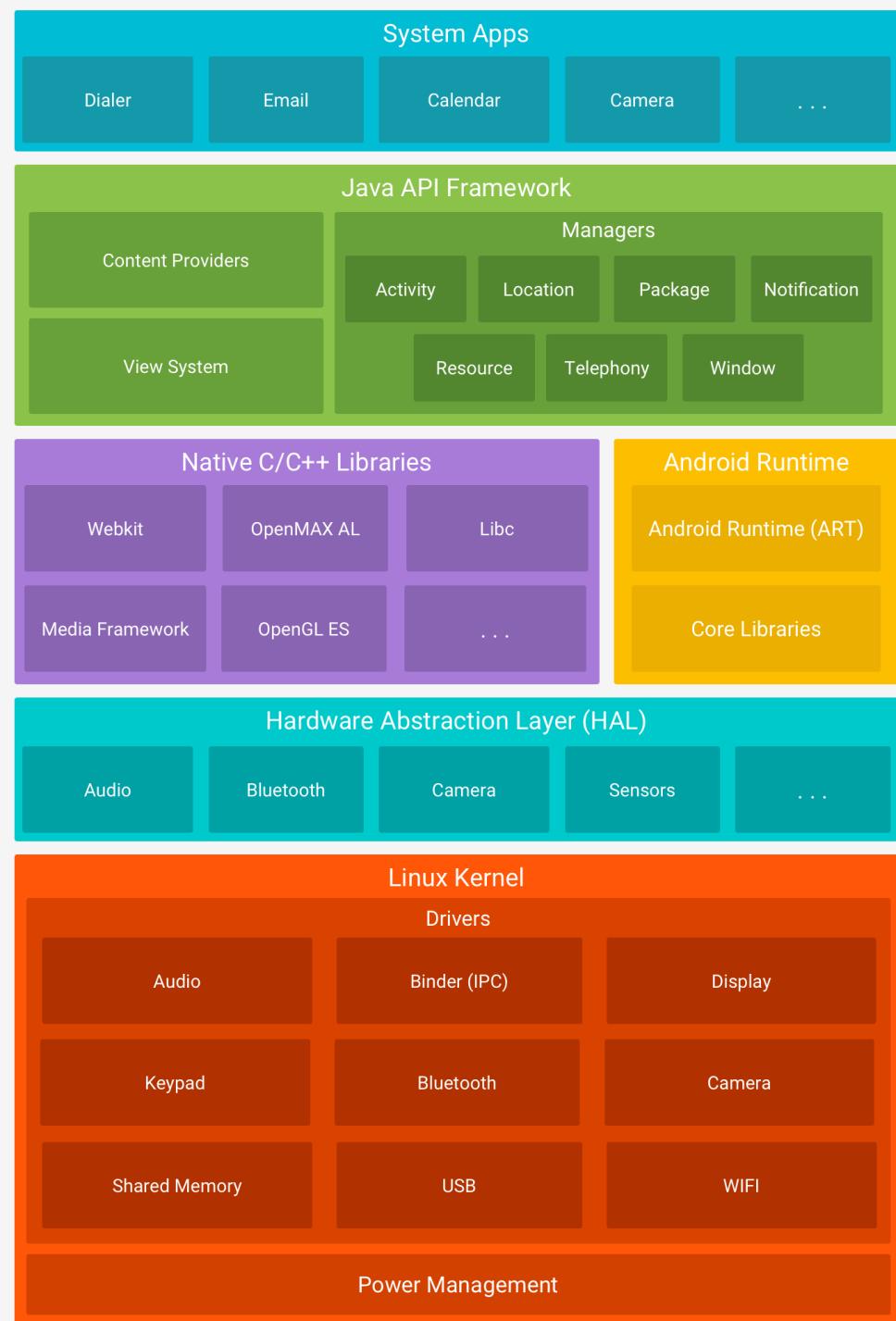
- Explicar as atividades do desenvolvimento associadas à arquitetura de software
- Definir a prática de Arquitetura Evolutiva, conforme especificado no OpenUP
- Identificar os elementos abstratos de uma arquitetura de software
- Identificar requisitos de arquitetura significativos num domínio e relacionar com atributos de qualidade
- Descreva os conceitos de camadas e partições (numa arquitetura em camadas)
- Construir um diagrama de pacotes para ilustrar uma arquitetura lógica
- Interpretar um diagrama de componentes para descrever as partes tangíveis do software
- Construir um diagrama de instalação para descrever a configuração de um sistema

Android stack (visão geral)



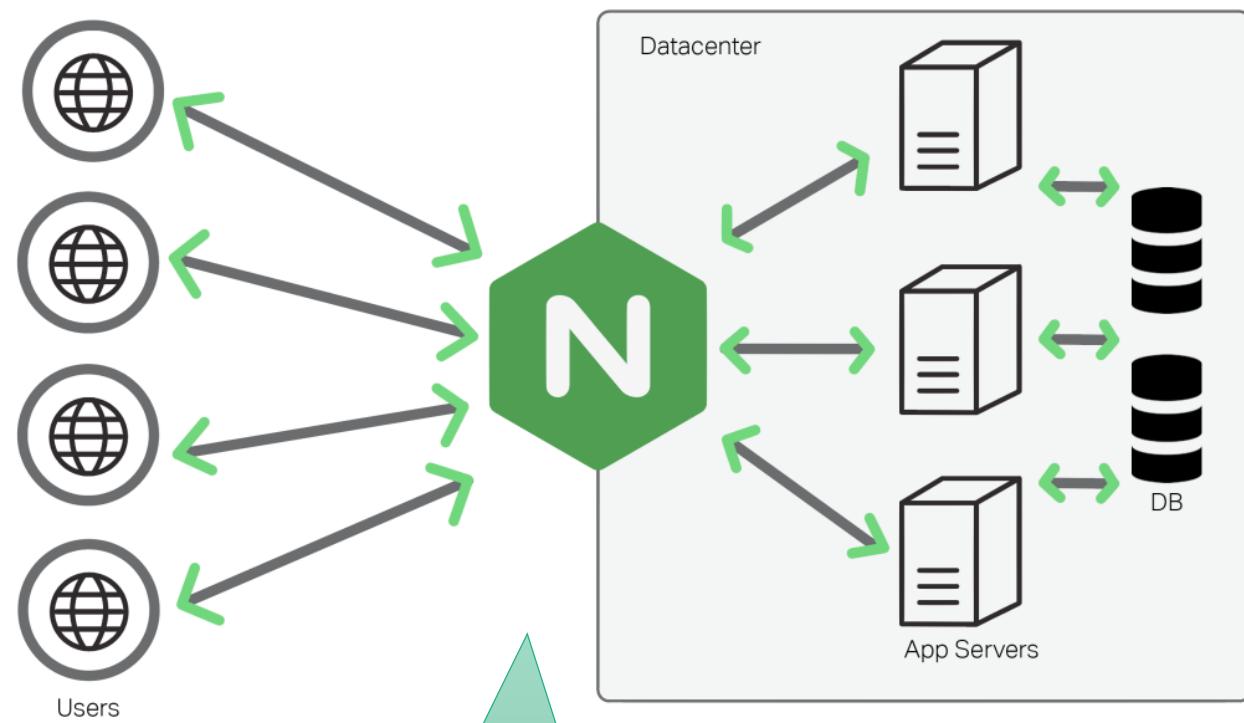
Um exemplo de arquitetura: a organização do sistema Android.

Android stack



Módulos em cada camada
(partições).

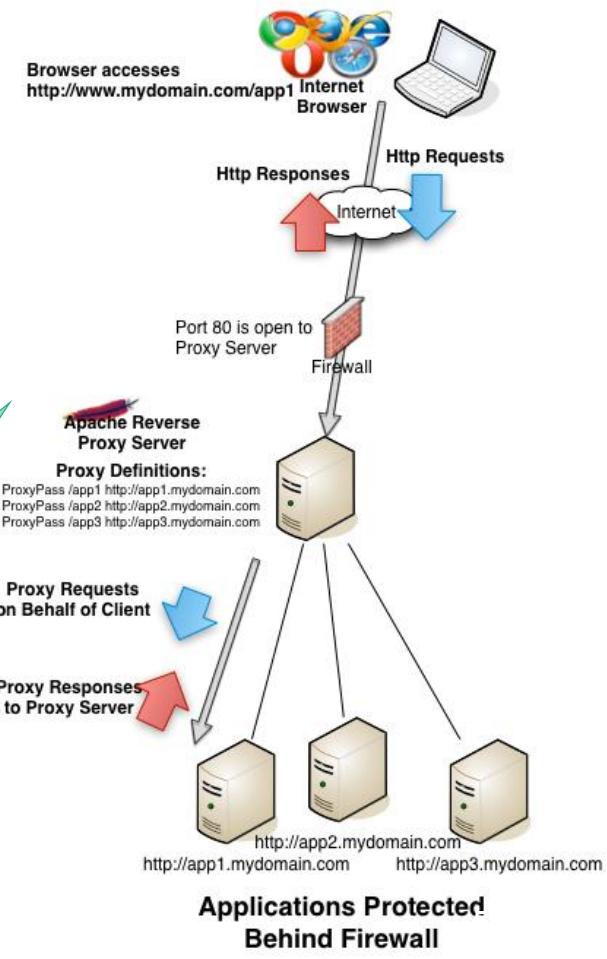
I Oliveira



Componentes organizados em 4 tiers numa solução de distribuição de carga (com Nginx).

Organização dos serviços de rede numa configuração de "reverse proxy"

Reverse Proxy Diagram



Elementos comuns

O que é que as “ilustrações” anteriores têm em comum?

- Explicar a organização de uma solução, em termos dos seus “grandes peças” (*high-level*), representando uma distribuição de responsabilidades
- Mostrar as principais linhas de dependência (colaboração/comunicação) entre os módulos
- Equilíbrio entre “Caixa aberta” (ver para dentro da solução) e “Caixa fechada” (sem mostrar a organização interna dos módulos)

Mas também há diferenças:

- Vista “lógica” (não mostra instalação) vs. vista de “sistema” (onde é que correm os componentes)

O que trata a arquitetura do software?

Concept: Software Architecture



The software architecture represents the structure or structures of the system, which consists of software components, the externally visible properties of those components, and the relationships among them.

Relationships

Related Elements

- Architecture Notebook
- Design
- Executable Architecture
- How to adopt the Evolutionary Ar

A arquitetura não é uma atividade separada do desenvolvimento / implementação. Trata as grandes decisões/estratégias para a implementação.

Main Description

Introduction

Software architecture is a concept that is **easy to understand**, and that most engineers intuitively feel, especially with little experience, but it is **hard to define precisely**. In particular, it is difficult to draw a sharp line between design and architecture-architecture is one aspect of design that concentrates on some specific features.

In An Introduction to Software Architecture, David Garlan and Mary Shaw suggest that software architecture is a level of design concerned with issues: "Beyond the algorithms and data structures of the computation: designing and

Software architecture

An architecture is **the set of significant decisions about the organization of a software system**

Plano (decisões) com a estratégia para a implementação.

...the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization these elements and their interfaces, their collaborations, and their composition. [BRJ99]

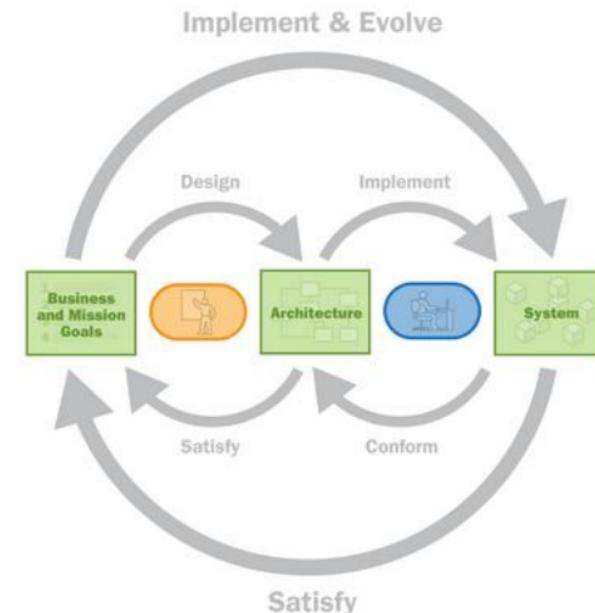
→ Big ideas on system organization and interactions



Why Architecture?

The software architecture of a program or computing system is a depiction of the system that aids in understanding how the system will behave.

Software architecture serves as the blueprint for both the system and the project developing it, defining the work assignments that must be carried out by design and implementation teams. The architecture is the primary carrier of system qualities such as performance, modifiability, and security, none of which can be achieved without a unifying architectural vision. Architecture is an artifact for early analysis to make sure that a design approach will yield an acceptable system. By building an effective architecture, you can identify design risks and mitigate them early in the development process.



work/display.cfm?customel_datapageid_4050=21328

Who Needs an Architect?

Outra maneira de colocar a questão...

In the words of Ralph Jordan:

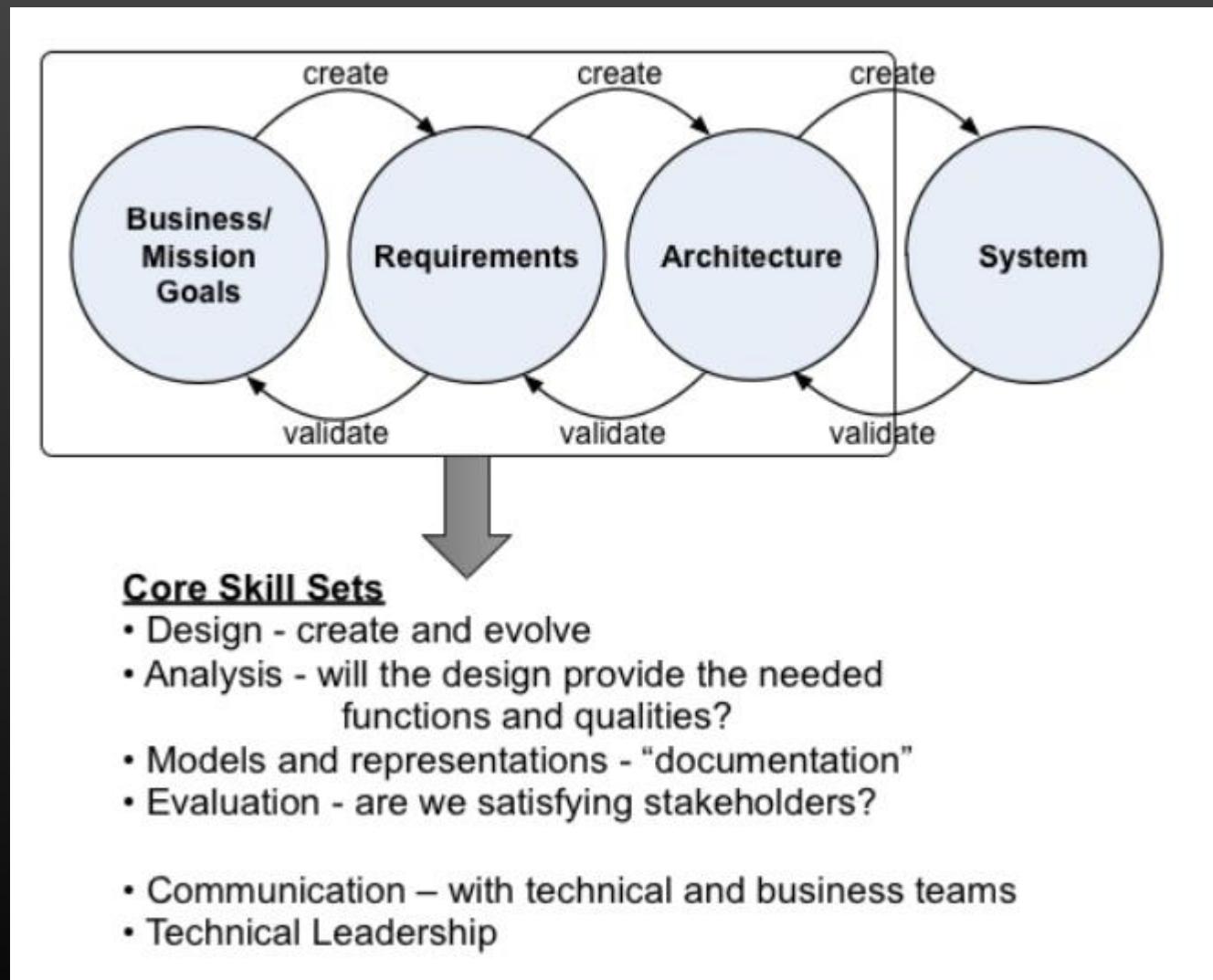
So, a better definition would be "In most successful software projects, the expert developers working on that project have a shared understanding of the system design. This shared understanding is called 'architecture.' This understanding includes how the system is divided into components and how the components interact through interfaces. These components are usually composed of smaller components, but the architecture only includes the components and interfaces that are understood by all the developers."

Martin Fowler

<https://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf>

There is another style of definition of architecture which is something like "architecture is the set of design decisions that must be made early in a project." I complain about that one, too, saying that architecture is the decisions that you wish you could get right early in a project, but that you are not necessarily more likely to get them right than any other.

Papel do arquiteto (de software)



<https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=455074>

Assuntos da Arquitetura do sistema

Organização estrutural do sistema em grandes blocos

Componentes desenvolvidos e/ou integrados

Topologia física: servidores, rede,...

Uma arquitetura é definida para satisfazer os requisitos

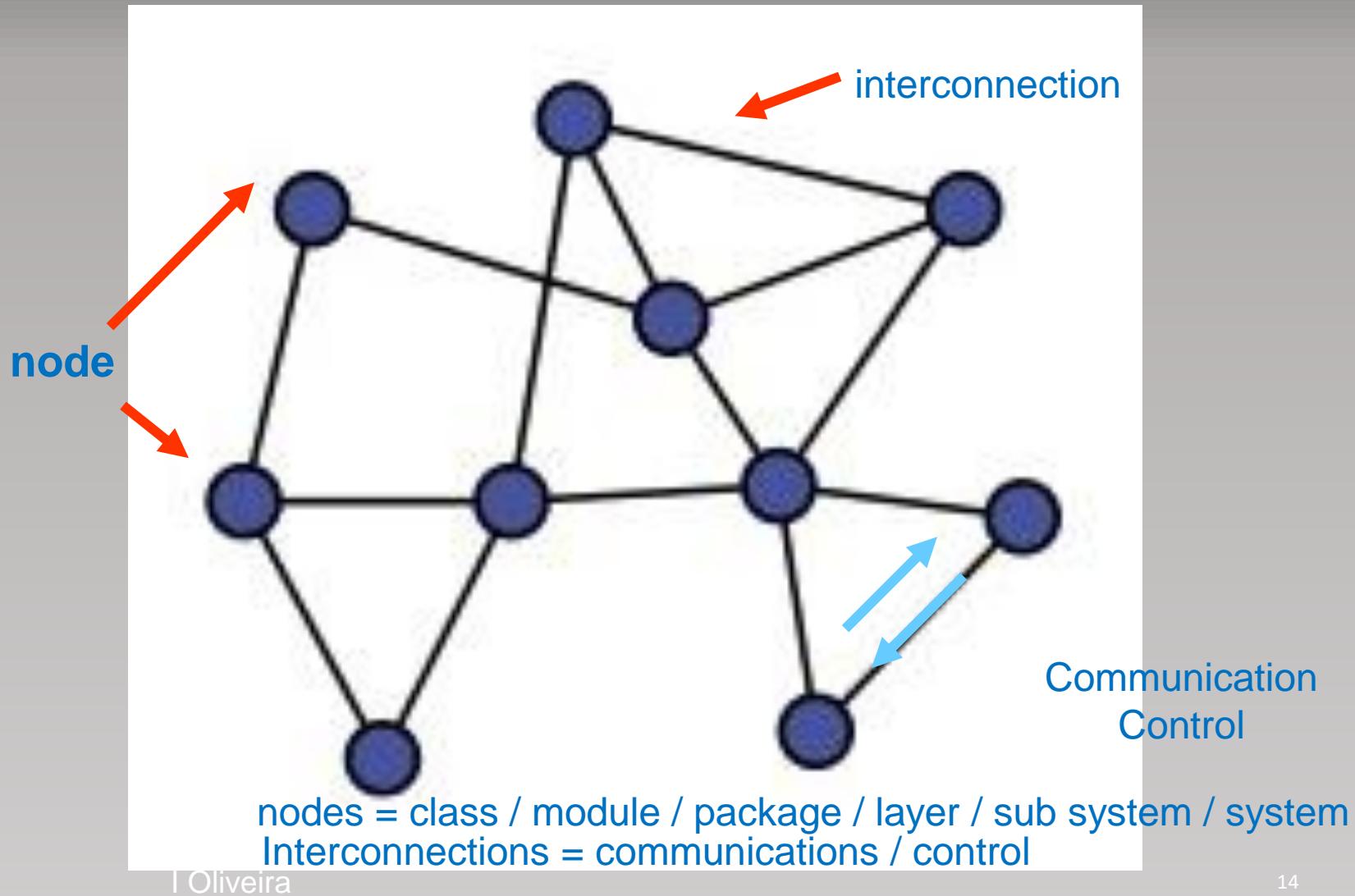
Requisitos não funcionais e restrições de operação são determinantes

E.g.: sessões simultâneas?

Licenciamento? Tolerância a falhas?

Compromissos e decisões!

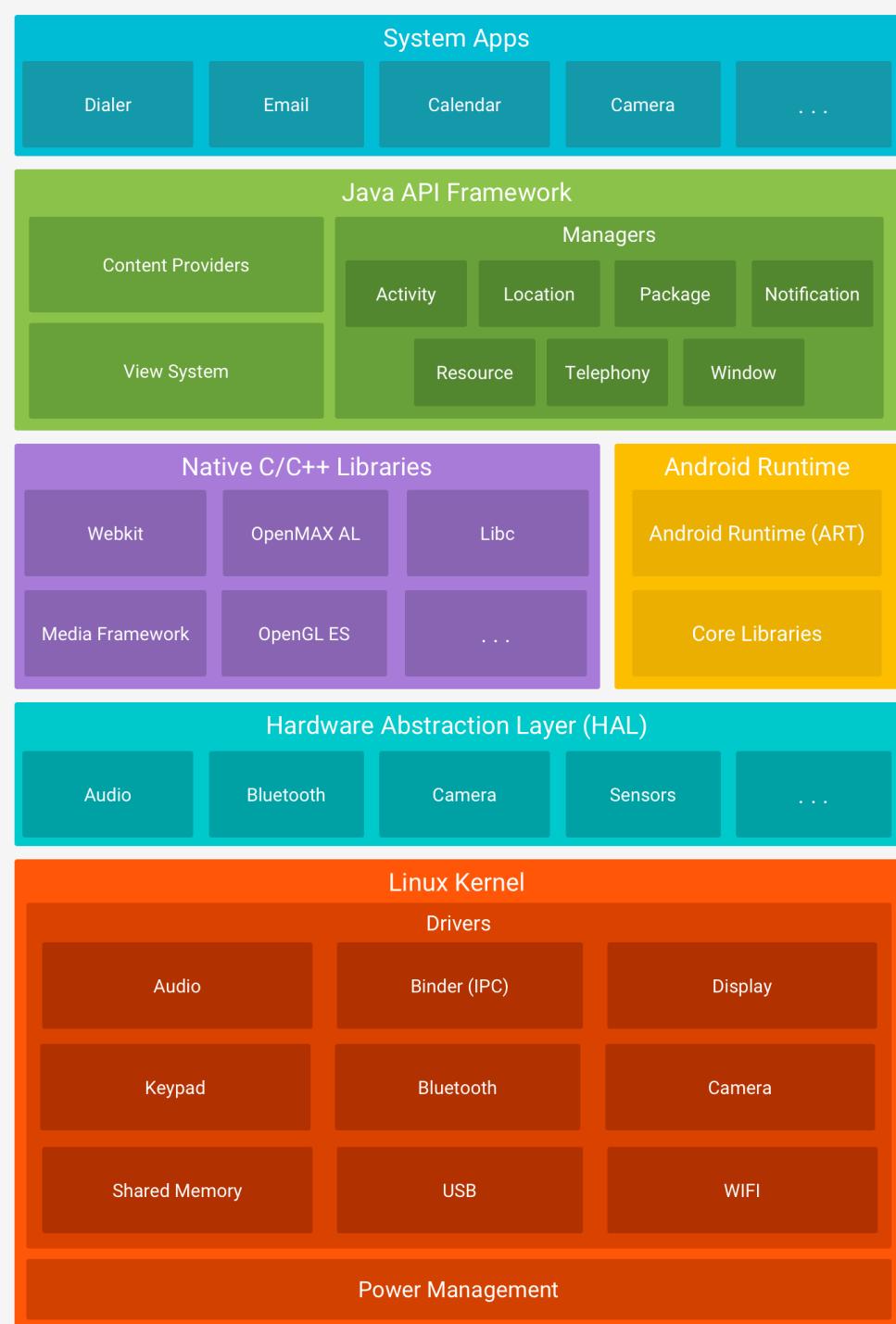
Elementos de uma arquitetura



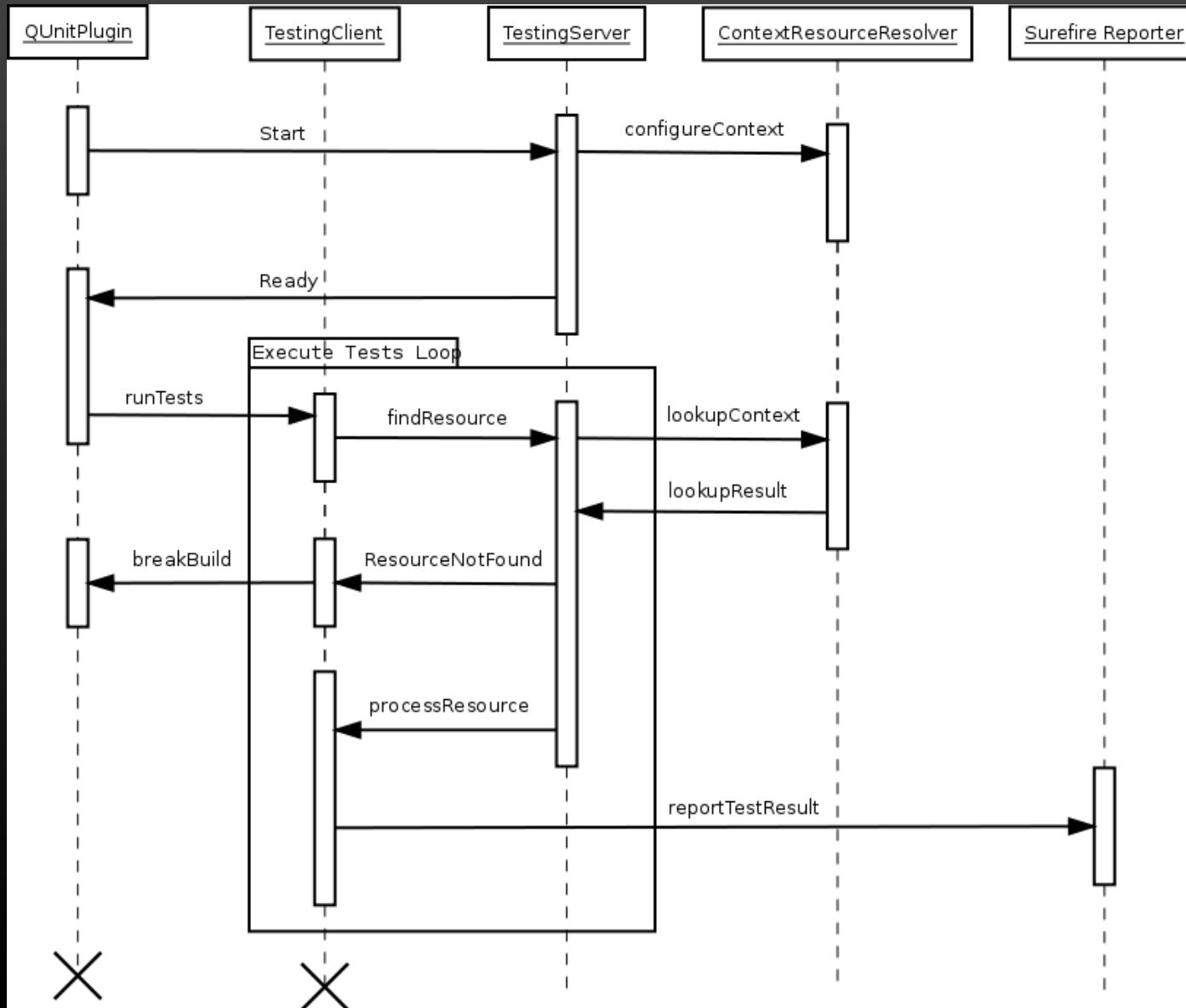
Vista estrutural (as partes constituintes)

Android platform architecture

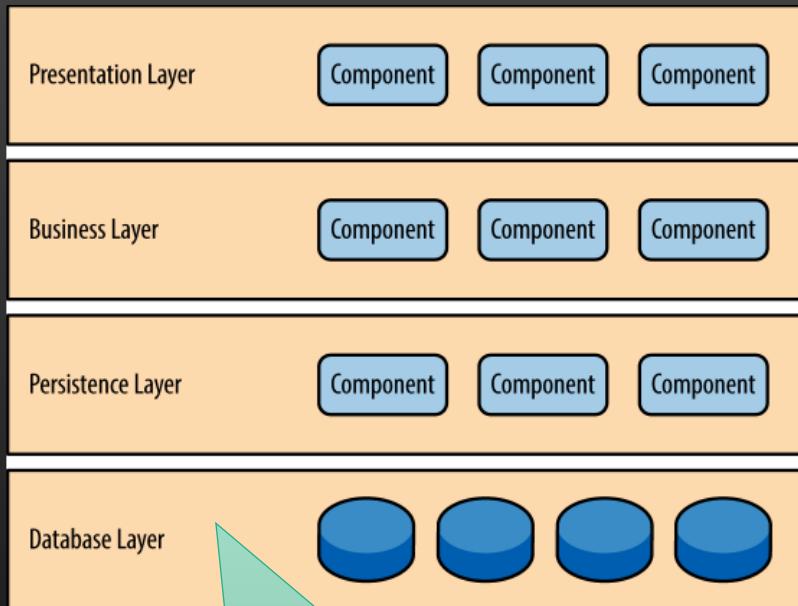
I Oliveira



Vista dinâmica (interação). Notar o nível de abstração (subsistema, não objetos)



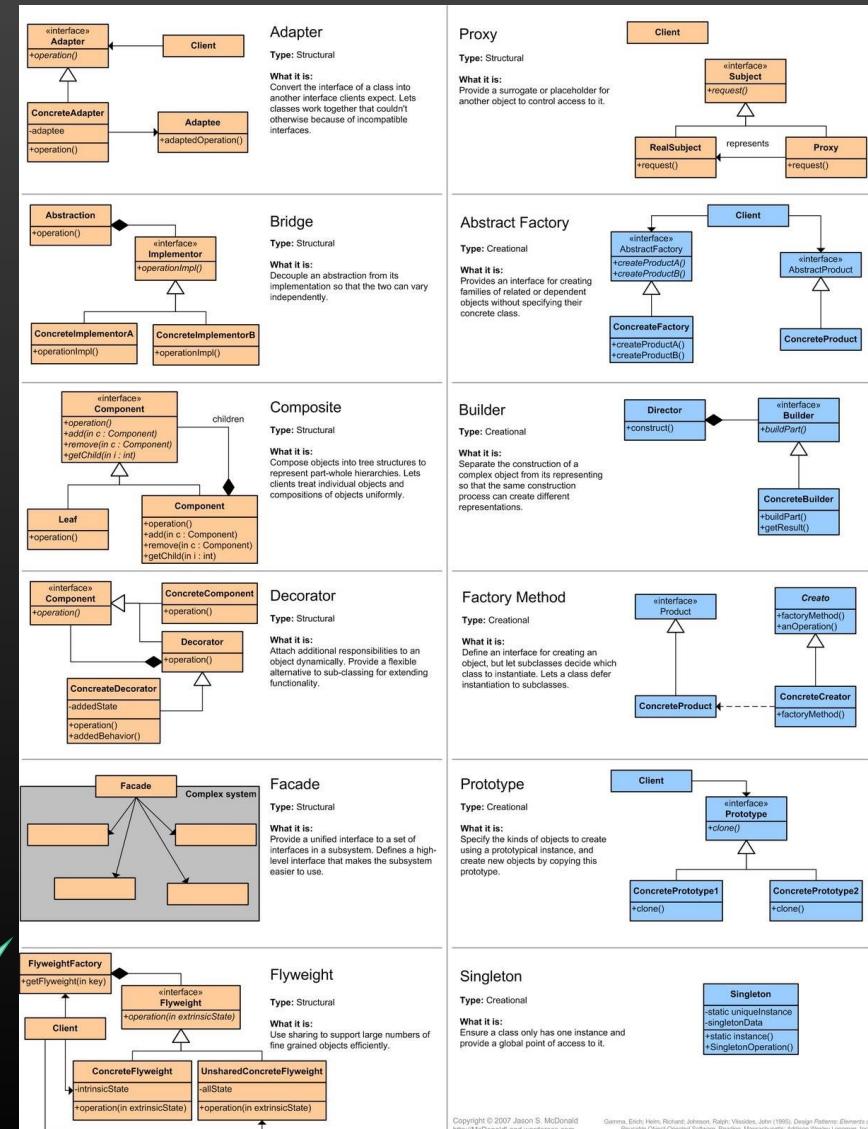
Arquitetura, componentes, classes



Como é que um sistema de software está organizado em grandes módulos. Os “grupos de funcionalidade” podem ser vistos como componentes.

Implementação interna de cada componente pode usar certos arranjos de classes (padrões frequentes).

I Oliveira



**A arquitetura é consequência dos
requisitos e estabelece compromissos**

Decisões de arquitetura

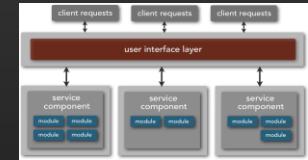
A decisão de usar uma aplicação web como camada de interação da solução



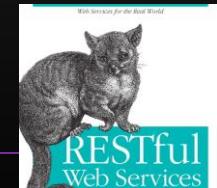
A decisão de usar Java Server Faces para o “web framework” de desenvolvimento



A decisão de distribuir componentes por vários nós para aumentar a capacidade de escalar.



A decisão de usar o modelo REST para organizar a integração com sistemas externos.



Exemplo de um sistema complexo: Feedzai



PLATFORM SOLUTIONS INDUSTRIES RESOURCES COMPANY CONTACT

LIFE OF TRANSACTION IN 3 MILLISECONDS

Feedzai uses advanced machine learning to minimize friction so you can maximize revenue

SEE FEEDZAI IN ACTION

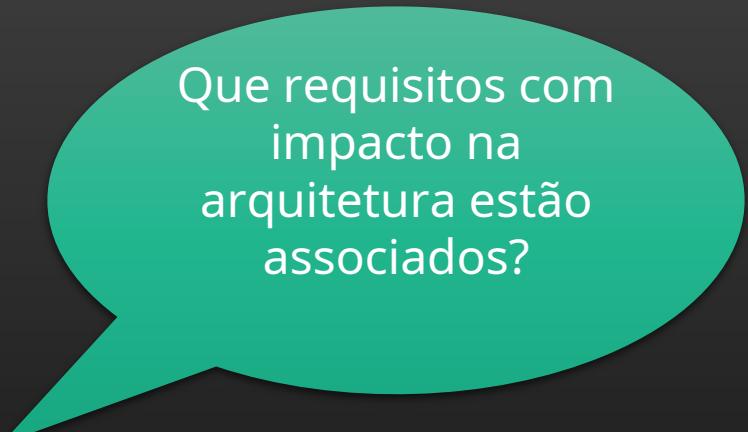
<https://feedzai.com>

- Número muito elevado de transações financeiras que devem ser analisadas em paralelo (intensidade variável)
- Processamento de eventos em larga escala em janelas temporais (solução otimizada para o processamento de feeds, i.e., séries de dados, e não interrogação de bases de dados convencionais)
- Respostas de muito baixa latência (indicação de fraude em <0,5s)
- Natureza sensível da informação: canais seguros e invioláveis.
- Há clientes que preferem a solução na cloud e outros que querem usar apenas instalações no seu datacenter

Precisamos de pensar na arquitetura de sistemas complexos

Alguns exemplos de sistemas complexos:

- a) Wikipedia
- b) Multi-player online RPG
- c) Amazon web store
- d) Twitter
- e) Netflix



Que requisitos com impacto na arquitetura estão associados?

→ Google doc

<http://107z.2o.xsl.pt/>

Exemplos

Wikipedia

Enorme quantidade de documentos de texto

Gerir milhões de documentos: armazenamento, recuperação

Pesquisar de conteúdo em documentos

Como criar e editar documentos de forma distribuída? Direitos de acesso dos utilizador?

Multi-player online RPG

Grande quantidade de jogadores interagindo entre si (por exemplo, milhares de utilizadores)

Como distribuir a carga? Como otimizar a latência?

Há utilizadores banidos?

Integrar faturaçāo (em compras de jogos), etc?

Como prevenir hackers/ batoteiros?

Exemplos

Amazon web store

Lidar com picos de utilização (por exemplo: *black friday*)

Sistema de recomendação de produtos (AI)

Quais os utilizadores que têm interesses semelhantes aos de outros utilizadores?

O que deve ser rastreado? Cliques? Compras? Comentários?

Há problemas de privacidade?

Twitter

Grande número de utilizadores, enorme quantidade de eventos, interações complexas

Interações complexas: redes, redes sociais, etc.

Sistemas de entrega fiáveis. Comprovativo de entrega?

Basear-se em protocolos Web

Exemplos

Netflix

Rede de distribuição de conteúdos em larga escala (CDN): equilíbrio de carga, réplicas,...

Utilização interativa, conteúdos multimédia (latência muito baixa)

Proteção de direitos (DRM)

Relevância do tema da arquitetura no OpenUP?

Elaboration: saber como construir, construindo uma parte

Importante mitigar riscos técnicos

Definir a arquitetura do Sistema, implementar uma parte (→ arquitetura executável)

Controlar os riscos técnicos, aprofundando os requisitos e desenvolvendo a arquitetura / plano técnico. Comprovar a arquitetura implementando uma parte.

Validar a arquitetura

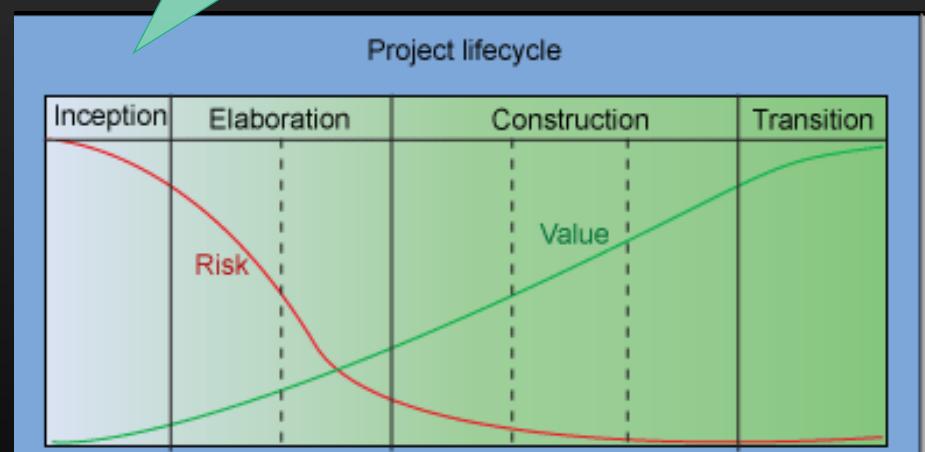
Construir “esqueletos” para os componentes principais e começar a sua integração.

Identificar dependências de sistemas e componentes externos e especificar como serão integrados.

~10% do código será implementado

Basear a arquitetura nos casos de utilização nucleares

20% dos casos de uso determinam 80% da arquitetura



Credit: Per Kroll (IBM)

OpenUP practice: evolutionary architecture

Practices > Technical Practices > Evolutionary Architecture

Practice: Evolutionary Architecture



Analyze the major technical concerns that affect the evolution, and capture the architectural decisions to ensure that those decisions are assessed and communicated.

Analisar as principais preocupações técnicas; recolher decisões de arquitetura; avaliar, documentar e comunicar decisões.

Relationships

Content References

- How to adopt the Evolutionary Architecture
- Key Concepts
 - Architectural Mechanism
 - Architectural Views and Viewpoints
 - Software Architecture
- Architecture Notebook
- Envision the Architecture
- Refine the Architecture
- Guidance
 - Guidelines
 - Abstract Away Complexity
 - Modeling the Architecture
 - Software Reuse

Inputs

- [Technical Design]

Porque é que é “evolutiva”?

No OpenUP:

- Analisar as principais questões técnicas que afetam a solução
- Documentar decisões de arquitetura para garantir que foram avaliadas e comunicadas
- Implementar e testar capacidades-chave como forma de lidar com os desafios de arquitetura
- Evoluir ao longo do tempo, a par com o trabalho de implementação “normal”

Ideia de fundo:

a escolha da arquitetura comporta riscos que devem ser controlados cedo, experimentando as capacidades-chave.

Vistas de arquitetura na UML

A arquitetura do sistema aborda diferentes perspetivas de análise

① Arquitetura lógica do software

Organização geral dos blocos de software

Independente da tecnologia de implementação

② Arquitetura de componentes do software

Peças construídas com uma tecnologia concreta

Construção “modular”

- E.g.: existem pré-feitos?

③ Arquitetura de instalação

Visão dos equipamentos e configuração de produção
(conectividade, distribuição,...)

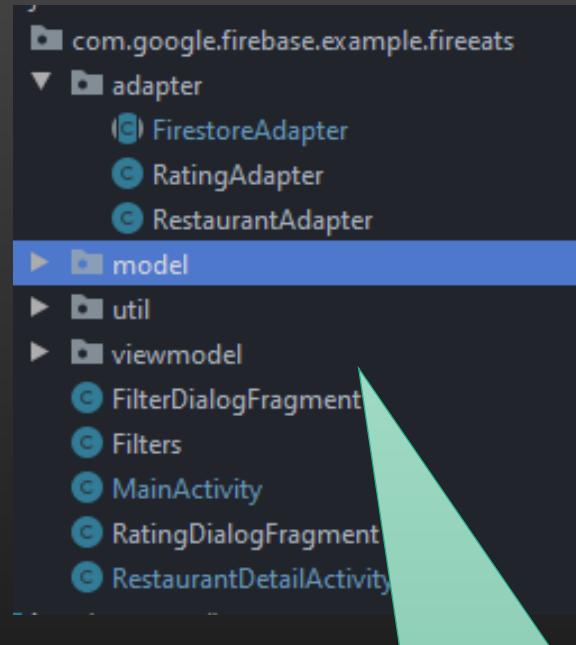
Arquitetura lógica

Organização geral da solução em blocos (*packages*)

Os *packages* podem representar agrupamentos muito diferentes.

E.g.:

- *Packages* num programa em Java
- *Packages* num modelo UML
- Subsistemas/divisões do sistema sob especificação

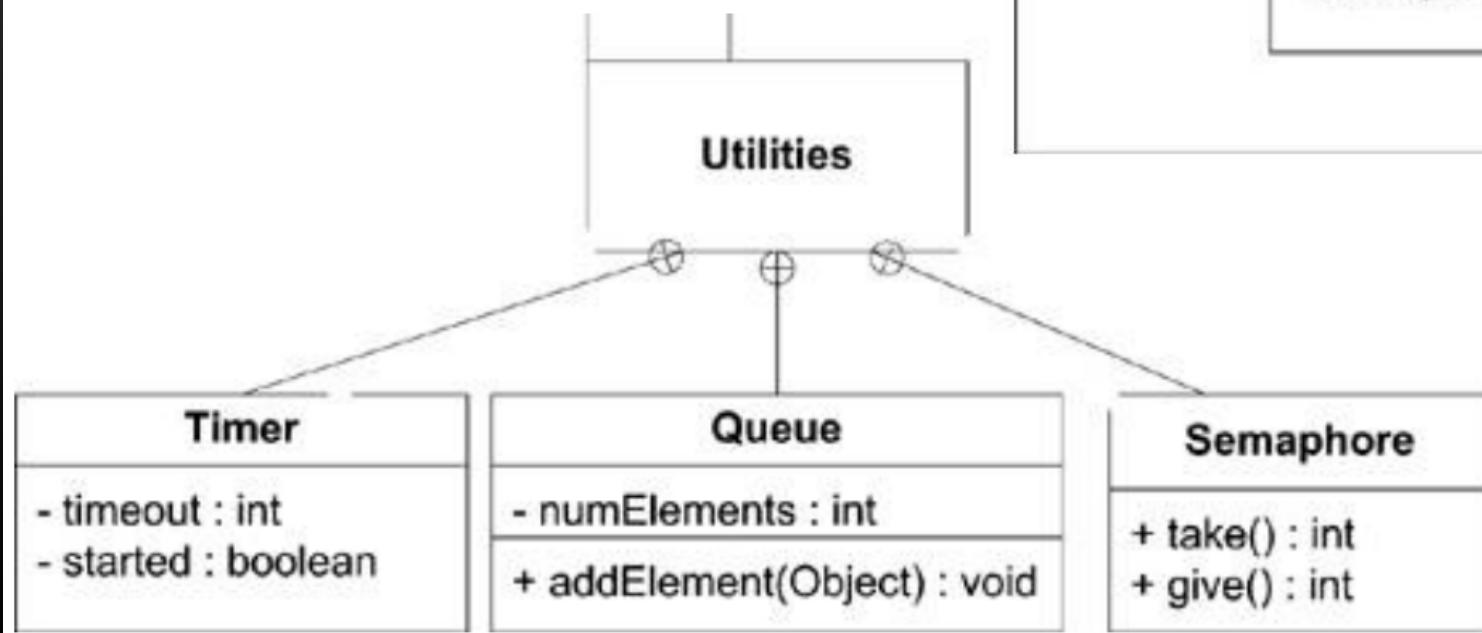
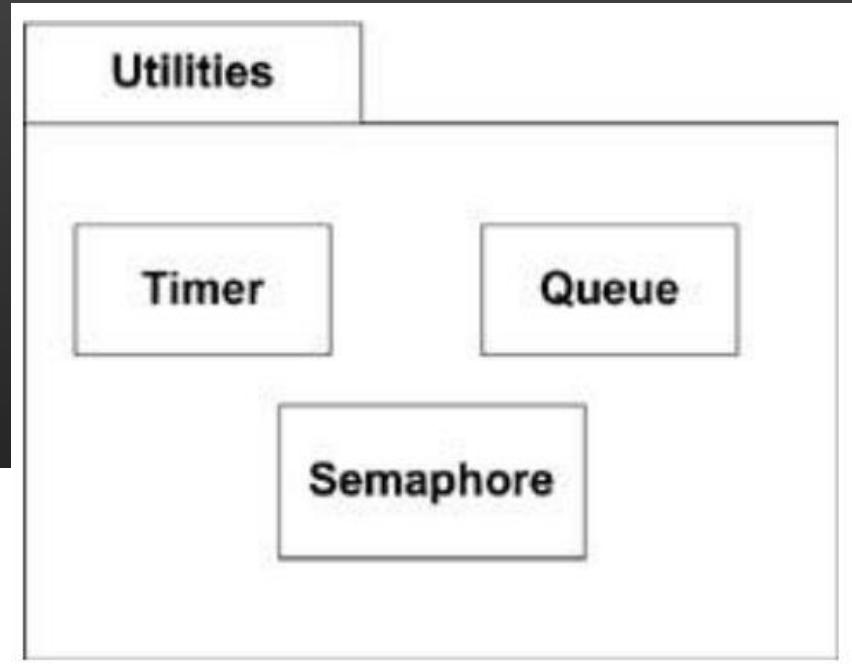


A ideia de package é usada em Java para formar grupos de entidades relacionadas.
Os *packages* podem ser hierárquicos.

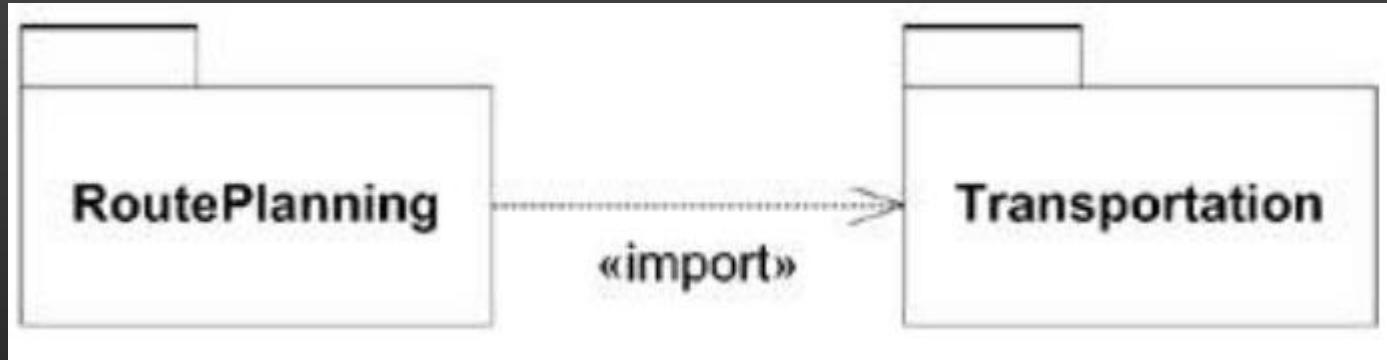
Comunicar a arquitetura lógica com pacotes (*packages*)

Classes contidas num pacote

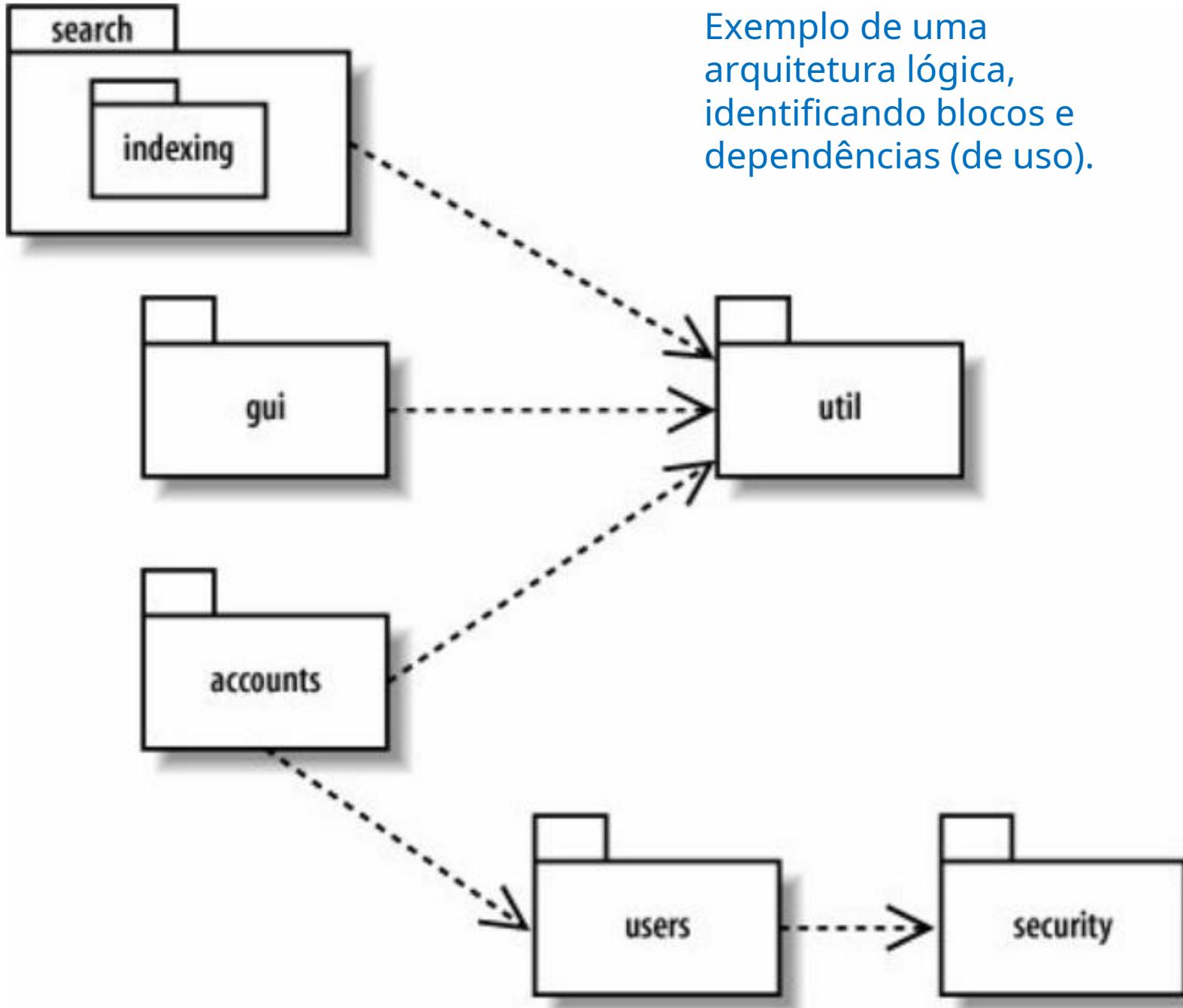
- duas representações alternativas



Associações entre pacotes



Dependência do tipo “import”



A arquitetura do sistema aborda diferentes perspetivas de análise

① Arquitetura lógica do software

Organização geral dos blocos de software

Independente da tecnologia de implementação

② Arquitetura de componentes do software

Peças construídas com uma tecnologia concreta

Construção “modular”

- E.g.: existem pré-feitos?

③ Arquitetura de instalação

Visão dos equipamentos e configuração de produção
(conectividade, distribuição,...)

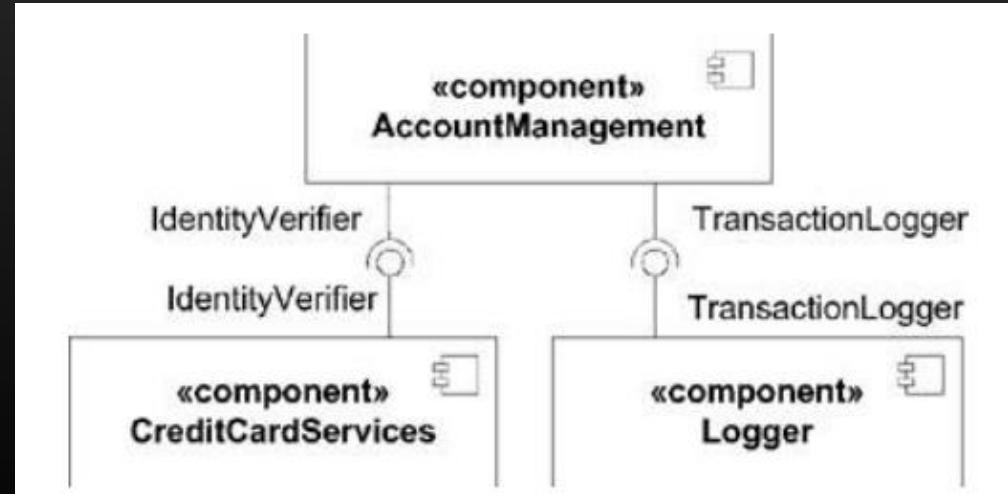
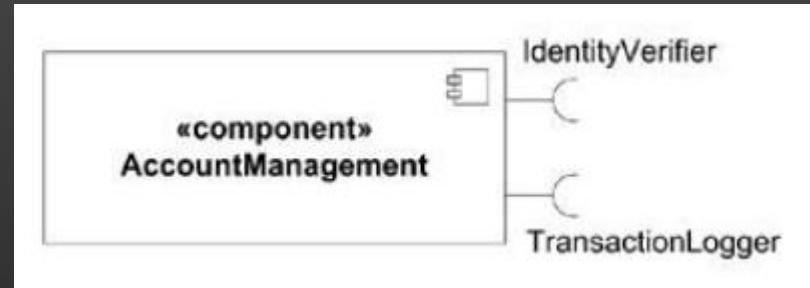
Componente

É normal dividir sistemas complexos em subsistemas mais geríveis

O componente é uma peça substituível, reusável de um sistema maior, cujos detalhes de implementação são abstraídos

A funcionalidade de um componente é descrita por um conjunto de interfaces fornecidos

Para além de implementar, o componente pode requerer funcionalidades de outros





A *component* is a self-contained, encapsulated piece of software that can be plugged into a system to provide a specific set of required functionalities. Today, there are many components available for purchase. A component has a well-defined *API* (application program interface). An API is essentially a set of method interfaces to the objects contained in the component. The internal workings of the component are hidden behind the API. Com-

Dennis, Alan, Barbara Wixom, David Tegarden.
Systems Analysis and Design: An Object Oriented Approach with UML, 5th Edition. Wiley.

Arquitetura de componentes do software

Ao contrário do *package*, o componente é uma peça tangível da solução (e.g.: ficheiro, arquivo)

Os componentes são implementados com tecnologia concreta

Propriedades desejáveis:

Encapsulamento (da estrutura interna)

Reutilizável (em vários projetos)

Substituível

Candidatos naturais:

Aspetos recorrentes em vários projetos

Módulos que se podem obter pré-feitos ou disponibilizar

Módulos definidos para ir de encontro às regras dos ambientes de execução (e.g.: módulos para *application servers*)



<https://mvnrepository.com>

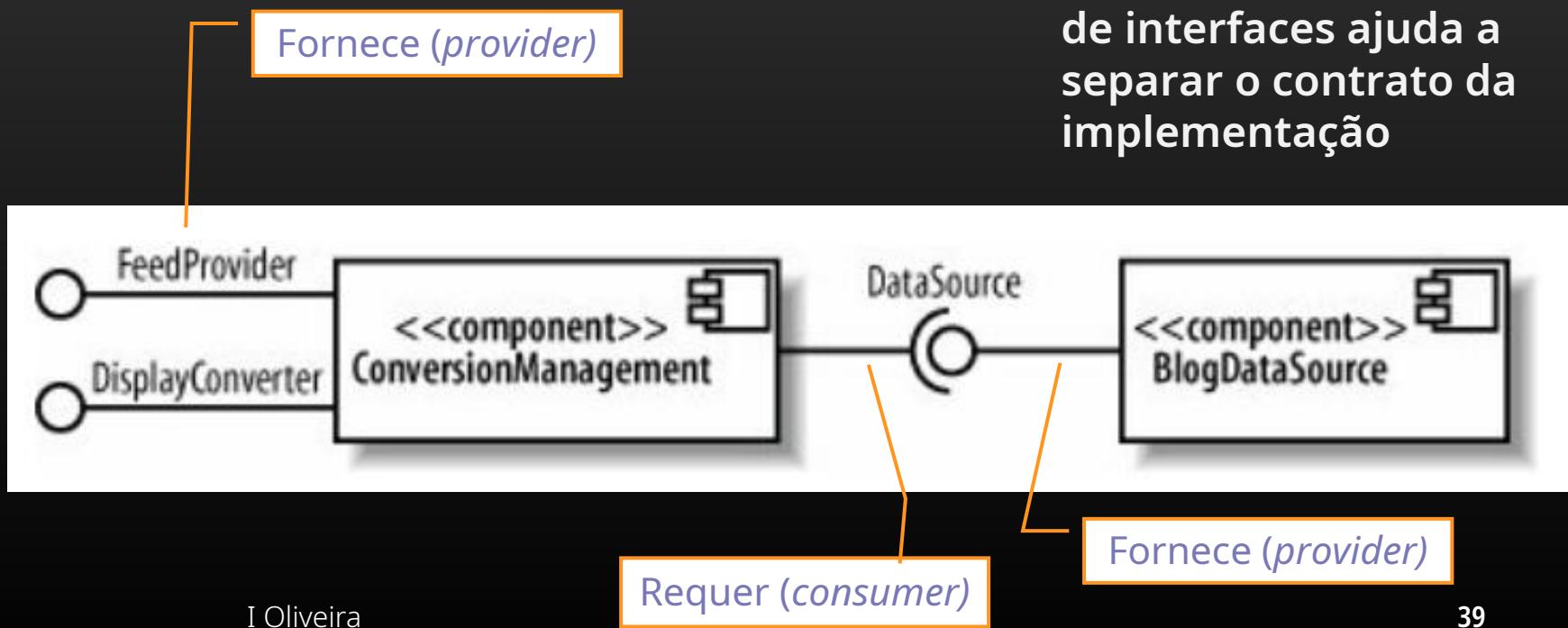
Uma “loja” de componentes, encapsulados, reutilizáveis e substituíveis.

Figure

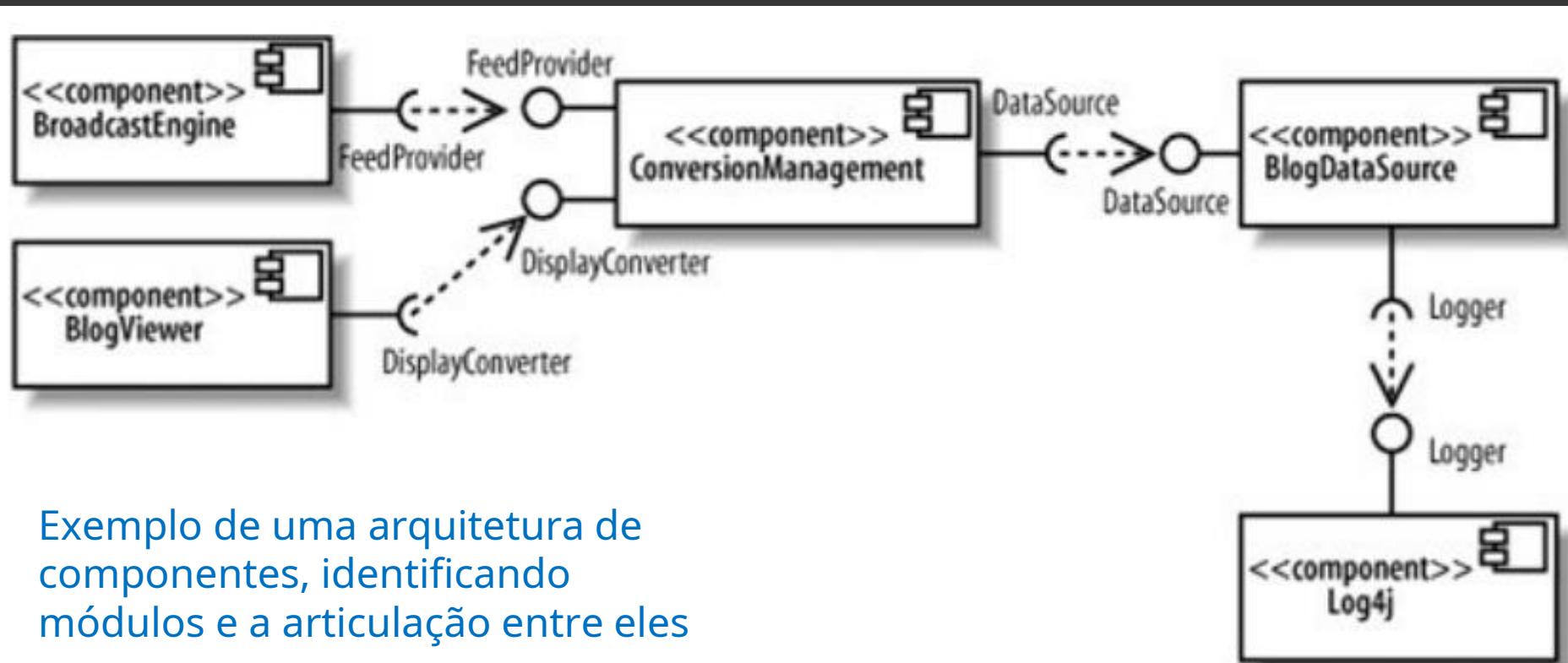
Solução modular com componentes

Com os componentes, pretende-se arquiteturas com baixo “coupling”

A exposição da funcionalidade através de interfaces ajuda a separar o contrato da implementação

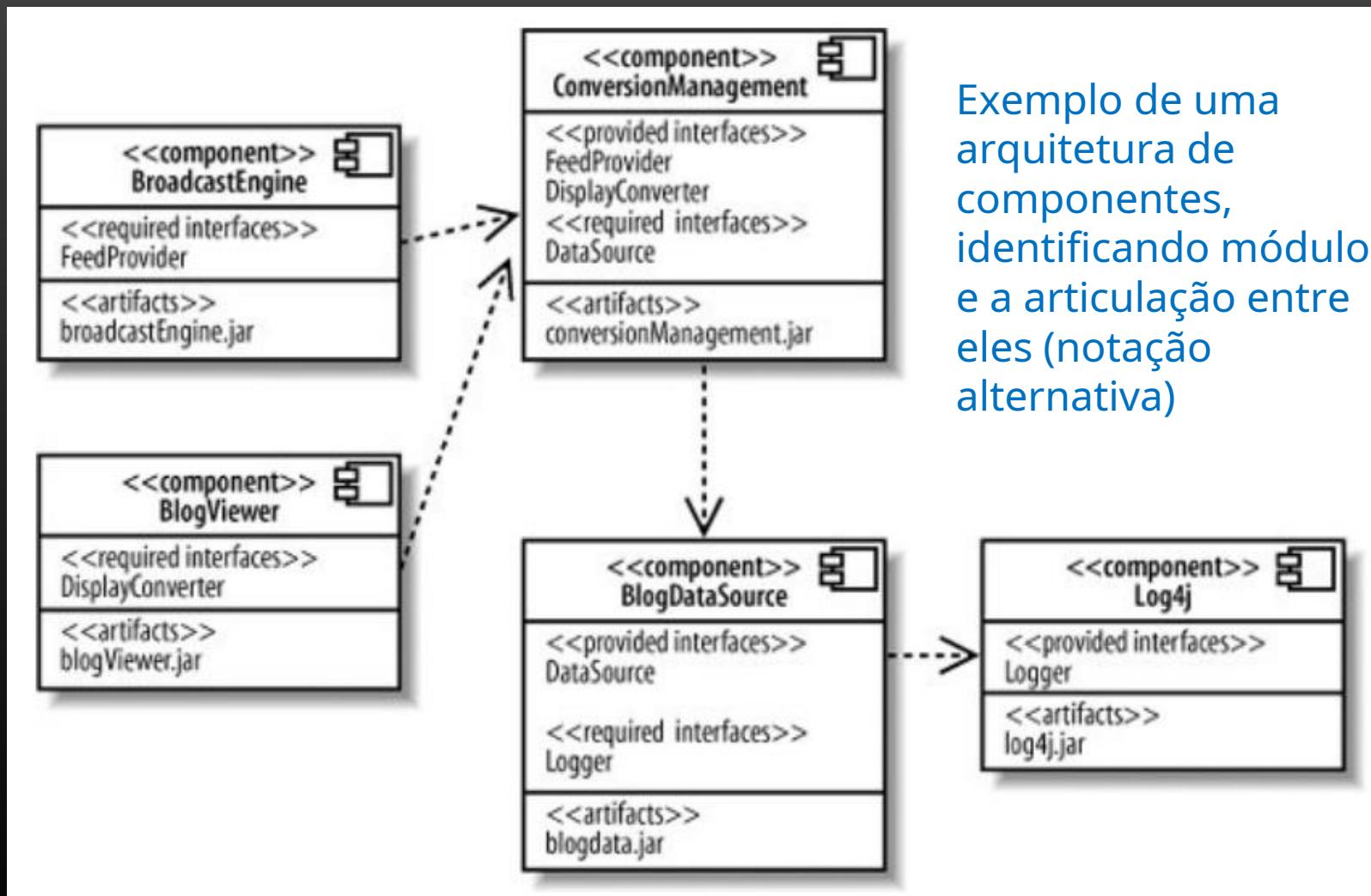


O mesmo modelo, notação ligeiramente diferente 1

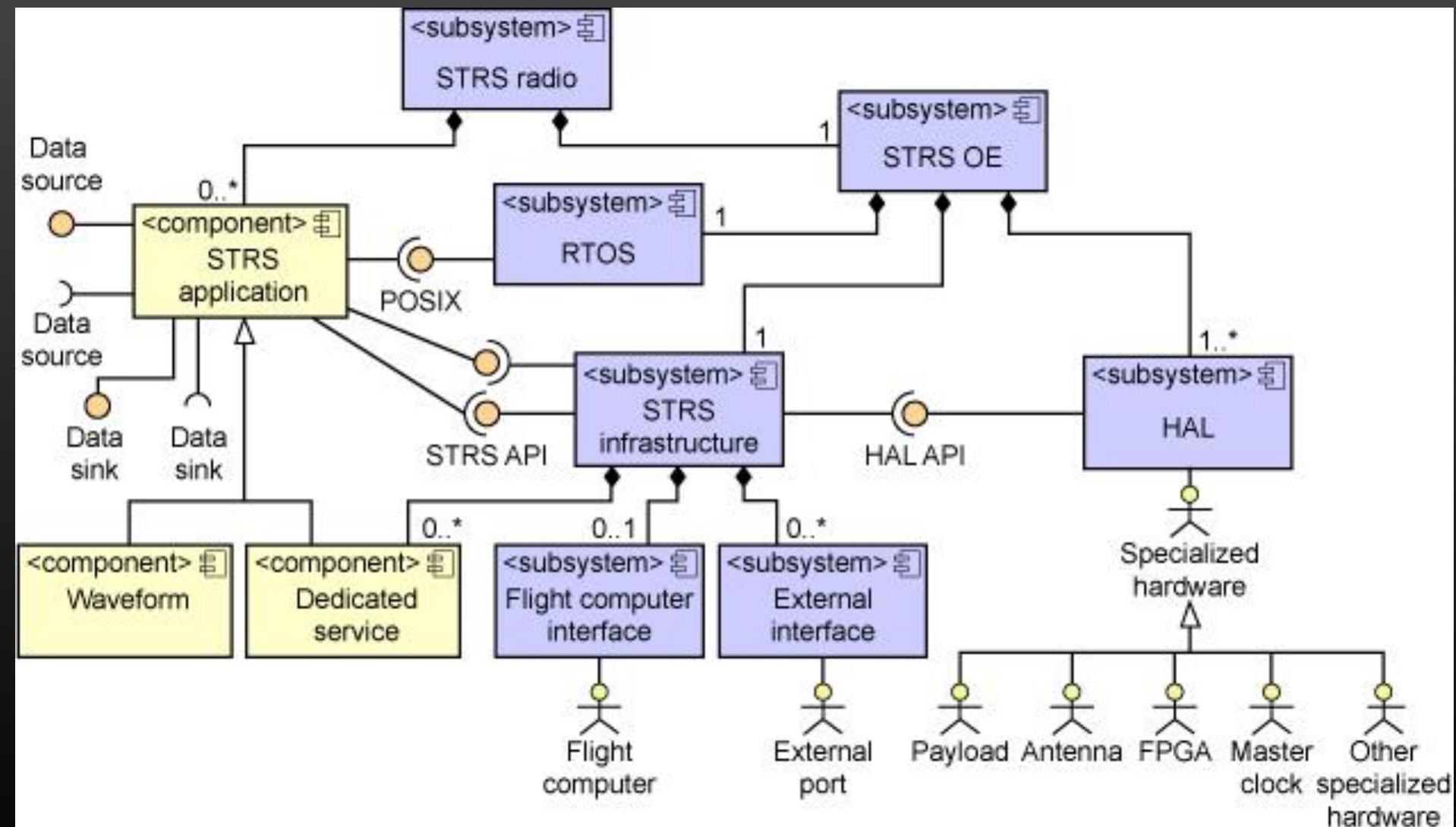


Exemplo de uma arquitetura de componentes, identificando módulos e a articulação entre eles

O mesmo modelo, notação ligeiramente diferente 2

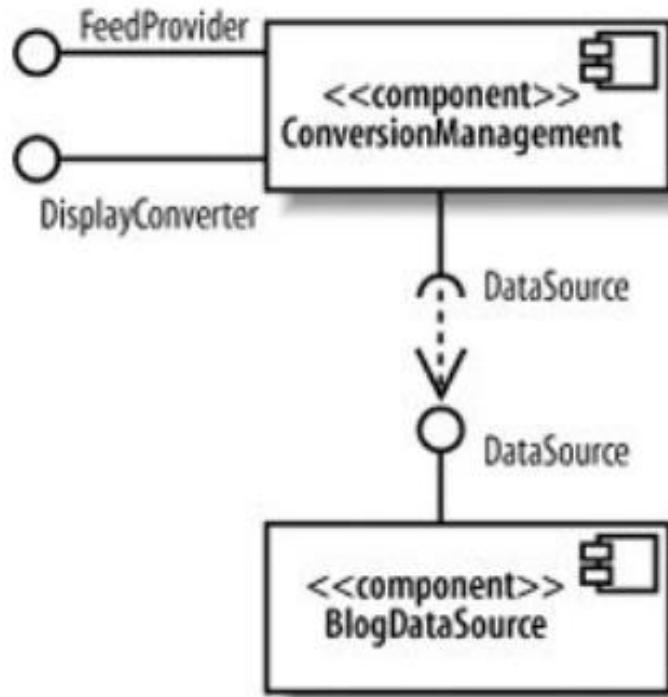


An example from NASA: Radio System

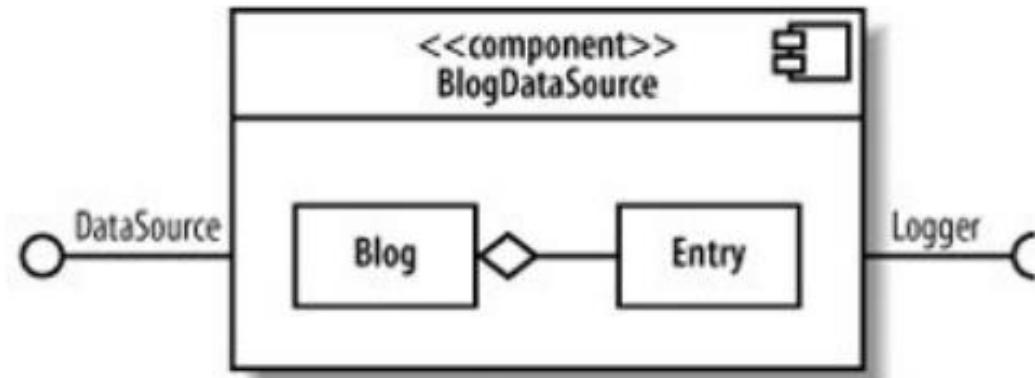


Notação “caixa fechada” / “caixa aberta”

Example Black-Box Component View



Example White-Box Component View



A arquitetura do sistema aborda diferentes perspetivas de análise

① Arquitetura lógica do software

Organização geral dos blocos de software

Independente da tecnologia de implementação

② Arquitetura de componentes do software

Peças construídas com uma tecnologia concreta

Construção “modular”

- E.g.: existem pré-feitos?

③ Arquitetura de instalação

Visão dos equipamentos e configuração de produção
(conectividade, distribuição,...)

Diagramas de instalação da UML

Nós (*node*)

Um equipamento de hardware

Ambiente de execução

Um ambiente externo à solução que proporciona o contexto necessário à sua execução

E.g.: SO, servidor web, servidor aplicacional

Artefactos (*artifact*)

Ficheiros concretos que são executados ou utilizados pela solução

E.g.: executáveis, bibliotecas, configurações, scripts

A node:

- Is a computational resource, e.g., a client computer, server, separate network, or individual network device.
- Is labeled by its name.
- May contain a stereotype to specifically label the type of node being represented, e.g., device, client workstation, application server, mobile device, etc.

<<stereotype>>
Node Name

An artifact:

- Is a specification of a piece of software or database, e.g., a database or a table or view of a database, a software component or layer.
- Is labeled by its name.
- May contain a stereotype to specifically label the type of artifact, e.g., source file, database table, executable file, etc.

<<stereotype>>
Artifact Name

A node with a deployed artifact:

- Portrays an artifact being placed on a physical node.

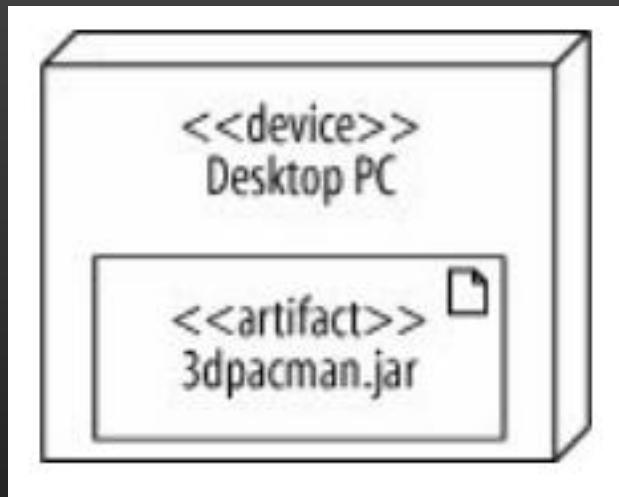
<<stereotype>>
Node Name
<<stereotype>>
Artifact Name

A communication path:

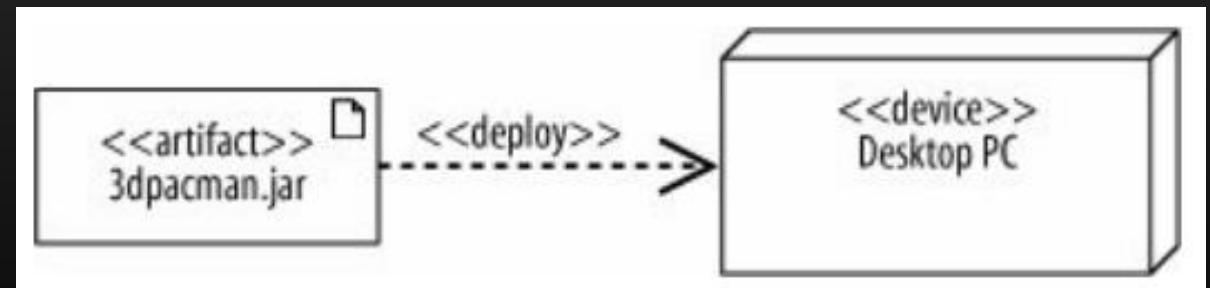
- Represents an association between two nodes.
- Allows nodes to exchange messages.
- May contain a stereotype to specifically label the type of communication path being represented, (e.g., LAN, Internet, serial, parallel).

<<stereotype>>

Os artefactos são executados em nós



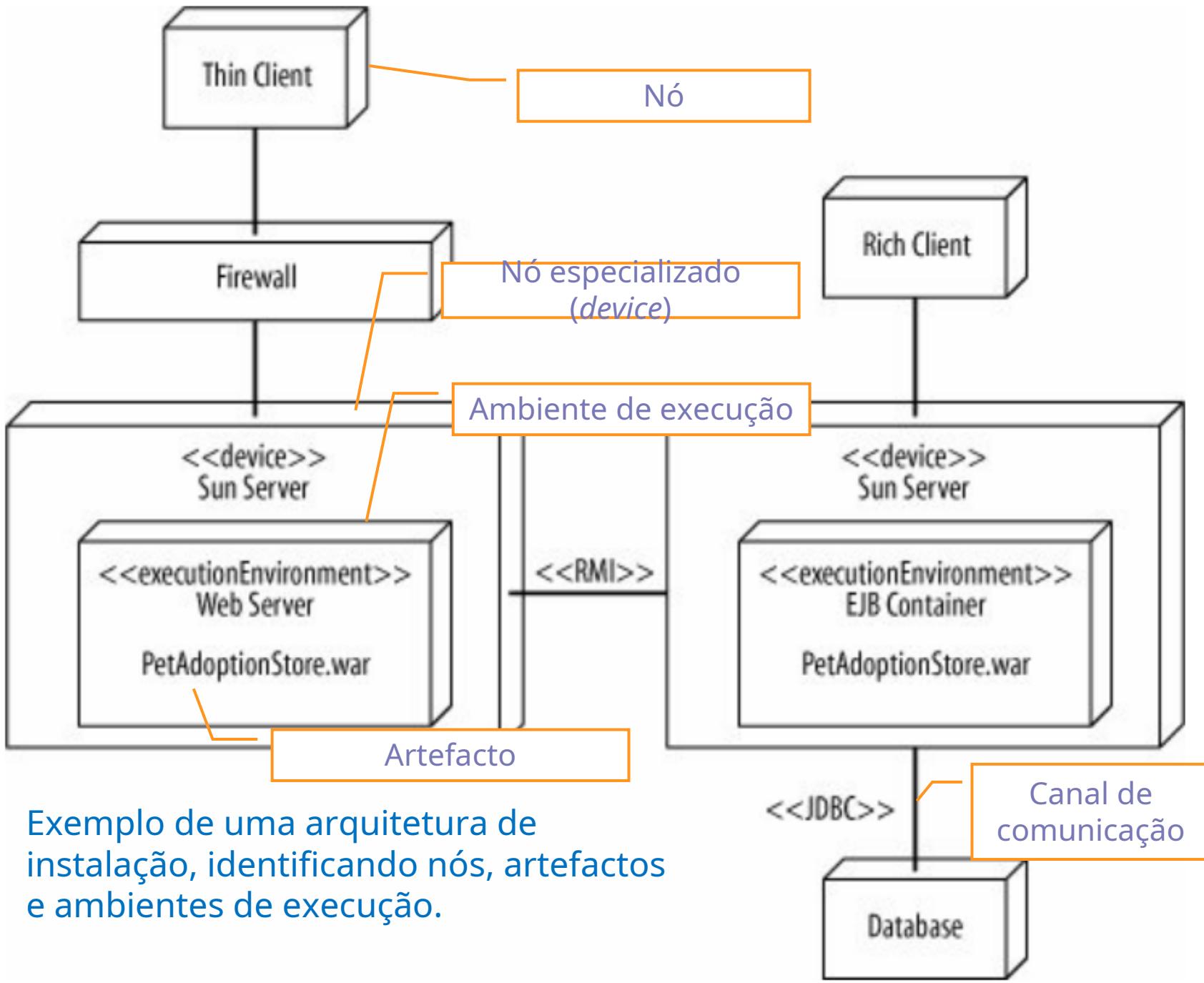
Notações alternativas.



Rastreabilidade até aos componentes

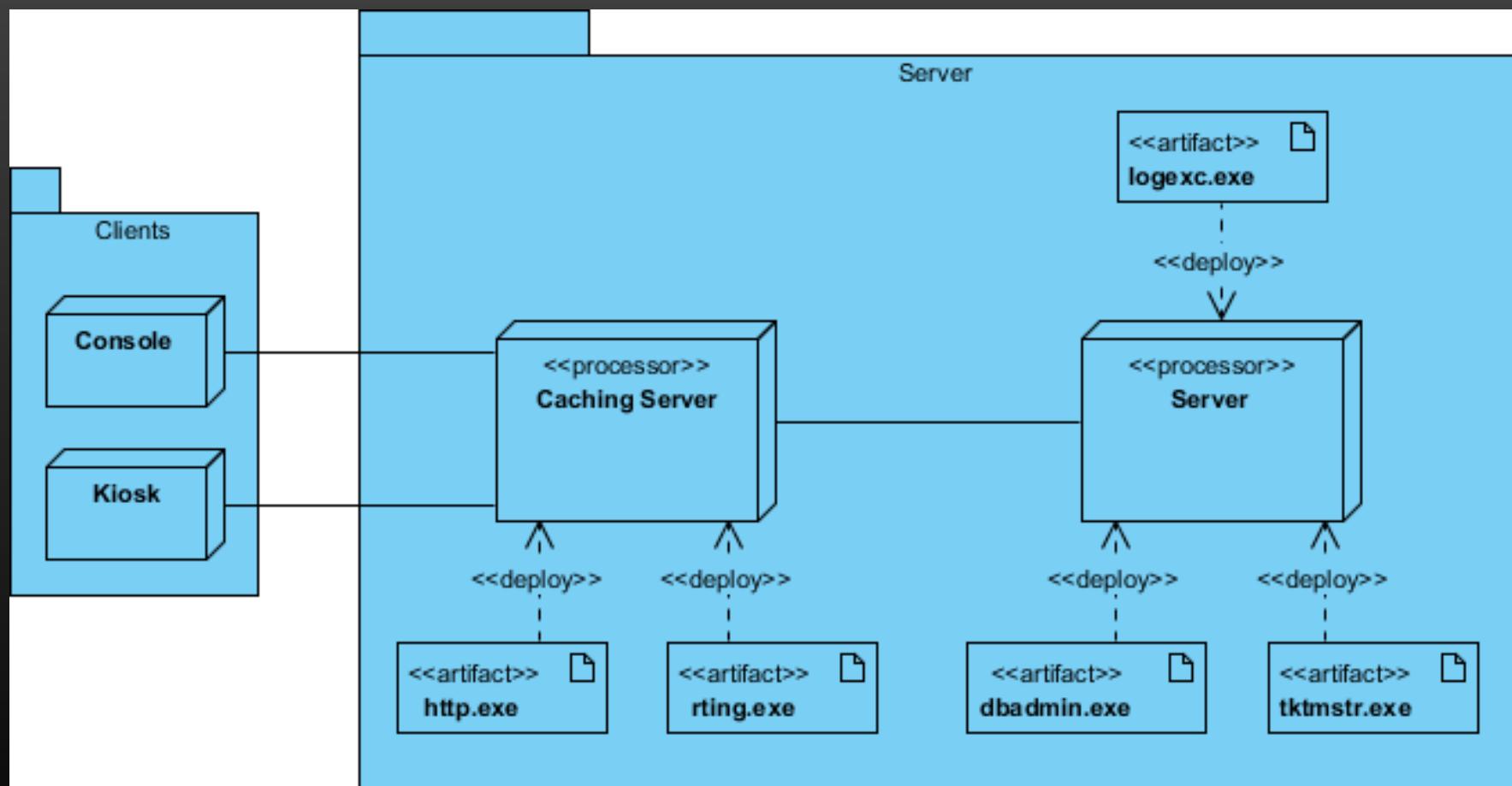


A relação “manifest” permite associar componentes a artefactos.



Exemplo de uma arquitetura de instalação, identificando nós, artefactos e ambientes de execução.

Exemplos (diagrama de instalação)



References

Core readings	Suggested readings
<ul style="list-style-type: none">• [Dennis15] – Chap. 7 & 11	

47006- ANÁLISE E MODELAÇÃO DE SISTEMAS

Quality assurance in the SDLC

Ilídio Oliveira

v2021/01/06, TP18

Learning objectives

Identify validation and verification activities in the SDLC

Describe the layers of the test pyramid

Describe the object of unit, integration, system and acceptance test

Explain the lifecycle of TDD

Explain how the QA activities are inserted in the development process in a classical approach and in agile methods

Relate the story acceptance criteria with agile testing

Explain the concept of executable specifications (and the relation with BDD).

Algumas ideias do desenvolvimento ágil

QUICK LOOK

What is it? Agile software engineering **combines a philosophy and a set of development guidelines.**

The philosophy encourages customer satisfaction and early incremental delivery of software; small, highly motivated project teams; informal methods; minimal software engineering work products; and overall development simplicity. The development guidelines stress **delivery over analysis and design** (although these activities are not discouraged), and active and continuous communication between developers and customers.

Who does it? Software engineers and other project stakeholders (managers, customers, end users) work together on an agile team—a team that is self-organizing and in control of its own destiny. An agile team fosters communication and collaboration among all who serve on it.

Why is it important? The modern business environment that spawns computer-based systems and software products is fast-paced and ever-changing. Agile software engineering represents a reasonable alternative to

conventional software engineering for certain classes of software and certain types of software projects. It has been demonstrated to deliver successful systems quickly.

What are the steps? Agile development might best be termed “software engineering lite.” The basic framework activities—communication, planning, modeling, construction, and deployment—remain. But they morph into a minimal task set that pushes the project team toward construction and delivery (some would argue that this is done at the expense of problem analysis and solution design).

What is the work product? Both the customer and the software engineer have the same view—the only really important work product is an operational “software increment” that is delivered to the customer on the appropriate commitment date.

How do I ensure that I’ve done it right? If the agile team agrees that the process works, and the team produces deliverable software increments that satisfy the customer, you’ve done it right.

O dinamismo do mercado obriga a igual dinamismo das TIC/desenvolvimento. Especialmente quando os produtos do desenvolvimento passam a assumir um papel fundamental na criação das vantagens competitivas.

A transformação digital (competitiva) obriga a uma eng.a de software competitiva.

Velocidade “furiosa”?

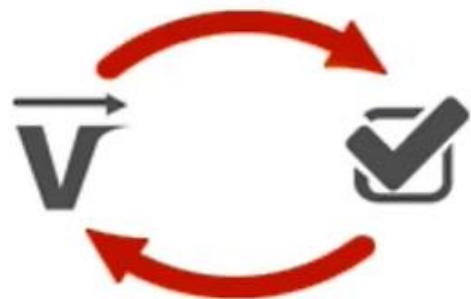


Greater speed may generate more risk and less quality...



... but

Velocity = Direction + Speed



quick feedback
improves direction
which improves quality which improves
speed
which improves feedback

Para avançar depressa e com segurança, é preciso preparar a “máquina”: mexer no próprio processo de engenharia de sw.

Crédito: Rui Gonçalves, Winning Consulting

É indispensável considerar as práticas que podem levar ou medir a qualidade do produto

GARANTIA DE QUALIDADE DE SOFTWARE

conjunto de atividades (práticas) para controlar e monitorizar o processo de desenvolvimento de software para atingir os objetivos do projeto com um certo nível de confiança em termos de qualidade

CONTROLO DE QUALIDADE DE SOFTWARE

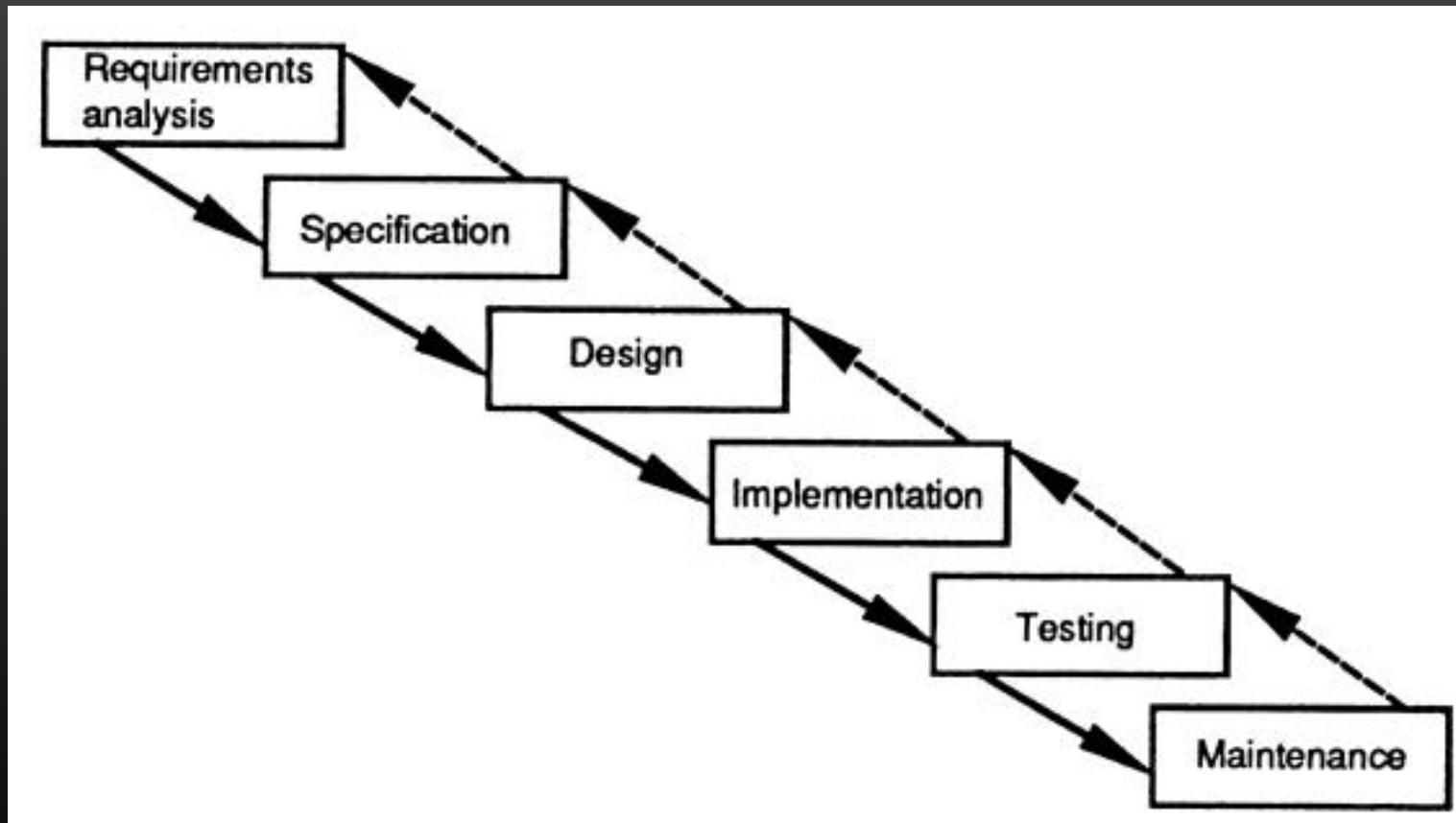
Avalia se os produtos de software estão dentro dos padrões de qualidade definidos recorrendo a inspeções e diferentes tipos de testes

SQA != SQC

O SQC visa detetar e corrigir defeitos. A SQA tem como objetivo impedi-los.

**A garantia de qualidade é parte integrante
de um processo de engenharia**

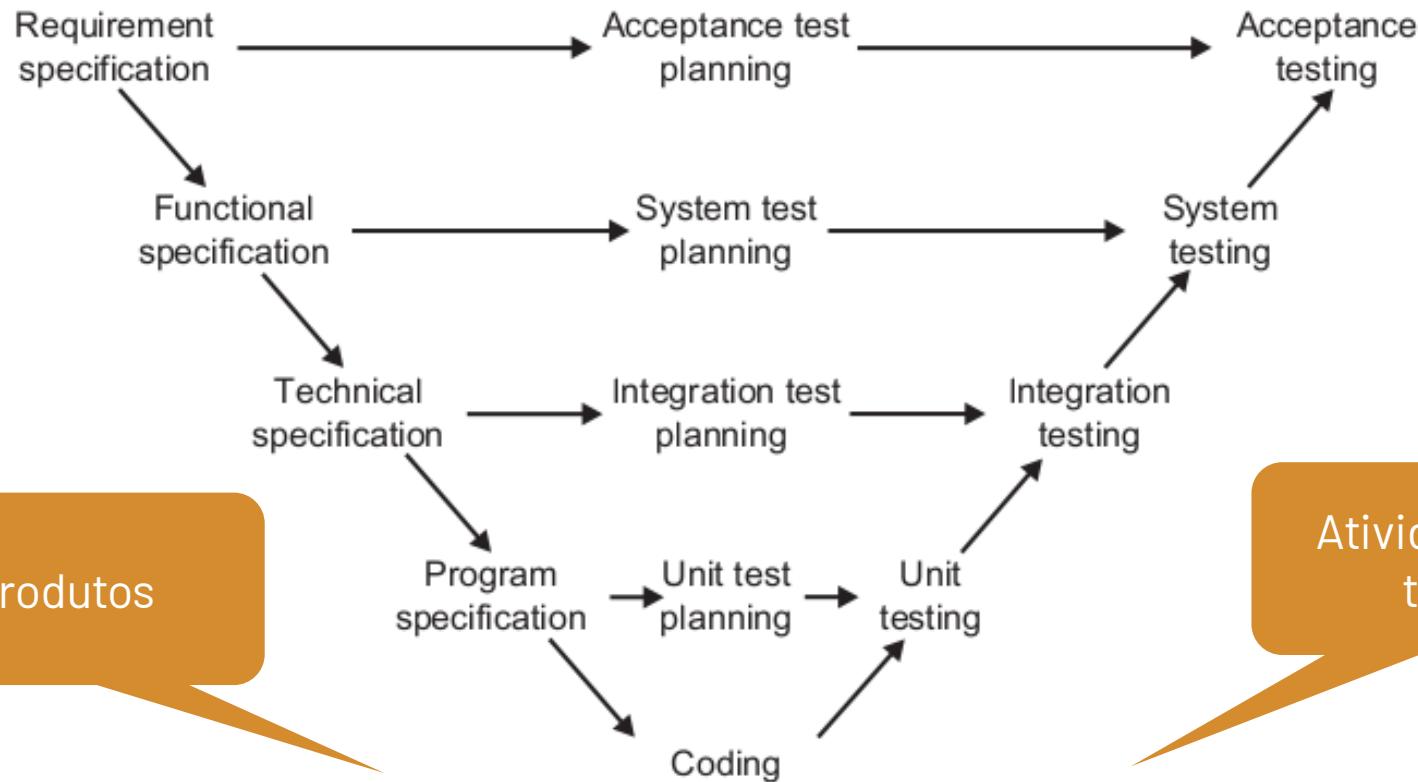
Abordagem de engenharia "clássica": Modelo waterfall



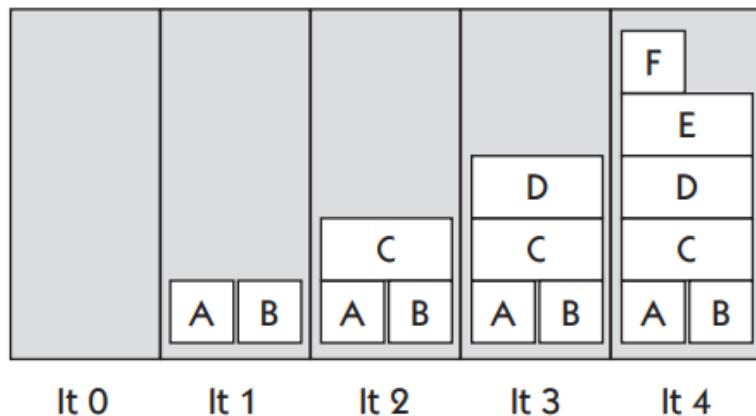
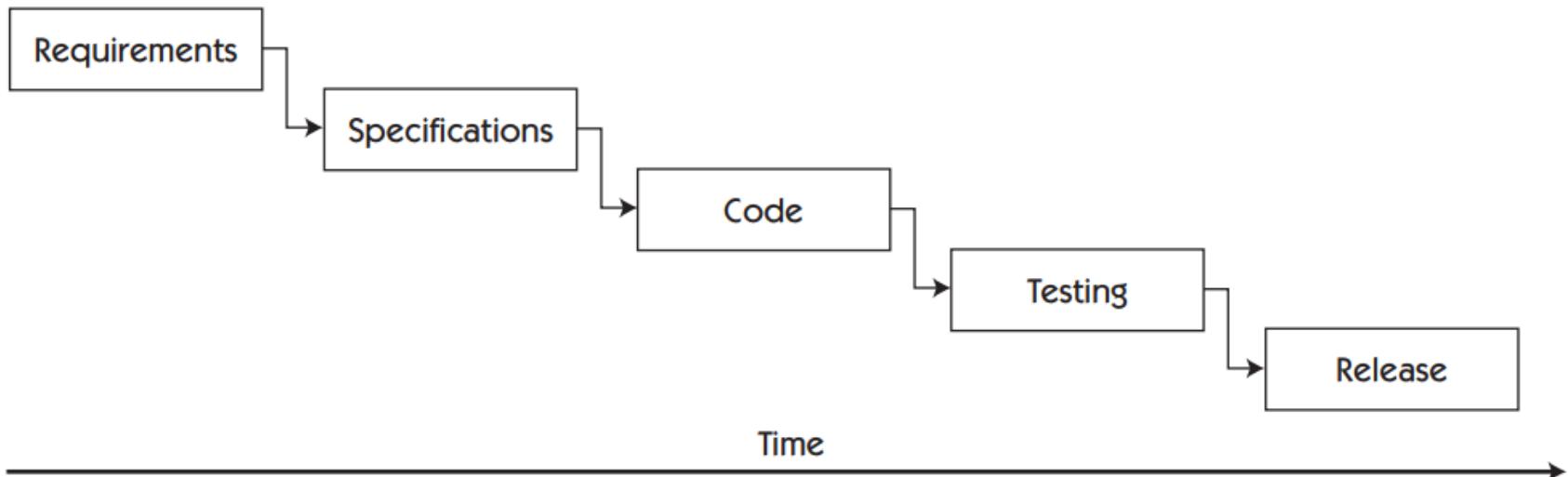
W. Royce, “Managing the Development of Large Software Systems,” *Proc. Westcon*, IEEE CS Press, 1970, pp. 328-339.

Ciclo de vida dos testes e o ciclo de vida do desenvolvimento do sw na abordagem sequencial

Figure 2.2 V-model for software development



Phased or gated—for example, Waterfall

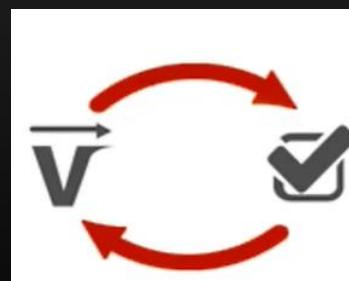
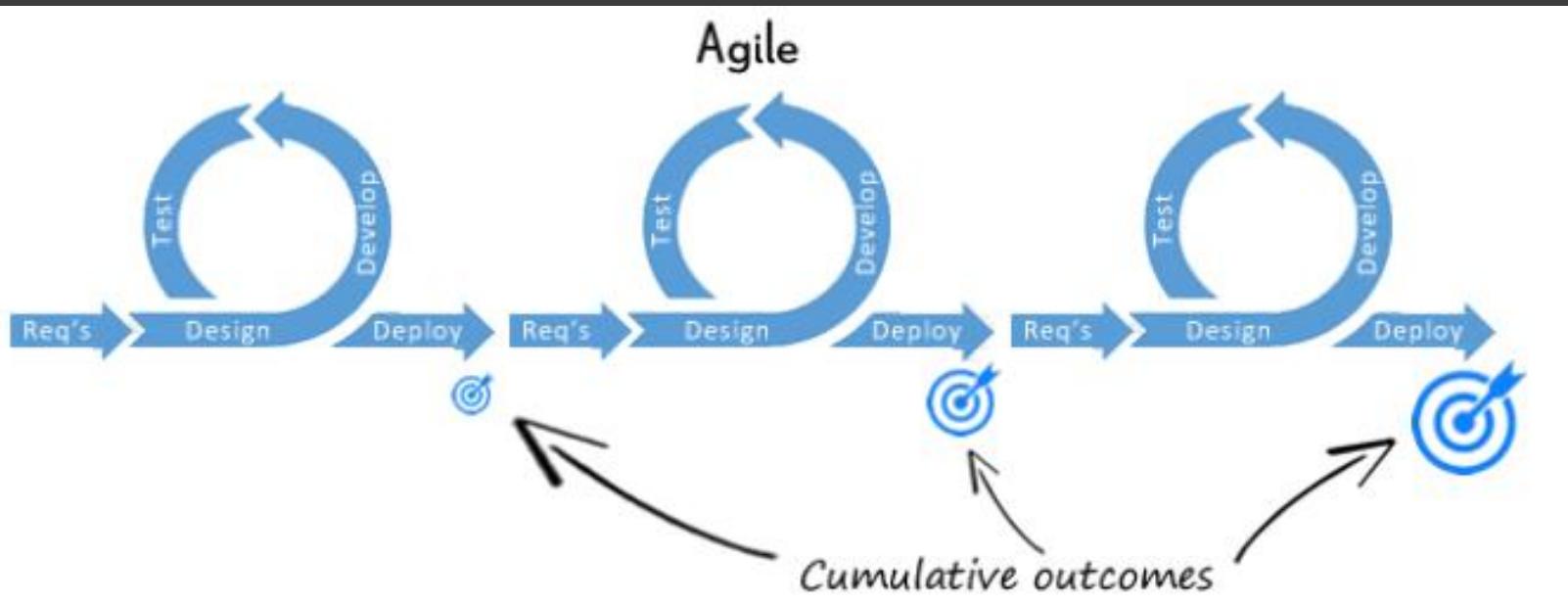


Agile:
Iterative & incremental

- Each story is expanded, coded, and tested
- Possible release after each iteration

Figure 1-4 Traditional testing vs. agile testing

Na abordagem ágil, a garantia de qualidade tem de ser aplicada em cada ciclo



quick feedback
improves direction
which improves quality which improves speed
which improves feedback

O papel dos testes de software

Verification vs Validation

VERIFICATION: ARE WE DOING THE SYSTEM IN THE RIGHT WAY?

Check work products against their specifications

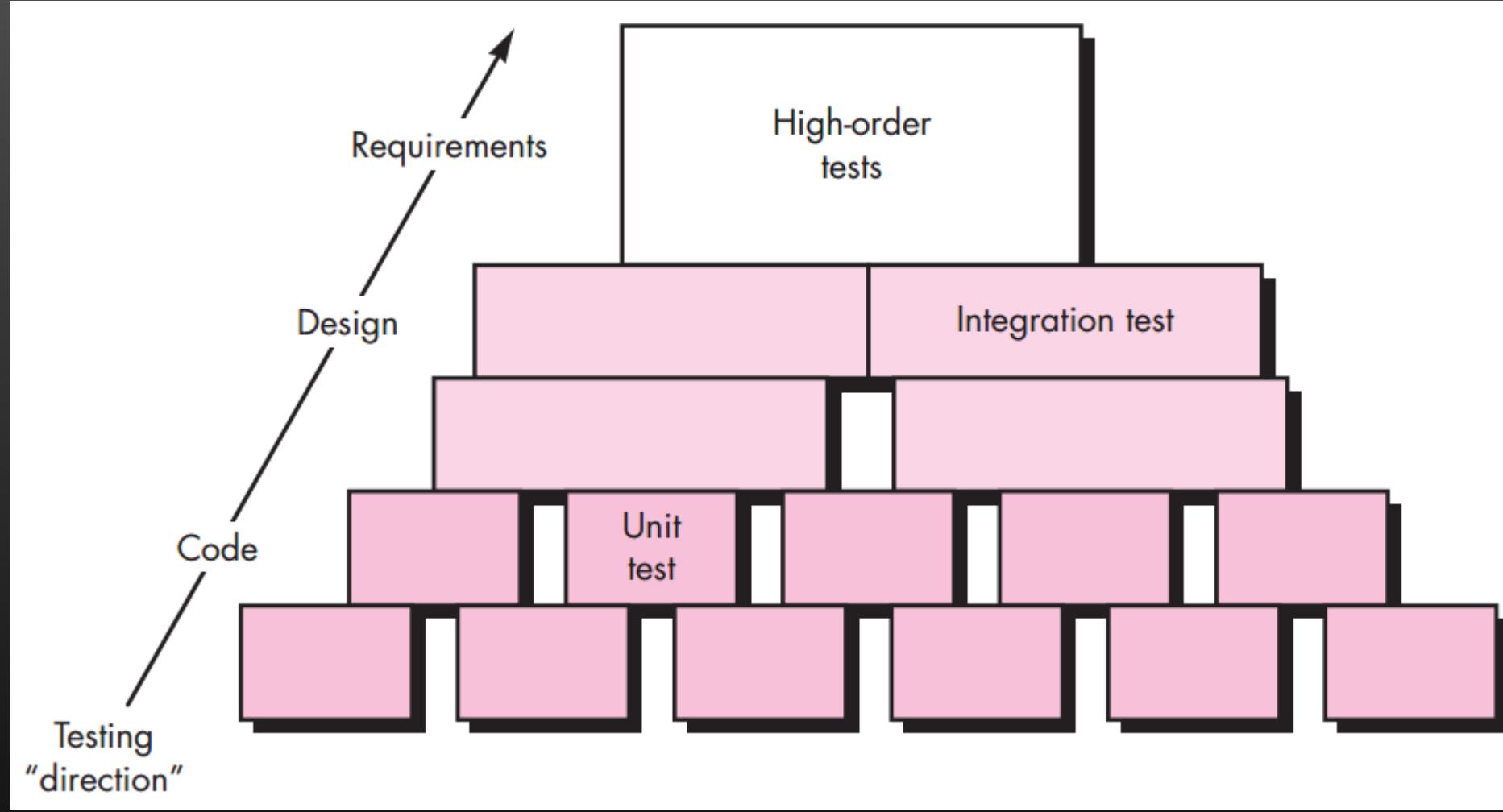
Check modules consistency

Check against industry best practices

...

VALIDATION: ARE WE DOING THE RIGHT SYSTEM?

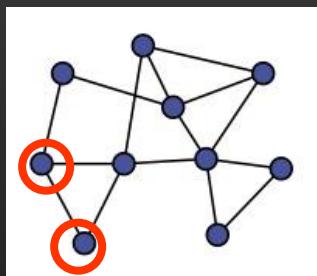
Check work-products against the user needs and expectations



Testing begins at component level and works outwards.

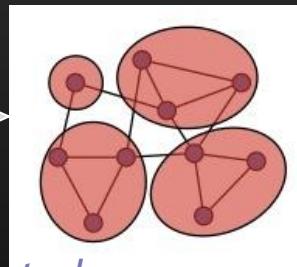
Different testing techniques are appropriate at different moments/software

Unit testing

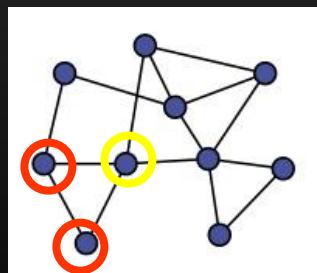


Each module does what it is supposed to do?

integration testing

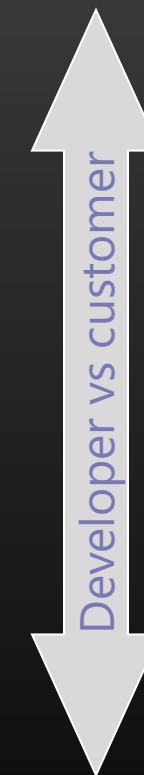


managing complexity



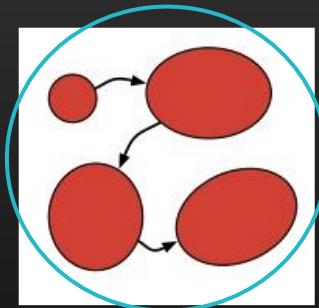
Do you get the expected results when the parts are put together?

Integration testing



Does the program satisfy the requirements?

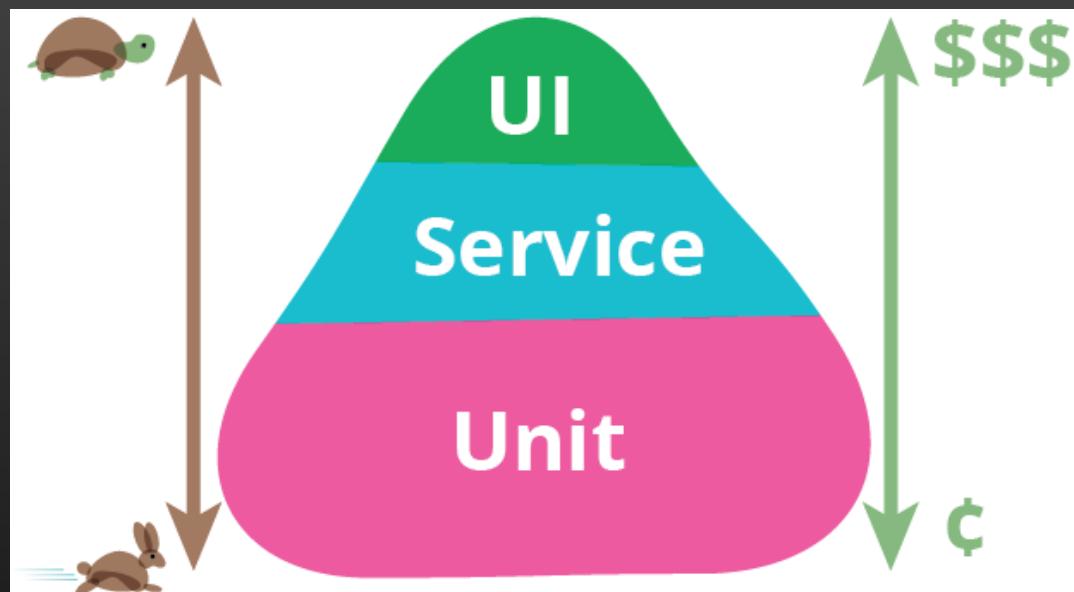
Acceptance / Functional Testing



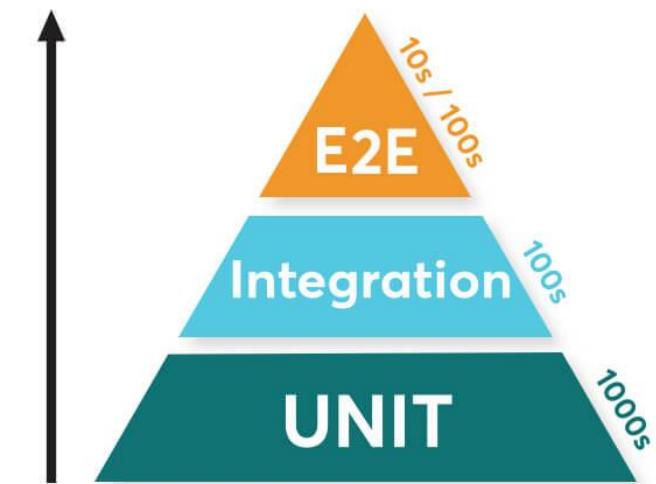
System testing

The whole system functions as expected, in the target config?

Test pyramid

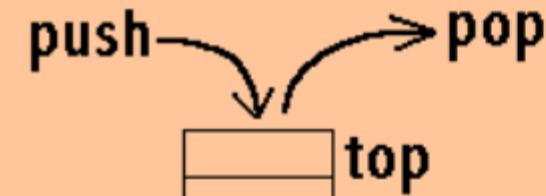


<https://martinfowler.com/bliki/TestPyramid.html>



<https://www.blazemeter.com/blog/agile-development-and-testing-an-introduction> 15

Unit test: stack contract



Operations

- `push(x)`: add an item on the top
- `pop`: remove the item at the top
- `peek`: return the item at the top (without removing it)
- `size`: return the number of items in the stack
- `isEmpty`: return whether the stack has no items

Unit test example: Verifying the unit contract

A stack is empty on construction

A stack has size 0 on construction

After n pushes to an empty stack, $n > 0$, the stack is not empty $\&\&$ its size is n

If one pushes x then pops, the value popped is x, the size is decreased by one.

If one pushes x then peeks, the value returned is x, but the size stays the same

If the size is n, then after n pops, the stack is empty and has a size 0

Popping from an empty stack does throw a
NoSuchElementException

Peeking into an empty stack does throw a
NoSuchElementException

For bounded stacks only, pushing onto a full stack does throw an
IllegalStateException

→ See also: Ray Toal's notes.

UI testing example: Web applications testing automation with Selenium

Test Suites			
	Command	Target	Value
livaria-fnac-suite*	open	https://www.fnac.pt/	
● results-valerio-5 *	type	id=Fnac_Search	Valério Romão
Untitled Test Case *	click	//button[@type='submit']	
no-results *	click	//div[3]/div/div[2]	
	assertTitle	Valério Romão, uma pesquisa em Livros na Fnac.pt	
	assertText	link=Autismo	Autismo
	assertText	link=Cair Para Dentro	Cair Para Dentro



Katalon Recorder

Ultimate Selenium IDE to record, play, and debug app. Fast and extensible!

Practices of SQA

Testing

Software configuration management

Versions management

Code improvement

Reviews, shared practices, static analysis,...

Issue tracking and task management

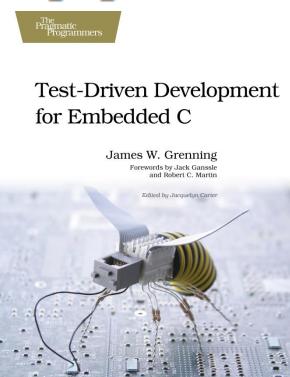
Continuous integration

Test-driven development

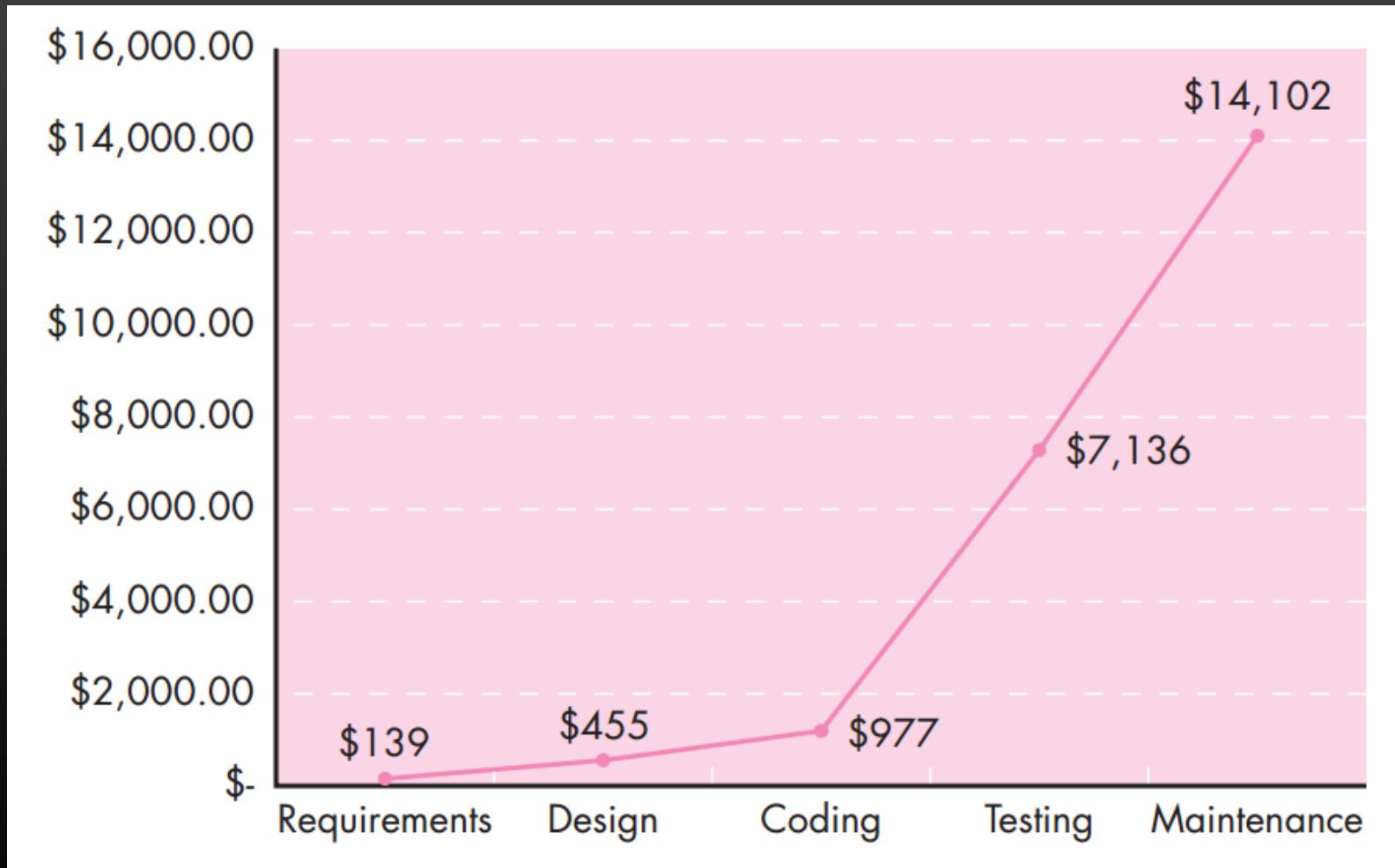
Debug Later Programming

We've all done it—written a bunch of code and then toiled to make it work. **Build it and then fix it.** Testing was something we did after the code was done. It was always an afterthought, but it was the only way we knew.

We would spend about half our time in the unpredictable activity affectionately called *debugging*. Debugging would show up in our schedules under the guise of test and integration. It was always a source of risk and uncertainty. Fixing one bug might lead to another and sometimes to a cascade of other bugs. We'd keep statistics to help predict

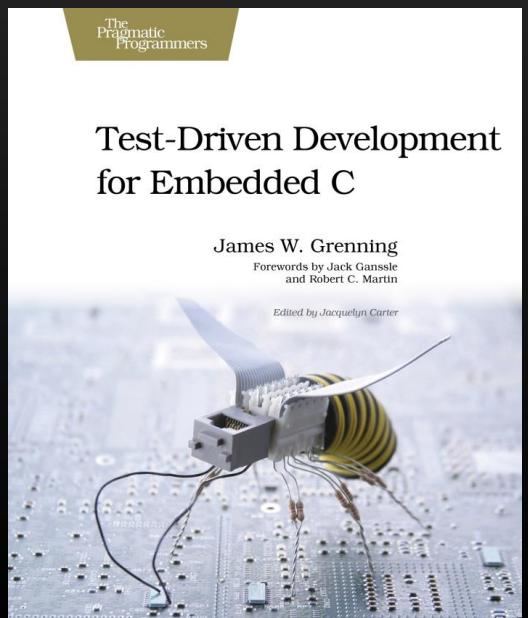


The cost of correcting an error raises exponentially along the sw lifecycle

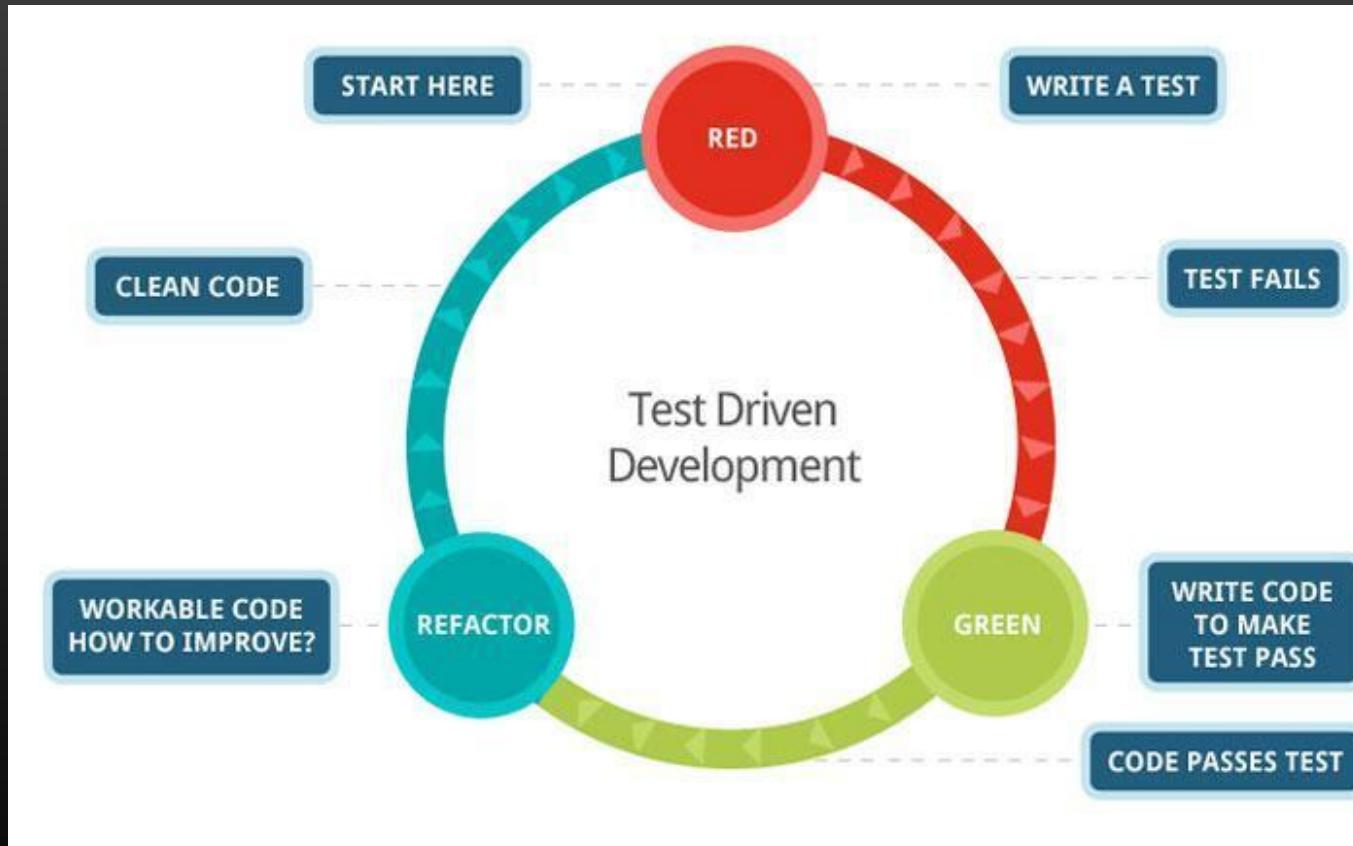


Boehm, B., and V. Basili, "Software Defect Reduction Top 10 List," IEEE Computer,
vol. 34, no. 1, January 2001, pp. 135-137. <http://doi.ieeecomputersociety.org/10.1109/2.962984>

Test-Driven Development is a technique for building software incrementally. Simply put, no production code is written without first writing a failing unit test. Tests are small. Tests are automated. Test-driving is logical. Instead of diving into the production code, leaving testing for later, the TDD practitioner expresses the desired behavior of the code in a test. The test fails. Only then do they write the code, making the test pass.



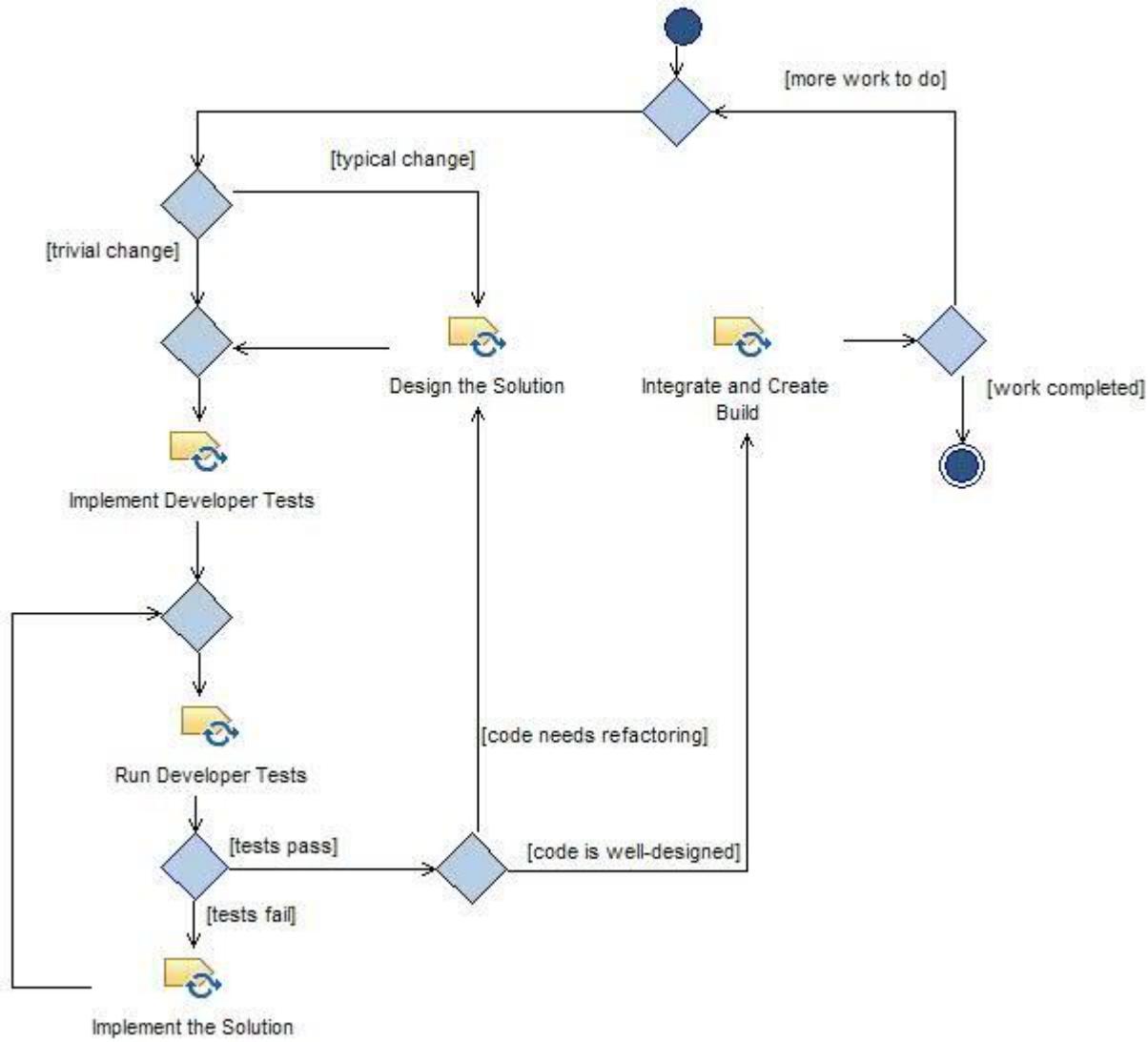
TDD: Test Driven Development



At the core of TDD is a repeating cycle of small steps known as the TDD microcycle. Each pass through the cycle provides feedback answering the question, does the new and old code behave as expected? The feedback feels good. Progress is concrete. Progress is measurable. Mistakes are obvious.

The steps of the TDD cycle in the following list are based on Kent Beck's description in his book *Test-Driven Development* [Bec02]:

1. Add a small test.
2. Run all the tests and see the new one fail, maybe not even compile.
3. Make the small changes needed to pass the test.
4. Run all the tests and see the new one pass.
5. Refactor to remove duplication and improve expressiveness.



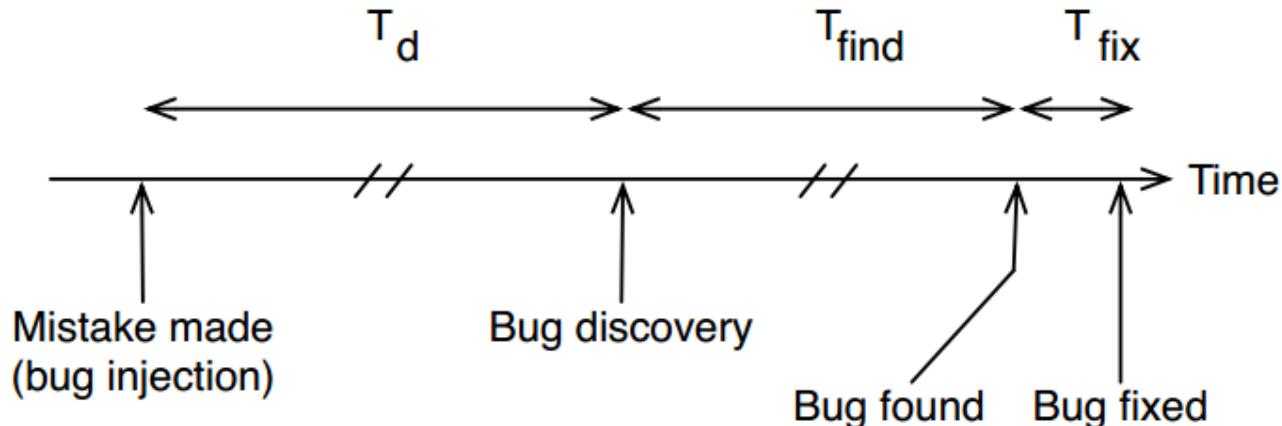


Figure 1.1: Physics of Debug-Later Programming

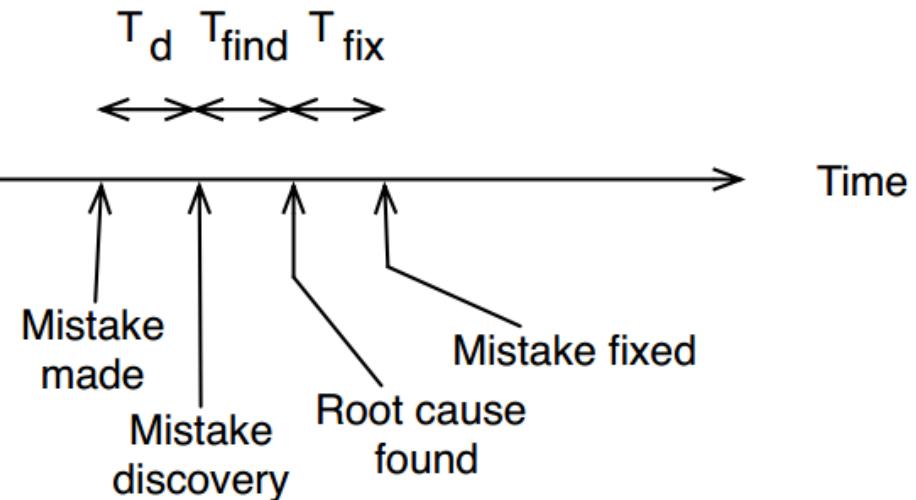


Figure 1.2: Physics of Test-Driven Development

The benefits of TDD: an investigation

A summary of selected empirical studies of test-driven development: industry participants*

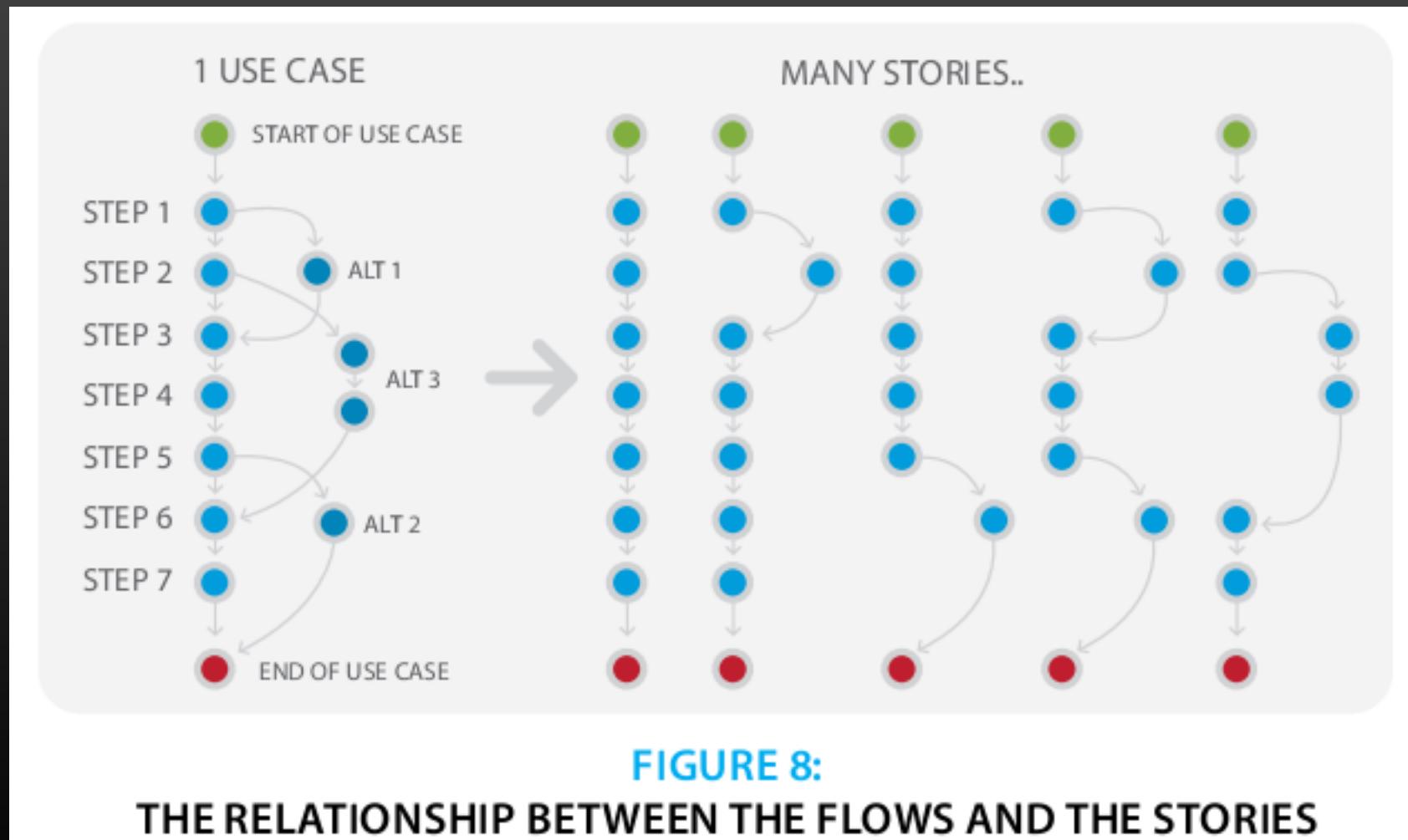
Family of studies	Type	Development time analyzed	Legacy project?	Organization studied	Software built	Software size	No. of participants	Language	Productivity effect
Sanchez et al. ⁶	Case study	5 years	Yes	IBM	Point-of-sale device driver	Medium	9–17	Java	Increased effort 19%
Bhat and Nagappan ⁷	Case study	4 months	No	Microsoft	Windows networking common library	Small	6	C/C++	Increased effort 25–35%
	Case study	≈7 months	No	Microsoft	MSN Web services	Medium	5–8	C++/C#	Increased effort 15%
Canfora et al. ⁸	Controlled experiment	5 hours	No	Soluziona Software Factory	Text analyzer	Very small	28	Java	Increased effort by 65%
Damm and Lundberg ⁹	Multi-case study	1–1.5 years	Yes	Ericsson	Components for a mobile network operator application	Medium	100	C++/Java	Total project cost increased by 5–6%



Story Testing

Executable Use Cases

Stories, use cases, scenarios



A story and tests

Title (one line describing the user story)

Narrative:

As a [role]

I want [feature]

So that [benefit]

Acceptance Criteria: (present in user stories)

Scenario 1: Title

Given [context]

And [some more context].

When [event]

Then [outcome]

And [another outcome]...

Scenario 2:

Frank Can Add Another Person as a Friend

ID #115218319 Close

STORY TYPE Feature

POINTS Unestimated

STATE Start Unscheduled

REQUESTER RJ Ryan Jones

OWNERS <none> +

FOLLOW THIS STORY (1 follower)

Updated: less than a minute ago

DESCRIPTION (edit)

As Frank I want to add a friend I searched for to my friend network so that I can see their posts, they can see my posts and I can direct message them

GIVEN I have searched for a friend's name
WHEN I select "Add Friend" next to my friend's name
THEN my friend's name should appear in my friend list on my homepage

Dev Notes: The added friend needs to be added to the Frank's friends in database
Design Notes: Attached are mocks for the button and placement

LABELS

add friend |x| individual user |x|

→ Principles for user stories content

Behavior Driven Development: Given, When, Then style

Structured syntax ([Gherkin](#))

Feature: what

Scenario: some determinable business situation

Given: preparation/setup (e.g.: requirements)

- And...

When: the set of actions (execute).

- And...

Then: specifies the expected result

- And...



The screenshot shows a code editor with a Gherkin feature file. The file starts with a feature block: **Feature: Multiple site support**. It then defines a background scenario with four given conditions: a global administrator named "Greg", a blog named "Greg's anti-tax rants", a customer named "Wilson", and a blog named "Expensive Therapy" owned by "Wilson". Following the background, there are two scenarios: one for Wilson posting to his own blog and one for Greg posting to a client's blog. Each scenario includes a given condition (I am logged in as Wilson/Greg), a when action (try to post to "Expensive Therapy"), and a then outcome (should see "Your article was published").

```
Feature: Multiple site support

  Background:
    Given a global administrator named "Greg"
    And a blog named "Greg's anti-tax rants"
    And a customer named "Wilson"
    And a blog named "Expensive Therapy" owned by "Wilson"

  Scenario: Wilson posts to his own blog
    Given I am logged in as Wilson
    When I try to post to "Expensive Therapy"
    Then I should see "Your article was published."

  Scenario: Greg posts to a client's blog
    Given I am logged in as Greg
    When I try to post to "Expensive Therapy"
    Then I should see "Your article was published."
```

→ Example from behat documentation.

Acceptance criteria should be executable

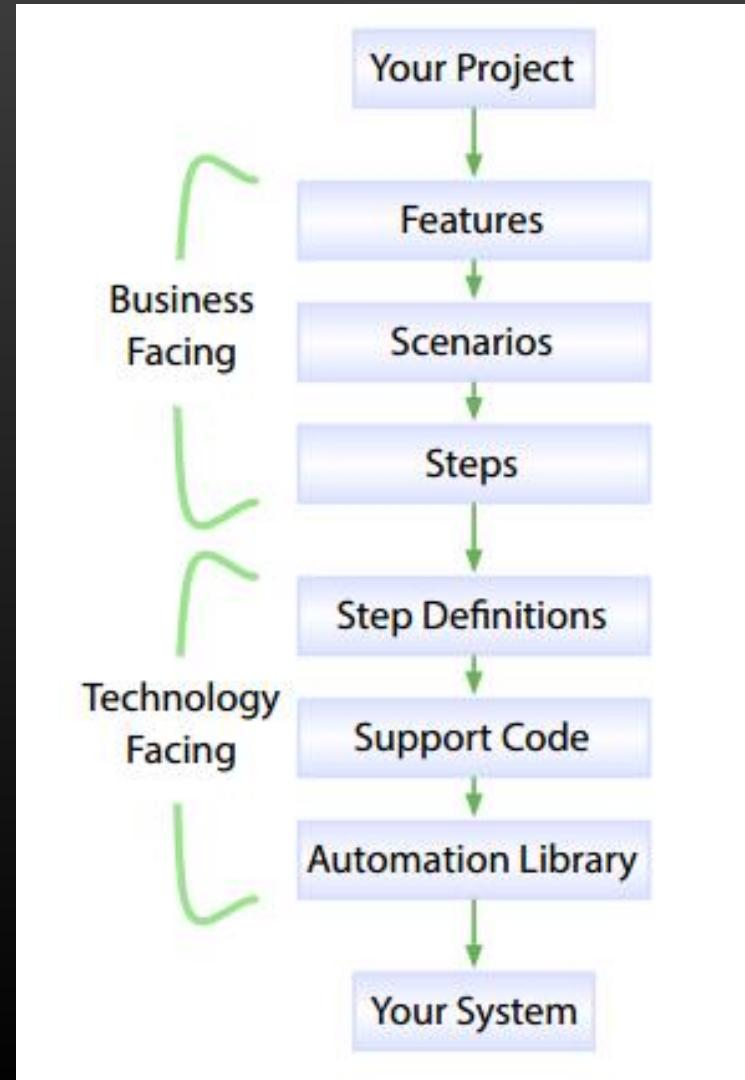
Cucumber reads specifications from plain-language text files called **features**, examines them for **scenarios** to test.

Each scenario is a list of **steps** for Cucumber to work through.

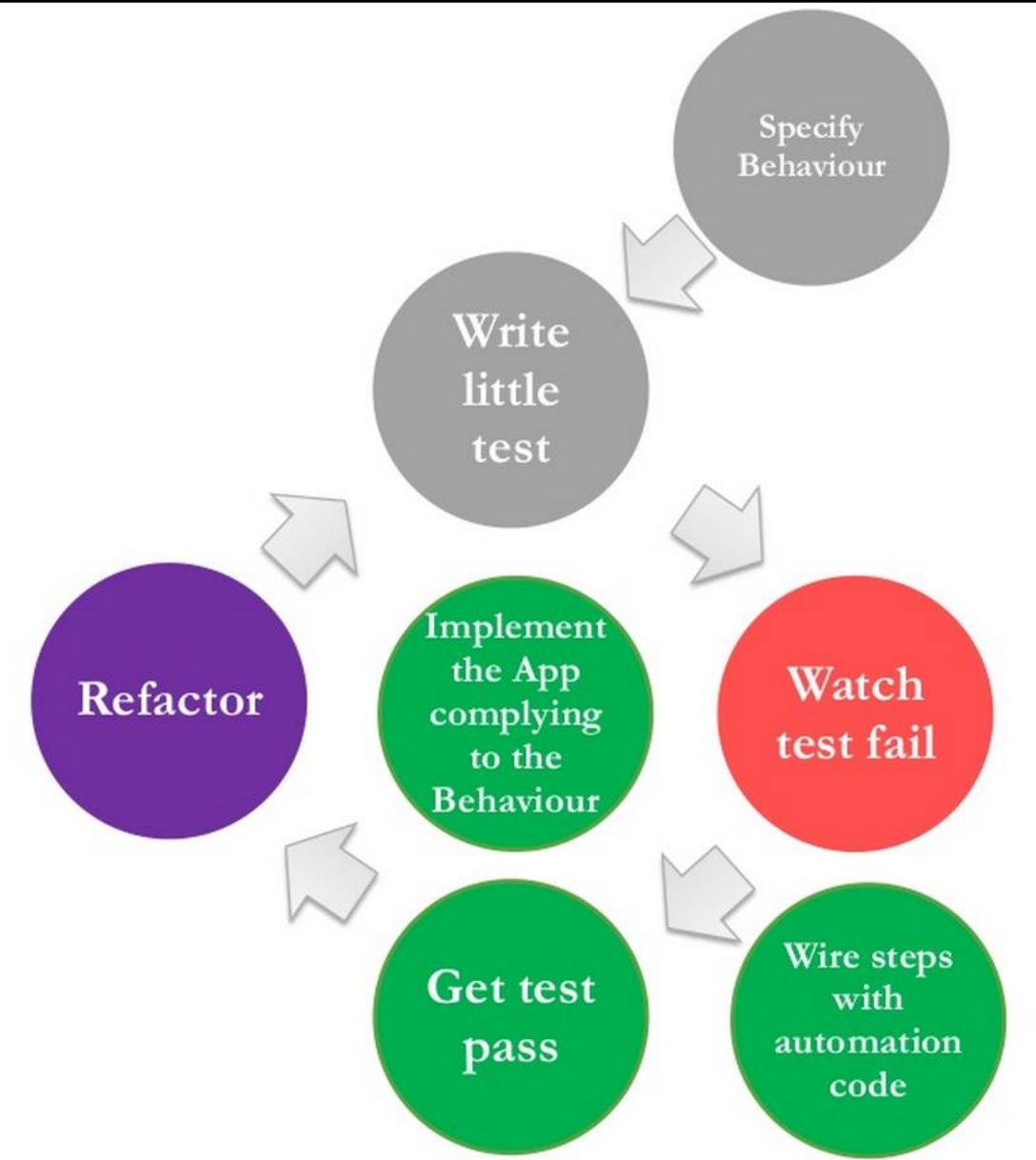
Along with the features, you give Cucumber a set of **step definitions**, which map the business-readable language of each step into code to carry out whatever action is being described by the step.

The step definition itself will probably just be one or two lines of code that delegate to a library of **support code**, specific to the domain of your application.

Sometimes that may involve using an **automation library**, like the browser automation library Selenium.



BDD: Behaviour-driven development



Credit: Nalin Goonawardana

Uncomplicate TDD and BDD

by JEFF NYMAN posted on 17 SEPTEMBER 2017

BDD IS AN ABSTRACTION OF TDD

So here's how I see it. The key value of TDD is that at each step of the way, you have demonstrably relevant working software as well as an itemized set of what we can call "executable specifications" that illustrate aspects of behavior. And we do this at the appropriate level of abstraction. Which takes us to BDD.

BDD is really just the addition of business concerns to the technical concerns that we deal with in TDD. BDD wasn't a reaction to TDD, as is often stated. BDD was simply an approach that let us move up the abstraction chain *as people became more comfortable with TDD*.

Simply put, BDD is about writing those conditions we talked about in the context of scenarios such that they will tell us the kind of behavior change they affect. A good barometer for adding a scenario might be asking if the scenario you are writing would be worth explaining to a business stakeholder. And you can frame that by asking what value it provides them. What aspect of the overall user experience is being captured in that scenario?

And not just "failure" in a singular sense, but failure modes based on the likely sensitivities of

<http://testerstories.com/2017/09/uncomplicate-tdd-and-bdd/>

context of your immediate work.

References

Core readings	Suggested readings
<ul style="list-style-type: none">• “<u>Story testing</u> - executable use cases - for embedded systems”, J. Grenning	<ul style="list-style-type: none">• [Dennis]- Chap. 12.• [Pressman] – Chap. 17 (“Software Testing Strategies)