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

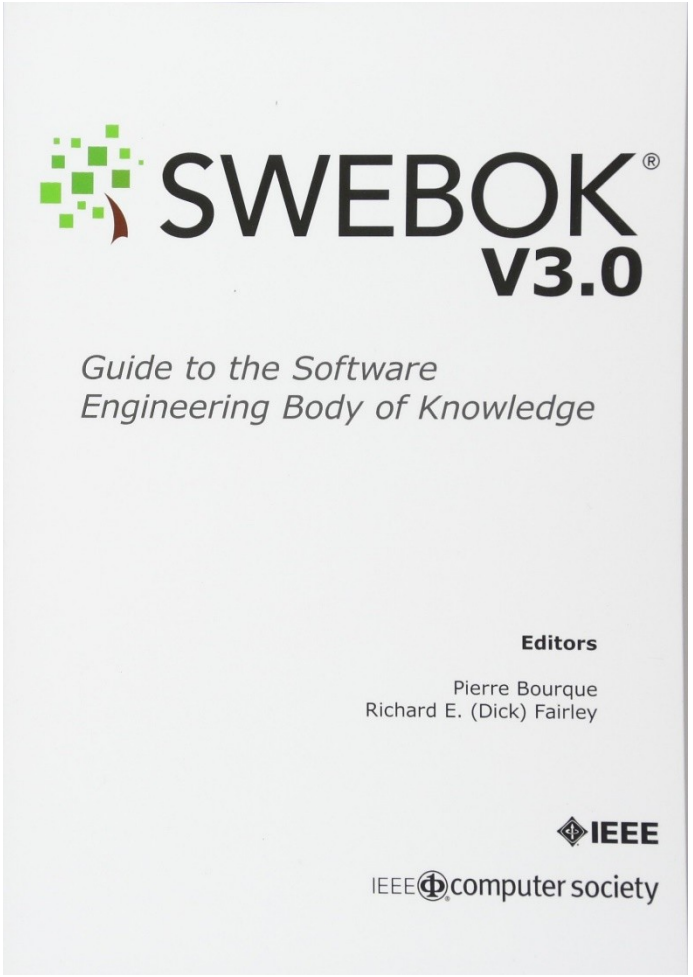
… called for the development of Software Engineering

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

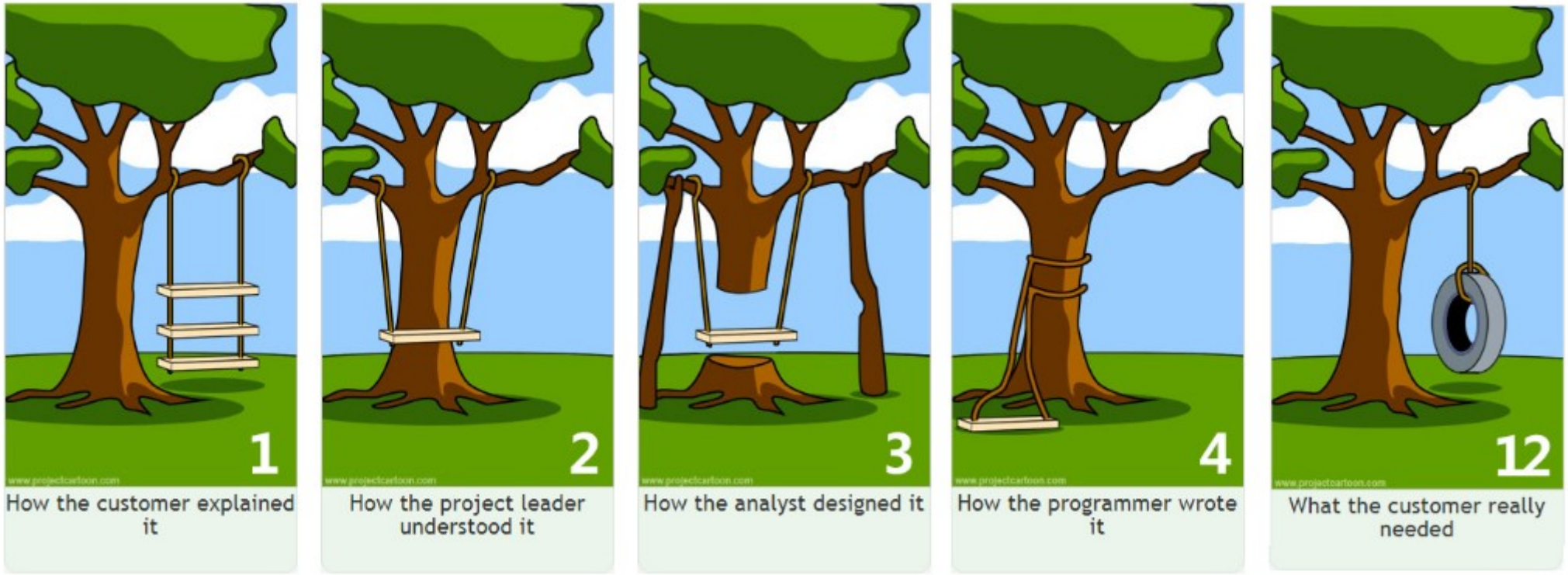
ISO/IEC TR 19759:2015 (v1 in 2005)



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, …

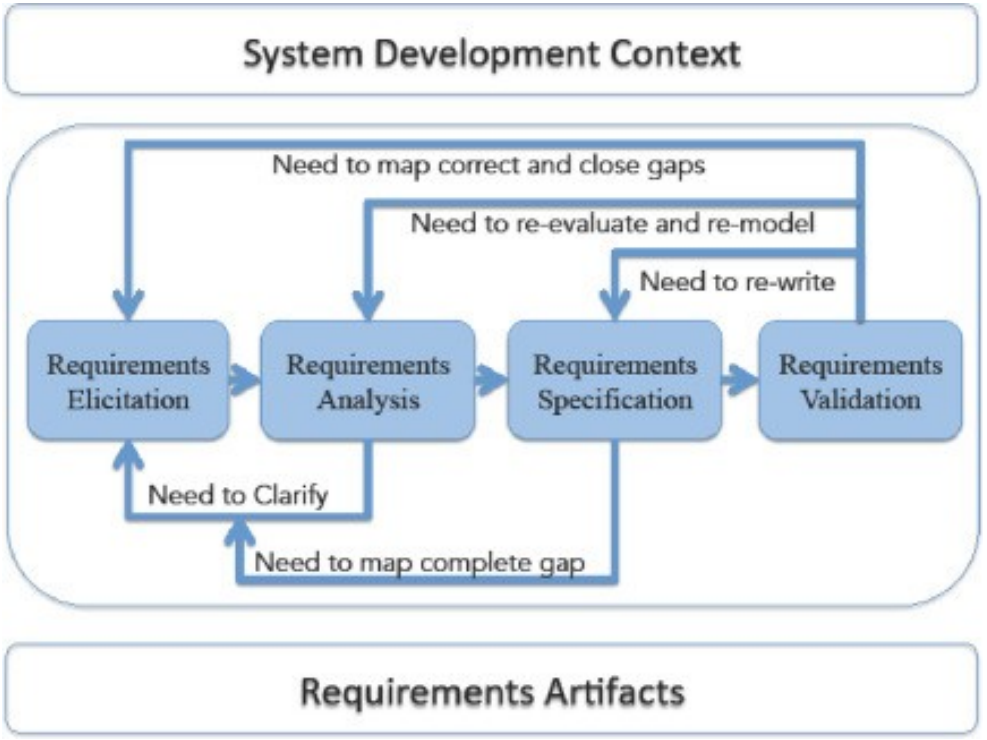
see also blog post on functional vs nonfunctional requirements https://theappsolutions.com/blog/development/functional-vs-non-functional-requirements/

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

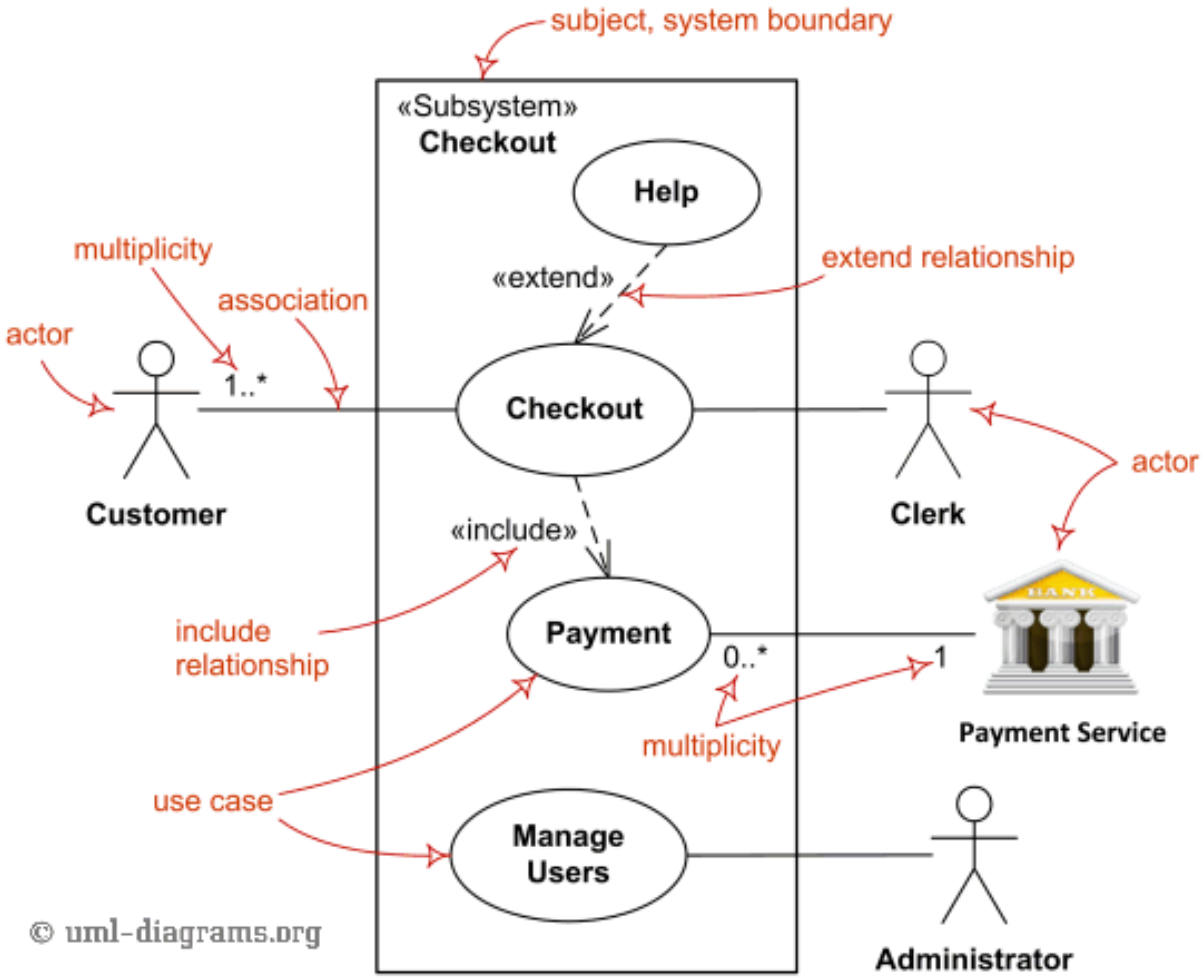
- functional/nonfunctional
- source (e.g., user, regulation)
- on the product or the process
- priority (mandatory, highly desirable, desirable, optional)
- scope (global, narrow)
- Volatility/stability

Requirements process – ii. analysis

Conceptual models

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

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



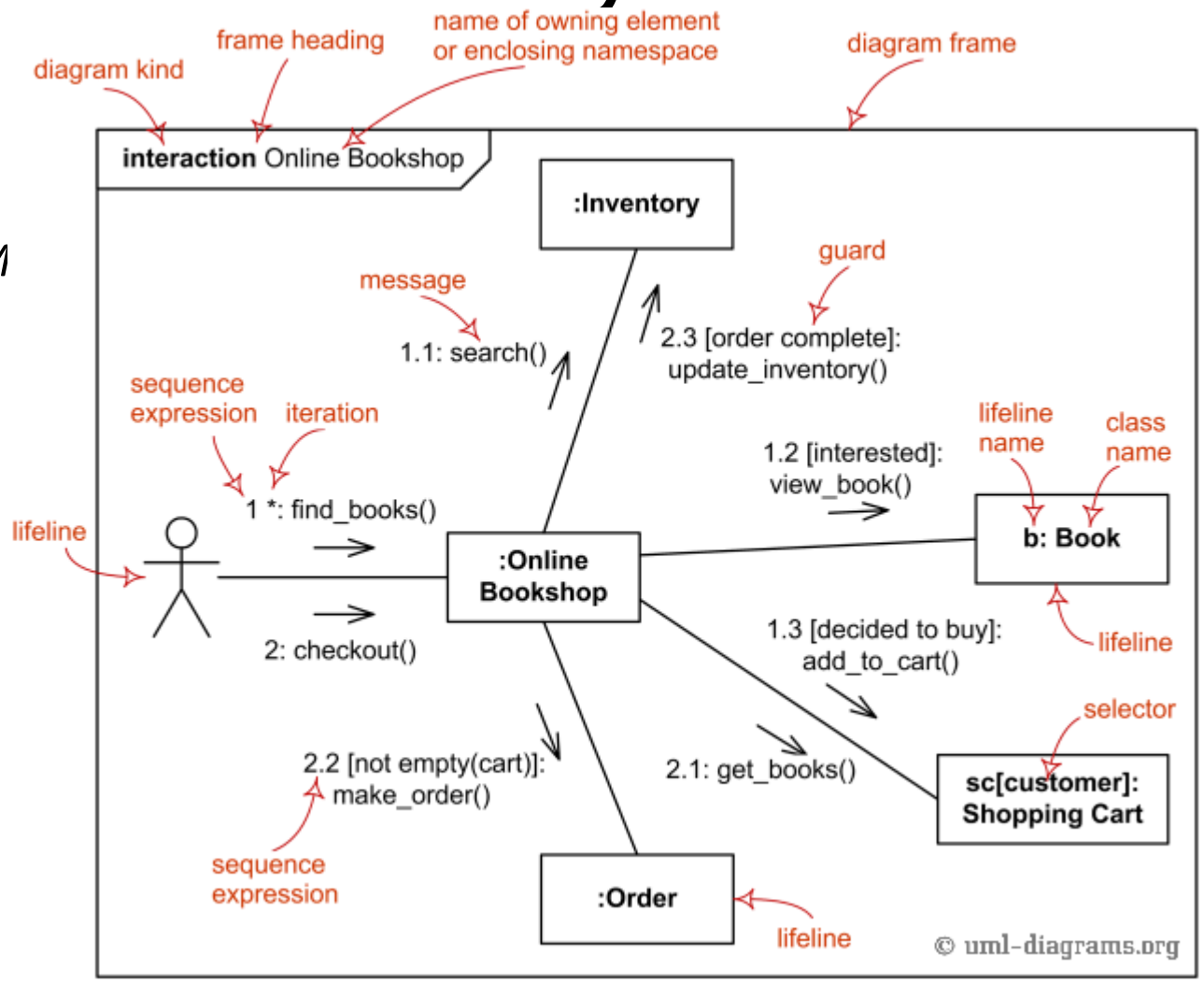
exemple: UML Use case diagrams
source: https://www.uml-diagrams.org/use-case-diagrams.html

Requirements process – ii. analysis

Conceptual models

For example with the *Unified Modeling Language (UM*

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



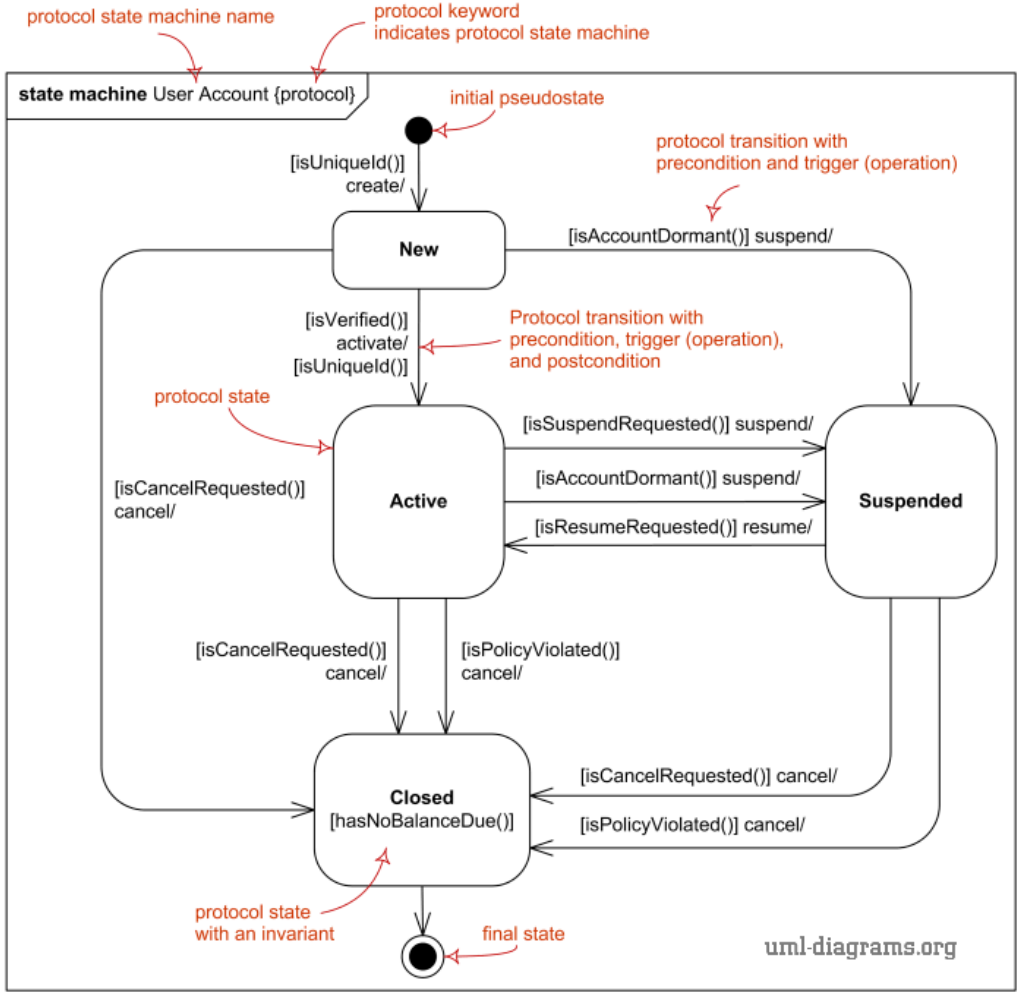
exemple: UML Communication diagrams
source: https://www.uml-diagrams.org/communication-diagrams.html

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)

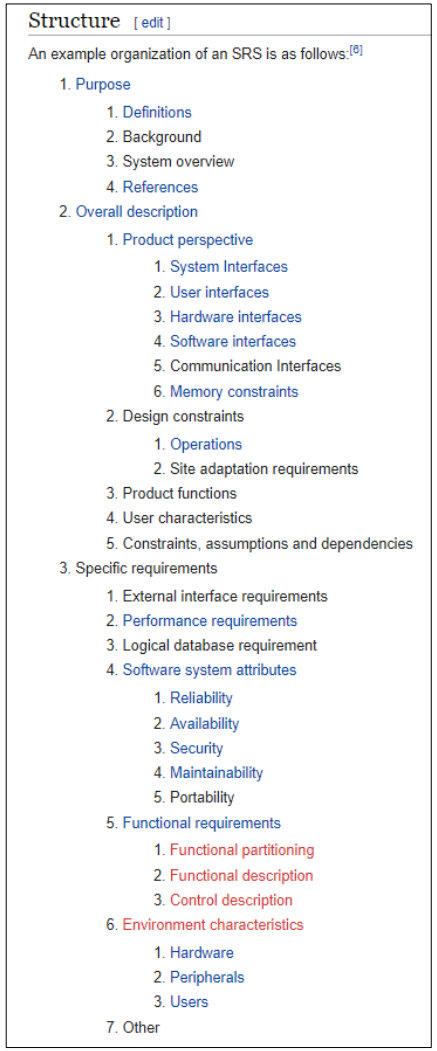
"production of a document that can be systematically reviewed, evaluated, and approved"

— 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

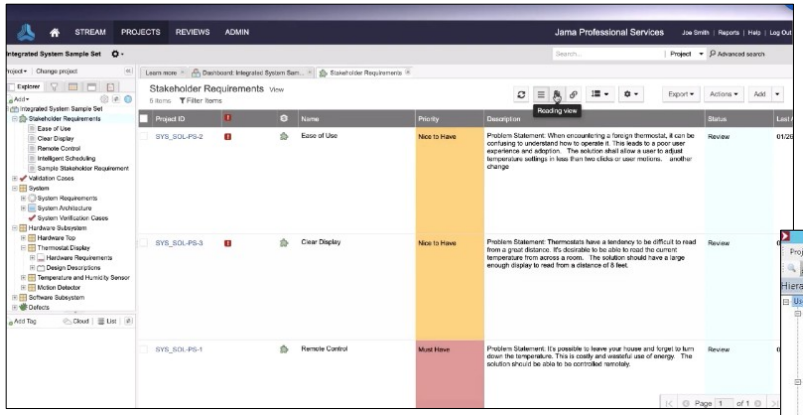
Techniques for requirement validation

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

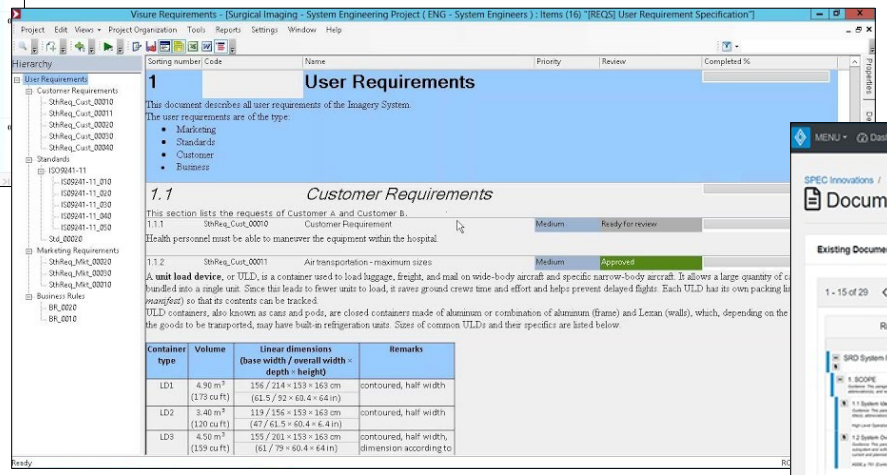
see for example https://en.wikipedia.org/wiki/Formal_methods

Software requirements tools

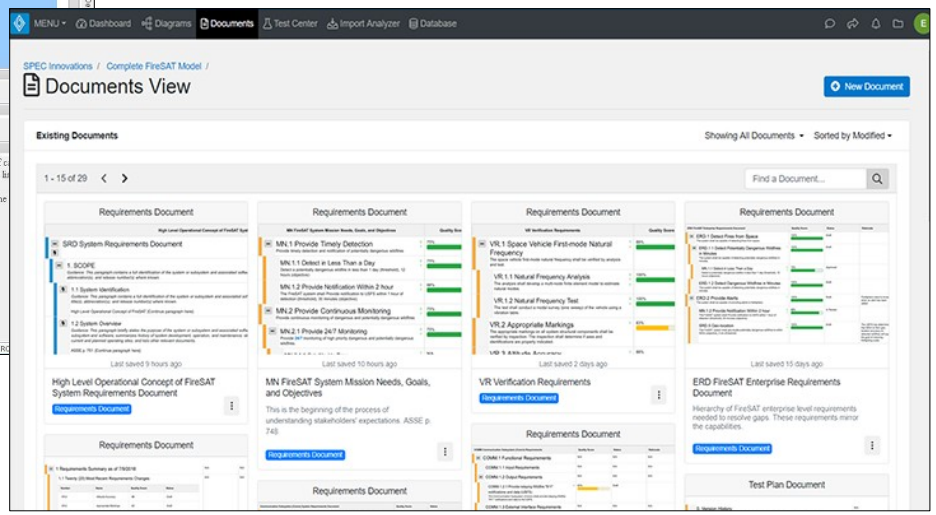
almost exclusively commercial tools



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

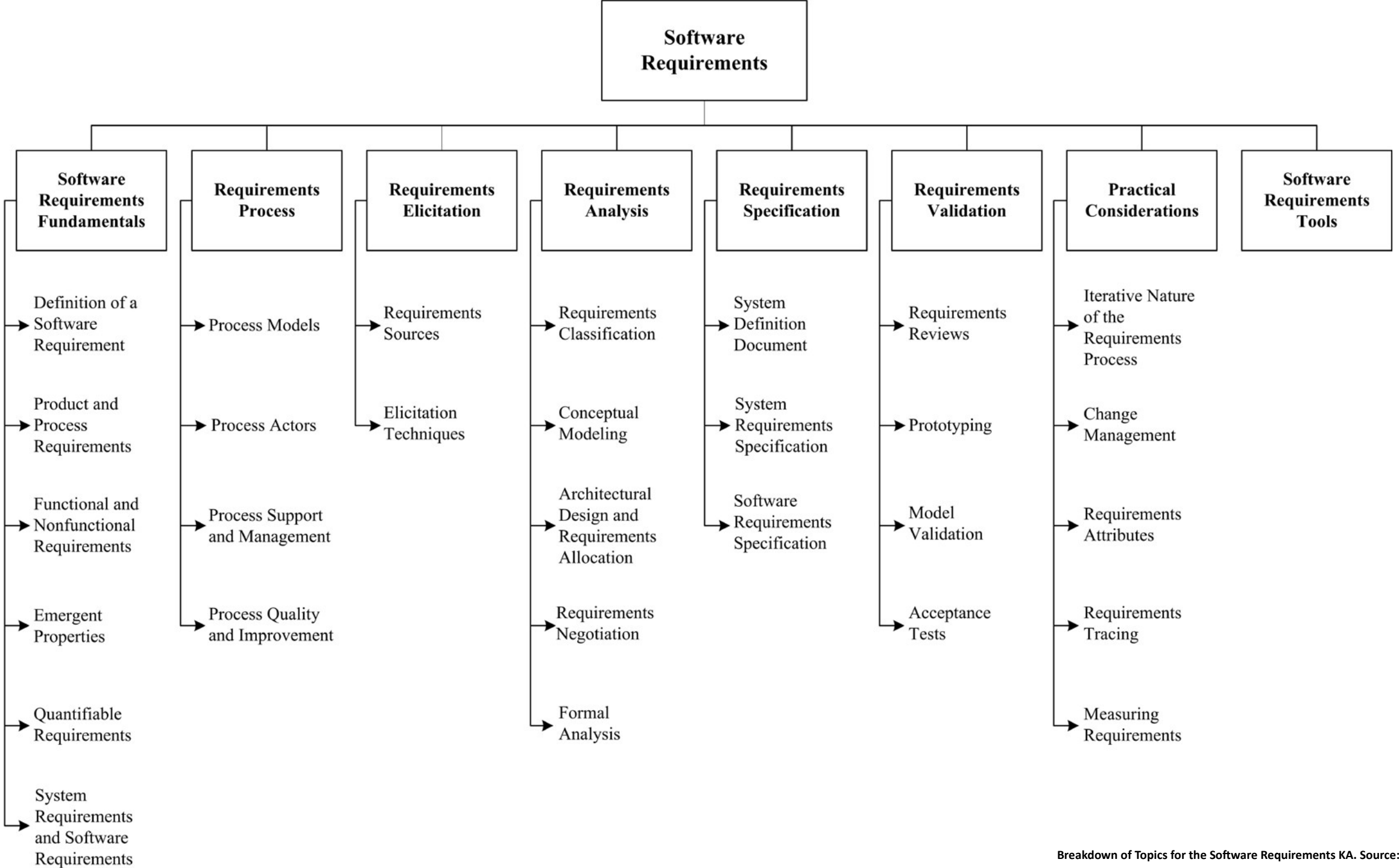


https://visuresolutions.com/



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)

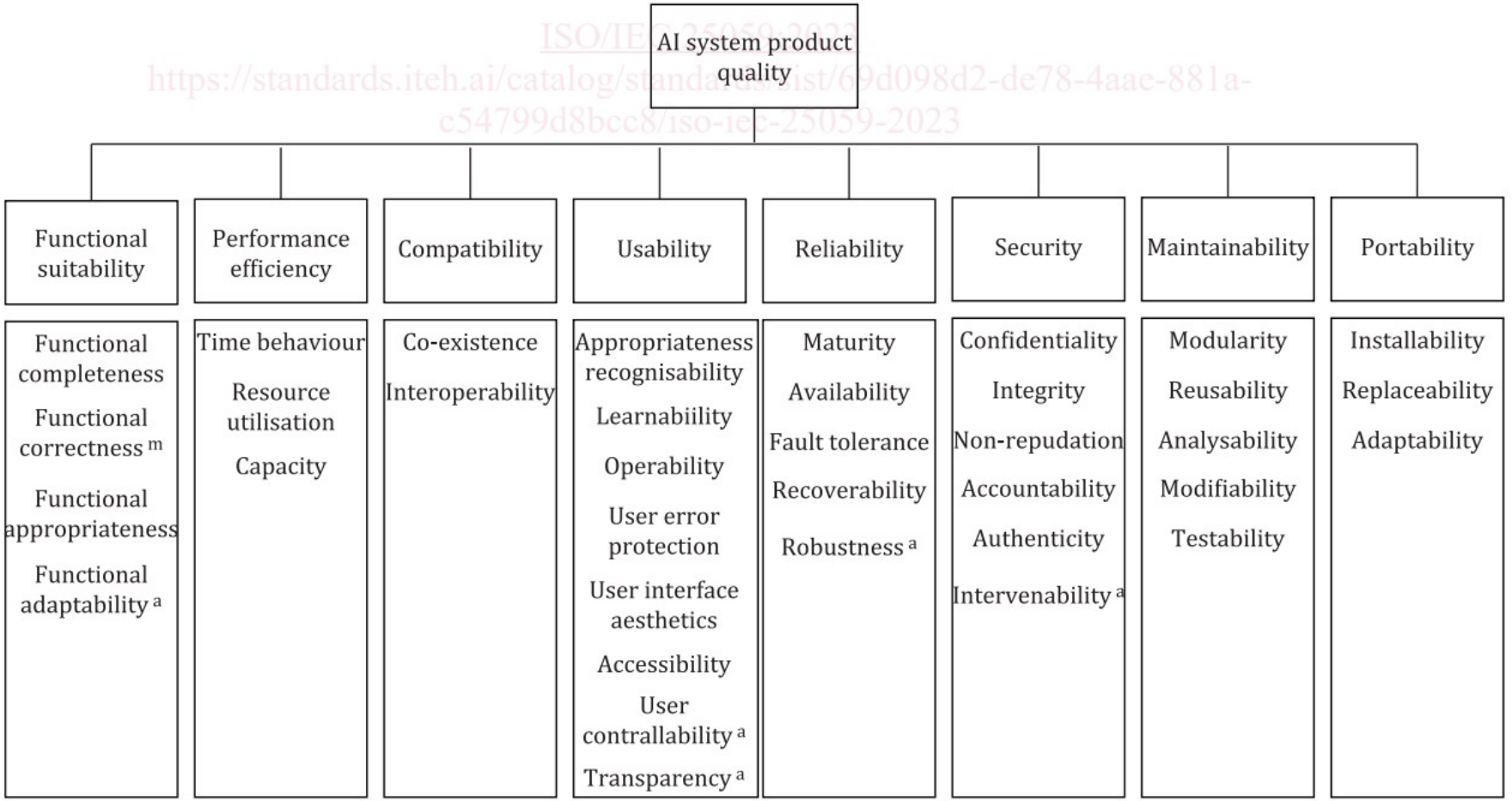
https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

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.