Software Engineering

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Maxime Lefrançois https://maxime.lefrancois.info
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Software Engineering

Part 1 – Introduction

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Maxime Lefrançois https://maxime.lefrancois.info
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Objectives of this course

The aim of this session is for you to learn about Software Engineering

Software engineering is the systematic application of engineering approaches to the development of software.
A software engineer is a person who applies the principles of software engineering to design, develop, maintain, test, and evaluate computer software. The term programmer is sometimes used as a synonym, but may also lack connotations of engineering education or skills.

— Wikipedia contributors - https://en.wikipedia.org/wiki/Software_engineering

60s-80s – The Software Crisis

The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

— Edsger Dijkstra, The Humble Programmer (EWD340), 1972 Turing Award Lecture

Fun story

Public loo guilty of making nuisance calls

Nick Rothwell <nick@cassiel.com>
21 Aug 1997 15:39:14 -0000

From *Computer Weekly* (UK), 21st August 1997:

A woman who was phoned repeatedly by a public lavatory asking her to fill it with cleaning fluid had to ask BT to put a stop to the calls.

The case is one of a growing number of nuisance calls generated by programming errors.

About 15% of all nuisance calls are caused by errors, most of which are traceable to faulty programming, according to a BT spokesperson.

The most common type of computer-controlled nuisance call is from soft drink vending machines which need refilling. Wrongly programmed fax machines and modems are another cause of complaints.

In a recent case, a North Sea oil rig called the wrong number at regular intervals to ask for a service. Potentially serious cases involve traffic lights, boilers and hospital refrigerators.

"The calls are mainly silent, because they are intended for modems to pick up, but some give a recorded message," said a BT spokesman.

Nick Rothwell, CASSIEL http://www.cassiel.com
contemporary dance projects music synthesis and control

[Not a new story in RISKS, but it seems to be happening more often. PGN]

source: http://catless.ncl.ac.uk/Risks/19.33.html%23subj1

Not fun story: Therac-25

MAN KILLED BY ACCIDENT WITH MEDICAL RADIATION
(excerpted from The Boston Globe, June 20, 1986, p. 1)
by Richard Saltos, Globe Staff

A series of accidental radiation overdoses from identical cancer therapy machines in Texas and Georgia has left one person dead and two others with deep burns and partial paralysis, according to federal investigators.

Evidently caused by a flaw in the computer program controlling the highly automated devices, the overdoses - unreported until now - are believed to be the worst medical radiation accidents to date.

The malfunctions occurred once last year and twice in March and April of this year in two of the Canadian-built linear accelerators, sold under the name Therac 25.

Two patients were injured, one who died three weeks later, at the East Texas Cancer Center in Tyler, Texas, and another at the Kennestone Regional Oncology Center in Marietta, Ga.

The defect in the machines was a "bug" so subtle, say those familiar with the cases, that although the accident occurred in June 1985, the problem remained a mystery until the third, most serious accident occurred on April 11 of this year.

Late that night, technicians at the Tyler facility discovered the cause of that accident and notified users of the device in other cities.

https://catless.ncl.ac.uk/Risks/3.09.html
https://en.wikipedia.org/wiki/History_of_software_engineering#1965_to_1985:_The_software_crisis

Fun bugs

F-16 Problems (from Usenet net.aviation)

Bill Janssen <janssen@mcc.com>
Wed, 27 Aug 86 14:31:45 CDT

A friend of mine who works for General Dynamics here in Ft. Worth wrote some of the code for the F-16, and he is always telling me about some neato-whiz-bang bug/feature they keep finding in the F-16:

o Since the F-16 is a fly-by-wire aircraft, the computer keeps the pilot from doing dumb things to himself. So if the pilot jerks hard over on the joystick, the computer will instruct the flight surfaces to make a nice and easy 4 or 5 G flip. But the plane can withstand a much higher flip than that. So when they were 'flying' the F-16 in simulation over the equator, the computer got confused and instantly flipped the plane over, killing the pilot [in simulation]. And since it can fly forever upside down, it would do so until it ran out of fuel.

(The remaining bugs were actually found while flying, rather than in simulation):

o One of the first things the Air Force test pilots tried on an early F-16 was to tell the computer to raise the landing gear while standing still on the runway. Guess what happened? Scratch one F-16. (my friend says there is a new subroutine in the code called 'wait_on_wheels' now...) [weight?]

o The computer system onboard has a weapons management system that will attempt to keep the plane flying level by dispersing weapons and empty fuel tanks in a balanced fashion. So if you ask to drop a bomb, the computer will figure out whether to drop a port or starboard bomb in order to keep the load even. One of the early problems with that was the fact that you could flip the plane over and the computer would gladly let you drop a bomb or fuel tank. It would drop, dent the wing, and then roll off.

https://catless.ncl.ac.uk/Risks/3.44.html

Many more stories: RISKS Digest

Forum on Risks to the Public in Computers and Related Systems
ACM Committee on Computers and Public Policy, Peter G. Neumann, moderator
http://catless.ncl.ac.uk/Risks/

Examples of Volume 1, 1985
Legend: ! = Loss of Life; * = Potentially Life-Critical; $ = Loss of Money/Equipment; S = Security/Privacy/Integrity Flaw

!S Arthritis-therapy microwaves set pacemaker to 214, killed patient (SEN 5 1)
*$ Mariner 18: aborted due to missing NOT in program (SEN 5 2)
*$ F18: plane crashed due to missing exception condition, pilot OK (SEN 6 2)
*$ El Dorado brake computer bug caused recall of all El Dorados (SEN 4 4)
* Second Space Shuttle operational simulation: tight loop upon cancellation of an attempted abort; required manual override (SEN 7 1)
* Gemini V 100mi landing err, prog ignored orbital motion around sun (SEN 9 1)
* F16 simulation: plane flipped over whenever it crossed equator (SEN 5 2)
* F16 simulation: upside-down F16 deadlock over left vs. right roll (SEN 9 5)
* SF BART train doors sometimes open on long legs between stations (SEN 8 5)
* IRS reprogramming cost USA interest on at least 1,150,000 refunds (SEN 10 3)
Santa Clara prison data system (inmate altered release date) (SEN 10 1).
Computerized time-bomb inserted by programmer (for extortion?) (10 3)
*$ Colorado River flooding in 1983, due to faulty weather data and/or faulty model; too much water was kept dammed prior to spring thaws.
$ 1979 AT&T program bug downed phone service to Greece for months (SEN 10 3)
Quebec election prediction gave loser big win [1981] (SEN 10 2, p. 25-26)
SW vendor rigs elections? (David Burnham, NY Times front page, 29 July 1985)
Vancouver Stock Index lost 574 points over 22 months -- roundoff (SEN 9 1)

Productivity and quality issues in software …

Due to:

- increase in size and complexity of systems
- shorter and shorter deadlines
- bigger and bigger teams, with multiple skills

Causing:

- Cost and Budget Overruns
- Property Damage
- Life and Death

Productivity and quality issues in software …

Due to:

- increase in size and complexity of systems
- shorter and shorter deadlines
- bigger and bigger teams, with multiple skills

Causing:

- Cost and Budget Overruns
- Property Damage
- Life and Death

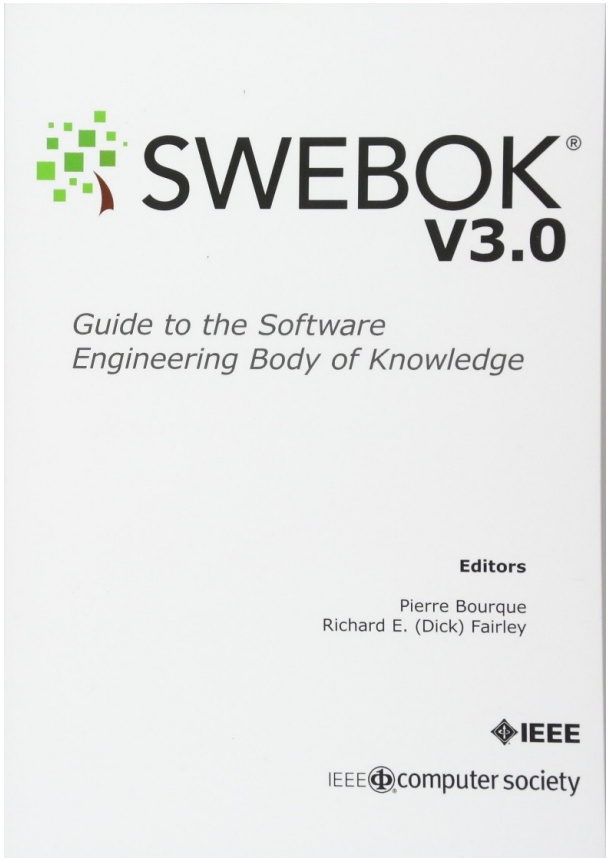
… called for the development of Software Engineering

Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software)

— ISO/IEC/IEEE 24765:2017 Systems and Software Engineering Vocabulary (SEVOCAB)

SWEBOK: **S**oft**w**are **E**ngineering **B**ody **o**f **K**nowledge

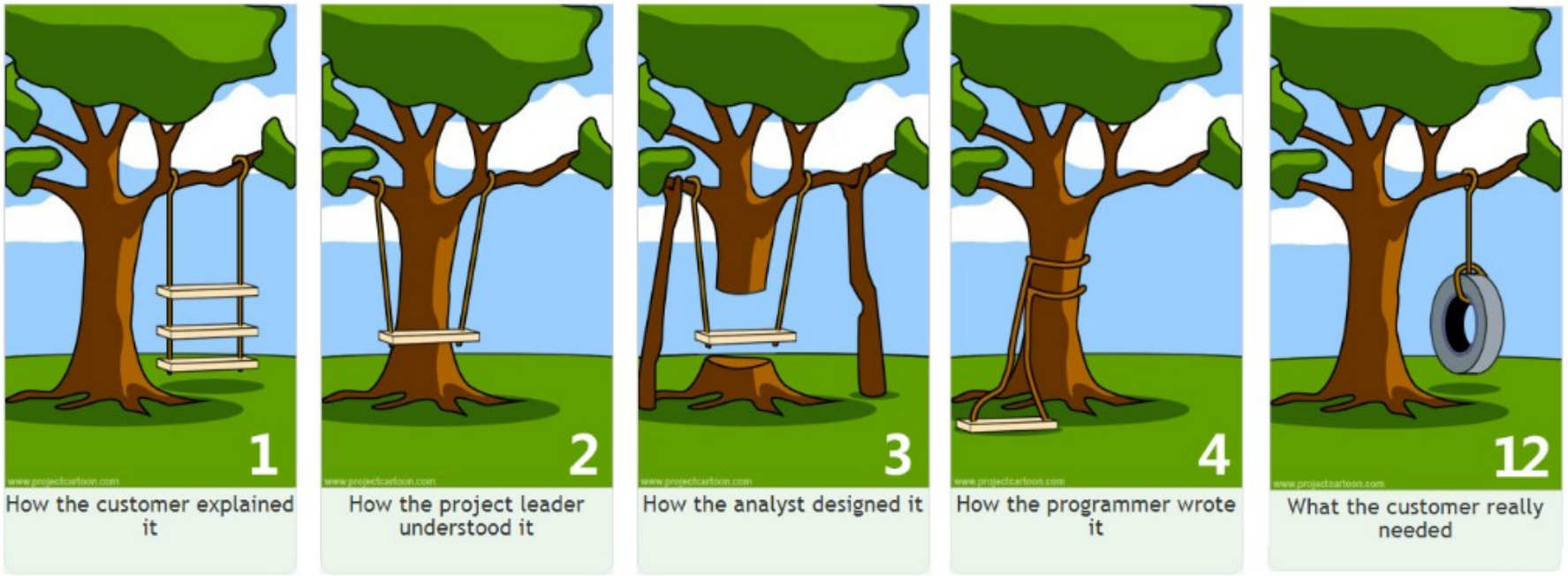
ISO/IEC TR 19759:2015 (v1 in 2005)



Describes 15 Knowledge Areas (KAs) in the field of software engineering

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Engineering Models and Methods
- Software Quality
- Software Engineering Professional Practice
- Software Engineering Economics
- Computing Foundations
- Mathematical Foundations
- Engineering Foundations

get your version here: https://www.computer.org/education/bodies-of-knowledge/software-engineering



source: https://www.zentao.pm/agile-knowledge-share/tree-swing-project-management-cartoon-97.mhtml

Software Engineering

Part 2 – Software Requirements

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Maxime Lefrançois https://maxime.lefrancois.info
Course unit URL: https://ci.mines-stetienne.fr/cps2/softeng/

Software Requirement - definition

1. software capability needed by a user to solve a problem or to achieve an objective
2. software capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document

— ISO/IEC/IEEE 24765:2017 Systems and Software Engineering Vocabulary (SEVOCAB)

verifiable

- if possible: quantifiable
- verification at the individual level, or at the system level
- verification may be difficult or costly

may be **prioritized** (enables tradeoffs)

may have **status values** (enables project progress monitoring)

Software Requirement - definition

1. software capability needed by a user to solve a problem or to achieve an objective
2. software capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document

— ISO/IEC/IEEE 24765:2017 Systems and Software Engineering Vocabulary (SEVOCAB)

Product and Process requirements

product requirement
refinement of customer requirements into the developers' language, making implicit requirements into explicit derived requirements

— ISO/IEC/IEEE 24765:2017 Systems and Software Engineering Vocabulary (SEVOCAB)

process requirement
constraint on the development of the software

— ISO/IEC TR 19759:2015 Software Engineering Body of Knowledge (SWEBOK)

example of product requirement: “The software shall verify that a student meets all prerequisites before he or she registers for a course”
example of process requirement: “The software shall be developed using a Agile process”

process requirements can be imposed by the dev organization, the customer, a third party such as safety regulator

Functional and Nonfunctional requirements

functional requirement

1. statement that identifies what results a product or process shall produce

2. requirement that specifies a function that a system or system component shall perform

— ISO/IEC/IEEE 24765:2017 Systems and Software Engineering Vocabulary (SEVOCAB)
— IEEE 730-2014 IEEE Standard for Software Quality Assurance Processes, 3.2

nonfunctional requirement

1. software requirement that describes not what the software will do but how the software will do it

— ISO/IEC/IEEE 24765:2017 Systems and Software Engineering Vocabulary (SEVOCAB)

example of functional requirement: Business Rules, Transaction corrections, adjustments, and cancellations, …
example of nonfunctional requirement: “The interface shall be user-friendly / the authentification must be secure / streaming must be lightning fast”

see also blog post on functional vs nonfunctional requirements https://theappsolutions.com/blog/development/functional-vs-non-functional-requirements/

Functional and Nonfunctional requirements

functional requirement

1. statement that identifies what results a product or process shall produce

2. requirement that specifies a function that a system or system component shall perform

— ISO/IEC/IEEE 24765:2017 Systems and Software Engineering Vocabulary (SEVOCAB)
— IEEE 730-2014 IEEE Standard for Software Quality Assurance Processes, 3.2

nonfunctional requirement

1. software requirement that describes not what the software will do but how the software will do it

— ISO/IEC/IEEE 24765:2017 Systems and Software Engineering Vocabulary (SEVOCAB)

example of functional requirement: Business Rules, Transaction corrections, adjustments, and cancellations, …
example of nonfunctional requirement: “The interface shall be user-friendly / the authentification must be secure / streaming must be lightning fast”

further classification of **nonfunctional requirements**:
performance, maintainability, safety, reliability, security, interoperability, …

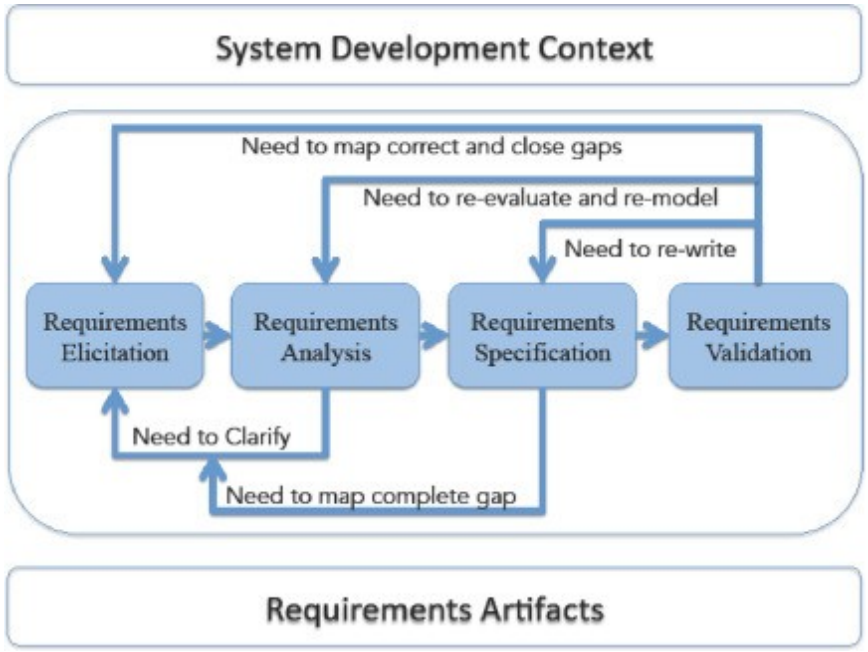
see also blog post on functional vs nonfunctional requirements https://theappsolutions.com/blog/development/functional-vs-non-functional-requirements/

Requirements process

includes i. **elicitation**, ii. **analysis**, iii. **specification**, and iv. **validation**.

is initiated at the beginning of a project, **and refined throughout the life cycle of the project**

requirements are configuration items. They can be managed during the life cycle of the project



Incremental and iterative requirements engineering process.
Source: Jin, Zhi. *Environment modeling-based requirements engineering for software intensive systems*. Morgan Kaufmann, 2018. Chapter 1 - Requirements and Requirements Engineering

⚠ Software projects are critically vulnerable when the requirements related activities are poorly performed.

Requirements process – actors

users
will operate the software. Heterogenous goup involving people with different roles

customers
those who commissioned the software or who represent the target market

market analyst
for mass-market software, marketing people act as proxy customers

regulators
impose requirements of the regulatory authorities (ex, banking, public transport, utilitie)

software engineers
legitimate interest in optimizing the actual development time and costs

tradeoffs need to be negotiated
see for example https://www.bbau.ac.in/dept/dit/TM/Requirement%20Validation.pdf

Requirements process – i. elicitation

Sources

- goals of the software
- domain knowledge
- stakeholders
- business rules
- operational environment
- organizational environment

Elicitation techniques

- interviews
- scenarios
- prototypes
- facilitated meetings
- observation
- user stories https://en.wikipedia.org/wiki/User_story

"As a <role>, I want <goal/desire> so that <benefit>."

- …

Requirements process – ii. analysis

Classify the requirements

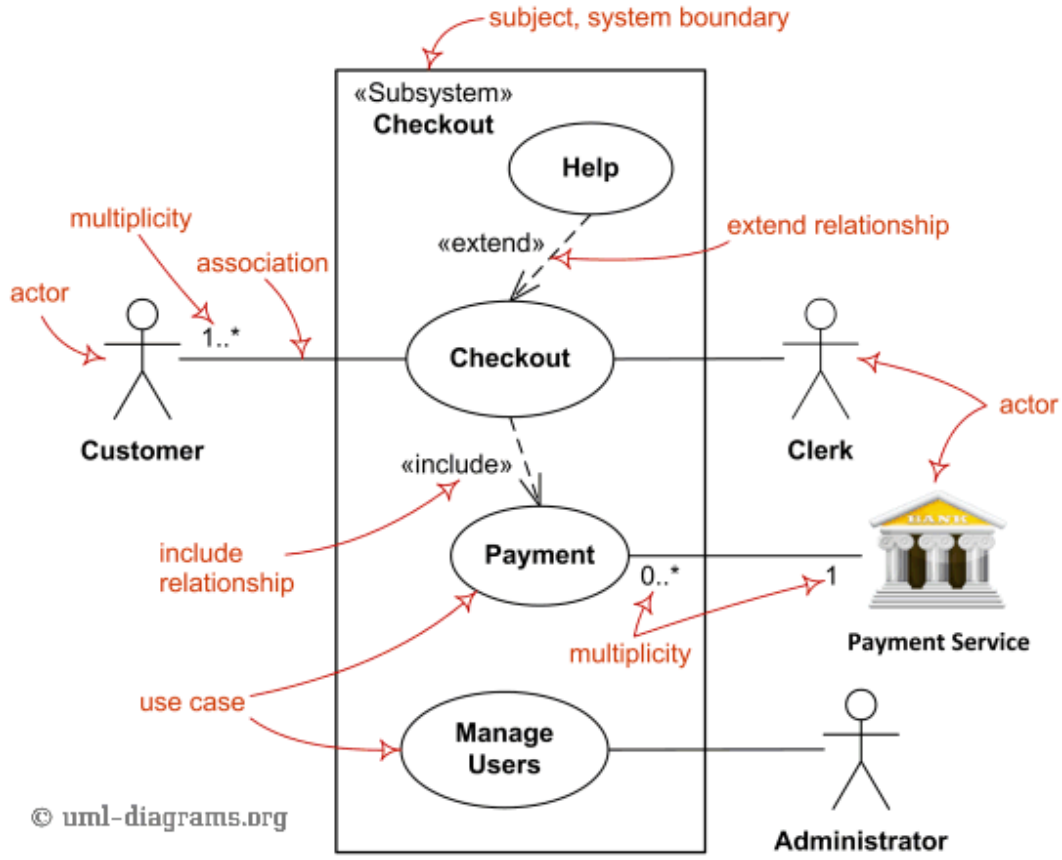
- functional/nonfunctional
- source (e.g., user, regulation)
- on the product or the process
- priority (mandatory, highly desirable, desirable, optional)
- scope (global, narrow)
- Volatility/stability

Requirements process – ii. analysis

Conceptual models

For example with the *Unified Modeling Language (UML)*

- use case diagrams
- communication diagrams
- state machine diagrams
- …



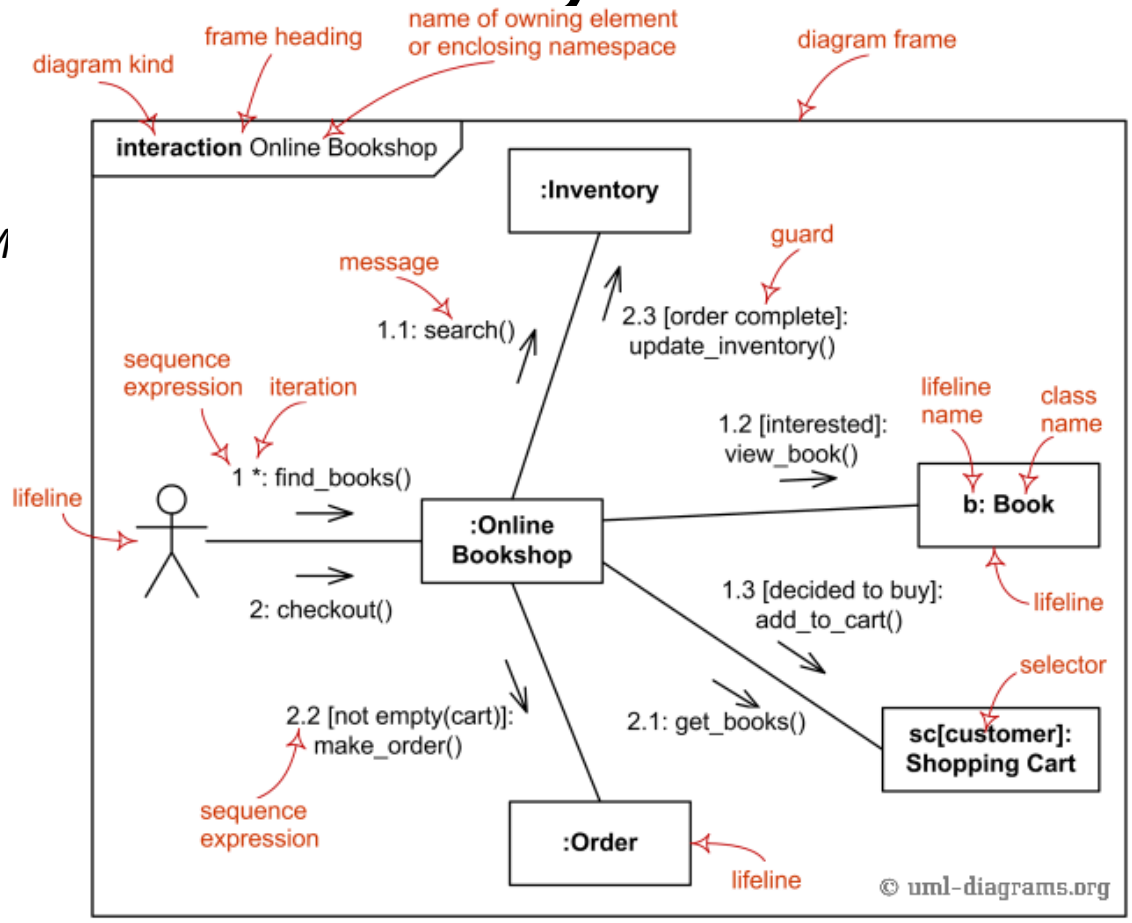
exemple: UML Use case diagrams
source: https://www.uml-diagrams.org/use-case-diagrams.html

Requirements process – ii. analysis

Conceptual models

For example with the *Unified Modeling Language (UM*

- use case diagrams
- communication diagrams
- state machine diagrams
- …



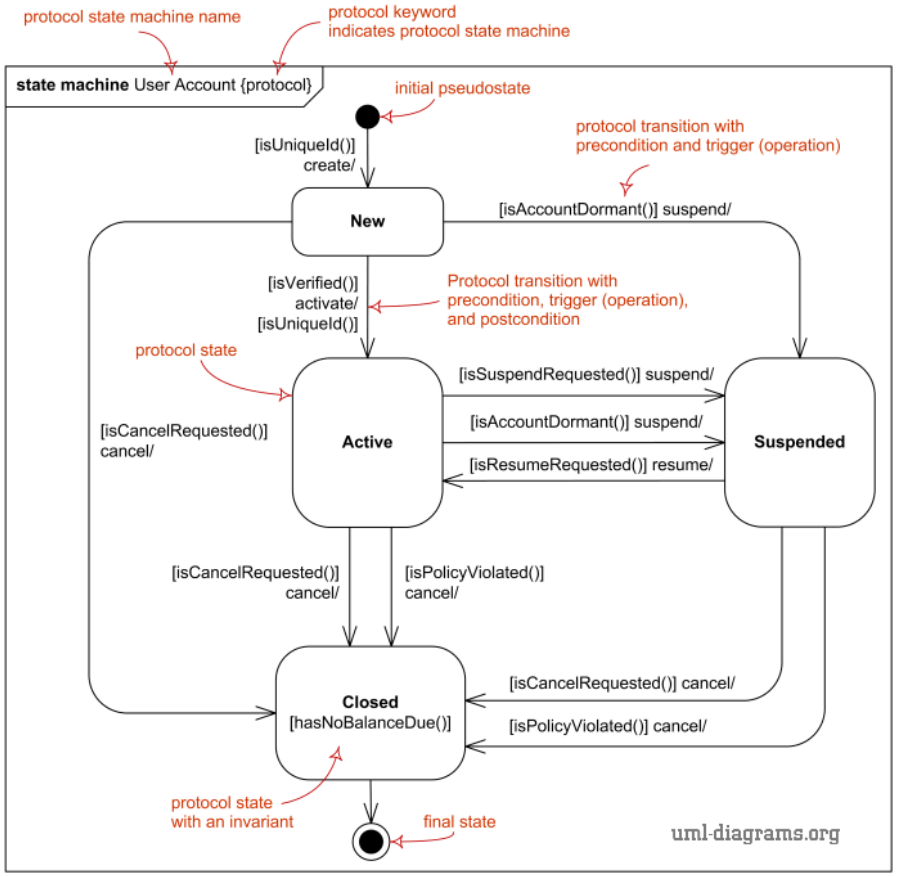
exemple: UML Communication diagrams
source: https://www.uml-diagrams.org/communication-diagrams.html

Requirements process – ii. analysis

Conceptual models

For example with the *Unified Modeling Language (UML)*

- use case diagrams
- communication diagrams
- state machine diagrams
- …



exemple: UML state machine diagrams
source: https://www.uml-diagrams.org/protocol-state-machine-diagrams.html

Requirements process – ii. analysis

Architectural Design
"point when the requirement process overlaps with the software/system design"

Requirement allocation
Requirements need to be allocated to the architecture/design component that will be responsible for satisfying the requirement

Demonstrates that the requirement process is not only an upfront analysis task !

Requirements process – ii. analysis

Conflicts may arise …

- stakeholders require incompatible features
- incompatibilities between requirements and resources
- incompatibilities between functional and nonfunctional requirements
- …

Negotiation is important

- consult with the stakeholders instead of making unilateral decisions
- refine priorization
- estimate wisely cost and time
- keep traces the decisions

see for example https://www.bbau.ac.in/dept/dit/TM/Requirement%20Validation.pdf

Requirements process – ii. analysis

formal analysis

"application of mathematically rigorous techniques for the specification, development, and verification of software and hardware systems"

– What is formal method - https://shemesh.larc.nasa.gov/fm/fm-what.html

- ✓ costly
- ✓ important for safety-critical or security-critical software/systems
- ✓ permits static validation (for example, absence of deadlocks)

Methods

logic calculi, formal languages, automata theory, discrete event dynamic system, program semantics, type systems, algebraic datatypes

see for example https://en.wikipedia.org/wiki/Formal_methods

Requirements process – iii. specification

specification (in software engineering)

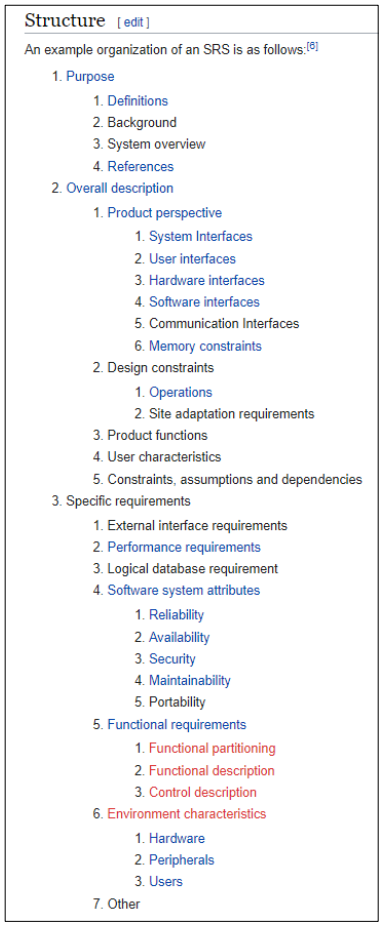
"production of a document that can be systematically reviewed, evaluated, and approved"

— ISO/IEC TR 19759:2015 Software Engineering Body of Knowledge (SWEBOK)

software requirements specification (SRS)

"structured collection of the essential requirements [functions, performance, design constraints and attributes] of the software and its external interfaces"

— IEEE 1012-2016 - IEEE Standard for System, Software, and Hardware Verification and Validation



Structure [edit]

An example organization of an SRS is as follows:[6]

1. Purpose
    1. Definitions
    2. Background
    3. System overview
    4. References
2. Overall description
    1. Product perspective
        1. System Interfaces
        2. User interfaces
        3. Hardware interfaces
        4. Software interfaces
        5. Communication Interfaces
        6. Memory constraints
    2. Design constraints
        1. Operations
        2. Site adaptation requirements
    3. Product functions
    4. User characteristics
    5. Constraints, assumptions and dependencies
3. Specific requirements
    1. External interface requirements
    2. Performance requirements
    3. Logical database requirement
    4. Software system attributes
        1. Reliability
        2. Availability
        3. Security
        4. Maintainability
        5. Portability
    5. Functional requirements
        1. Functional partitioning
        2. Functional description
        3. Control description
    6. Environment characteristics
        1. Hardware
        2. Peripherals
        3. Users
    7. Other

example organization of a SRS document
source: https://en.wikipedia.org/wiki/Software_requirements_specification

see also https://en.wikipedia.org/wiki/Software_requirements_specification

Requirements process – iv. validation

requirement validation
"confirmation by examination that requirements (individually and as a set) define the right system as intended by the stakeholders"

– ISO/IEC/IEEE 29148:2011 Systems and software engineering — Life cycle processes — Requirements engineering

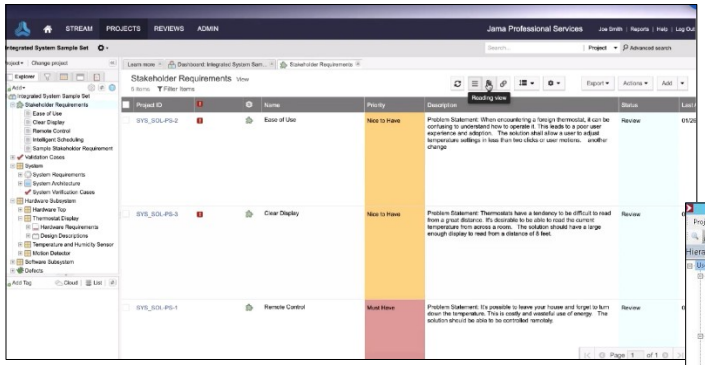
Techniques for requirement validation

- reviews
- inspections
- prototyping
- user manual development
- model validation
- requirements testing

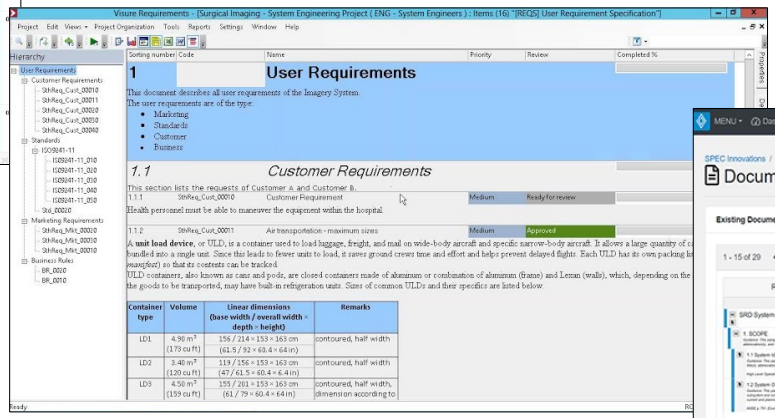
see for example https://en.wikipedia.org/wiki/Formal_methods

Software requirements tools

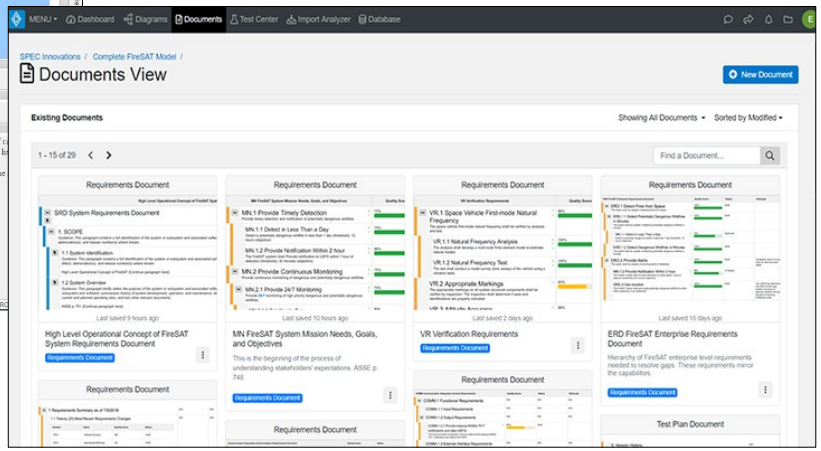
almost exclusively commercial tools



https://www.jamasoftware.com/solutions/software-development/

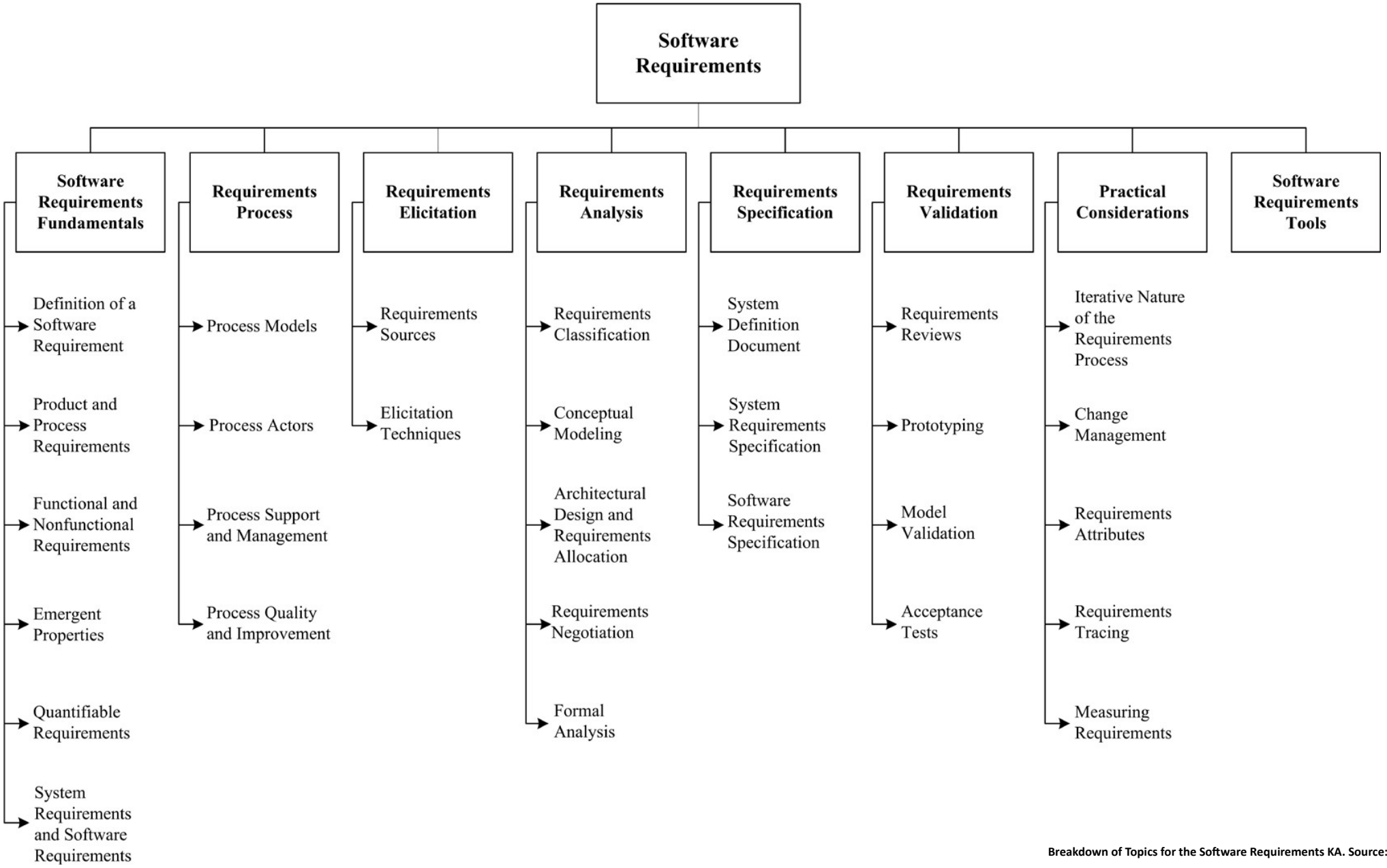


https://visuresolutions.com/



https://www.innoslate.com/requirements-management/

Detailed list here (see column « RM »)
https://en.wikipedia.org/wiki/List_of_requirements_engineering_tools



Breakdown of Topics for the Software Requirements KA. Source: SWEBOK V3

Software Engineering

Part 3 – The ISO/IEC 25010 System and software quality models

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Maxime Lefrançois https://maxime.lefrancois.info
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Software quality model (ISO/IEC 25010:2011)

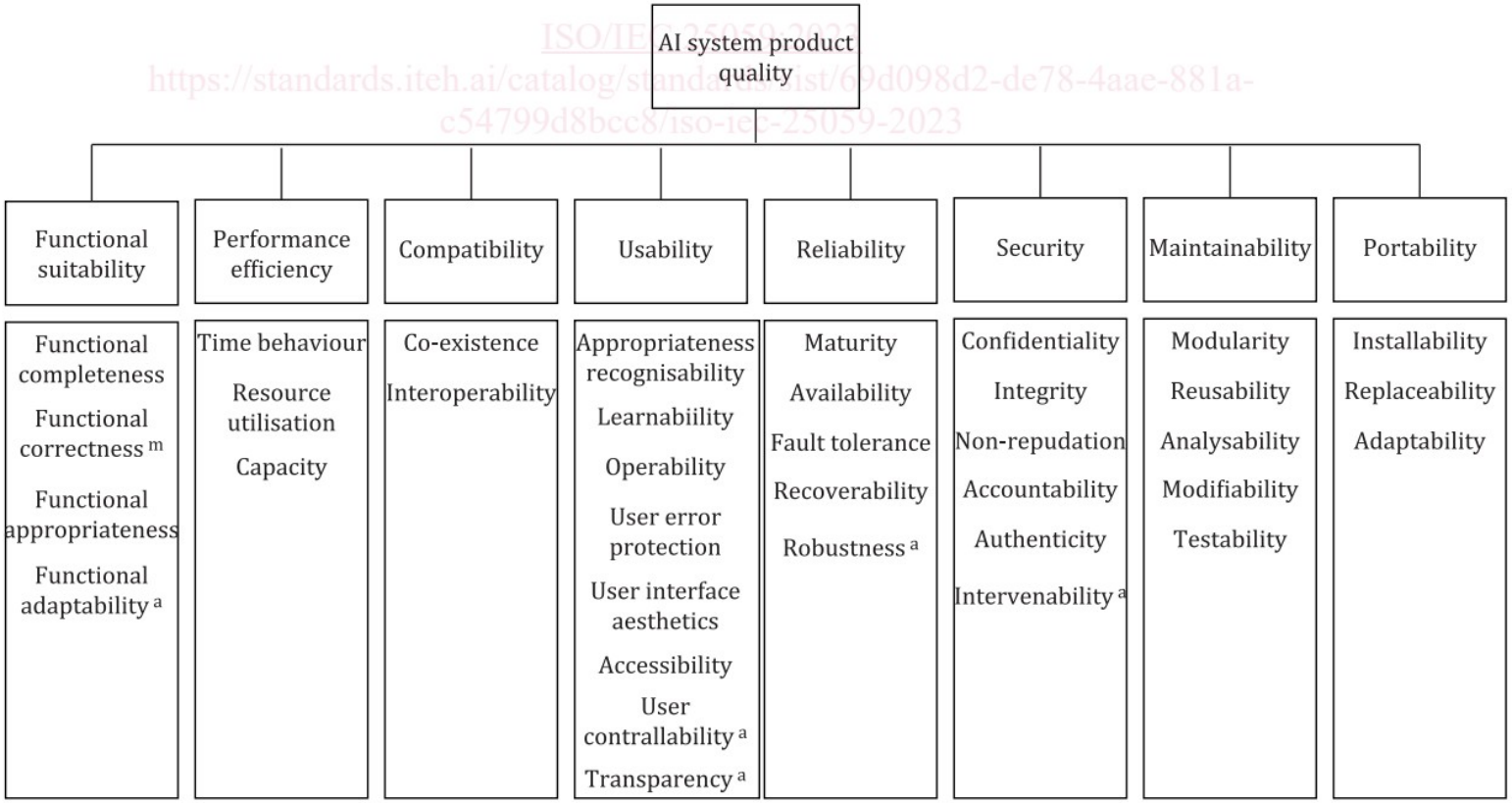
https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

software quality
degree to which a software product satisfies stated and implied needs when used under specified conditions

— ISO/IEC 25010:2011 System and software quality models

Quality model for AI systems (ISO/IEC 25059:2024)

https://iso25000.com/index.php/en/iso-25000-standards/iso-25059



Software quality model (ISO/IEC 25010:**2023**)

https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

software quality
degree to which the system satisfies the stated and implied needs of its various stakeholders, and thus provides value.

— ISO/IEC 25010:2011 System and software quality models

SOFTWARE PRODUCT QUALITY (NEW!)								
FUNCTIONAL SUITABILITY	**PERFORMANCE EFFICIENCY**	**COMPATIBILITY**	**INTERACTION CAPABILITY**	**RELIABILITY**	**SECURITY**	**MAINTAINABILITY**	**FLEXIBILITY**	**SAFETY**
FUNCTIONAL COMPLETENESS	TIME BEHAVIOUR	CO-EXISTENCE	APPROPRIATENESS RECOGNIZABILITY	FAULTLESSNESS (NEW!)	CONFIDENTIALITY	MODULARITY	ADAPTABILITY	OPERATIONAL CONSTRAINT
FUNCTIONAL CORRECTNESS	RESOURCE UTILIZATION	INTEROPERABILITY	LEARNABILITY	AVAILABILITY	INTEGRITY	REUSABILITY	SCALABILITY	RISK IDENTIFICATION
FUNCTIONAL APPROPRIATENESS	CAPACITY		OPERABILITY	FAULT TOLERANCE	NON-REPUDIATION	ANALYSABILITY	INSTALLABILITY	FAIL SAFE
			USER ERROR PROTECTION	RECOVERABILITY	ACCOUNTABILITY	MODIFIABILITY	REPLACEABILITY	HAZARD WARNING
		MODIFIED	USER ENGAGEMENT		AUTHENTICITY	TESTABILITY		SAFE INTEGRATION
			INCLUSIVITY		RESISTANCE (NEW!)			
			USER ASSISTANCE					
			SELF-DESCRIPTIVENESS					

iso25000.com

Evaluation process (ISO/IEC 25040:2011)

https://iso25000.com/index.php/en/iso-25000-standards/iso-25040

Five activities, each having different steps

1	**Define the evaluation**
2	**Design the evaluation**
3	**Plan the evaluation**
4	**Execute the evaluation**
5	**Conclude the evaluation**

iso25000.com

Activity 1: Define the evaluation

The first step in the evaluation process is to define the scope by establishing the purpose, evaluation criteria, target entities, and other relevant factors.

Task 1.1: Establish the purpose

The goal of this task is to define the purpose of the quality evaluation (evaluate suitability to a specific context of use, evaluate qualitfication to a quality standard, check requirements satisfaction, evaluate for suitability to the market, etc.).

Task 1.2: Identify target entities

The goal of this task is to identify all target entities needed for the evaluation.

Task 1.3:Define quality evaluation criteria

The quality evaluation criteria shall be defined or identified. Quality evaluation criteria are a set of specific quality requirements used to evaluate the quality of the target entities, and can include factors such as functional suitability, reliability, performance efficiency, compatibility, interaction capability, maintainability, flexibility, security, safety, or their subcharacteristics.

Task 1.4: Define requirements for the rigor of evaluation

The rigor (thoroughness, precision, and strictness) of the evaluation shall be defined in order to ensure the accuracy, reliability, and validity of the results.

Software Engineering

Part 4 – Software Engineering Process

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Software engineering methods – definition

software engineering method
organized and systematic approach to developing software for a target computer

— ISO/IEC TR 19759:2015 Software Engineering Body of Knowledge (SWEBOK)

objectives:

- facilitate human understanding, communication, and coordination
- aid management of software projects
- measure and improve the quality of software products in an efficient manner
- support process improvement
- provide a basis for automated support of process execution

Software *development* life cycles (SDLC)

software development life cycle process
software processes used to specify and transform software requirements into a deliverable software product

— ISO/IEC TR 19759:2015 Software Engineering Body of Knowledge (SWEBOK)

history

1970s

- Structured programming since 1969
- Cap Gemini SDM, originally from PANDATA, the first English translation was published in 1974. SDM stands for System Development Methodology

1980s

- Structured systems analysis and design method (SSADM) from 1980 onwards
- Information Requirement Analysis/Soft systems methodology

1990s

- Object-oriented programming (OOP) developed in the early 1960s, and became a dominant programming approach during the mid-1990s
- Rapid application development (RAD), since 1991
- Dynamic systems development method (DSDM), since 1994
- Scrum, since 1995
- Team software process, since 1998
- Rational Unified Process (RUP), maintained by IBM since 1998
- Extreme programming, since 1999

2000s

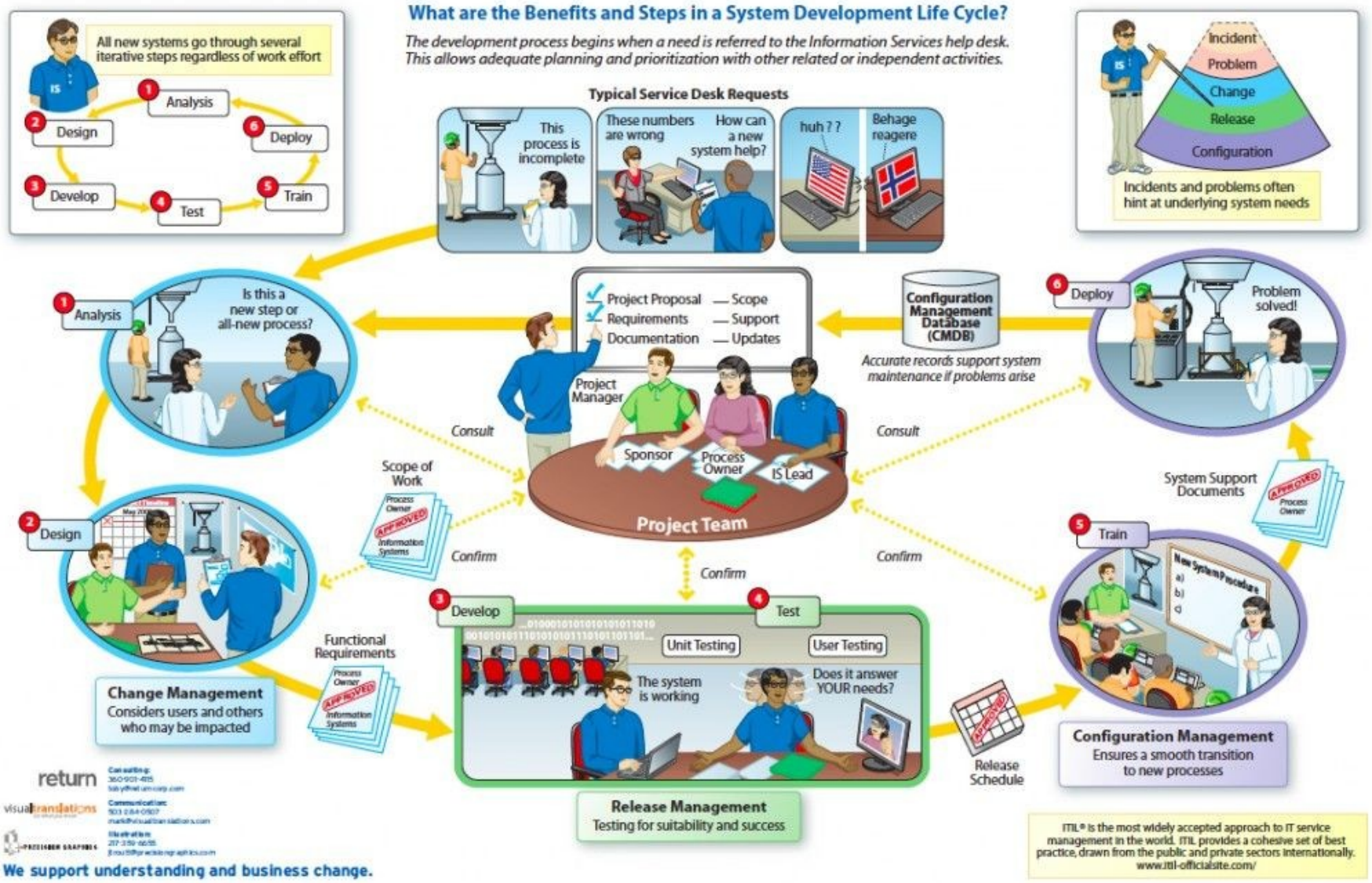
- Agile Unified Process (AUP) maintained since 2005 by Scott Ambler
- Disciplined agile delivery (DAD) Supersedes AUP

2010s

- Scaled Agile Framework (SAFe)
- Large-Scale Scrum (LeSS)
- DevOps

source https://en.wikipedia.org/wiki/Software_development_process

Example of a software development life cycle



Analysis vs. Design
What vs. How

During Analysis

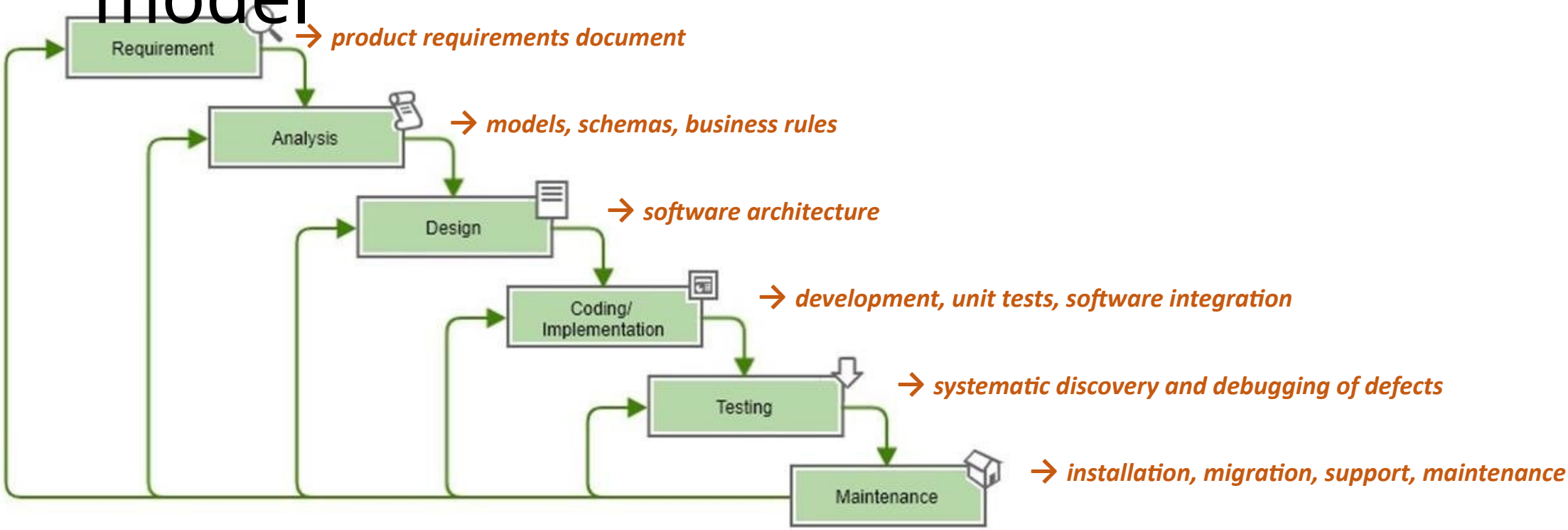
- To know about the application domain and the requirements
- Development of a coarse-grained model to show where responsibilities are, and how objects interact
- Models show a message being passed, but no worry too much about the contents of each message

During Design

- To know how the software should work
- Development of fine-grained models to show exactly what will happen when the system runs

The waterfall development life cycle model

W.W. Royce, 1970



Many variants of this model

☺ Well-documented and well structured

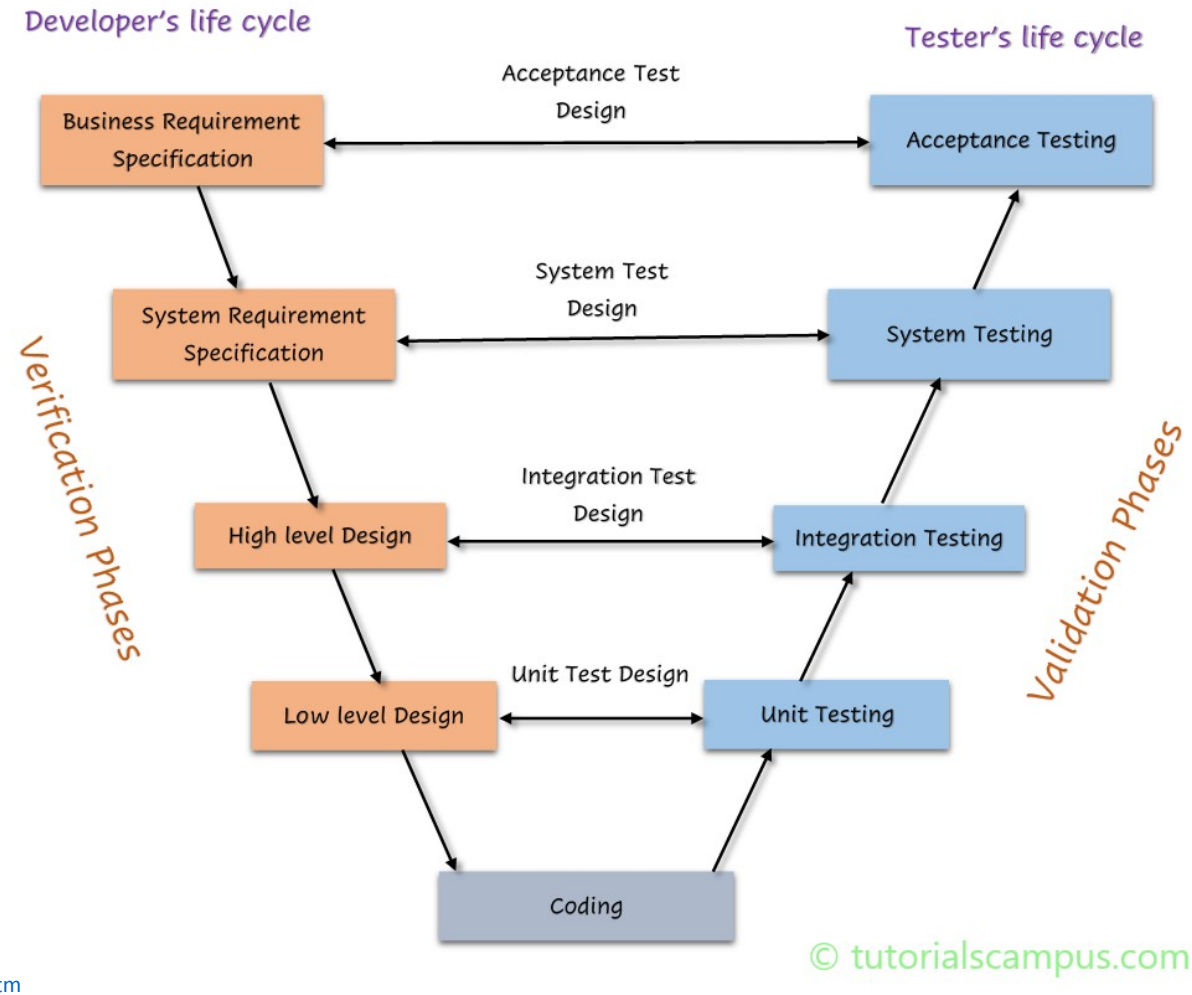
☺ Easy to maintain

☹ No prototype, late feedback to customer

☹ Major project risks, e.g. Implementation technology, are faced at the end of the project

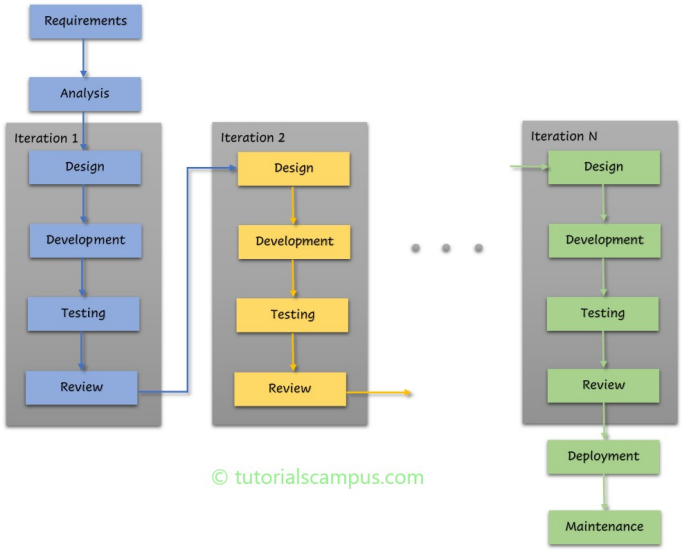
1980s

The V-model life cycle model

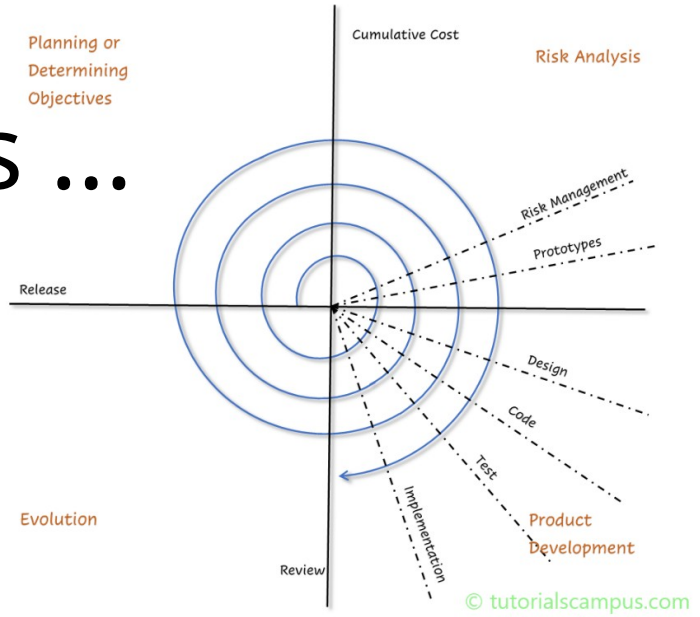


source: https://www.tutorialscampus.com/sdlc/v-model.htm

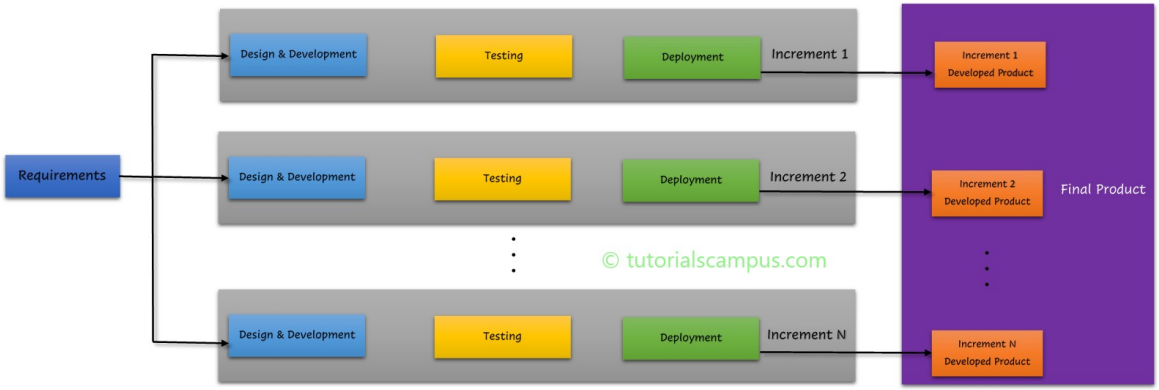
Many more SDLC models ...



iterative SDLC model



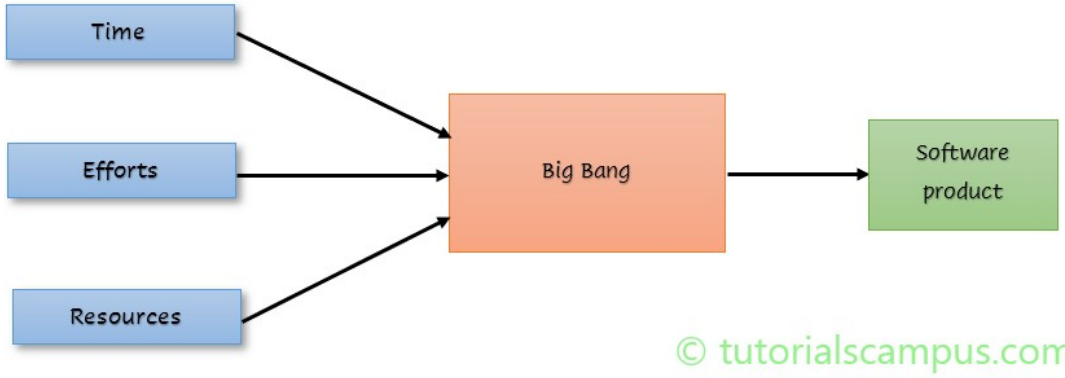
spiral SDLC model



incremental SDLC model

source: https://www.tutorialscampus.com/sdlc/

Simplest SDLC models …



Big Bang methodology

One main rule:

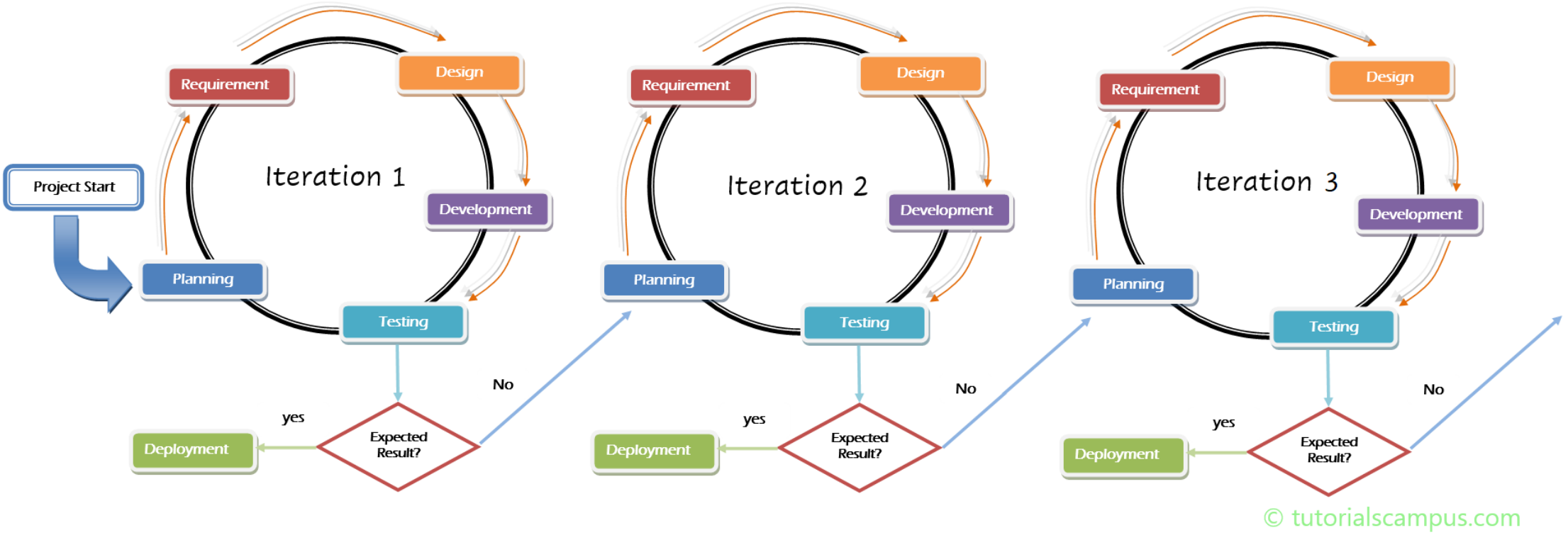
always resolve the most important issue first

Chaos model

source: https://www.tutorialscampus.com/sdlc/

1990s

Agile methods **(not a SDLC model!)**



Many Agile SDLC models

Lightweight methods; short, iterative development cycles; self-organizing teams; simpler designs; code refactoring; test-driven development; frequent customer involvement; create demonstrable working product with each development cycle

source: https://www.tutorialscampus.com/sdlc/agile-model.htm

2001

4 values

source: https://agilemanifesto.org/

2001

Principles behind the Agile Manifesto

We follow these 12 principles:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

source: https://agilemanifesto.org/

4 values and 12 principles

Agile vs « traditional » methods

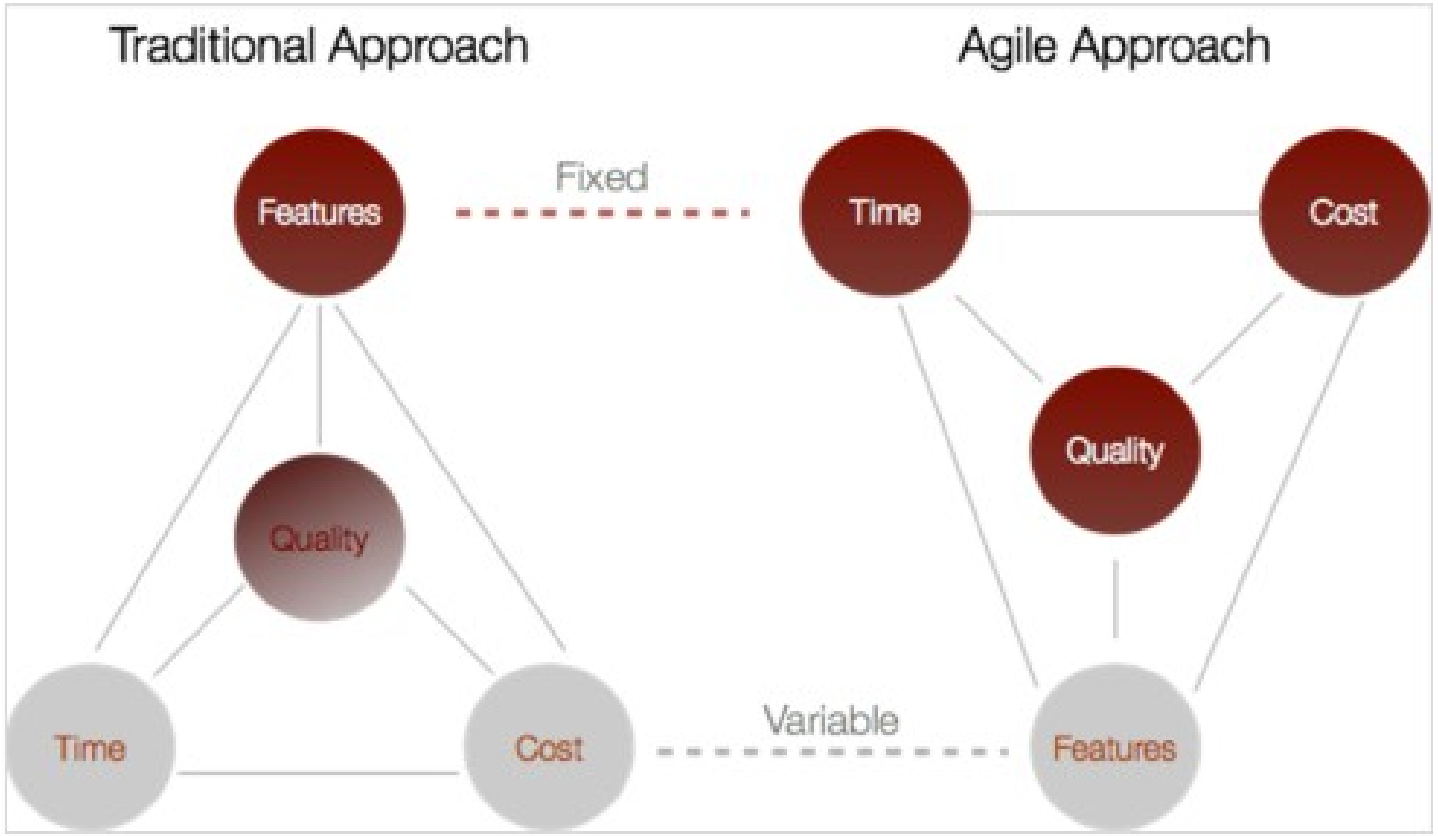


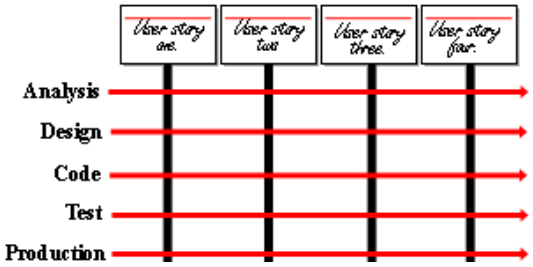
Figure 1. Comparison of the project triangles for traditional and Agile approaches (*source:* Awad, 2012; Beck et al., 2001)

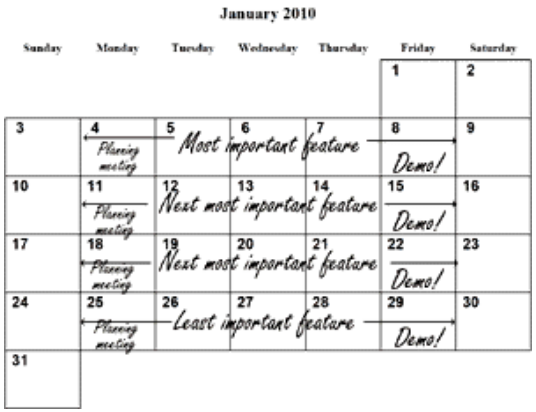
Agile development methods

- Dynamic System Development Methodology and RAD (www.dsdm.org, 1995)
- Scrum (Sutherland and Schwaber, 1995)
- XP - eXtreme Programming (Beck, 1999)
- Feature Driven Development (DeLuca, 1999)
- Adaptive Sw Development (Highsmith, 2000)
- Lean Development (Poppendieck, 2003)
- Crystal Clear (Cockburn, 2004)
- Agile Unified Process (Ambler, 2005)
- DevOps (

Extreme Programming (XP)

Manage Goals Instead of Activities





The values of XP

simplicity
communication
feedback
respect
courage

The Rules of Extreme Programming

Extreme Programming

Planning

- User stories are written.
- Release planning creates the release schedule.
- Make frequent small releases.
- The project is divided into iterations.
- Iteration planning starts each iteration.

Full Size Extreme Programming Project

Managing

- Give the team a dedicated open work space.
- Set a sustainable pace.
- A stand up meeting starts each day.
- The Project Velocity is measured.
- Move people around.
- Fix XP when it breaks.

Coding

- The customer is always available.
- Code must be written to agreed standards.
- Code the unit test first.
- All production code is pair programmed.
- Only one pair integrates code at a time.
- Integrate often.
- Set up a dedicated integration computer.
- Use collective ownership.

Designing

- Simplicity.
- Choose a system metaphor.
- Use CRC cards for design sessions.
- Create spike solutions to reduce risk.
- No functionality is added early.
- Refactor whenever and wherever possible.

Testing

- All code must have unit tests.
- All code must pass all unit tests before it can be released.
- When a bug is found tests are created.
- Acceptance tests are run often and the score is published.

Let's review the values of Extreme Programming (XP) next.

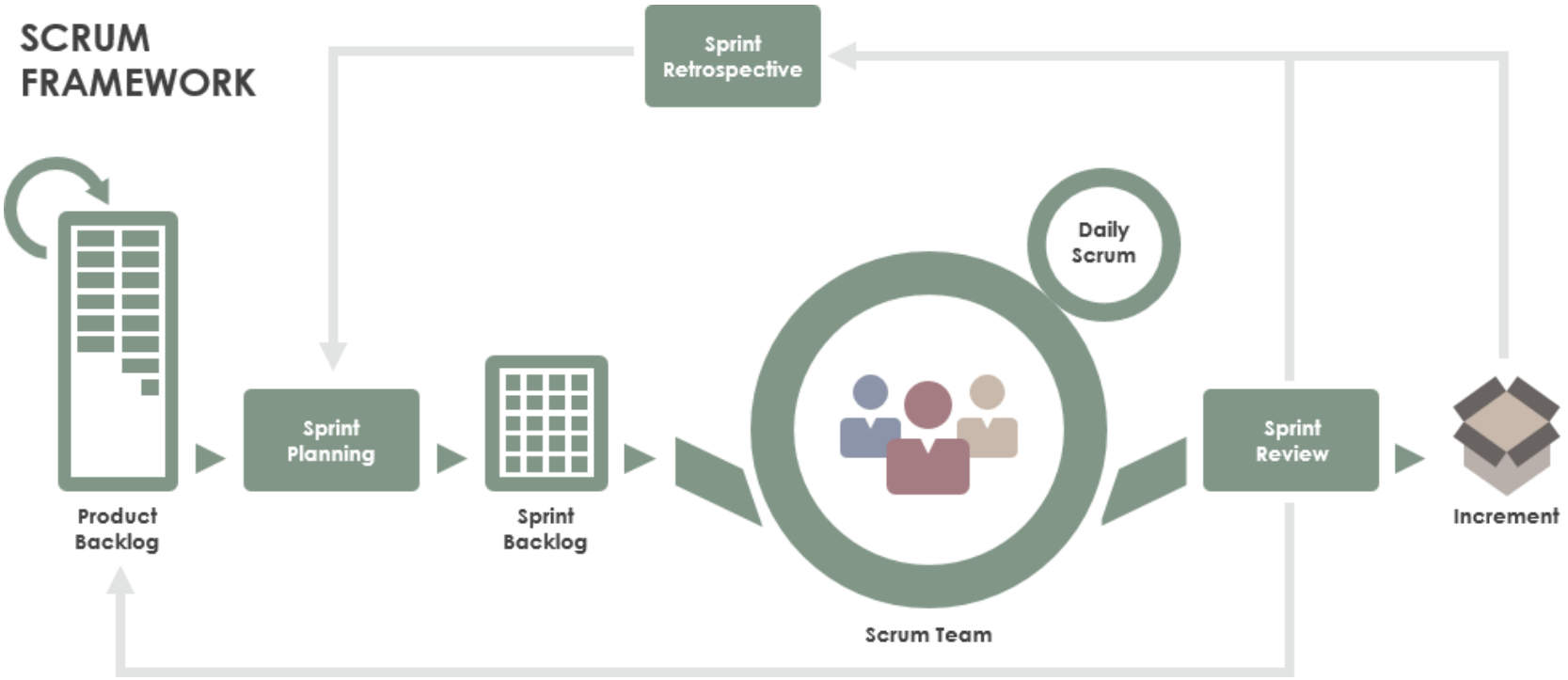
ExtremeProgramming.org home | XP Map | XP Values | Test framework | About the Author

source: http://www.extremeprogramming.org

Scrum - Framework

A **Scrum Master** fosters an environment where

1. A **Product Owner** orders the work for a complex problem into a **Product Backlog**.
2. The **Scrum Team** turns a selection of the work into an **Increment** of value during a **Sprint**.
3. The Scrum Team and its stakeholders **inspect the results and adjust** for the next Sprint.
4. *Repeat*



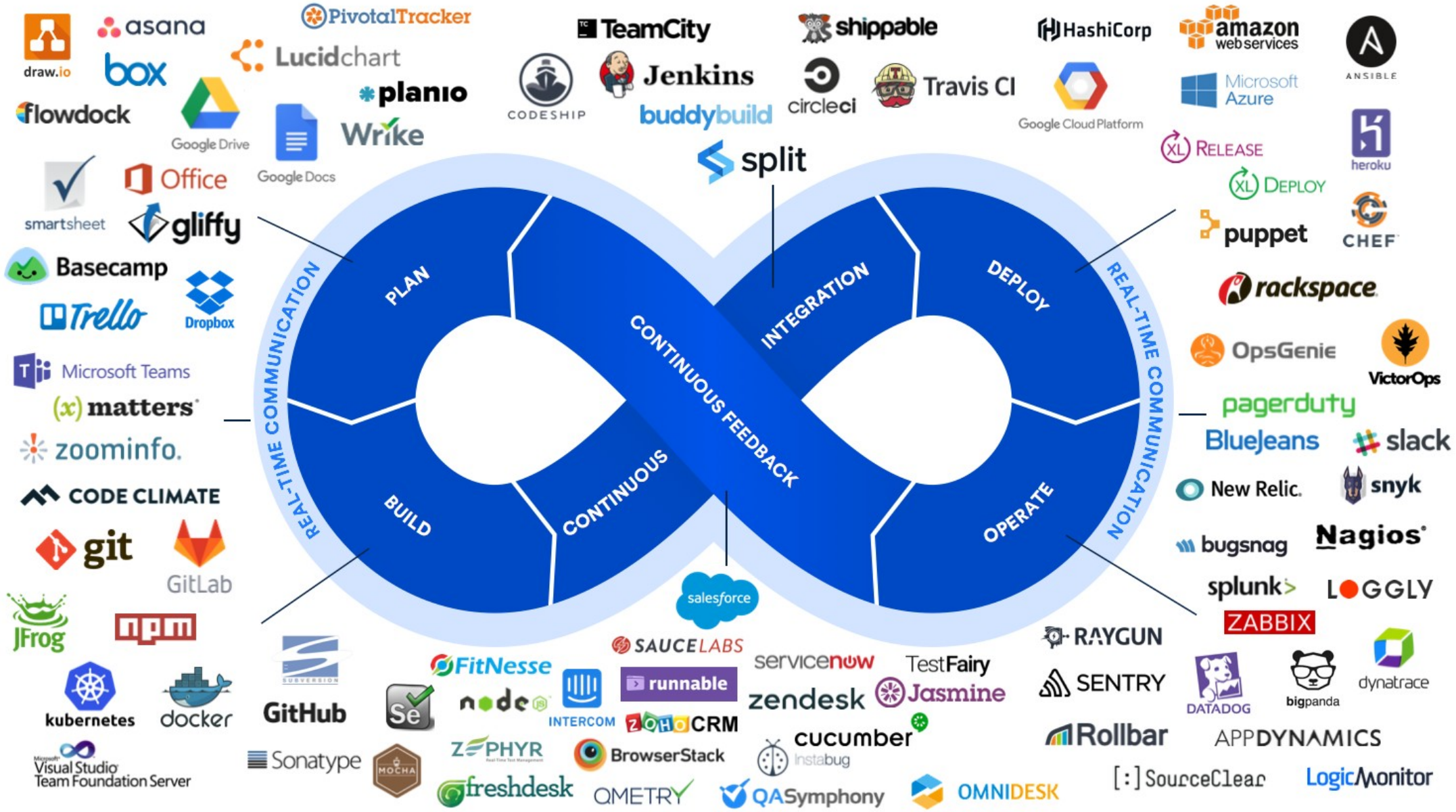
https://www.scrum.org/ (with videos, guide)
https://www.scrumguides.org/index.html

DevOps

DevOps is a set of practices that combines software development (Dev) and IT operations (Ops). It aims to shorten the systems development life cycle and provide continuous delivery with high software quality. DevOps is complementary with Agile software development; several DevOps aspects came from the Agile methodology.

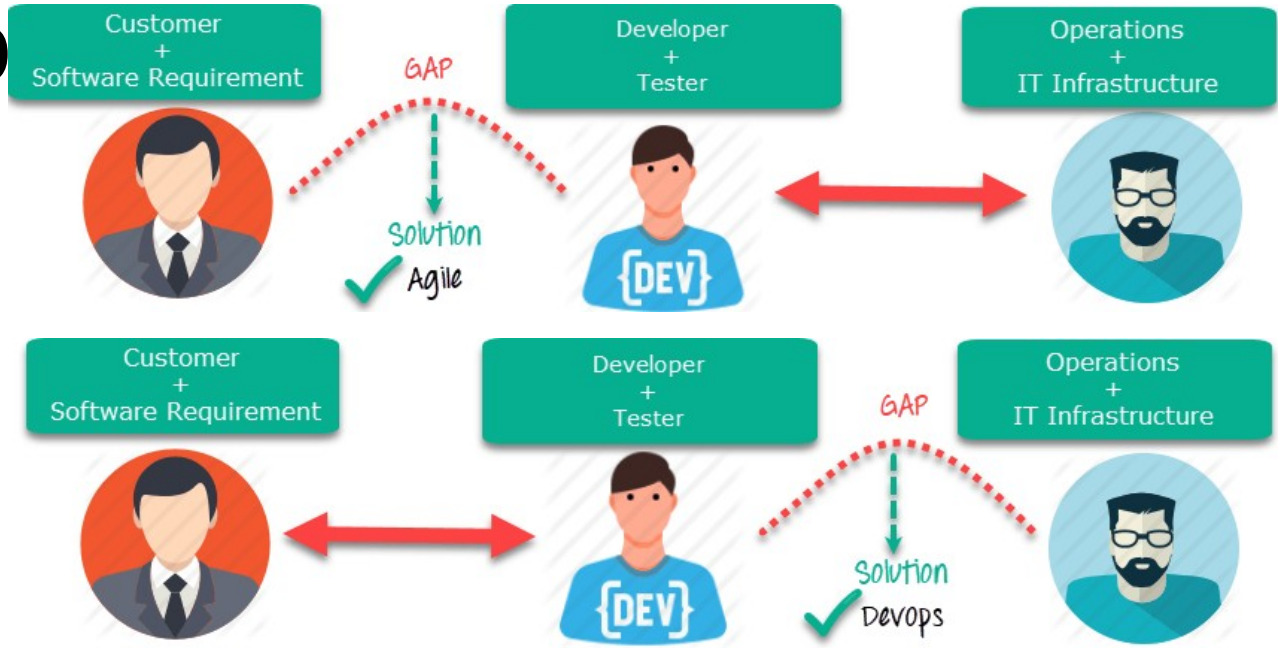
— Contributors, Wikipedia https://en.wikipedia.org/wiki/DevOps

⚠ There are many variants



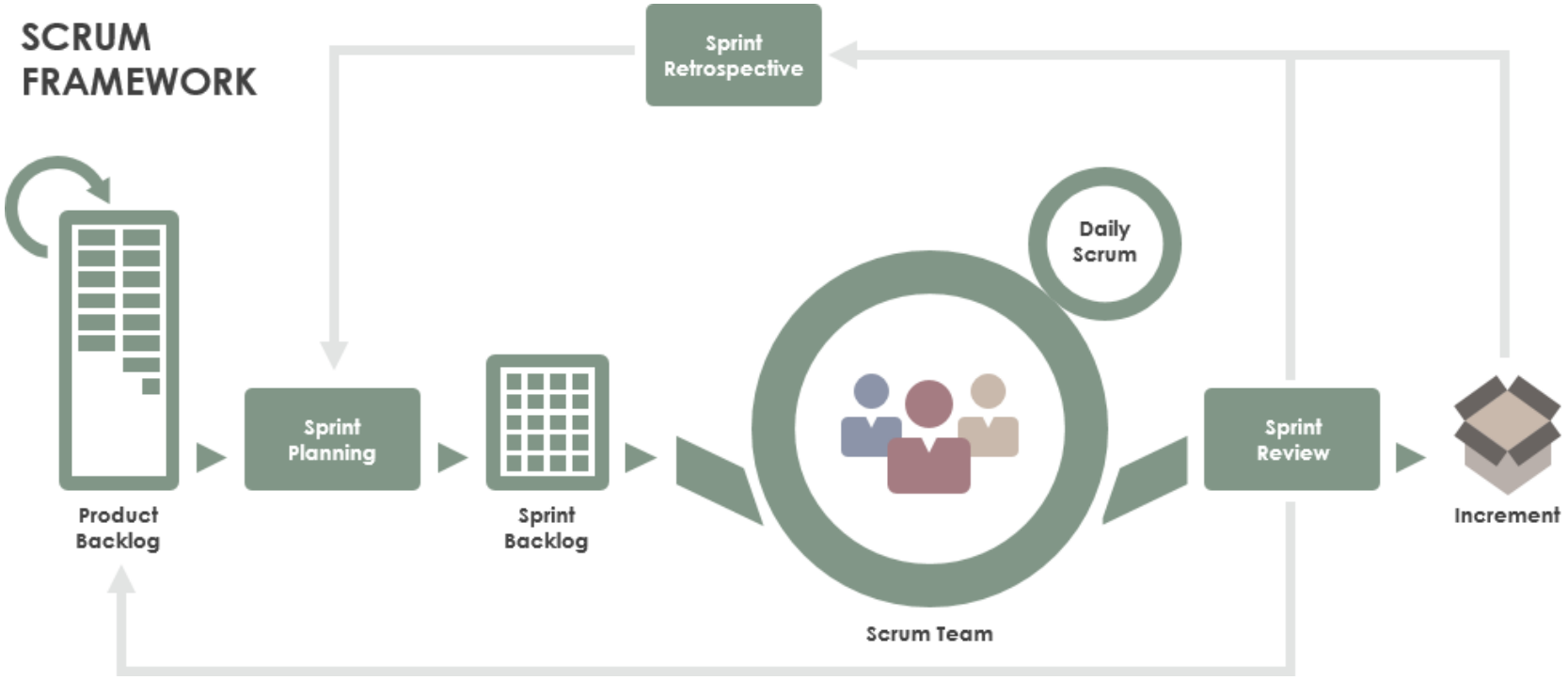
https://s32860.pcdn.co/wp-content/uploads/2019/11/devops-hero-1-87966cfbc9c5713ae047551c7b22985c.png

Agile vs DevOp



- DevOps is a practice of bringing development and operations teams together whereas Agile is an iterative approach that focuses on collaboration, customer feedback and small rapid releases.
- DevOps focuses on constant testing and delivery while the Agile process focuses on constant changes.
- DevOps requires relatively a large team while Agile requires a small team.
- DevOps leverages both shifts left and right principles, on the other hand, Agile leverage shift-left principle.
- The target area of Agile is Software development whereas the Target area of DevOps is to give end-to-end business solutions and fast delivery.
- DevOps focuses more on operational and business readiness whereas Agile focuses on functional and non-function readiness.

Source: https://www.guru99.com/agile-vs-devops.html



Software Engineering

Part 5 – Focus on the Scrum methodology

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Scrum - roles

The Scrum Team consists of one **Scrum Master**, one **Product Owner**, and **Developers**

Developers are the people in the Scrum Team that are committed to creating any aspect of a usable Increment each Sprint.

- Create a plan for the Sprint, the Sprint Backlog;
- Instill quality by adhering to a Definition of Done;
- Adapt their plan each day toward the Sprint Goal; and,
- Hold each other accountable as professionals.

https://www.scrum.org/ (with videos, guide)
https://www.scrumguides.org/index.html

Scrum - roles

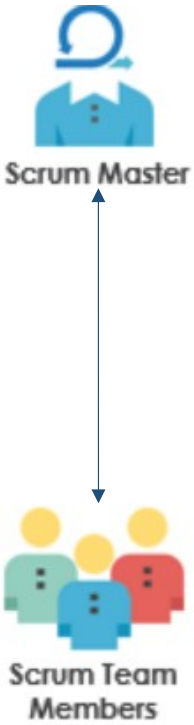
The Scrum Team consists of one **Scrum Master**, one **Product Owner**, and **Developers**

Product Owner is accountable for maximizing the value of the product resulting from the work of the Scrum Team. How this is done may vary widely across organizations, Scrum Teams, and individuals.

- Develop and explicitly communicate the Product Goal;
- Create and clearly communicate Product Backlog items;
- Order Product Backlog items; and,
- Ensure that the Product Backlog is transparent, visible and understood.

https://www.scrum.org/ (with videos, guide)
https://www.scrumguides.org/index.html

Scrum - roles



The Scrum Team consists of one **Scrum Master**, one **Product Owner**, and **Developers**

Scrum Master is accountable for establishing Scrum as defined in the Scrum Guide. They do this by helping everyone understand Scrum theory and practice, both within the Scrum Team and the organization.

The Scrum Master is accountable for the Scrum Team's effectiveness. They do this by enabling the Scrum Team to improve its practices, within the Scrum framework.

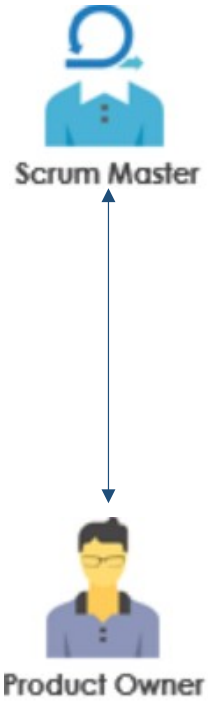
Scrum Masters are true leaders who serve the Scrum Team and the larger organization.

Serves the Scrum Team:

- Coaching the team members in self-management and cross-functionality;
- Helping the Scrum Team focus on creating high-value Increments that meet the Definition of Done;
- Causing the removal of impediments to the Scrum Team's progress; and,
- Ensuring that all Scrum events take place and are positive, productive, and kept within the timebox.

https://www.scrum.org/ (with videos, guide)
https://www.scrumguides.org/index.html

Scrum - roles



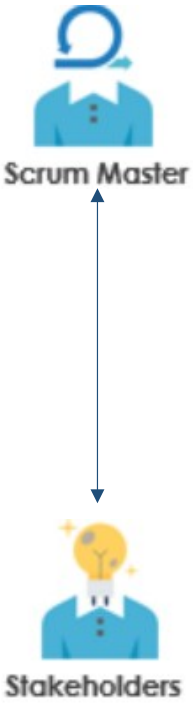
The Scrum Team consists of one **Scrum Master**, one **Product Owner**, and **Developers**

Scrum Master is accountable for establishing Scrum as defined in the Scrum Guide. They do this by helping everyone understand Scrum theory and practice, both within the Scrum Team and the organization.
The Scrum Master is accountable for the Scrum Team's effectiveness. They do this by enabling the Scrum Team to improve its practices, within the Scrum framework.
Scrum Masters are true leaders who serve the Scrum Team and the larger organization.
Serves the Product Owner:

- Helping find techniques for effective Product Goal definition and Product Backlog management;
- Helping the Scrum Team understand the need for clear and concise Product Backlog items;
- Helping establish empirical product planning for a complex environment; and,
- Facilitating stakeholder collaboration as requested or needed.

https://www.scrum.org/ (with videos, guide)
https://www.scrumguides.org/index.html

Scrum - roles



The Scrum Team consists of one **Scrum Master**, one **Product Owner**, and **Developers**

Scrum Master is accountable for establishing Scrum as defined in the Scrum Guide. They do this by helping everyone understand Scrum theory and practice, both within the Scrum Team and the organization.

The Scrum Master is accountable for the Scrum Team's effectiveness. They do this by enabling the Scrum Team to improve its practices, within the Scrum framework.

Scrum Masters are true leaders who serve the Scrum Team and the larger organization.

Serves the Organization:

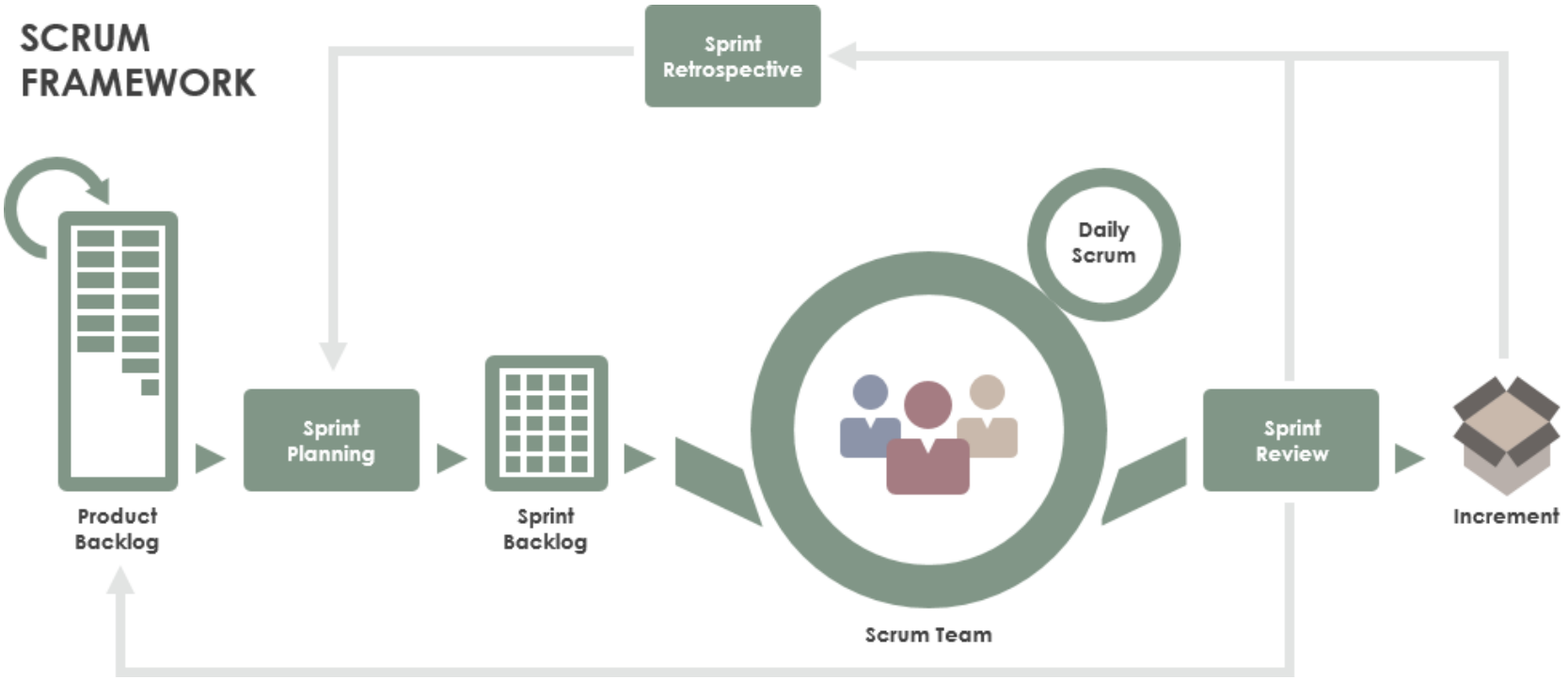
- Leading, training, and coaching the organization in its Scrum adoption;
- Planning and advising Scrum implementations within the organization;
- Helping employees and stakeholders understand and enact an empirical approach for complex work; and,
- Removing barriers between stakeholders and Scrum Teams

https://www.scrum.org/ (with videos, guide)
https://www.scrumguides.org/index.html

Scrum - Framework

A **Scrum Master** fosters an environment where

1. A **Product Owner** orders the work for a complex problem into a **Product Backlog**.
2. The **Scrum Team** turns a selection of the work into an **Increment** of value during a **Sprint**.
3. The Scrum Team and its stakeholders **inspect the results and adjust** for the next Sprint.
4. *Repeat*



https://www.scrum.org/ (with videos, guide)
https://www.scrumguides.org/index.html

Scrum – product backlog

The Product Backlog is an emergent, ordered list of what is needed to improve the product. It is the single source of work undertaken by the Scrum Team.

ToDo List

ID	Story	Estimation	Priority
7	As an unauthorized User I want to create a new account	3	1
1	As an unauthorized User I want to login	1	2
10	As an authorized User I want to logout	1	3
9	Create script to purge database	1	4
2	As an authorized User I want to see the list of items so that I can select one	2	5
4	As an authorized User I want to add a new item so that it appears in the list	5	6
3	As an authorized User I want to delete the selected item	2	7
5	As an authorized User I want to edit the selected item	5	8
6	As an authorized User I want to set a reminder for a selected item so that I am reminded when item is due	8	9
8	As an administrator I want to see the list of accounts on login	2	10
Total		**30**	

https://www.scrum.org/ (with videos, guide)
https://www.scrumguides.org/index.html

Example Product Backlog
Source: https://www.scrum-institute.org/The_Scrum_Product_Backlog.php

Scrum – sprint planning

Sprint Planning initiates the Sprint by laying out the work to be performed for the Sprint. This resulting plan is created by the collaborative work of the entire Scrum Team.

Topic One: Why is this Sprint valuable?

Product Owner proposes how to increase the value and utility of the product

Scrum Team collaborates to define **Sprint Goal**

Topic Two: What can be Done this Sprint?

Developers discuss with Product Owner and select items from the Product Backlog to include in the current Sprint

Refine items, estimate how much can be done in the Sprint timebox

Topic Three: How will the chosen work get done?

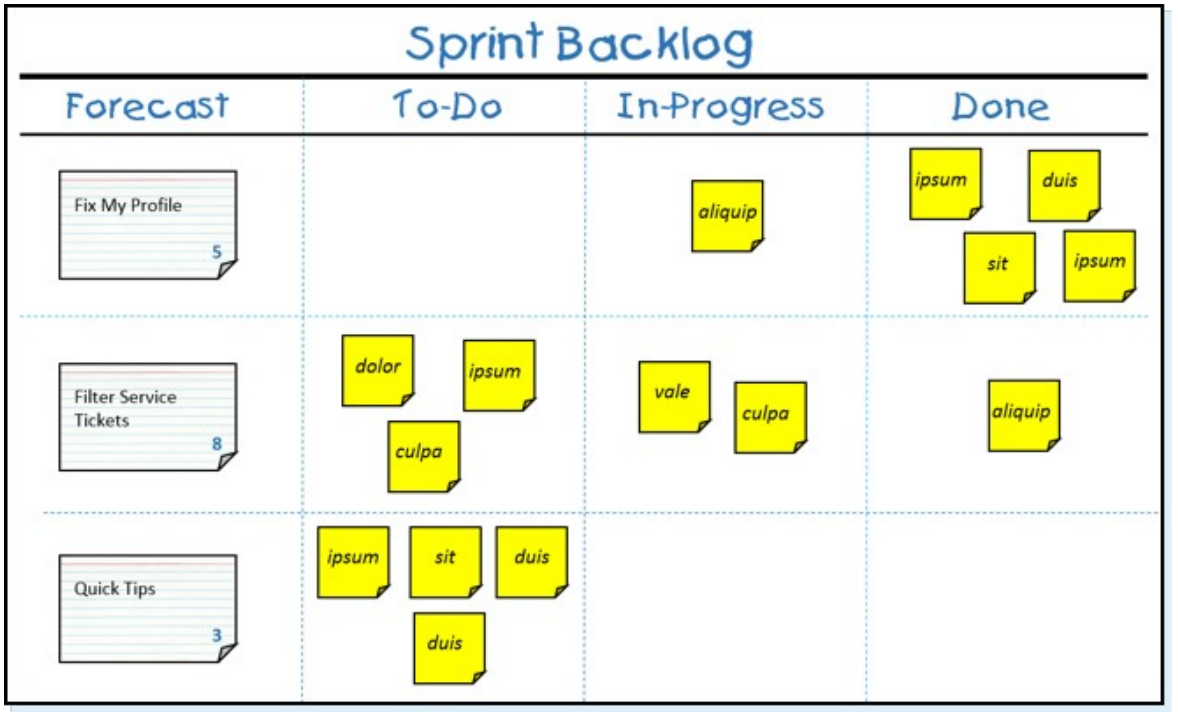
Developers **plan** the work. Decompose Product Backlog into smaller work items

Output: **Sprint Backlog**: Sprint Goal, Product Backlog selected for the Sprint, Plan for delivering them

https://www.scrum.org/ (with videos, guide)
https://www.scrumguides.org/index.html

Scrum – sprint backlog

The Sprint Backlog is composed of the Sprint Goal (why), the set of Product Backlog items selected for the Sprint (what), as well as an actionable plan for delivering the Increment (how)

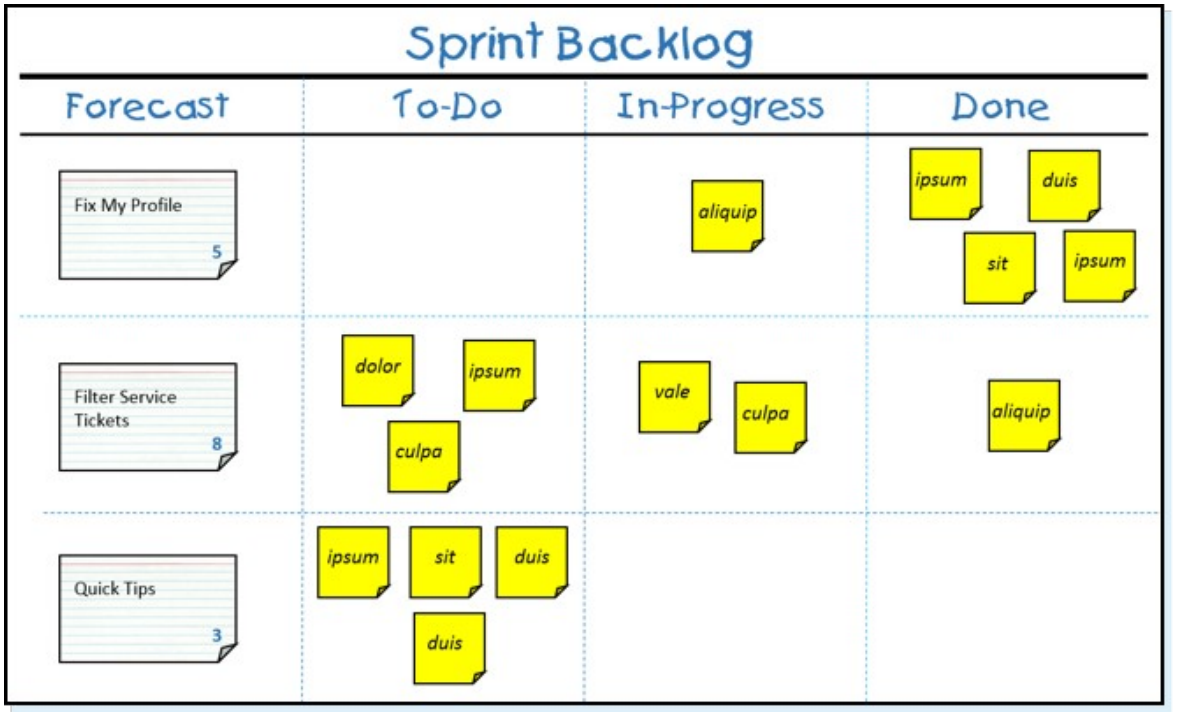


Example Sprint Backlog
Source: https://www.scrum.org/resources/what-is-a-sprint-backlog

https://www.scrum.org/ (with videos, guide)
https://www.scrumguides.org/index.html

Scrum – daily scrum

For developers only - The purpose of the Daily Scrum is to inspect progress toward the Sprint Goal and adapt the Sprint Backlog as necessary, adjusting the upcoming planned work.



Example Sprint Backlog
Source: https://www.scrum.org/resources/what-is-a-sprint-backlog

https://www.scrum.org/ (with videos, guide)
https://www.scrumguides.org/index.html

Scrum – sprint review

inspect the outcome of the Sprint and determine future adaptations. The Scrum Team presents the results of their work to key stakeholders and progress toward the Product Goal is discussed.

During the event, the Scrum Team and stakeholders review what was accomplished in the Sprint and what has changed in their environment. Based on this information, attendees collaborate on what to do next. The Product Backlog may also be adjusted to meet new opportunities. The Sprint Review is a working session and the Scrum Team should avoid limiting it to a presentation.

https://startinfinity.com/product-management-framework/scrum-sprint/sprint-review-vs-sprint-retrospective

https://www.scrum.org/ (with videos, guide)
https://www.scrumguides.org/index.html

Scrum – Increment and Done

An Increment is a concrete stepping stone toward the Product Goal. Each Increment is additive to all prior Increments and thoroughly verified, ensuring that all Increments work together. In order to provide value, the Increment must be usable.

The Definition of Done is a formal description of the state of the Increment when it meets the quality measures required for the product

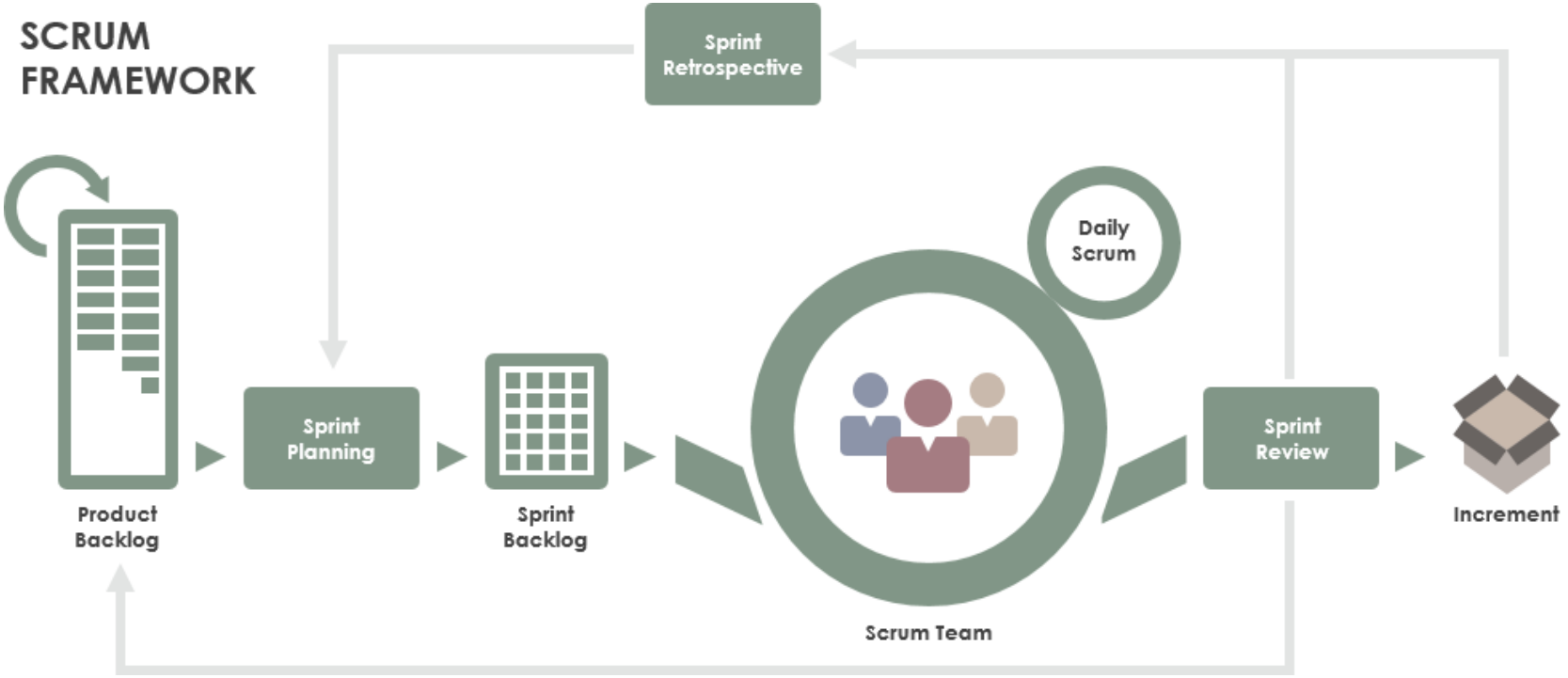
Scrum Foundations: Increment and Done (15) source: https://youtu.be/0MTWysPmaoU

https://www.scrum.org/ (with videos, guide)
https://www.scrumguides.org/index.html

Scrum – Sprint retrospective

Sprint Retrospective

The purpose of the Sprint Retrospective is to plan ways to increase quality and effectiveness.
The Scrum Team inspects how the last Sprint went with regards to individuals, interactions, processes, tools, and their Definition of Done.
The Scrum Team identifies the most helpful changes to improve its effectiveness.



https://www.scrum.org/ (with videos, guide)
https://www.scrumguides.org/index.html

Scrum - values

https://www.scrum.org/ (with videos, guide)
https://www.scrumguides.org/index.html

Software Engineering

Part 6 – The Unified Modeling Language

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

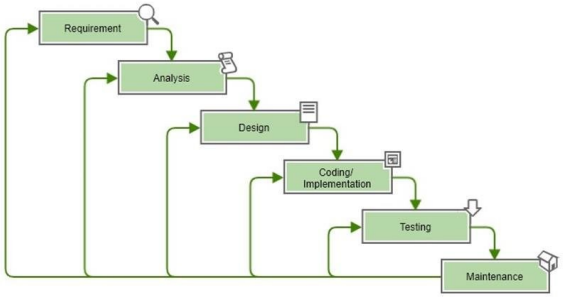
Software Engineering

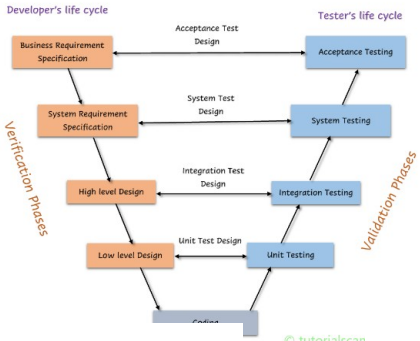
Part 6 – The Unified Modeling Language

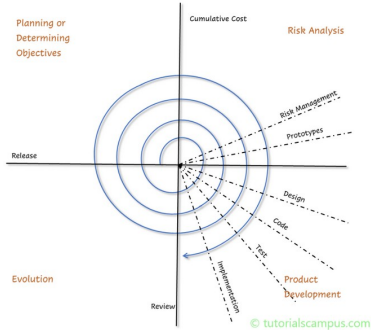
6.1 Introduction

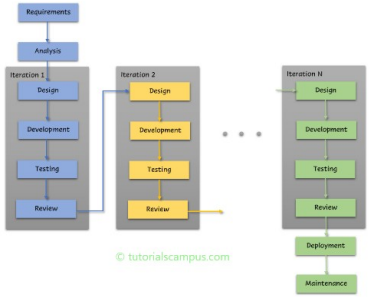
ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

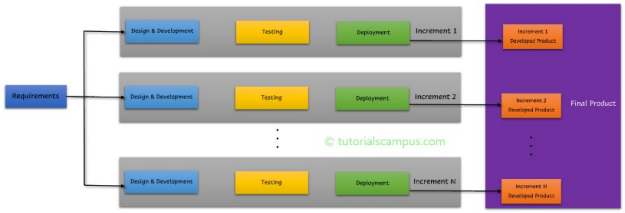
Software Development Methods

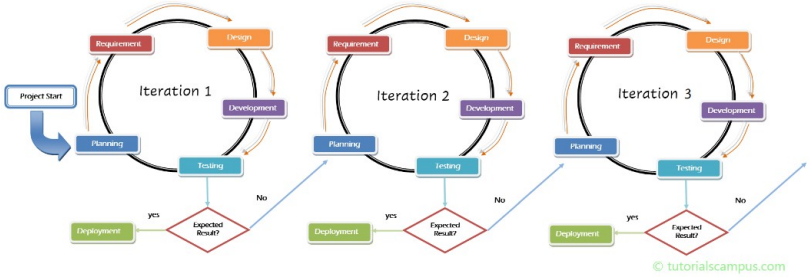


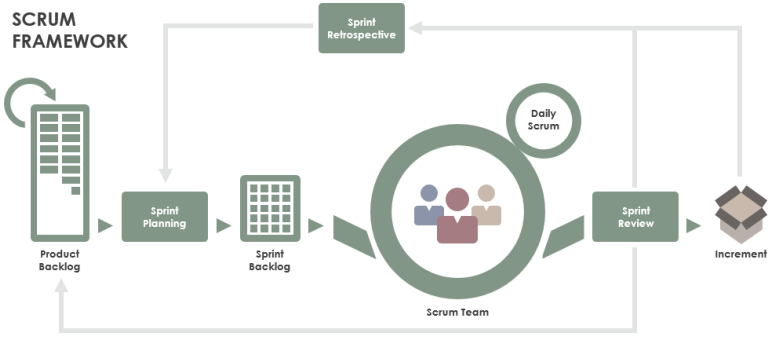












Software Development Methods

- *processes* that distinguish development stages in the software life cycle. Should:
  - be modular, reduce complexity, reuseable, at the right level of abstraction

- Using a *representation formalism* that facilitates communication, organization and verification

- Production of a set of *artifacts* that facilitate design feedback and application evolution
  - documents, models, prototypes

Existing software Development Methods

- Hierarchical functional methods
  - Data-Flow/SADT/SA-SD, Structure-Chart, ...
- Data oriented methods
  - Entité-Relation, MERISE, ...
- Behaviour oriented methods
  - SA-RT, Petri Net, ...
- Object oriented methods
  - OMT, OOA, Classe-Relation, OOD, ...

Object-oriented SD methods

- Statement:
  - at the beginning of the 90's, there are about 50 object oriented methods,
  - linked only by a consensus around common ideas (object, class, subsystems, ...)
  - BUT each with its own notation,
  - WITHOUT being able to fulfill all the needs and to correctly model the various fields of application.
- **Definition of a single common language**
  - usable by any object method,
  - in all phases of the life cycle,
  - compatible with current production techniques.

    → **UML**
- **Definition a common unified development process**

  → Unified Process (obsolete, use Scrum or other more recent processes)

UML (Unified Modeling Language)

- Based on:
  - *OMT* notations (J. Rumbaugh) for the analysis and design of data-based information systems
  - *G. Booch's* method notations for the design and implementation phases
  - *OOSE* notations (I. Jacobson) for requirement analysis through "use cases".
- Proposes:
  - Standardized development artifacts (models, notation, diagrams) WITHOUT standardizing the development process,
- Important role played by RATIONAL and OMG (http://www.omg.org/)

UML : Evolution

Latest version

Revision Task Force, Jul 2005

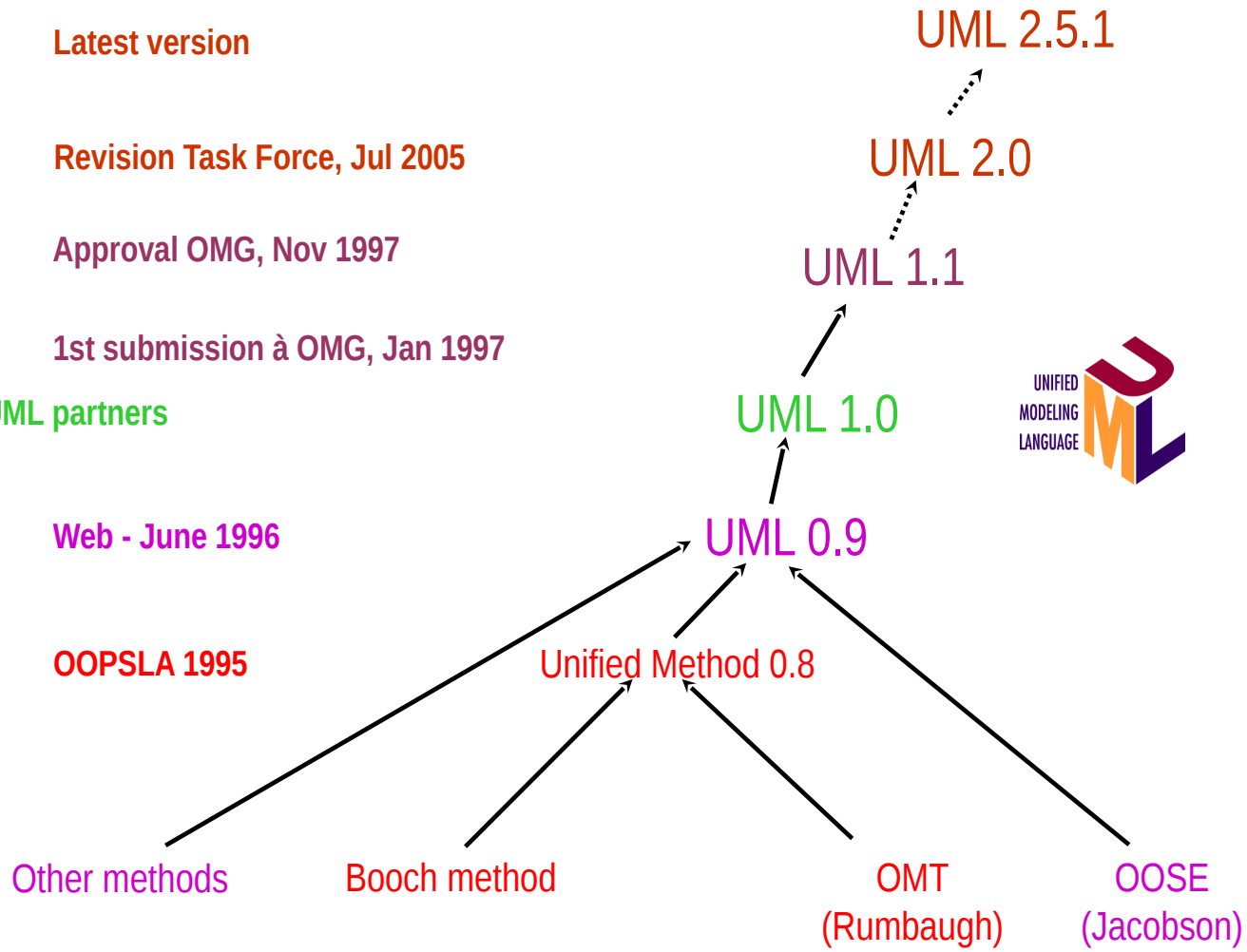
Approval OMG, Nov 1997

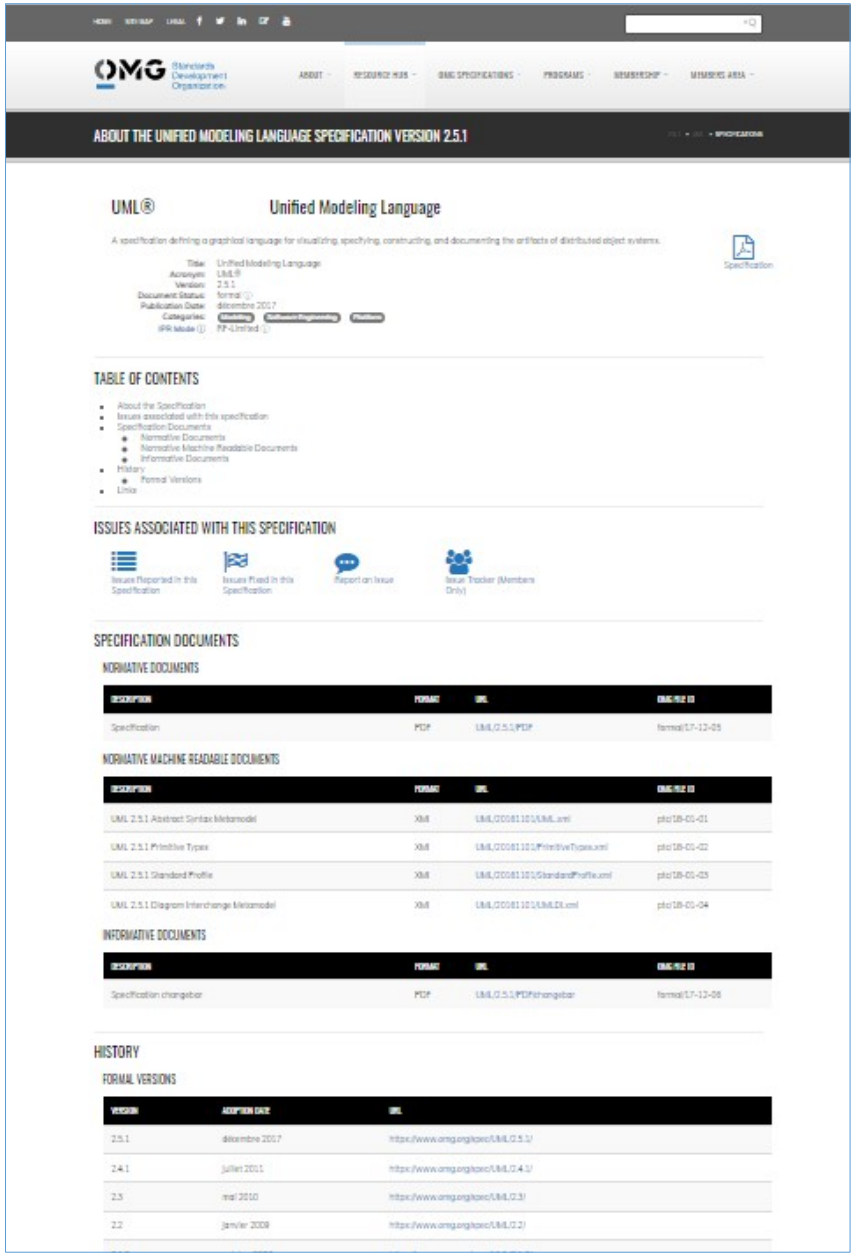
1st submission à OMG, Jan 1997

UML partners

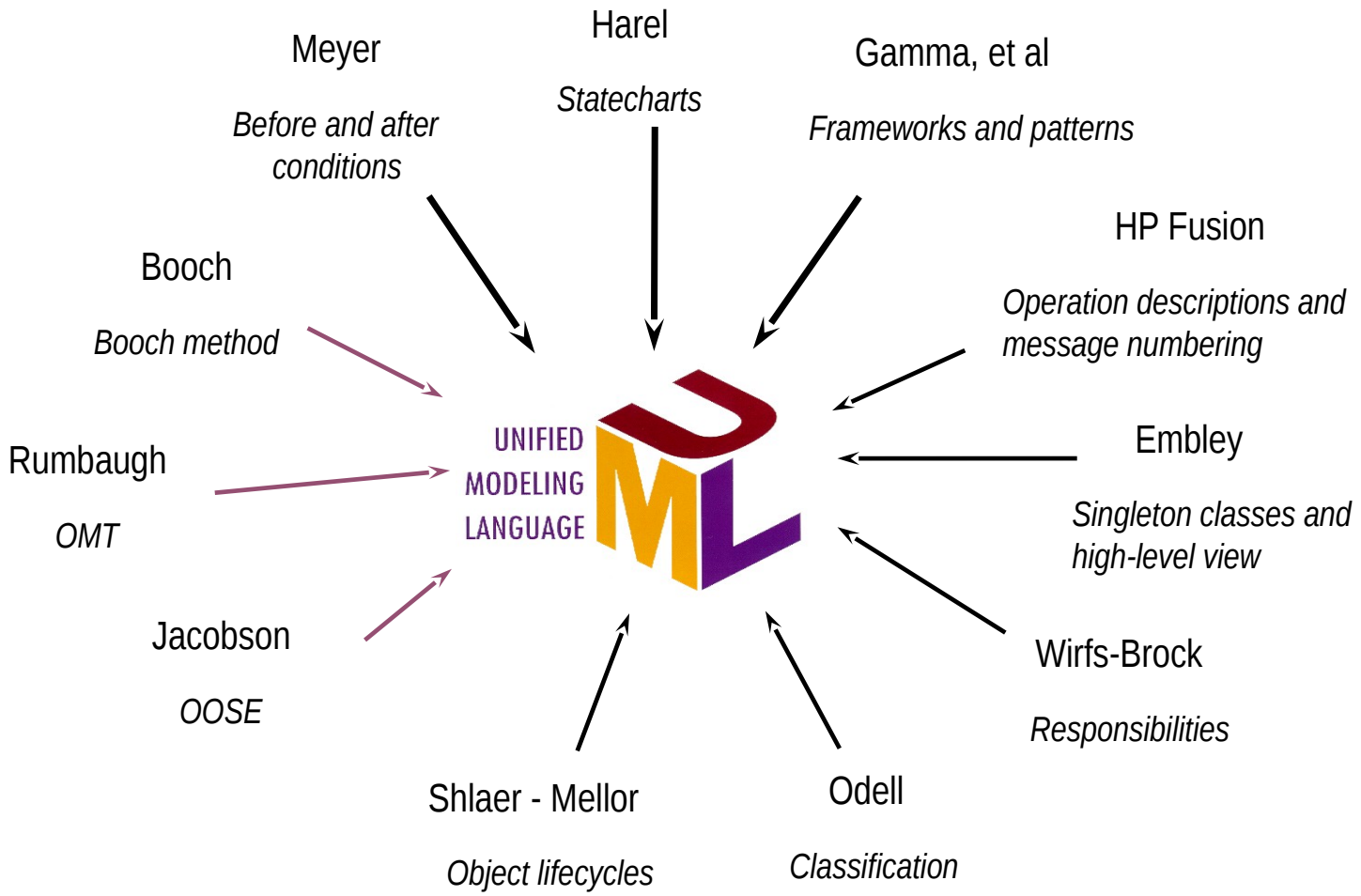
Web - June 1996

OOPSLA 1995

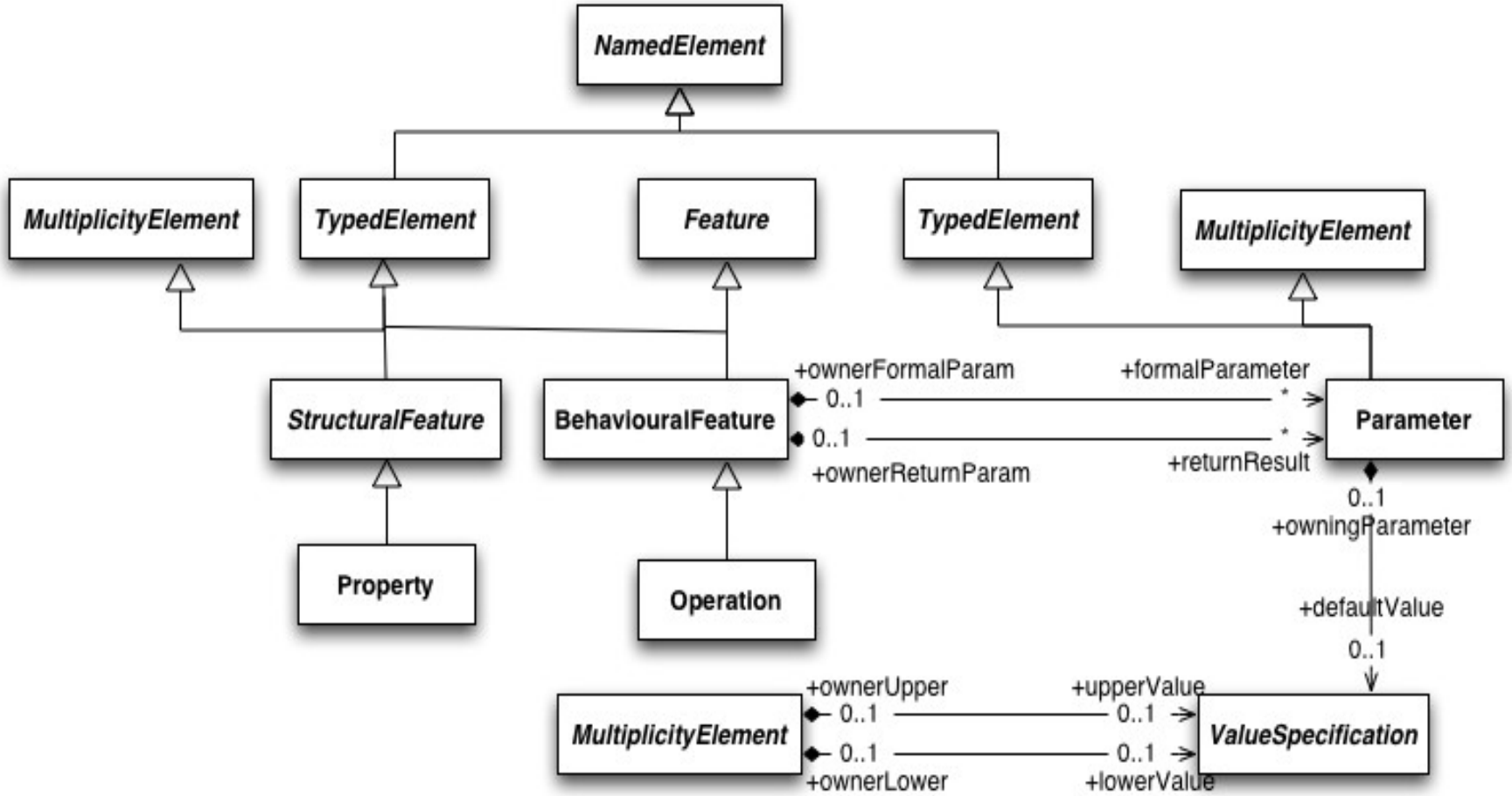




Contributions to UML 1.X

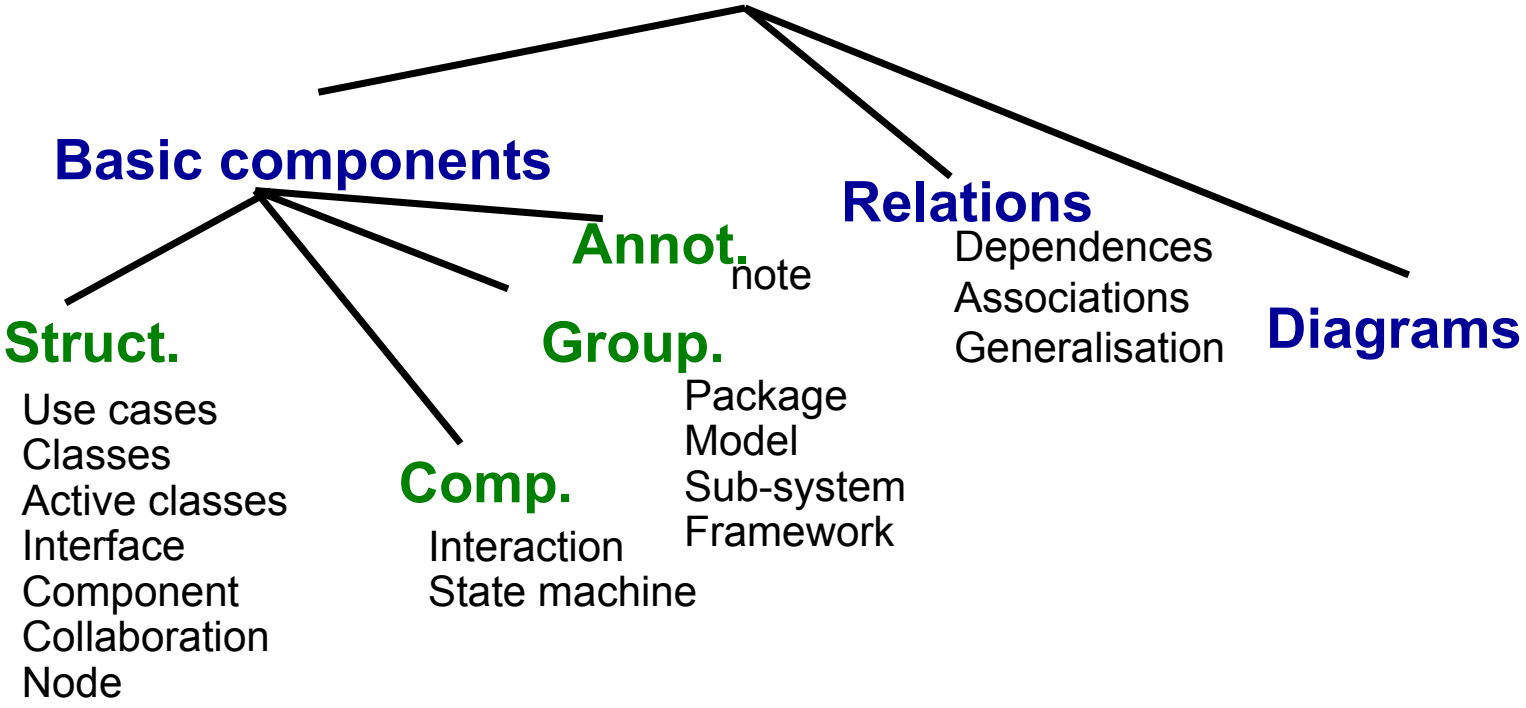


UML Meta-Model



Based on Martin Fowler UML Distilled and Viviane Jonckers OOSD-UML course

UML Vocabulary



UML Diagrams

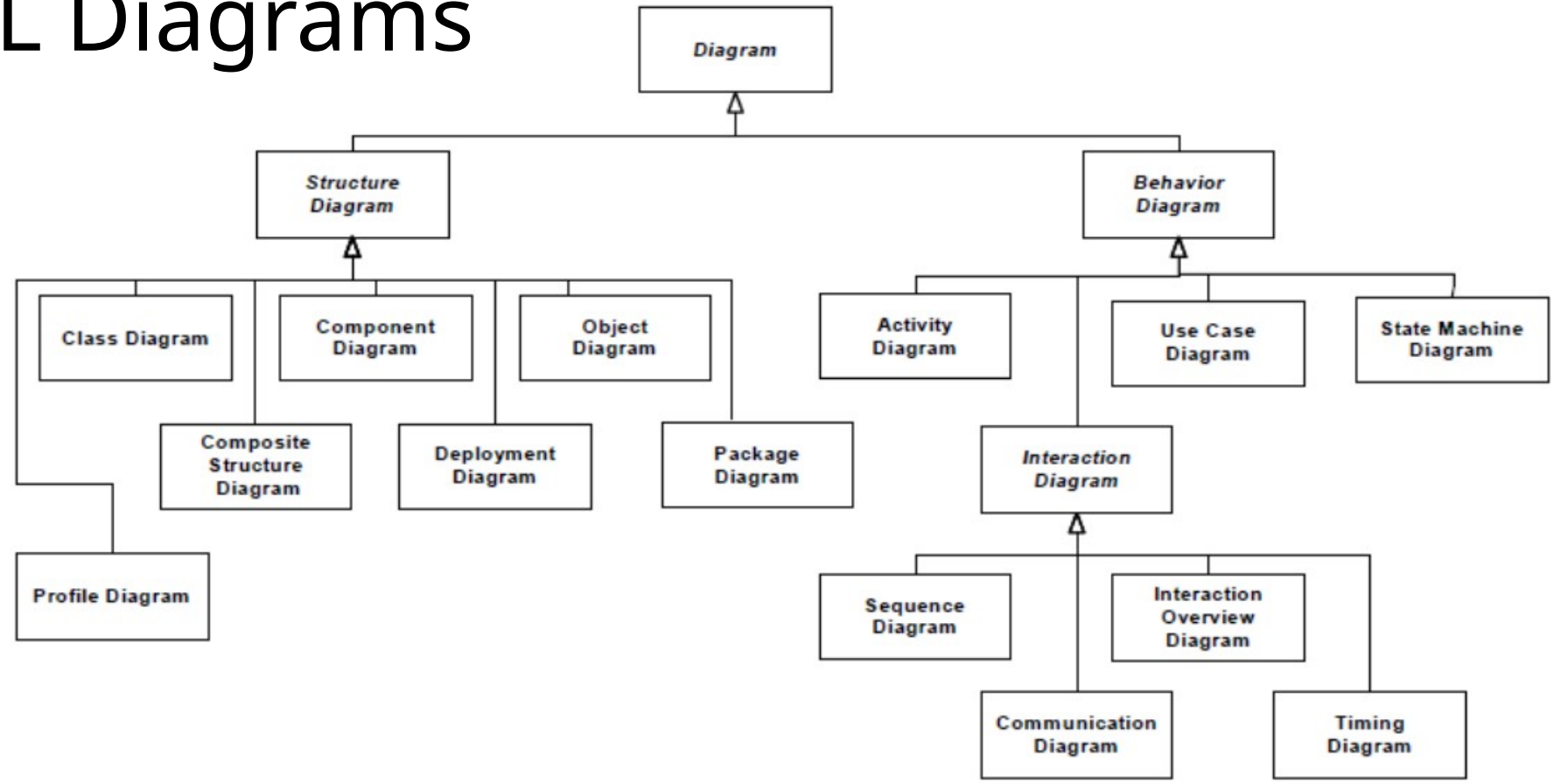
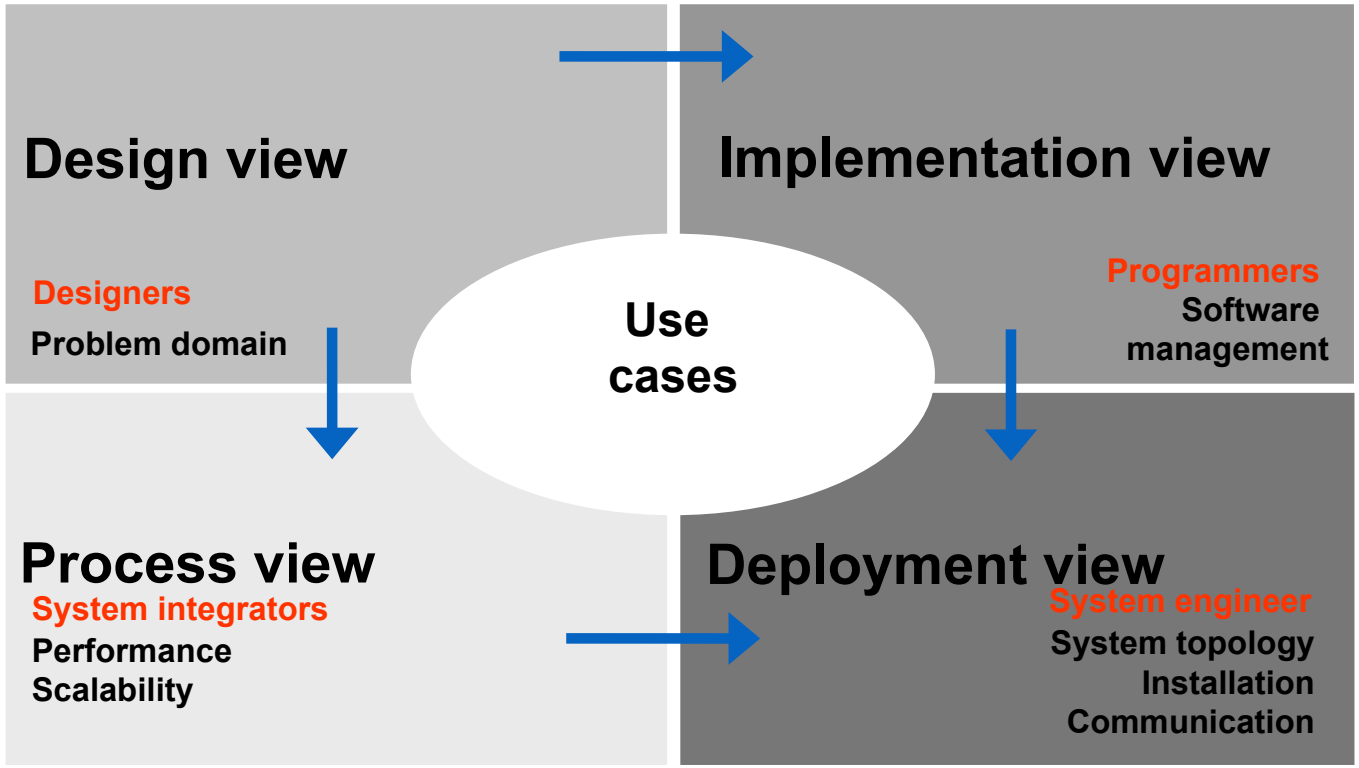


Figure A.5 The taxonomy of structure and behavior diagrams

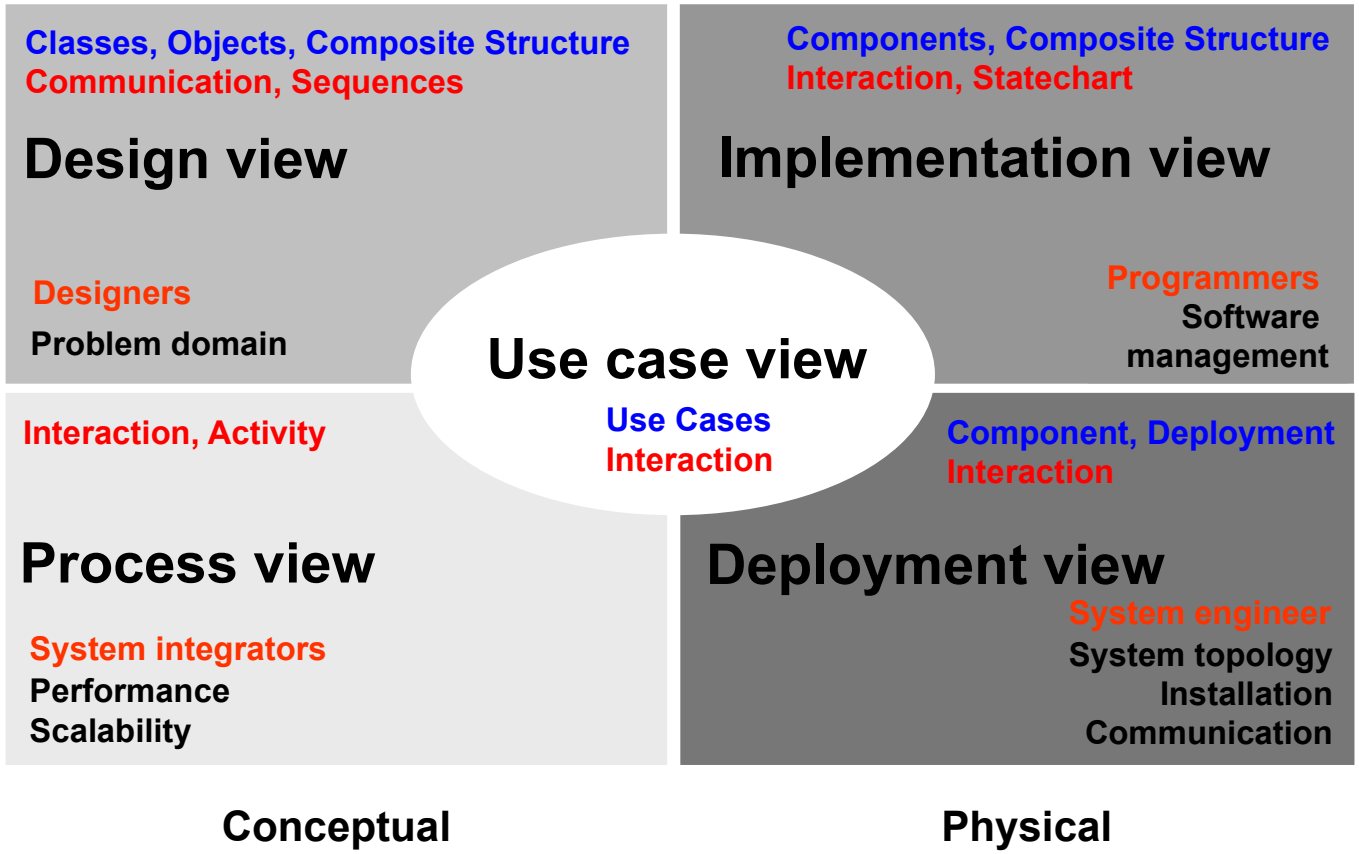
UML Specification, v2.5.1, p727 https://www.omg.org/spec/UML/2.5.1/PDF

Possibility of representing the same diagram at different levels of detail

Views on the Software



Diagrams within Views on the Software



Rules of thumb

- Nearly everything in UML is optional
- UML provides a language to capture information that varies greatly depending on the domain of the problem.
- Parts of UML either don't apply to your particular problem or may not lend anything to the particular view you are trying to convey.
- You don't need to use every part of UML in every model you create.
- You don't need to use every allowable symbol for a diagram type in every diagram you create.
- Show only what helps clarify the message you are trying to

Pointers

- The UML Specification https://www.omg.org/spec/UML/About-UML/
- https://www.uml-diagrams.org/

Software Engineering

Part 6 – The Unified Modeling Language

6.2 – diagrams we'll use for the analysis phase

6.2.1 – Use case diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

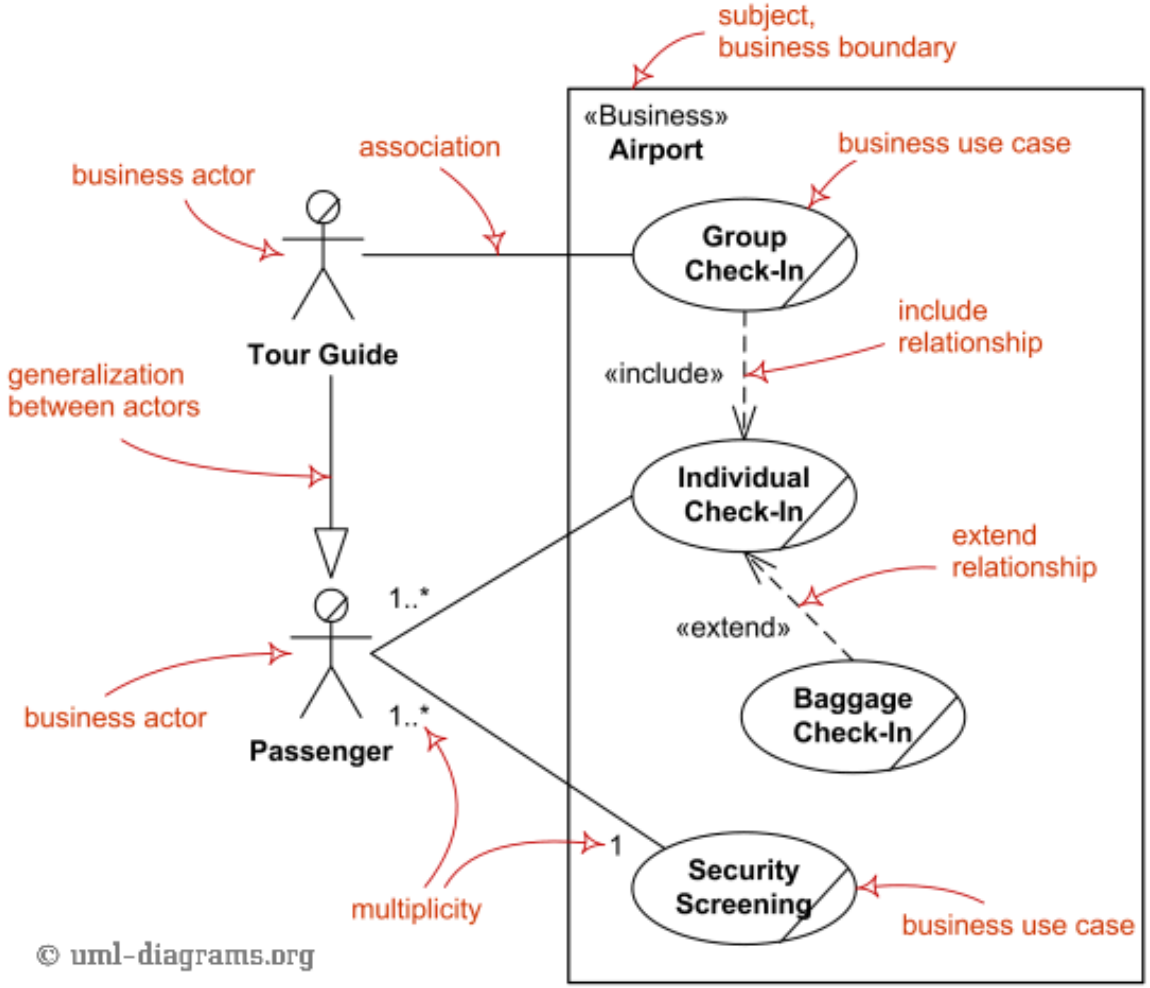
Use case diagrams

*Describes a set of actions (**use cases**) that some system or systems (**subject**) should or can perform in collaboration with one or more external users of the system (**actors**) to provide some observable and valuable results to the actors or other stakeholders of the system(s).*

terminology: **use case**, **actor**, **subject**, **extend**, include, **association**.

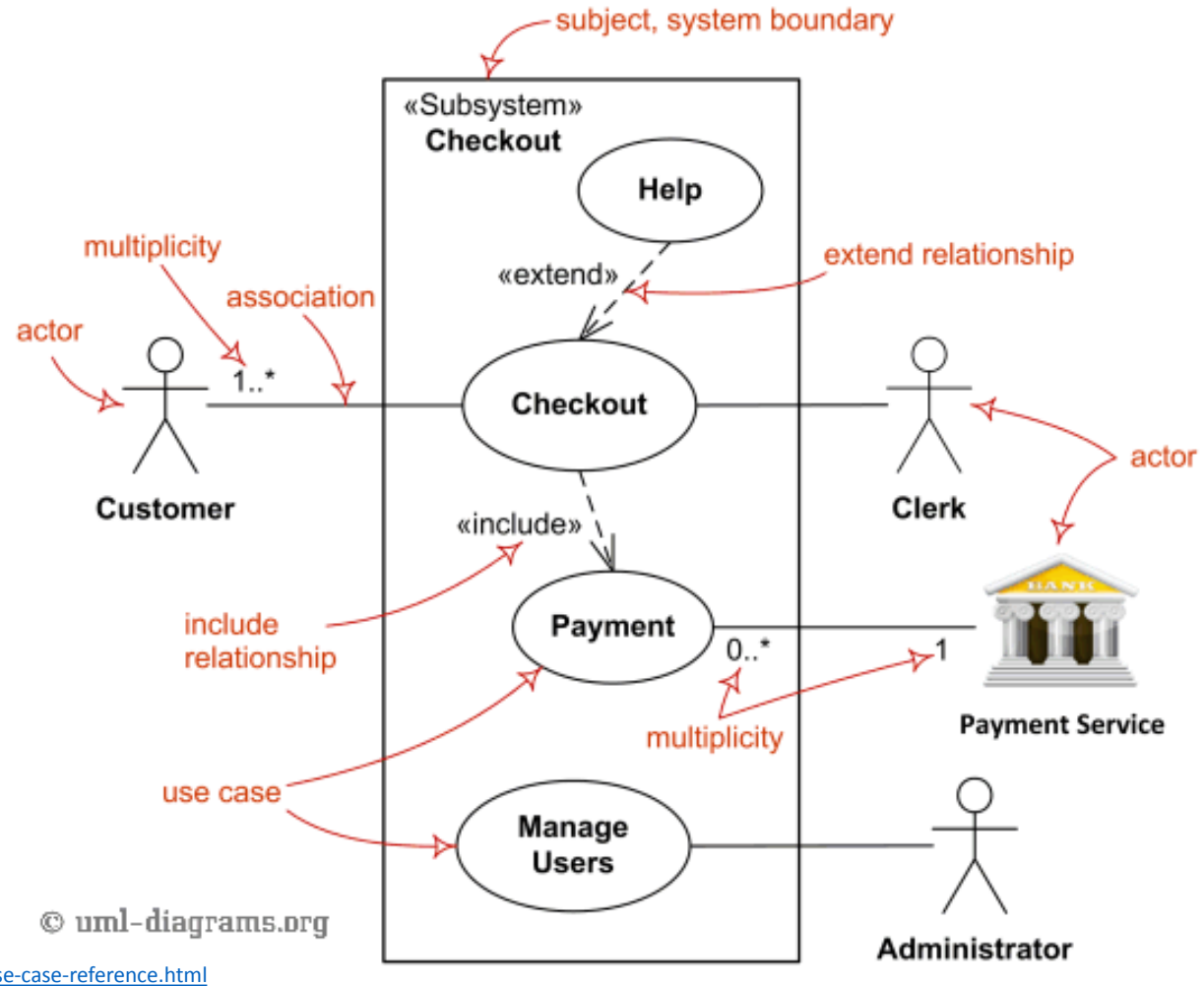
see also: https://www.uml-diagrams.org/use-case-reference.html

Business Use Case Diagrams



see also: https://www.uml-diagrams.org/use-case-reference.html

System Use Case Diagrams



see also: https://www.uml-diagrams.org/use-case-reference.html

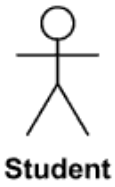
Actors and use cases

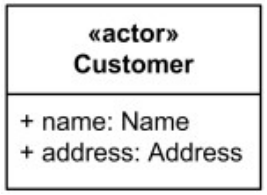
Actor

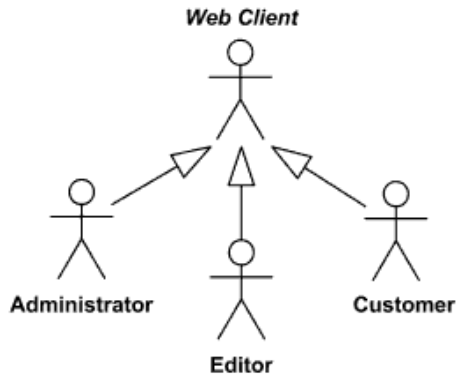
An **actor** is **behaviored classifier** which specifies a **role** played by an **external entity** that interacts with the **subject** (e.g., by exchanging signals and data), a human user of the designed system, some other system or hardware using services of the subject.

Use case

A **use case** is a kind of **behaviored classifier** that specifies a [complete] unit of [useful] functionality performed by [one or more] **subjects** to which the **use case applies** in collaboration with one or more **actors**, and which [for complete use cases] yields an observable result that is of some value to those actors [or other stakeholders] of each subject.







Generalization between actors

see also: https://www.uml-diagrams.org/use-case-reference.html

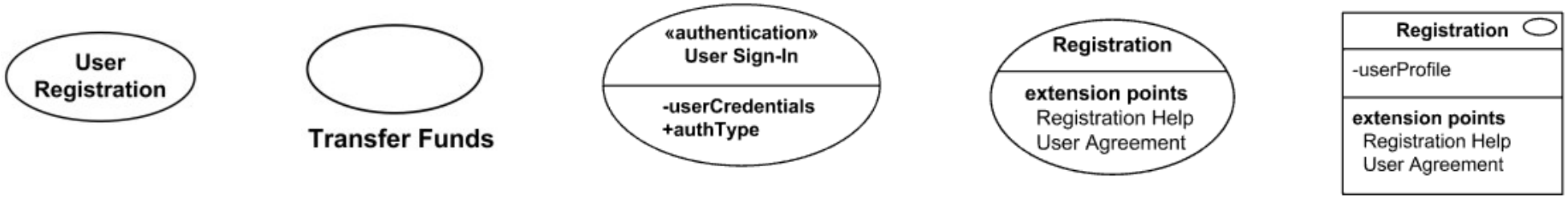
Actors and use cases

Actor

An **actor** is **behaviored classifier** which specifies a **role** played by an **external entity** that interacts with the **subject** (e.g., by exchanging signals and data), a human user of the designed system, some other system or hardware using services of the subject.

Use case

A **use case** is a kind of **behaviored classifier** that specifies a [complete] unit of [useful] functionality performed by [one or more] **subjects** to which the **use case applies** in collaboration with one or more **actors**, and which [for complete use cases] yields an observable result that is of some value to those actors [or other stakeholders] of each subject.

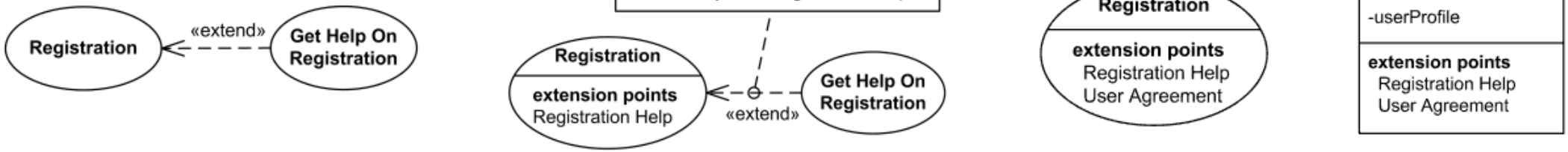


see also: https://www.uml-diagrams.org/use-case-reference.html

Includes and Extends

Extends

Extend is a **directed relationship** that specifies how and when the behavior defined in usually supplementary (optional) **extending use case** can be inserted into the **behavior** defined in the **extended use case**.

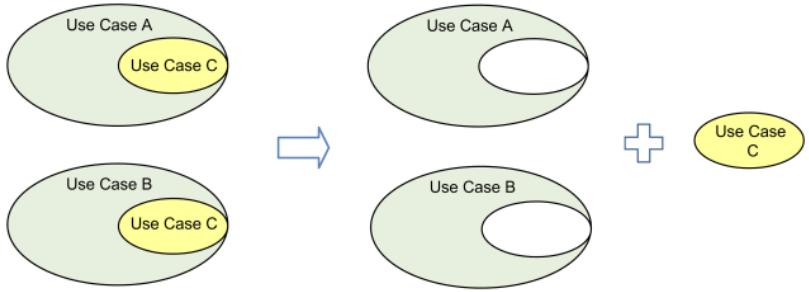


Includes

A **use case** is a kind of **behaviored classifier** that specifies a [complete] unit of [useful] functionality performed by [one or more] **subjects** to which the **use case applies** in collaboration with one or more **actors**, and which [for complete use cases] yields an observable result that is of some value to those actors [or other stakeholders] of each subject.

see also: https://www.uml-diagrams.org/use-case-reference.html

Includes and Extends

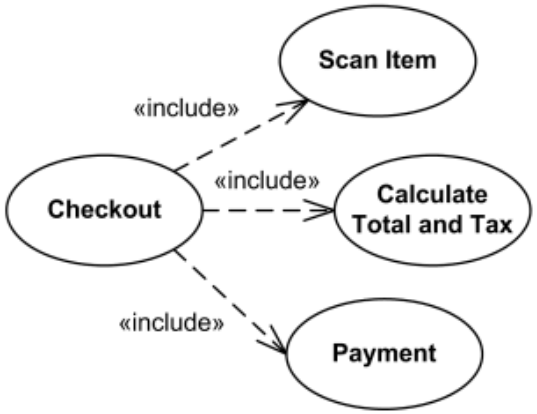


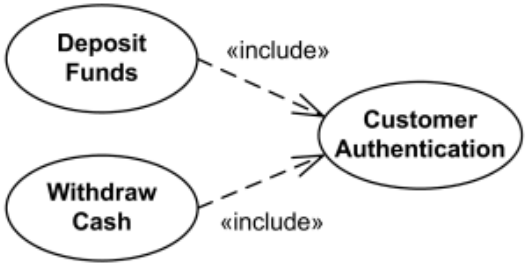
Includes

Use case include is a **directed relationship** between two **use cases** which is used to show that behavior of the **included** use case (the addition) is inserted into the **behavior** of the **including** (the base) use case.

The **include** relationship could be used:

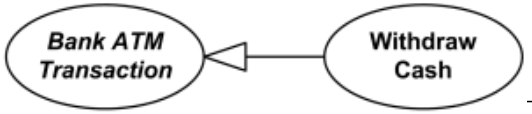
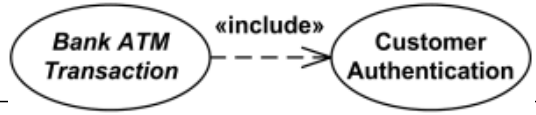
- to simplify large use case by splitting it into several use cases,
- to extract **common parts** of the behaviors of two or more use cases.





see also: https://www.uml-diagrams.org/use-case-reference.html

Use Case Relationships Compared

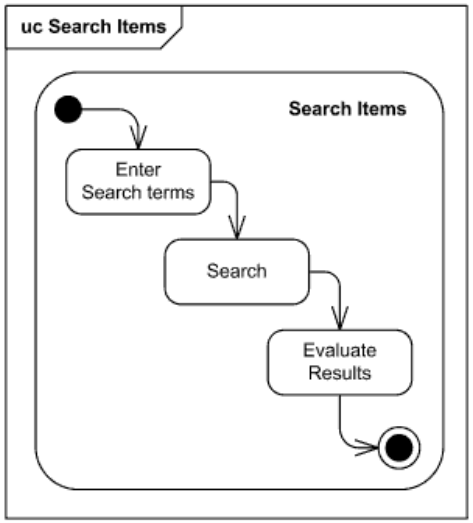
Generalization	Extend	Include
		
Base use case could be **abstract use case** (incomplete) or concrete (complete).	Base use case is complete (concrete) by itself, defined independently.	Base use case is incomplete (**abstract use case**).
Specialized use case is required, not optional, if base use case is abstract.	Extending use case is optional, supplementary.	Included use case required, not optional.
No explicit location to use specialization.	Has at least one explicit extension location.	No explicit inclusion location but is included at some location.
No explicit condition to use specialization.	Could have optional extension condition.	No explicit inclusion condition.

see also: https://www.uml-diagrams.org/use-case-reference.html

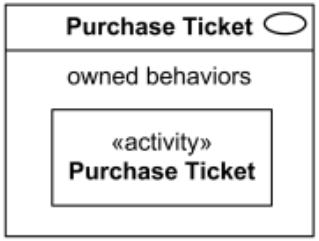
Describe Use Case Behavior

Use case **behaviors** may be described in a natural language text (opaque behavior), which is current common practice, or by using UML **behavior diagrams** for specific behaviors such as

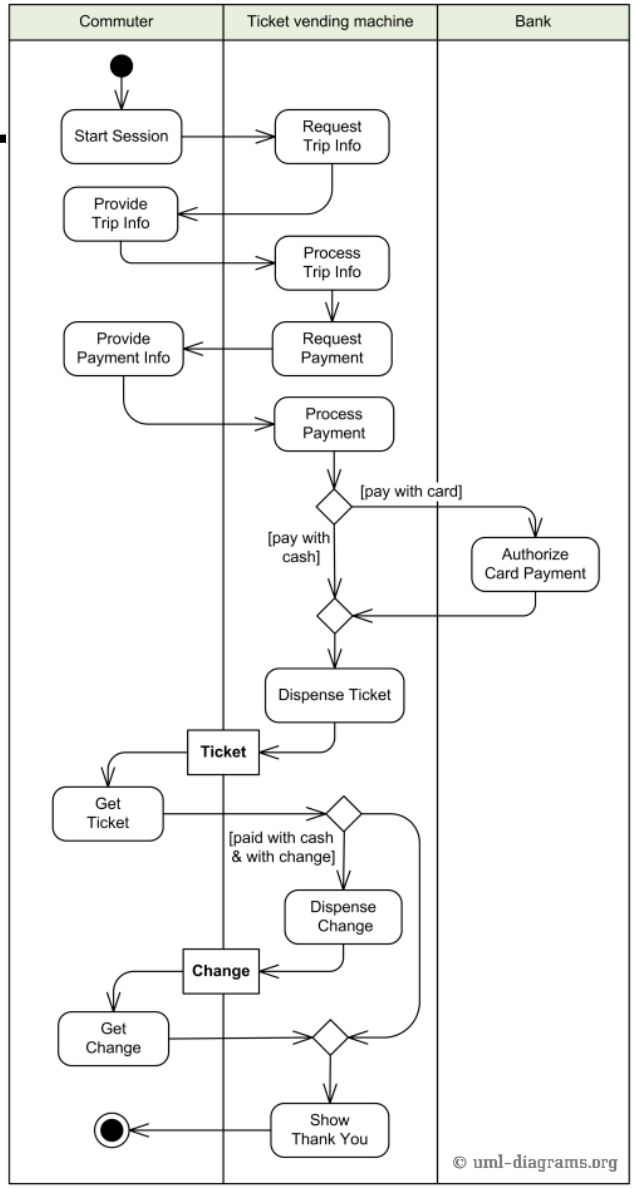
- **activity**,
- **state machine**,
- interaction.



description using a state machine diagram



link between a use case and an activity



Activity diagram: description of the Purchase Ticket activity

see also: https://www.uml-diagrams.org/use-case-reference.html

Use Case diagrams examples

See https://www.uml-diagrams.org/use-case-diagrams-examples.html

Software Engineering

Part 6 – The Unified Modeling Language

6.2 – diagrams we'll use for the analysis phase

6.2.2 – Activity diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

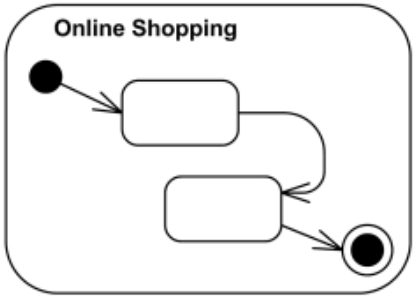
Activity diagrams

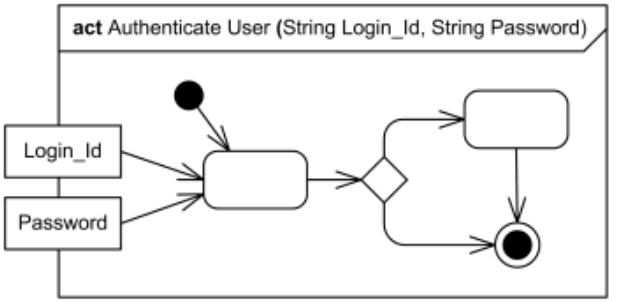
Activity diagram *is UML* ***behavior diagram*** *which shows* ***flow of control*** *or* ***object flow*** *with emphasis on the sequence and conditions of the flow. The actions coordinated by activity models can be initiated because other actions finish executing, because objects and data become available, or because some events external to the flow occur.*

terminology: **activity**, **partition**, **action**, **object**, **control**, **activity edge**.

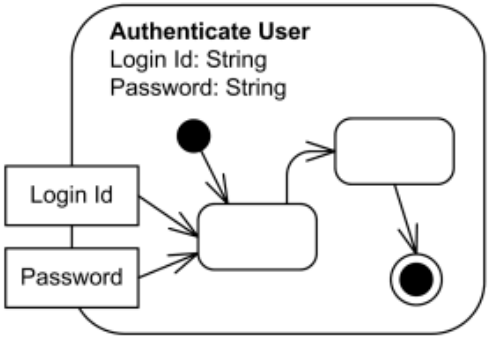
see also: https://www.uml-diagrams.org/activity-diagrams-reference.html

Activity diagrams

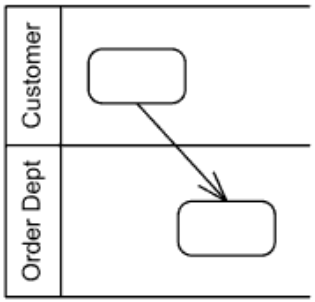




Rendered with the frame notation for diagrams: keyword **act**



With parameters

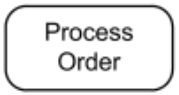


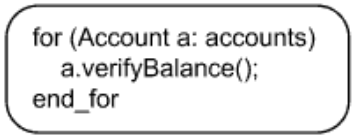
With swimlanes

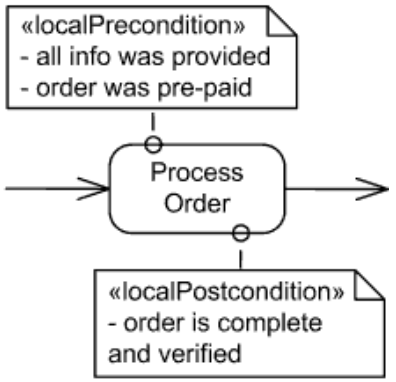
terminology: **activity**, partition, action, **object**, control, activity edge.

see also: https://www.uml-diagrams.org/activity-diagrams-reference.html

Actions



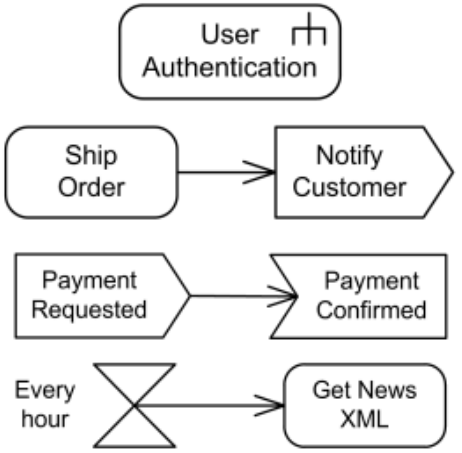




Types of actions

Action is a named element which represents a single atomic step within activity, i.e. that is not further decomposed within the activity. Activity represents a behavior that is composed of individual elements that are actions.

- Object actions include different actions on objects, e.g. create and destroy object, test object identity, specify value, etc.
- Variable actions include variable read, write, add, remove and clear actions.
- Invocation actions include several call actions, signal send and broadcast actions and send object action.
- Send signal action
- Accept signal action
- Wait time action
- ...



terminology: **activity**, partition, action, **object**, control, activity edge.

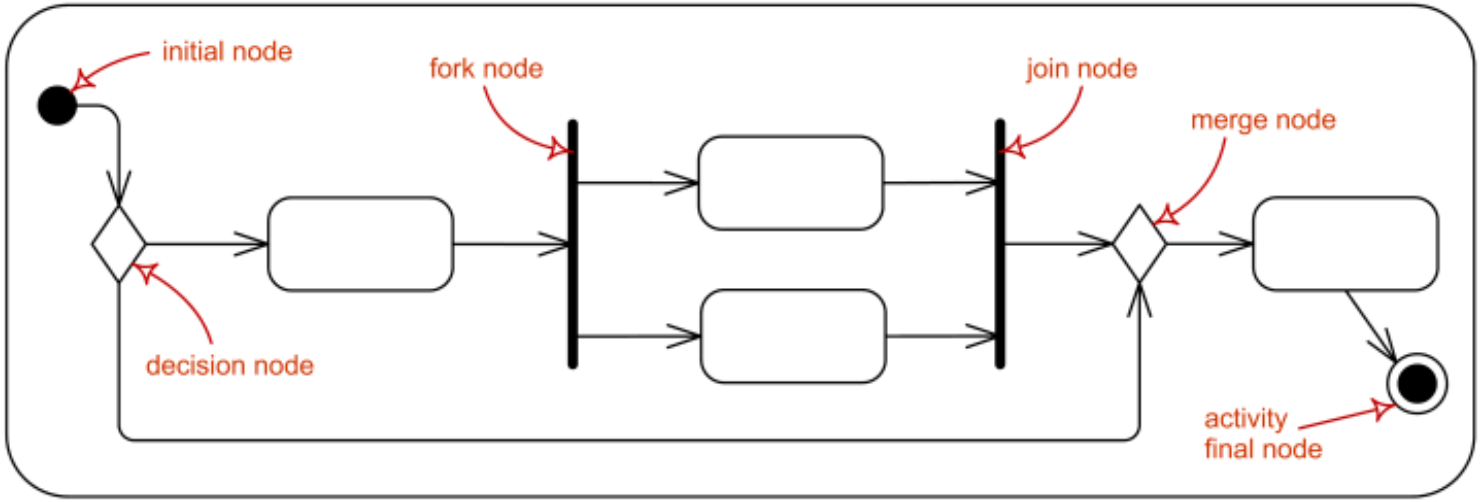
see also: https://www.uml-diagrams.org/activity-diagrams-actions.html

Controls

Types of controls

Control node is an activity node used to coordinate the flows between other nodes. It includes:

- initial node
- flow final node
- activity final node
- decision node
- merge node
- fork node
- join node



terminology: **activity**, partition, action, **object**, control, activity edge.

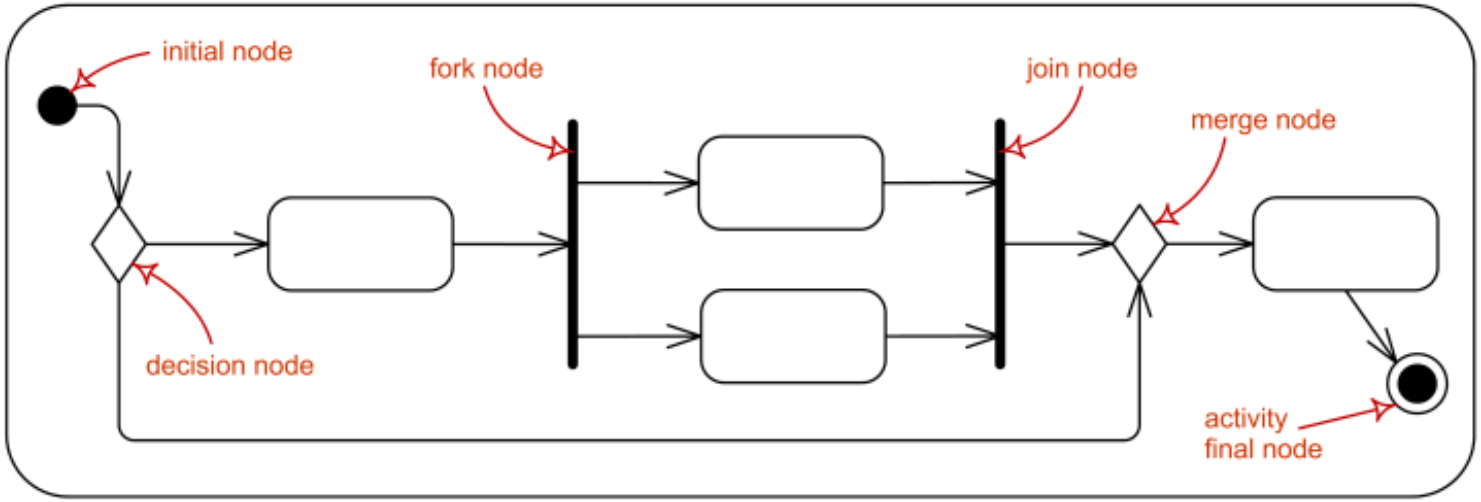
see also: https://www.uml-diagrams.org/activity-diagrams-reference.html

Controls

Types of controls

Control node is an activity node used to coordinate the flows between other nodes. It includes:

- initial node
- flow final node
- activity final node
- decision node
- merge node
- fork node
- join node



terminology: **activity**, partition, action, **object**, control, activity edge.

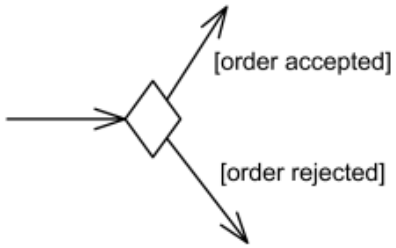
see also: https://www.uml-diagrams.org/activity-diagrams-objects.html

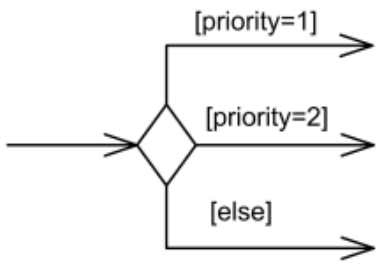
Controls

Types of controls

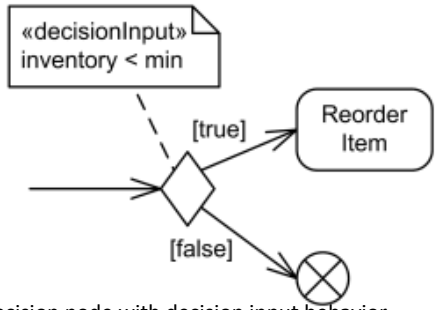
Control node is an activity node:

- initial node
- flow final node
- activity final node
- **decision node**
- merge node
- fork node
- join node

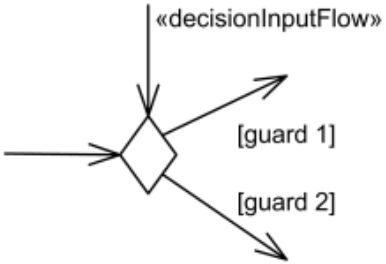




Decision node with outgoing edges with guards



Decision node with decision input behavior.



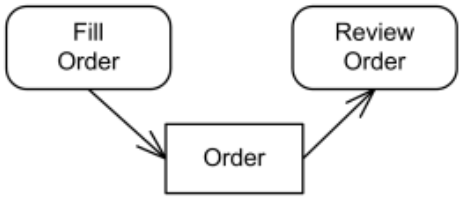
Decision node with decision input flow.

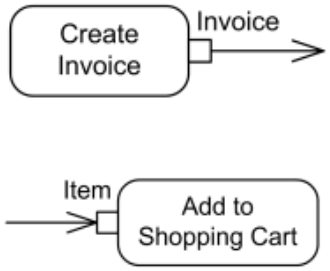
terminology: **activity**, partition, action, **object**, control, activity edge.

see also: https://www.uml-diagrams.org/activity-diagrams-controls.html

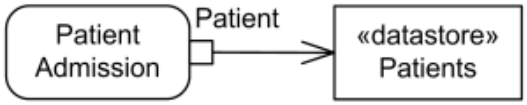
Objects

Objects flow in an activity





input and output pins



A data store is a central buffer node for non-transient information.

terminology: **activity**, partition, action, **object**, control, activity edge.

see also: https://www.uml-diagrams.org/activity-diagrams-controls.html

Activity diagrams examples

See https://www.uml-diagrams.org/activity-diagrams-examples.html

Software Engineering

Part 6 – The Unified Modeling Language

6.2 – diagrams we'll use for the analysis phase

6.2.3 – State machine diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

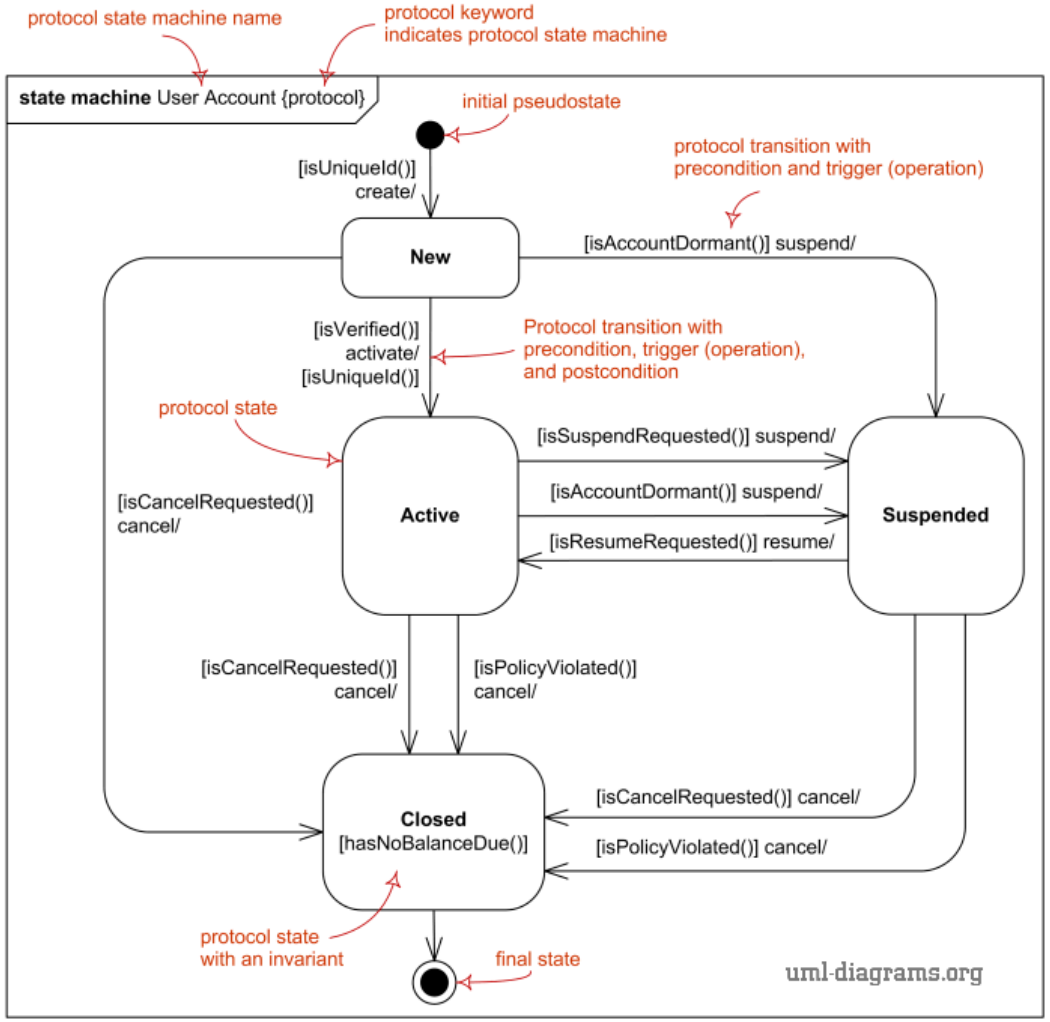
State machine diagrams

*Used for modeling discrete behavior through finite state transitions. In addition to expressing the **behavior** of a part of the system, state machines can also be used to express the **usage protocol** of part of a system. These two kinds of state machines are referred to as **behavioral state machines** and **protocol state machines**.*

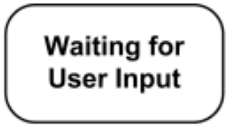
terminology: **behavioral state**, **behavioral transition**, **protocol state**, **protocol transition**, different **pseudostates**.

see also: https://www.uml-diagrams.org/activity-diagrams-reference.html

State machine diagrams

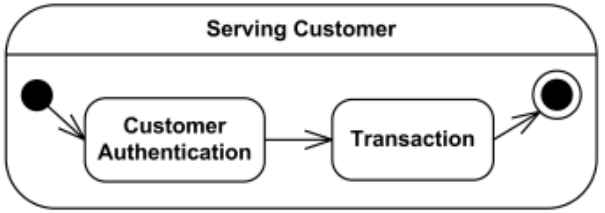


States



Simple state

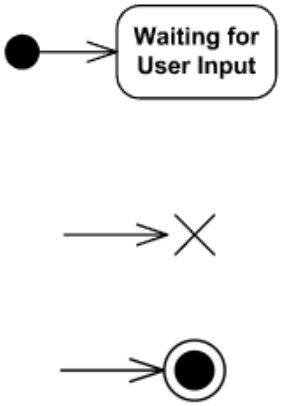
List of internal activities



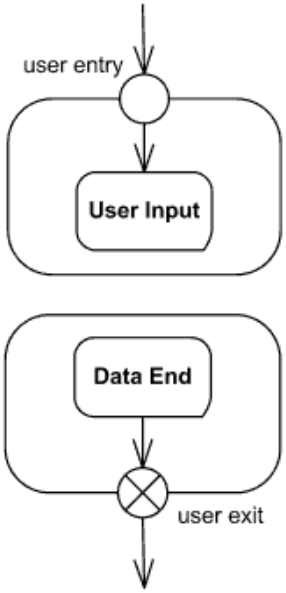
Composite state

Composite state with hidden decomposition

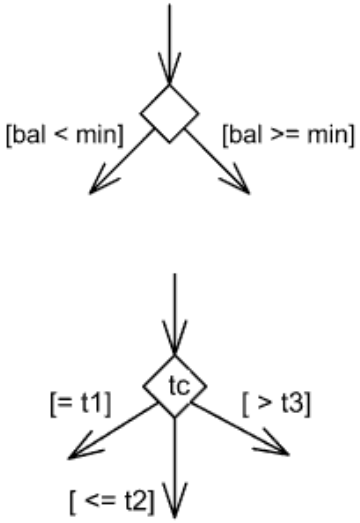
Pseudostates



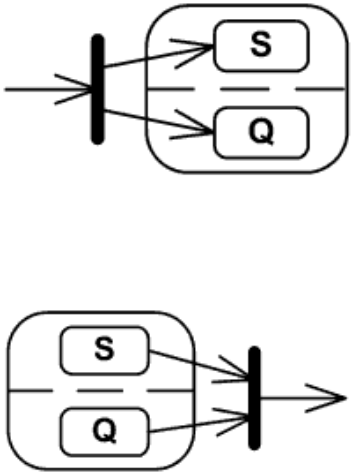
initial/terminate(destroy)/final



entry/exit



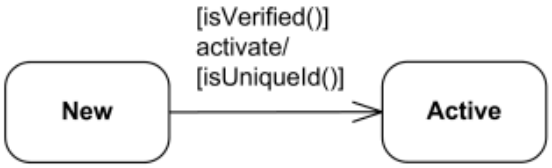
choice



fork/join

Protocol transition

A **protocol transition** is specialization of **(behavioral) transition** used for the protocol state machines which specifies a legal transition for an **operation**. Protocol transition has the following **features**: a pre-condition (guard), **trigger**, and a post-condition.



protocol-transition ::= [***pre-condition***] ***trigger*** '/' [***post-condition***]
pre-condition ::= '[' ***constraint*** ']'
post-condition ::= '[' ***constraint*** ']'

State Machine diagram examples

See https://www.uml-diagrams.org/state-machine-diagrams-examples.html

Software Engineering

Part 6 – The Unified Modeling Language

6.2 – diagrams we'll use for the analysis phase

6.2.3 – Communication diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

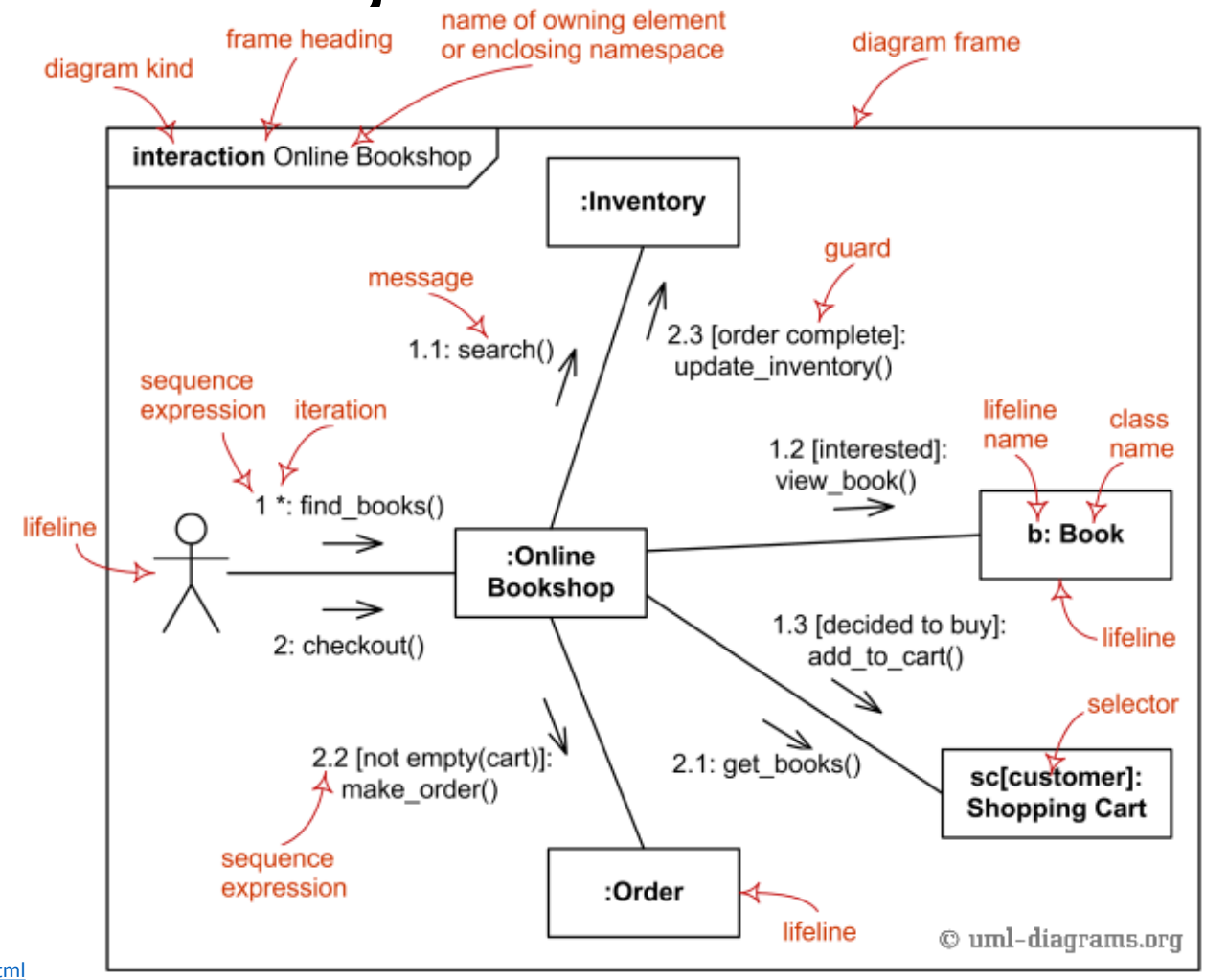
Communication diagrams

Communication diagram *(called* ***collaboration diagram*** *in UML 1.x) is a kind of UML* ***interaction diagram*** *which shows interactions between objects and/or* ***parts*** *(represented as* ***lifelines****) using sequenced messages in a free-form arrangement.*

terminology: **frame**, **lifeline**, **message**

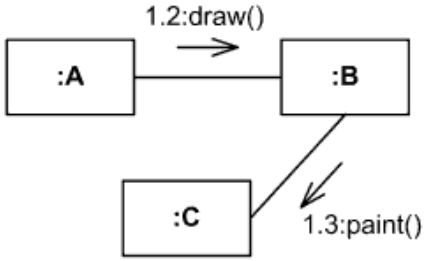
see also: https://www.uml-diagrams.org/communication-diagrams-reference.html

Communication diagrams

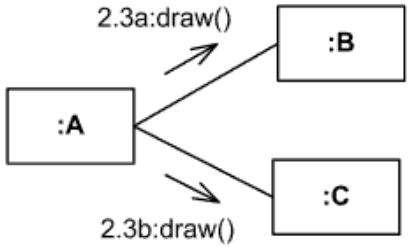


see also: https://www.uml-diagrams.org/communication-diagrams-reference.html

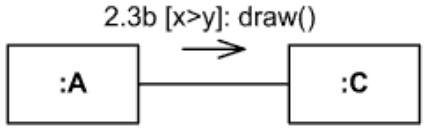
Communication diagrams



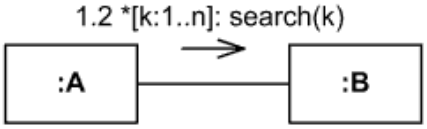
in sequence



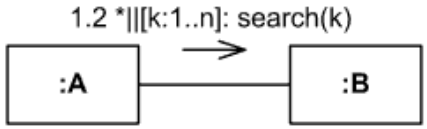
in parallel



guards



n times, in sequence



n times in parallel

see also: https://www.uml-diagrams.org/communication-diagrams-reference.html

Communication diagram examples

See https://www.uml-diagrams.org/communication-diagrams-examples.html

Software Engineering

Part 6 – The Unified Modeling Language

6.3 – diagrams we'll use for the design phase

6.3.1 – Class diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Class diagrams

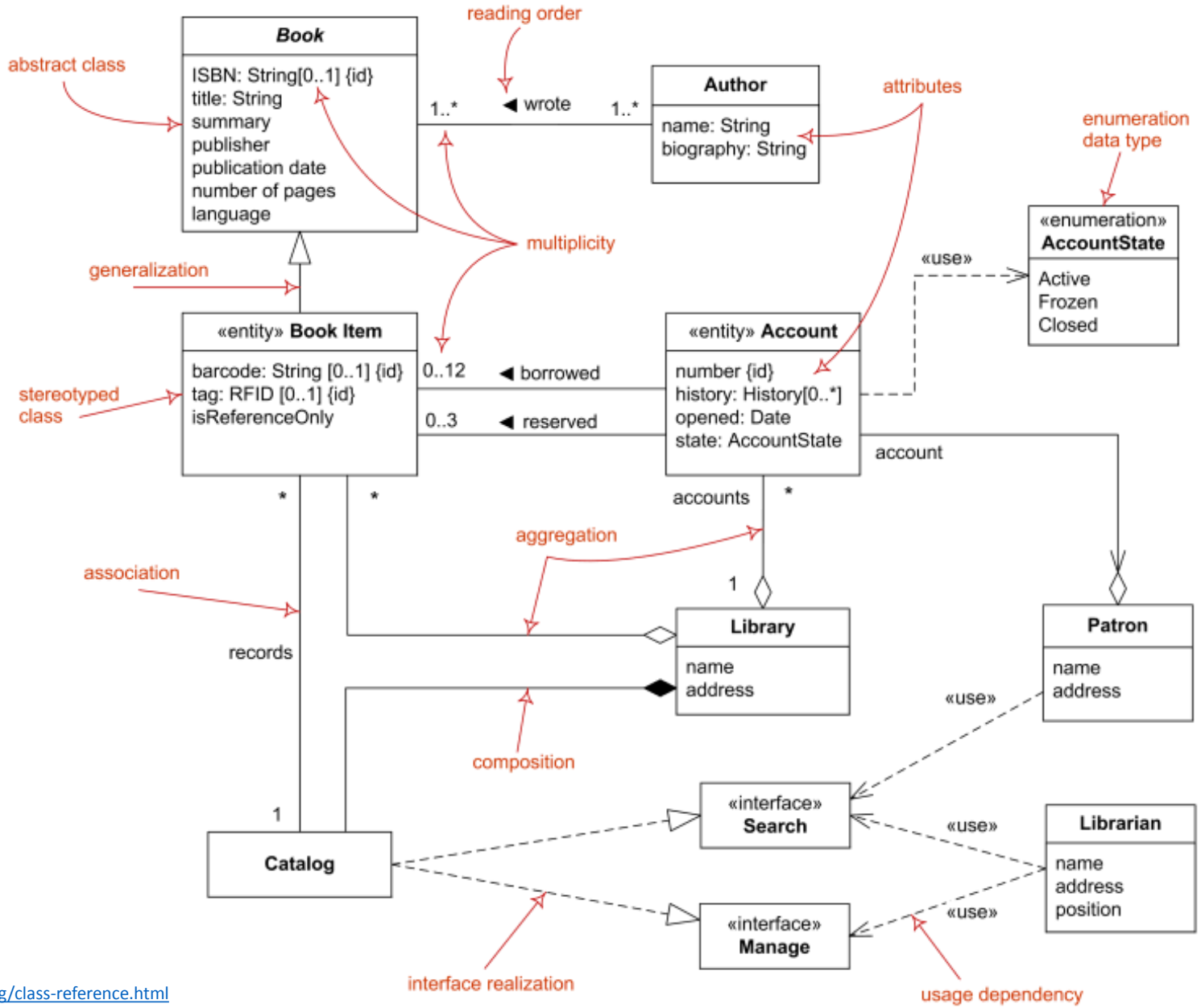
Class diagram *is* ***UML*** ***structure diagram*** *which shows structure of the designed system at the level of* ***classes*** *and* ***interfaces****, shows their* ***features****,* ***constraints*** *and relationships -* ***associations****,* ***generalizations****,* ***dependencies****, etc.*

types of class diagrams are:

- **domain model diagram**,
- **diagram of implementation classes**.

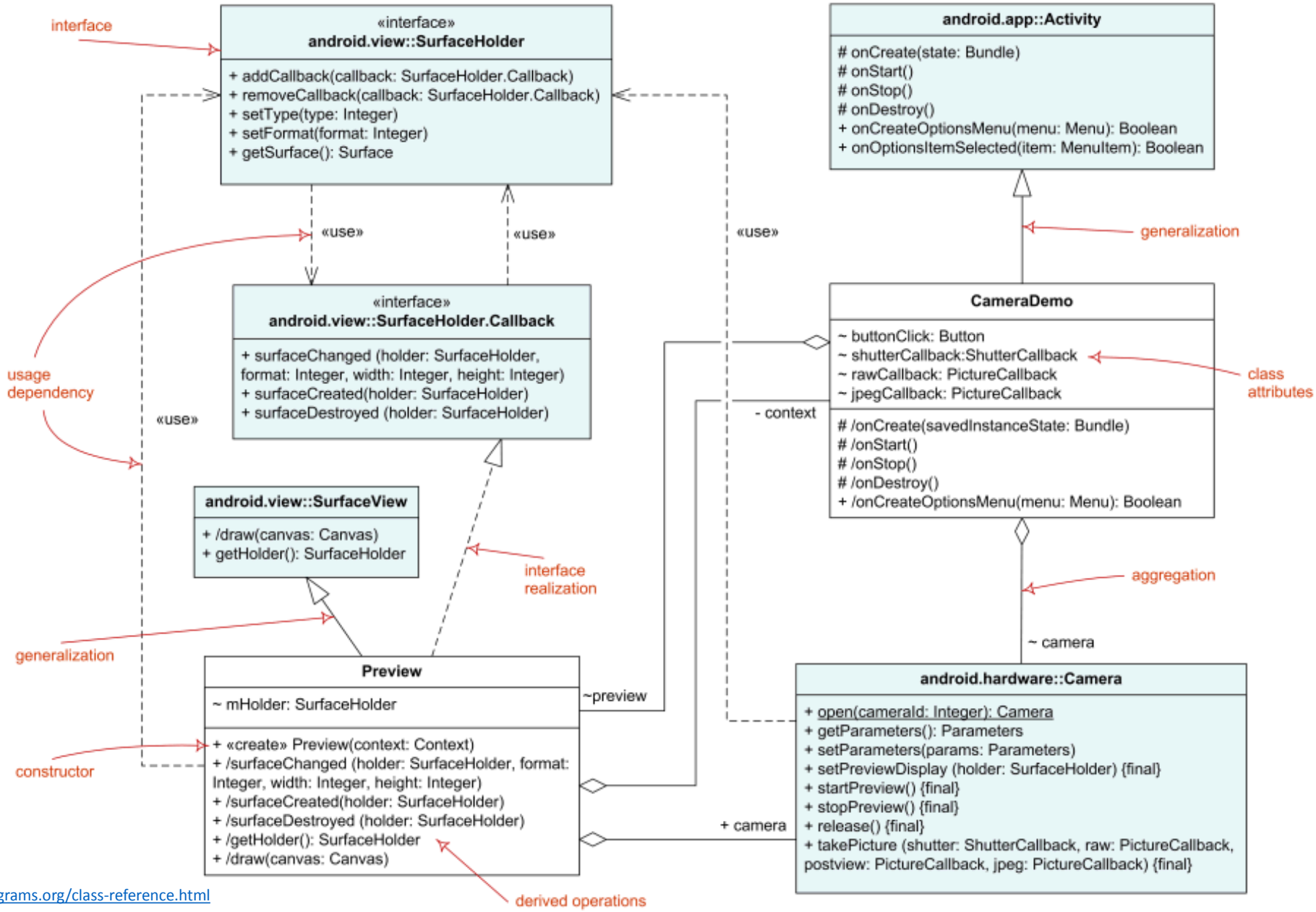
see also: https://www.uml-diagrams.org/class-reference.html

Domain model diagrams



see also: https://www.uml-diagrams.org/class-reference.html

Diagram of Implementation Classes



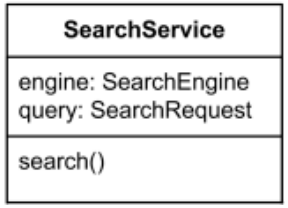
see also: https://www.uml-diagrams.org/class-reference.html

Class

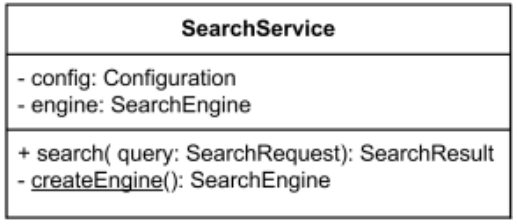
Class

A **class** is a **classifier** which describes a set of objects that share the same:

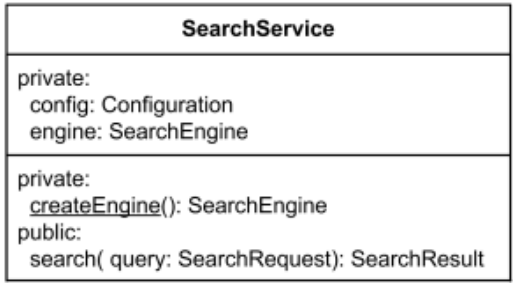
- **features**,
- **constraints**,
- **semantics (meaning)**.



***Class** SearchService - analysis level details*



***Class** SearchService - implementation level details.*
*The createEngine is **static** operation*

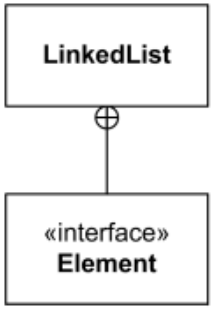


***Class** SearchService - attributes and operations grouped by visibility*

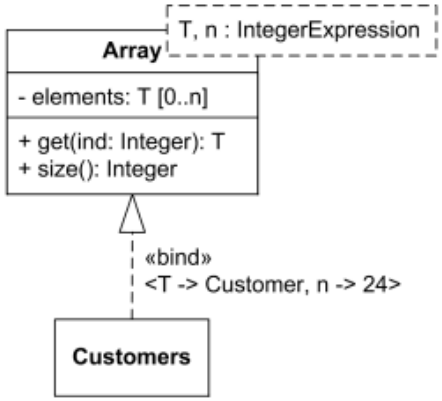
see also: https://www.uml-diagrams.org/class-reference.html

Abstract, Nested, Template, Interface

Abstract class (italics)

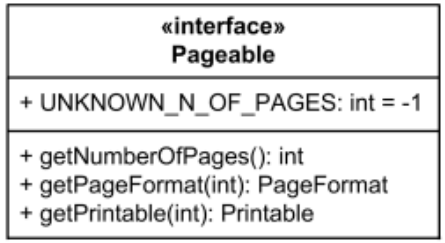


Class LinkedList is nesting the Element interface. The Element is in scope of the LinkedList namespace.



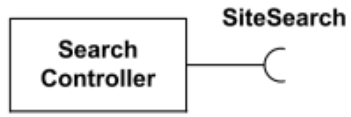
Template class Array and bound class Customers. The Customers class is an Array of 24 objects of Customer class.

An interface



Various kinds of constraints or protocol specifications (ordering restrictions)

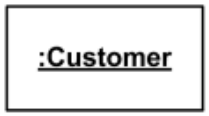
Interface realization



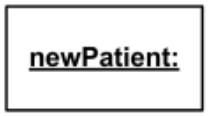
Interface usage

see also: https://www.uml-diagrams.org/class-reference.html

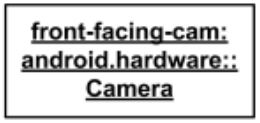
Objects



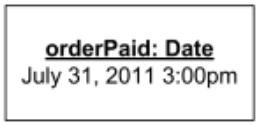
Anonymous instance of the Customer class



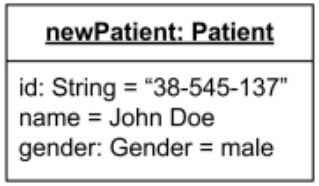
Instance newPatient of the unnamed or unknown class



Instance front-facing-cam of the Camera class from android.hardware package.



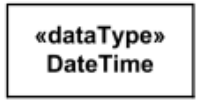
Instance orderPaid of the Date class has value July 31, 2011 3:00 pm.



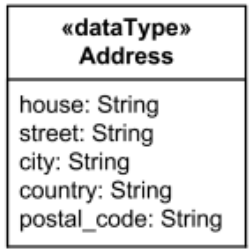
Instance newPatient of the Patient class has slots with values specified.

see also: https://www.uml-diagrams.org/class-reference.html

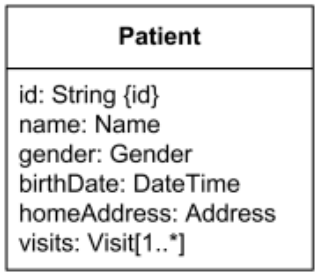
Data Type, primitive type, enumeration type



DateTime data type

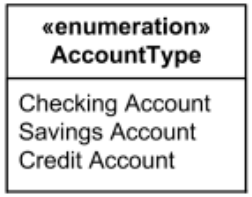


Structured data type



Attributes of the Patient class are of data types Name, Gender, DateTime, Address and Visit.

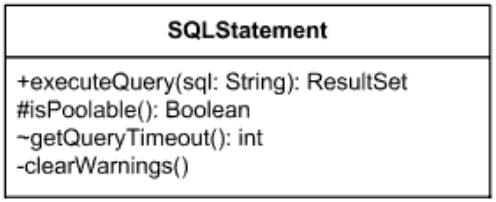
Primitive data type. Standard UML primitive types include:
- Boolean,
- Integer,
- UnlimitedNatural,
- String.



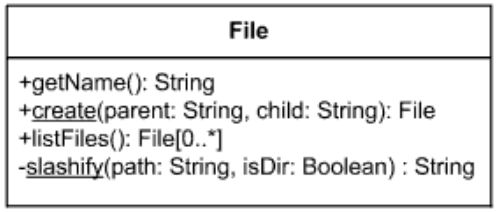
values are enumerated in the model as user-defined enumeration literals

see also: https://www.uml-diagrams.org/class-reference.html

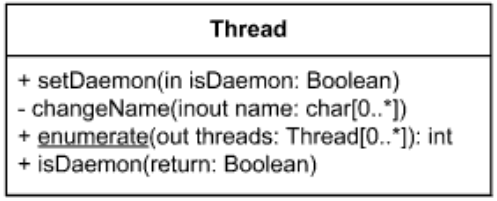
Operations



Operations with different visibilities
executeQuery is public, isPoolable is protected, getQueryTimeout has package visibility,
clearWarnings is private.



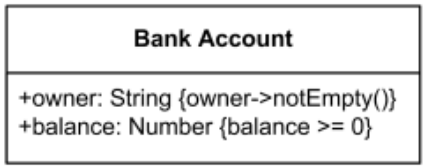
static operations are underlined
operations have a signature, with parameters, and a return type.
File has two static operations - create and slashify. Create has two parameters and returns File. Slashify is private operation. Operation listFiles returns array of files. Operations getName and listFiles either have no parameters or parameters were suppressed.



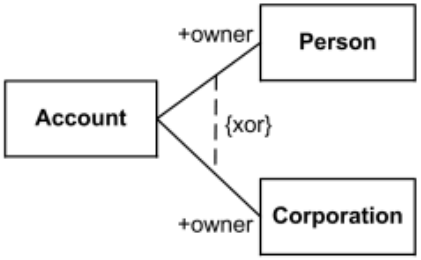
Operation setDaemon has one input parameter, while single parameter of changeName is both input and output parameter. Static enumerate returns integer result while also having output parameter - array of threads. Operation isDaemon is shown with return type parameter. It is presentation option equivalent to returning operation result as: +isDaemon(): Boolean.

see also: https://www.uml-diagrams.org/class-reference.html

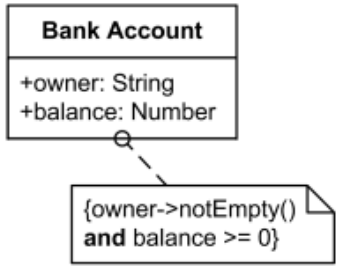
Write constraints



Bank account attribute constraints - non empty owner and positive balance.



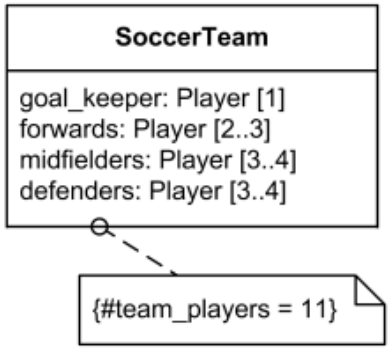
Account owner is either Person or Corporation, {xor} is predefined UML constraint.



Bank account constraints - non empty owner and positive balance

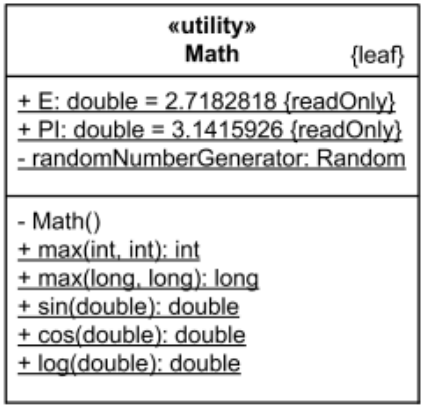
see also: https://www.uml-diagrams.org/class-reference.html

Members and multiplicity



Multiplicity of players for SoccerTeam class

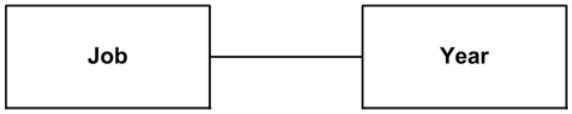
0	Collection must be empty
1	Exactly one instance
5	Exactly 5 instances
*	Zero or more instances
0..1	No instances or one instance
1..1	Exactly one instance
0..*	Zero or more instances
1..*	At least one instance
m..n	At least m but no more than n instances



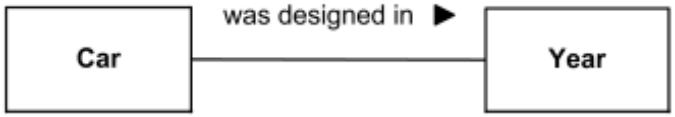
Utility: class that has only class scoped **static attributes and operations**.

see also: https://www.uml-diagrams.org/class-reference.html

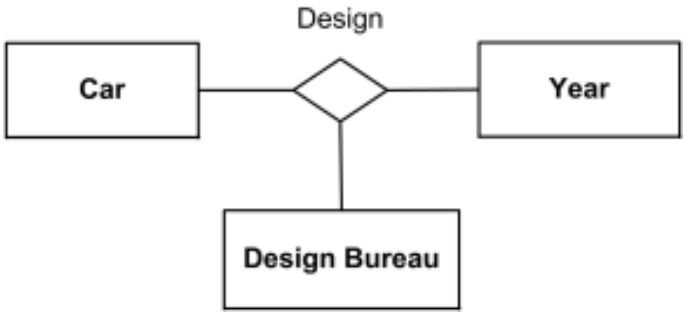
Associations



Association

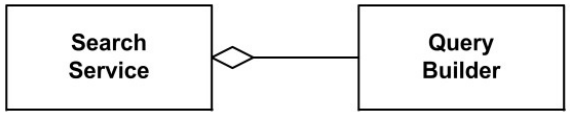


Order of the ends and reading: Car - was designed in - Year

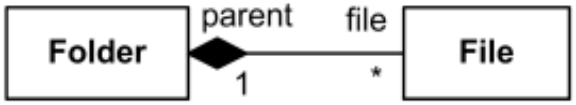


Ternary association Design relating three classifiers.

Aggregation/composition



Aggregation



Composite Aggregation (= composition)
If folder is deleted, all files are deleted as well

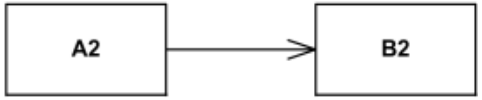
Ownership

Association end qb is an attribute of SearchService class and is owned by the class.

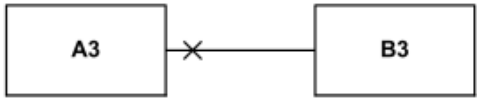
Association qualifier

Given a company and a social security number (SSN) at most one employee could be found.

Navigability



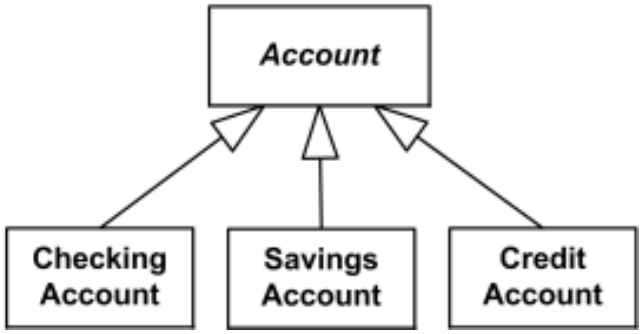
A2 has unspecified navigability while B2 is navigable from A2.



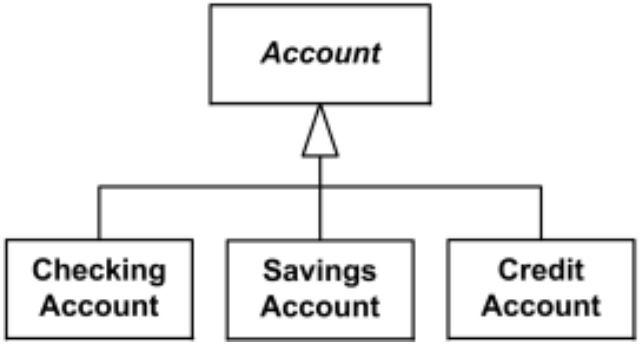
A3 is not navigable from B3 while B3 has unspecified navigability.

see also: https://www.uml-diagrams.org/class-reference.html

Generalization



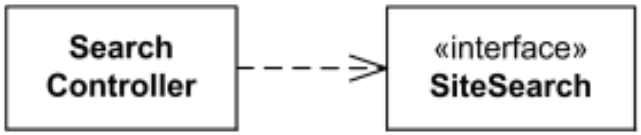
=

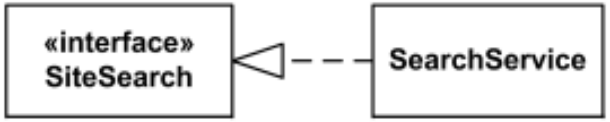


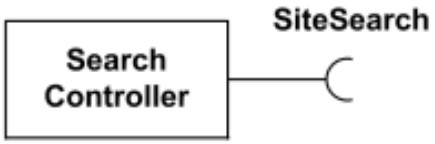
Checking, Savings, and Credit Accounts are generalized by Account.

see also: https://www.uml-diagrams.org/class-reference.html

Interfaces





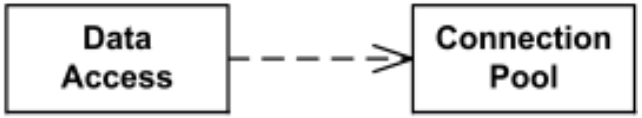


Interface SiteSearch is used (required) by Search Controller.

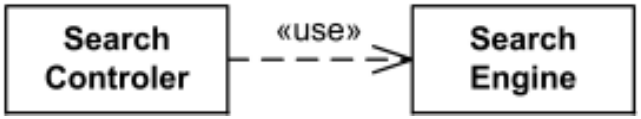
Interface SiteSearch is realized (implemented) by SearchService.

see also: https://www.uml-diagrams.org/class-reference.html

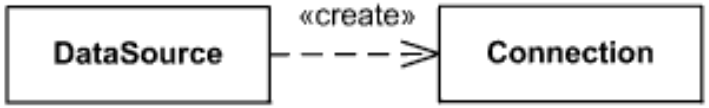
Dependency



Data Access depends on Connection Pool



Search Controller uses Search Engine.



Data Source creates Connection

see also: https://www.uml-diagrams.org/class-reference.html

Class diagram examples

See https://www.uml-diagrams.org/class-diagrams-examples.html

Pointers

- The UML Specification https://www.omg.org/spec/UML/About-UML/
- https://www.uml-diagrams.org/

Software Engineering

Part 6 – The Unified Modeling Language

6.3 – diagrams we'll use for the design phase

6.3.2 – Package diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Package diagrams

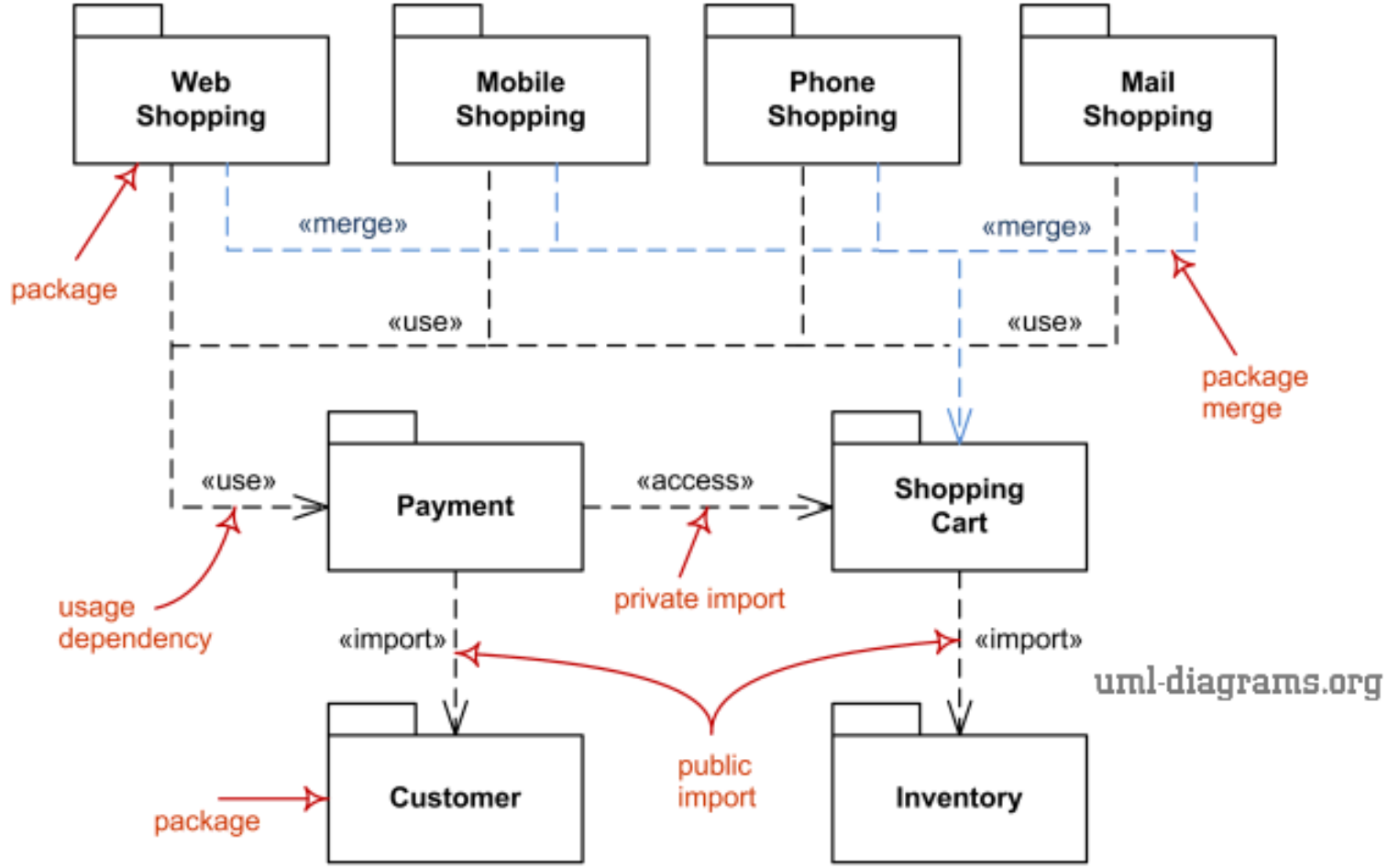
Package diagram *is* ***UML*** ***structure diagram*** *which shows structure of the designed system at the level of* ***packages****.*

Elements:

- **package**,
- **packageable element**,
- **dependency**,
- **element import**,
- **package import**,
- **package merge**.

see also: https://www.uml-diagrams.org/class-reference.html

Package diagrams

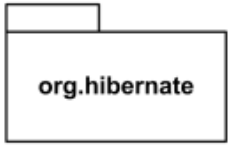


see also: https://www.uml-diagrams.org/package-diagrams-reference.html

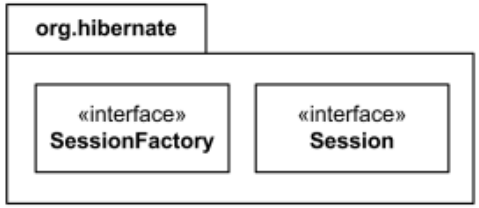
Package

Package

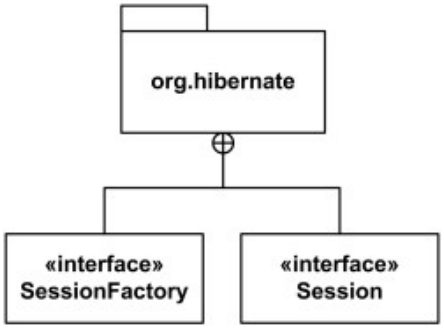
A **package** is a **namespace** used to group together elements that are semantically related and might change together. It is a general purpose mechanism to organize elements into groups to provide better structure for system model.



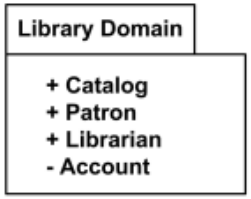
Package org.hibernate



Package org.hibernate contains SessionFactory and Session.



Package org.hibernate contains interfaces SessionFactory and Session.

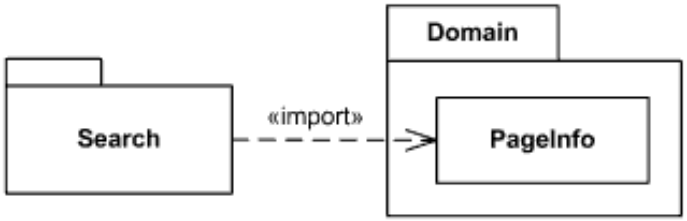


All elements of Library Domain package are public except for Account.

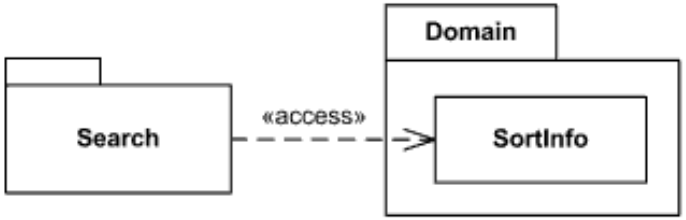
see also: https://www.uml-diagrams.org/package-diagrams-reference.html

Import

Element Import

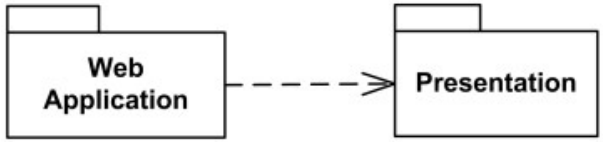


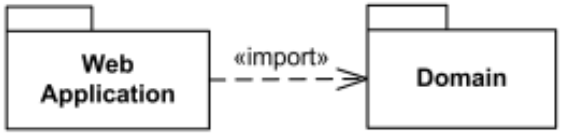
Public import of PageInfo element into Search namespace from Domain package.
Imported element are added to the namespace and made visible outside the namespace
.



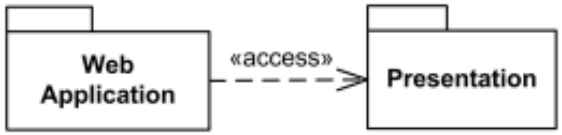
Private import of SortInfo element into Search namespace from Domain package.
Imported element are added to the namespace but not visible outside the namespace
.

Package Import





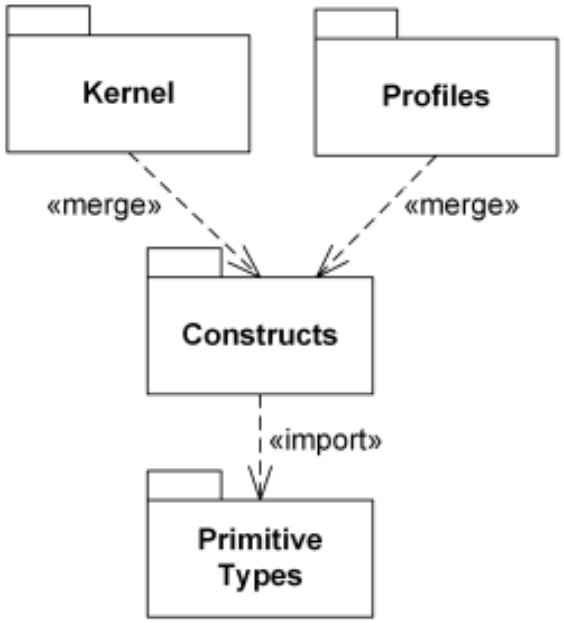
Public import:
All elements are added to the namespace and made visible outside the namespace



Private import:
All elements are added to the namespace but not visible outside the namespace

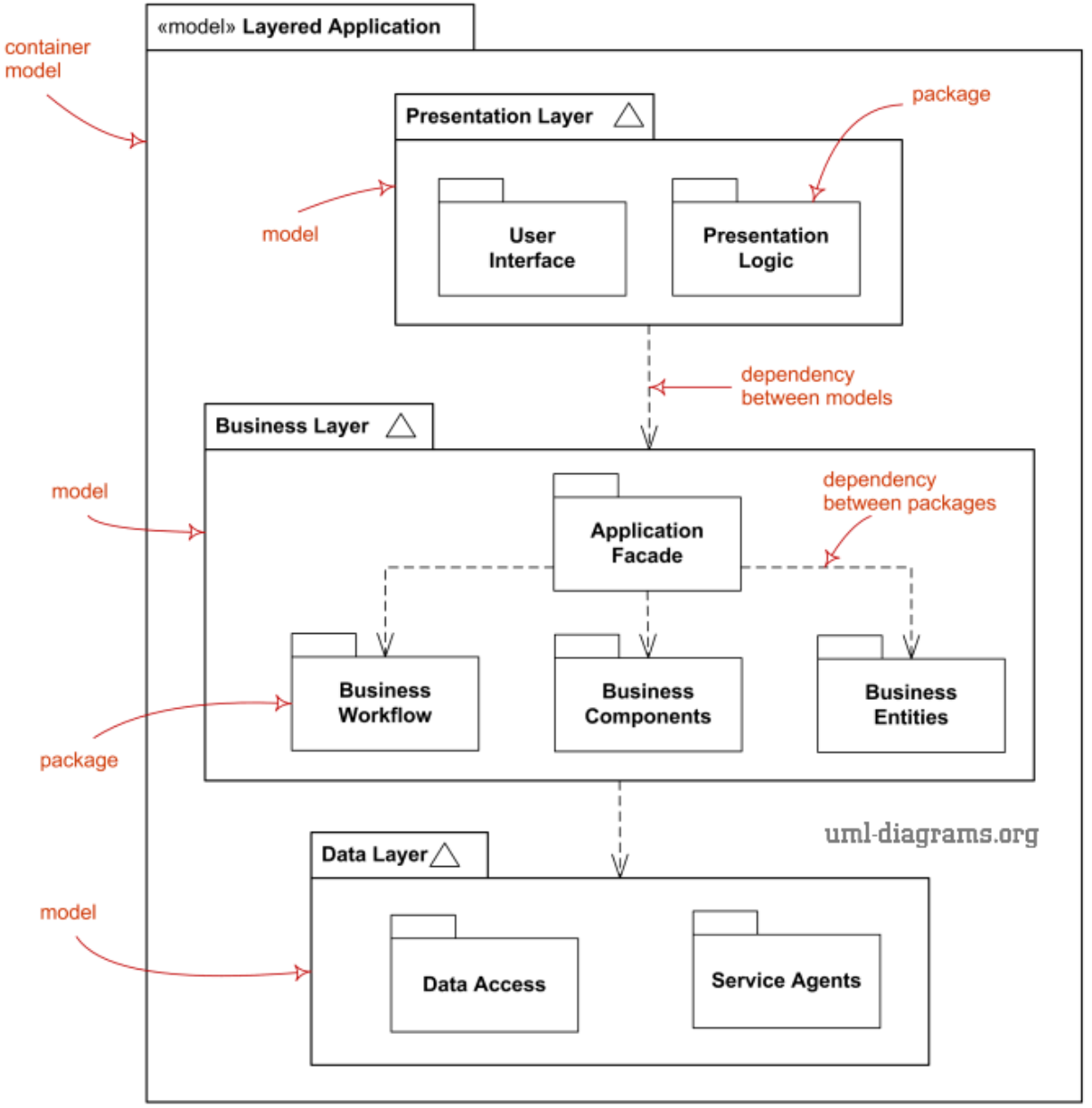
see also: https://www.uml-diagrams.org/package-diagrams-reference.html

Package merge



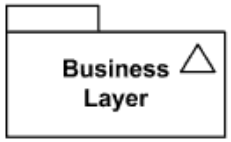
Kernel package merges Constructs package which imports Primitive Types.
The contents of Constructs is combined with the one of Kernel

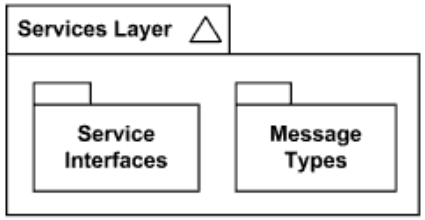
Model diagra

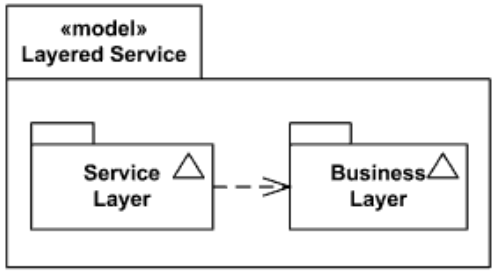


see also: https://www.uml-diagrams.org/package-diagrams-reference.html

Model







Different notations for Models

see also: https://www.uml-diagrams.org/package-diagrams-reference.html

Package diagram examples

See https://www.uml-diagrams.org/package-diagrams-examples.html

Pointers

- The UML Specification https://www.omg.org/spec/UML/About-UML/
- https://www.uml-diagrams.org/

Software Engineering

Part 6 – The Unified Modeling Language

6.3 – diagrams we'll use for the design phase

6.3.3 – Composite Structure diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Composite Structure diagrams

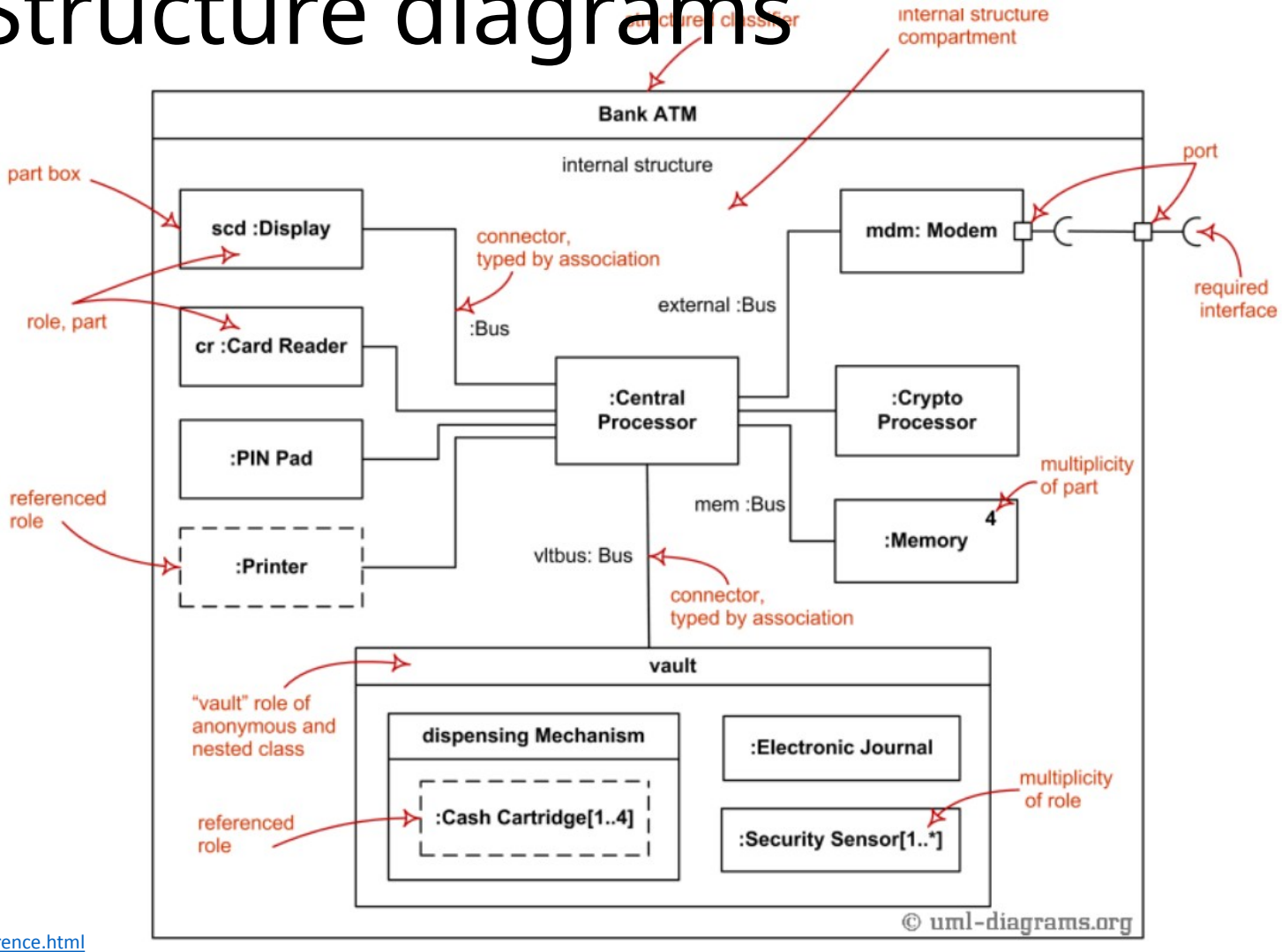
Composite Structure Diagram *could be used to show: internal structure of a classifier -* ***internal structure diagram****, classifier interactions with environment through* ***ports****, a behavior of a collaboration -* ***collaboration use diagram****.*

Elements:

- class,
- part,
- port,
- connector,
- usage

see also: https://www.uml-diagrams.org/composite-structure-diagrams-reference.html

Composite Structure diagrams

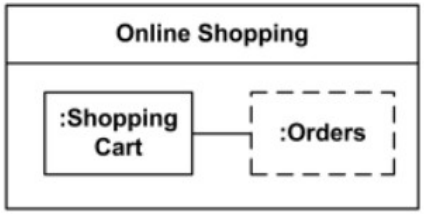


see also: https://www.uml-diagrams.org/composite-structure-diagrams-reference.html

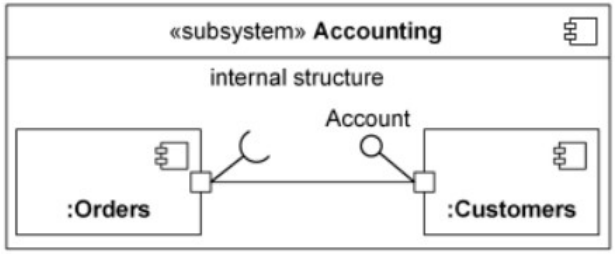
Structured classifier

Structured classifier

Structured classifier is classifier having **internal structure** and whose behavior can be fully or partially described by the collaboration of owned or referenced instances.



Different notations for structured classifiers

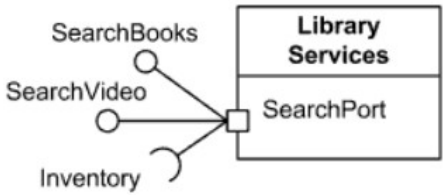


Simple ports joined directly by connector, mandatory UML notation. Customers component part provides Account interface to Orders part.

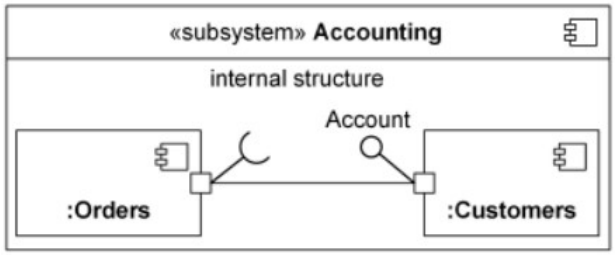
see also: https://www.uml-diagrams.org/composite-structure-diagrams-reference.html

Encapsulated Classifier

Encapsulated classifier is structured classifier extended with the ability to own ports.



Library Services is classifier encapsulated through Search Port



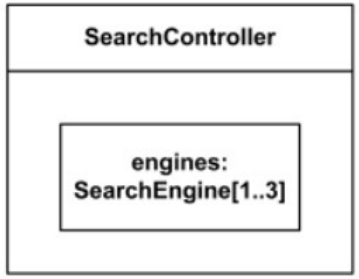
Simple ports joined directly by connector, mandatory UML notation. Customers component part provides Account interface to Orders part.

see also: https://www.uml-diagrams.org/composite-structure-diagrams-reference.html

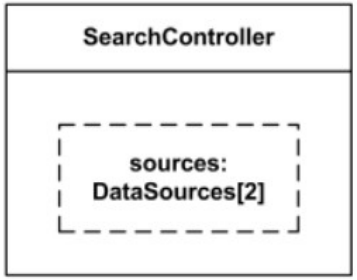
Part

represents a set of instances that are owned by a containing instance of a classifyer.

all parts are destroyed when the containing classifier instance is destroyed (composition)



Search Controller has 1 to 3 engines - Search Engine part

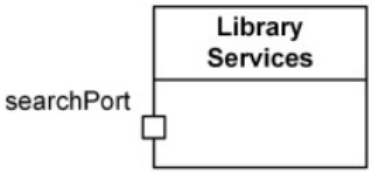


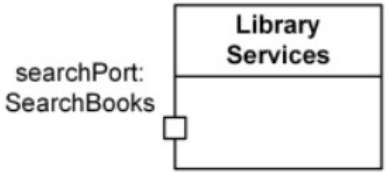
Two Data Sources is sources property - but not part - of Search Controller

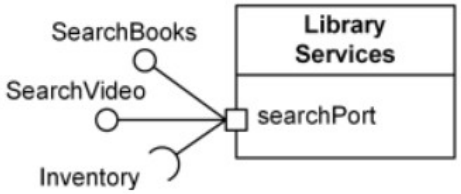
see also: https://www.uml-diagrams.org/composite-structure-diagrams-reference.html

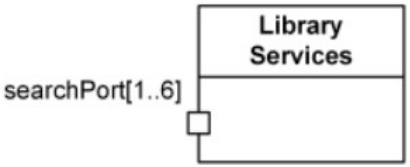
Port

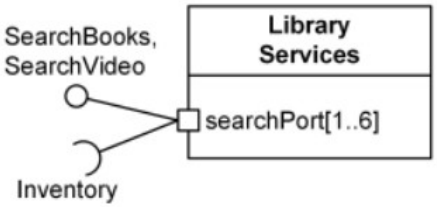
feature which specifies a link that enables communication between two or more instances playing some roles within a structured classifier.







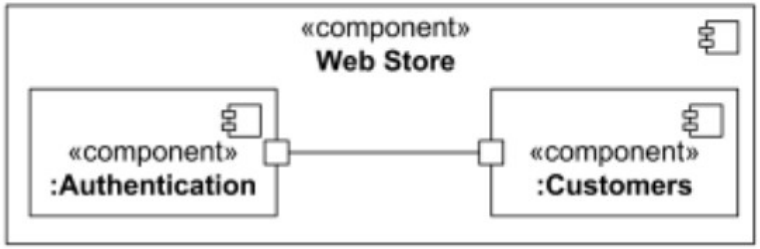




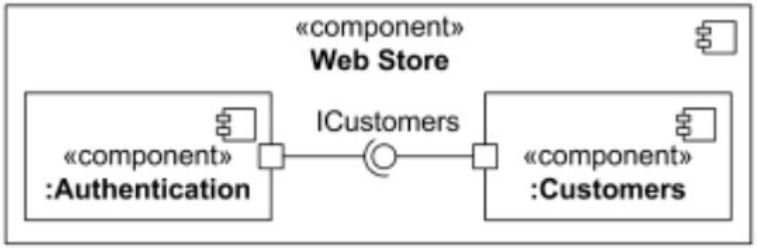
see also: https://www.uml-diagrams.org/composite-structure-diagrams-reference.html

Connectors

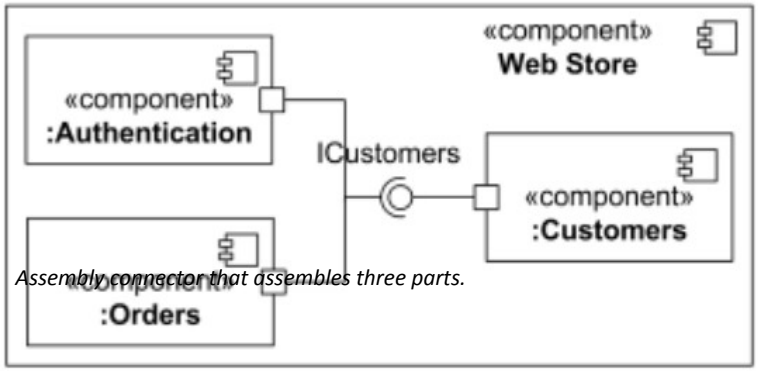
feature which specifies a link that enables communication between two or more instances playing some roles within a structured classifier.



Assembly connector between ports of Authentication and Customers components.



Assembly connector between simple ports of Authentication and Customers components.



Assembly connector that assembles three parts.

see also: https://www.uml-diagrams.org/composite-structure-diagrams-reference.html

Composite structure diagram examples

See https://www.uml-diagrams.org/composite-structure-examples.html

Software Engineering

Part 6 – The Unified Modeling Language

6.3 – diagrams we'll use for the design phase

6.3.4 – Component diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Component diagrams

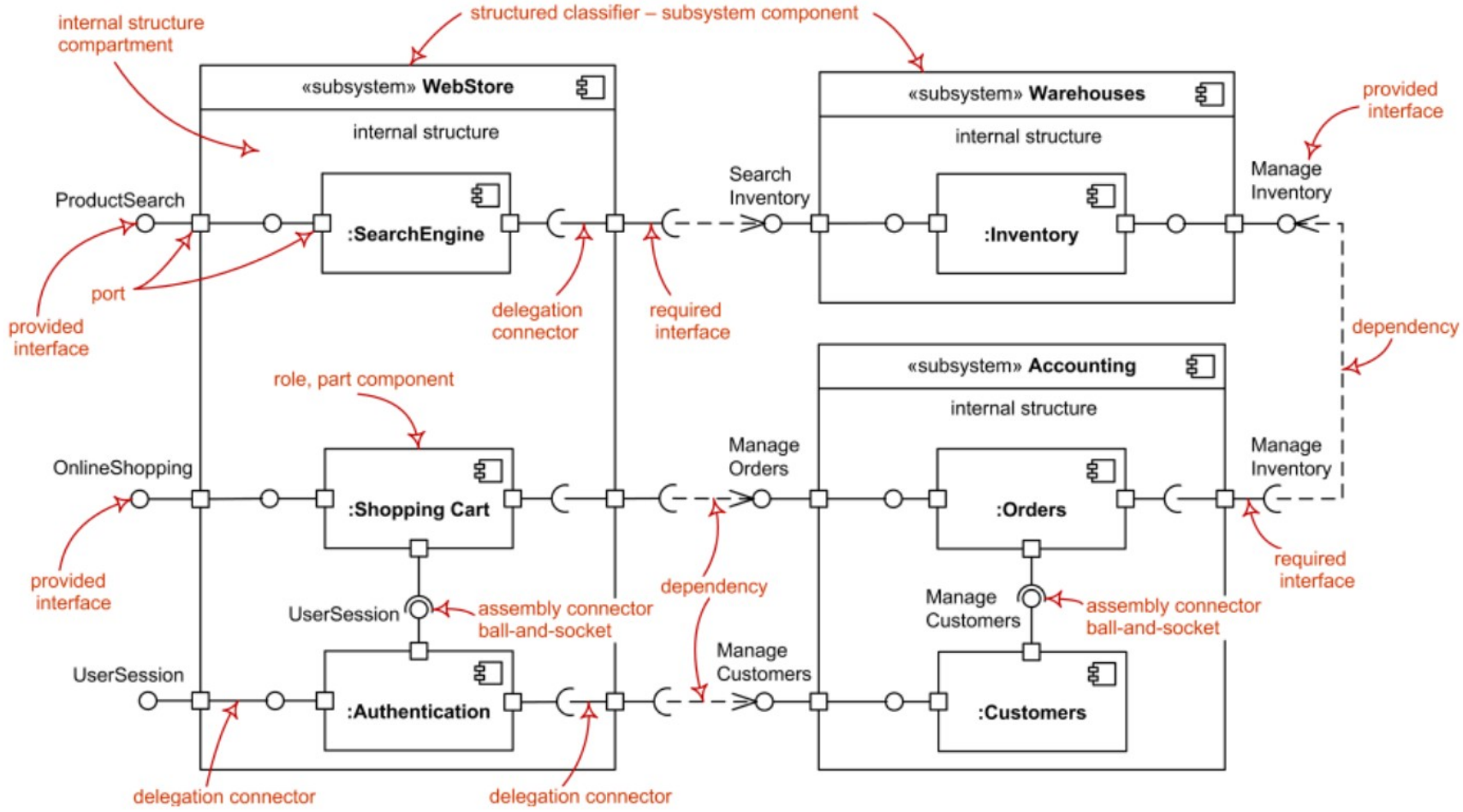
Component diagram *shows components, provided and required interfaces, ports, and relationships between them. This type of diagrams is used in* ***Component-Based Development*** *(****CBD****) to describe systems with* ***Service-Oriented Architecture*** *(****SOA****).*

Elements:

- component,
- provided interface,
- required interface,
- port,
- connectors.

see also: https://www.uml-diagrams.org/component-diagrams-reference.html

Component diagrams



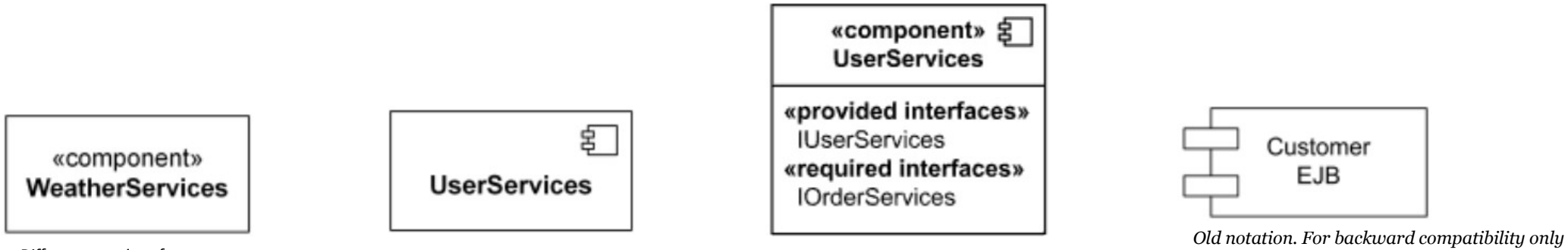
see also: https://www.uml-diagrams.org/component-diagrams-reference.html

Components

Component

A *component* is a **class** representing a modular part of a system with encapsulated content and whose **manifestation** is replaceable within its environment.

A component has its behavior defined in terms of **provided interfaces** and **required interfaces** (potentially exposed via **ports**)

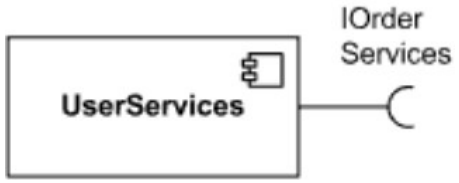


Different notations for components

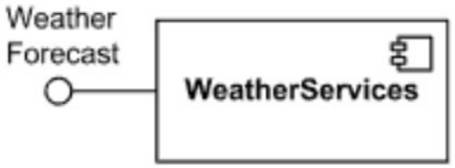
Old notation. For backward compatibility only

see also: https://www.uml-diagrams.org/package-diagrams-reference.html

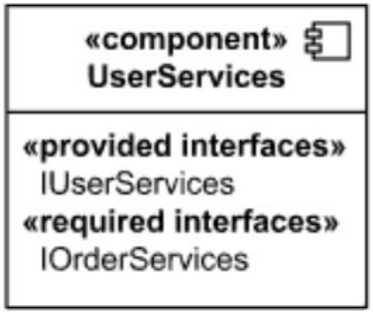
Interfaces



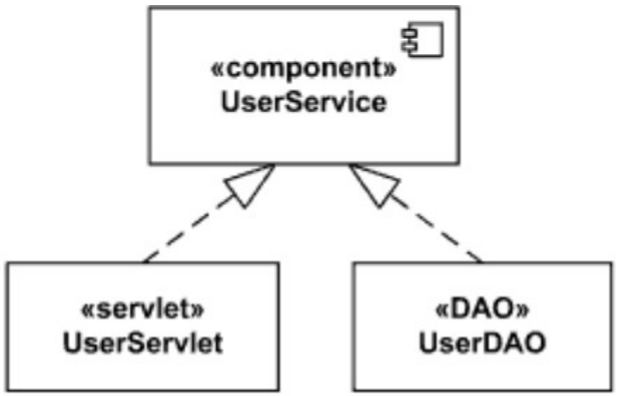
User Services component requires IOrderServices interface.

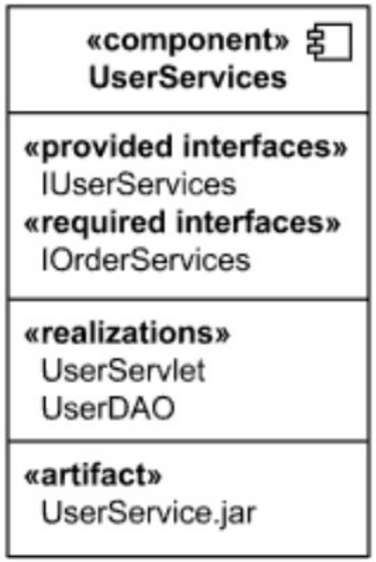


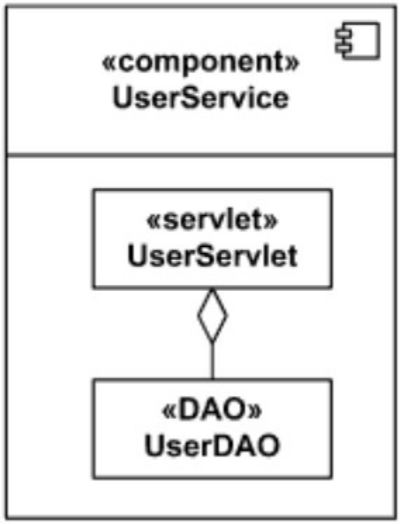
Weather Services component provides (implements) Weather Forecast interface.



Realization

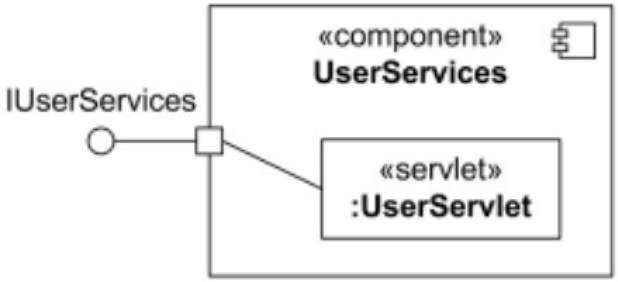




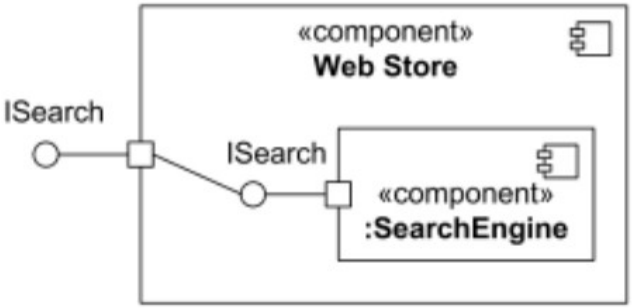


Different notations for:
Component UserService realized by UserServlet and UserDAO..

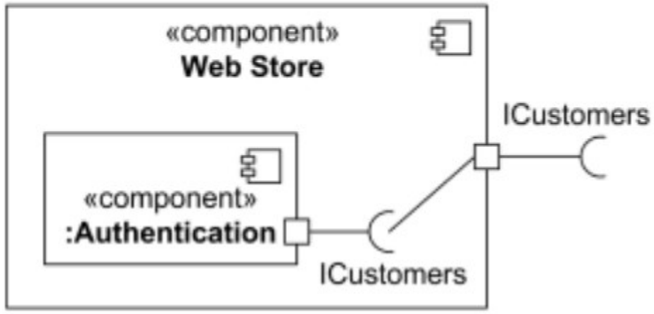
Delegation



Delegation connector from the delegating port to the UserServlet part.

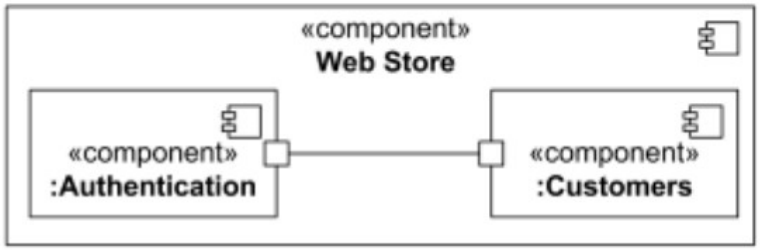


Delegation handled by a single port

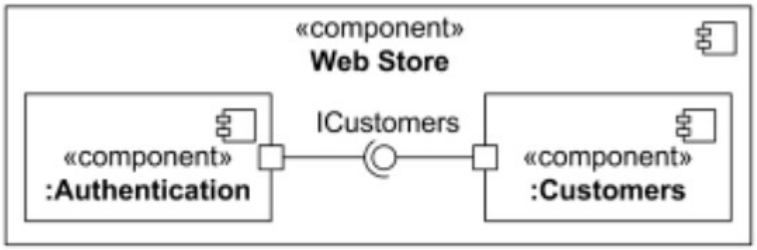


Delegation connector from the simple port of Authentication component to the delegating port.

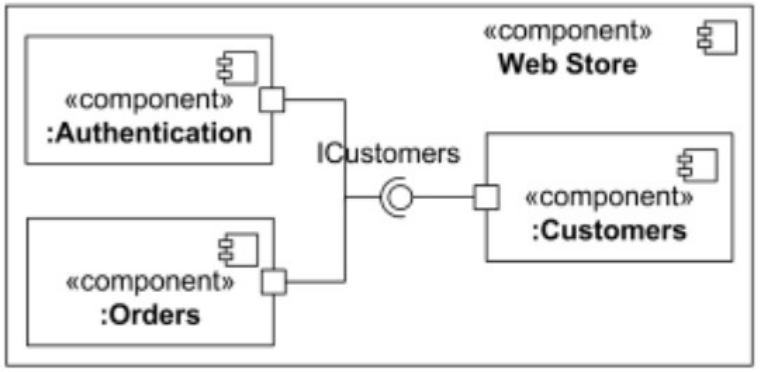
Assembly



Assembly connector between ports of Authentication and Customers components.



Assembly connector between simple ports of Authentication and Customers components.



Assembly connector that assembles three parts.

Component diagram examples

See https://www.uml-diagrams.org/component-diagrams-examples.html

Software Engineering

Part 6 – The Unified Modeling Language

6.3 – diagrams we'll use for the design phase

6.3.5 – Deployment diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Deployment diagrams

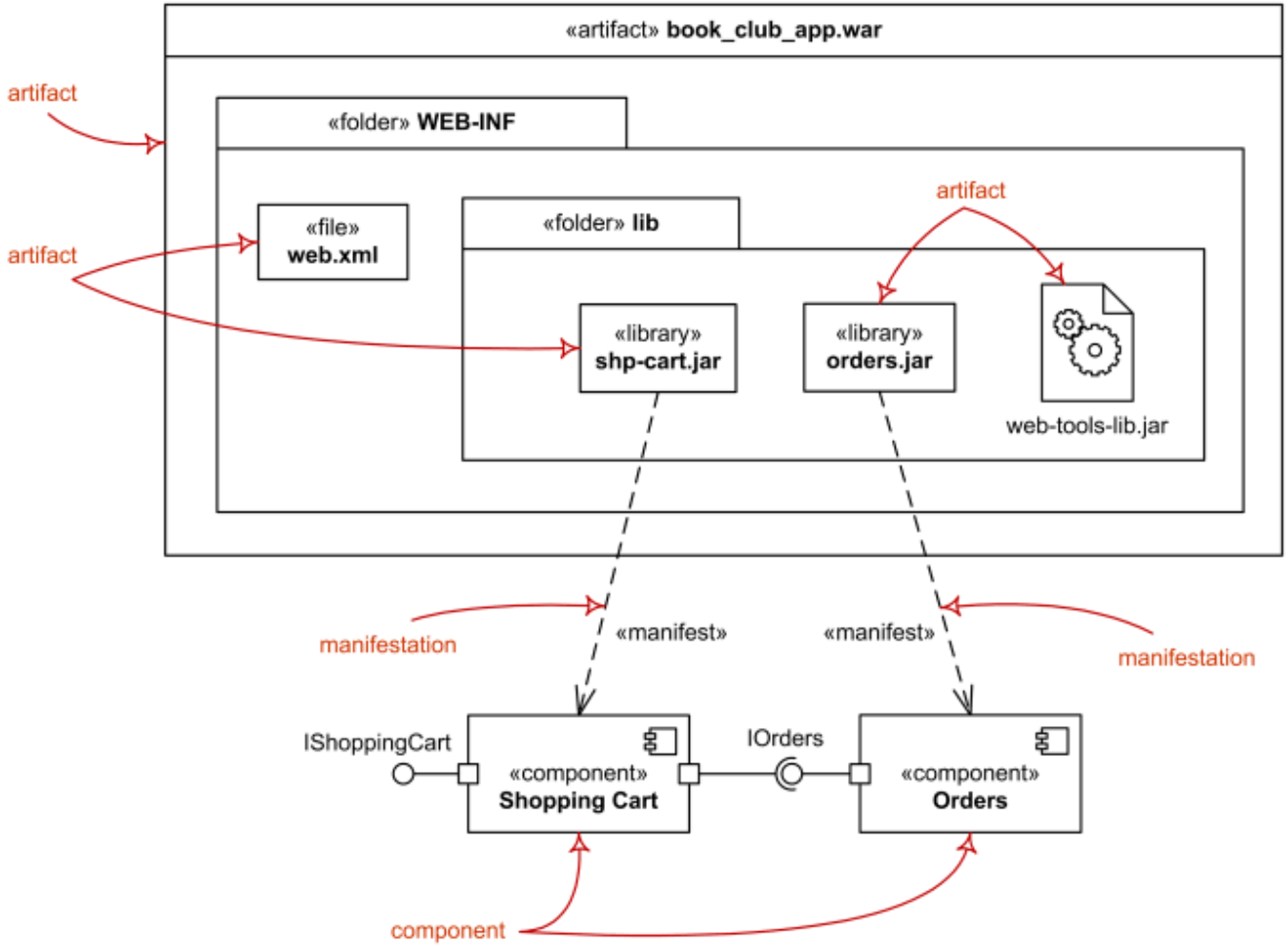
Deployment Deployment diagram *is a* ***structure diagram*** *which shows architecture of the system as deployment (distribution) of software artifacts to deployment targets.*

Some common types of deployment diagrams are:

- **Implementation (manifestation) of components by artifacts**,
- **Specification level deployment diagram**,
- **Instance level deployment diagram**,
- **Network architecture of the system**.

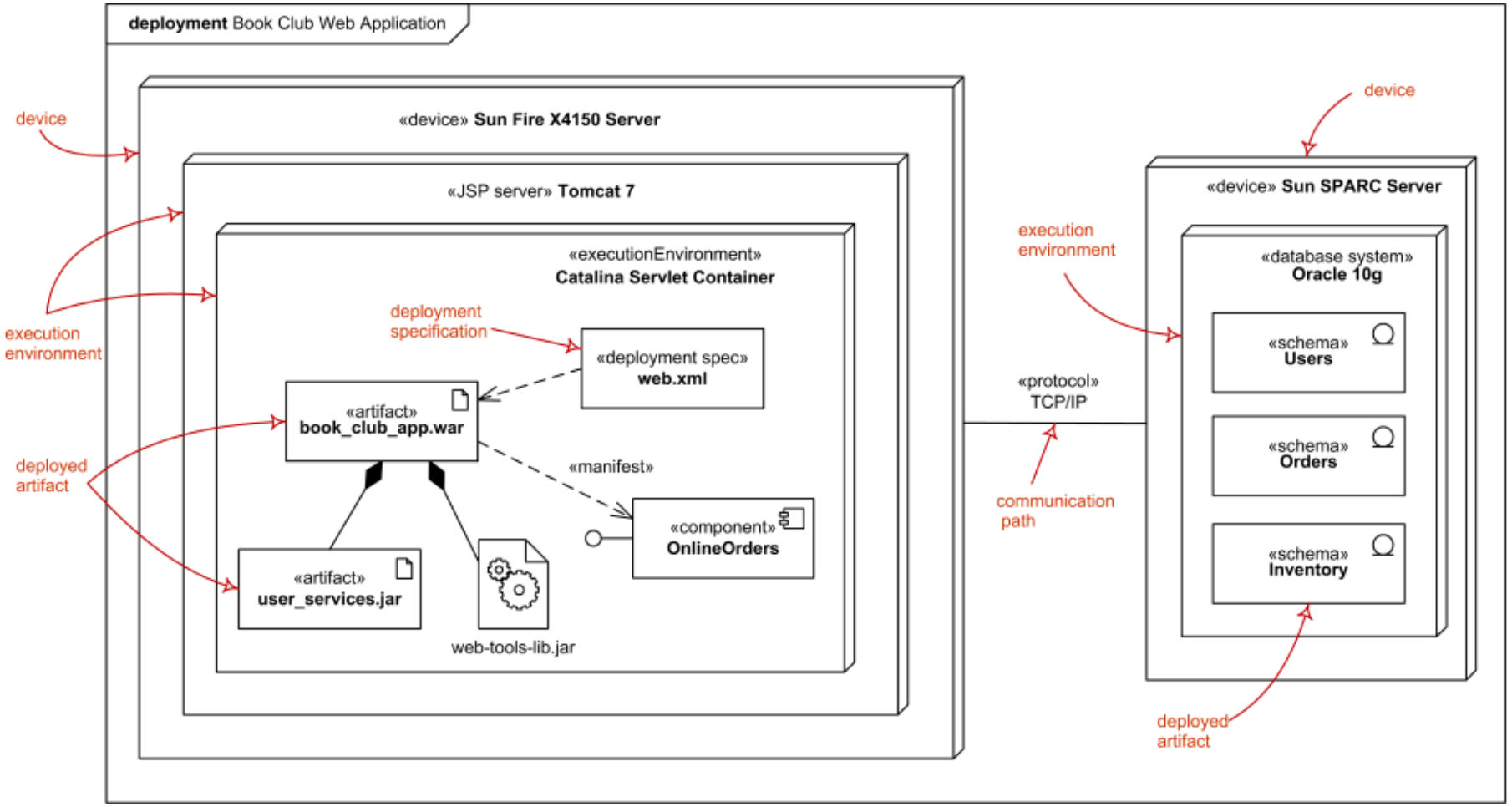
see also: https://www.uml-diagrams.org/deployment-diagrams-reference.html

Manifestation of Components by Artifacts



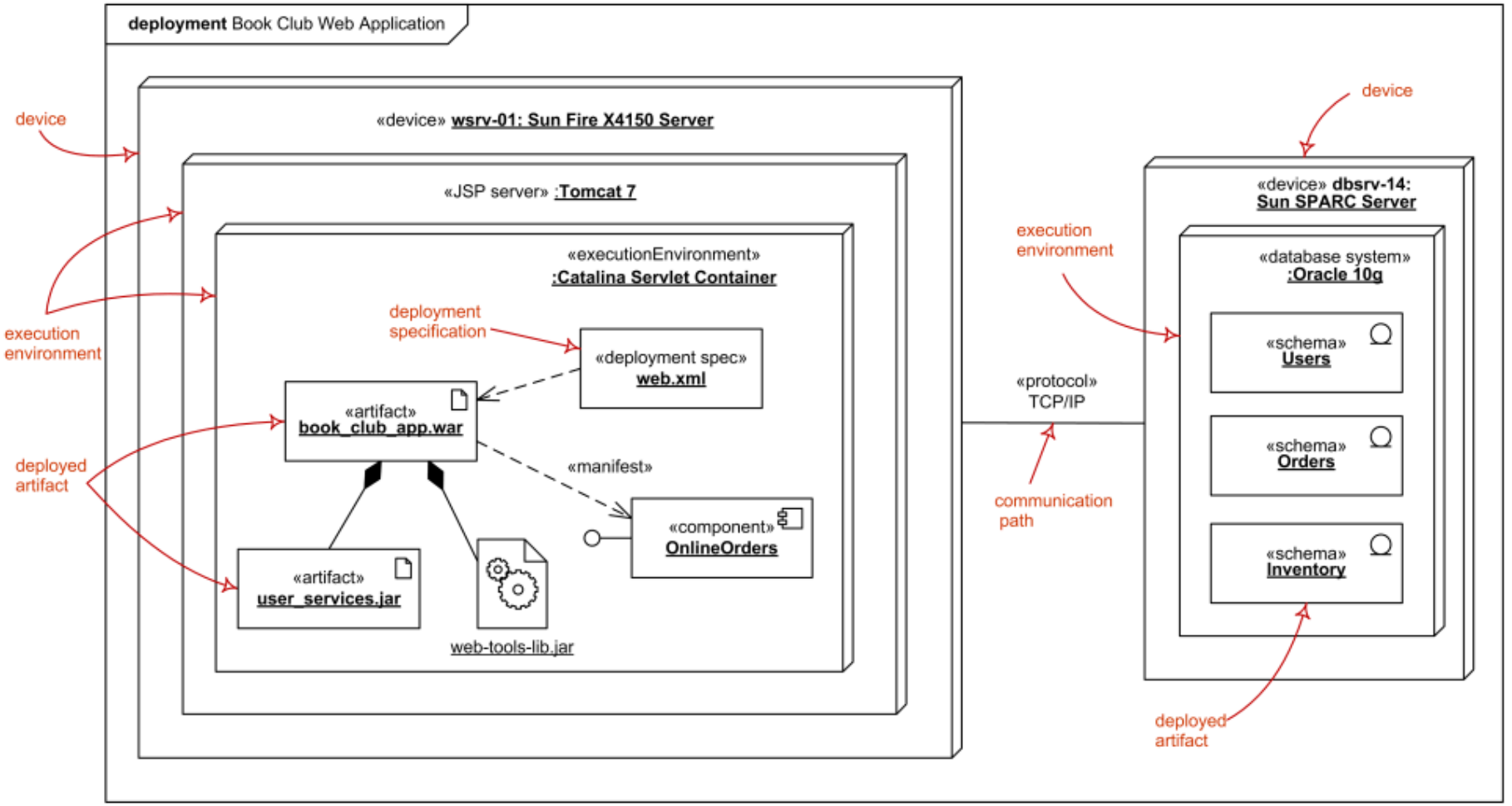
see also: https://www.uml-diagrams.org/component-diagrams-reference.html

Specification Level Deployment Diagram



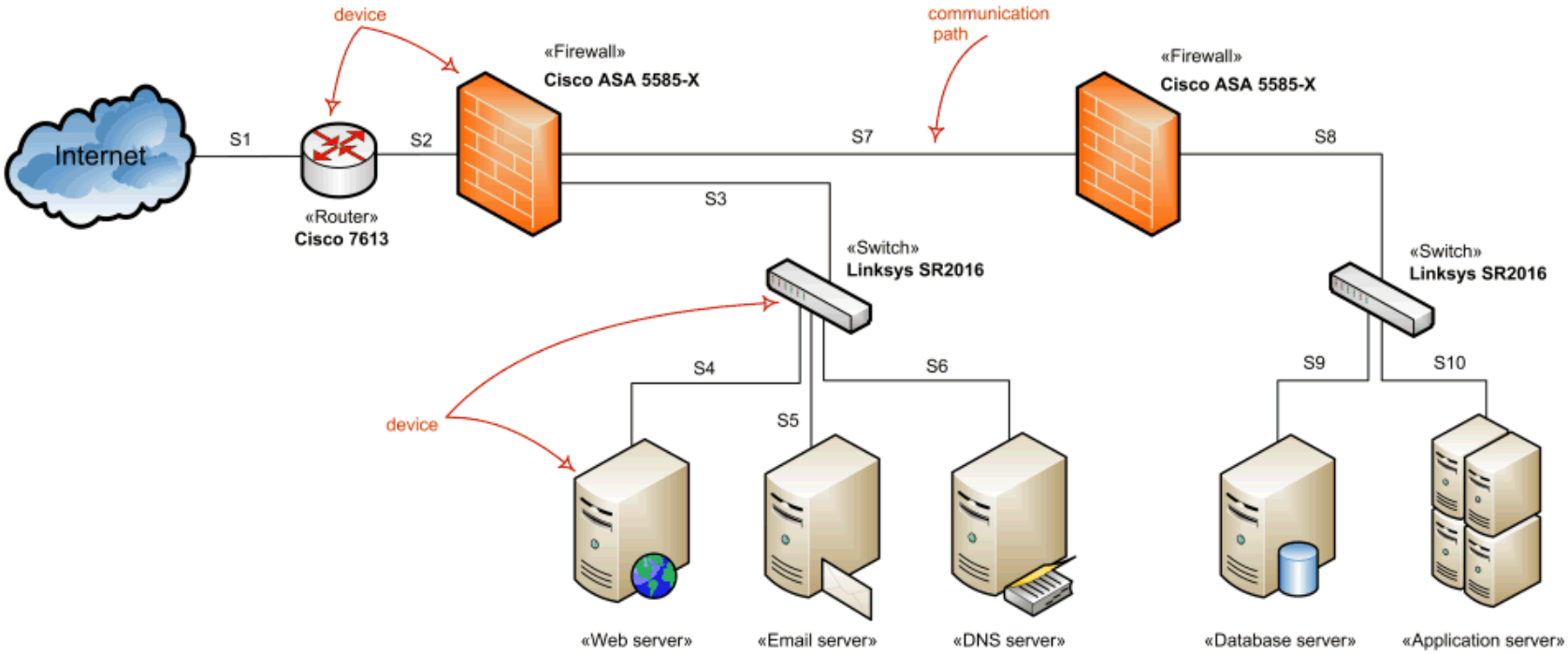
see also: https://www.uml-diagrams.org/component-diagrams-reference.html

Instance Level Deployment Diagram



see also: https://www.uml-diagrams.org/component-diagrams-reference.html

Network Architecture Diagrams



see also: https://www.uml-diagrams.org/component-diagrams-reference.html

Deployment diagram examples

See https://www.uml-diagrams.org/deployment-diagrams-examples.html

Software Engineering

Part 6 – The Unified Modeling Language

6.3 – diagrams we'll use for the design phase

6.3.6 – Sequence diagrams

ICM – Computer Science Major – Software Engineering - Part 1: Introduction
M1 Cyber Physical and Social Systems – CPS2 engineering and development - Part 3: Software Engineering
Guillaume Muller
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng/

Sequence diagrams

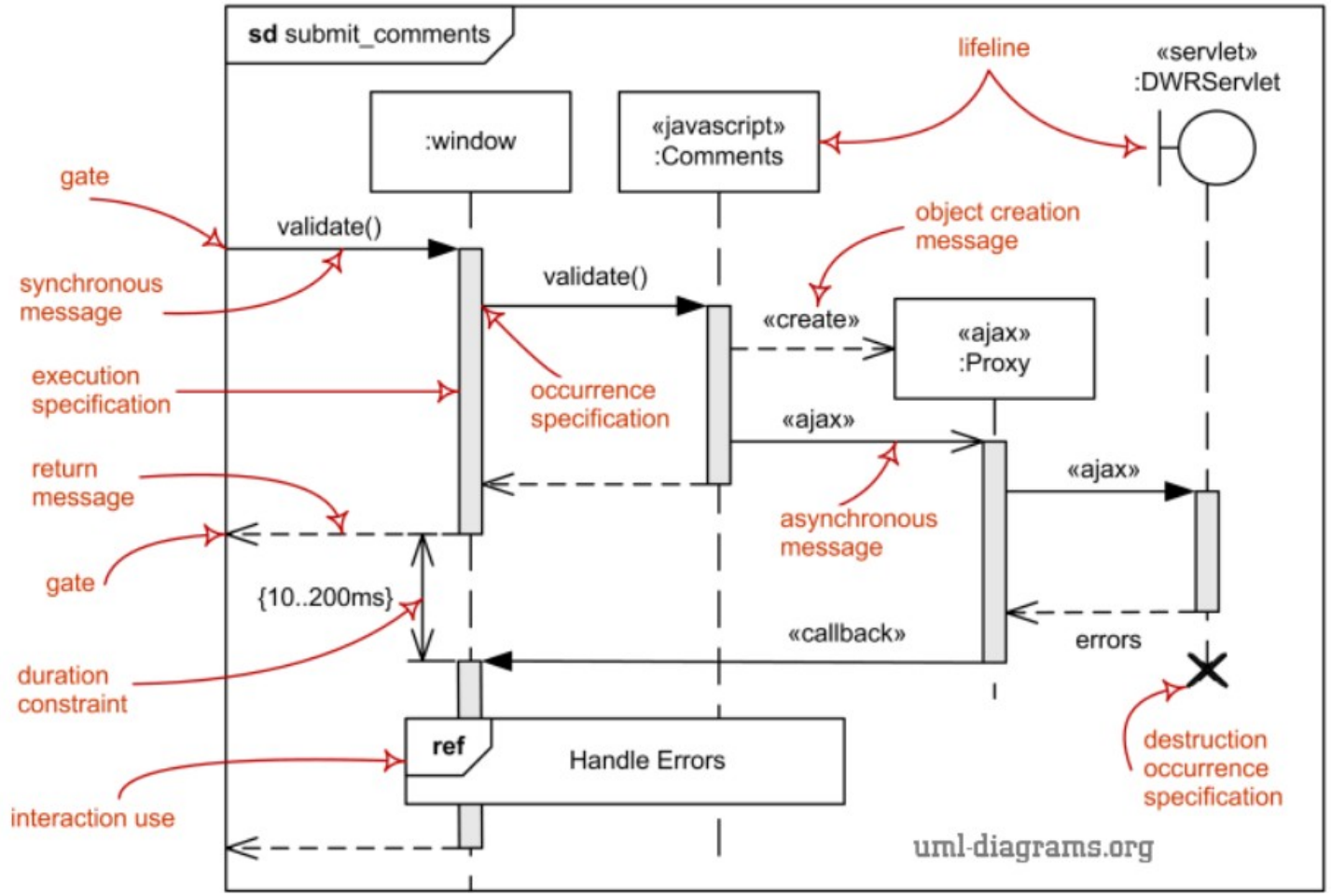
Sequence diagram *is the most common kind of* ***interaction diagram****, which focuses on the* ***message*** *interchange between a number of* ***lifelines****.*

Types of nodes

- **lifeline**,
- **execution specification**,
- **message**,
- **combined fragment**,
- **interaction use**,
- **state invariant**,
- **continuation**,
- **destruction occurrence**.

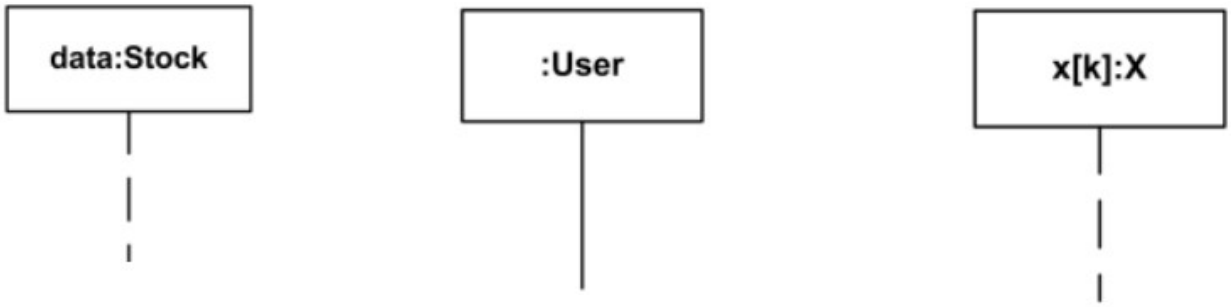
see also: https://www.uml-diagrams.org/sequence-diagrams-reference.html

Sequence diagrams



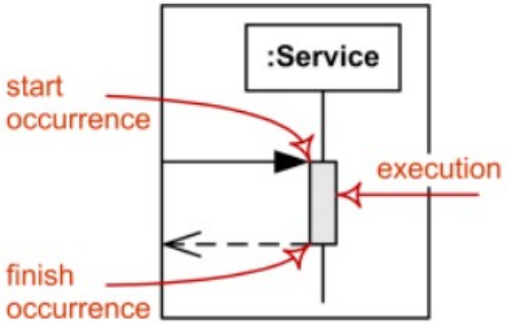
see also: https://www.uml-diagrams.org/sequence-diagrams-reference.html

Lifeline

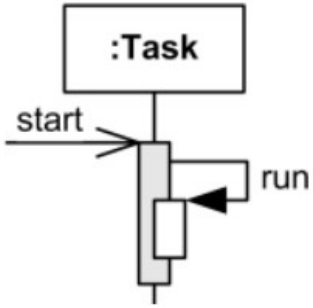


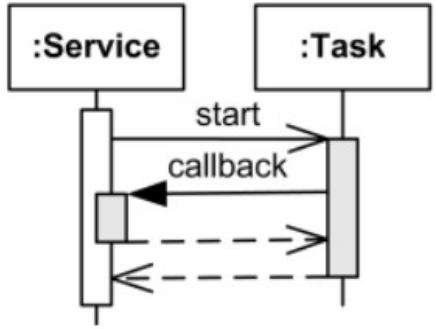
see also: https://www.uml-diagrams.org/sequence-diagrams-reference.html

Execution

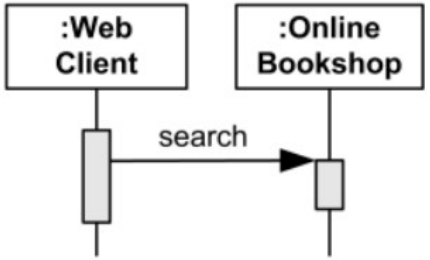

:Service
start
occurrence
execution
finish
occurrence


:Web
Client
search

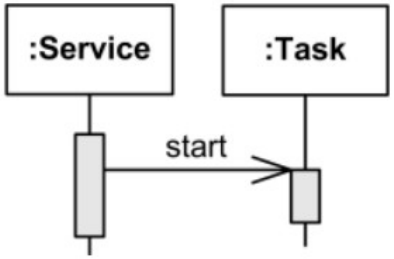

:Task
start
run


:Service
:Task
start
callback

Calls

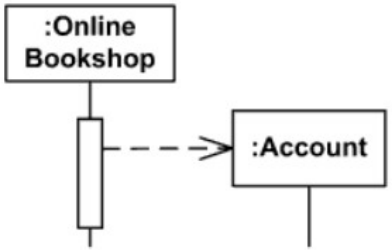


Synchronous
Web Client searches Online Bookshop and waits for results.

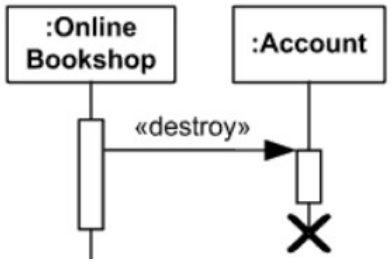


Asynchronous
Service starts Task and proceeds in parallel without waiting.

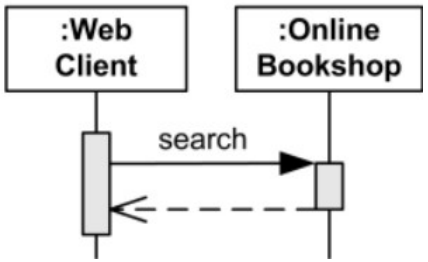
Messages



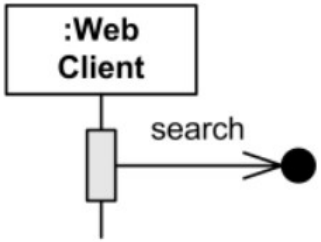
Create



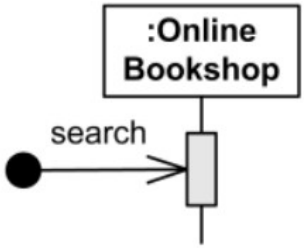
Delete



Reply



Lost

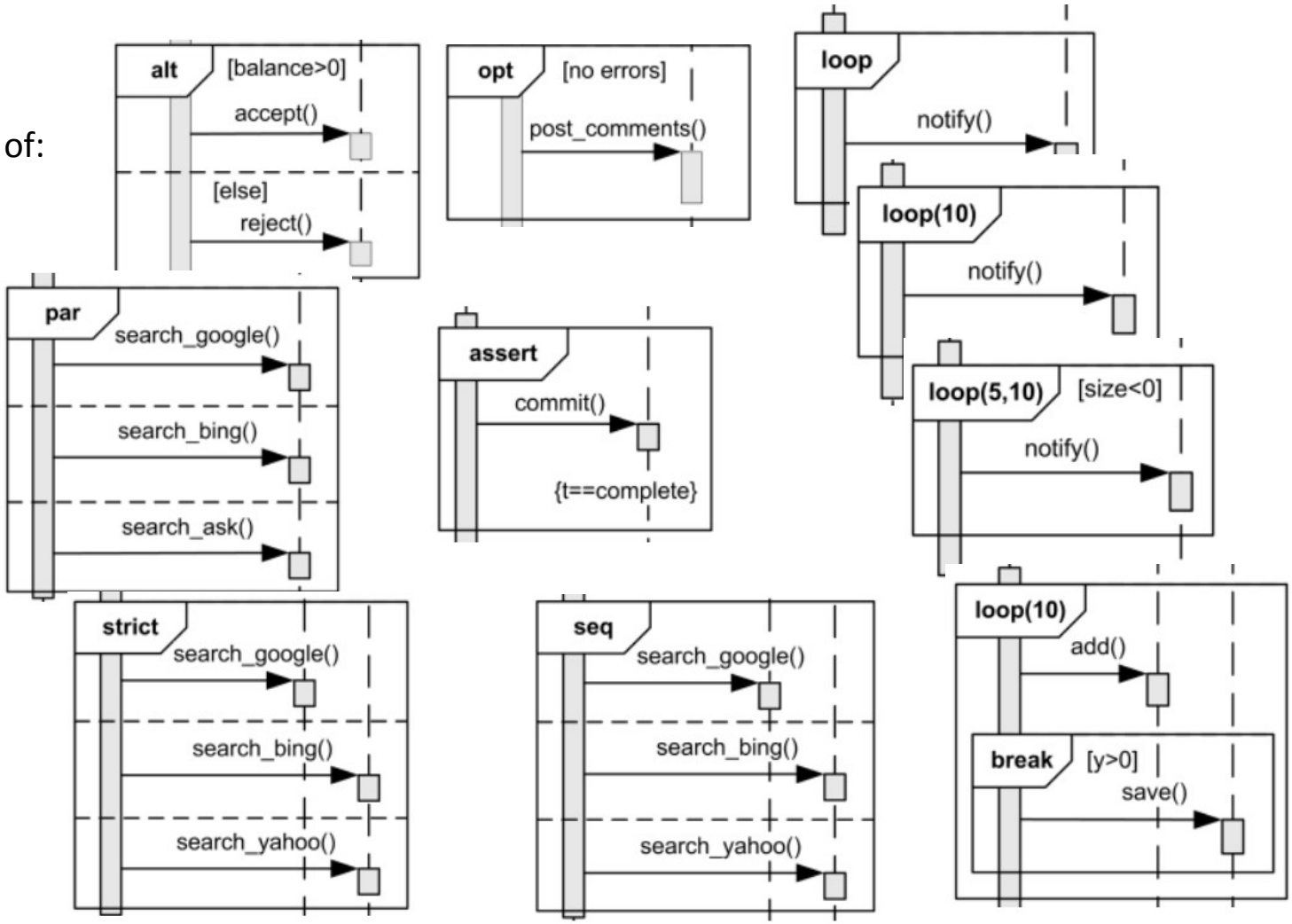


Found

Combined fragment with interaction operator

Interaction operator could be one of:

- **alt** - alternatives
- **opt** - option
- **loop** - iteration
- **break** - break
- **par** - parallel
- **strict** - strict sequencing
- **seq** - weak sequencing
- **critical** - critical region
- **ignore** - ignore
- **consider** - consider
- **assert** - assertion
- **neg** - negative



Sequence diagram examples

See https://www.uml-diagrams.org/sequence-diagrams-examples.html

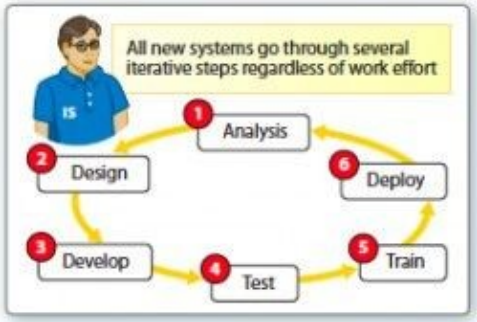
Software Engineering

Syllabus and Course Organization

ICM – Computer Science Major – Course unit on Software Engineering
M1 Cyber Physical and Social Systems – Course unit on System Modeling
Maxime Lefrançois https://maxime.lefrancois.info
Course unit URL: https://ci.mines-stetienne.fr/cps2/course/softeng

What are the Benefits and Steps in a System Development Life Cycle?

The development process begins when a need is referred to the Information Services help desk. This allows adequate planning and prioritization with other related or independent activities.



Typical Service Desk Requests

Configuration Management Database (CMDB)

Accurate records support system maintenance if problems arise

Consult

Consult

Scope of Work

Confirm

Confirm

Confirm

System Support Documents

Change Management
Considers users and others who may be impacted

Functional Requirements

Release Schedule

Release Management
Testing for suitability and success

Configuration Management
Ensures a smooth transition to new processes

ITIL® is the most widely accepted approach to IT service management in the world. ITIL provides a cohesive set of best practice, drawn from the public and private sectors internationally. www.itil-officialsite.com/

Course objectives

The objective is to know some core concepts, processes, and models, that are useful to comprehend common issues and problems in the engineering of IT systems that result from the integration of existing systems (systems of systems), operate in distributed environments Web, IoT, Cloud, ...

Topics:

- Software Engineering Introduction
  - Motivations and Definitions
- Software Requirements
  - Definitions
  - Elicitation, analysis, specification, validation
- Software Quality
  - The ISO/IEC 25010:2011 System and software quality models
- Software Engineering process
  - Software development life cycles
  - Agile methodologies
  - The Scrum methodology
  - DevOps methodologies
- Software Engineering Models
  - The Unified Modeling Language

Positioning wrt Teaching Module

M1 CPS2 students

Teaching Module on **CPS2 engineering and development**

1. Everything from the command line (ECL: weight 18)
2. Technological foundations of software development (TFSD: weight 20)
3. **Software Engineering** (Softeng: weight 12)

ICM students

Teaching Module on **CPS2 software engineering**

1. **Introduction to Software Engineering**
2. Software development best practices
3. Software architectures

Course organization

See https://ci.mines-stetienne.fr/cps2/course/softeng

Grading policy for this part

Behavior (B, -2 to +2)
Written Exam (WE, 0 to 20)
Grade = WE + B