

# Software Development Life Cycle Models

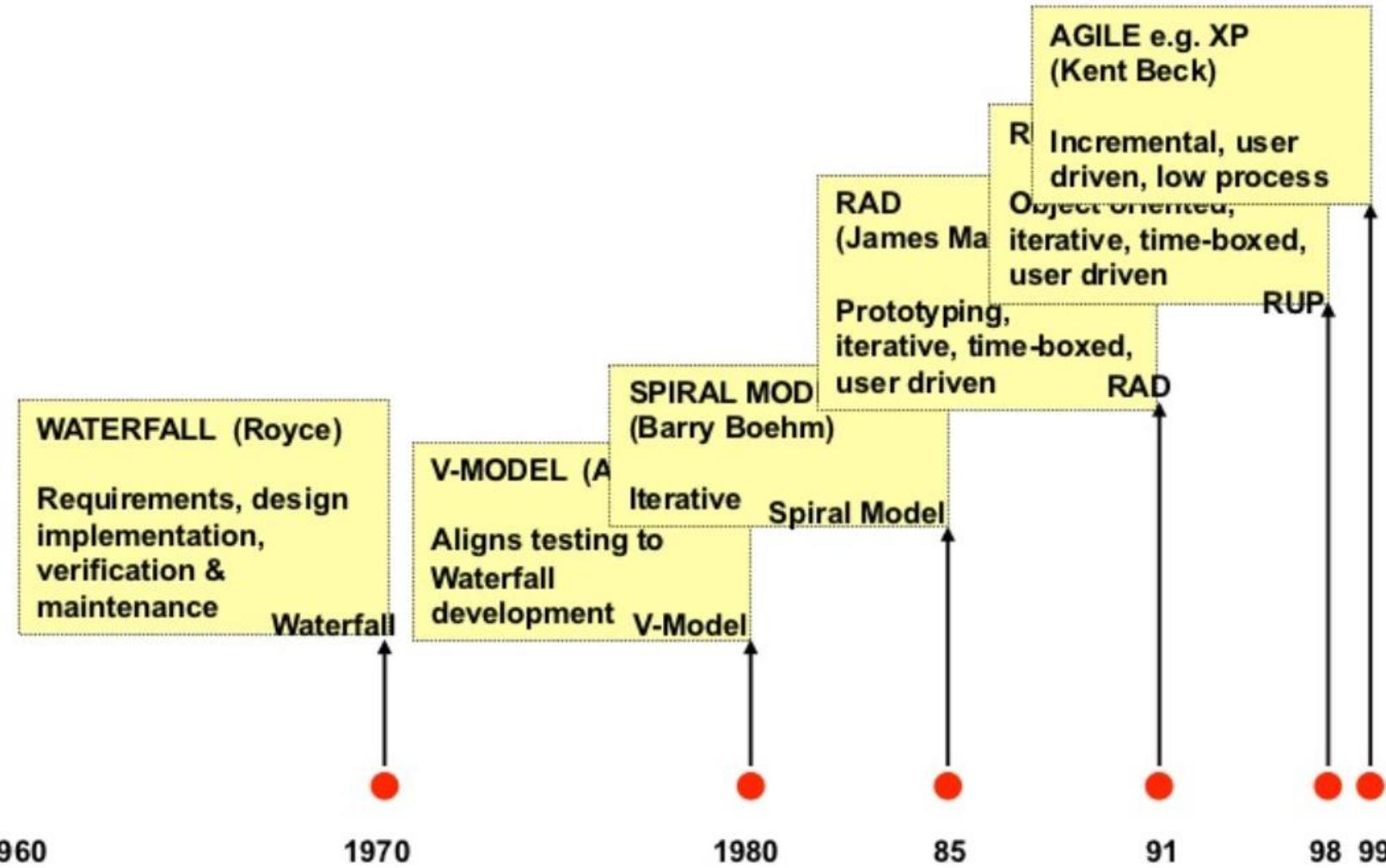
dr. Asta Slotkienė

[asta.slotkiene@vgtu.lt](mailto:asta.slotkiene@vgtu.lt)

# The Six Phases of a (Big) Project

- |                                   |                                |
|-----------------------------------|--------------------------------|
| 1. Enthusiasm,                    | 1. Entuziazmas,                |
| 2. Disillusionment,               | 2. Nusivylimas,                |
| 3. Panic and hysteria             | 3. Panika ir isterija          |
| 4. Hunt for the guilty,           | 4. Ieškome kaltų               |
| 5. Punishment of the<br>innocent, | 5. Bausmė nekaltiems,          |
| 6. Reward for the<br>uninvolved.  | 6. Atlygis už<br>nedalyvavimą😊 |

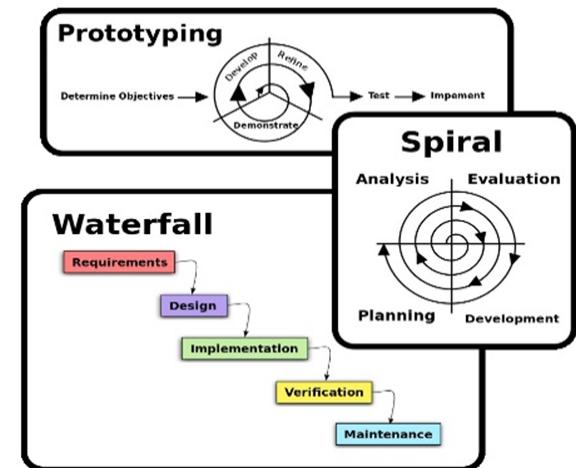
# History of Software process model



# Software process models

A framework that describes the activities performed at each stage of a software development project.

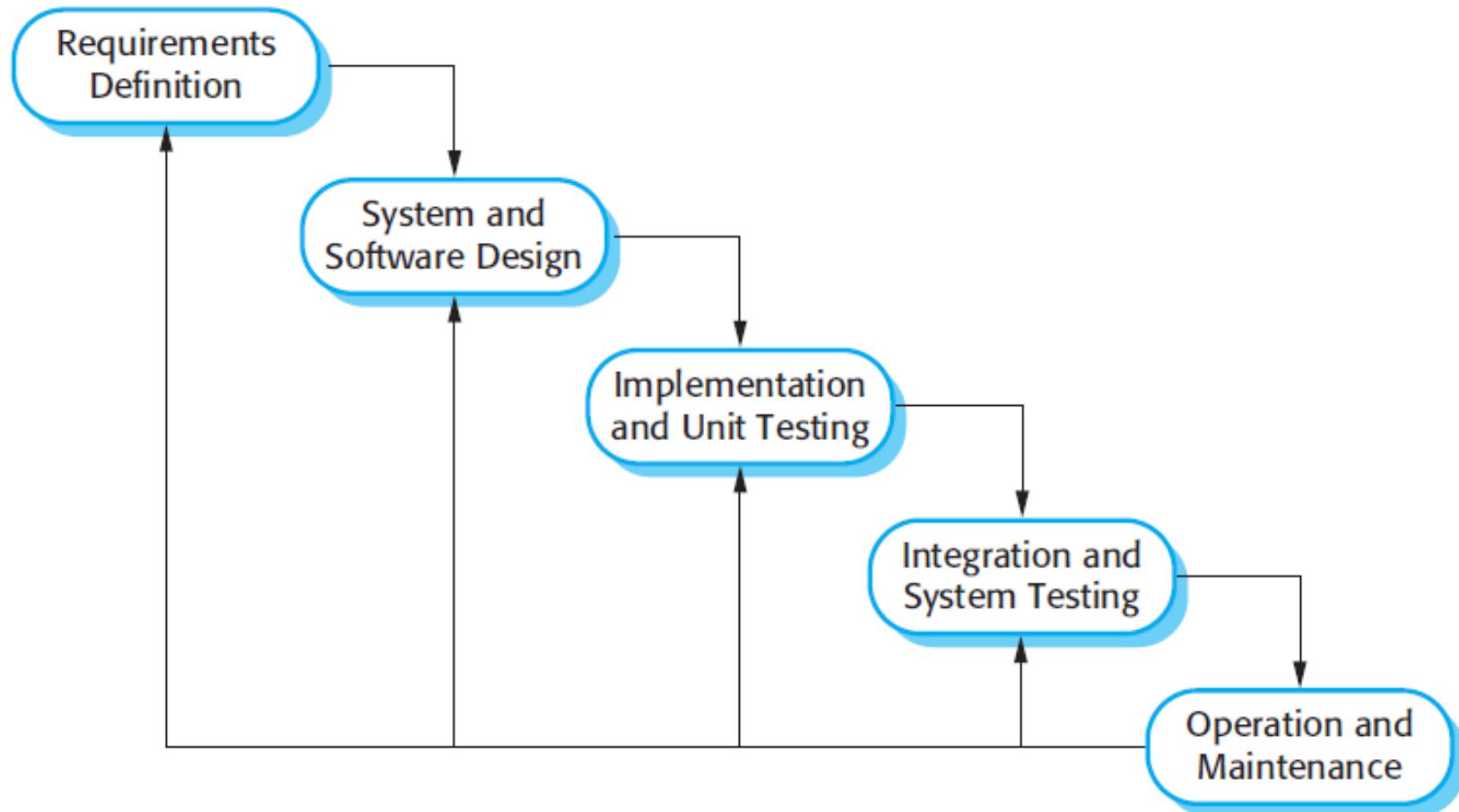
- Waterfall model
- V-Model
- Component-based model
- Prototyping
- iterative, incremental and evolutionary models



# Waterfall by Royce

- The first published model of the software development process was derived from engineering process models used in large military systems engineering (Royce 1970)
- W.W.Royce about Waterfall sad,
- „*I believe in this concept, but the implementation described above is risky and invites failure.*“
- The Rational Model (waterfall) may seem naive to us today. But it is a very natural model for people to conceive ([F.Brooks, The Design of Design](#))

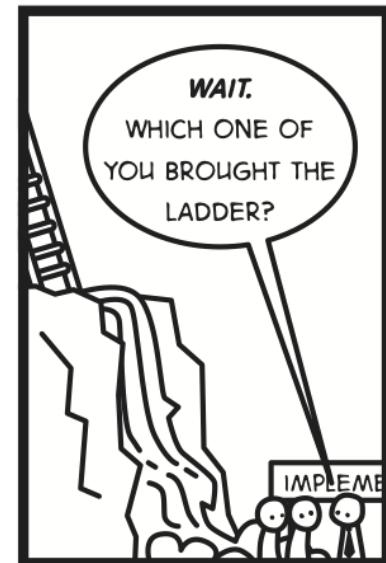
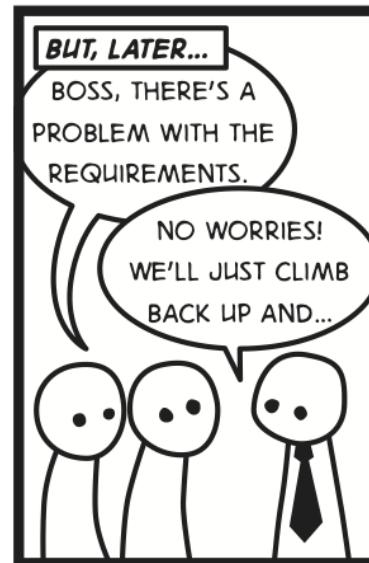
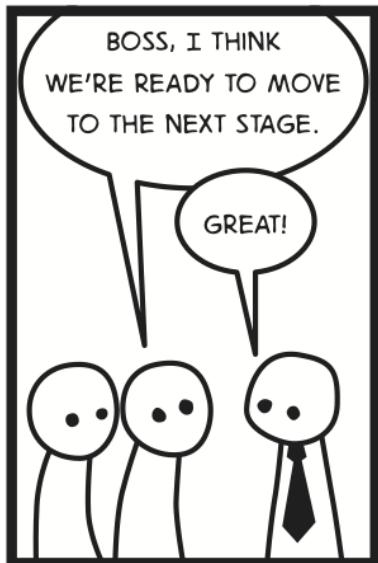
# Waterfall Model



# Characteristics of Waterfall Model

1. The following phase **should not start until the previous phase has finished.**
2. The software process, in practice, is never a simple linear model but involves feedback from one phase to another.
3. The waterfall model is not the right process model in situations where informal team communication is possible and **software requirements change quickly.**
4. An important variant of the waterfall model is **formal system development**, where a mathematical model of a system specification is created

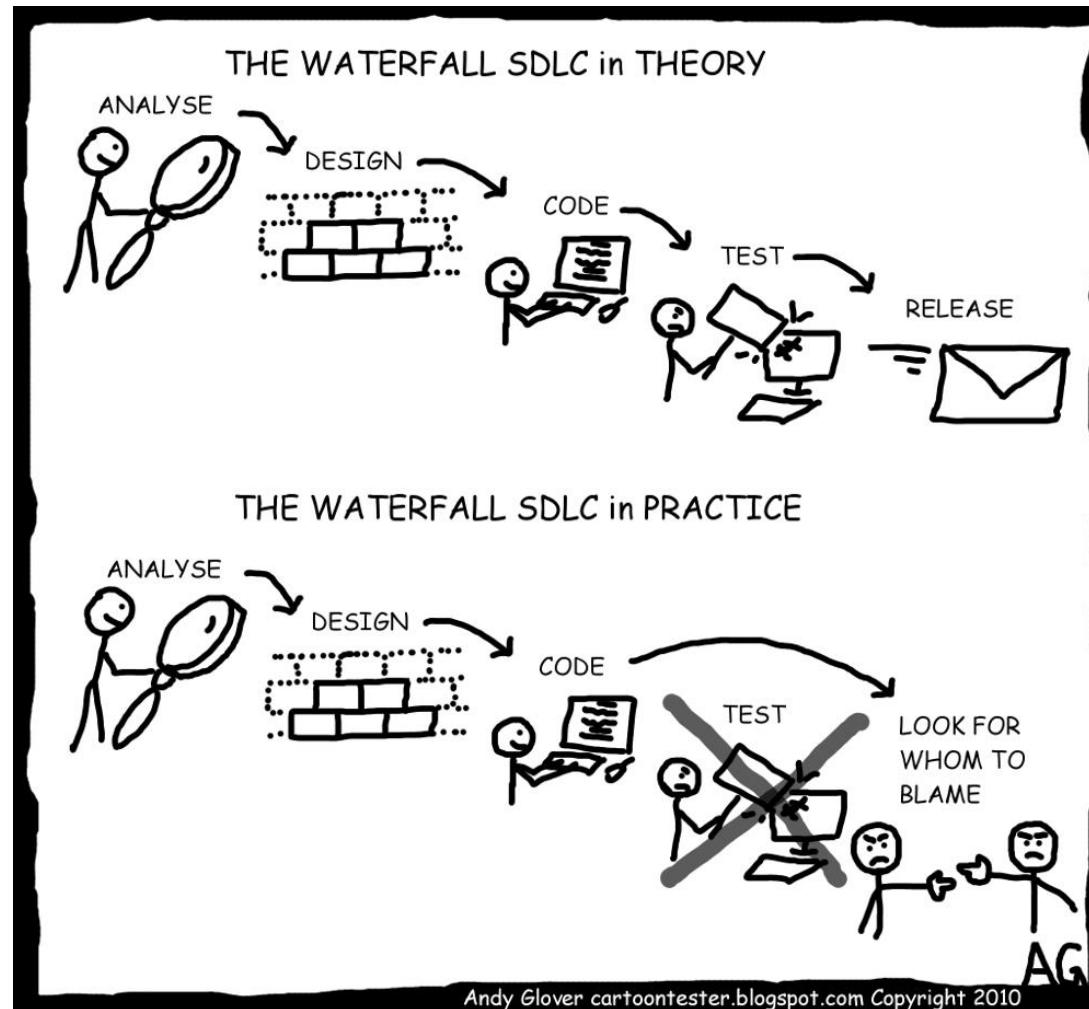
# Waterfall Model



# Waterfall Strengths

- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule

# WaterFall model



# Waterfall Disadvantages

- All requirements must be **known upfront**
- Deliverables created for each phase are considered **frozen – inhibits flexibility**
- Can give a false impression of progress
- **Does not reflect problem-solving** nature of software development – iterations of phases
- Integration is **one big bang** at the end
- Little opportunity for customer to preview the system (until **it may be too late**)

# When to use the Waterfall Model

- Requirements are very well known
- Product definition is stable
- Technology is understood
- New version of an existing product
- Porting an existing product to a new platform.

# When to use the Waterfall Model

- **High risk for new systems** because of specification and design problems.
- **Low risk for well-understood** developments using familiar technology.

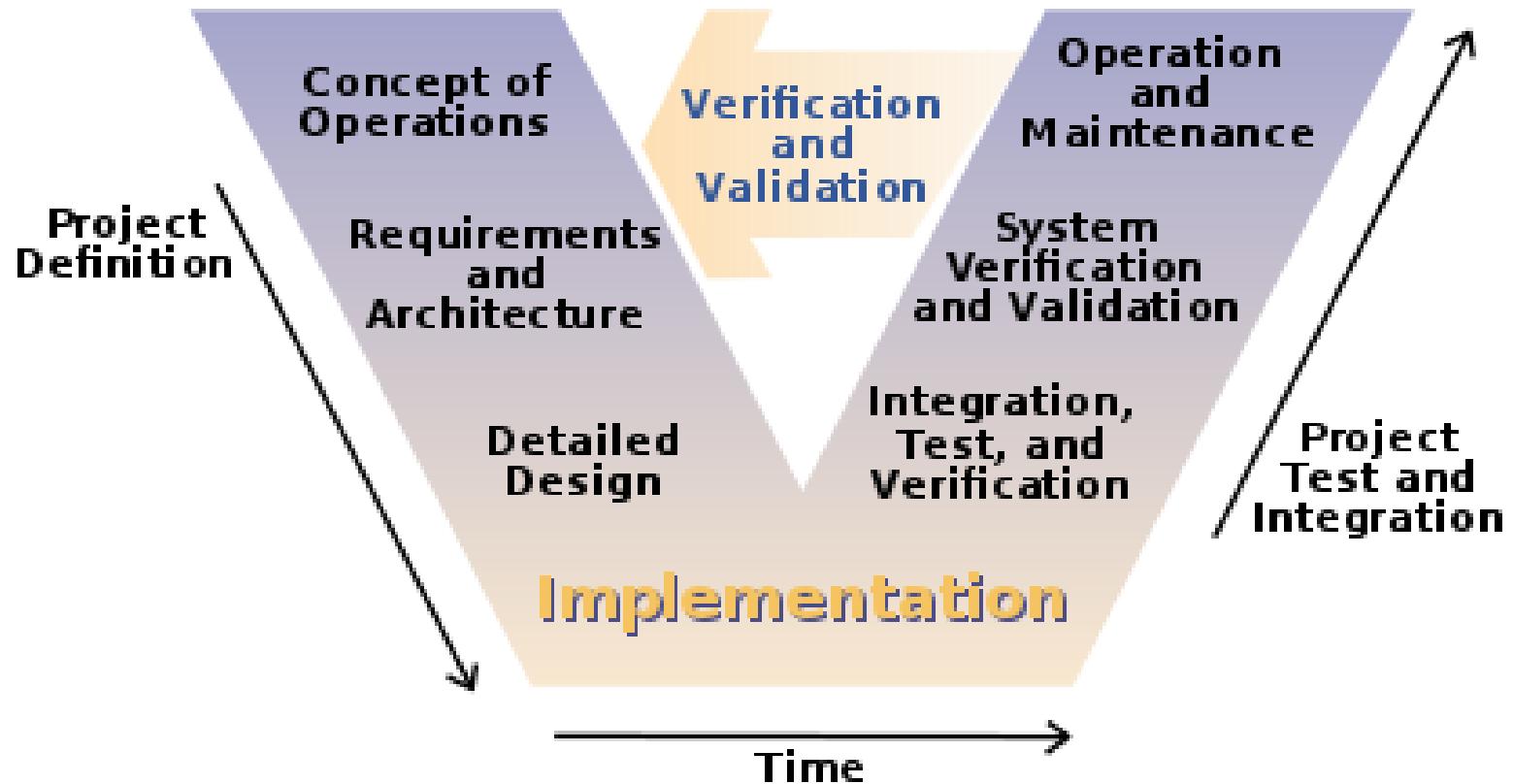
# V model

- In order to put **more emphasis on the verification and validation tasks** compared to the waterfall model, **Boehm** in described a **V-shaped model** now often called the V-Model
- A variation of the waterfall model

# V model

- In order to put **more emphasis on the verification and validation tasks** compared to the waterfall model,
  - Uses unit testing to verify procedural design
  - Uses integration testing to verify architectural (system) design
  - Uses acceptance testing to validate the requirements
- If problems are found during verification and validation, the left side of the V can be re-executed before testing on the right side is reenacted

# V model



# Disadvantages of V-Model

- The V-Model is emphasis on verification and validation, testing and other quality assurance tasks **start too late**,
- The V-Model does not sufficiently **support the early preparation of testing, nor the early review and analysis of design documentation**

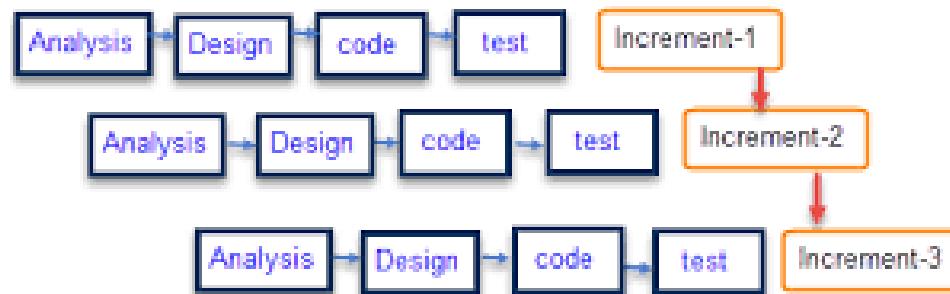
# Iterative, Incremental and Evolutionary Development

- **Increment.** A tested, deliverable version of a software product that provides new or modified capabilities
- **Iteration:**
  - 1. process of performing a **sequence of steps repeatedly**
  - 2. single execution of the sequence of steps in (1)

# Incremental Development

- **Incremental development**

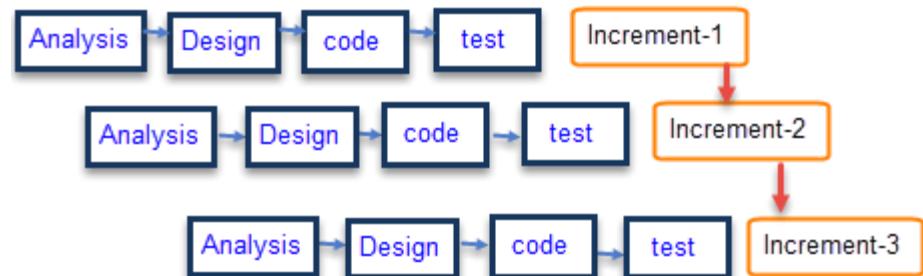
- Software development technique in which requirements definition, design, implementation, and testing occur in an overlapping, iterative (rather than sequential) manner, resulting in incremental completion of the overall software product



Incremental Model

# Incremental software development

- The total extension of a system under development remains open;
  - it is realized in stages of expansion.
  - The first stage is the core system.
- **Each stage of expansion extends the existing system and is subject to a separate project.**
- Providing a new stage of expansion typically includes (as with iterative development) an improvement of the old components.



Incremental Model

# Incremental development

- Incremental development is based on the idea of:
  - developing an initial implementation,
  - getting feedback from users and others,
  - evolving the software through several versions until the required system has been developed
- By developing the software incrementally, it is **cheaper and easier to make changes in the software as it is being developed.**

# Incremental vs. iterative Development

Iterative

1



2



3



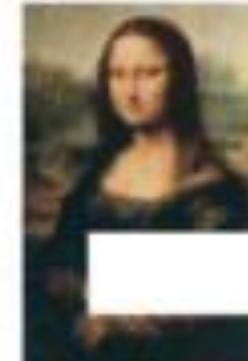
4



5



Incremental



# Incremental/ iterative Development

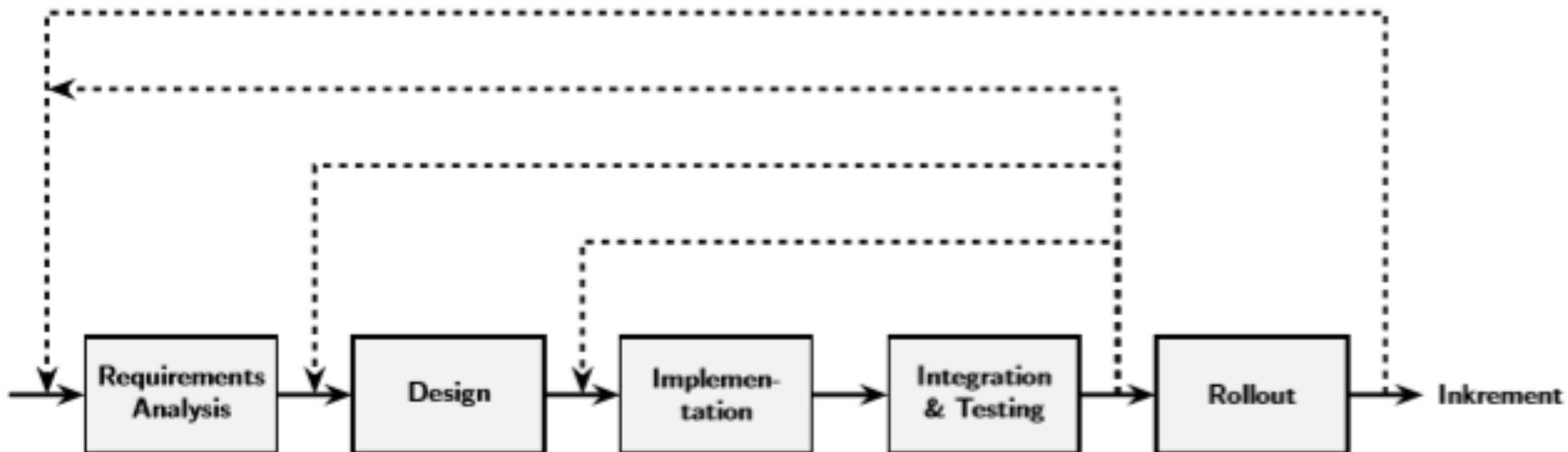


# Incremental/ iterative Development

- A **project life cycle**, where the project scope is generally determined early in the project life cycle, **but time and cost estimates are routinely modified** as the project team understanding of the product increases.
  - Iterations develop the product through a series of repeated cycles, while **increments successively add to the functionality of the product.**

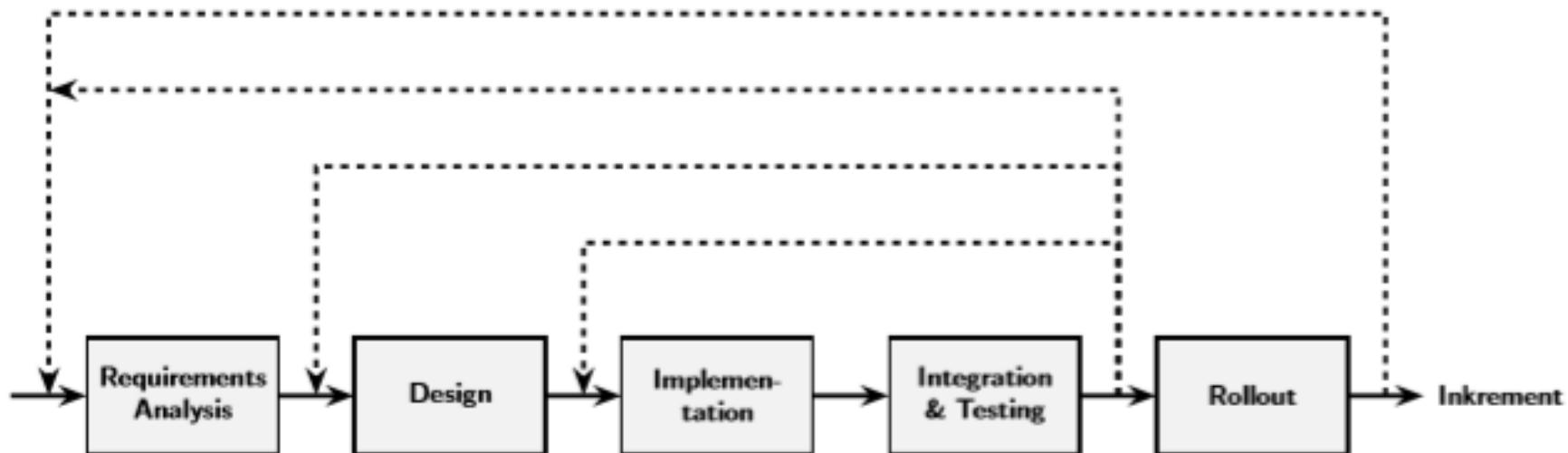
# Different types of iteration

- An iteration refers to the sequence of steps which is repeatedly performed
- Alternatively, it may consist of a more **limited set of development steps**, such as iterating the requirements analysis steps, or the sequence **correction of bugs–test–delivery**



# Different types of iteration

- An iteration **may or may not include the delivery** of a new version of the software product
- The **result of an iteration may be a new increment** if the iteration includes all the necessary steps for such a delivery.



# Benefits of incremental development

- The main benefit of iterative development is the **reduction of risk** by collecting early feedback
- **Any new functionality can be put into production very soon** after having been implemented, thus providing the relevant benefit early on

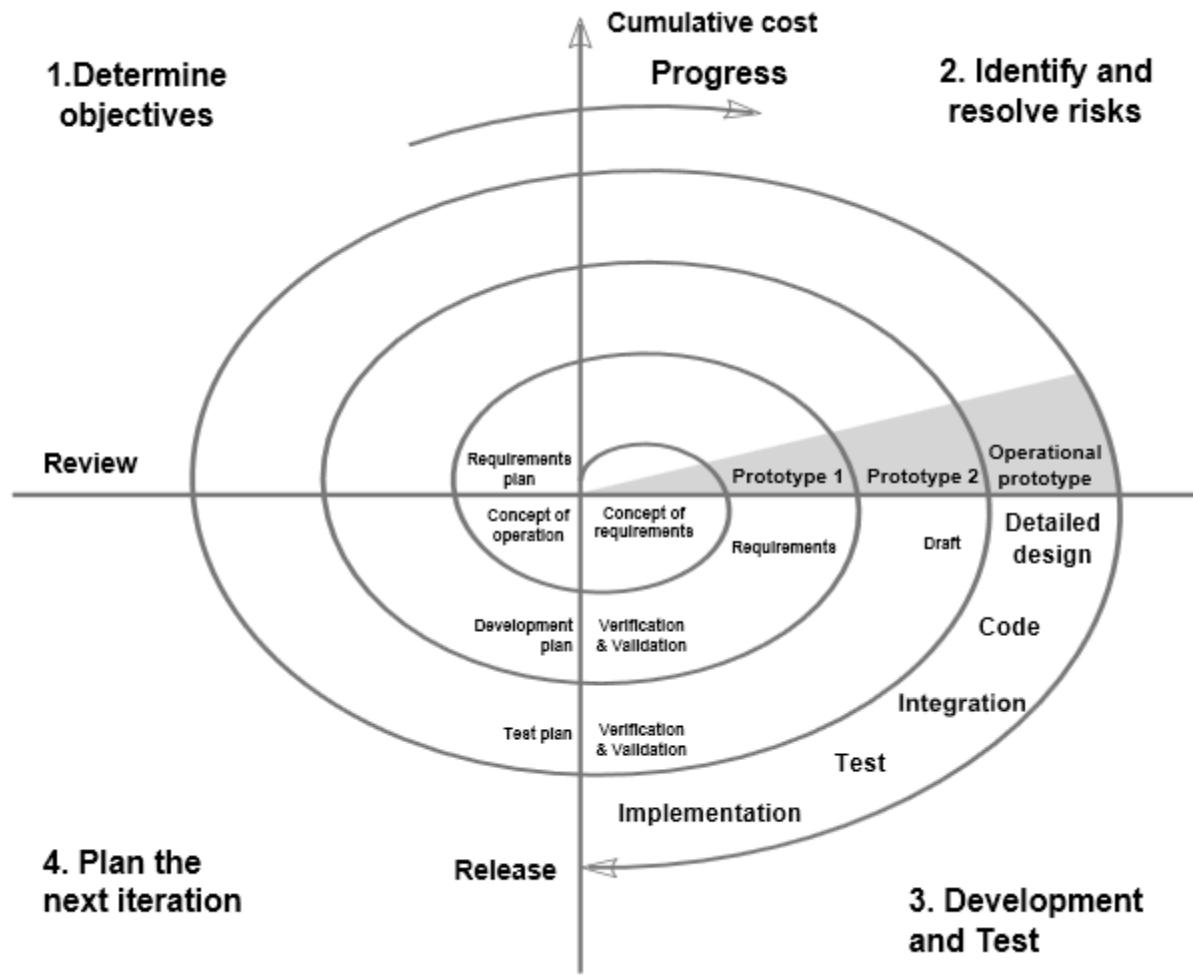
# Spiral proces model

- Suggested by Barry Boehm (1988)
- Combines development activities with risk management to minimize and control risks
- Spiral model
  - Model of the software development process in which the constituent activities, typically requirements analysis, preliminary and detailed design, coding, integration, and testing, are **performed iteratively until the software is complete**

# Spiral proces model

- The model is presented as a spiral in which each iteration is represented by a circuit around four major activities:
  - Plan
  - Determine goals, alternatives and constraints
  - Evaluate alternatives and risks
  - Develop and test

# Stages of Incremental process

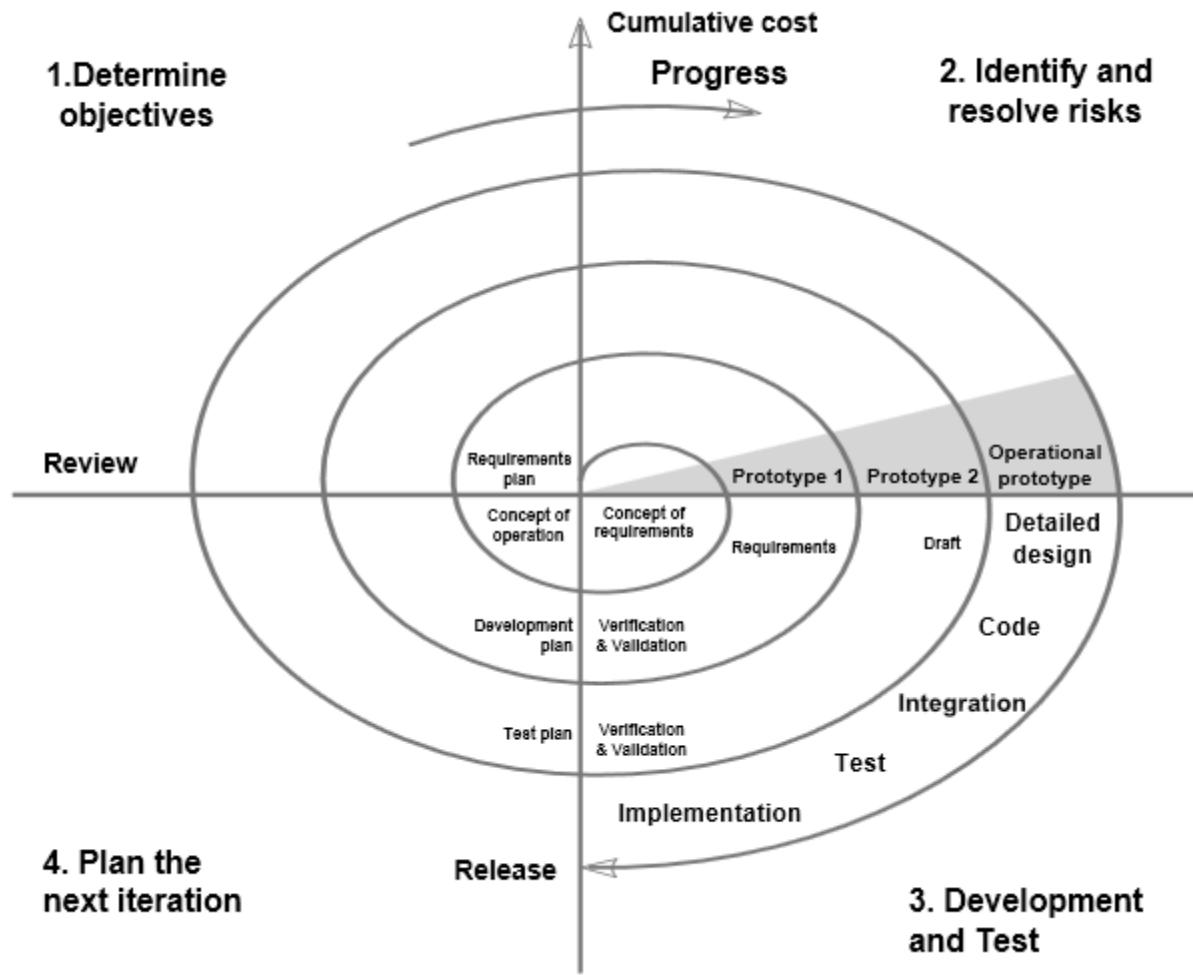


# Incremental process

## 1 stage: Determine objective

- Objectives: functionality, performance, hardware/software interface, critical success factors, etc.
- Alternatives: build, reuse, buy, sub-contract, etc.
- Constraints: cost, schedule, interface, etc.

# Stages of Incremental process

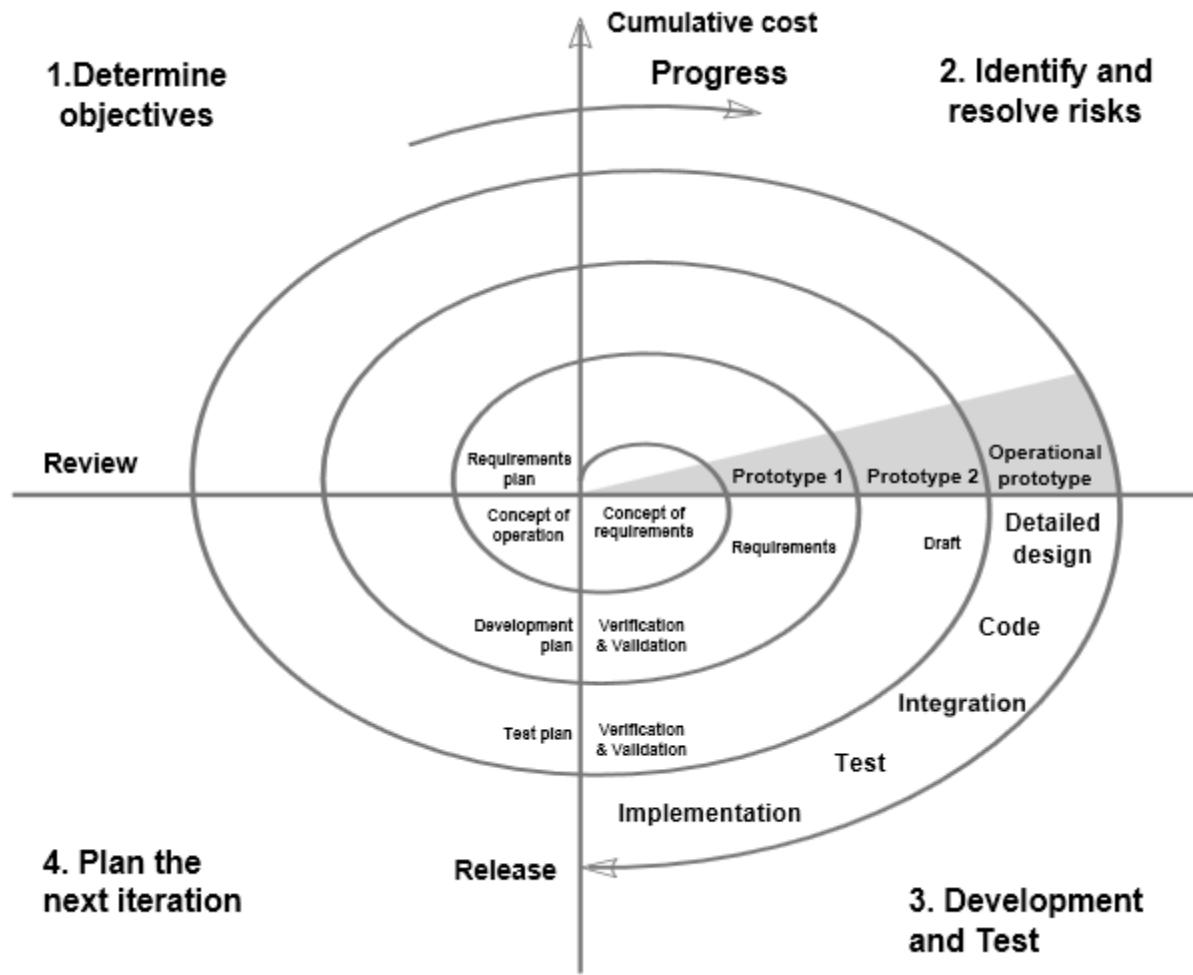


# Incremental process

## 2 stage: identify and resolve risks

- Study alternatives relative to objectives and constraints
- Identify risks (lack of experience, new technology, tight schedules, poor process, etc.)
- Resolve risks (evaluate if money could be lost by continuing system development)

# Stages of Incremental process

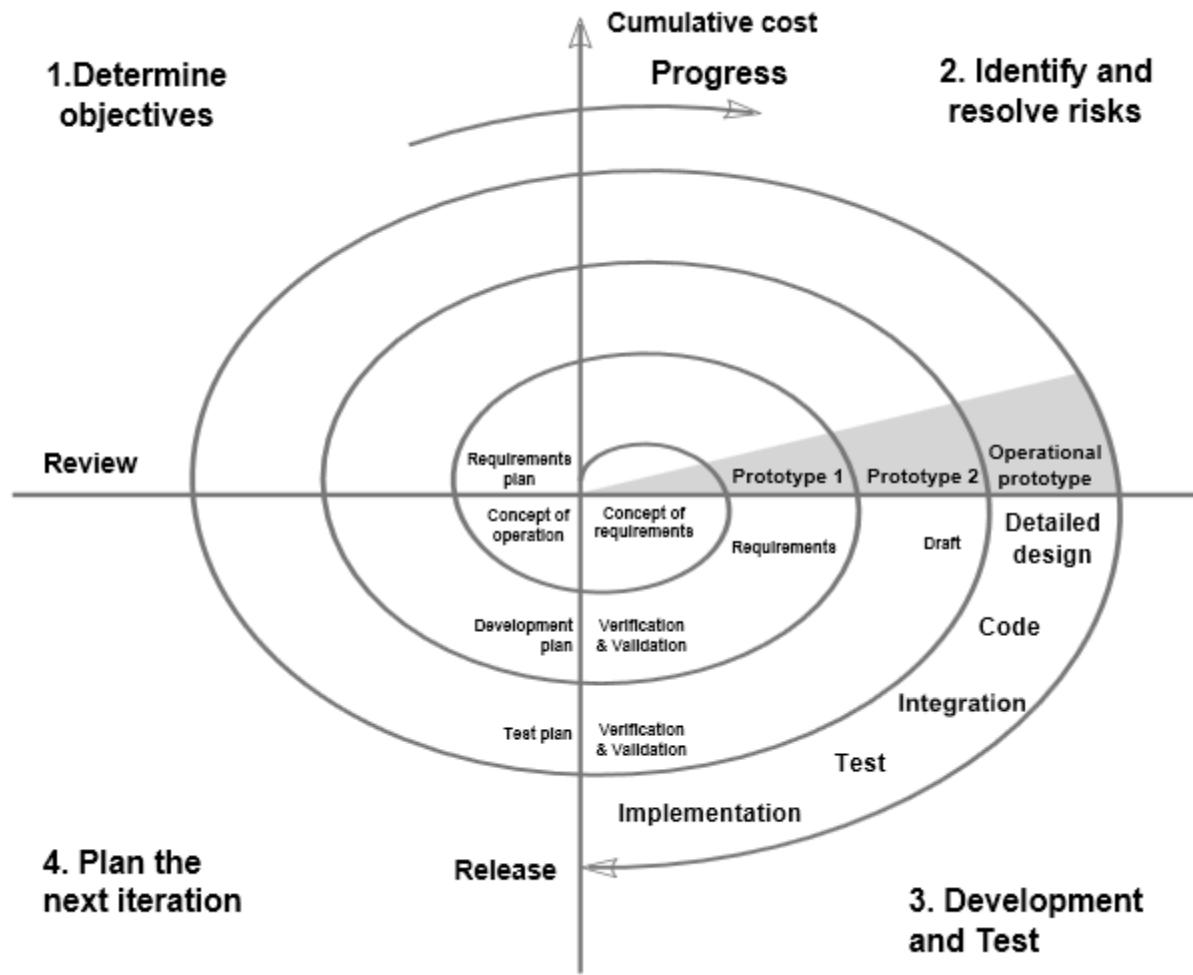


# Incremental process

## 3 stage: Development and test

- Typical activities:
  - Create a design
  - Review design
  - Develop code
  - Inspect code
  - Test product

# Stages of Incremental process

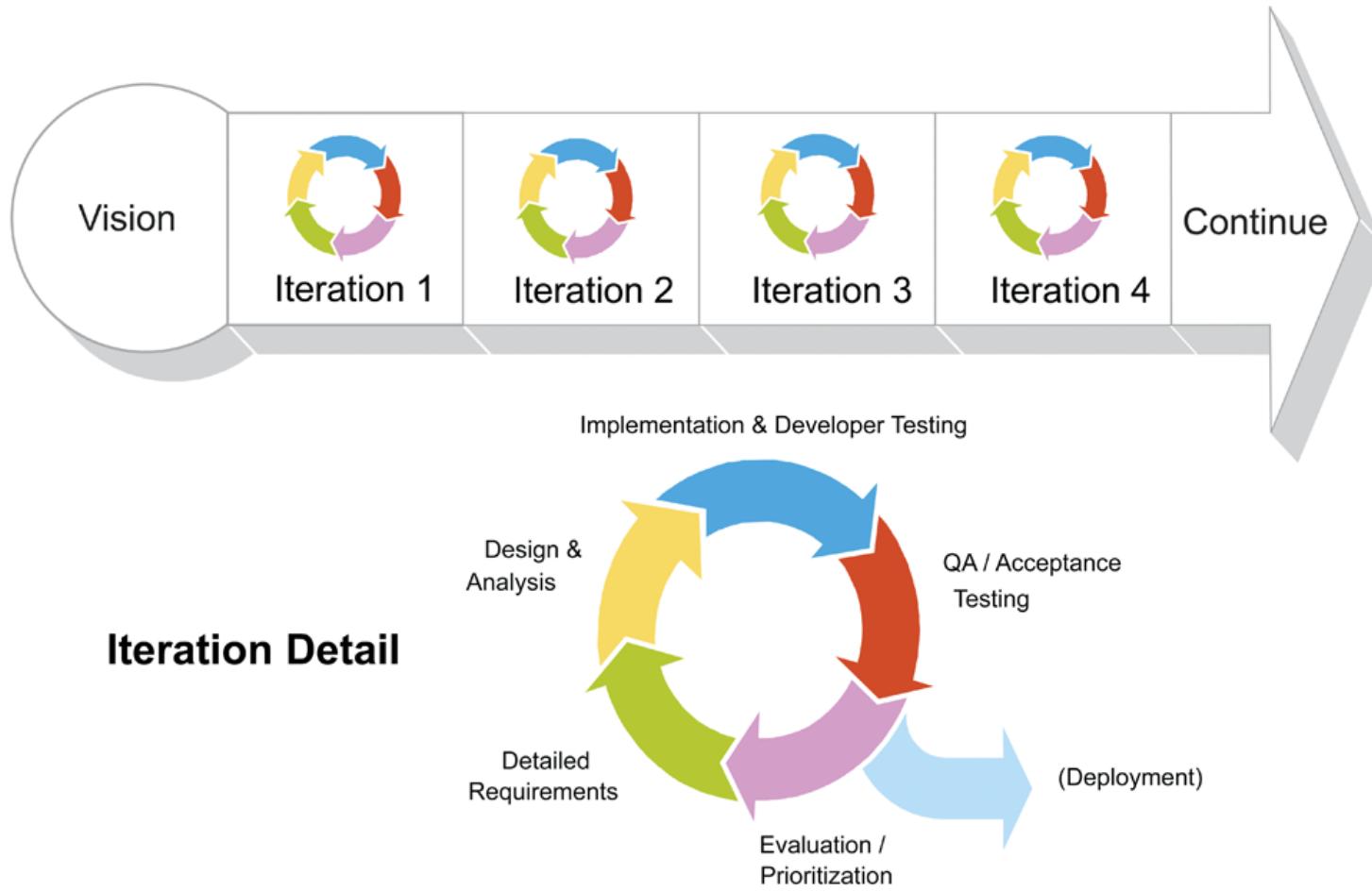


# Incremental process

## 4 stage: Plan next iteration

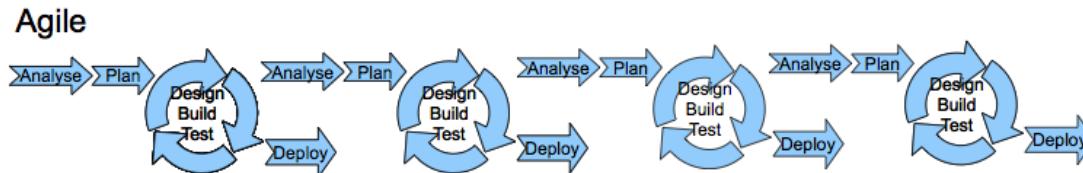
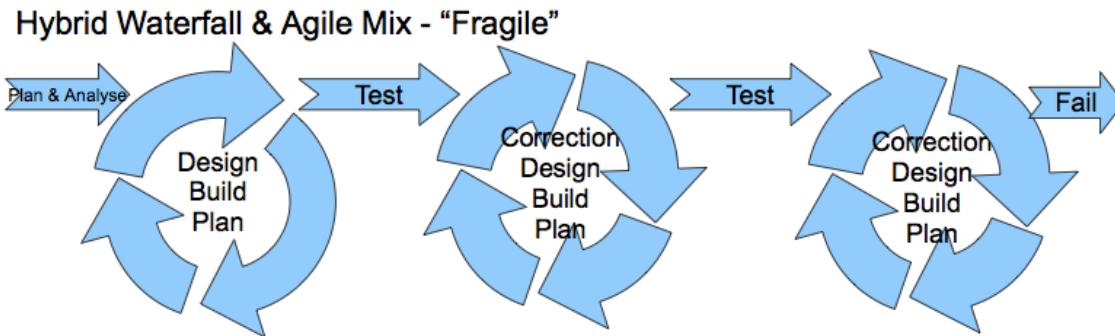
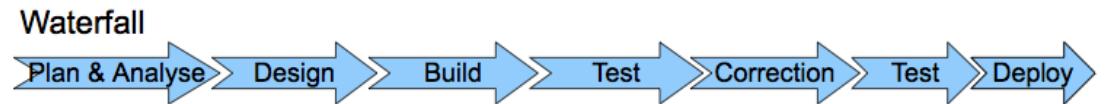
- Typical activities
  - Develop project plan
  - Develop configuration management plan
  - Develop a test plan
  - Develop an installation plan

# Iterative development



# Incremental vs Waterfall

- Incremental software development, which is a fundamental part of agile development methods, is better than a waterfall approach for systems whose requirements are likely to change during the development process.



# Incremental process Strengths

- **Users see the system early** because of rapid prototyping tools
- **Critical high-risk functions are developed first**
- The design does not have to be perfect
- Users can be closely tied to all lifecycle steps
- **Early and frequent feedback from users**
- Cumulative costs assessed frequently

# Incremental process Disadvantages

- **Time spent for evaluating risks too large** for small or low-risk projects
- Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive
- **The model is complex**
- Risk assessment expertise is required
- Developers must be reassigned during non-development phase activities
- May be **hard to define objective, verifiable milestones** that indicate readiness to proceed through the next iteration

# When to use Incremental process

- When creation of a **prototype is appropriate**
- When costs and risk evaluation is important
- **For medium to high-risk projects**
- **Long-term project commitment unwise because of potential changes to economic priorities**
- Users are unsure of their needs
- Requirements are complex
- New product line
- **Significant changes are expected** (research and exploration)

# Software process: Prototyping

- **Prototype - an experimental model, either functional or nonfunctional, of the system or part of the system**
- The main goal of prototypes is **to get a better understanding of a particular aspect** of the system under development

# Software process: Prototyping

- A prototype is an early version of a software system that is used:
  - **to demonstrate concepts,**
  - **try out design options,**
  - **To find out more about the problem and its possible solutions**
- System prototypes allow potential users **to see how well the system supports their work.**

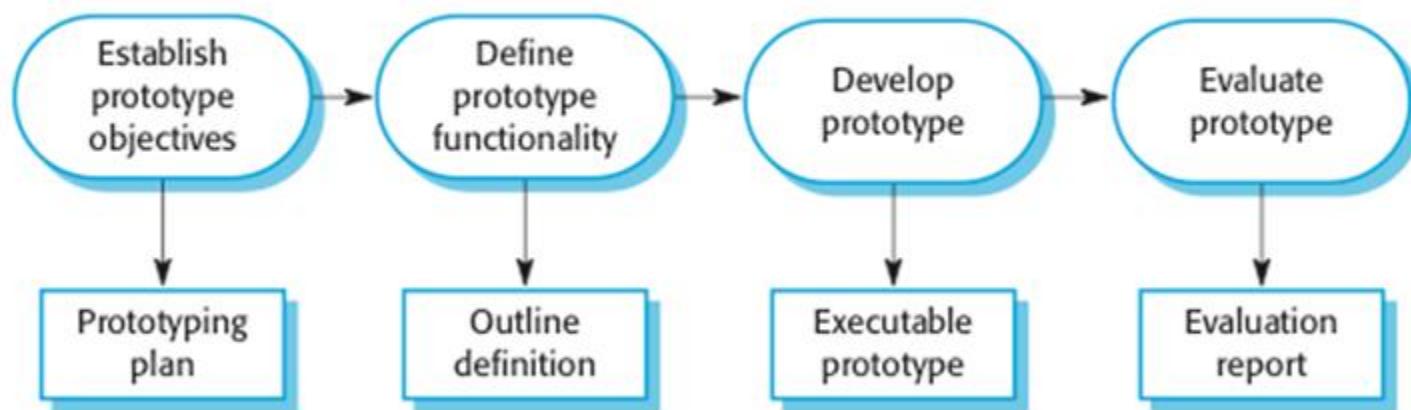
# Exploratory prototypes

- An exploratory prototype is used in the initial phase of development, with the main purpose:
  - of **helping to identify the requirements** on the system to be developed.
  - It is above all a communication medium that helps to get a common **understanding about expected system properties between developers or analysts** on the one side, and stakeholders or users on the other.

# Experimental prototypes

- An experimental prototype is used somewhat later, once at least an initial set of requirements has been identified.
- Its main purpose is **to experiment with the current solution** (initial requirements, design, technical solution, etc.) in order to validate it, e.g. checking the technical feasibility.

# The process of prototype development



# Why software prototype?

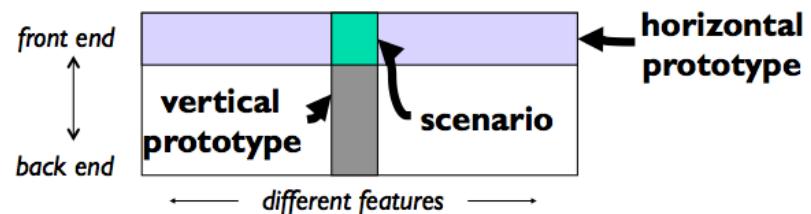
1. In the requirements engineering process, a **prototype can help with the elicitation and validation of system requirements.**
2. In the system design process, a **prototype can be used to explore software solutions** and in the development of a user interface for the system.

# Basic Steps of Prototyping

1. Identify basic requirements
  - Including input and output info
  - Details (e.g., security) generally ignored
2. Develop initial prototype
  - UI first
3. Review
  - Customers/end –users review and give feedback
4. Revise and enhance the prototype & specs
  - Negotiation about scope of contract may be necessary

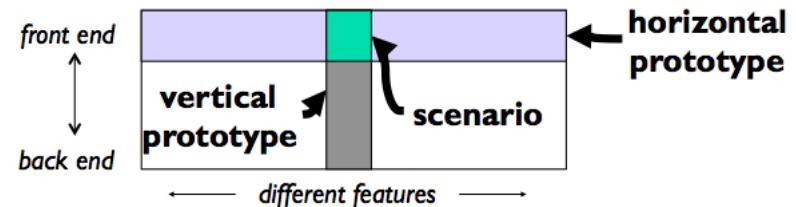
# Dimensions of prototyping

- Horizontal prototype
  - Broad view of entire system/sub-system
  - Focus is on user interaction more than low-level system functionality (e.g., database access)
  - Useful for:
    - Confirmation of UI requirements and system scope
    - Demonstration version of the system to obtain buy-in from business/customers
    - Develop preliminary estimates of development time, cost, effort

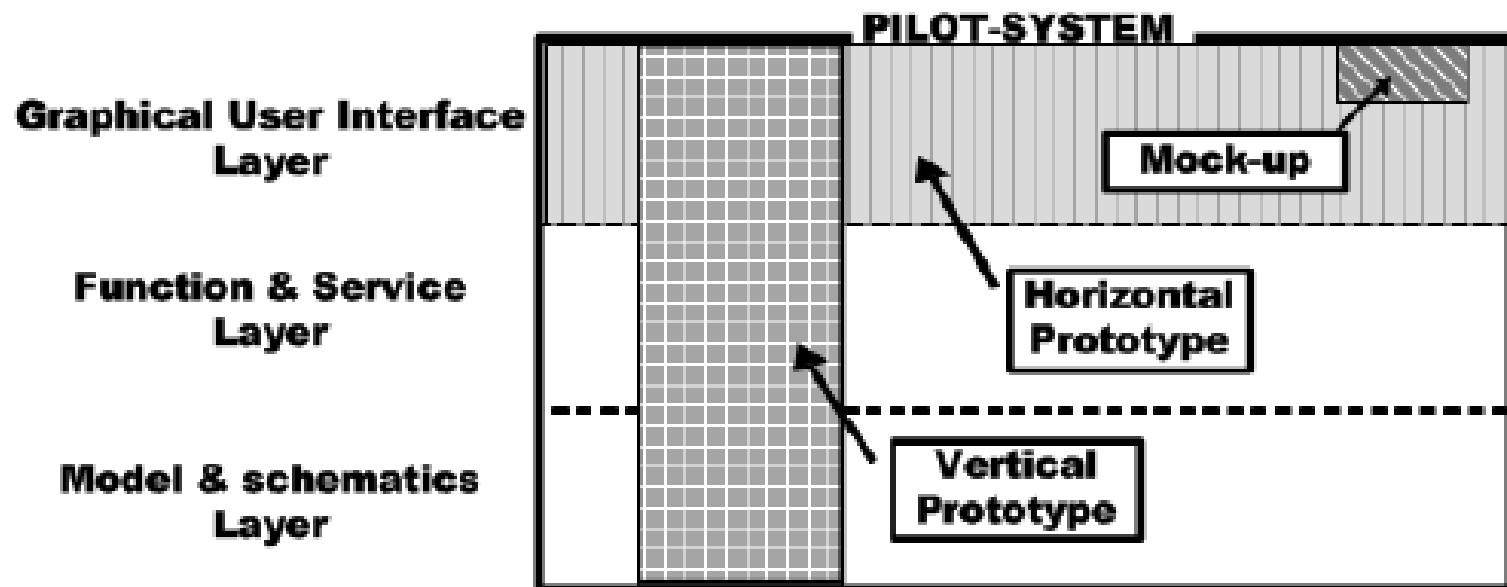


# Dimensions of Prototyping

- Vertical prototype
  - More complete elaboration of a single sub-system or function
  - Useful for:
    - Obtaining detailed requirements for a given function
    - Refining database design
    - Obtaining info on system interface needs
    - Clarifying complex requirements by drilling down to actual system functionality



# Horizontal vs. vertical prototyping



# Horizontal vs. vertical prototyping

Horizontal prototype:

- broad coverage of features
- less detail for each feature
- less realistic evaluation

Vertical prototype:

- fewer features
- more detail for each feature
- more realistic evaluation

# Prototyping advantages

- Reduced time and cost
  - Can improve the quality of requirements and specifications provided to developers
    - Early determination of what the user really wants can result in faster and less expensive software
- Improved/increased user involvement
  - User can see and interact with the prototype, allowing them to provide better/more complete feedback and specs
  - Misunderstandings/miscommunications revealed
  - Final product more likely to satisfy their desired look/feel/performance

# Disadvantages of prototyping 1

- Insufficient analysis
  - Focus on limited prototype can distract developers from analyzing complete project
  - May overlook better solutions
  - Conversion of limited prototypes into poorly engineered final projects that are hard to maintain
  - Limited functionality may not scale well if used as the basis of a final deliverable
    - May not be noticed if developers too focused on building prototype as a model

# Disadvantages of prototyping 2

- User confusion of prototype and finished system
  - Users can think that a prototype (intended to be thrown away) is actually a final system that needs to be polished
    - Unaware of the scope of programming needed to give prototype robust functionality
  - Users can become attached to features included in prototype for consideration and then removed from final specification

# Disadvantages of prototyping 3

- Developer attachment to prototype
  - If spend a great deal of time/effort to produce, may become attached
  - Might try to attempt to convert a limited prototype into a final system
    - Bad if the prototype does not have an appropriate underlying architecture

# Disadvantages of prototyping 4

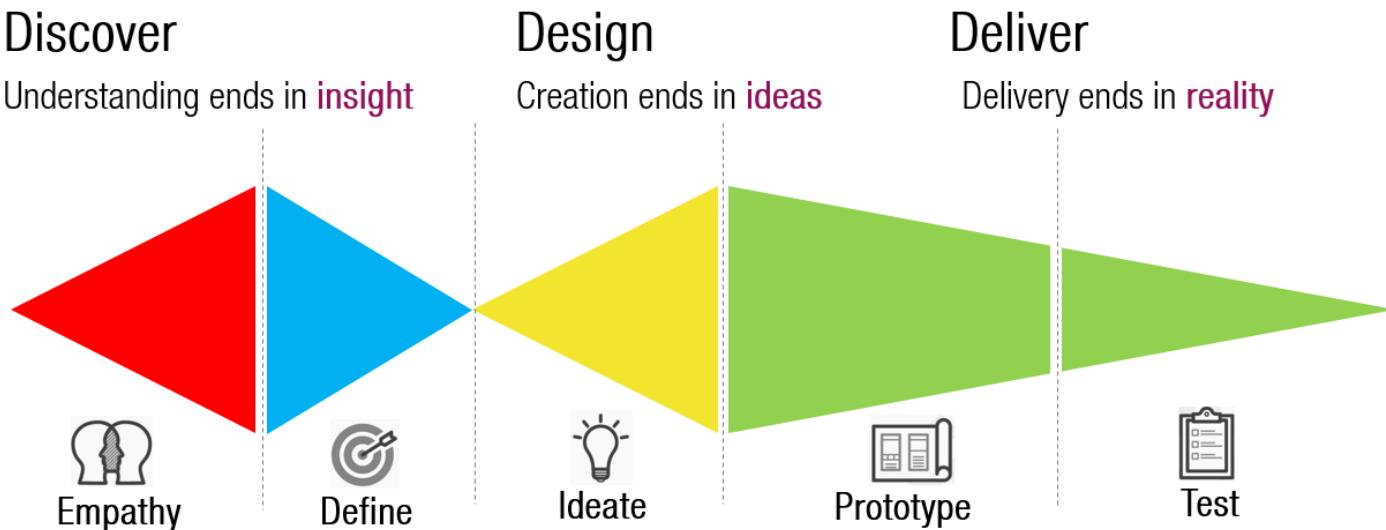
- Excessive development time of the prototype
  - Prototyping supposed to be done quickly
  - If developers lose sight of this, can try to build a prototype that is too complex
  - For throw away prototypes, the benefits realized from the prototype (precise requirements) may not offset the time spent in developing the prototype – expected productivity reduced
  - Users can be stuck in debates over prototype details and hold up development process

# Disadvantages of prototyping 5

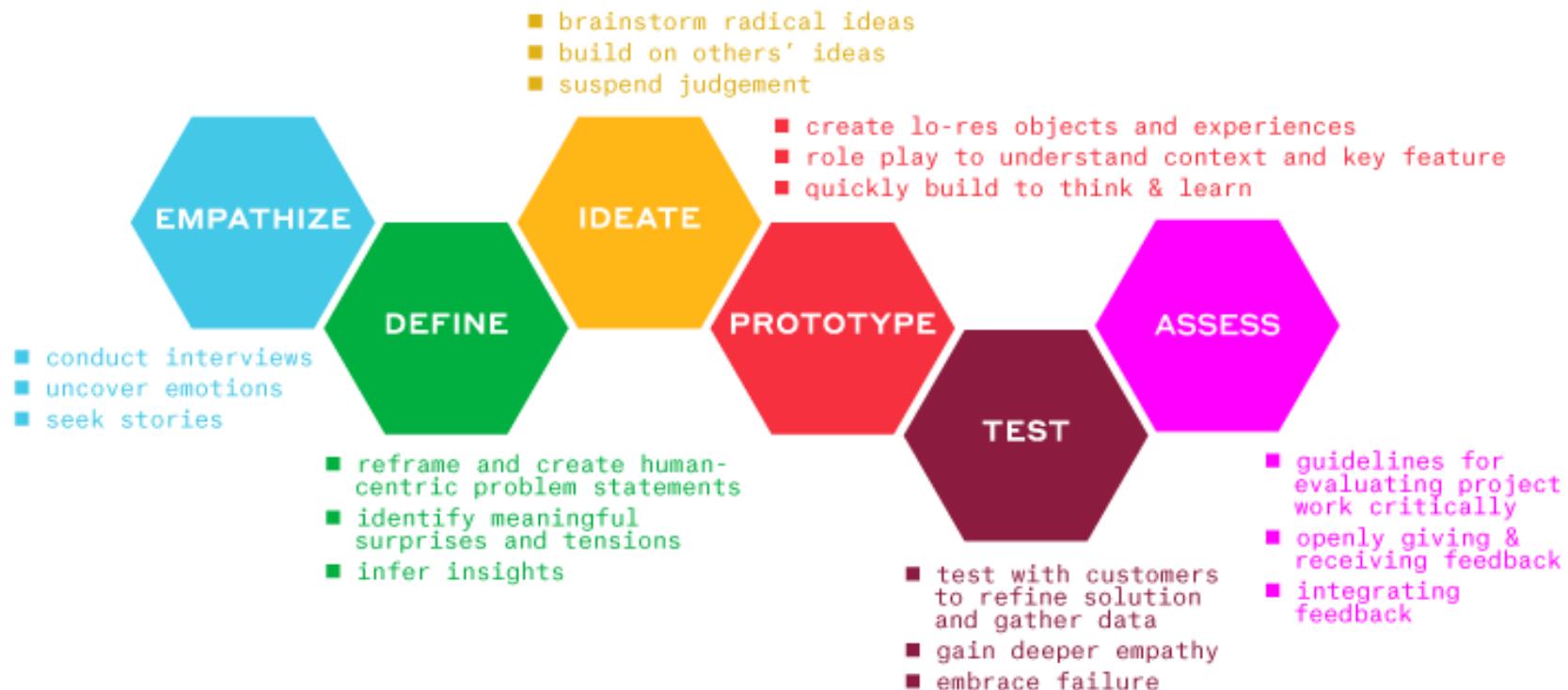
- Expense of implementing prototyping
  - Start up costs of prototyping may be high
  - Expensive to change development methodologies in place (re-training, re-tooling)
  - Slow development if proper training not in place
    - High expectations for productivity unrealistic if insufficient recognition of the learning curve
  - Lower productivity can result if overlook the need to develop corporate and project specific underlying structure to support the technology

# Design thinking

- The main purpose of design thinking is to get an understanding of **what users really need**, and to provide innovative solutions to that need.



# Design Thinking Process Diagram\*



d.school Executive Education  
Hasso Plattner Institute of Design at Stanford University

\*not necessarily linear, apply as needed ©2019

<https://www.interaction-design.org/literature/topics/design-thinking>

<https://voltagecontrol.co/a-step-by-step-guide-to-the-design-thinking-process-d0a95a28b9db>

# Steps of Design thinking

1. **Empathise with users to understand them**, their needs, and the problems to be solved
2. **Define the needs of the users** and the resulting problem to be solved
3. **Ideate is about generating ideas** to solve the problems, deliberately taking a very wide view of possible solutions and typically using some form of brainstorming techniques
4. **Prototype the best problem solutions found**
5. **Test the prototype solution** to refine the prototype solution iteratively.

**Map–Sketch–Decide–Prototype–Test**

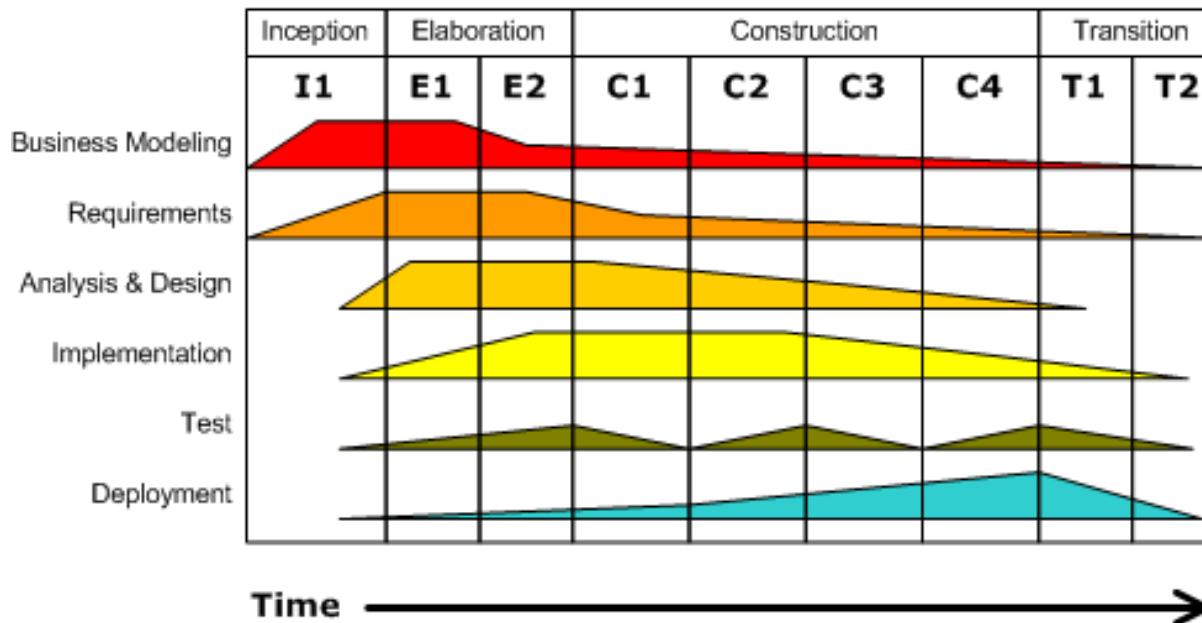
# The Rational Unified Process

- This is Combined Software Life Cycle and Process Models
- With the growing importance **of object-oriented development**, a number of different methods for object-oriented analysis, design and programming were published in the early 1990s by different authors.
- RUP is a **component-based model**

# The Rational Unified Process

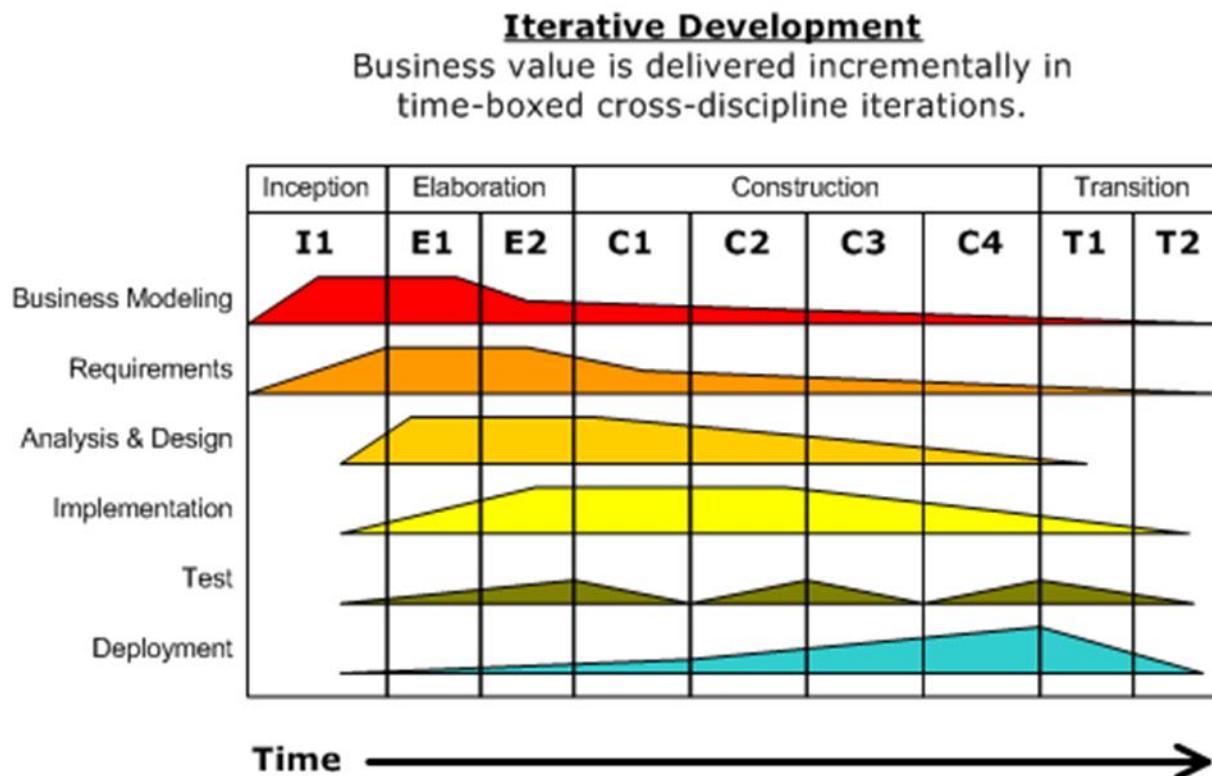
- RUP describes an iterative, incremental approach to software development,
- Is not an agile approach

**Iterative Development**  
Business value is delivered incrementally in time-boxed cross-discipline iterations.



# The Rational Unified Process

- The final version 7.0 was published in 2005.
- Today, RUP is still fairly well-known but no longer supported by Rational or its parent company IBM.



# Other Software Process Models

- Formal System Development:
  - Transforms a mathematical based specification through different representation → executable program.
  - Program correctness is easy to demonstrate, as the transformations preserve the correctness.
- Reuse-Oriented Development:
  - Concentrates on integrating new system with existing components/systems.
  - Growing rapidly as development cost increase.

# Why are there so many models?

- The choice of a **model depends on the project circumstances and requirements**
- A good choice of a model can result in a vastly more productive environment than a bad choice
- **A cocktail of models is frequently used in practice to get the best of all worlds** – models are often combined or tailored to environment
- “Models” are as often descriptive as they are prescriptive

# The “best” model depends on...

- The task at hand
- Risk management
- Quality / cost control
- Predictability
- Visibility of progress
- Customer involvement and feedback
- Team experience
- ...

# Software Engineering

dr. Asta Slotkienė  
[asta.slotkiene@vgtu.lt](mailto:asta.slotkiene@vgtu.lt)

# Why iterative and incremental process?

- Software is part of almost all business operations, so new software has to be **developed quickly** to take advantage of new opportunities and to respond to competitive
- Rapid software development and delivery is the **most critical requirement for most business systems**

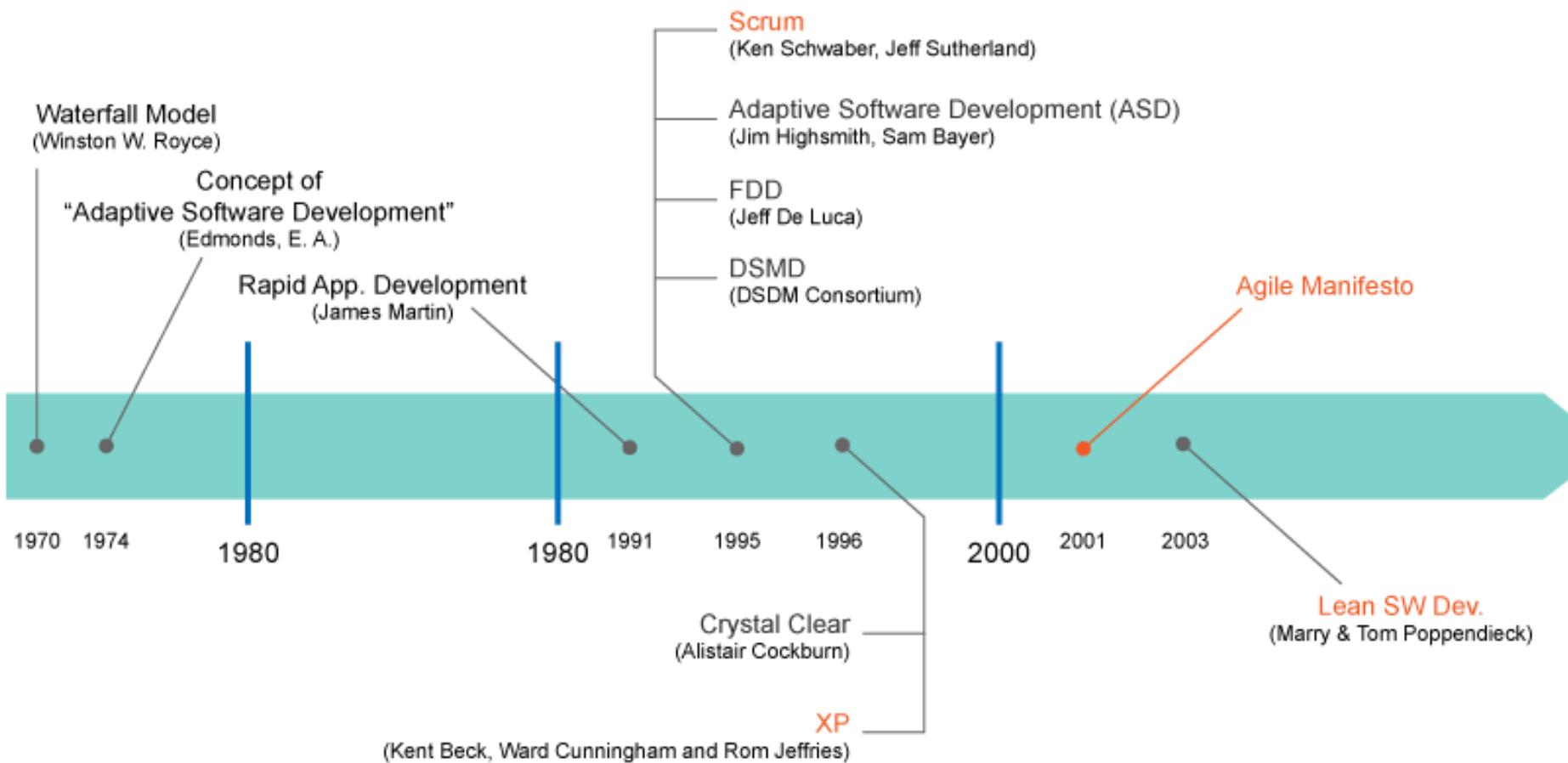
# Iterative and incremental process

- Iterative: we develop software through repeated iterations (cycles).
- Incremental: we develop software in small portions at a time.

# Software development vs. businesses

- Businesses are operating in a changing environment
- Requirements change because customers find it impossible to predict:
  1. how a system will affect working practices?
  2. how it will interact with other systems?
  3. what user operations should be automated?

# History of Agile



<https://www.agilealliance.org/agile101/practices-timeline/>

# History of rapid software development

- 1990s with the development of the idea of “agile methods”
  - Extreme Programming (Beck 1999),
  - Scrum (Schwaber and Beedle 2001),
  - DSDM (Stapleton 2003).

Rapid software development became known as **agile development** or **agile methods**.

# Agile manifesto

*We are uncovering better ways of developing software by doing it and helping others do it.  
Through this work we have come to value:*

- ***Individuals and interactions over processes and tools***
- ***Working software over comprehensive documentation***
  - ***Customer collaboration over contract negotiation***
  - ***Responding to change over following a plan***

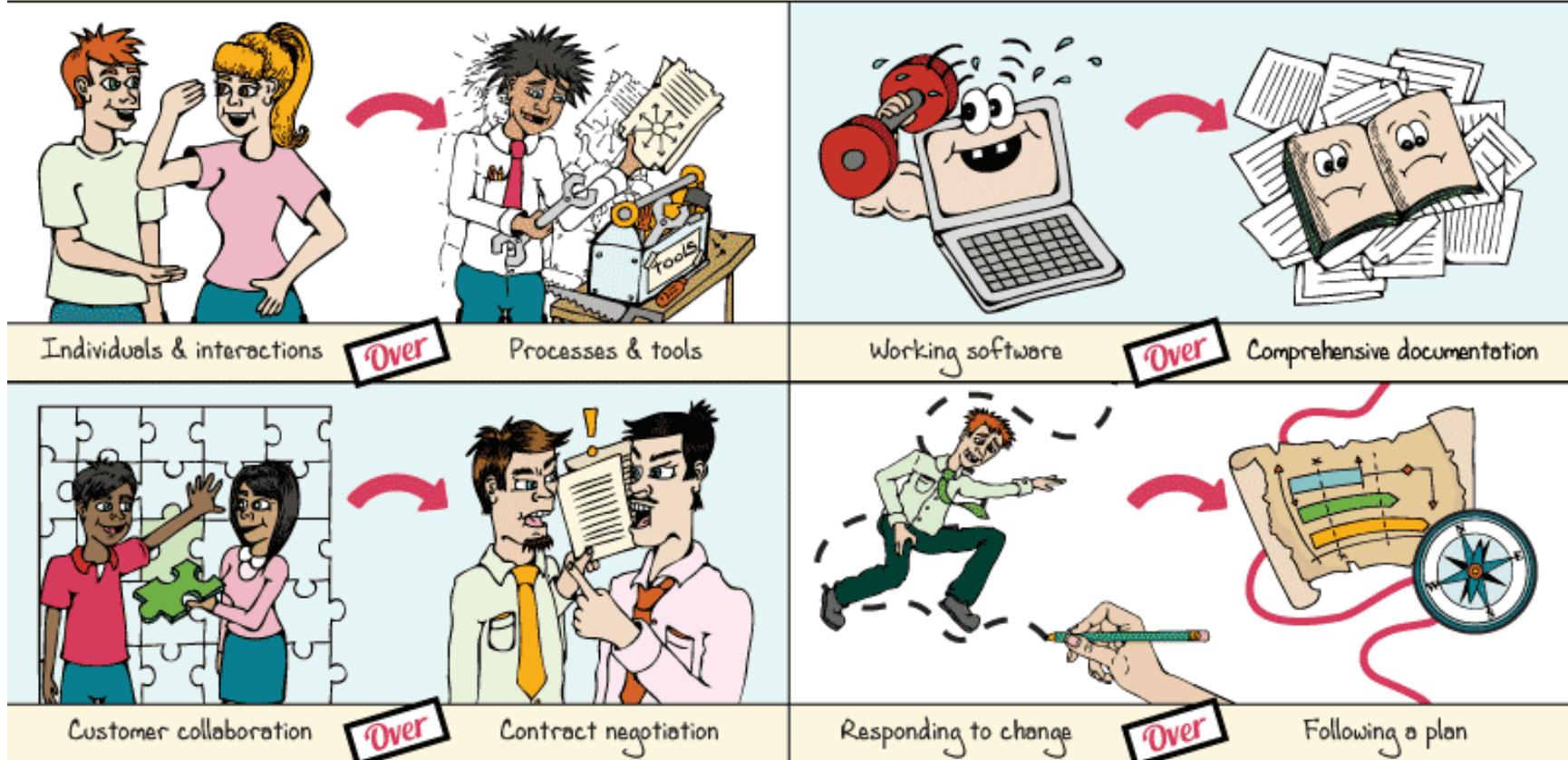
*That is, while there is value in the items on the right, we value the items on the left more.*

# Agile manifesto



# **Manifesto for Agile Software Development\***

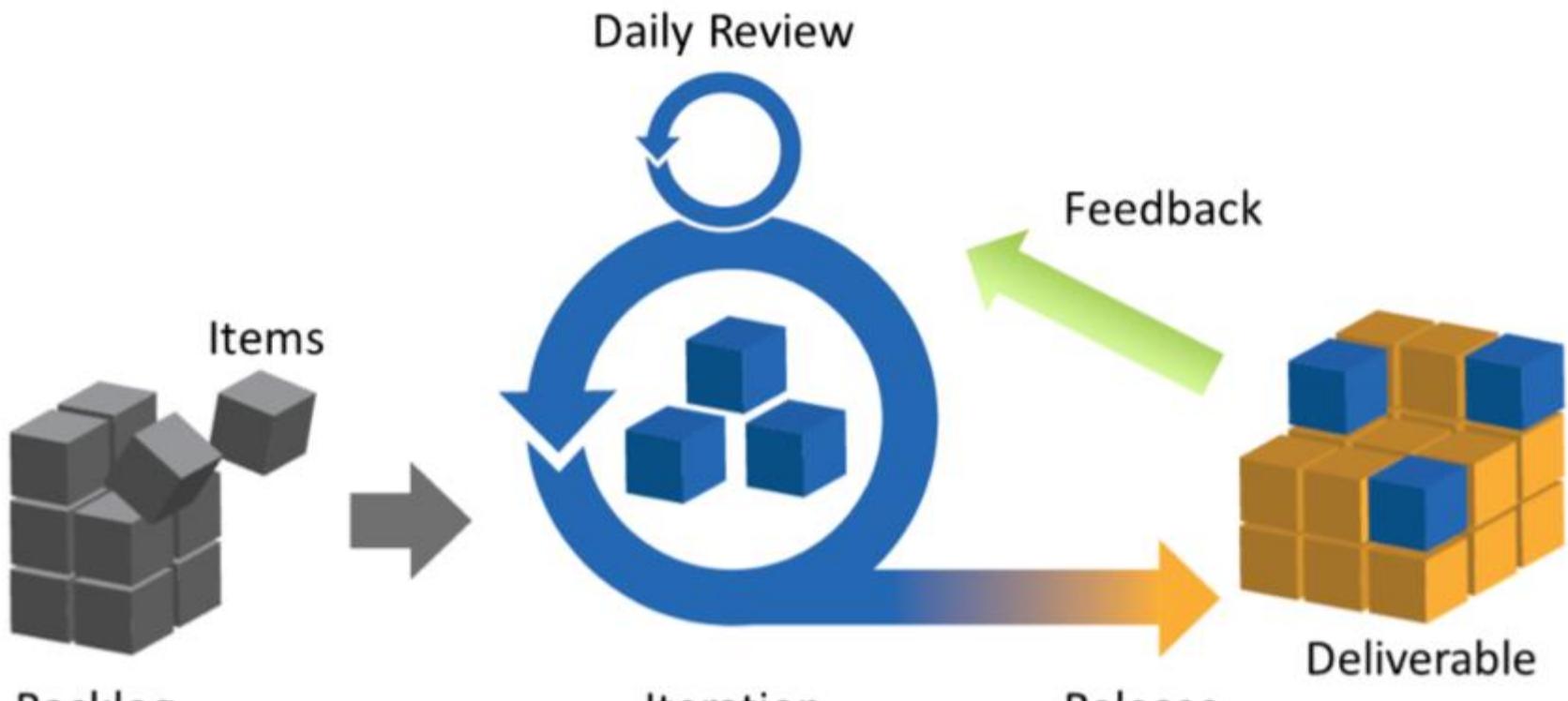
"We are uncovering better ways of developing software by doing it and helping others do it.  
Through this work we have come to value:



That is, while there is value in the items on the right, we value the items on the **left more**."

Copyright © 2016 Knowledge Train Limited. [www.knowledgetrain.co.uk](http://www.knowledgetrain.co.uk). \*Quoted from [www.agilemanifesto.org](http://www.agilemanifesto.org).

# Agile methodology



*Plan*

*Collaborate*

*Deliver*

# The 12 agile principles\*

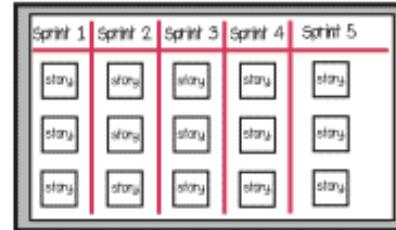
## 1 Satisfy the **customer**



## Welcome **change**



## Deliver **frequently**



## 4 Work **together**



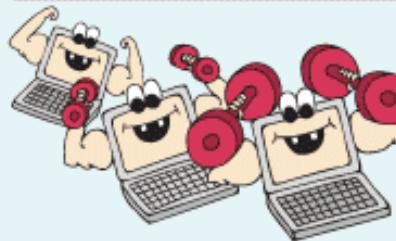
## 5 Trust and **support**



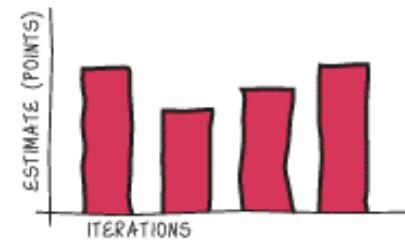
## Face-to-face **conversation**



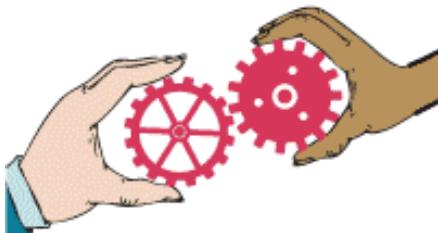
## Working **software**



## 8 Sustainable **development**



## 9 Continuous **attention**



## 10 Maintain **simplicity**



## 11 Self-organizing **teams**

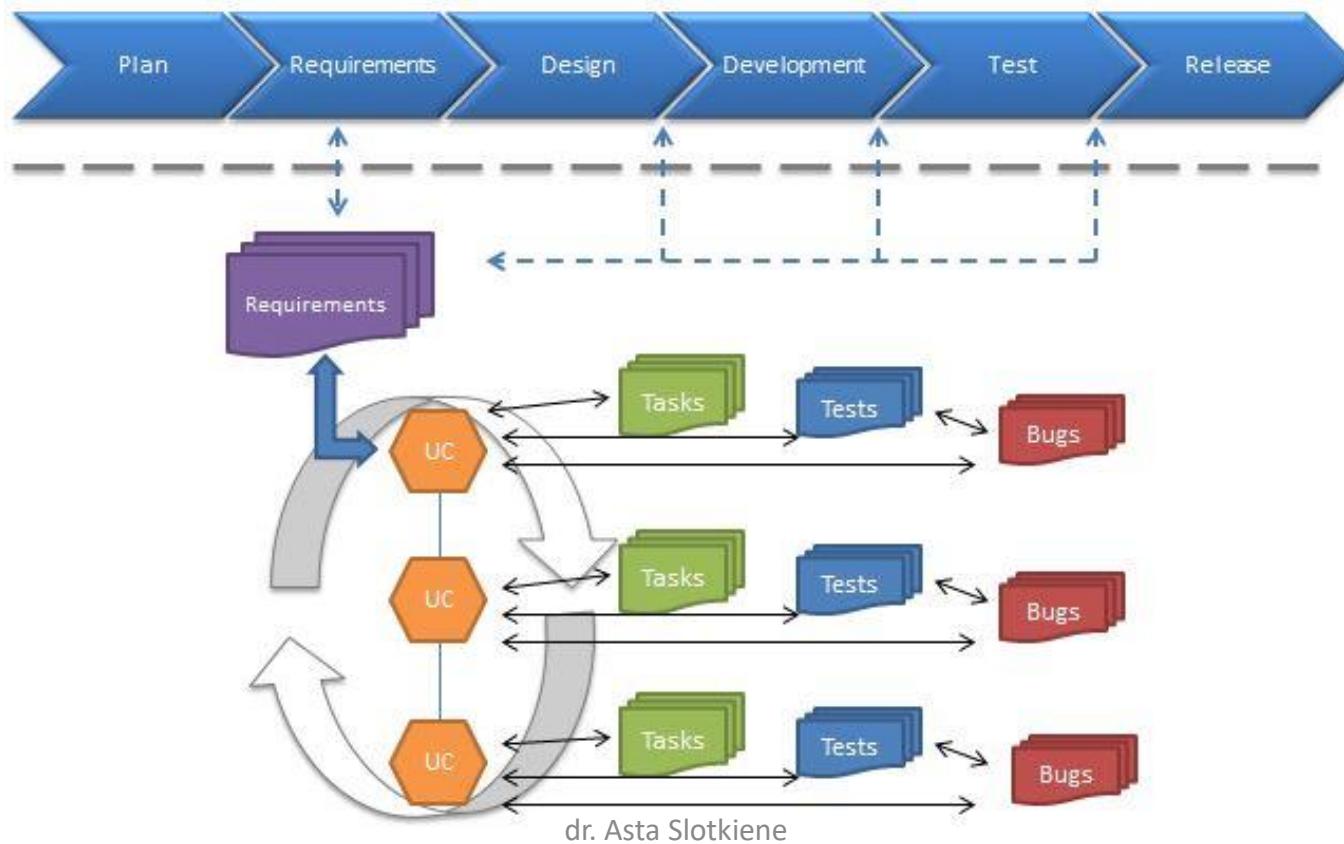


## 12 Reflect and **adjust**



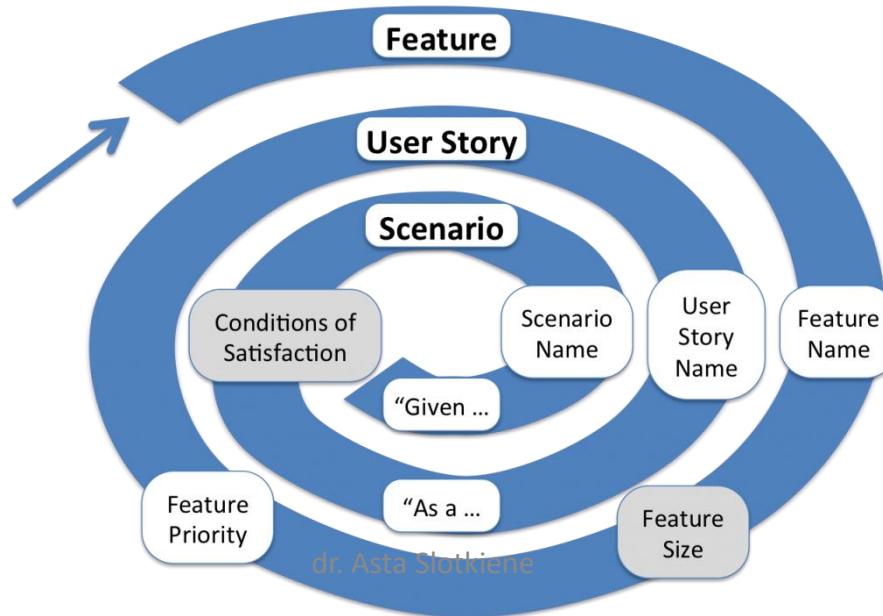
# Characteristics of Agile 1

- The processes of specification, design and implementation are interleaved.



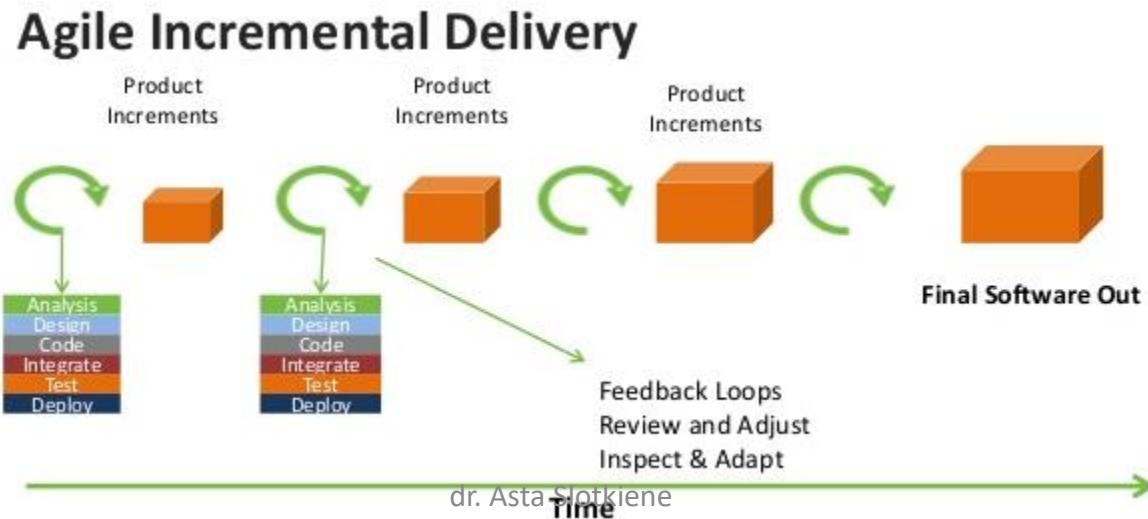
# Characteristics of Agile 2

- There is **no detailed system specification**, and **design documentation** is minimized or generated automatically by the programming environment used to implement the system.
- The user requirements document is an outline definition of the most important characteristics of the system.



# Characteristics of Agile 3

- The system is developed in a series of increments.
- End-users and other system stakeholders are involved in specifying and evaluating each increment.



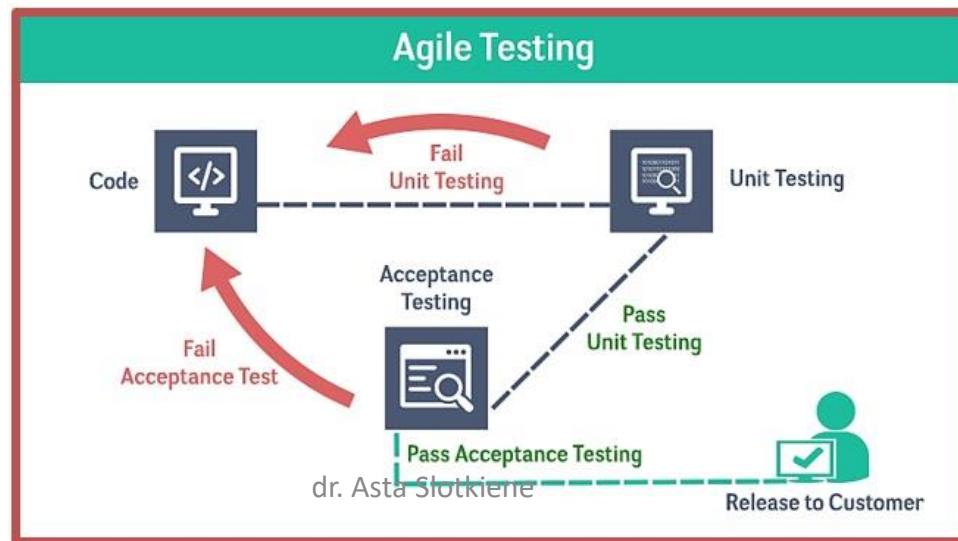
# Characteristics of Agile 4

- They may propose changes to the software and new requirements that should be implemented in a later version of the system.



# Characteristics of Agile 5

- Extensive tool support is used to support the development process.
- Tools that may be used include automated testing tools, tools to support configuration management, and system integration and tools to automate user interface production



# Characteristics of Agile 6

- They involve customers in the development process to get rapid feedback on changing requirements.
- They minimize documentation by using informal communications rather than formal meetings with written documents

# Characteristics of Agile 6

- Based on iterative and incremental development
- The increments are small, and, typically, new releases of the system are created and made available to customers every two or three weeks.



# Agile project management statistics

- Almost 71% of organizations report using Agile approaches sometimes, often, or always. ([source](#))
- Agile projects are 28% more successful than traditional projects. ([source](#))
- By 2030, artificial intelligence will automate 80% of routine Agile work ([source](#))
- 76% of users choosing an enterprise Agile planning tool do so to increase their projects' visibility. ([source](#))

# WaterFall vs Agile

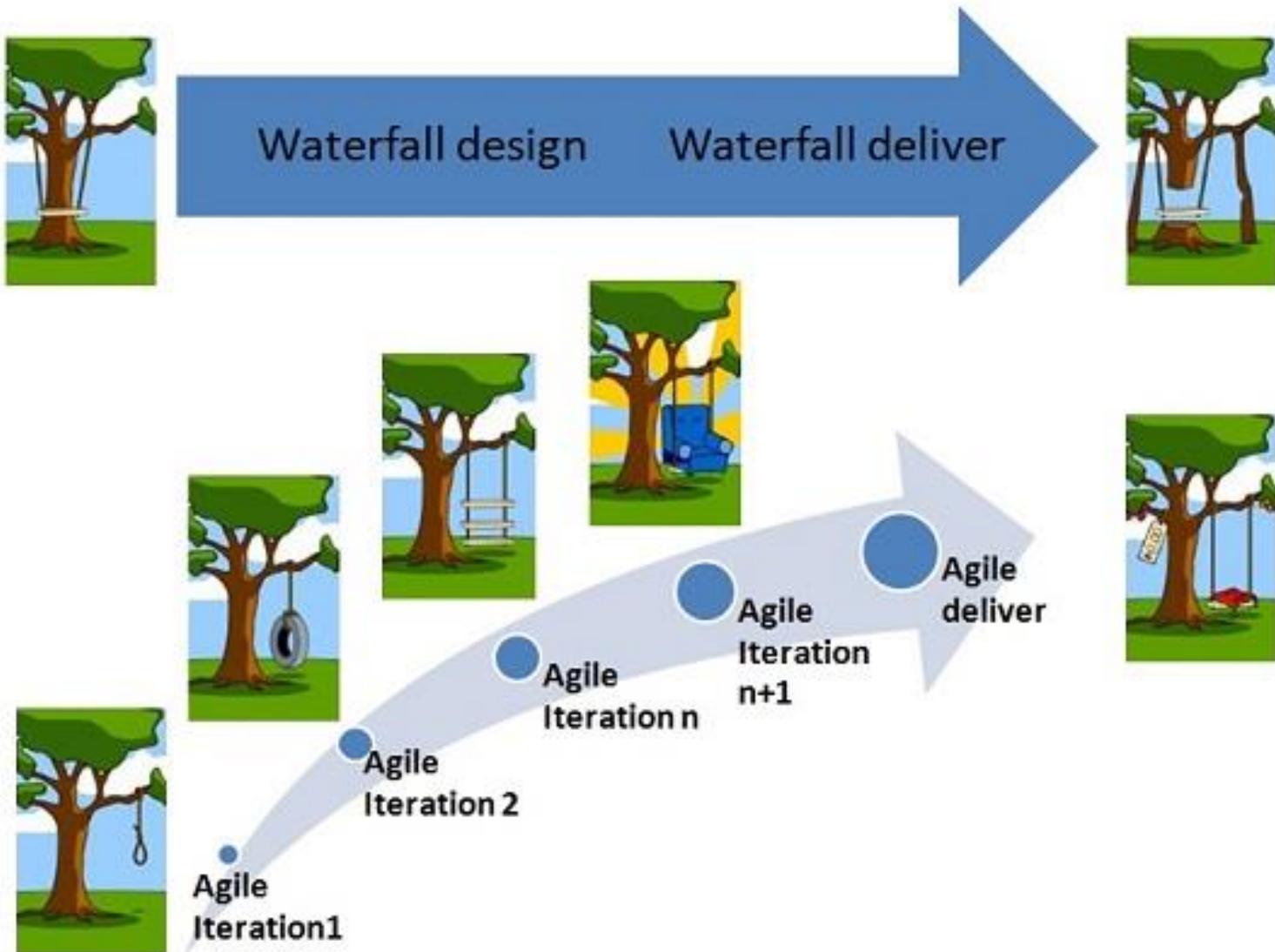
## Traditional

- Client **knows what he wants**
- Creators **know how to create**
- **Nothing will change**

## Agile

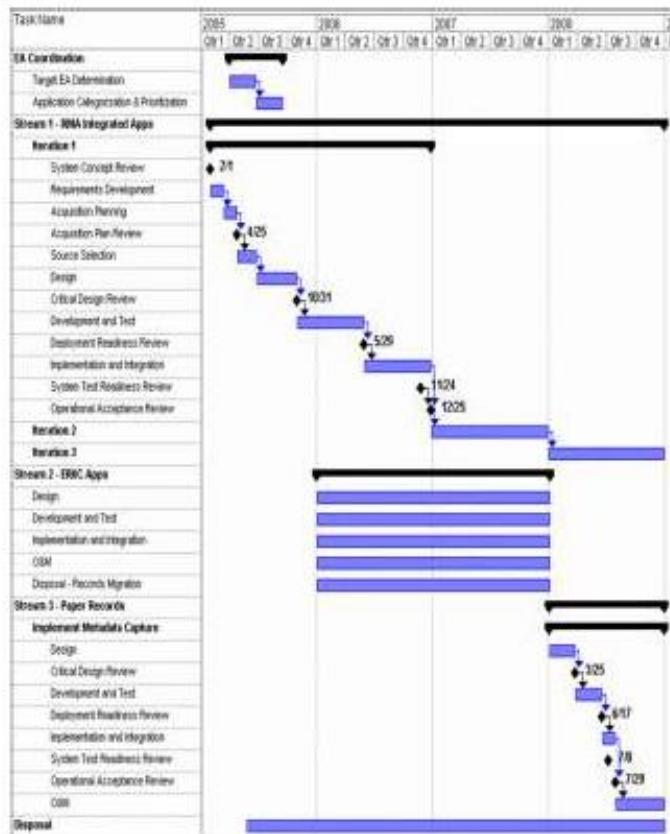
- Client will **figure out what he wants**
- Creators **will find out how to create**
- There **will be changes**

# WaterFall vs Agile



# Traditional (waterfall) project

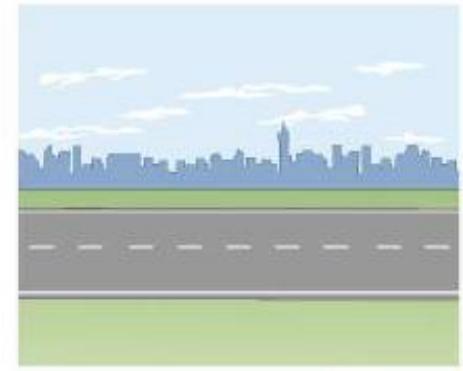
## Plan



## Requirements

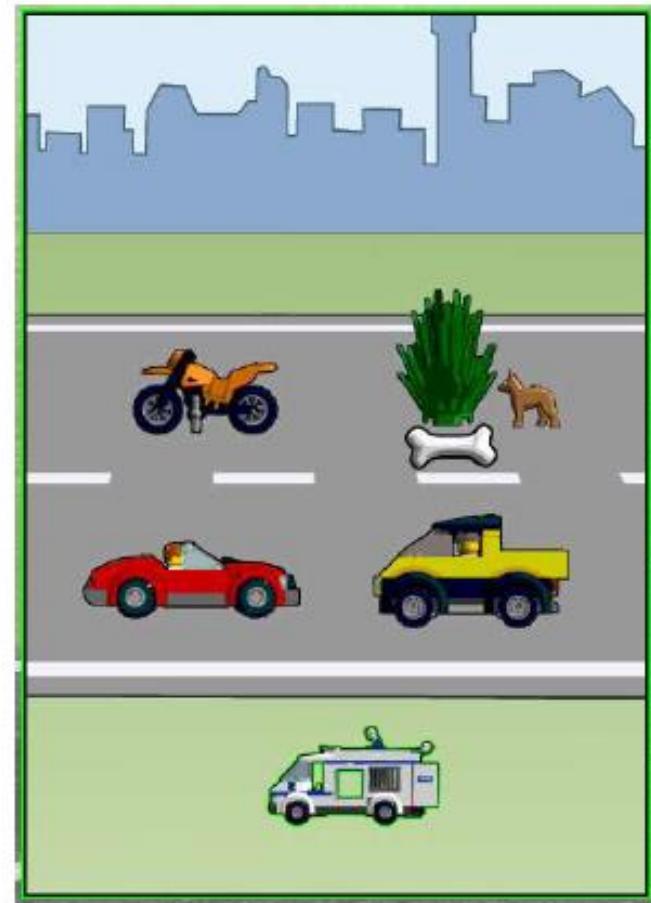


## Development



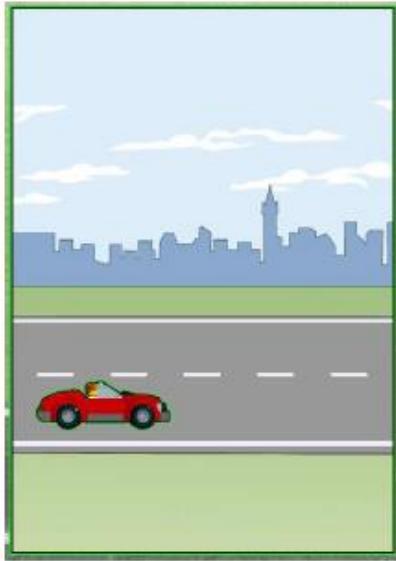
# Traditional (waterfall) project

## Integration and testing



dr. Asta Slotkiene

# Agile is process to create products or services in increments!

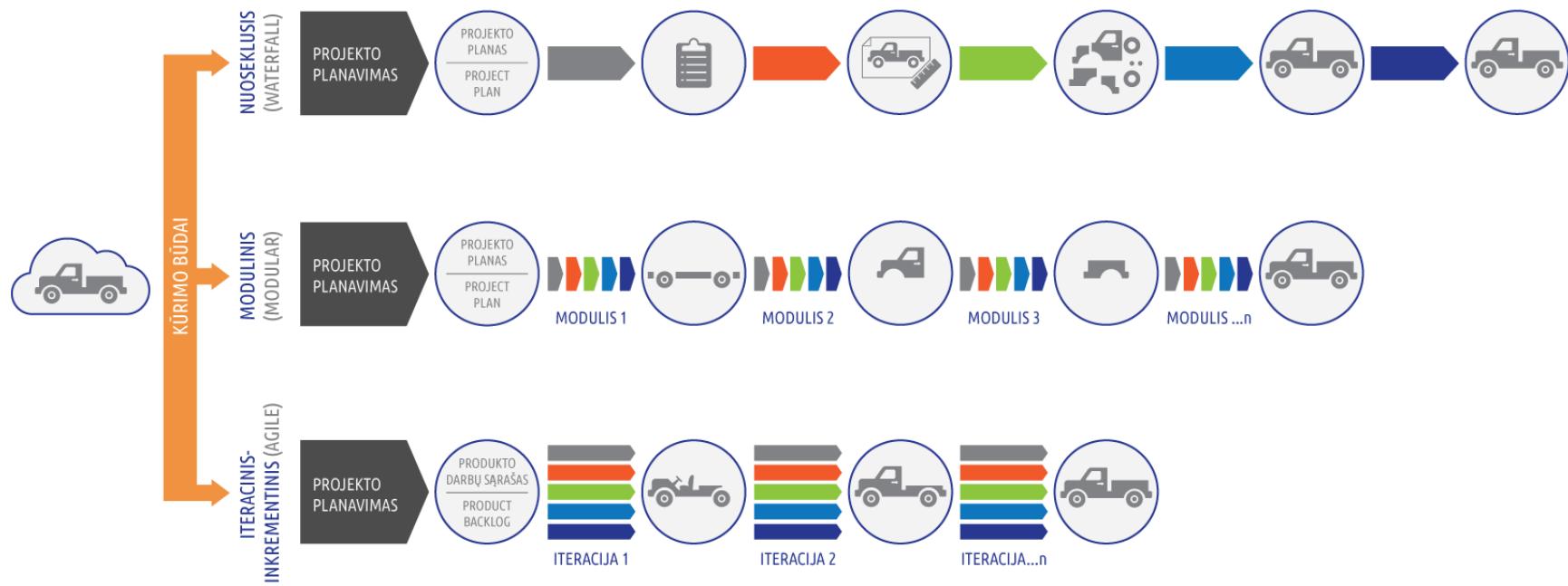


dr. Asta Slotkiene

<https://www.slideshare.net/AgileLietuva/iteracnio-inkrementinioagilemetodonaudosirrizikosuzsakovuivaidasadomauskas>

# WaterFall vs Agile

## Palyginimas pagal kūrimo etapus



### Žymėjimai:

- Analizė
- Projektavimas
- Konstravimas

- Testavimas
- Diegimas
- Rezultatas

Pradinis tikslas

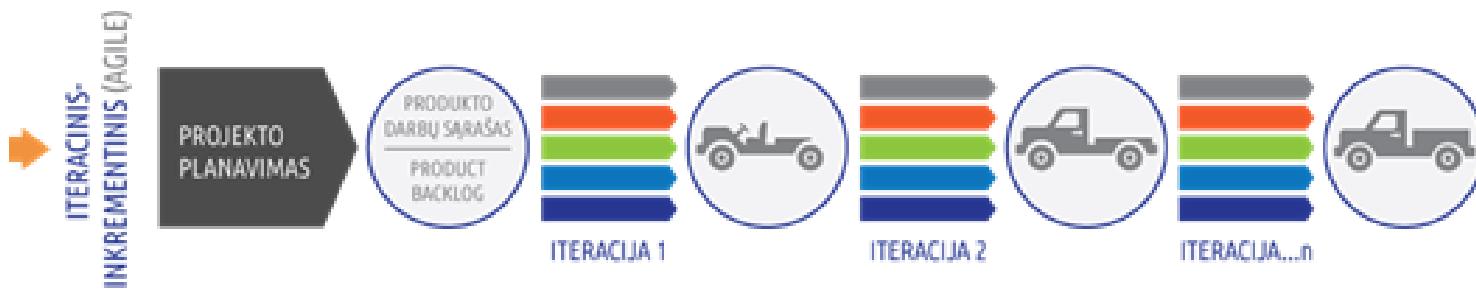


**Agile**  
LIETUVA

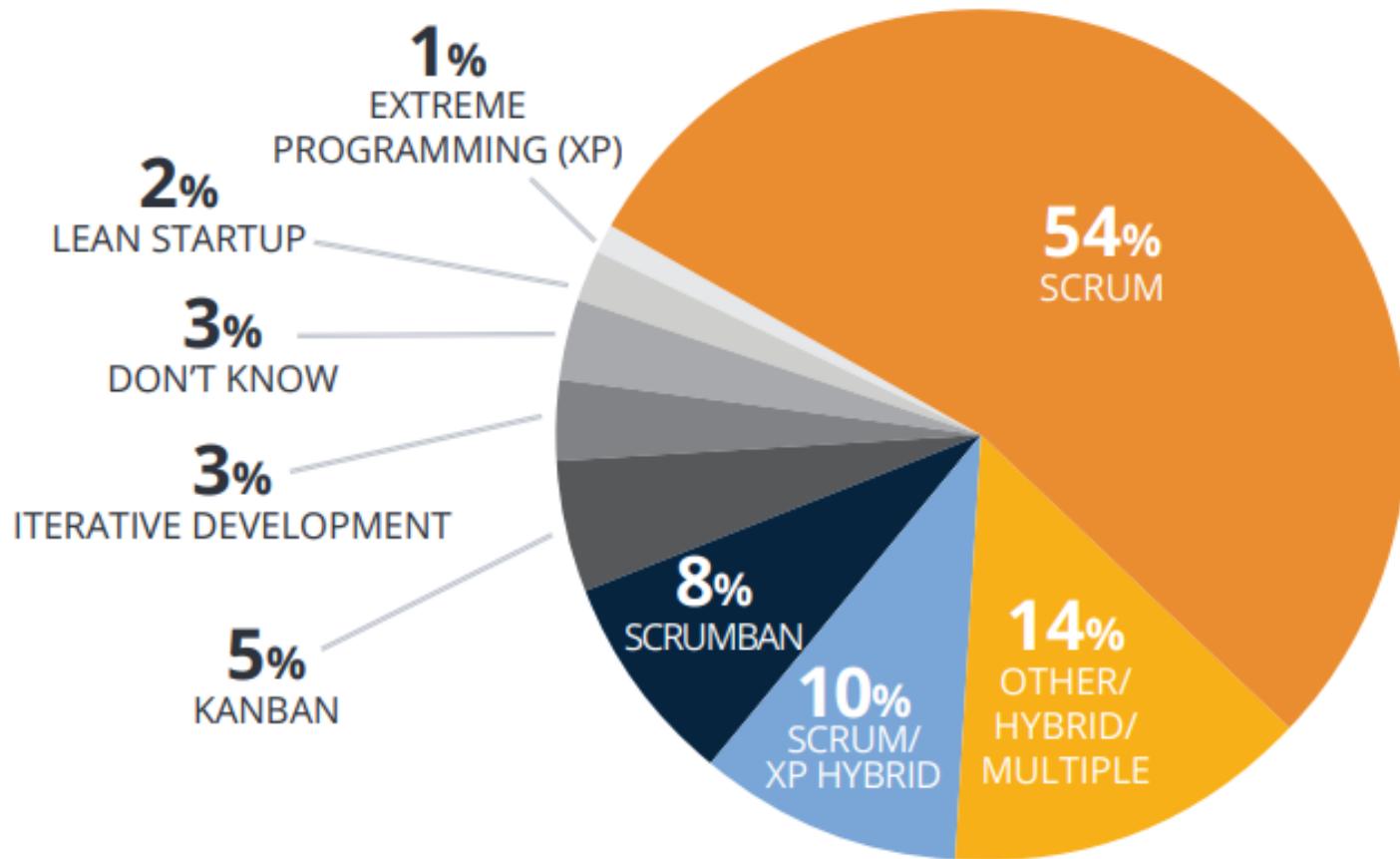
dr. Asta Slotkiene

# Iterative software development

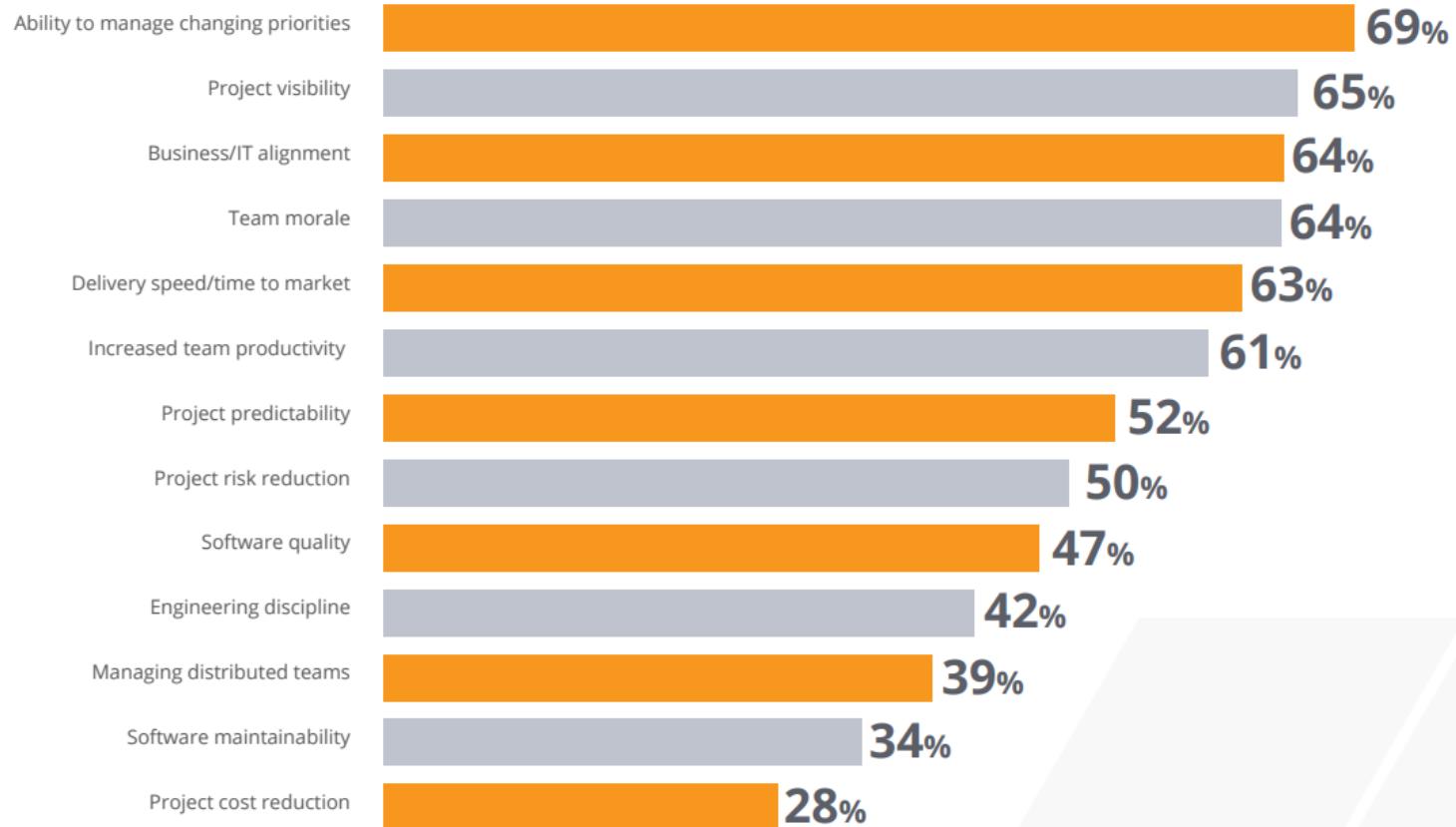
- Iterative development is an approach to building software (or anything) in which the overall lifecycle is composed of several iterations in sequence.
- Each iteration is a self-contained mini-project composed of activities such as requirements analysis, design, programming, and test.



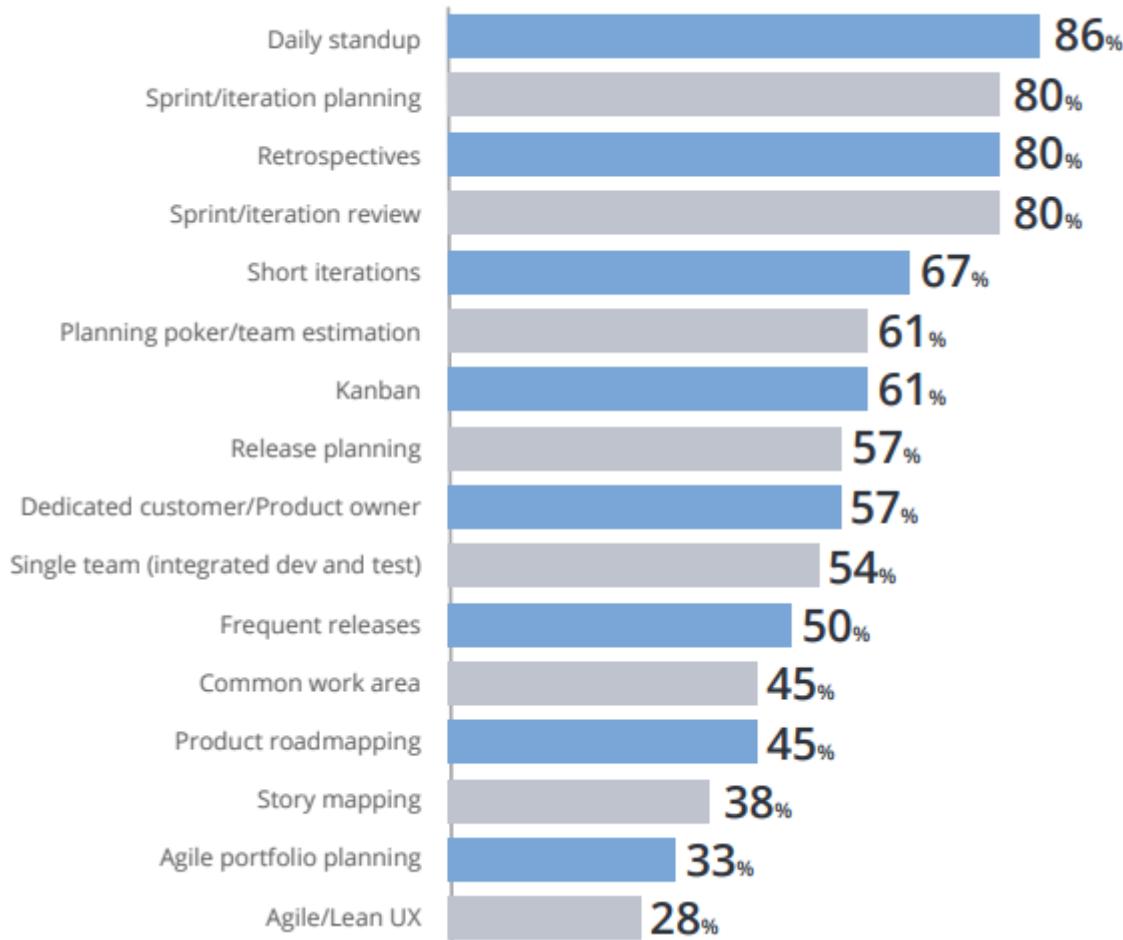
# Agile Methodologies Used



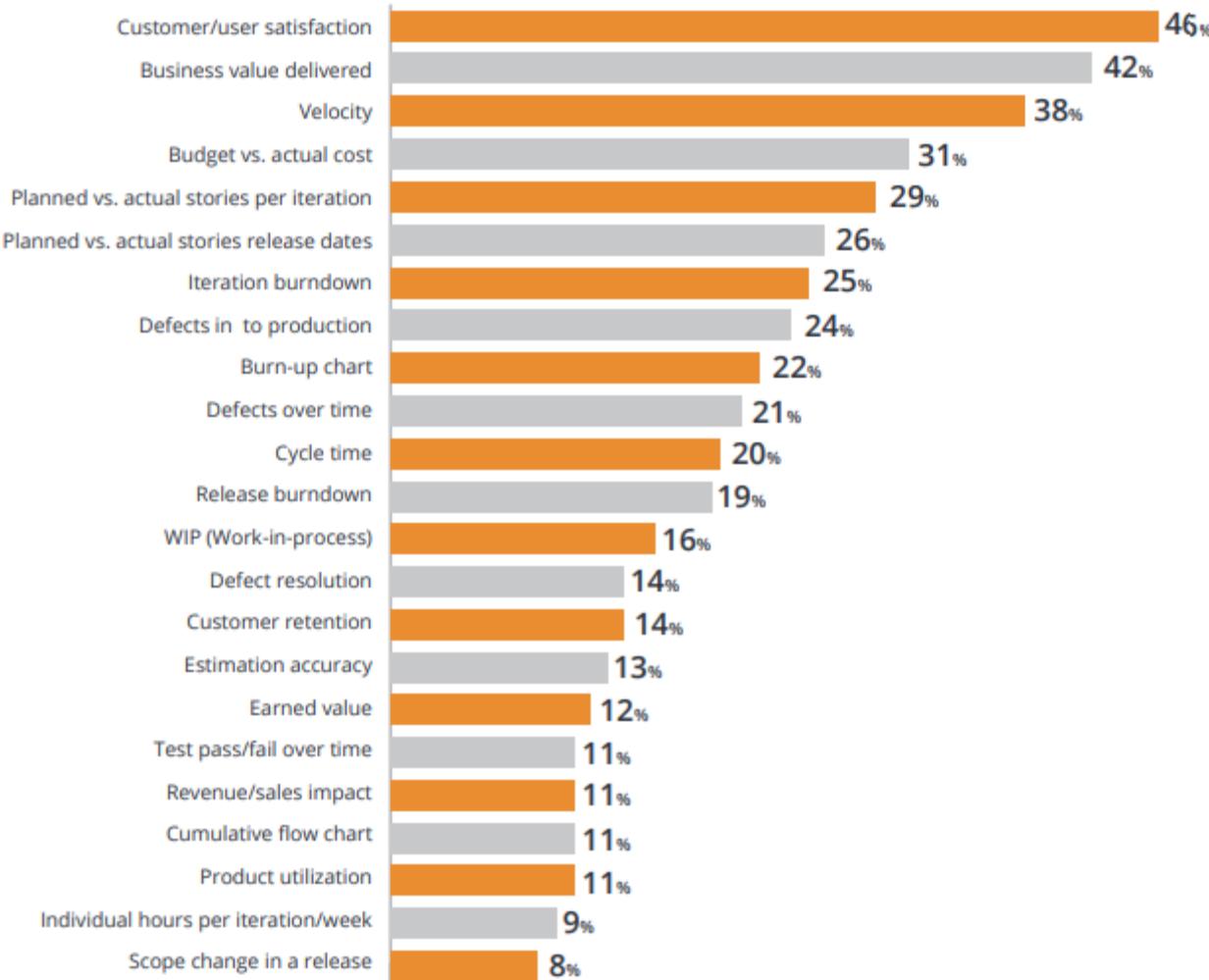
# Benefits of Adopting Agile



# Agile Techniques Employed



# How success is measured with agile



# When to apply Agile methods

1. Product development where a software company is developing a small or medium-sized product for sale.
  - Virtually all software products and apps are now developed using an agile approach.

# When to apply Agile methods

2. Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are few external stakeholders and regulations that affect the software.

# When recommended to apply Agile methods

- Agile methods may not be suitable for other types of software development:
  - embedded systems engineering
  - the development of large and complex systems

# Problems with agile methods

1. The informality of agile development is incompatible with the legal approach to contract definition that is commonly used in large companies.
2. Agile methods are most appropriate for new software development rather than for software maintenance. Yet the majority of software costs in large companies come from maintaining their existing software systems.
3. Agile methods are designed for small co-located teams, yet much software development now involves worldwide distributed teams

# What is Agile Software Development?

- Agile software development is more than frameworks such as Scrum, Extreme Programming or Feature-Driven Development (FDD).
- Agile software development is more than practices such as pair programming, test-driven development, stand-ups, planning sessions and sprints.



Scrum  
Lean software development  
Kanban (process + method)  
Extreme Programming (**XP**)  
Continuous Integration (**CI**)  
Continuous Delivery (**CD**)  
Feature Driven development (**FDD**)  
Test Driven Development (**TDD**)  
Crystal Clear  
...

Lightweight approaches

Scrum-of-Scrums  
Scrum at Scale (**Scrum@Scale**)  
Large-scale Scrum (**LeSS**)  
Scaled Agile Framework (**SAFe**)  
Disciplined Agile Delivery (**DAD**)  
Dynamic Systems Development Method (**DSDM**)  
Agile Project Management (**AgilePM**)  
Agile Unified Process (**AUP**)  
Open Unified Process (**OpenUP**)  
...

Fuller approaches (beyond 1 team)

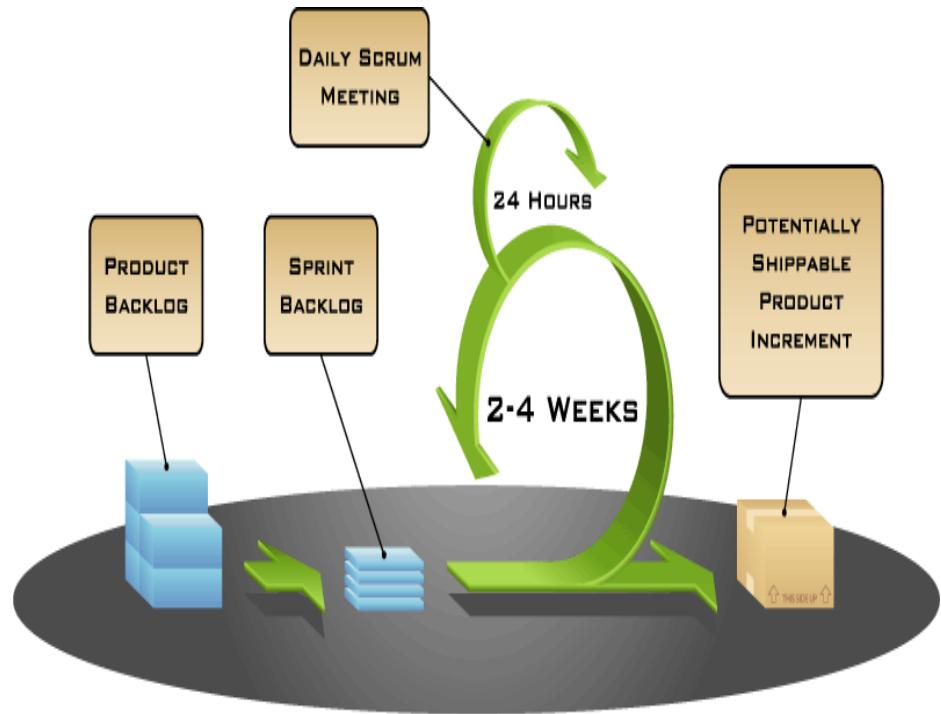
# Scrum

Scrum is a framework for developing and sustaining complex products.

- Moreover, Scrum is:
  - An empirical process
  - Lightweight
  - Simple to understand
  - Difficult to master

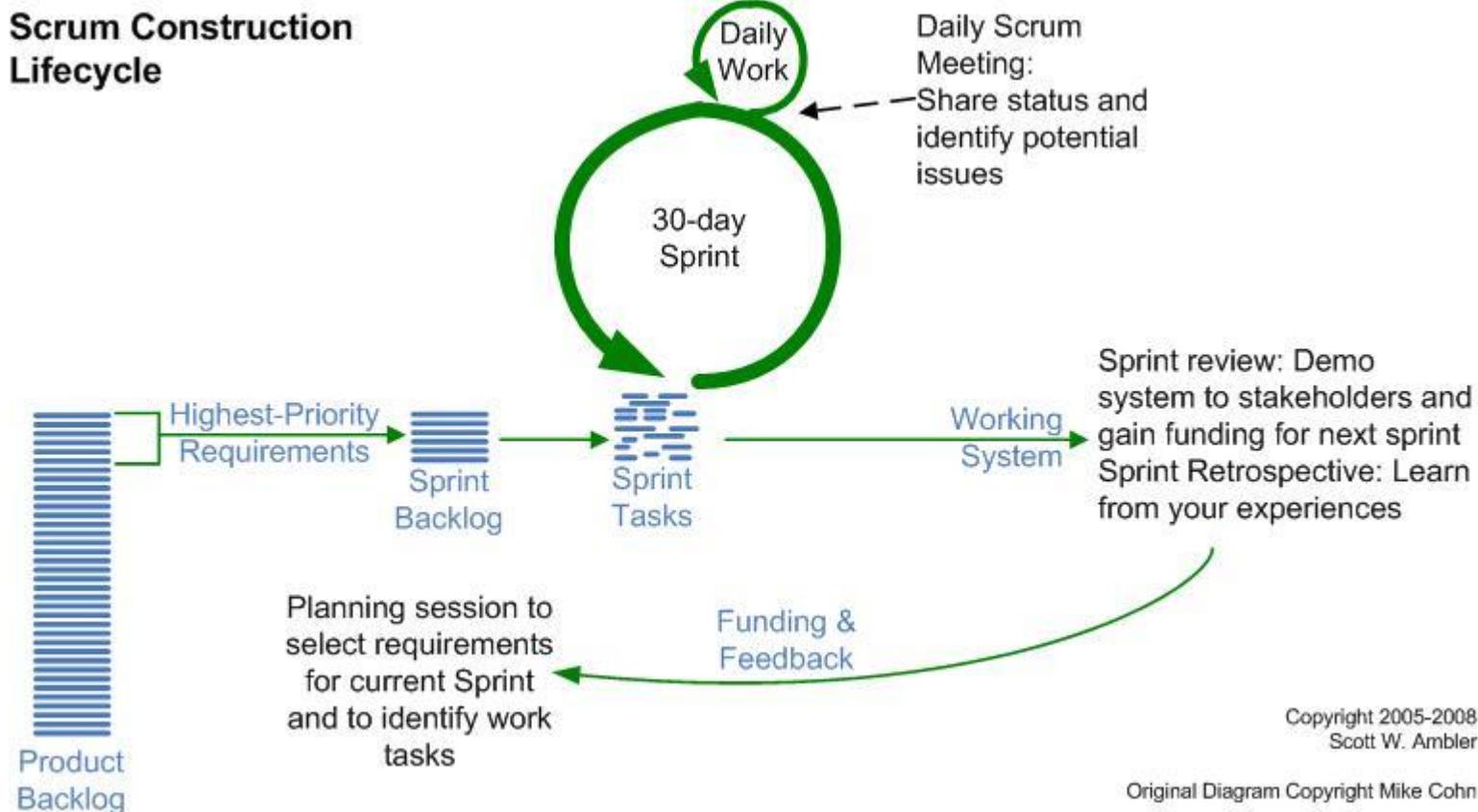
# Scrum

- Scrum is an iterative and incremental agile software development framework
- A flexible, holistic product development strategy
- Development team works as an atomic unit and is committed to a Sprint Goal
- Fixed Iterations



COPYRIGHT © 2005, MOUNTAIN GOAT SOFTWARE

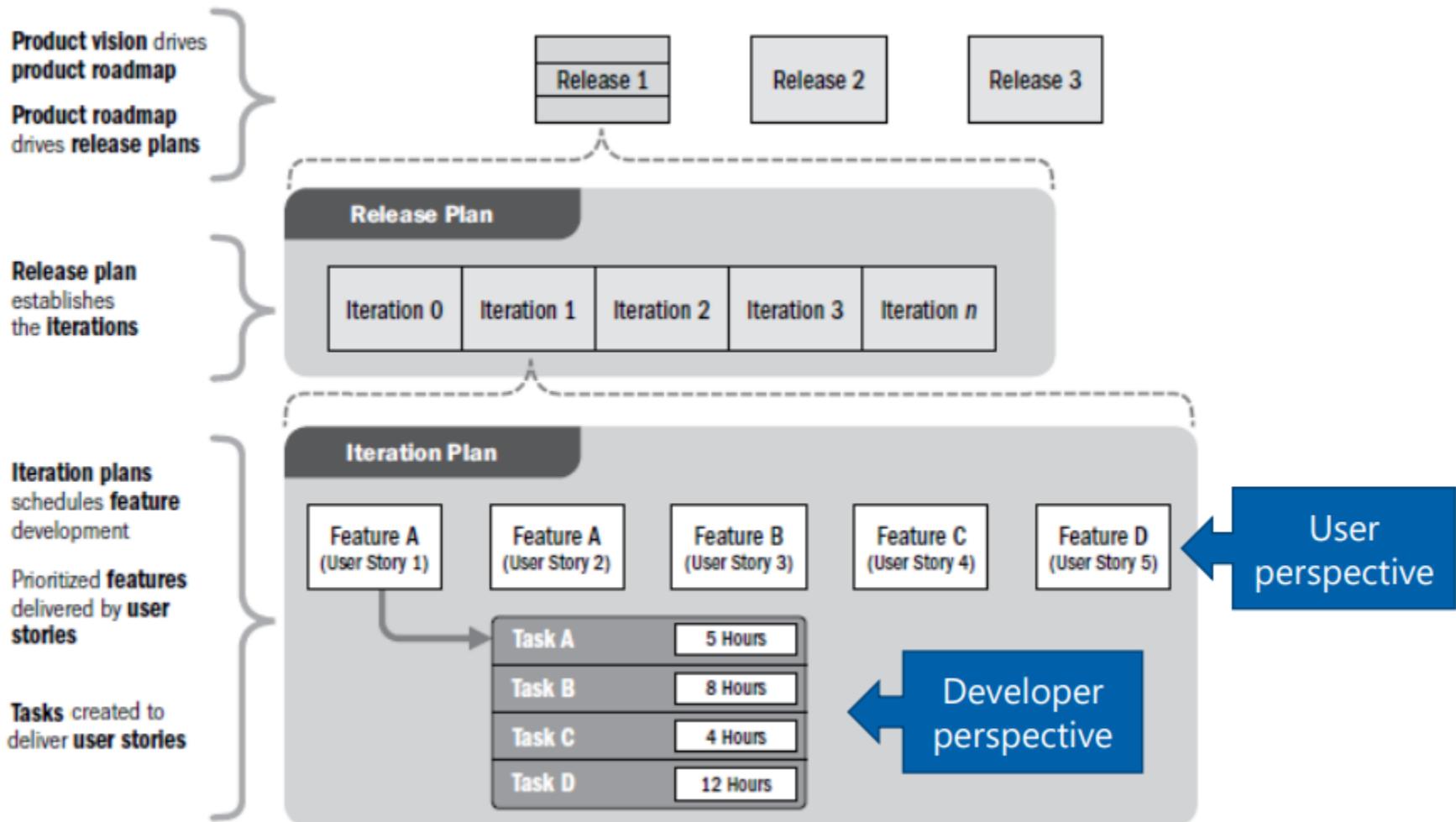
# Scrum Life Cycle



# Release planning

- A release is made up **of one or more iterations**
- Release planning refers to determining a balance between a **projected timeline and a desired set of functionality**

# Release planning

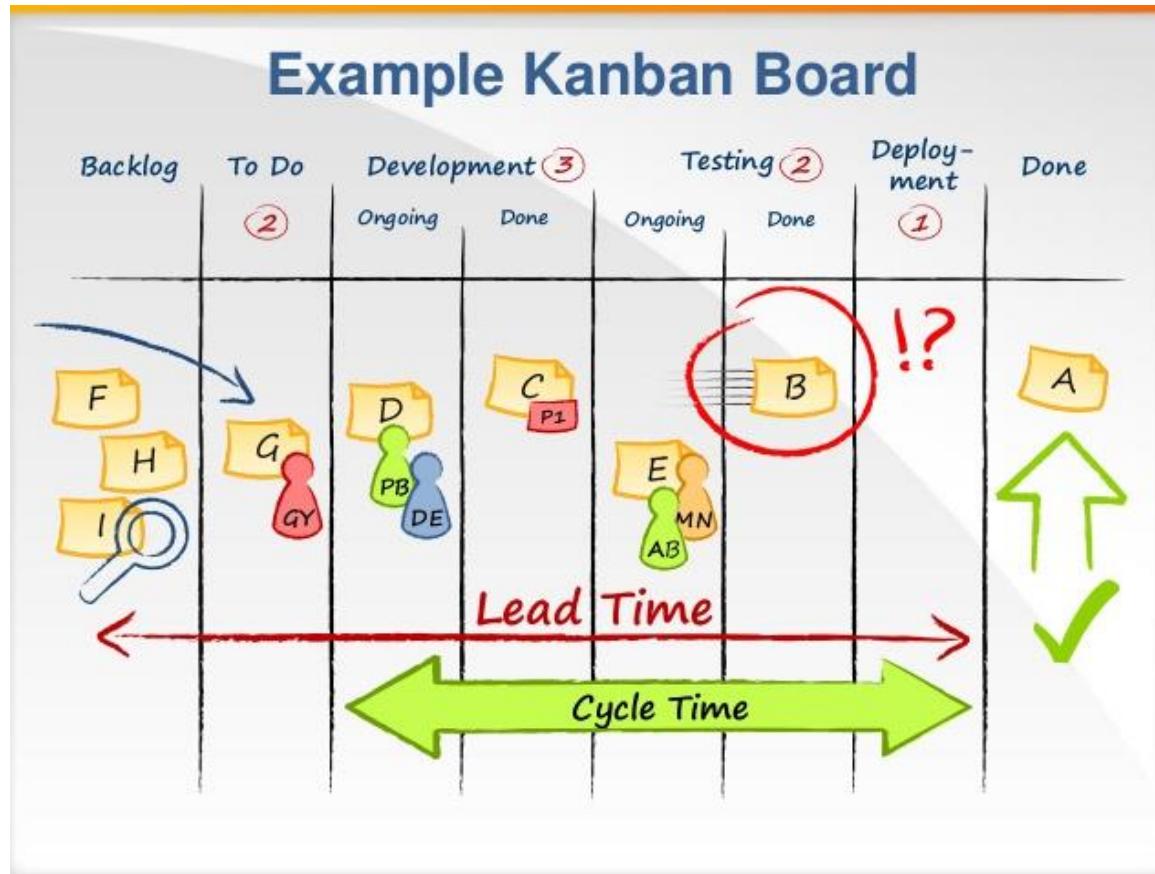


# Kanban

- Kanban is an approach for *changes management* and *process improvements* of a supply chain.



# Kanban



- Limit Work-in-Progress
- Visualize and Manage Workflow
- Measure Cycle Time

# Kanban principles

- Kanban is based on a continuous workflow structure that keeps teams nimble and ready to adapt to changing priorities
- Common workflow stages are **To Do, In Progress, In Review, Blocked, and Done.**
- In Kanban, updates are released whenever they are ready, without a regular schedule or predetermined due dates:
  - if the task gets completed earlier (or later), it can be released as needed without having to wait for a release milestone like sprint review.

# Scrum vs. Kanban

- Iteration: regular fixed length sprints (2 weeks)
  - Release: At the end of each sprint
  - Metric: Velocity
- No iteration: Continuous flow
  - Release: Continuous delivery
  - Metric: Lead time, cycle time, WIP

# Scrum vs. Kanban

- Change management
  - Teams should not make changes during the sprint.
  - During the sprint retrospective, **scrum teams should discuss how to limit change in future**, as changes put the potentially shippable increment at risk.
- Change management:
  - Change can happen at any time

# Lean Software Development

- Originates from Toyota Production System (TPS)
  - Also called Just-In-Time system
- Inspiration for TPS found in the 1950's from U.S. supermarkets
  - **Customers could get what they wanted, when they wanted it and shelves were refilled when items were about to run out.**
- The concepts transferred to the domain of software engineering by Mary and Tom Poppendieck (2003, 2007).

# Lean Principles

- |                                       |   |
|---------------------------------------|---|
| 1. <i>Eliminate waste</i>             | 1. Pašalinkite atliekas                       |
| 2. <i>Amplify learning</i>            | 2. Skatinkite/stiprinkite mokymąsi            |
| 3. <i>Decide as late as possible</i>  | 3. Nuspreškite kaip galima vėliau             |
| 4. <i>Deliver as fast as possible</i> | 4. Pristatykite produktą kaip galima greičiau |
| 5. <i>Empower the team</i>            | 5. Įgalinkite/motyvuokite komandą             |
| 6. <i>Build integrity in</i>          | 6. Integruokite                               |
| 7. <i>See the whole</i>               | 7. Įvertinkite visumą                         |

# Lean Software Development

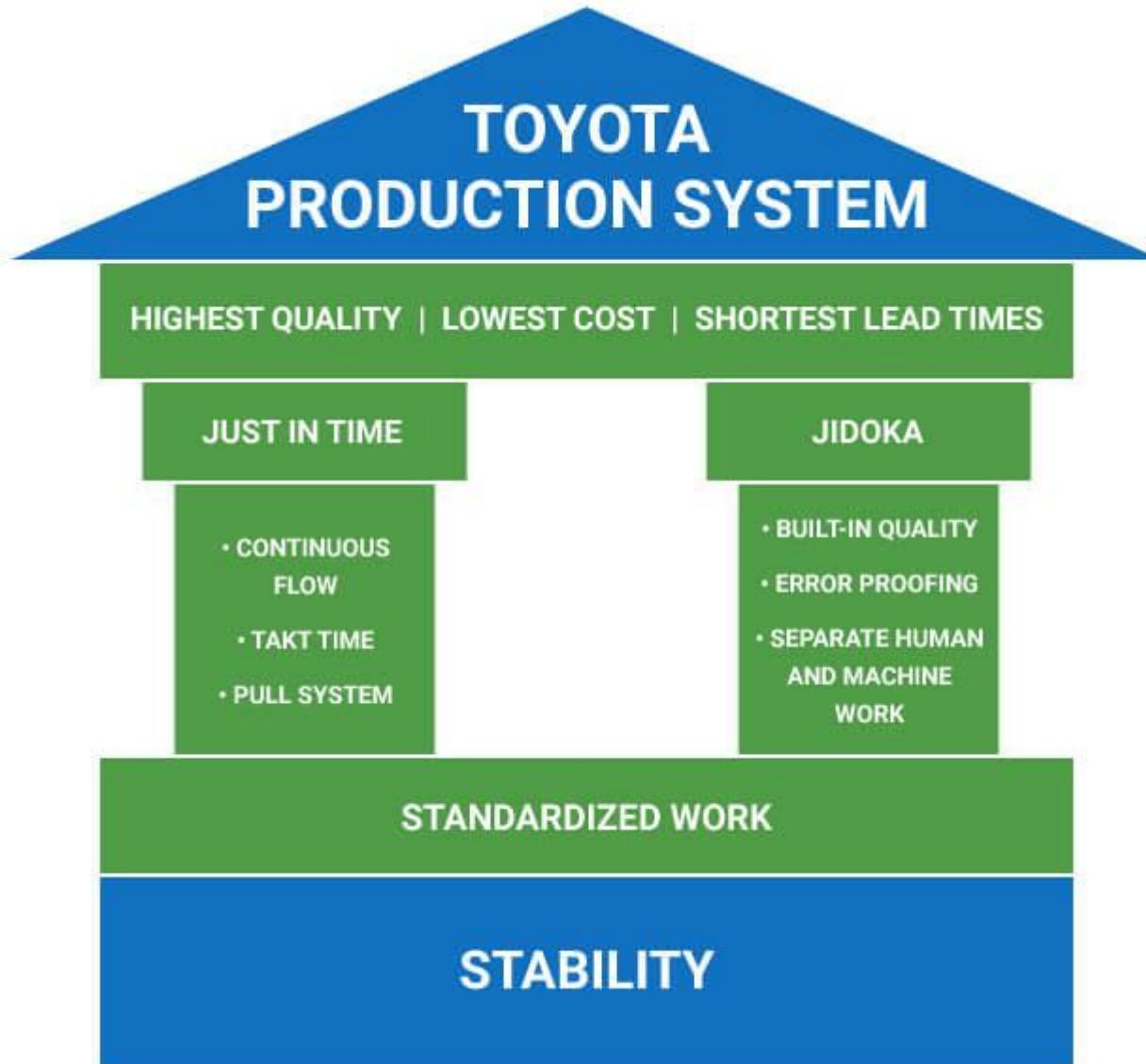
- All processes shall give **value**
- Ensure good flow in the processes to avoid **bottlenecks and queues**
- All activity shall be based on need
- Become a learning organization with focus on continuous stepwise improvement

# Lean Software Development

What do we need to learn?

1. Push and pull systems
2. Kanban
3. Systems Thinking
4. Flow
5. Work cell

# Lean Software Development



# Agile vs. Lean

- The goal of Agile is ultimately **to make the developing process flexible**, which is done by delivering in small, frequent iterations.
- The goal of Lean is **to make the developing process sustainable**, which is done by continuously **improving processes**.

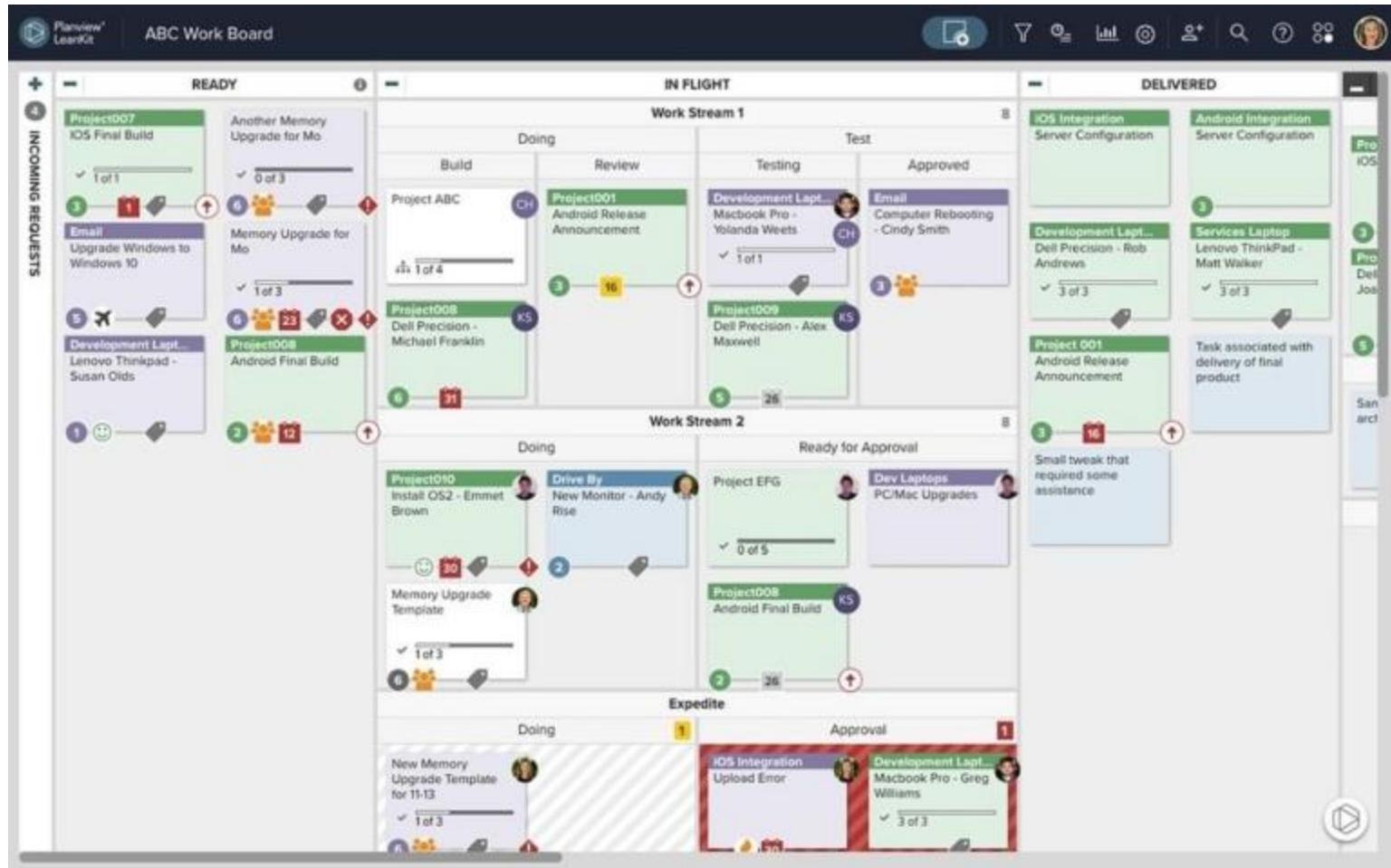
# Agile vs. Lean

- Agile teams work in **feature-focused iterations**, with a pre-defined definition of “done” as the **measure** of progress for each iteration.
- Lean teams operate in a cycle of “**Build-Measure-Learn**,” defining progress as validated learning.
- Lean development involves testing, measuring, and validating hypotheses **based on trends in the market and past work**
- When planning and prioritizing work, Lean teams **focus on efforts** that would provide greatest value to the customer.
- They continuously identify ways to reduce waste while maximizing customer value.

# Agile vs. Lean

- Scrum boards
- Kanban boards
- Lean teams use Kanban boards
- They also use Kaizen, a continuous improvement method, to habitually identify and **remove waste.**

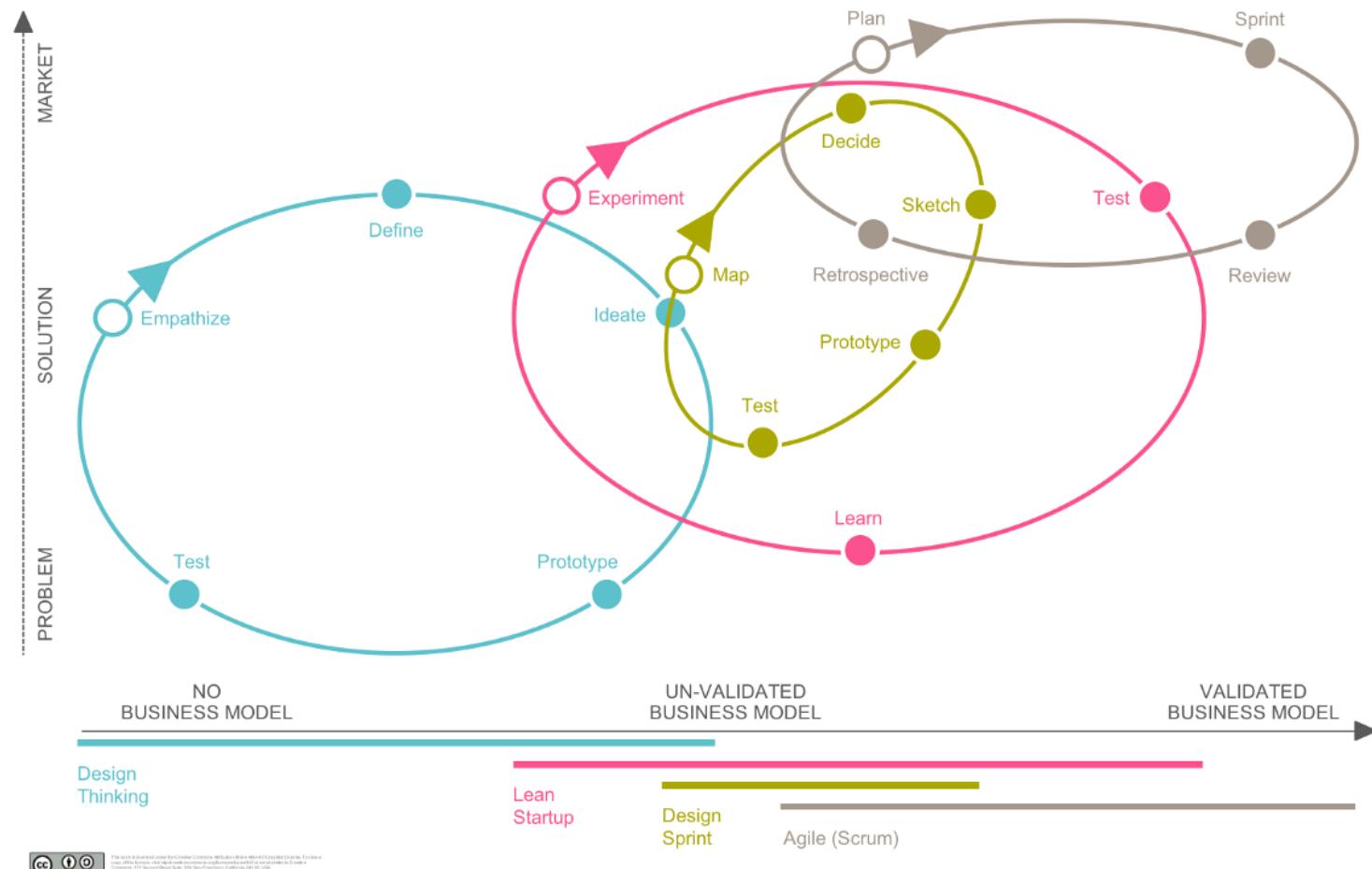
# Agile Lean board



# Agile vs. Lean

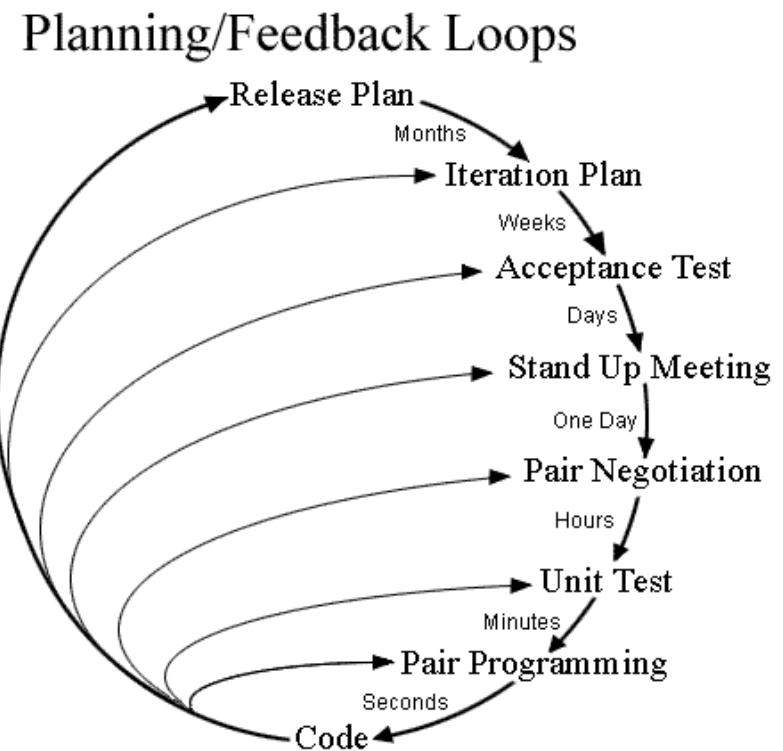
- Agile focuses on evolving products to better meet customer requirements, **but it does not address how to evolve processes to better support the evolution of the product.**
- Lean use continuous improvement for **how to evolve processes to better support the evolution of the product.**

# Design Thinking, Lean, Design Sprint, Agile



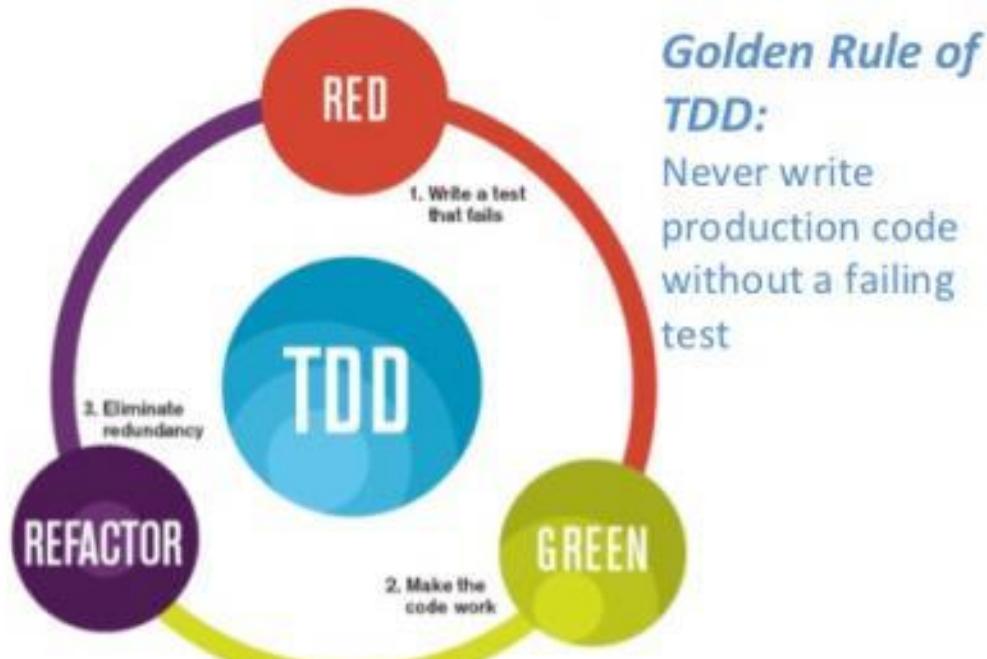
# Extreme Programming (XP)

- Improve software quality and responsiveness to changing customer requirements
- A type of agile software development
- Frequent "releases" in short development cycles
- Introduce checkpoints where new customer requirements can be adopted.



# Test-driven development (TDD)

## Test Driven Development



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

<http://reddotnews.com/articles/2007/11/01/testdriven-development-tdd.aspx>

# Test-driven development (TDD)

1. First the developer writes an (initially failing) automated test case that defines a desired improvement or new function,
2. Then produces the minimum amount of code to pass that test,
3. Finally refactors the new code to acceptable standards.

# Test-driven development (TDD)

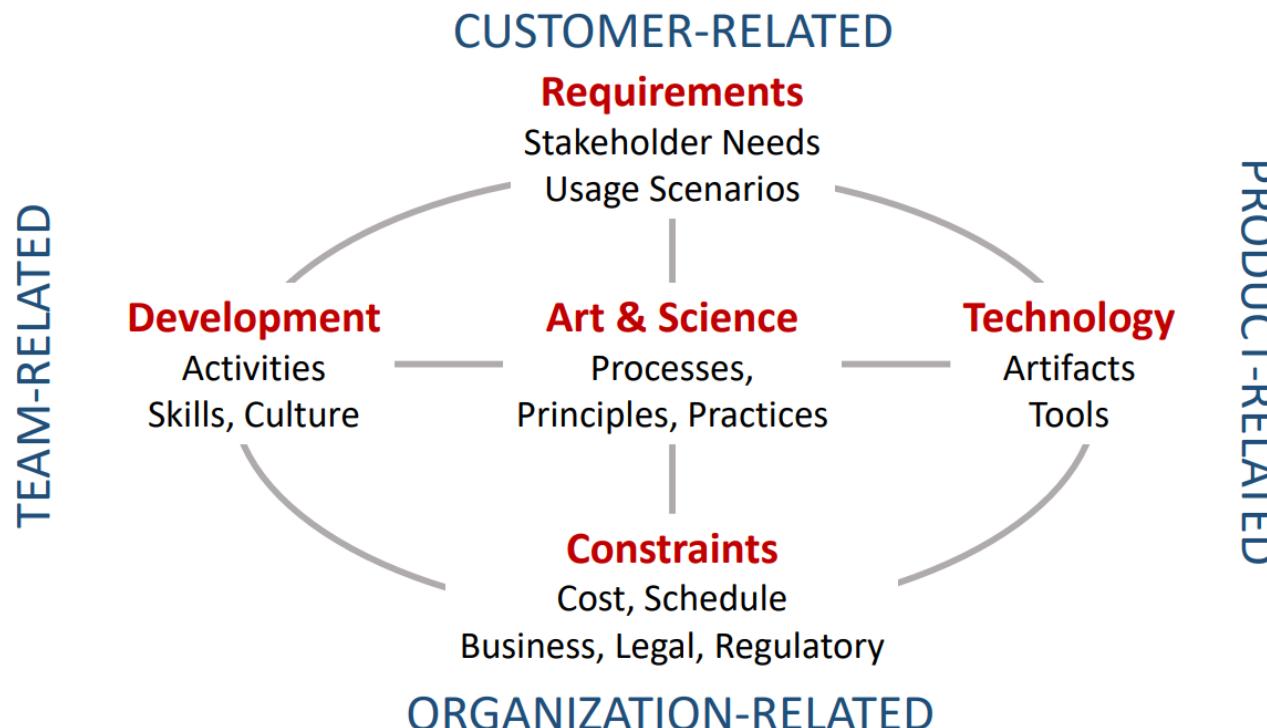
1. First the developer writes an (initially failing) automated test case that defines a desired improvement or new function,
2. Then produces the minimum amount of code to pass that test,
3. Finally refactors the new code to acceptable standards.

# Programų kūrimo procesas

Dr. Asta Slotkienė

# Software context

- The art and science of developing reliable software systems that address customer needs, subject to cost and schedule constraints



# General Terminology

- **System:**
  - Combination of interacting elements organised to achieve one or more stated purposes.  
(ISO/IEC 15288:2015)
- **Software**
  - All or part of the programs, procedures, rules, and associated documentation of an information processing system

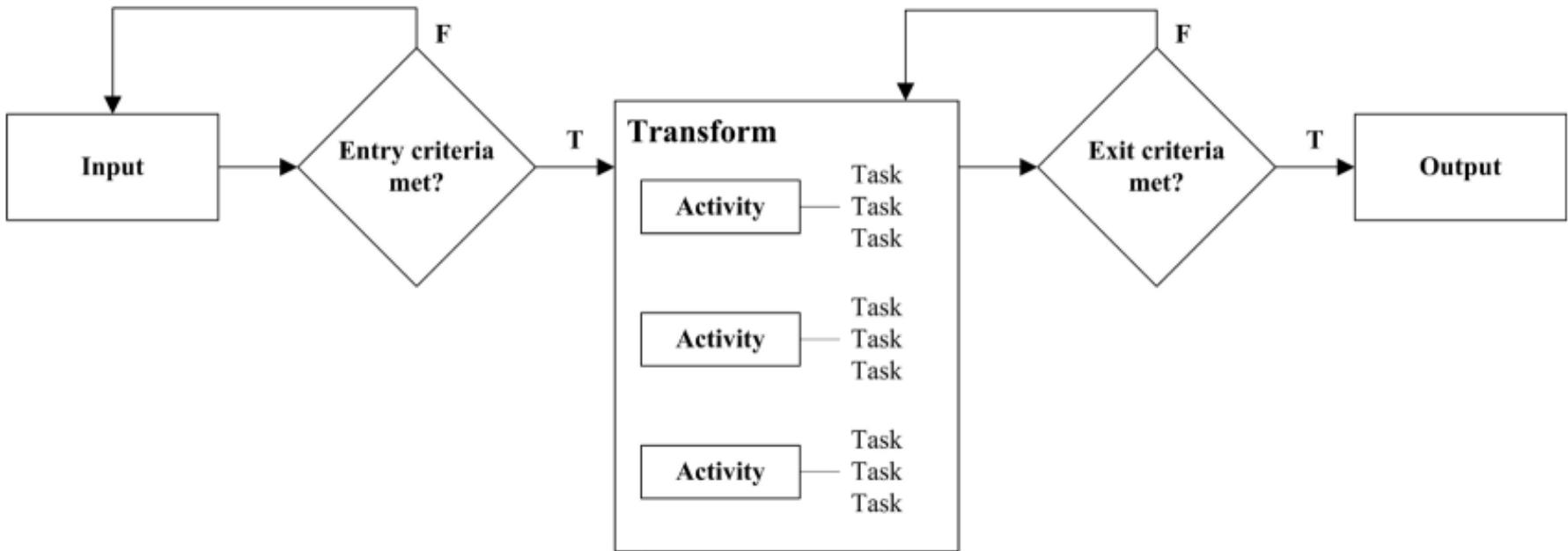
# General Terminology

- The IEEE SE definition:
  - *The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.*

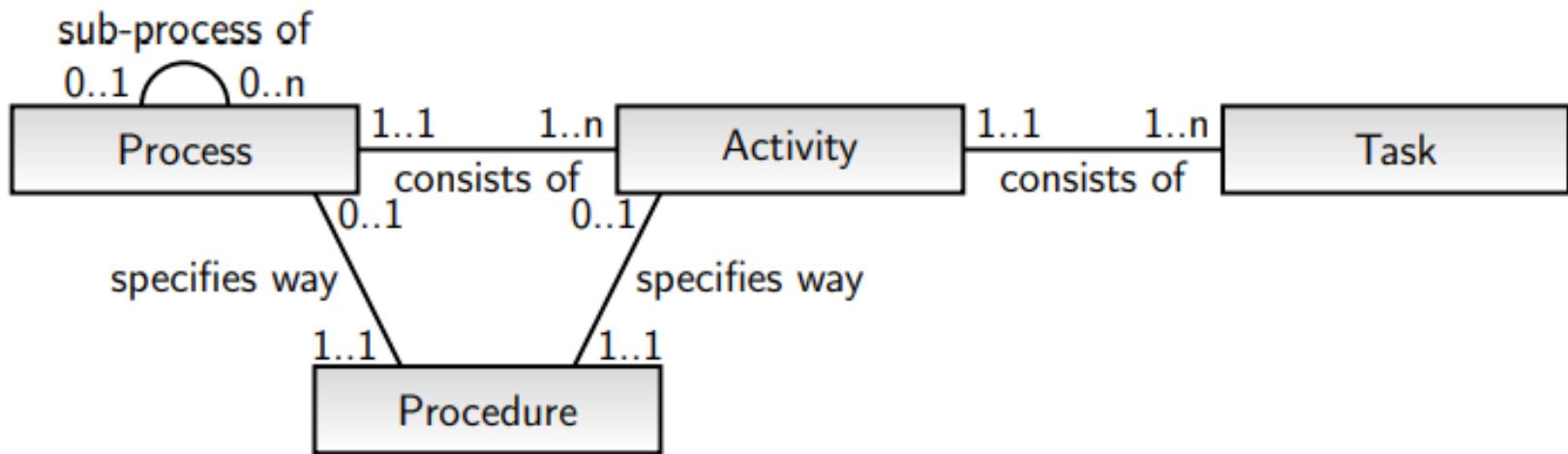
# General Terminology

- **Process**
  - **Set of interrelated or interacting activities which use inputs to deliver an intended result.**  
(ISO 9000:2015)
  - Processes are therefore often described as consisting of **input, processing and output**
  - A process may be performed, enacted or executed

# Elements of a Software Process

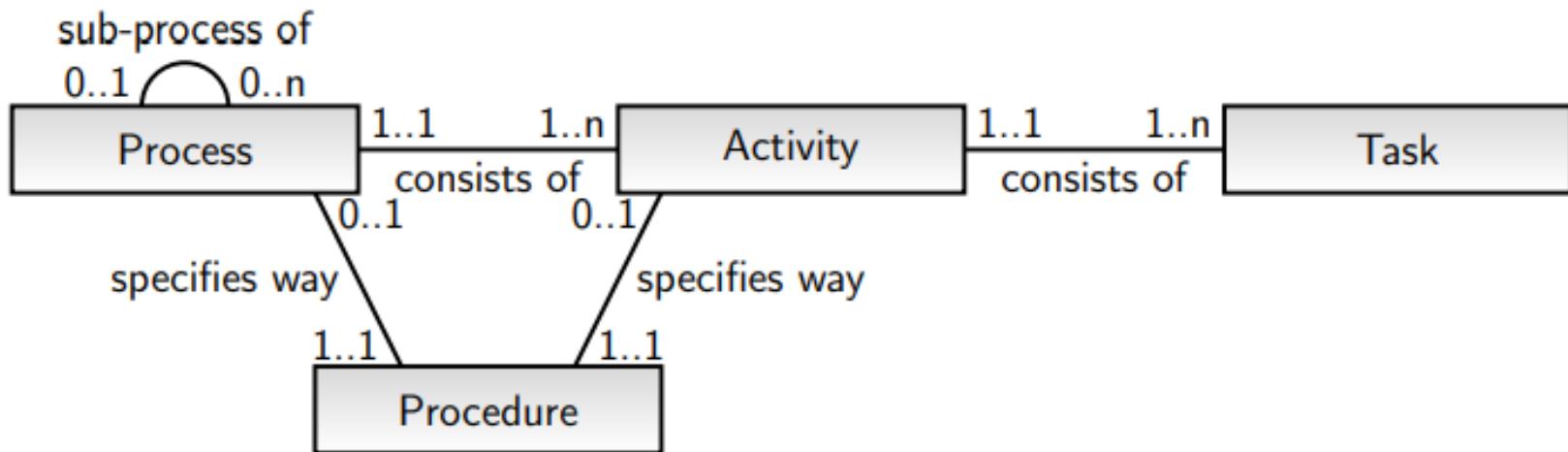


# Relationship between processes and related terms



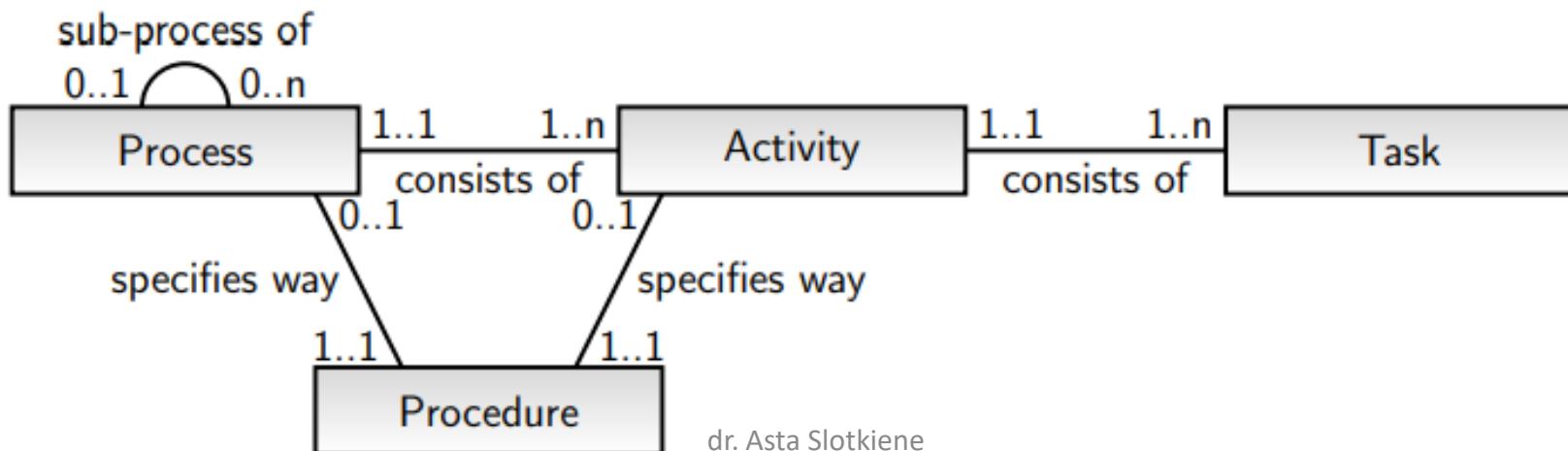
# Process Terminology

- **Activity**
    - Set of cohesive tasks of a process.
  - **Task**
    - Required, recommended, or permissible action, intended to contribute to the achievement of one or more outcomes of a process
- (ISO/IEC 12207:2008)



# Process Terminology

- **Sub-process**
  - A process that is part of a larger process.  
(CMMI R v1.3)
- **Procedure**
  - Specified way to carry out an activity or a process.  
(ISO 9000:2015)



# Example of Software Process

- **Software requirements validation** is a process used to determine whether the requirements will provide an adequate basis for software development
- It is a sub-process of the **software requirements process**.
- The tasks of the requirements validation activity might include
  - **requirements reviews**,
  - **Prototyping**
  - **Model validation**

# Example of Software Process

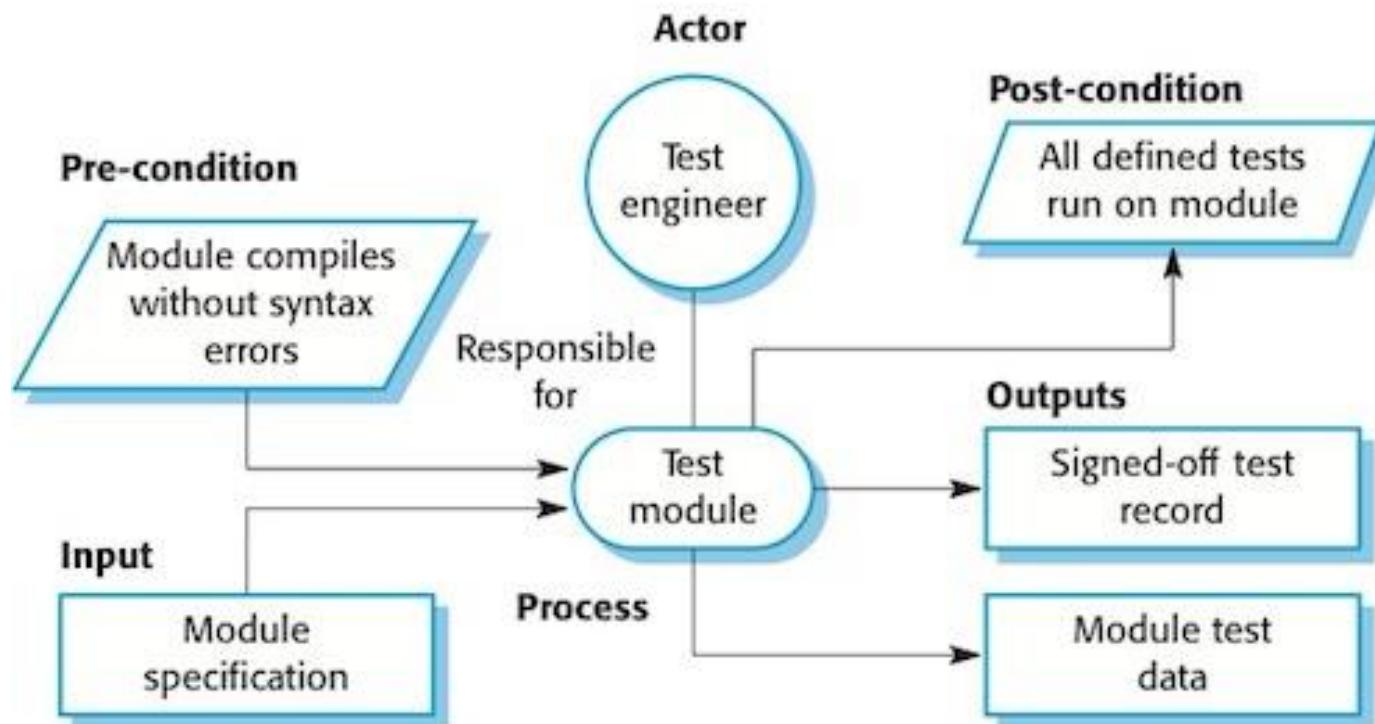
- **Inputs** for requirements validation are typically a **software requirements specification** and the resources needed to perform validation (personnel, validation tools, sufficient time).
- **The output** of requirements validation is typically a **validated software requirements specification** that provides inputs to the software design and software testing processes.

# Example of Software Process

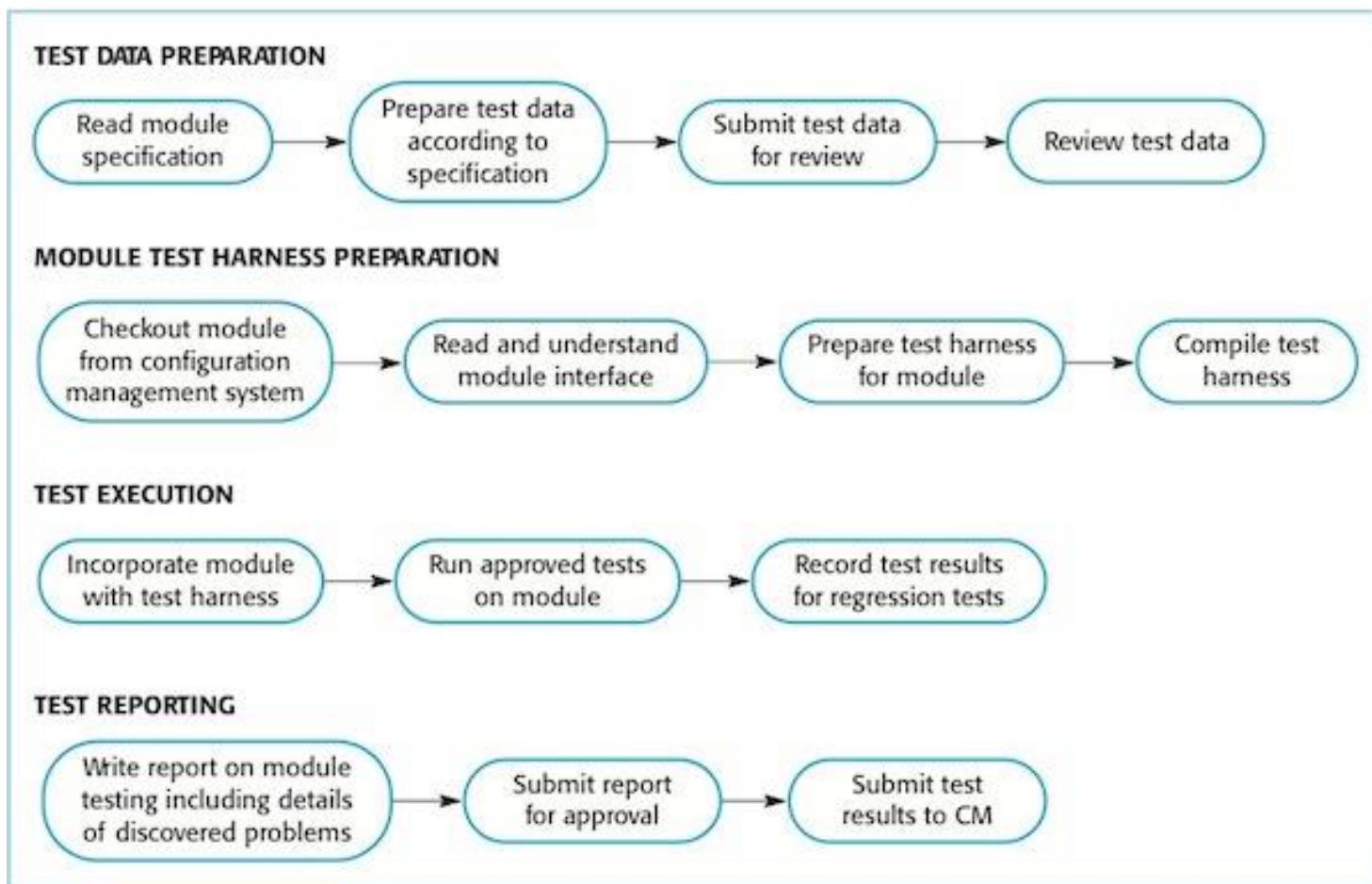
- A software process also include:
  - roles
  - competencies,
  - IT support,
  - software engineering techniques
  - software engineering tools,
  - work environment needed to perform the process,
  - approaches and measures used to determine the efficiency and effectiveness of performing the process.

In addition, a software process may include interleaved technical, collaborative, and administrative activities

# Example of Software Process



# Example of Software Process



# Software Process Terminology

- Life cycle
  - Evolution of a system, product, service, project or other human-made entity **from conception through retirement.** (ISO/IEC 12207:2008)
- Life cycle model
  - **Framework of processes** and activities concerned with the life cycle, which can be organised into stages, and which acts as a common reference for communication and understanding.  
(ISO/IEC 12207:2008)

# Software Process Terminology

- Software process
  - A process that forms part of the life cycle of software.

## Life cycle model

- Framework of processes and activities concerned with the life cycle, which can be organised into stages, and which acts as a common reference for communication and understanding.

(ISO/IEC 12207:2008)

- requirements analysis,
- design,
- implementation,
- maintenance and operations,
- quality assurance
- project planning

# Software Process Terminology

## Software development process

1. Process by which user needs are translated into a software product.
2. A software process that is performed (fully or at least partly) during the development part of the software life cycle.

# Software Process Terminology

## Software development life cycle (SDLC)

- Life cycle that includes the software development processes used to **specify software requirements and transform them into a deliverable software product.**

(Based on SWEBOK)

# Software Process Terminology

- Software (product) life cycle (SPLC))
  - Life cycle that includes the software development life cycle
    - plus additional software processes that provide for deployment, maintenance, support, evolution, retirement, and all other inception-to-retirement processes for a software product,
    - including the software configuration management and software quality assurance processes that are applied through out a software product life cycle.

(Based on SWEBOK)

# Software processes and software development

- Software development is a complex task, consisting of many individual activities that **need to be coordinated**.
- This is where **software processes and life cycle models** come into play, providing an infrastructure to coordinate and manage these various activities.

# SWEBOK 3.0

1. Software requirements
2. Software design
3. Software construction
4. Software testing
5. Software configuration management
6. Software quality

# The Rational Unified Process

## **Development Disciplines**

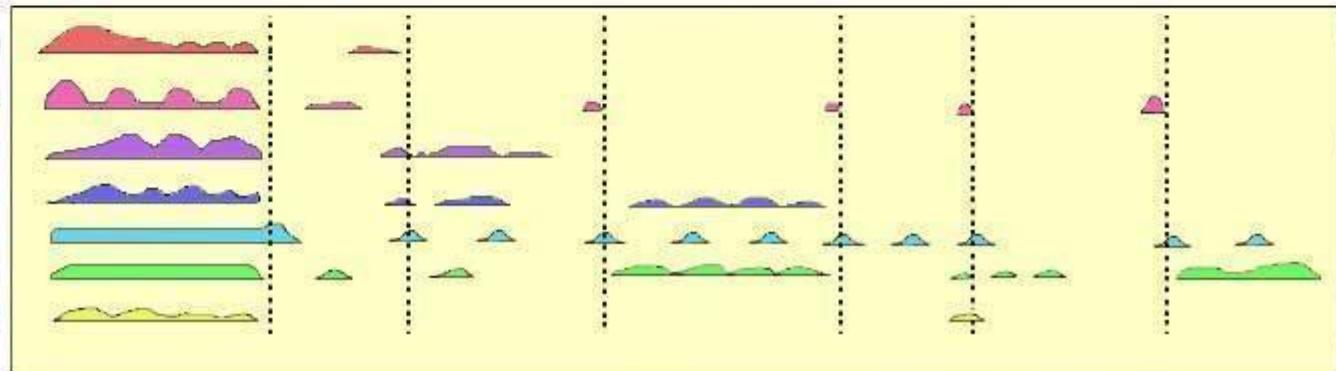
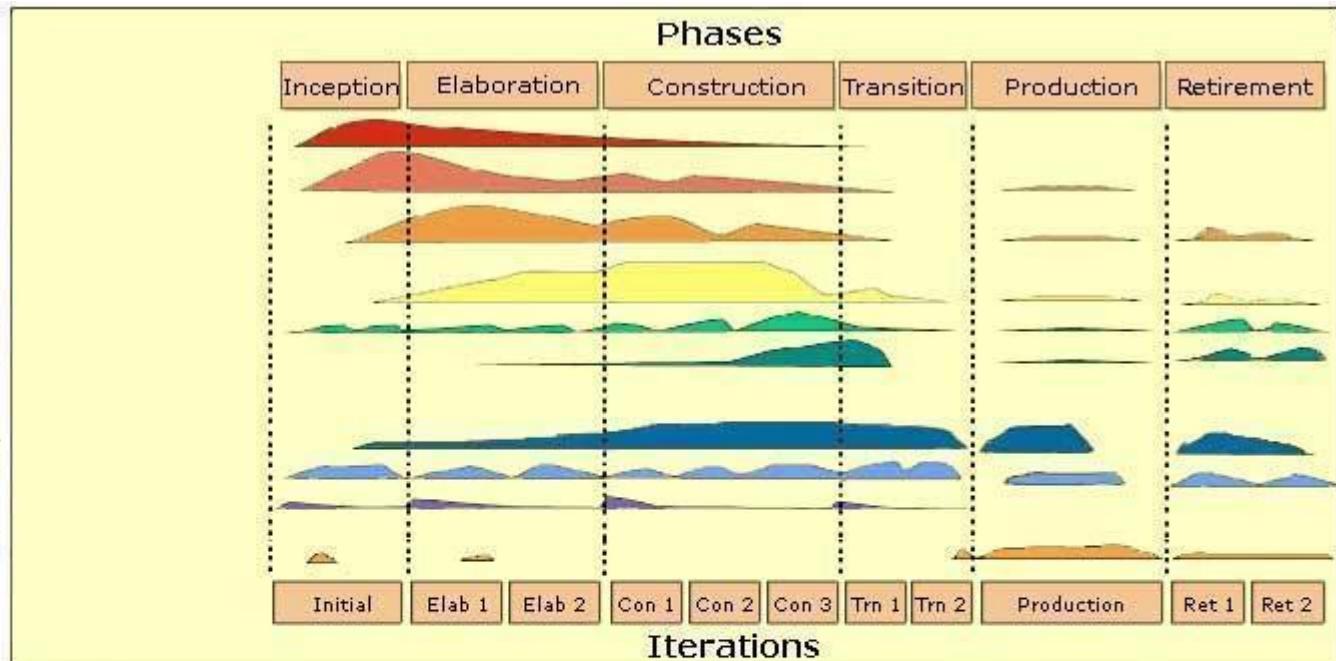
Business Modeling  
Requirements  
Analysis & Design  
Implementation  
Test  
Deployment

## **Support Disciplines**

Configuration and Change Mgmt.  
Project Management  
Environment  
Operations & Support

## **Enterprise Disciplines**

Enterprise Business Modeling  
Portfolio Management  
Enterprise Architecture  
Strategic Reuse  
People Management  
Enterprise Administration  
Software Process Improvement



# The Rational Unified Process

- Business modelling
  - The business processes are modelled using business use cases.
- Requirements
  - Actors who interact with the system are identified and use cases are developed to model the system requirements.
- Analysis and design
  - A design model is created and documented using architectural models; component models; object models and sequence models.
- Implementation
  - The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.

# The Rational Unified Process

- Testing
  - Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
- Deployment
  - A product release is created; distributed to users and installed in their workplace.
- Configuration and change management
  - This supporting workflow managed changes to the system (see Chapter 25).
- Project management
  - This supporting workflow manages the system development
- Environment
  - This workflow is concerned with making appropriate software tools available to the software development team.

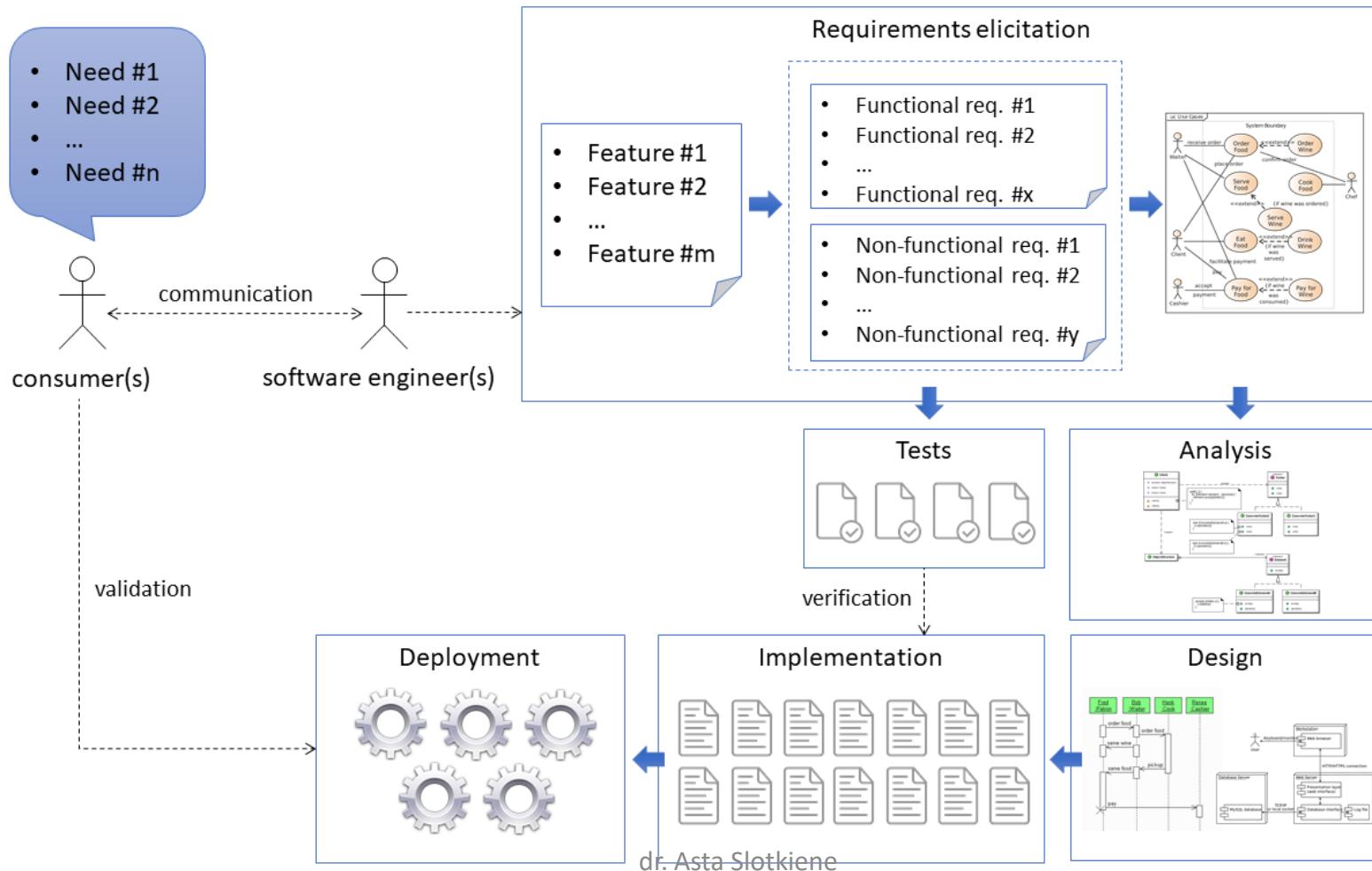
# Software process models

vs

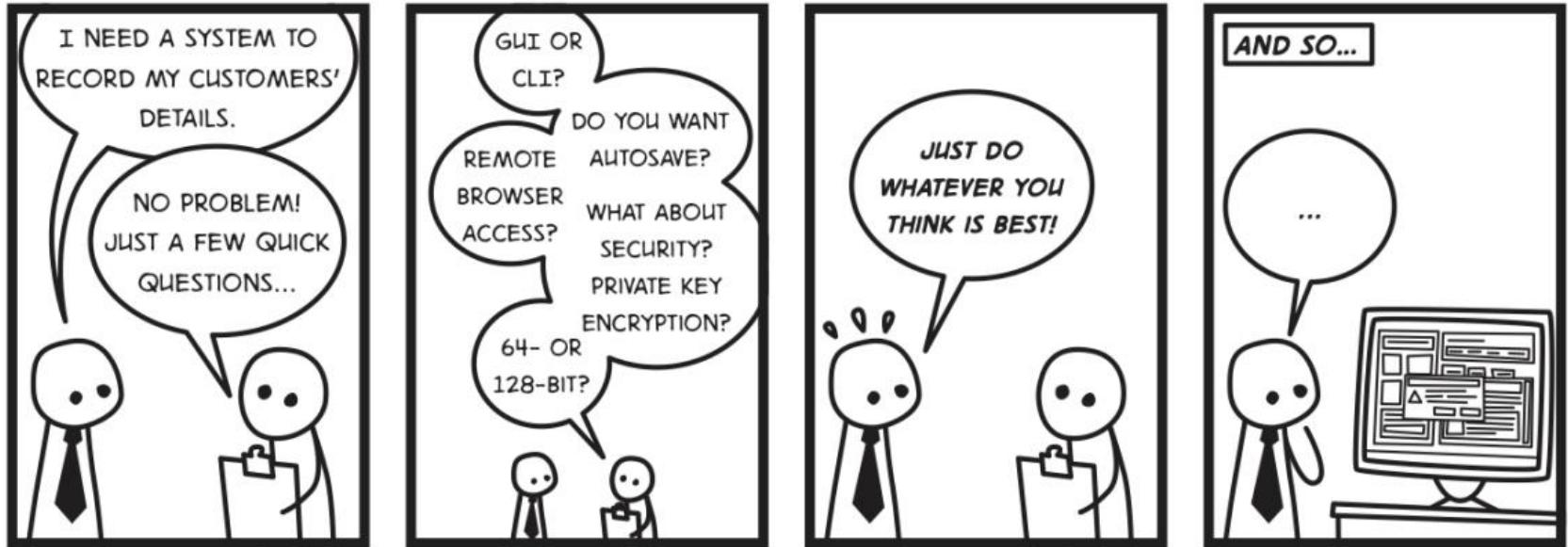
# Software life cycle models

- Software life cycle models **define the main steps** (phases or stages) in the software life cycle and **their sequence**
- **Software process models provide more detail, breaking** the main steps down into sub-steps, and adding information about the results generated and the roles involved

# Software engineering generic development process



# Why software engineering?



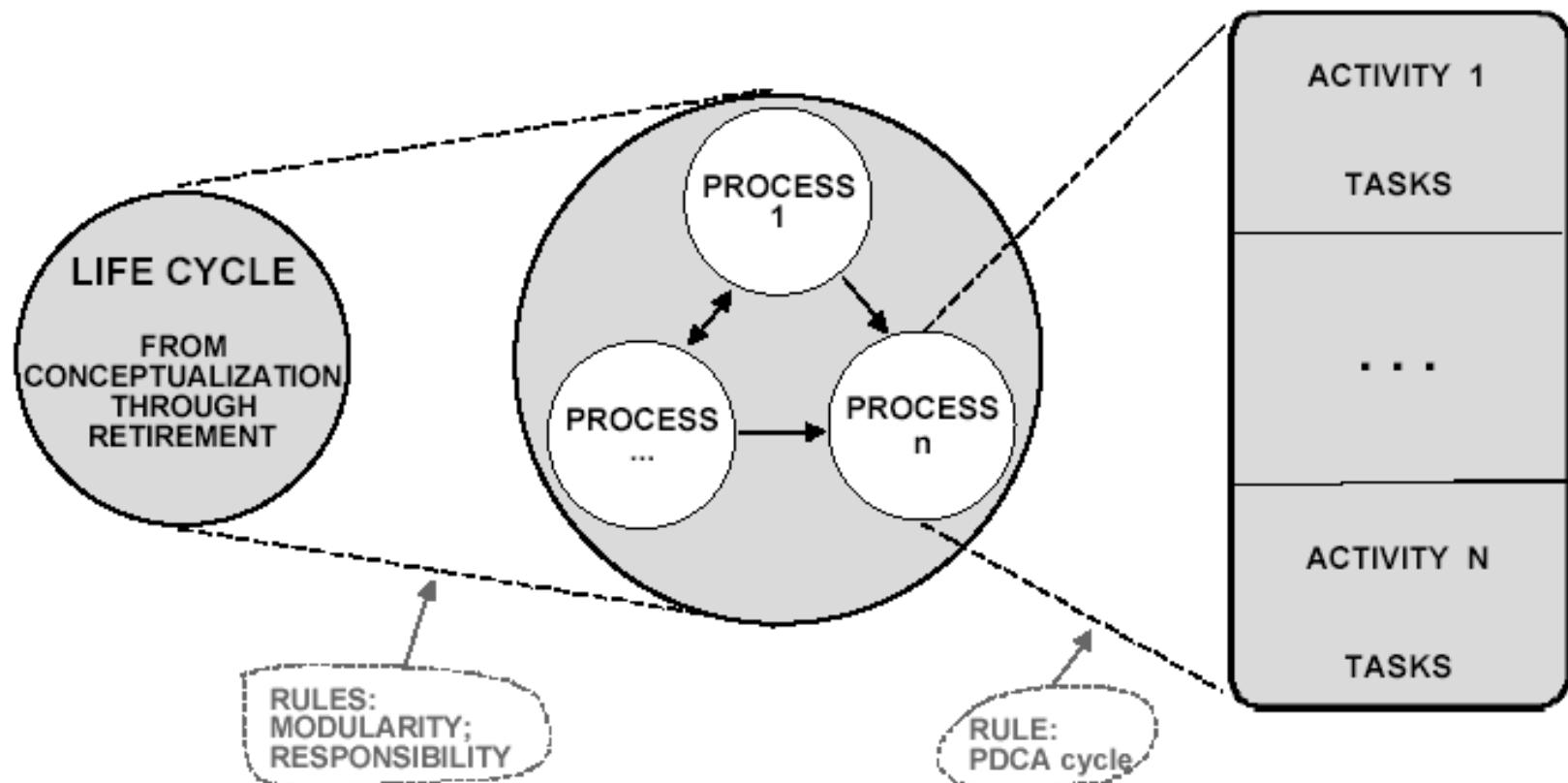
# *ISO/IEC 12207 – Systems and software engineering – Software life cycle processes*

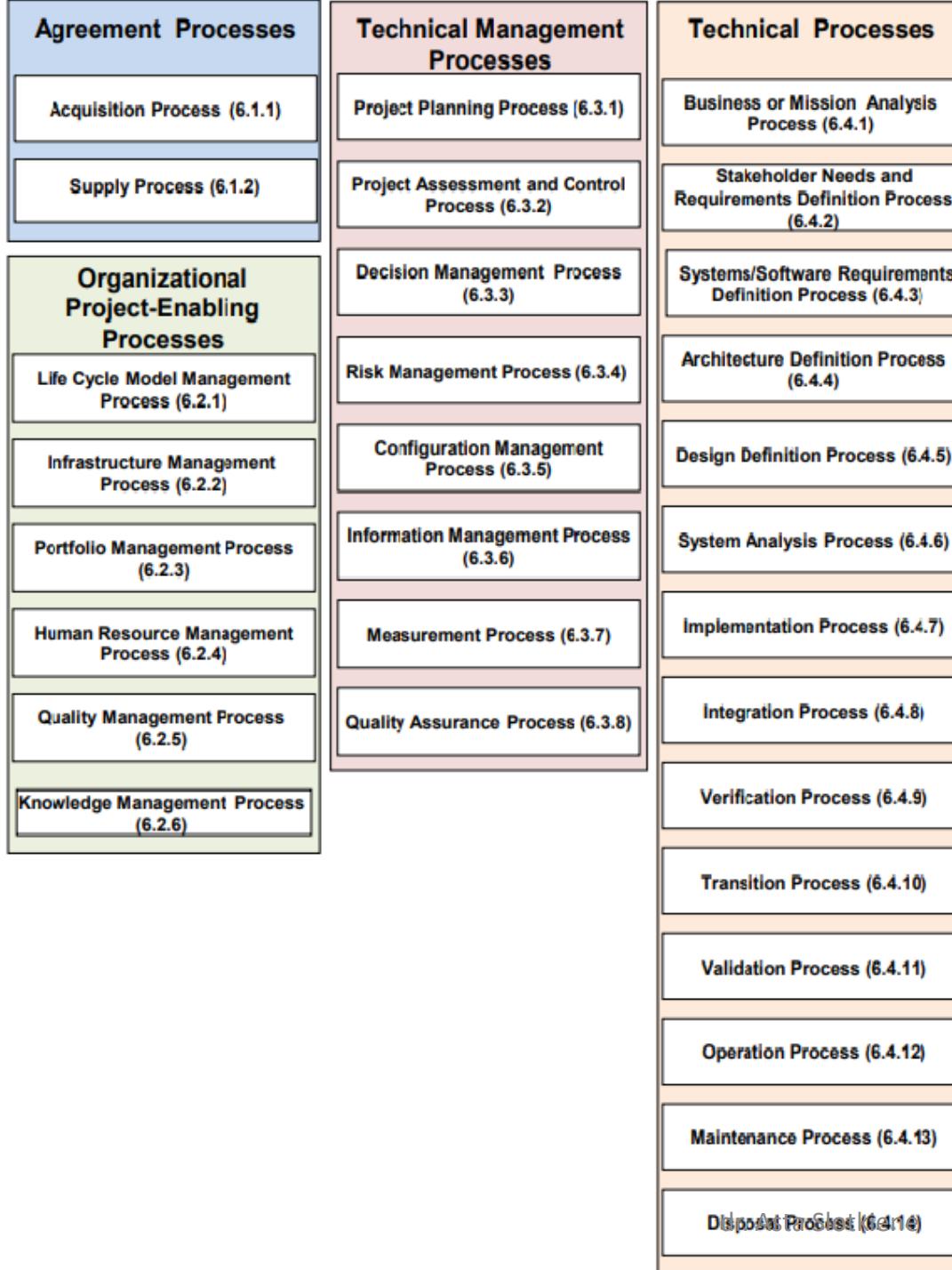
- **ISO/IEC/IEEE 12207 Systems and software engineering – Software life cycle processes<sup>[1]</sup> is an international standard for software lifecycle processes.**
- First introduced in 1995, it aims to be a **primary standard that defines all the processes required for developing and maintaining software systems**, including the outcomes and/or activities of each process.

# *ISO/IEC 12207 – Systems and software engineering – Software life cycle processes*

- ISO/IEC/IEEE 12207:2017 divides software life cycle processes into four main process groups:
  1. Agreement
    - address **the cooperation and agreement with other organisations**
  2. Organizational project-enabling processes
    - processes are performed on the level of the development organisation and **provide the environment needed to perform projects.**
  3. Technical management
    - **Project processes**
  4. Technical processes
    - describe the various processes or phases within a software product life cycle, **from stakeholder requirements definition to software disposal.**

# ISO/IEC 12207:2005 – Information Technology – Software life cycle processes





# ISO/IEC 12207 – *Systems and software engineering – Software life cycle processes*

# Technical processes

- to define the **requirements** for a software system,
- to transform the requirements into an **effective product**,
- to permit consistent **reproduction of the product** where necessary,
- to **dispose of the product when it is retired from service**

# Technical processes

**Business or Mission Analysis Process (6.4.1)**

**Stakeholder Needs and Requirements Definition Process (6.4.2)**

**Systems/Software Requirements Definition Process (6.4.3)**

**Architecture Definition Process (6.4.4)**

**Design Definition Process (6.4.5)**

**System Analysis Process (6.4.6)**

**Implementation Process (6.4.7)**

**Integration Process (6.4.8)**

**Verification Process (6.4.9)**

**Transition Process (6.4.10)**

**Validation Process (6.4.11)**

**Operation Process (6.4.12)**

**Maintenance Process (6.4.13)**

**Disposal Process (6.4.14)**

# Business or Mission Analysis process

- The purpose of the Business or Mission Analysis process is to define:
  - the business or mission problem or opportunity,
  - characterize the solution space,
  - determine potential solution class(es) that could address a problem or take advantage of an opportunity.

# Stakeholder Needs and Requirements

## Definition process

- The purpose of the Stakeholder Needs and Requirements Definition process is to define:
  - the stakeholder requirements for a system that can provide the capabilities needed by users and other stakeholders in a defined environment.

# System/Software requirements definition process

- The purpose of the System/Software Requirements Definition process is **to transform the stakeholder, user- oriented view of desired capabilities into a technical view of a solution that meets the operational needs of the user.**

# Architecture Definition process

- The purpose of the Architecture Definition process is
  - to generate system architecture alternatives,
  - to select one or more alternative(s) that frame stakeholder concerns and meet system requirements,
  - to express this in a set of consistent views.

# Design Definition process

- The purpose of the Design Definition process is:
  - to provide sufficient **detailed data and information** about the system and its elements
  - to enable the implementation consistent with **architectural entities** as defined in models and views of the system architecture.

# System Analysis process

- The purpose of the System Analysis process is **to provide a rigorous basis of data and information for technical understanding** to aid decision-making across the life cycle.

# Implementation process

- The purpose of the Implementation process is to realize a specified system element.
- This process **transforms requirements, architecture, and design, including interfaces, into actions that create a system element** according to the practices of the selected implementation technology, using appropriate technical specialties or disciplines.

# Integration process

- The purpose of the Integration process is to **synthesize a set of system elements into a realized system** that satisfies system/software requirements, architecture, and design.
- This process **assembles the implemented system elements**.

# Verification process

- The purpose of the Verification process is to provide **objective evidence that a system or system element fulfils its specified requirements and characteristics.**
- 
- This process **provides the necessary information to determine resolution of identified anomalies.**

# Verification process

- The Verification process identifies the anomalies:
  - errors, defects, or faults in any information item (e.g., system/software requirements or architecture description),
  - implemented system elements,
  - life cycle processes using appropriate methods, techniques, standards or rules.

# Transition process

- The purpose of the Transition process is to establish a capability for a system to provide services specified by stakeholder requirements in the operational environment.
  - This process **moves the system in an orderly, planned manner into the operational status, such that the system is functional, operable and compatible with other operational systems.**
  - It installs a verified system, together with relevant enabling systems, e.g., planning system, support system, operator training system, user training system, as defined in agreements.

# Validation process

- The purpose of the Validation process is **to provide objective evidence** that the system, when in use, fulfils its business or mission objectives and stakeholder requirements, achieving its intended use in its intended operational environment.
  - The objective of validating a system or system element is to acquire confidence in its ability to achieve its intended mission, or use, under specific operational conditions. Validation is ratified by stakeholders.
  - This process provides the necessary information so that identified anomalies can be resolved by the appropriate technical process where the anomaly was created.

# Operation process

- The purpose of the Operation process is to use the system to deliver its services.
  - In order to sustain services, **it identifies and analyzes operational anomalies** in relation to agreements, stakeholder requirements and organizational constraints.

# Maintenance process

- The purpose of the Maintenance process is to sustain the capability of the system to provide a service.
  - This process monitors the system's capability to deliver services, records incidents for analysis, takes corrective, adaptive, perfective and preventive actions and confirms restored capability.

# Disposal process

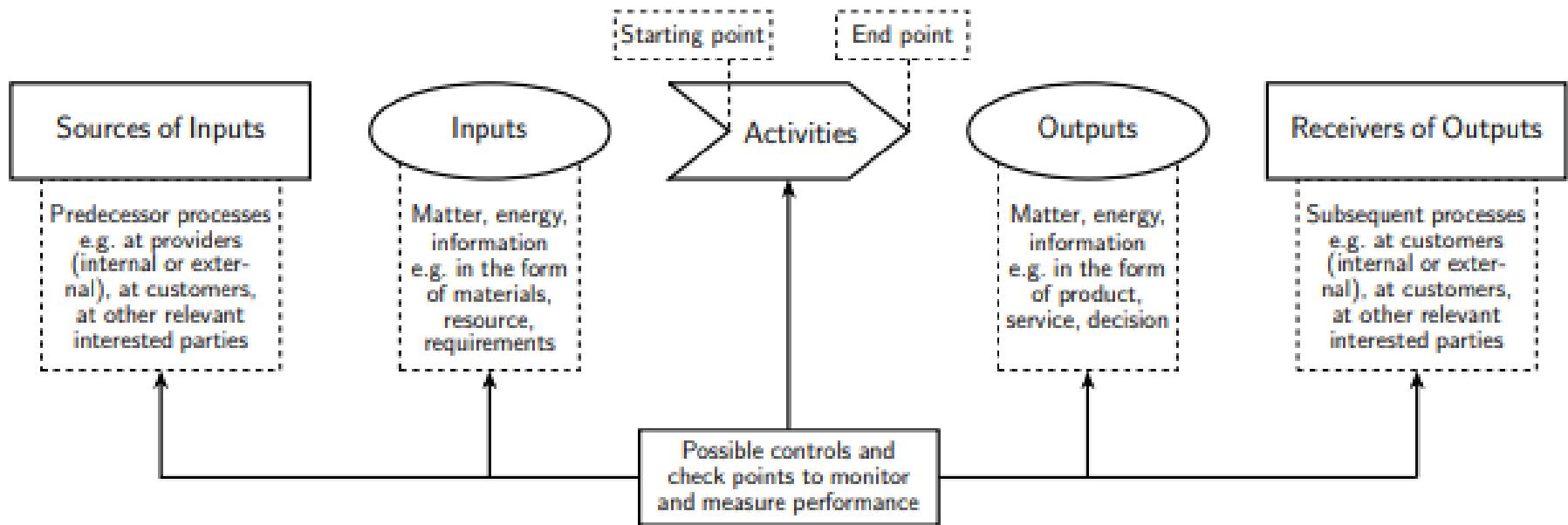
- The purpose of the Disposal process is to end the existence of a system element or system for a specified intended use, appropriately handle **replaced or retired elements**, and to properly attend **to identified critical disposal needs** (e.g., per an agreement, per organizational policy, or for environmental, legal, safety, security aspects).

# Representation of processes according SIPOC

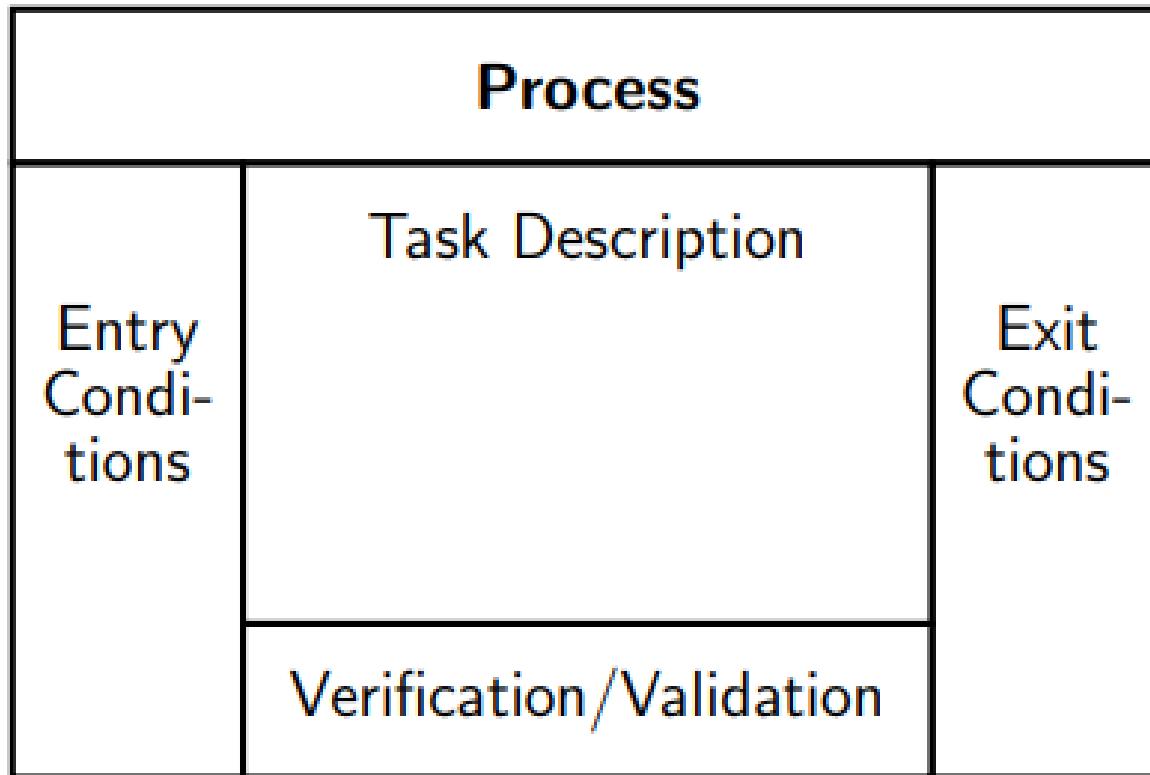
- SIPOC is a process notation that is commonly used in Six Sigma

Supplier	Input	Process	Output	Customer
Project stakeholders (end users etc.), product owner	Product backlog	Sprint planning, daily scrum, sprint review, sprint retrospective, development	Increment	Project customer, product owner

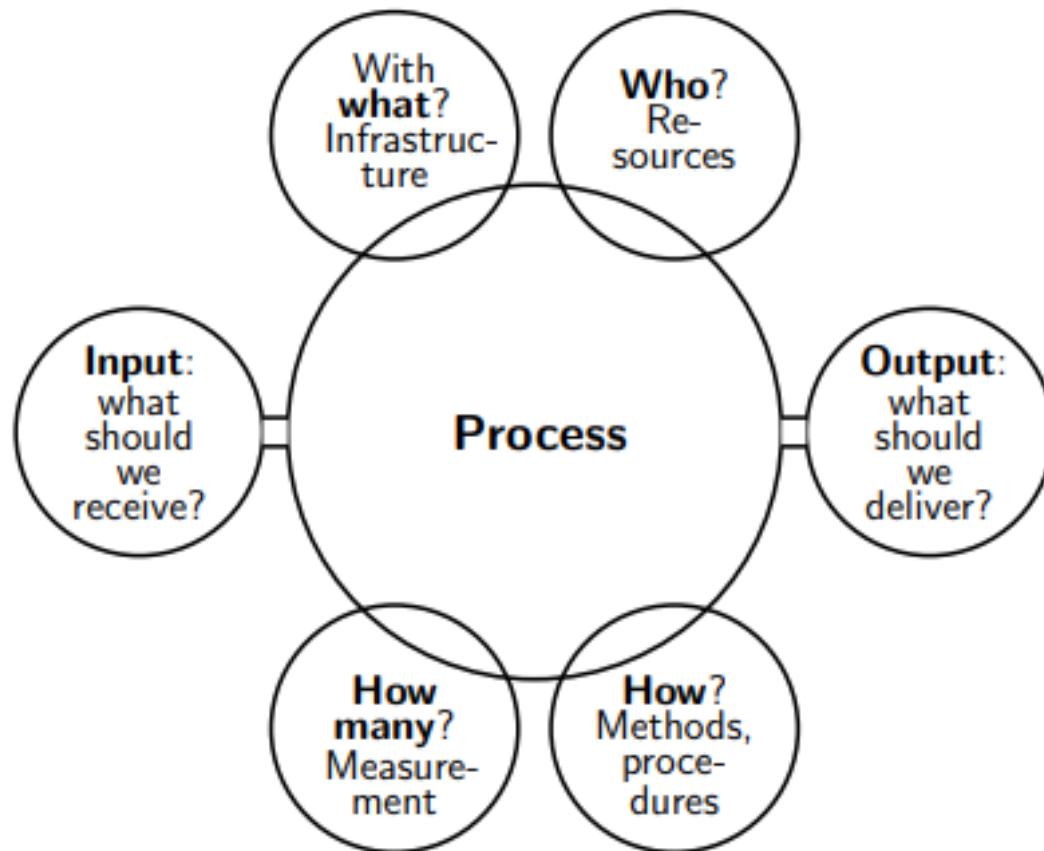
# Representation of processes according to ISO 9001



# Representation of processes according The Entry-Task-Verification-Exit (ETVX) notation



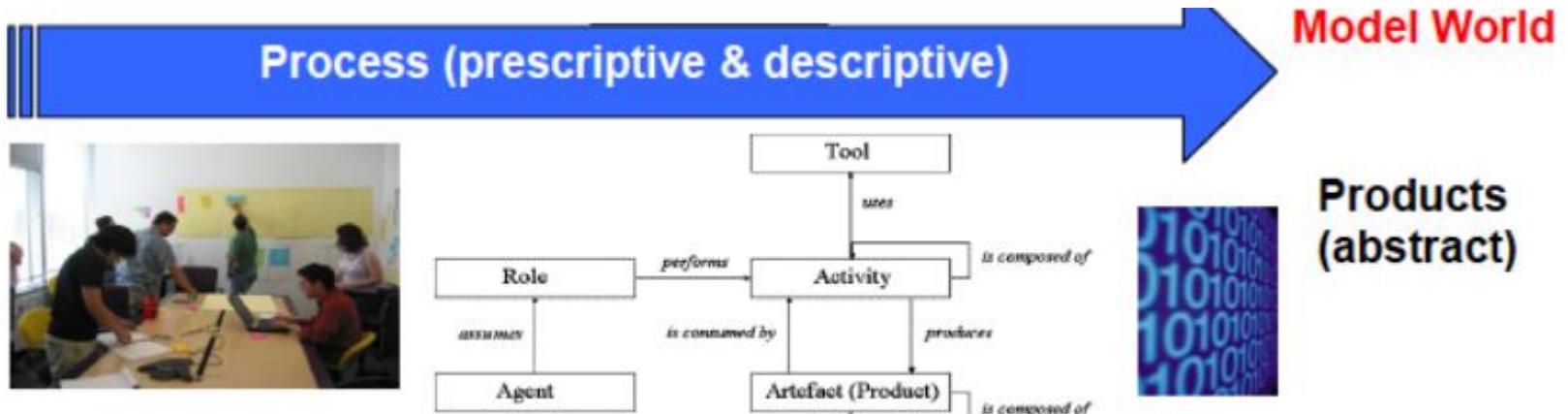
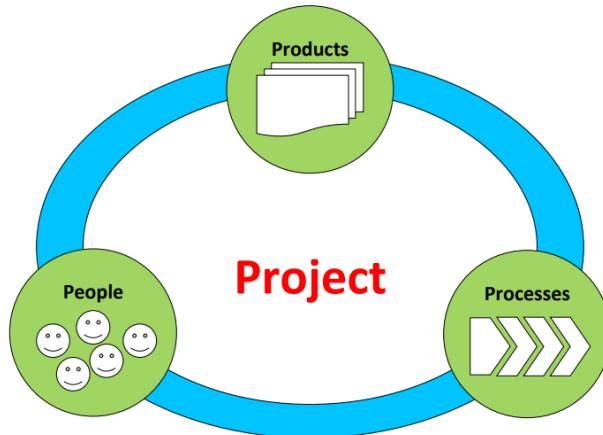
# Representation of processes according Turtle diagrams



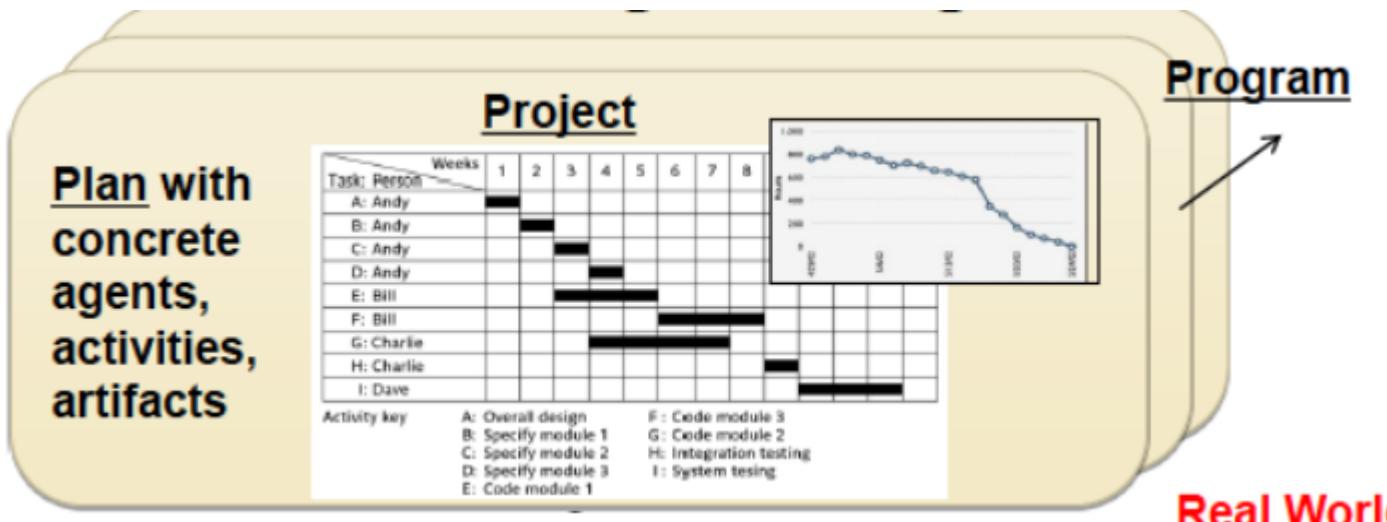
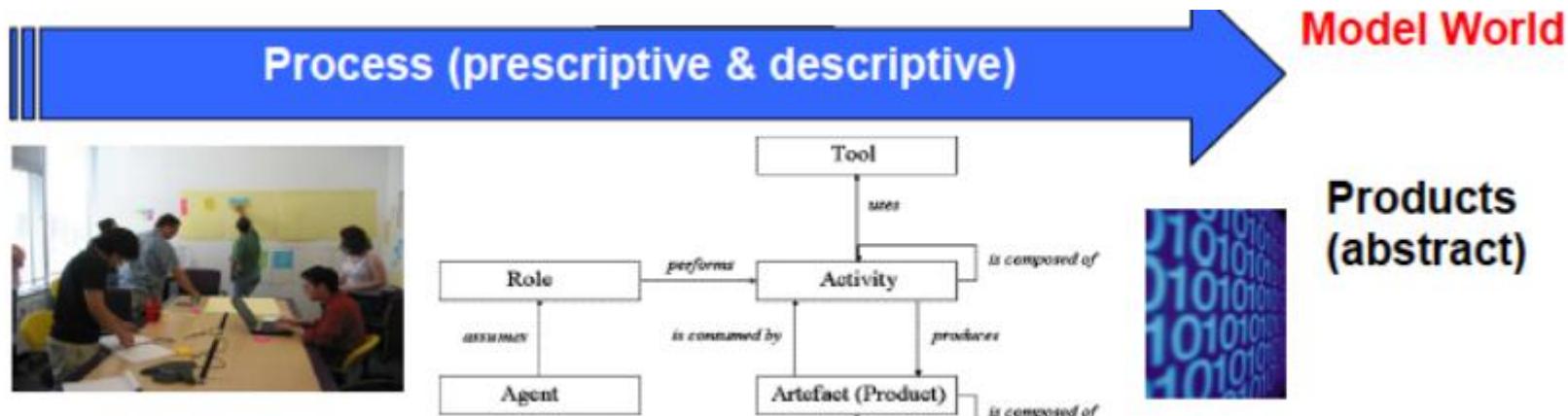
# Programų kūrimo procesas

Dr. Asta Slotkienė

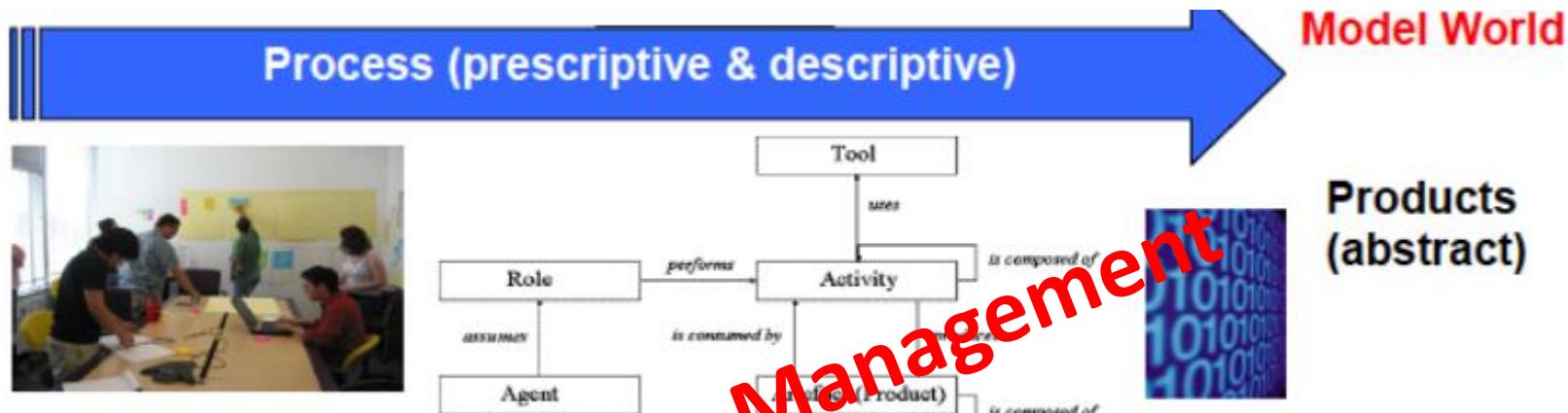
# The Three Ps in Software Engineering



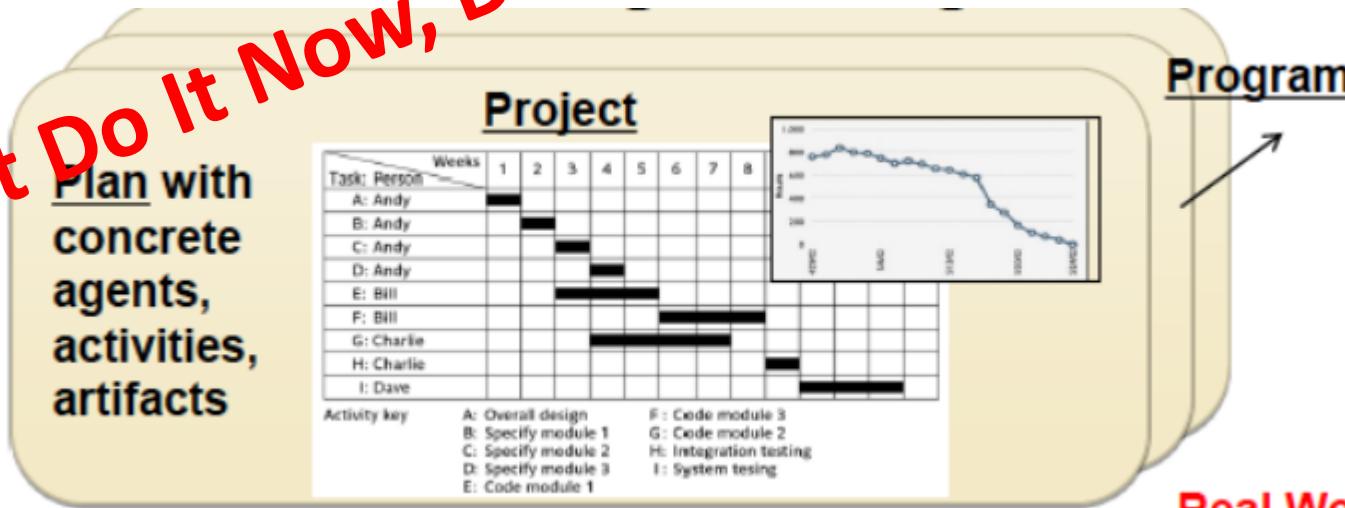
# The six Ps in Software Engineering



# The six Ps in Software Engineering

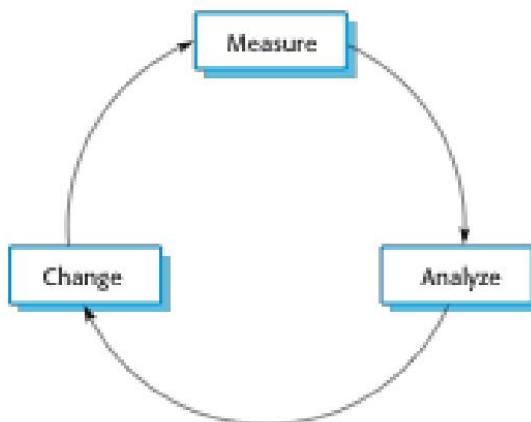


*Just Do It Now, Damage Management*



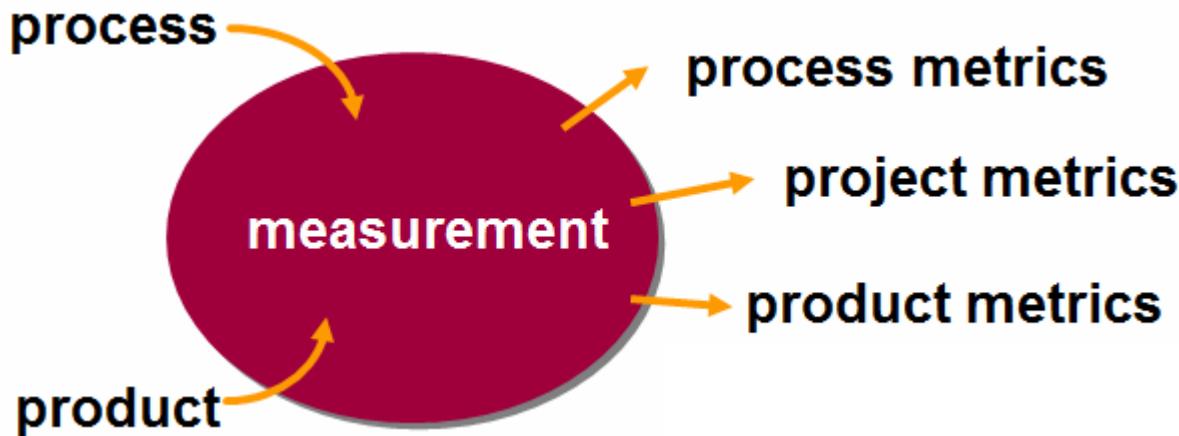
# Software Process Improvement

- **Software Process Improvement (SPI)** are actions taken to change the processes of an organization so that they achieve more effectively their business goals.
- Software process improvement models emphasize iterative cycles of continuous improvement.
- A **software process improvement cycle** typically involves the subprocesses of measuring, analyzing, and changing.  
[SWEBOK 3]



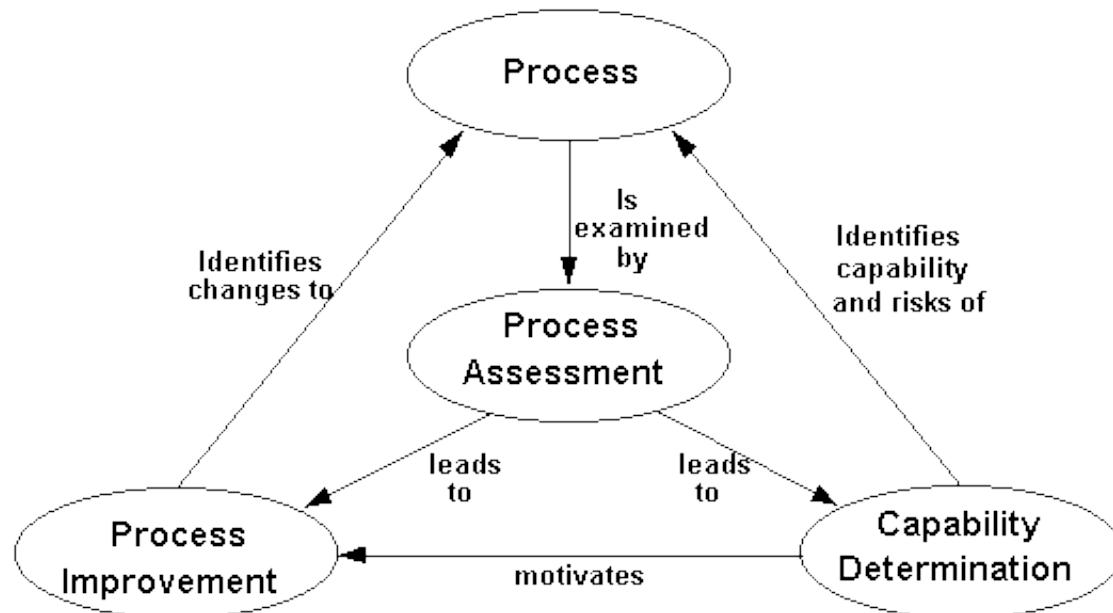
# Software Engineering Measurement

- The importance of measurement and its **role in better management and engineering practices**
- Effective measurement has become one of the cornerstones of organizational maturity.
- Measurement can be applied to **organizations, projects, processes, and work products**



# Software Process Improvement

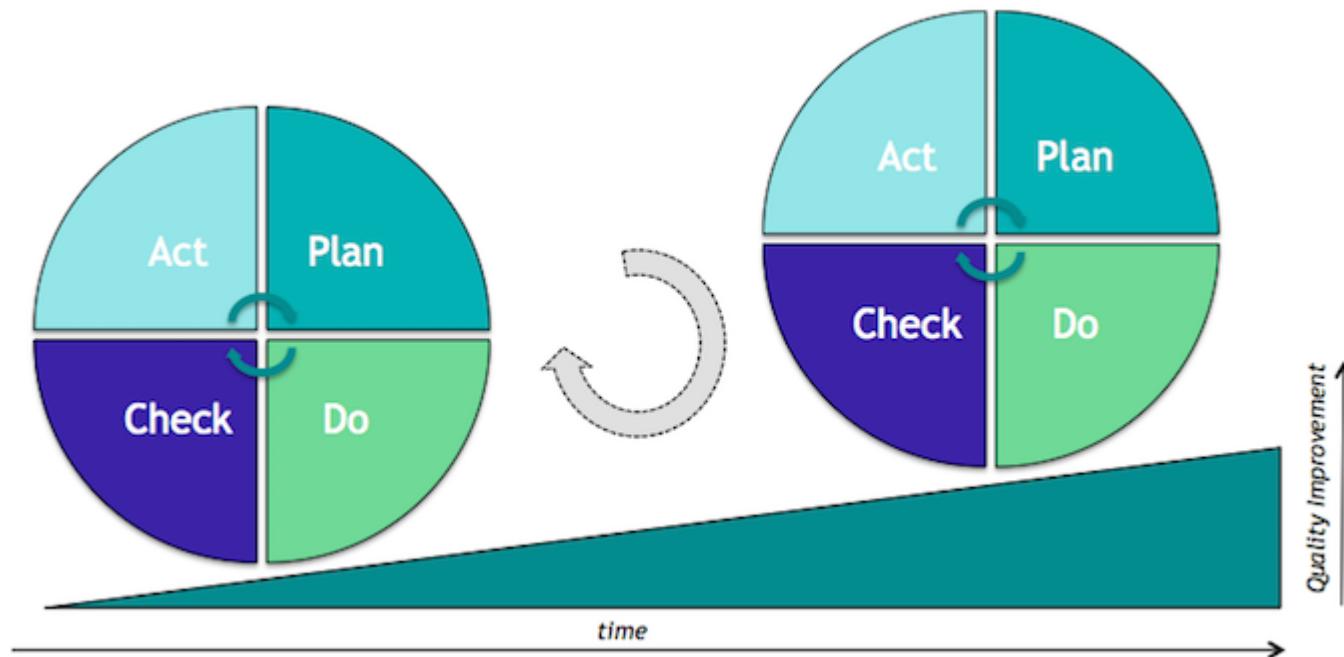
- Typically, software process and assessment are guided by:
  - a **maturity level**
  - a **process capability** profile based on capability/maturity model(s).



# PDCA

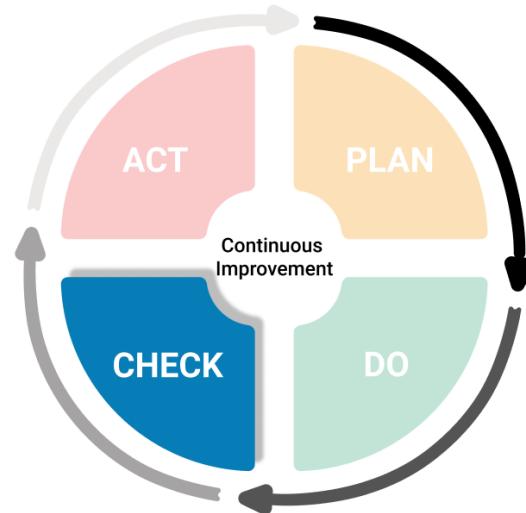
## Plan-Do-Check-Act

- ISO/IEC 27001



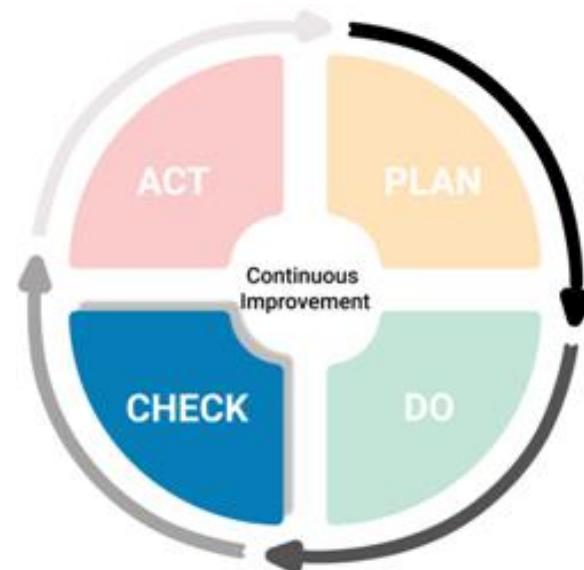
# Software process assessments

- Software process assessments are used to evaluate **the form and content of a software process**, which may be specified by a **standardized set of criteria**



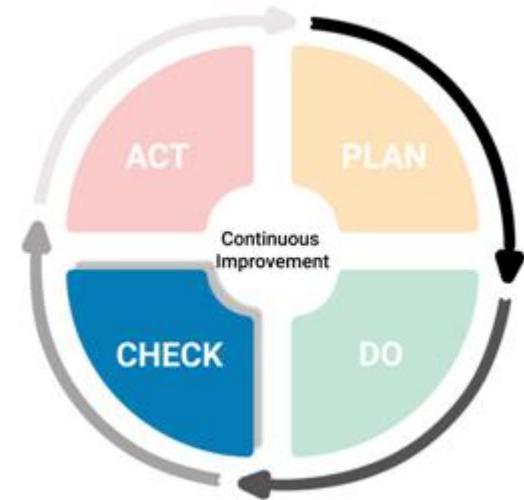
# Plan-Do-Check-Act

- Planing:
  - Improvement activities include **identifying and prioritizing desired improvements** ;
- Doing
  - introducing an improvement, including **change management and training**



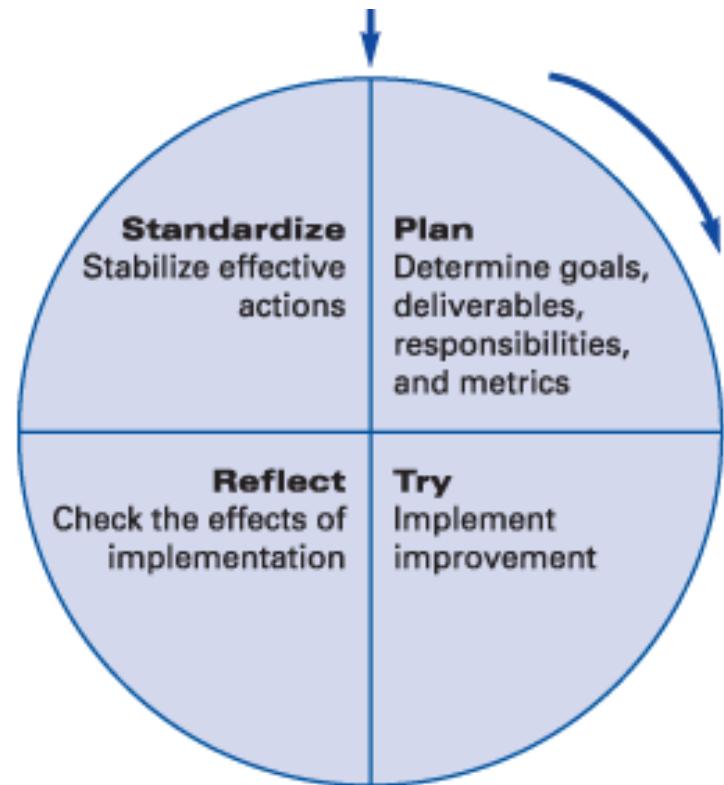
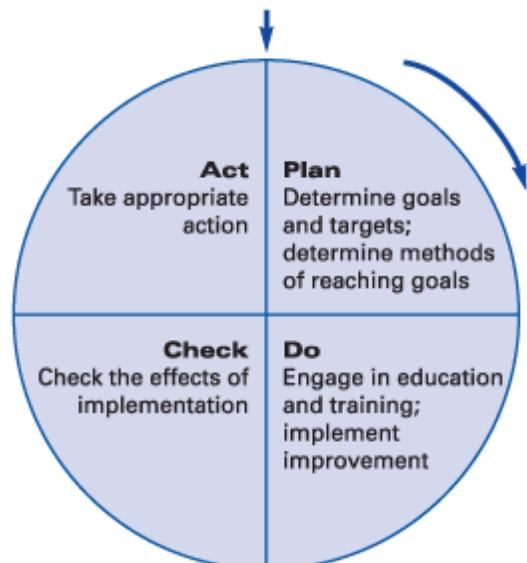
# Plan-Do-Check-Act

- Checking
  - **evaluating the improvement** as compared to previous or exemplary process results and costs
- Acting:
  - **making further modifications**



# Plan-Do-Check-Act

A Common Version of the PDCA Wheel



# Maturity

- Maturity levels apply to your organization's process improvement achievement in multiple process areas.
- These levels are a means of improving the processes corresponding to a given set of process areas (i.e., maturity level).
  - The five maturity levels are numbered 1 through 5.

*Branda – organizacijos charakteristika, nusakanti, kiek organizacijos procesas yra apibrėžtas, valdomas, matuojamas, kontroliuojamas ir nuolatos gerinamas*

Proceso branda yra gebėjimas sujungti žmogiškuosius išteklius, metodus, procedūras ir įrankius tam, kad būtų patenkinti kliento poreikiai (Poulin, 2006)

# Proceso brandos lygiai (CMMI)

- 0. Neegzistuojantis procesas
  - Nėra sistemišką veikimą pagrindžiančių įrodymų
- 1. Pradinis procesas (Initial)
  - Chaotiškas – neprognozuojamos kainos, terminai ir kokybė
- 2. Atkartojamas procesas (Repeatable)
  - Intuityvus – kaina ir kokybė smarkiai varijuoja, pakankamas terminų valdymas, neformalios procedūros
- 3. Apibrėžtas procesas (Defined)
  - Kokybinis – patikimai nustatomos kainos ir terminai, gerėjanti, bet neprognozuojama kokybė
- 4. Kiekybiškai valdomas procesas (Quantitatively Managed)
  - Kiekybinis – pakankama statistinė produktų kokybės kontrolė
- 5. Nuolatos gerinamas procesas (Optimizing)
  - Kiekybinės metrikos naudojamos nuolatiniam proceso automatizavimo plėtimui ir proceso gerinimui

# Maturity

- Maturity levels represent a staged path for an organization's performance and process improvement efforts based on predefined sets of practice areas.
- Maturity:
  - Processes are documented
  - Assigned responsibilities
  - Processes are managed
  - Processes are structured
  - Customer-centric processes
- Each maturity level builds on the previous maturity levels by adding new functionality or rigor.

# Maturity

- Maturity levels represent a staged path for an organization's performance and process improvement efforts based on predefined sets of practice areas.
- Maturity:
  - Processes are documented
  - Assigned responsibilities
  - Processes are managed
  - Processes are structured
  - Customer-centric processes
- Each maturity level builds on the previous maturity levels by adding new functionality or rigor.

# Capability

- Capability levels apply to your organization's process **improvement achievement in individual process areas.**
  - The four capability levels are numbered 0 through 3.

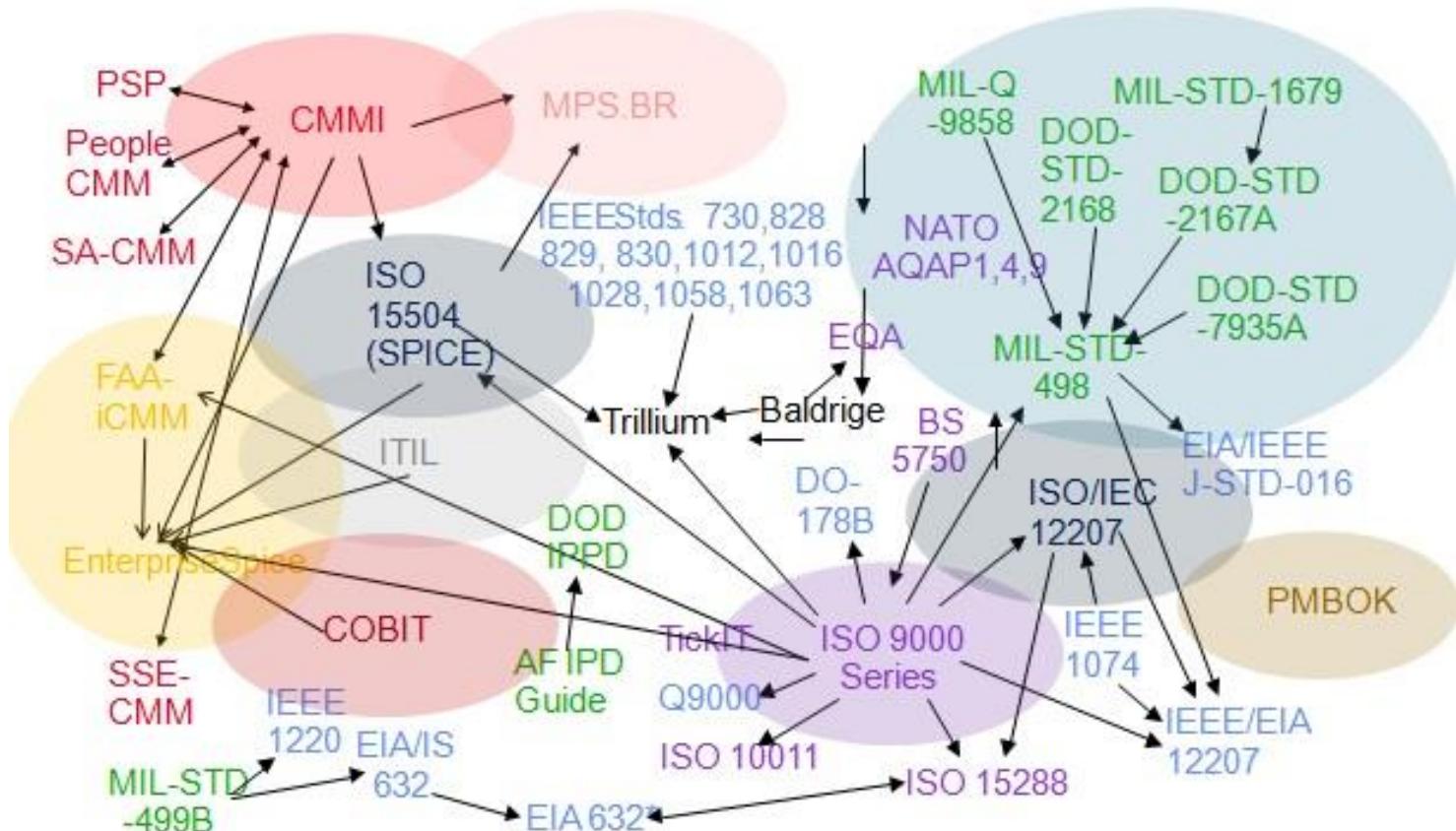
Proceso gebėjimas yra **skirtumo tarp užsibrėžtų tikslų ir realių rezultatų kontrolė**, gebėjimas nustatyti, ar bus pasiekti užsibrėžti tikslai ir užtikrintas efektyvumas (Lockamy, McCormack, 2004)

Gebėjimas – proceso charakteristika, nusakanti laukiamų rezultatų, kuriuos galima gauti taikant tą procesą, pasiskirstymą

# Capability

- **Capability:**
  - Team have the necessary competencies
  - Responsibilities are assigned to the process
  - The results of the process are controlled
  - The activities of the organization are effective

# Capability/Maturity models

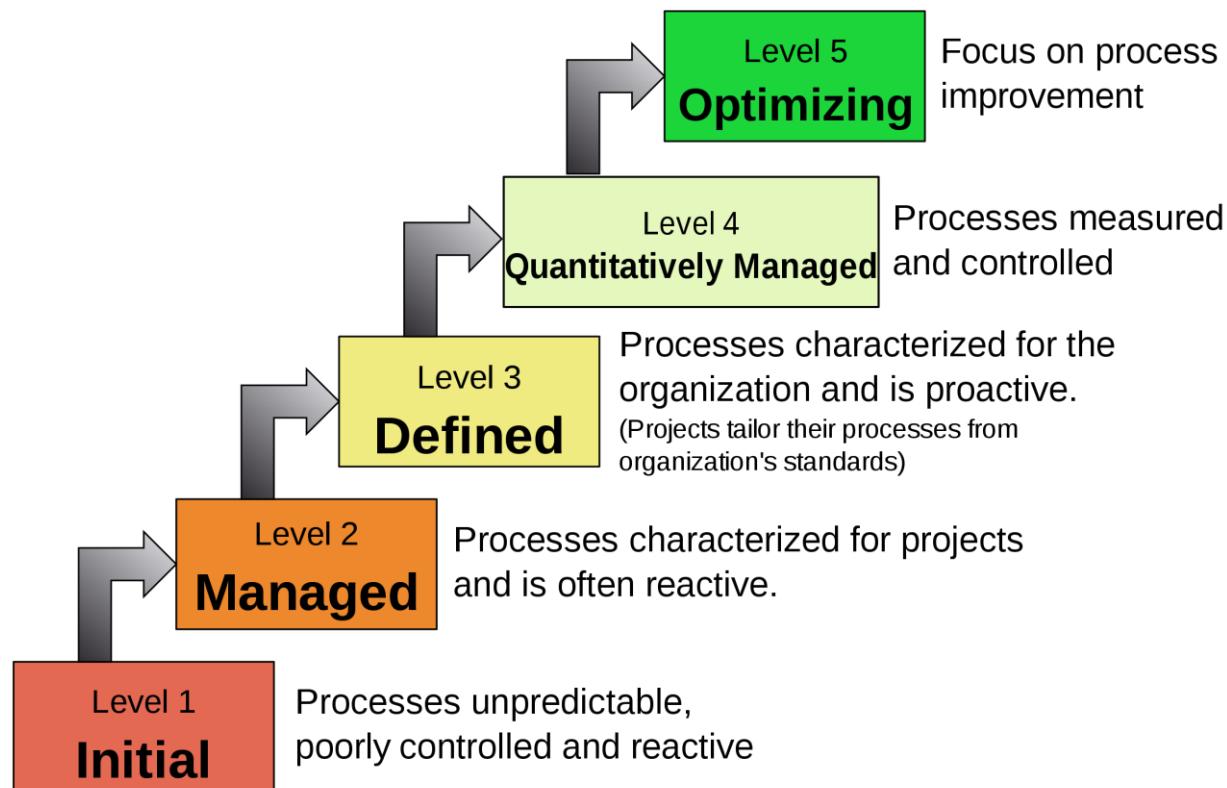


Adapted from [S. A. Sheard, Evolution of the Frameworks Quagmire. IEEE Computer, July 2001]

# Maturity models

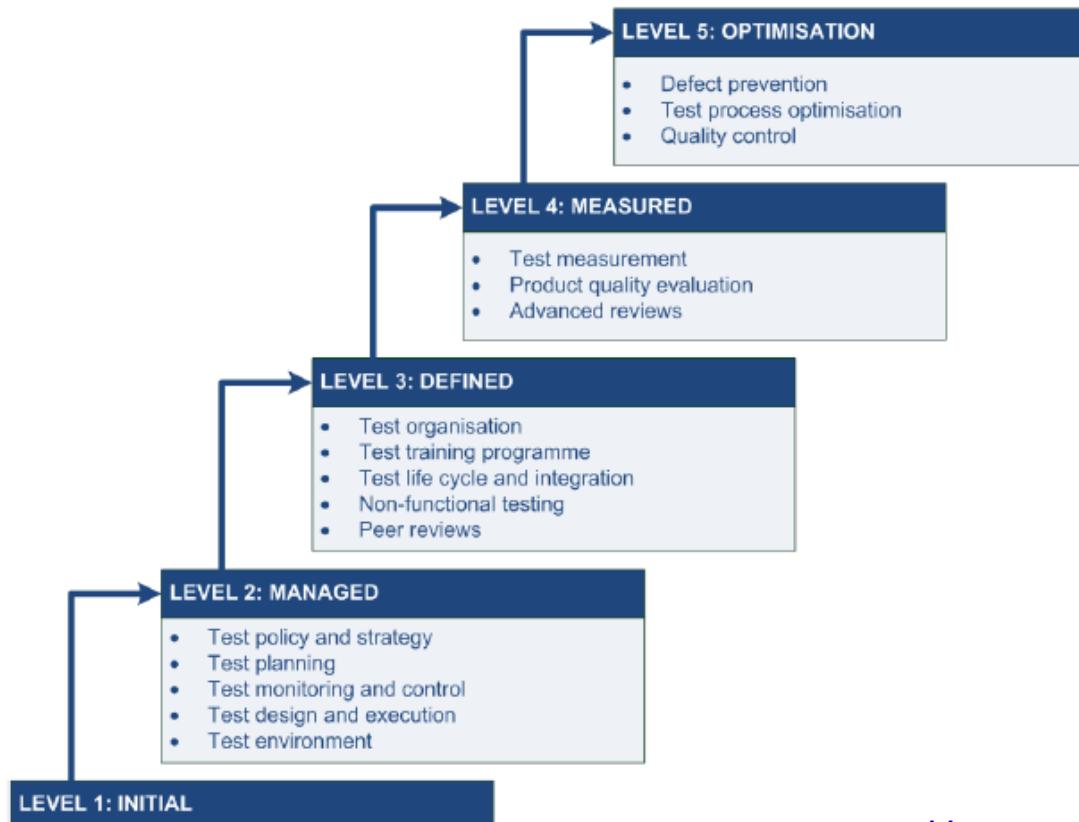
- CMM - Capability Maturity Model

## Characteristics of the Maturity levels



# Maturity models

- TMM Test Maturity Model



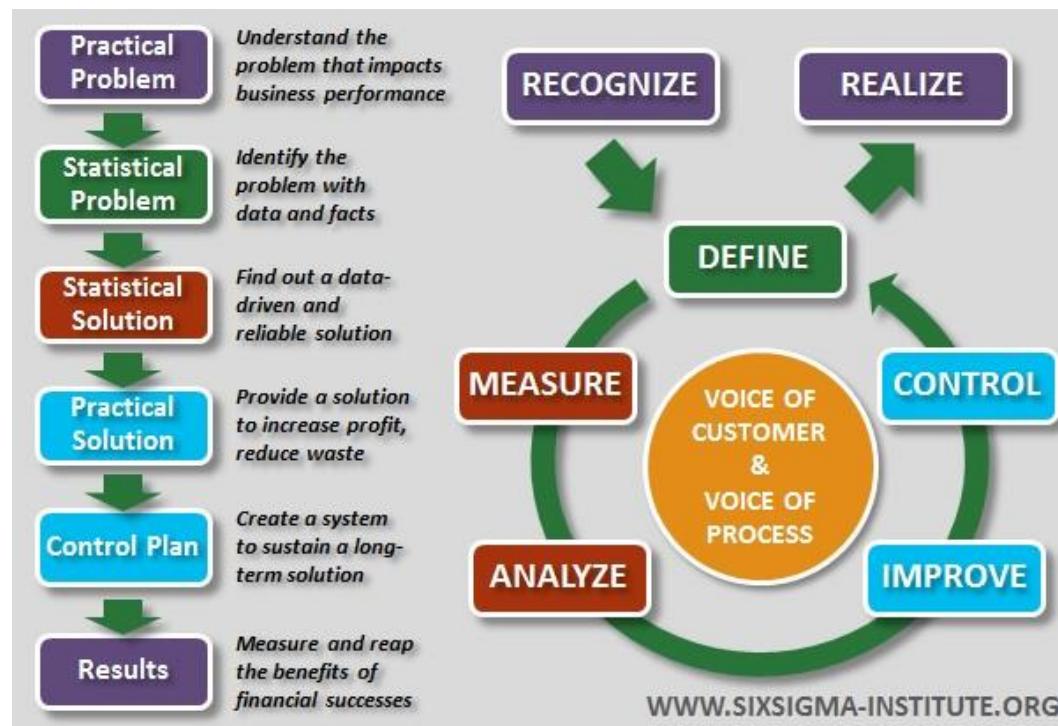
<https://www.tmmi.org/tmmi-model/>

Figure 1 - TMMi model.

# Maturity models

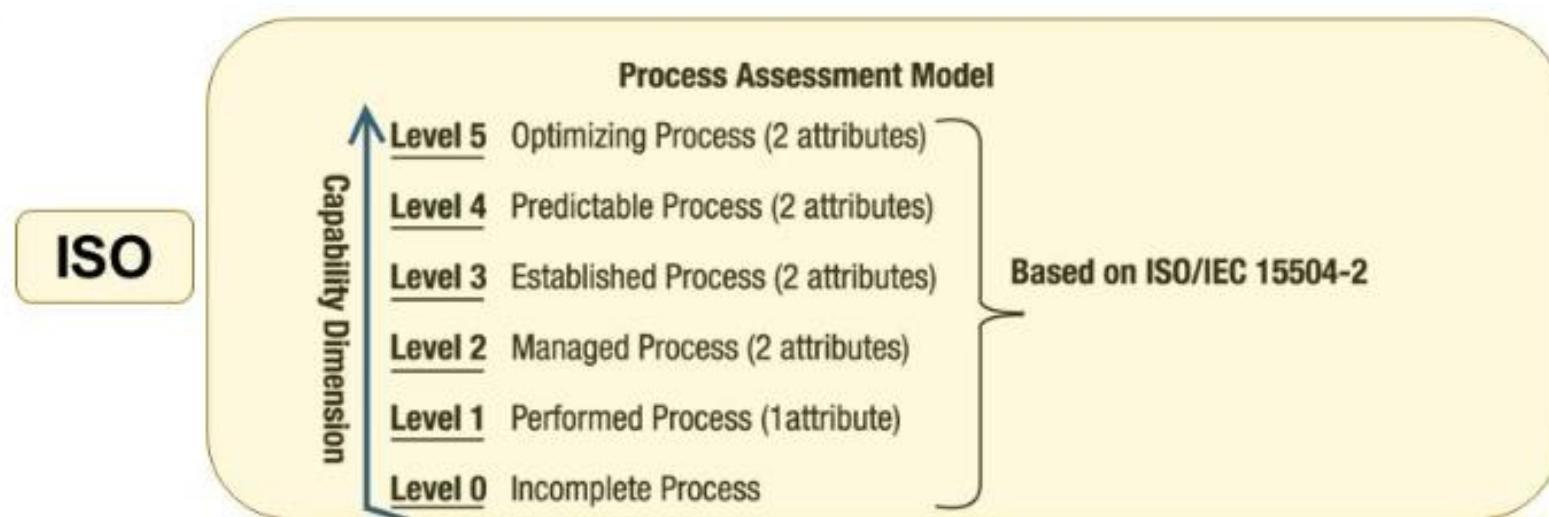
- SixSigma

- Six Sigma is a **quality program** that, when all is said and done, **improves your customer's experience, lowers your costs, and builds better leaders.** — Jack Welch*



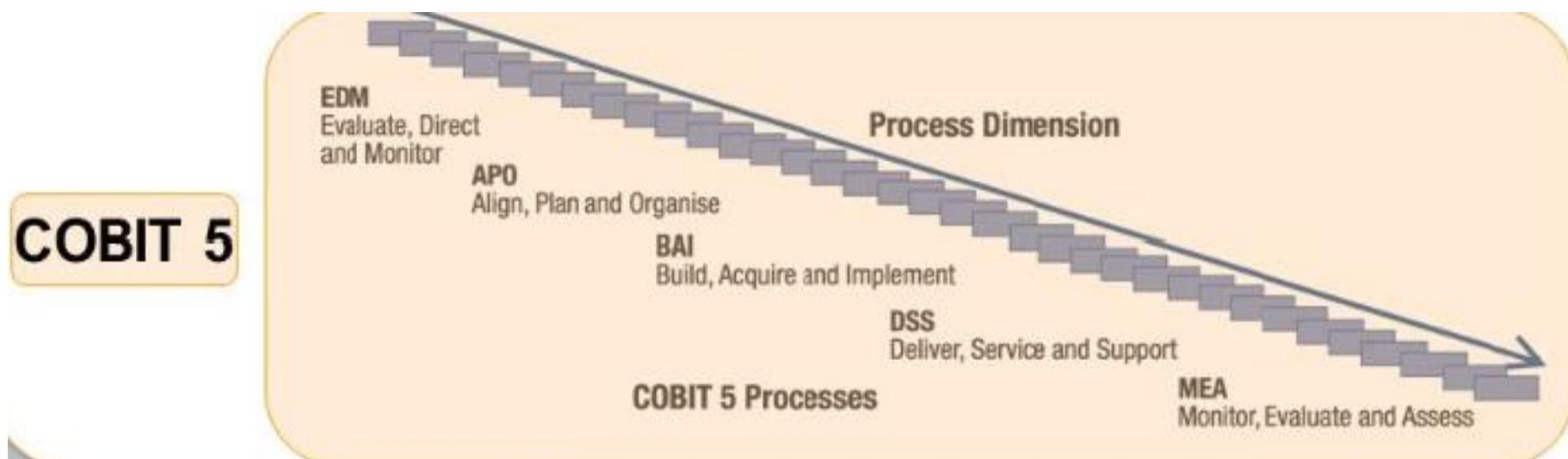
# Maturity models

- COBIT (Control Objectives for Information and Related Technologies) is a framework created by ISACA for information technology (IT) management and IT governance.



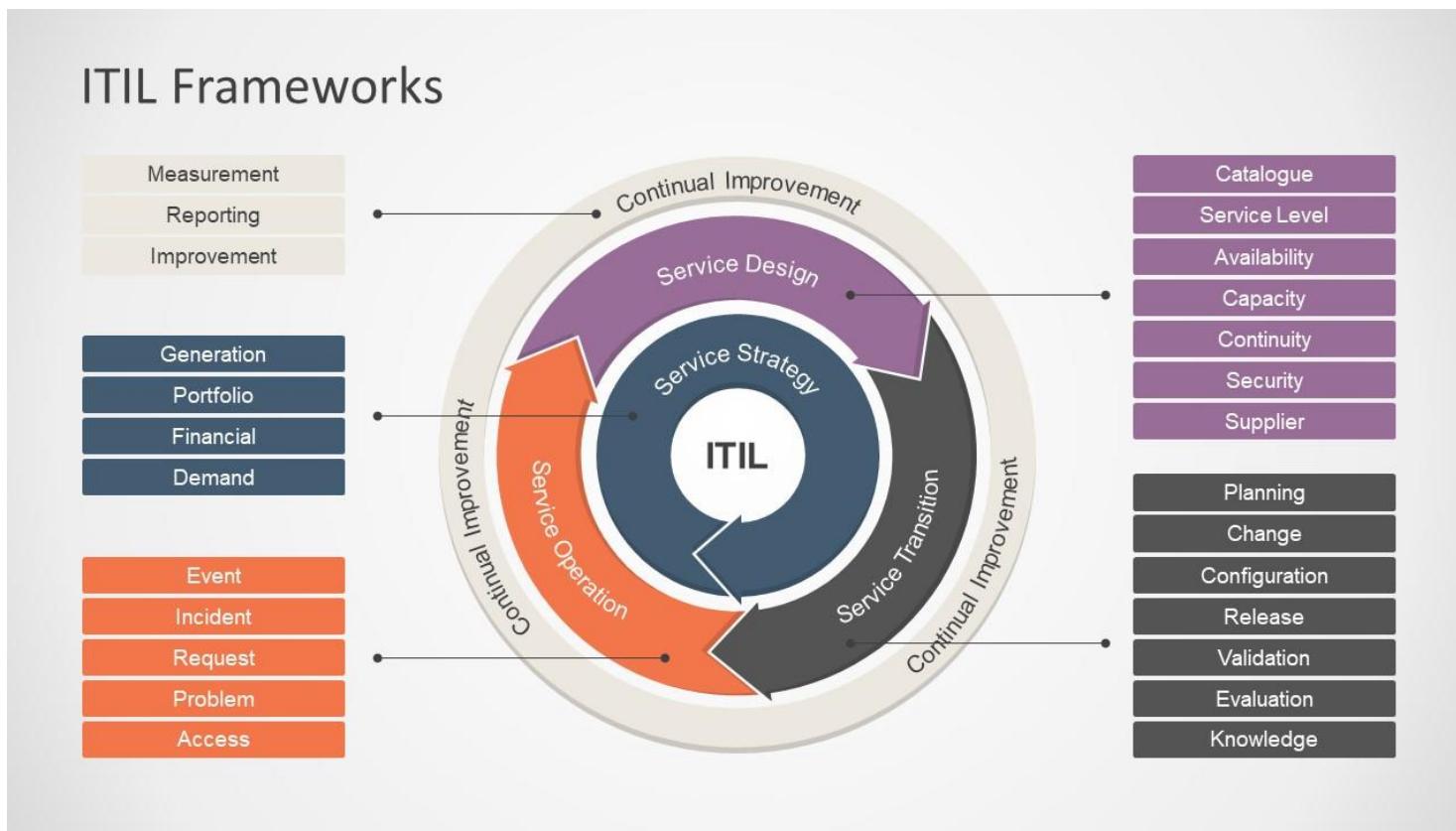
# Maturity models

- The framework defines a set of generic processes **for the management of IT**, with each process defined together with *process inputs and outputs, key process-activities, process objectives, performance measures and an elementary maturity model.*



# Maturity models

- **ITIL Information Technology Infrastructure Library** - Maturity model and self-assessment service has been developed to help organizations **improve their IT service management** within the ITIL framework.



# Maturity models

- ITIL describes processes, procedures, tasks, and checklists which are not organization-specific nor technology-specific, but can be applied by an organization toward strategy, delivering value, and maintaining a minimum level of competency.
- It allows the organization to establish a baseline from which it **can plan, implement, and measure.**

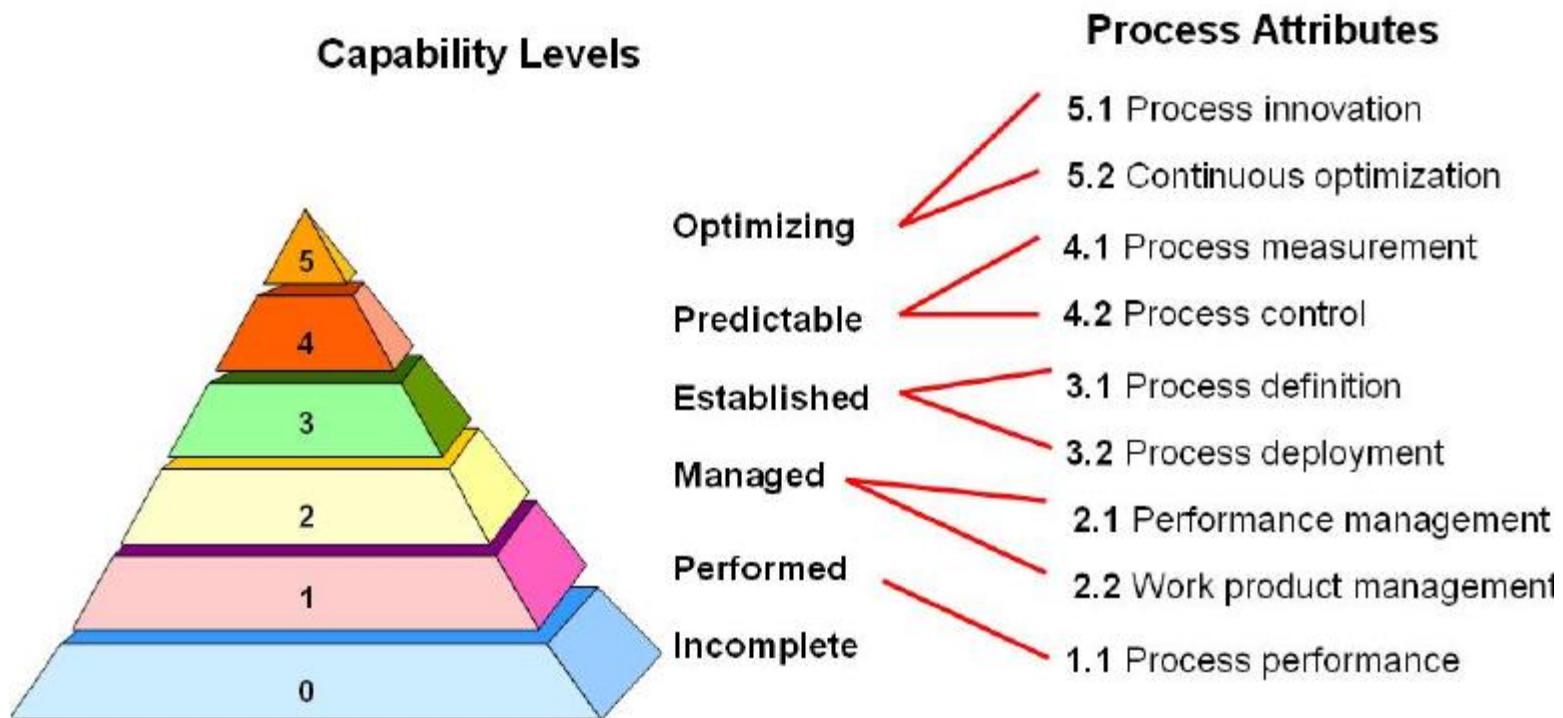


# Maturity models

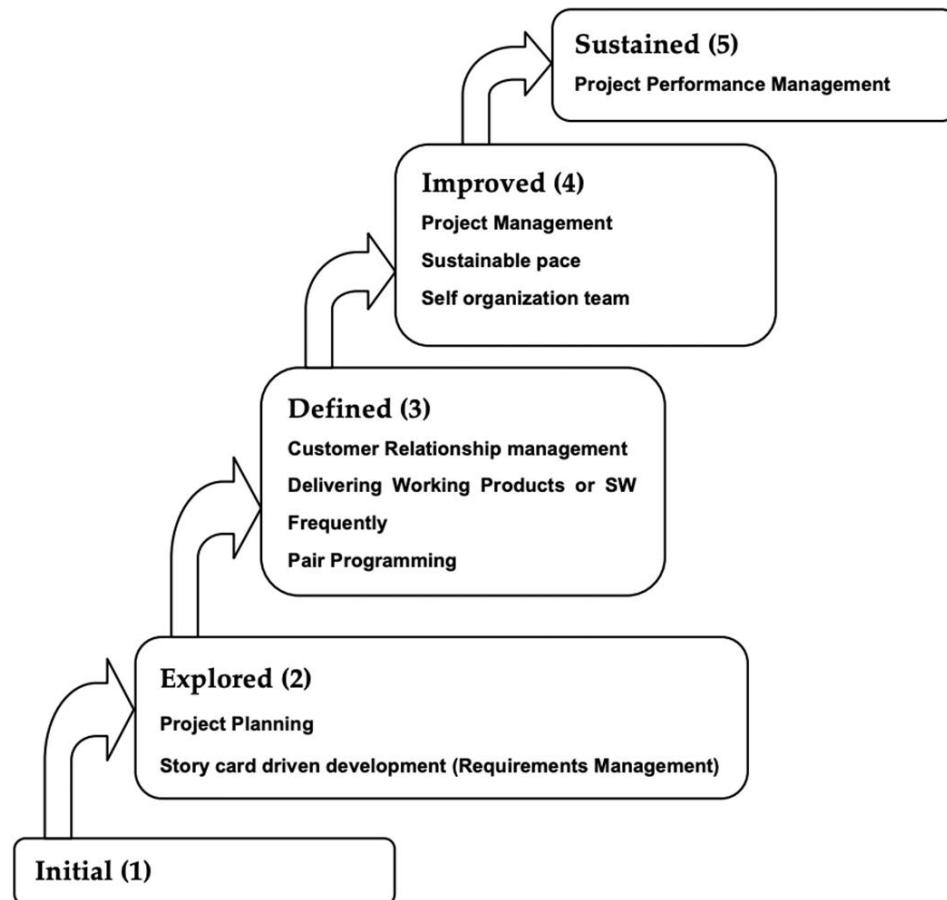
- **ISO/IEC 15504** *Information technology – Process assessment*, also termed **Software Process Improvement and Capability Determination (SPICE)**,
- Is a set of technical standards documents for the **computer software development process and related business management functions**.
- *It is one of the joint International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) standards, which was developed by the ISO and IEC joint subcommittee, ISO/IEC JTC 1/SC 7.*

# Maturity models

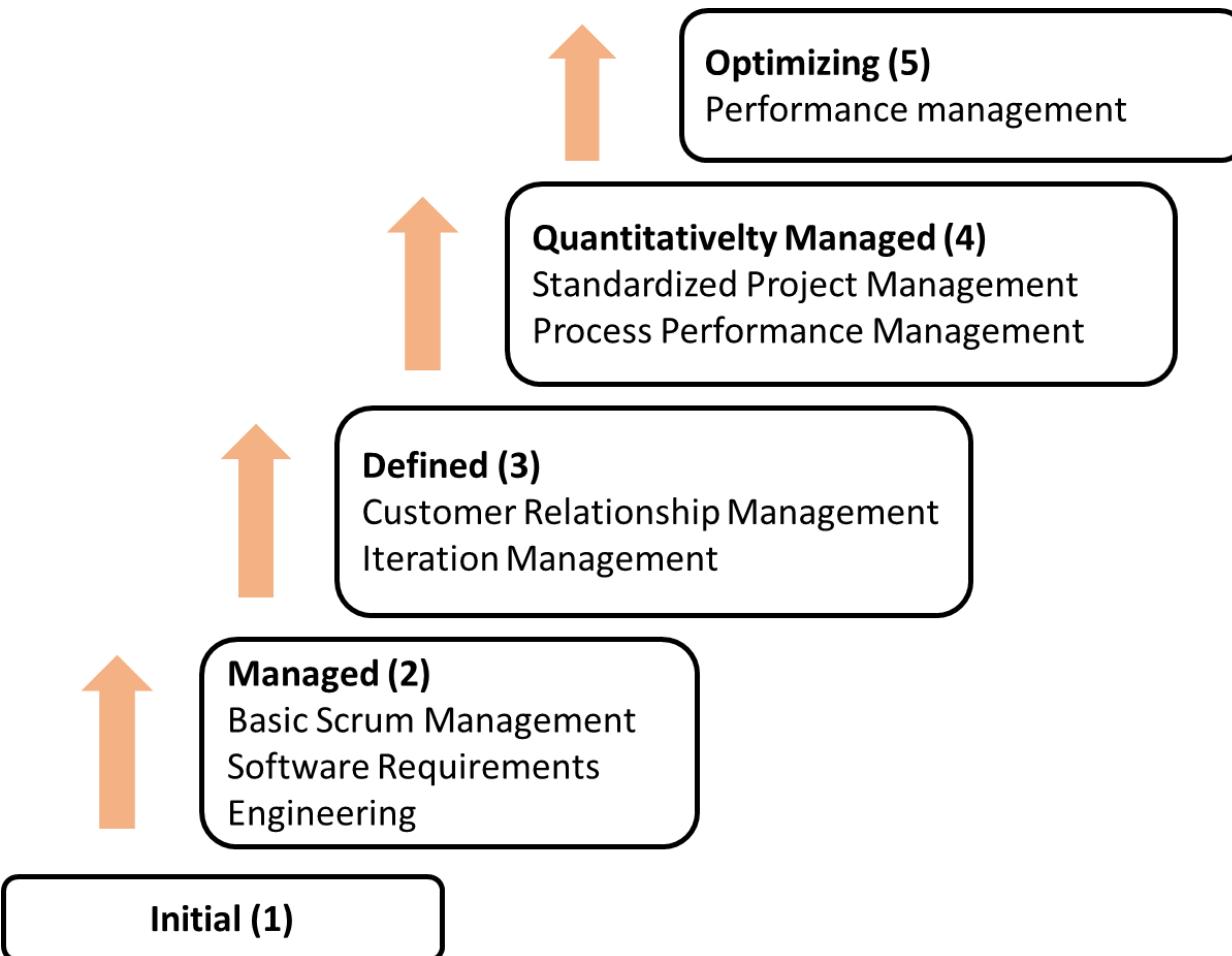
- ISO/IEC 15504



# Agile Maturity Model (Pathel and Ramachandran, 2009)



# Scrum Maturity Model (Yin, 2011)



# Capability Maturity Model - CMM

- CMMI (angl. Capability Maturity Model Integration).
- Department of Defense decided in the 80s to do something about the **many problems in its expensive software projects**
- CMMI models have expanded beyond software engineering to **help any organization in any industry build, improve, and measure their capabilities and improve performance.**
- CMMI is a collection of **BEST PRACTICES**.

**The CMMI describes WHAT to do,  
not HOW to do it.**

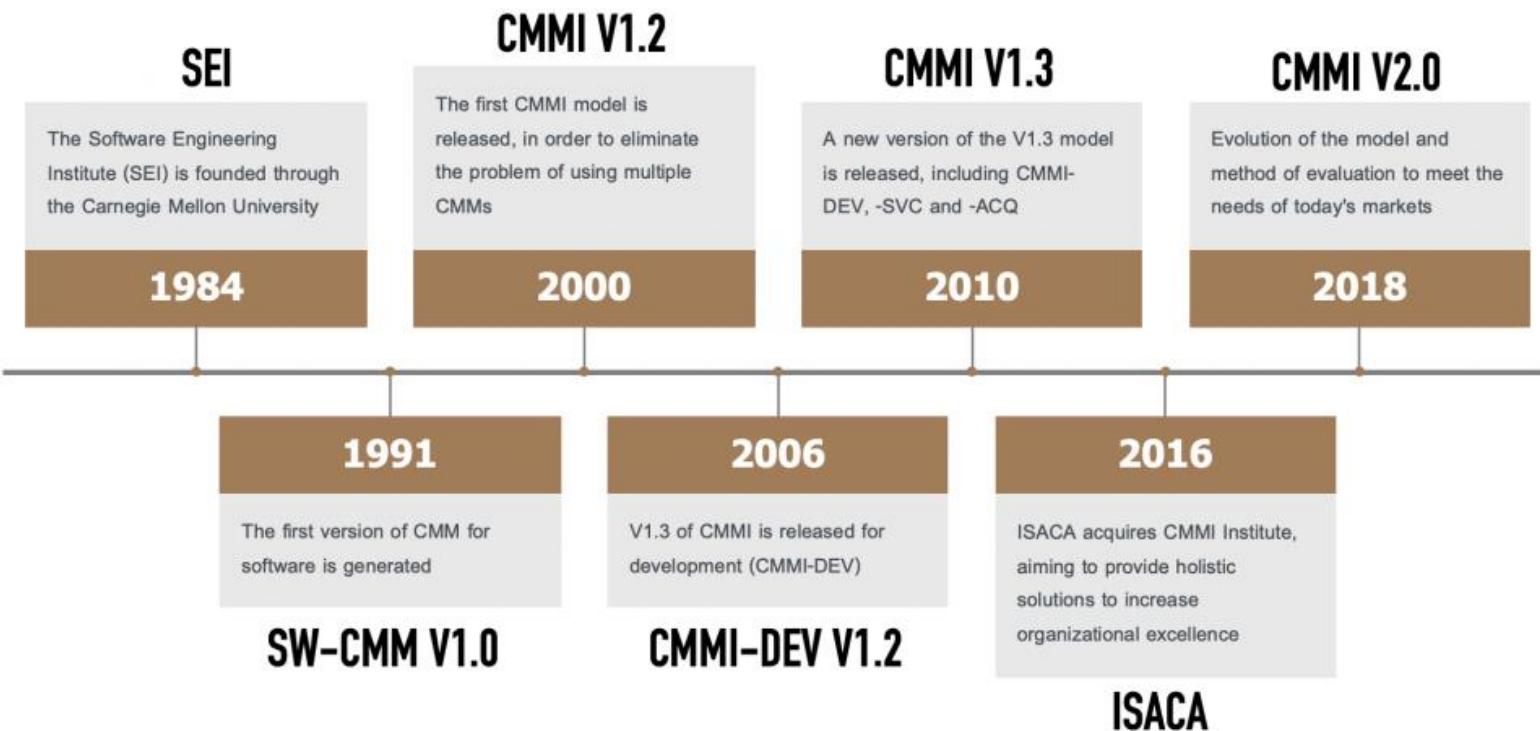
# What is a CMM?

- Capability Maturity Model:
  - A reference model of **mature practices** in a specified discipline, used to assess a **group's capability to perform that discipline**
- NOT:
  - It is not a ready-made scheme or template for describing processes
  - It contains no methods for the processes
- *The CMMI model is very complex, with more than 1,000 pages of description.*

# The CMM Explosion

- The first CMM (CMM v1.0) was developed for software and released in August 1991
- **Based on this success and the demand from other interests CMMs were developed for other disciplines and functions:**
  - Systems Engineering
  - People
  - Integrated Product Development
  - Software Acquisition
  - Software Quality Assurance
  - Measurement
  - Others.....

# History of CMMI



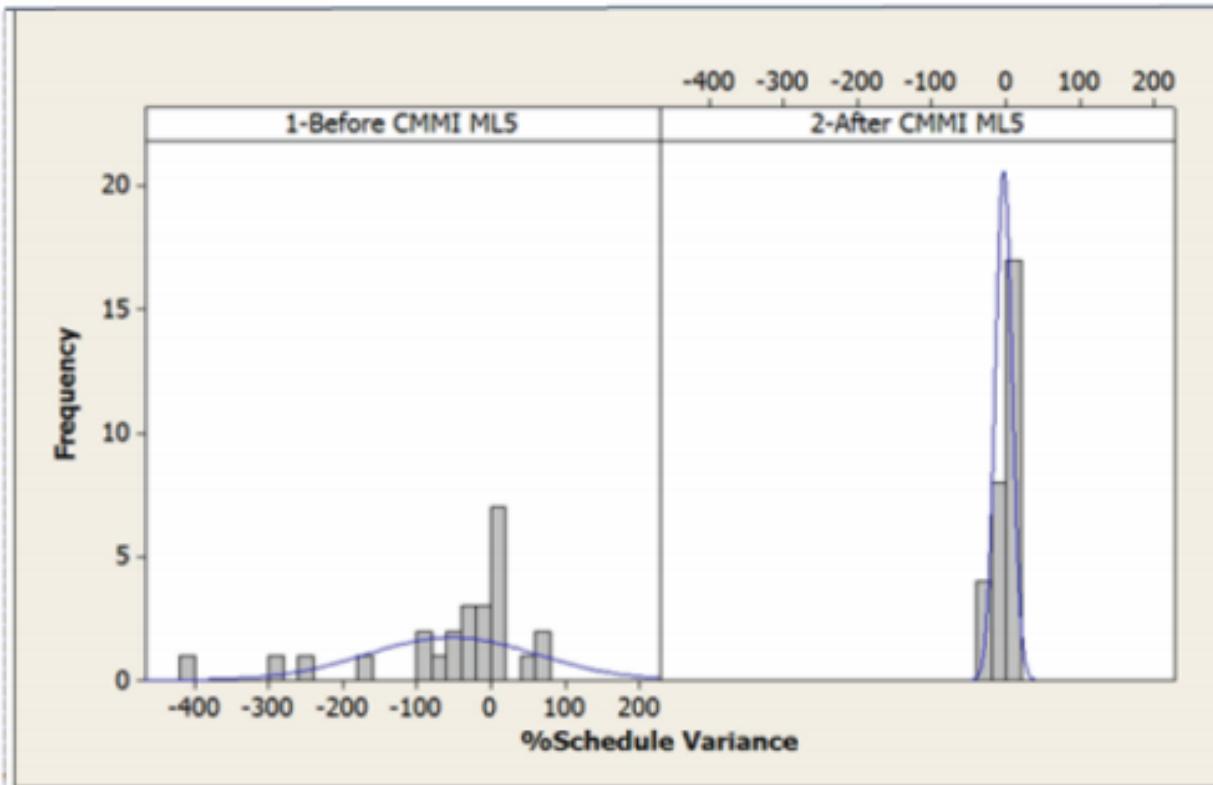
# Software Process Improvement : CMMI

- Software Engineering Institute collects quantitative measures of CMMI performance improvement:

Performance Category	Median Improvement
COST REDUCTION	34 %
SCHEDULE REDUCTION	54 %
PRODUCTIVITY INCREASE	61 %
PRODUCT QUALITY INCREASE	48 %
CUSTOMER SATISFACTION INCREASE	14 %
RETURN ON INVESTMENT	4:1

# Benefits from Using CMMI

- Reduce schedule variance

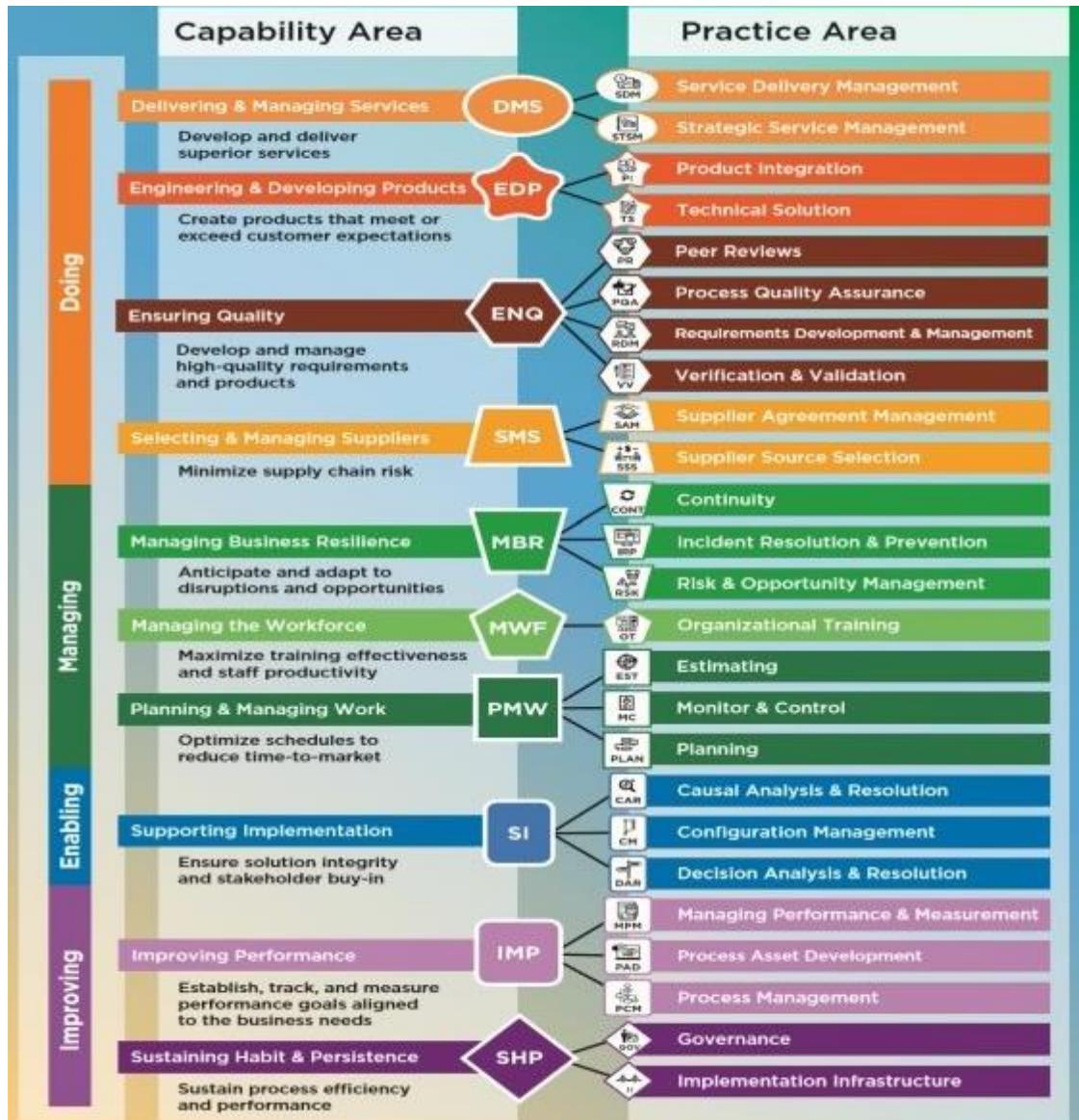


# Benefits from Using CMMI

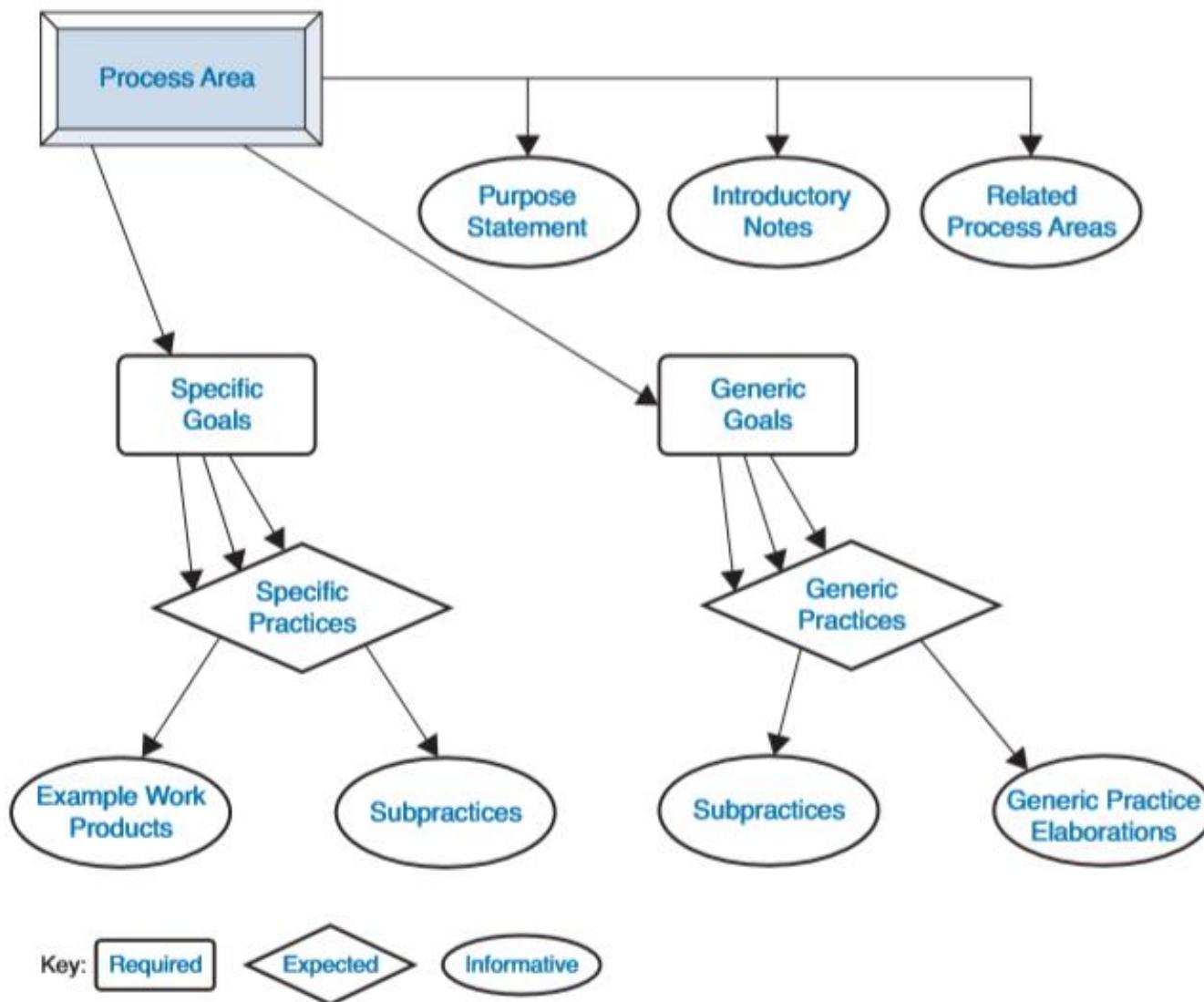
1. Better customer satisfaction
2. Increased quality
3. More accurate schedules
4. Lower development costs
5. Substantial return on investment
6. Improved employee morale and reduced turnover

# Benefits from Using CMMI

- Organization's activities are explicitly **linked to its business objectives.**
- **Visibility into the organization's** activities is increased to help to ensure that the product or service **meets the customer's expectations.**
- The teams learn from **new areas of best practice** (e.g., measurement, risk)
- *CMMI is being adopted worldwide, including North America, Europe, Asia, Australia, South America, and Africa.*



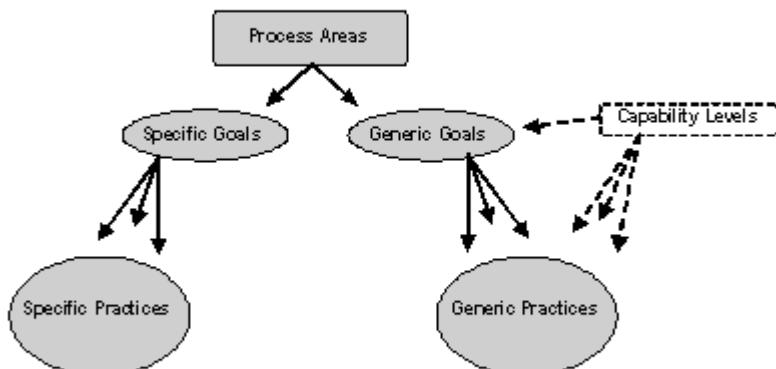
# CMMI Model Components



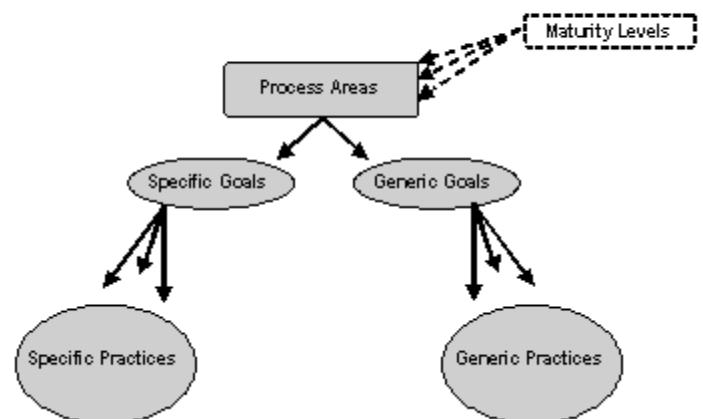
# CMMI - DEV

- Continues architecture Tolydinė architektūra
- Staged architecture Pakopinė architektūra

Continuous Representation

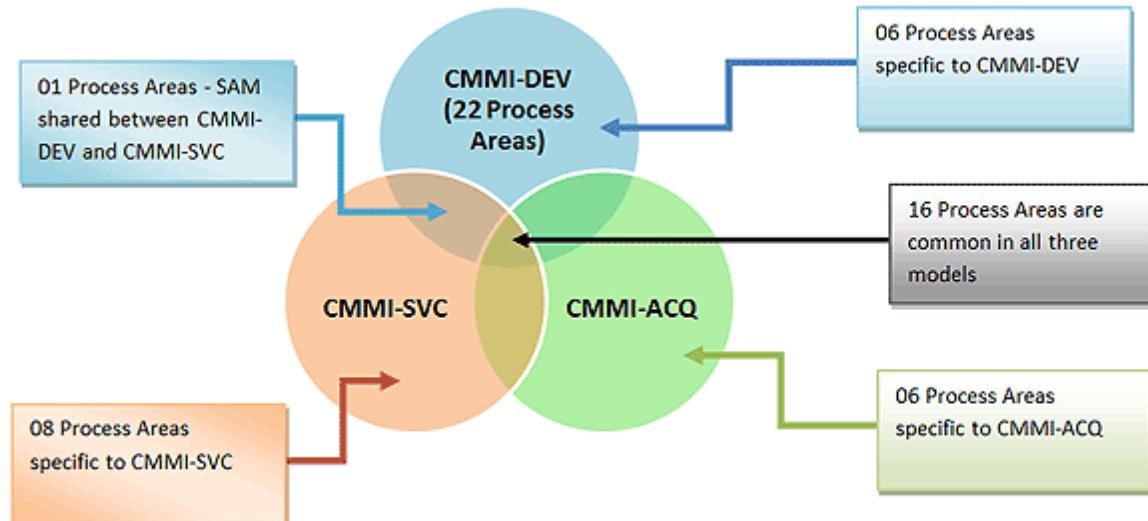


Staged Representation

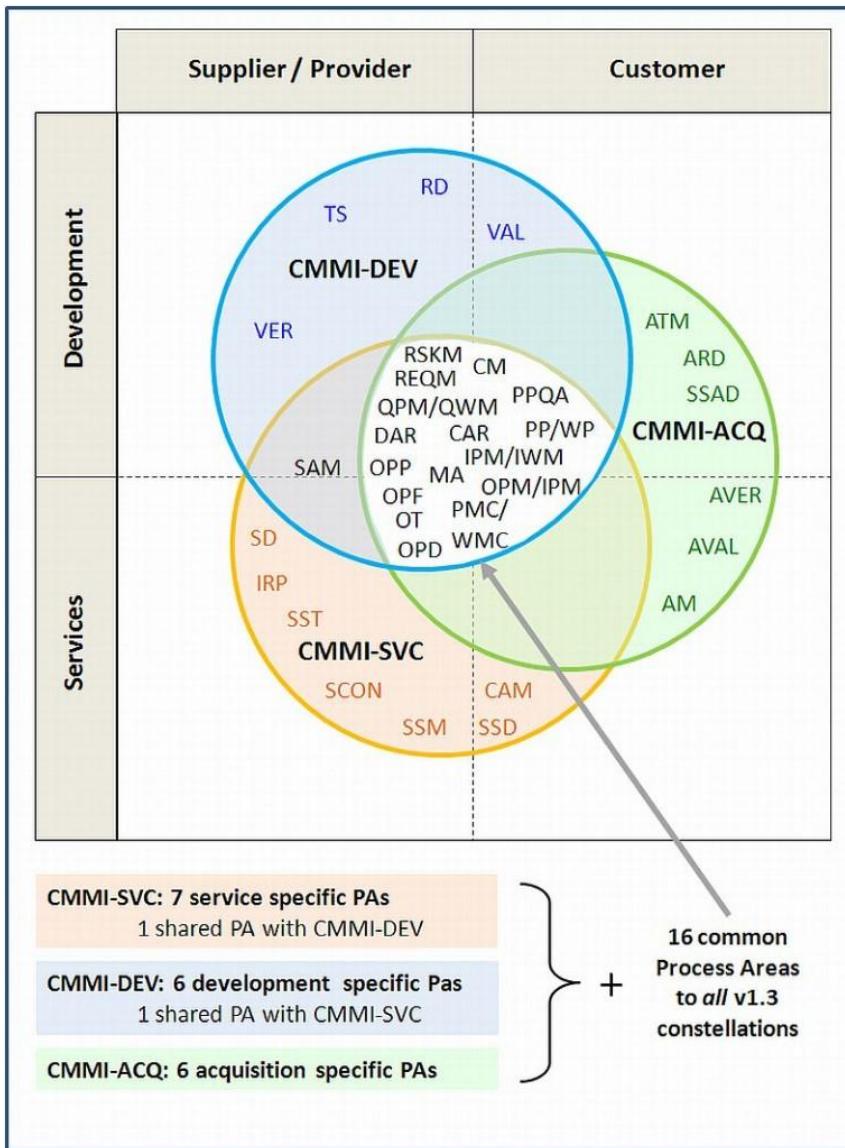


# CMMI – DEV Staged architecture

- CMMI for Development (CMMI-DEV)
  - It addresses product and service development processes.
- CMMI for Acquisition (CMMI-ACQ),
  - It addresses supply chain management, acquisition, and outsourcing processes in government and industry.
- CMMI for Services (CMMI-SVC),
  - It addresses guidance for delivering services within an organization and to external customers.

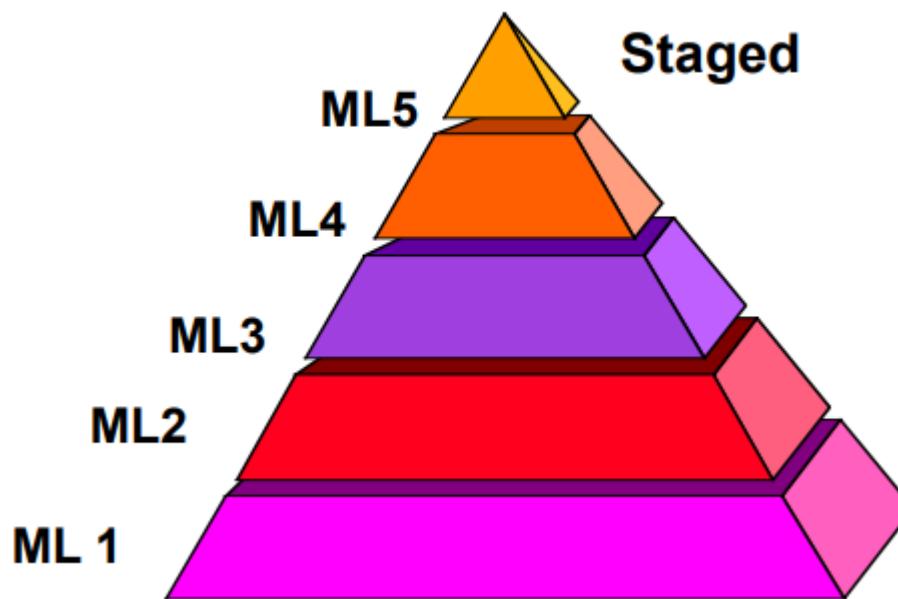


# CMMI – DEV Staged architecture



# CMMI – DEV Staged architecture

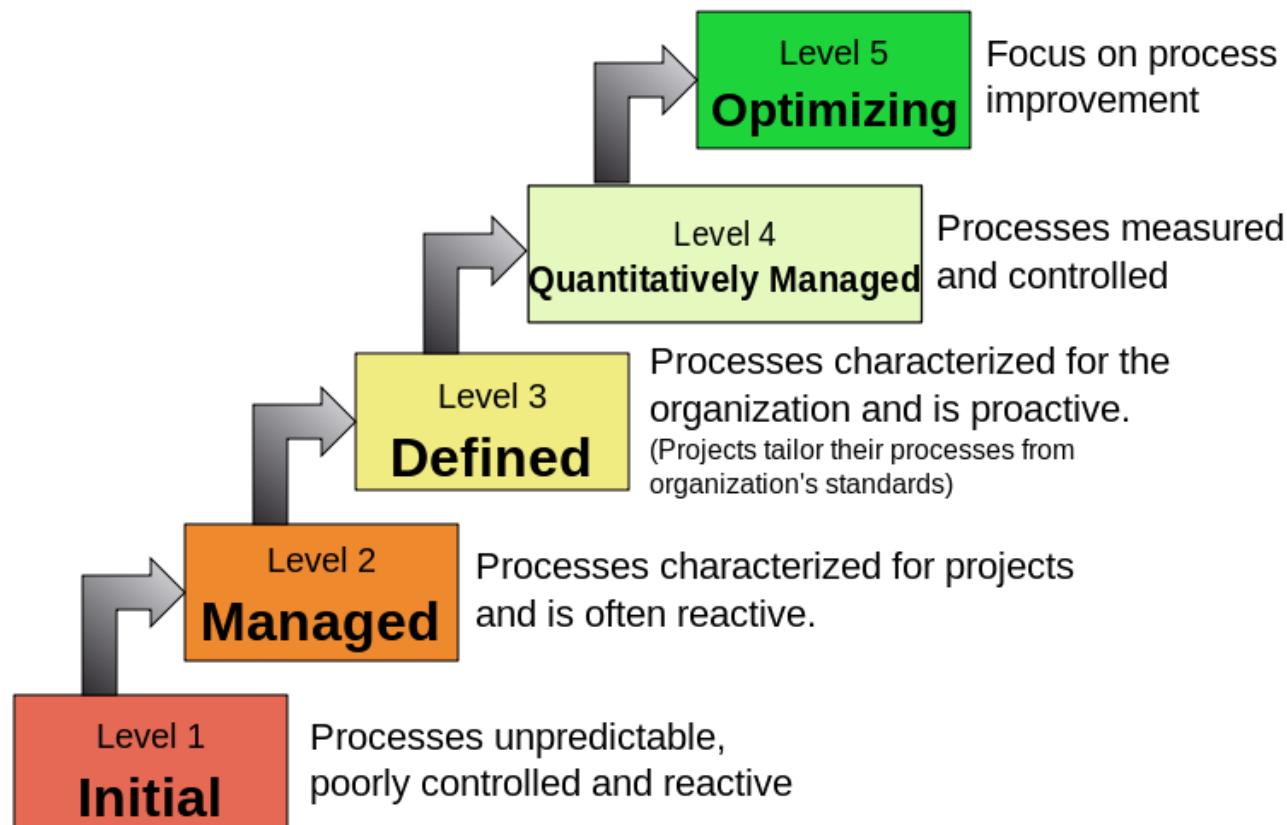
- Defines 5 maturity levels (MLs)
- In order to achieve a maturity level all process areas associated to this level,
- Plus all process areas associated with levels below must have a certain minimal capability.



# Capability Maturity Model Integration

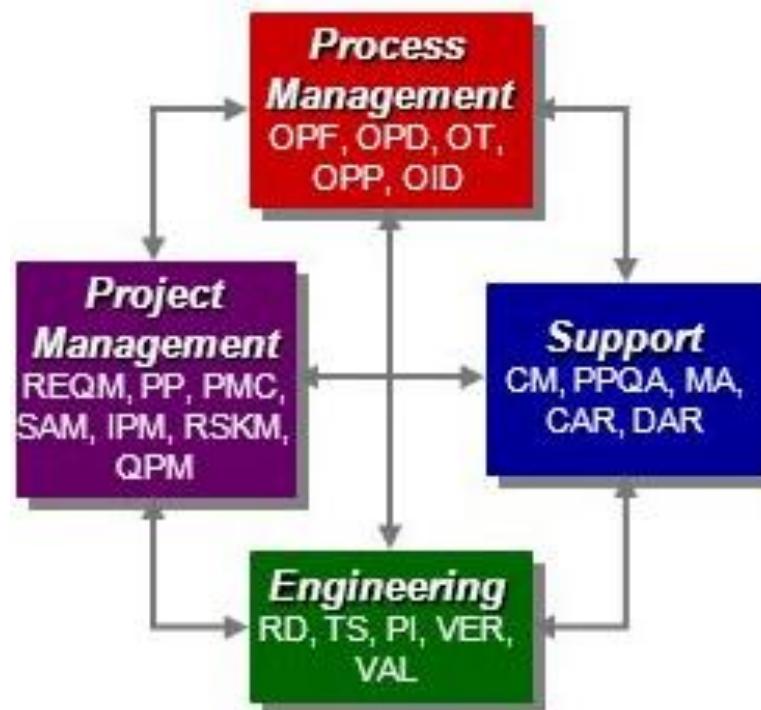
## CMMI

### Characteristics of the Maturity levels



# CMMI – DEV Staged architecture

- The 22 Process Areas (PAs) are grouped into four categories:
  1. Engineering
  2. Process Management
  3. Project Management
  4. Support



# CMMI – DEV

## Staged architecture

- The 22 Process Areas

### Process Areas (CMMI-Dev)

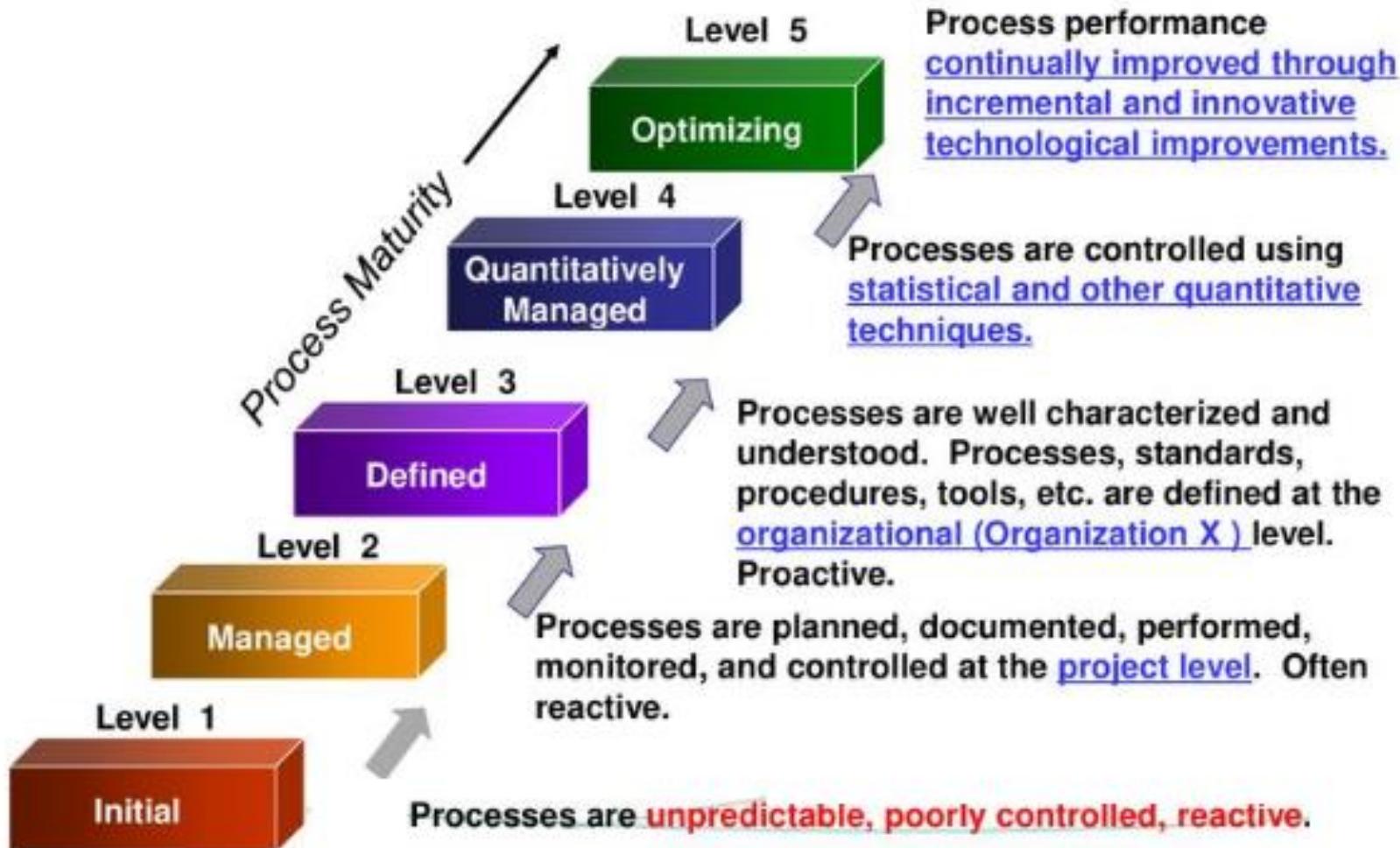
Requirements Management (REQM)  
Project Planning (PP)  
Project Monitoring and Control (PMC)  
Measurement and Analysis (MA)  
Process and Product Quality Assurance (PPQA)  
Configuration Management (CM)  
Supplier Agreement Management (SAM)

Requirements Development (RD)  
Technical Solution (TS)  
Product Integration (PI)  
Verification (VER)  
Validation (VAL)  
Organizational Process Focus (OPF)  
Organizational Process Definition (OPD)  
Organizational Training (OT)  
Integrated Project Management (IPM)  
Risk Management (RSKM)  
Decision Analysis and Resolution (DAR)

Organizational Process Performance (OPP)  
Quantitative Project Management (QPM)

Organizational Innovation & Deployment (OID)  
Causal Analysis and Resolution (CAR)

# CMMI – DEV Staged architecture



# CMMI – DEV Staged architecture

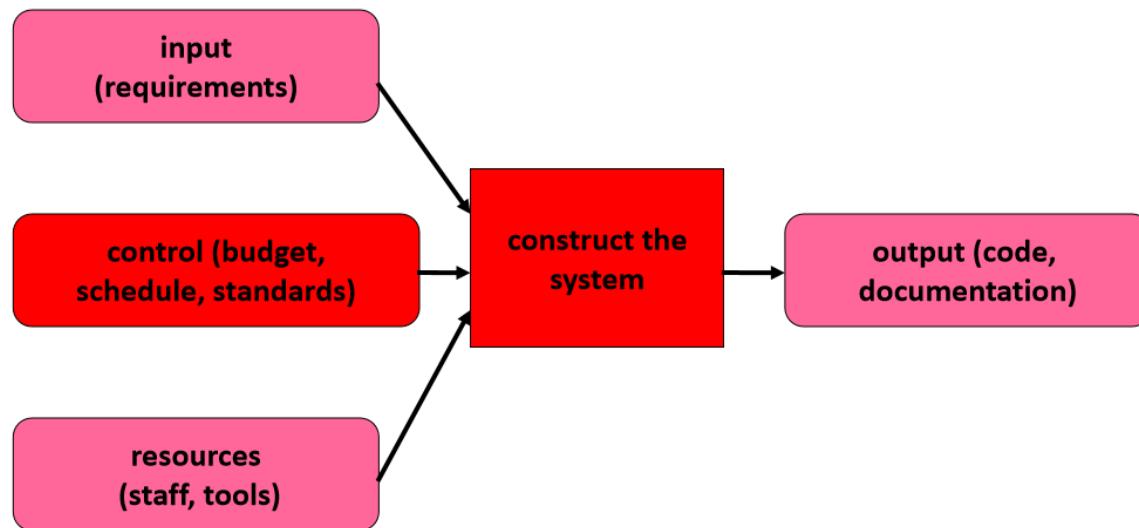
Level	Project Management	Engineering	Support	Process Management
5 Optimizing			CAR: Causal Analysis and Resolution	OPM: Organizational Performance Management
4 Quantitatively Managed	QPM: Quantitative Project Management			OPP: Organizational Process Performance
3 Defined	IPM: Integrated Project Management RSKM: Risk Management	RD: Requirements Development TS: Technical Solution PI: Product Integration VER: Verification VAL: Validation	DAR: Decision Analysis and Resolution	OPF: Organizational Process Focus OPD: Organizational Process Definition OT: Organizational Training
2 Managed	PP: Project Planning PMC: Project Monitoring and Control SAM: Supplier Agreement Management REQM: Requirements Management		MA: Measurement and Analysis PPQA: Process & Product Quality Assurance CM: Configuration Management	
1 Initial				

# Maturity Levels: Initial

- Processes are **usually ad hoc and chaotic.**
- The organization usually does **not provide a stable environment.**
- Success in these organizations **depends on the competence and heroics of the people** in the organization and not on the use of proven processes.
- Maturity level INITIAL organizations often produce products and services that work;
  - however, they **frequently exceed the budget and schedule of their projects.**

# Maturity Levels

- **Managed or Repeatable**
- The projects of the organization have **ensured that requirements are managed and that processes are planned, performed, measured, and controlled.**
- Maturity level 2 helps to ensure that existing practices are **retained during times of stress**

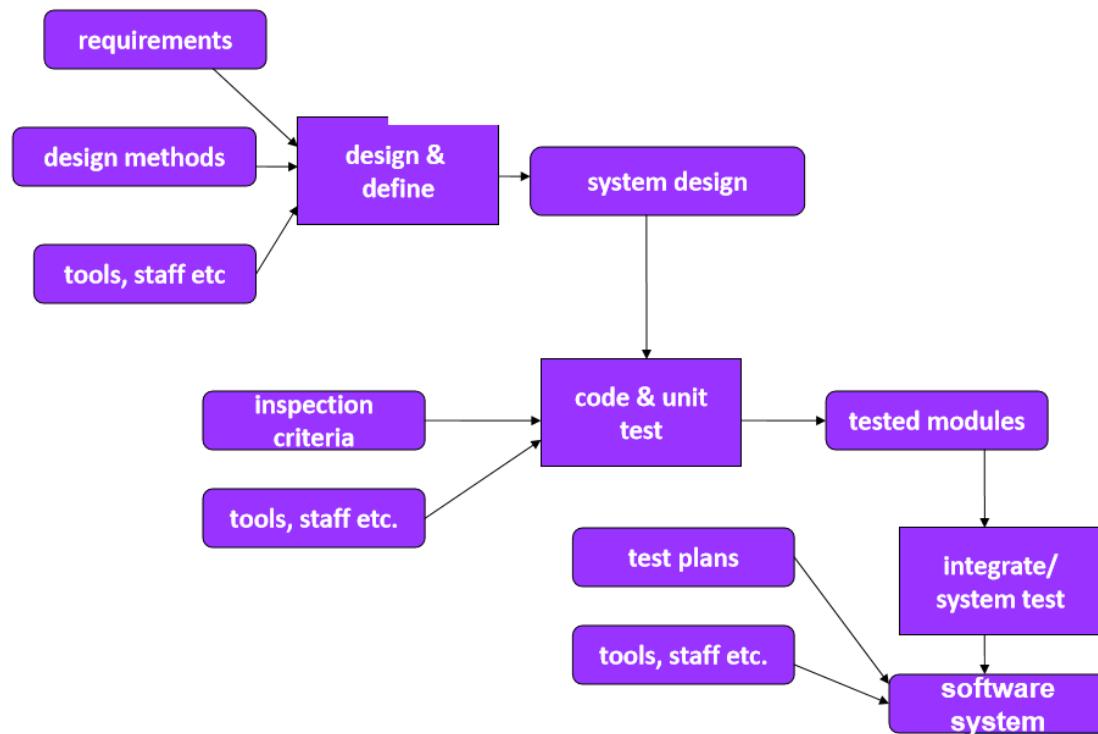


# Maturity Levels

- Managed
- To move to ML 2 (Managed) focus is on process areas:
  - Requirements management
  - Project planning
  - Project monitoring & control
  - Sub-contract management
  - Measurement and analysis
  - Quality assurance
  - Configuration management

# Maturity Levels: Defined

- Processes are well characterized and **understood**, and are **described in standards, procedures, tools, and methods**.
- A critical distinction between maturity level 2 and maturity level 3 **is the scope of standards, process descriptions, and procedures**.



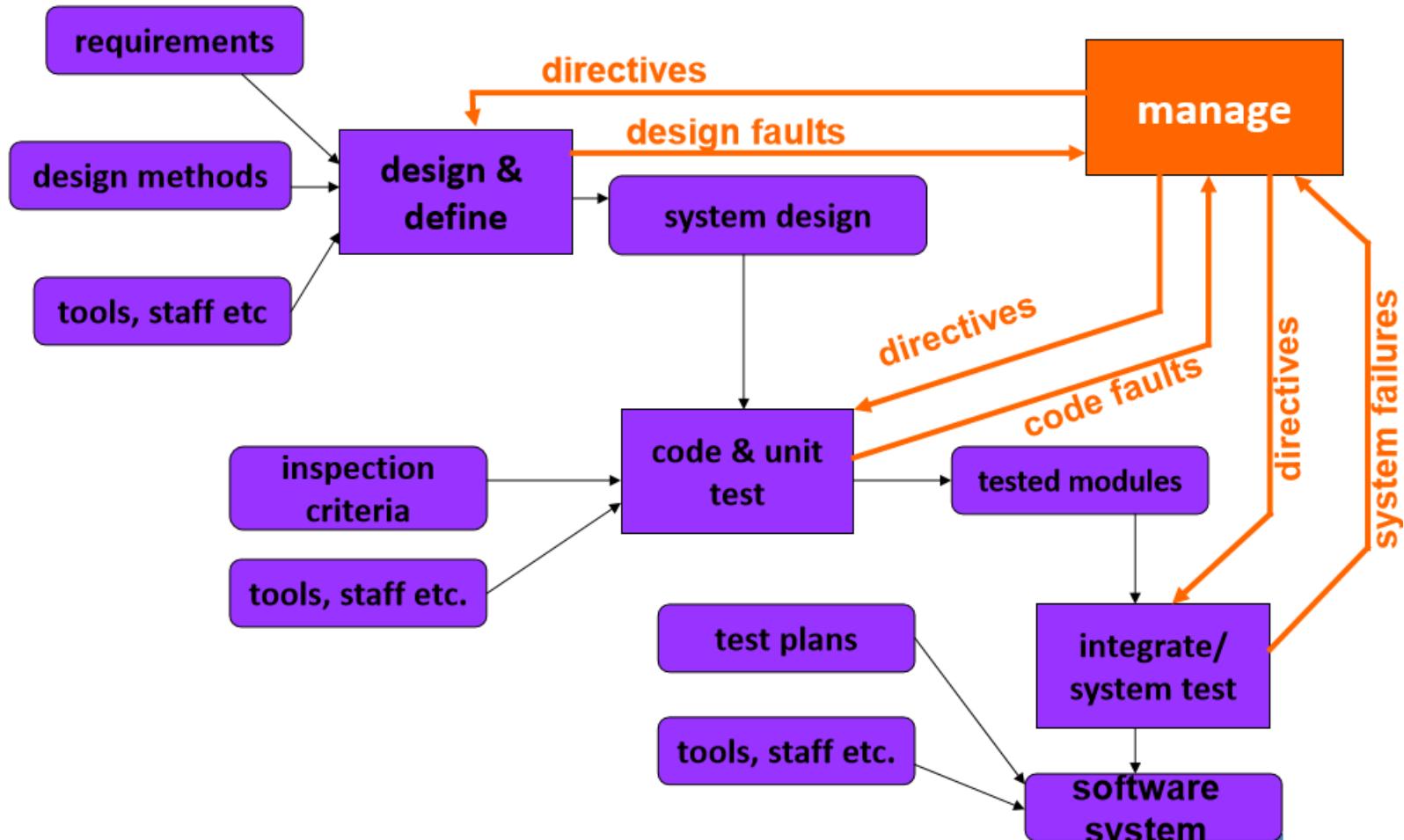
# Maturity Levels: Defined

- To move to ML 3 (Defined) focus on process areas:
  - Requirements development and technical solution
  - Product integration
  - Verification and validation
  - Organizational process definition
  - Organizational process focus
  - Organizational training
  - Risk management
  - Integrated project management
  - Decision analysis and resolution

# Maturity Levels: Quantitatively Managed

- **Subprocesses are selected that significantly contribute to overall process performance.**
  - These selected subprocesses are controlled using statistical and other quantitative techniques.
- **Quantitative objectives for quality and process performance are established** and used as criteria in managing processes.
- Quantitative objectives are **based on the needs of the customer, end users, organization, and process implementers.**

# Maturity Levels: Quantitatively Managed



# Maturity Levels: Quantitatively Managed

- Maturity level 5 focuses on **continually improving process performance through both incremental and innovative technological improvements**
- Processes are **continually improved based on a quantitative understanding of the common causes of variation inherent in processes.**
- Quantitative process-improvement objectives for the organization are established, **continually revised to reflect changing business objectives**, and used as criteria in managing process improvement.

# Maturity Levels: Optimizing

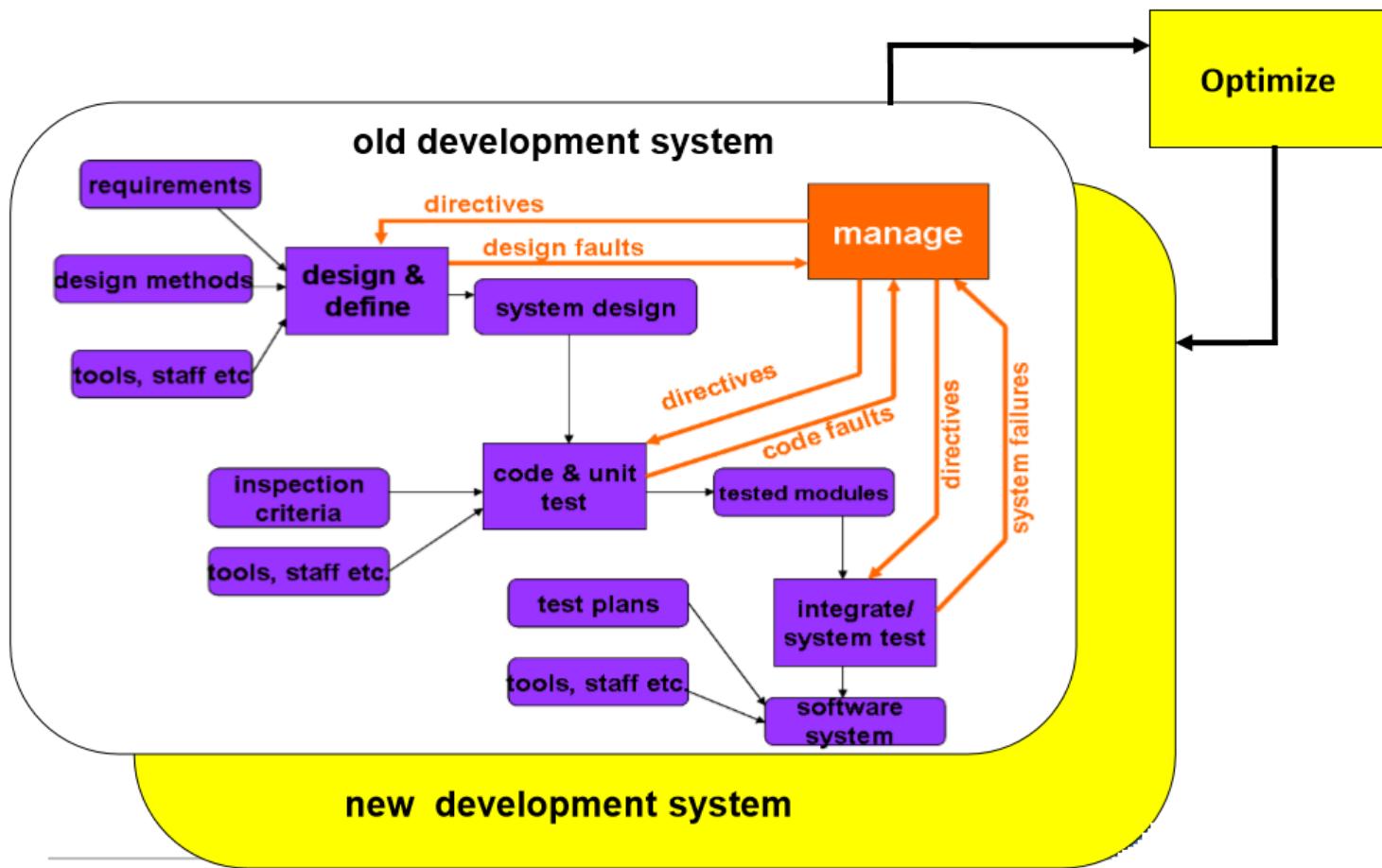
- To move to ML 5 (Optimizing) focus on process areas:
  - Causal analysis and resolution
  - Organizational performance management

# Maturity Levels: Optimizing

- To move to ML 5 (Optimizing) focus on process areas:
  - Causal analysis and resolution
  - Organizational performance management

# Maturity Levels: Optimizing

- Optimizing

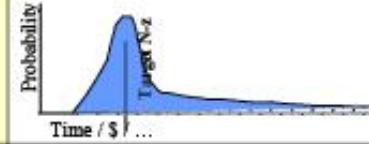
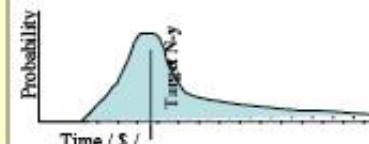
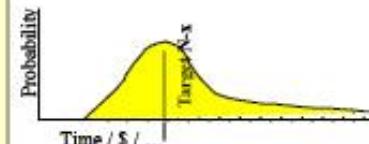
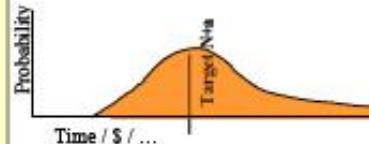
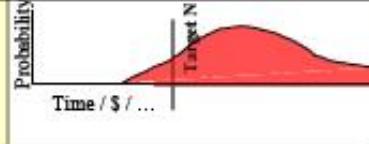


# CMMI Evaluation – Questionnaire Example

- Related to Process Areas  
**Requirements Development**

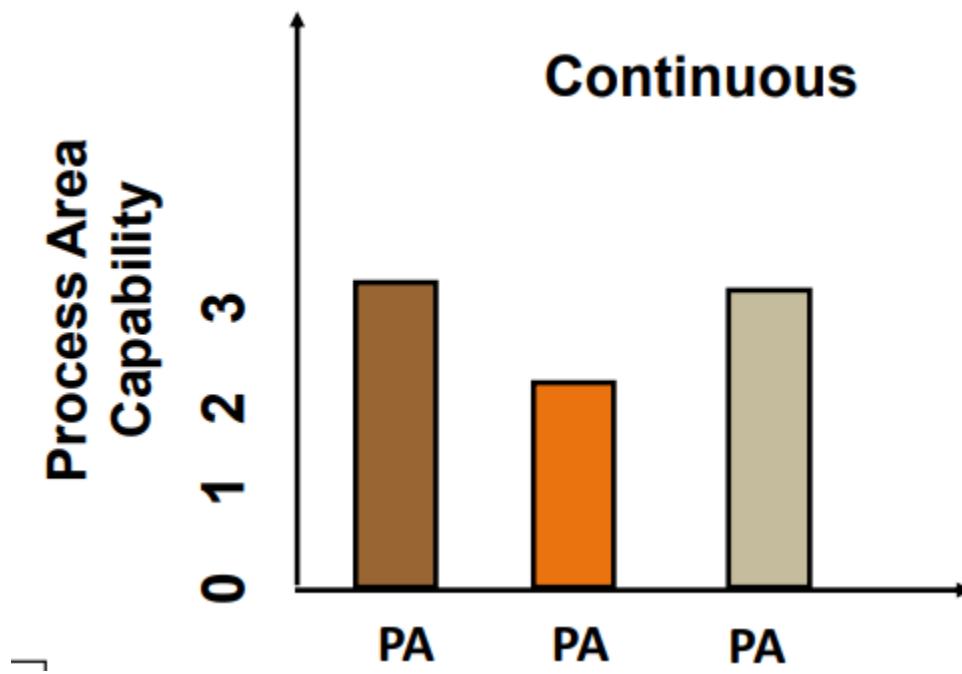
REQUIREMENTS ANALYSIS															
<ul style="list-style-type: none"><li>• Functional activities<ul style="list-style-type: none"><li>➢ Project development</li></ul></li></ul>															
1. Was more detailed information about the software requirements gathered from the customer requirements document, internal constraints and other sources supplied by the customer?															
<table border="1"><tr><td>No</td><td>Yes</td><td>Not applicable</td><td>Don't know</td></tr></table>				No	Yes	Not applicable	Don't know								
No	Yes	Not applicable	Don't know												
Observations:															
<ul style="list-style-type: none"><li>• Is it documented?<table border="1"><tr><td>No</td><td>Yes</td><td>Not applicable</td><td>Don't know</td></tr></table></li><li>• Was it carried out according to an established procedure?<table border="1"><tr><td>No</td><td>Yes</td><td>Not applicable</td><td>Don't know</td></tr></table></li><li>• Is the procedure documented?<table border="1"><tr><td>No</td><td>Yes</td><td>Not applicable</td><td>Don't know</td></tr></table></li></ul>				No	Yes	Not applicable	Don't know	No	Yes	Not applicable	Don't know	No	Yes	Not applicable	Don't know
No	Yes	Not applicable	Don't know												
No	Yes	Not applicable	Don't know												
No	Yes	Not applicable	Don't know												
2. Was a logic model of the product built from the functional requirements and the development methodology selected for the project?															
<table border="1"><tr><td>No</td><td>Yes</td><td>Not applicable</td><td>Don't know</td></tr></table>				No	Yes	Not applicable	Don't know								
No	Yes	Not applicable	Don't know												
Observations:															
<ul style="list-style-type: none"><li>• Is it documented?<table border="1"><tr><td>No</td><td>Yes</td><td>Not applicable</td><td>Don't know</td></tr></table></li><li>• Was it carried out according to an established procedure?<table border="1"><tr><td>No</td><td>Yes</td><td>Not applicable</td><td>Don't know</td></tr></table></li><li>• Is the procedure documented?<table border="1"><tr><td>No</td><td>Yes</td><td>Not applicable</td><td>Don't know</td></tr></table></li></ul>				No	Yes	Not applicable	Don't know	No	Yes	Not applicable	Don't know	No	Yes	Not applicable	Don't know
No	Yes	Not applicable	Don't know												
No	Yes	Not applicable	Don't know												
No	Yes	Not applicable	Don't know												
3. Were the requirements obtained from the customer analysed and verified to check that they were satisfactory and to detect any errors with regard to ambiguous, incomplete, unfeasible requirements, etc.?															
4. Was the logic model of the product validated against the functional requirements to assure that it satisfactorily met customer specifications?															
5. Was a prioritized list of requirements drawn up to enable gradual product development?															
6. Was a software requirements document drawn up on the basis of customer requirements internal organizational constraints?															
7. Was the software requirements document revised and approved by the customer? <ul style="list-style-type: none"><li>7.1. Was the software requirements document presented formally and comprehensibly to the customer?</li><li>7.2. In the event that modifications were proposed to the software requirements document,<ul style="list-style-type: none"><li>Was their impact on the product logic model assessed?</li><li>Was the completeness, consistency and non-ambiguity of the new set of software requirements assessed?</li></ul></li></ul>															
7.3. Did the customer approve the software requirements document in writing?															
8. Was the software requirements document managed and controlled? <div style="border: 1px solid black; padding: 5px;">Management and control imply that a check is kept on versions and changes to the Software Requirements Document, that is, that the latest version is known at all times and amendments are introduced in a controlled manner.</div>															
9. Was the software requirements document delivered to the Software Configuration Management group after acceptance and approval?															

# CMMI – DEV Staged architecture

Level	Characteristic	Process Area	Predicted Performance	Result
5 Optimising	Continuous process improvement	Causal Analysis and Resolution Organisational Innovation and Deployment		<b>Productivity &amp; Quality</b>
4 Quantitatively Managed	Process measured and controlled	Quantitative Project Management Organisational Process Performance		
3 Defined	Process characterised for the organisation and is proactive	Requirements Development Technical Solution Product Integration Verification Validation Organisational Process Focus Organisational Process Definition Organisational Training Integrated Project Management Risk Management Integrated Teaming Integrated Supplier Management Decision Analysis and Resolution Organisational Environment for Integration		R I S
2 Managed	Process characterised for projects and is often reactive	Requirements Management Project Planning Project Monitoring and Control Supplier Agreement Management Measurement and Analysis Process and Product Quality Assurance Configuration Management		K
1 Initial	Process unpredictable, poorly controlled, reactive			

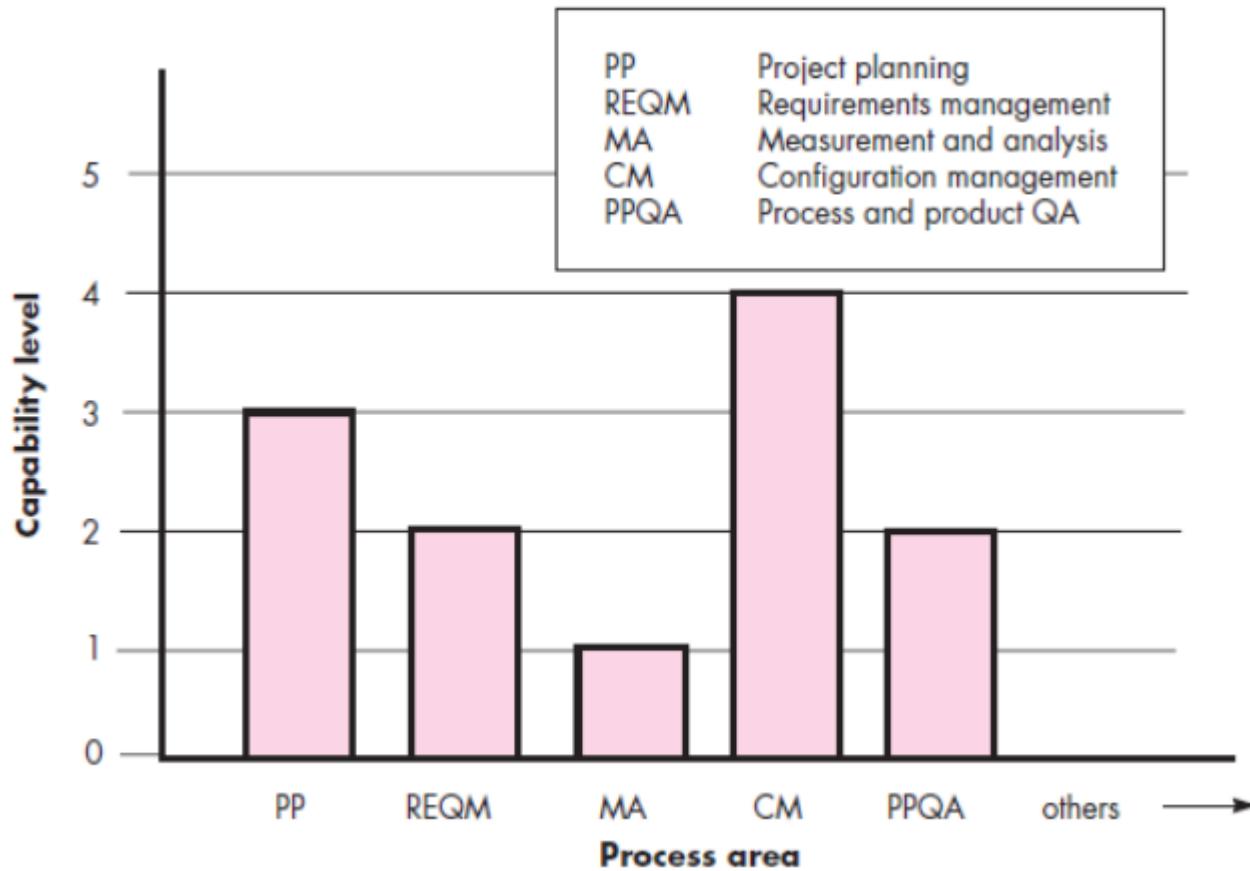
# CMMI – DEV Continues architecture

- A maturity profile is established based on the **capabilities of individual process areas**



# CMMI – DEV Continues architecture

- A maturity profile is established based on the capabilities of individual process areas



# Capability levels

- **Capability Level 0: Incomplete**
  - An incomplete process is a process that either is **not performed** or is **partially performed**.

# Capability Levels

- **Capability Level 1: Performed**
  - A capability level 1 process is characterized as a **performed process**.
  - A performed process is a process that:
    - **accomplishes the needed work to produce work products**
    - **without relying entirely on process documentation or plan**

# Capability Levels

- **Capability Level 2: Managed**
  - A capability level 2 process is characterized as a **managed process**.
  - A managed process is a performed process that is planned and executed in accordance with:
    - **policy**;
    - **employs skilled** people having adequate resources to produce controlled outputs;
    - **involves relevant stakeholders**;
    - **is monitored**,
    - **controlled, and reviewed**;
    - **is evaluated for adherence to its process description**.

# Capability Levels

- **Capability Level 3: Defined**
  - A defined process is a managed process that is tailored **from the organization's set of standard processes according to the organization's tailoring guidelines**;
  - has a **maintained process description**;
  - and **contributes process related experiences to the organizational process assets**.

# Continuous Representation: Capability Levels

<i>Level</i>	<i>Continuous Representation Capability Levels</i>	<i>Staged Representation Maturity Levels</i>
Level 0	Incomplete	
Level 1	Performed	Initial
Level 2	Managed	Managed
Level 3	Defined	Defined
Level 4		Quantitatively Managed
Level 5		Optimizing

# Target Profiles and Equivalent Staging

Name	Abbr	ML	CL1	CL2	CL3
Requirements Management	REQM	2	<i>Target Profile 2</i>		
Project Planning	PP	2			
Project Monitoring and Control	PMC	2			
Supplier Agreement Management	SAM	2			
Measurement and Analysis	MA	2			
Process and Product Quality Assurance	PPQA	2			
Configuration Management	CM	2			
Requirements Development	RD	3	<i>Target Profile 3</i>		
Technical Solution	TS	3			
Product Integration	PI	3			
Verification	VER	3			
Validation	VAL	3			
Organizational Process Focus	OPF	3			
Organizational Process Definition + IPPD	OPD + IPPD	3			
Organizational Training	OT	3			
Integrated Project Management + IPPD	IPM + IPPD	3			
Risk Management	RSKM	3			
Decision Analysis and Resolution	DAR	3	<i>Target Profile 4</i>		
Organizational Process Performance	OPP	4			
Quantitative Project Management	QPM	4	<i>Target Profile 5</i>		
Organizational Innovation and Deployment	OID	5			
Causal Analysis and Resolution	CAR	5			

# ISO 15504 Maturity Models

- SPICE (ISO 15504)
  - Different structure of processes than in CMMI (roughly following ISO 12207)
  - 6 Maturity levels (beginning at Level 0)
- <https://cmmiinstitute.com/>

# ISO/IEC 15504 IT process assessment

- To provide **guidance on the assessment of software development processes**
- Process Reference Model:
  - Needs a defined set of processes that represent good practice to be the benchmark
  - **ISO 12207 is the default process reference model**
  - Could use others in specific environments

# Process Categories ISO 15504

- Customer-supplier (CUS)
- Engineering (ENG)
- Project (PRO)
- Support (SUP)
- Organizing (ORG)

<http://www.rad.fr/spice1.htm>

# Summary list of base practices

## CUS.1 Acquire software product and/or service

- CUS.1.1 Identify the need
- CUS.1.2 Define the requirements
- CUS.1.3 Prepare acquisition strategy
- CUS.1.4 Prepare request for proposal
- CUS.1.5 Select software product supplier

## CUS.2 Establish contract

- CUS.2.1 Review before contract finalization
- CUS.2.2 Negotiate contract
- CUS.2.3 Determine interfaces to independent agents
- CUS.2.4 Determine interfaces to subcontractors

## CUS.3 Identify customer needs

- CUS.3.1 Obtain customer requirements and requests
- CUS.3.2 Understand customer expectations
- CUS.3.3 Keep customers informed

## CUS.4 Perform joint audits and reviews

- CUS.4.1 Establish joint audits and reviews
- CUS.4.2 Prepare for customer audits and reviews
- CUS.4.3 Conduct joint management reviews
- CUS.4.4 Conduct joint technical reviews
- CUS.4.5 Support customer acceptance review
- CUS.4.6 Perform joint process assessment

## CUS.6 Support operation of software

- CUS.6.1 Identify operational risks
- CUS.6.2 Perform operational testing
- CUS.6.3 Operate the software
- CUS.6.4 Resolve operational problems
- CUS.6.5 Handle user requests
- CUS.6.6 Document temporary work-arounds
- CUS.6.7 Monitor system capacity and service

## CUS.7 Provide customer service

- CUS.7.1 Train customer
- CUS.7.2 Establish product support
- CUS.7.3 Monitor performance
- CUS.7.4 Install product upgrades

## CUS.8 Assess customer satisfaction

- CUS.8.1 Determine customer satisfaction level
- CUS.8.2 Compare with competitors
- CUS.8.3 Communicate customer satisfaction

## ENG Engineering process category

- ### ENG.1 Develop system requirements and design
- ENG.1.1 Specify system requirements
  - ENG.1.2 Describe system architecture
  - ENG.1.3 Allocate requirements
  - ENG.1.4 Determine release strategy

# Summary list of base practices

## **ENG.1 Develop system requirements and design**

- ENG.1.1 Specify system requirements
- ENG.1.2 Describe system architecture
- ENG.1.3 Allocate requirements
- ENG.1.4 Determine release strategy

## **ENG.2 Develop software requirements**

- ENG.2.1 Determine software requirements
- ENG.2.2 Analyze software requirements
- ENG.2.3 Determine operating environment impact
- ENG.2.4 Evaluate requirements with customer
- ENG.2.5 Update requirements for next iteration

## **ENG.3 Develop software design**

- ENG.3.1 Develop software architectural design
- ENG.3.2 Design interfaces at top level
- ENG.3.3 Develop detailed design
- ENG.3.4 Establish traceability

## **ENG.4 Implement software design**

- ENG.4.1 Develop software units
- ENG.4.2 Develop unit verification procedures
- ENG.4.3 Verify the software units

## **ENG.5 Integrate and test software**

- ENG.5.1 Determine regression test strategy
- ENG.5.2 Build aggregates of software units
- ENG.5.3 Develop tests for aggregates
- ENG.5.4 Test software aggregates
- ENG.5.5 Develop tests for software
- ENG.5.6 Test integrated software

## **ENG.6 Integrate and test system**

- ENG.6.1 Build aggregates of system elements
- ENG.6.2 Develop tests for aggregates
- ENG.6.3 Test system aggregates
- ENG.6.4 Develop tests for system
- ENG.6.5 Test integrated system

## **ENG.7 Maintain system and software**

- ENG.7.1 Determine maintenance requirements
- ENG.7.2 Analyze user problem and enhancements
- ENG.7.3 Determine modifications for next upgrade
- ENG.7.4 Implement and test modifications
- ENG.7.5 Upgrade user system

# Summary list of base practices

## PRO.1 Plan project life cycle

- PRO.1.1 Evaluate options for product development
- PRO.1.2 Select software life cycle model
- PRO.1.3 Describe activities and tasks
- PRO.1.4 Establish task sequences
- PRO.1.5 Document activities

## PRO.2 Establish project plan

- PRO.2.1 Develop work breakdown structure
- PRO.2.2 Identify project standards
- PRO.2.3 Identify specialized facilities
- PRO.2.4 Determine reuse strategy
- PRO.2.5 Develop project estimates
- PRO.2.6 Identify initial project risks
- PRO.2.7 Identify project measures
- PRO.2.8 Establish project schedule
- PRO.2.9 Establish project commitments
- PRO.2.10 Document project plans

## PRO.3 Build project teams

- PRO.3.1 Define project teams
- PRO.3.2 Empower project teams
- PRO.3.3 Maintain project team interactions
- PRO.3.4 Manage inter-team issues

## PRO.4 Manage requirements

- PRO.4.1 Agree on requirements
- PRO.4.2 Establish customer requirements baseline
- PRO.4.3 Manage customer requirements changes
- PRO.4.4 Use customer requirements
- PRO.4.5 Maintain traceability

## PRO.5 Manage quality

- PRO.5.1 Establish quality goals
- PRO.5.2 Define quality metrics
- PRO.5.3 Identify quality activities
- PRO.5.4 Perform quality activities
- PRO.5.5 Assess quality
- PRO.5.6 Take corrective action

## PRO.6 Manage risks

- PRO.6.1 Establish risk management scope
- PRO.6.2 Identify risks
- PRO.6.3 Analyze and prioritize risks
- PRO.6.4 Develop mitigation strategies
- PRO.6.5 Define risk metrics
- PRO.6.6 Implement mitigation strategies
- PRO.6.7 Assess results of mitigation strategies
- PRO.6.8 Take corrective action

## PRO.7 Manage resources and schedule

- PRO.7.1 Acquire resources
- PRO.7.2 Track progress
- PRO.7.3 Conduct management reviews
- PRO.7.4 Conduct technical reviews
- PRO.7.5 Manage commitments

## PRO.8 Manage subcontractors

- PRO.8.1 Establish statement of work
- PRO.8.2 Qualify potential subcontractors
- PRO.8.3 Select subcontractor
- PRO.8.4 Establish and manage commitments
- PRO.8.5 Maintain communications
- PRO.8.6 Assess compliance
- PRO.8.7 Assess subcontractor quality

# Summary list of base practices

<b>SUP.1</b>	<b>Develop documentation</b>
SUP.1.1	Determine documentation requirements
SUP.1.2	Develop document
SUP.1.3	Check document
SUP.1.4	Distribute document
SUP.1.5	Maintain document
<b>SUP.2</b>	<b>Perform configuration management</b>
SUP.2.1	Establish configuration management library system
SUP.2.2	Identify configuration items
SUP.2.3	Maintain configuration item descriptions
SUP.2.4	Manage change requests
SUP.2.5	Control changes
SUP.2.6	Build product releases
SUP.2.7	Maintain configuration item history
SUP.2.8	Report configuration status
<b>SUP.3</b>	<b>Perform quality assurance</b>
SUP.3.1	Select project standards
SUP.3.2	Review software engineering activities
SUP.3.3	Audit work products
SUP.3.4	Report results
SUP.3.5	Handle deviations
<b>SUP.4</b>	<b>Perform problem resolution</b>
SUP.4.1	Prepare problem report
SUP.4.2	Track problem report
SUP.4.3	Prioritize problems
SUP.4.4	Determine resolution
SUP.4.5	Correct the defect
SUP.4.6	Distribute the correction
<b>SUP.5</b>	<b>Perform peer reviews</b>
SUP.5.1	Select work products
SUP.5.2	Identify review standards
SUP.5.3	Establish completion criteria
SUP.5.4	Establish re-review criteria
SUP.5.5	Distribute review materials
SUP.5.6	Conduct peer review
SUP.5.7	Document action items
SUP.5.8	Track action items

# Summary list of base practices

<b>ORG.1</b>	<b>Engineer the business</b>
ORG.1.1	Establish strategic vision
ORG.1.2	Deploy vision
ORG.1.3	Establish quality culture
ORG.1.4	Build integrated teams
ORG.1.5	Provide incentives
ORG.1.6	Define career plans
<b>ORG.2</b>	<b>Define the process</b>
ORG.2.1	Define goals
ORG.2.2	Identify current activities, roles and responsibilities
ORG.2.3	Identify inputs and outputs
ORG.2.4	Define entry and exit criteria
ORG.2.5	Define control points
ORG.2.6	Identify external interfaces
ORG.2.7	Identify internal interfaces
ORG.2.8	Define quality records
ORG.2.9	Define process measures
ORG.2.10	Document the standard process
ORG.2.11	Establish policy
ORG.2.12	Establish performance expectations
ORG.2.13	Deploy the process
<b>ORG.3</b>	<b>Improve the process</b>
ORG.3.1	Identify improvement opportunities
ORG.3.2	Define scope of improvement activities
ORG.3.3	Understand the process
ORG.3.4	Identify improvements
ORG.3.5	Prioritize improvements
ORG.3.6	Define measures of impact
ORG.3.7	Change the process
ORG.3.8	Confirm the improvement
ORG.3.9	Deploy improvement
<b>ORG.4</b>	<b>Perform training</b>
ORG.4.1	Identify training needs
ORG.4.2	Develop or acquire training
ORG.4.3	Train personnel
ORG.4.4	Maintain training records

<b>ORG.5</b>	<b>Enable reuse</b>
ORG.5.1	Determine organizational reuse strategy
ORG.5.2	Identify reusable components
ORG.5.3	Develop reusable components
ORG.5.4	Establish a reuse library
ORG.5.5	Certify reusable components
ORG.5.6	Integrate reuse into life cycle
ORG.5.7	Propagate change carefully
<b>ORG.6</b>	<b>Provide software engineering environment</b>
ORG.6.1	Identify software engineering environment requirements
ORG.6.2	Provide a software engineering environment
ORG.6.3	Provide support for developers
ORG.6.4	Maintain software engineering environment
<b>ORG.7</b>	<b>Provide work facilities</b>
ORG.7.1	Provide productive workspace
ORG.7.2	Ensure data integrity
ORG.7.3	Provide data backups
ORG.7.4	Provide building facilities
ORG.7.5	Provide remote access facility

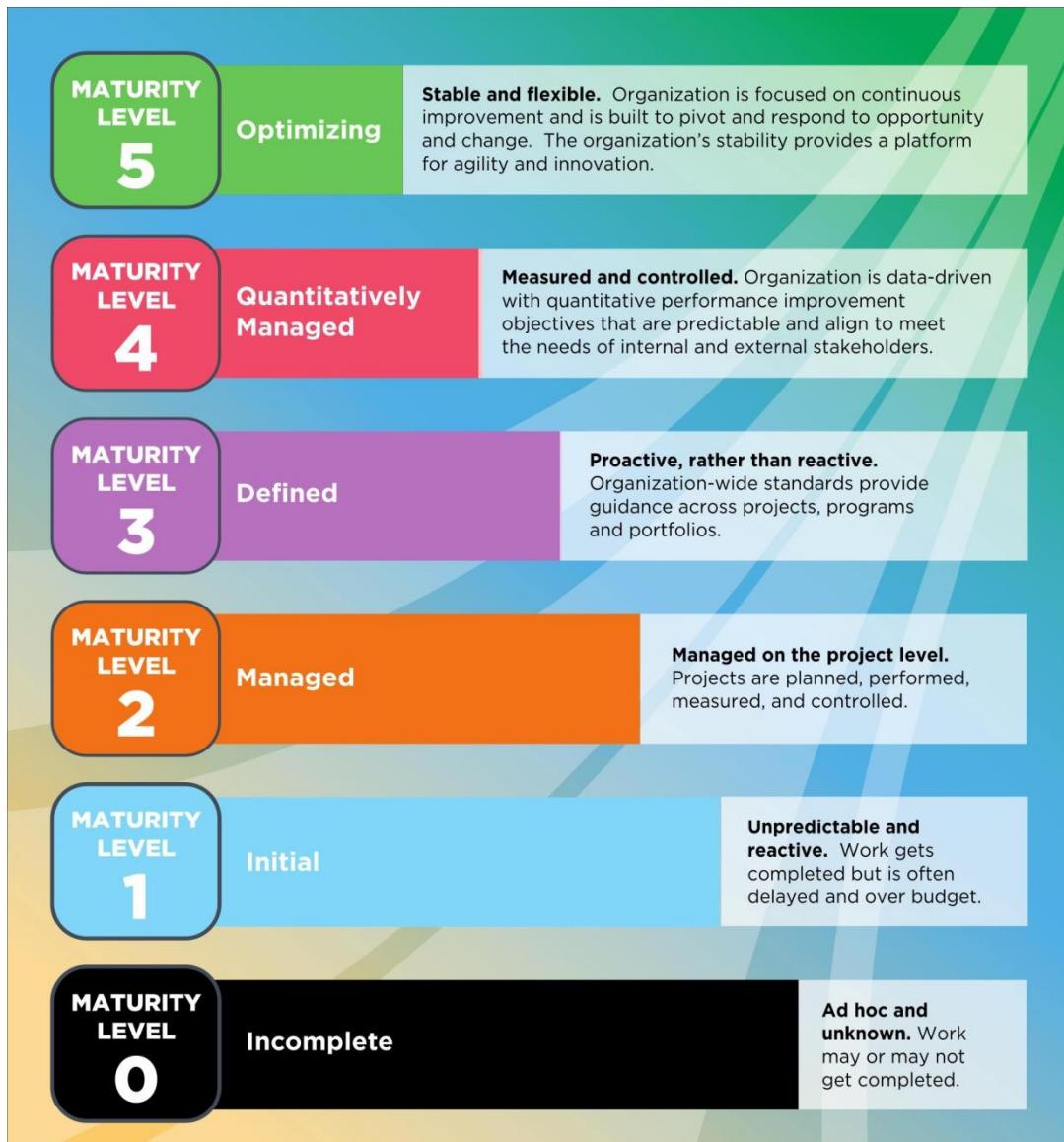
# ISO 15504 Process Assessment

- For each process in the relevant Process Reference Model:
  - For each set of attribute level criteria
  - Assess whether:
    - N: not achieved 0-15%
    - P: partially achieved >15%-50%
    - L: largely achieved >50%-85%
    - F: fully achieved >85%

# Produkto reikalavimų inžinerijos procesas

Dr. Asta Slotkienė

# CMMI 2.0 Maturity levels



# **CMMI for Development (CMMI-DEV)**

- **CMMI for Development is a framework for any organization that builds products and/or services.**
  - Organizations that are involved with developing may include **hardware and software companies**
- CMMI-DEV applies to all organizations that must consider **design and engineering during the development of a product or service.**

# New and Changed Practices

- Requirement Development and Management!!!

Level		Focus	Process Areas	Quality Productivity
5 Optimizing	Continuous Process Improvement	Causal Analysis and Resolution (CAR) Organizational Performance Management (OPM)		
4 Quantitatively Managed	Quantitative Management	Organizational Process Performance (OPP) Quantitative Project Management (QPM)		
3 Defined	Process Standardization	Integrated Project Management (IPM) Risk Management (RSKM) Decision Analysis and Resolution (DAR) Requirements Development (RD) Technical Solution (TS) Product Integration (PI) Verification (VER) Validation (VAL) Organizational Process Focus (OPF) Organizational Process Definition (OPD) Organizational Training (OT)		Risk Rework
2 Managed	Basic Project Management	Configuration Management (CM) Measurement and Analysis (MA) Project Monitoring and Control (PMC) Project Planning (PP) Process and Product Quality Assurance (PPQA) Requirements Management (REQM) Supplier Agreement Management (SAM)		
1 Initial				

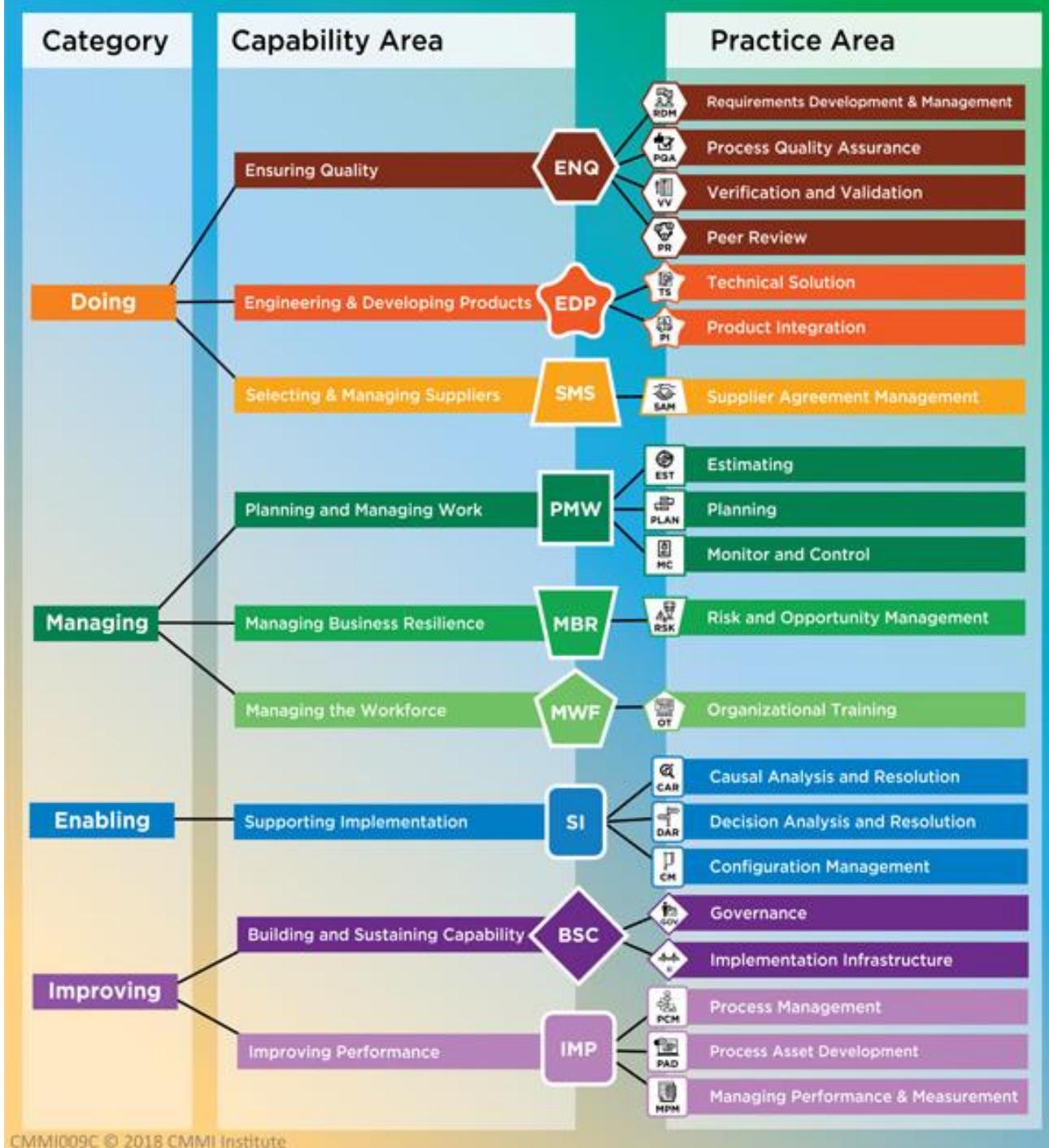
CMMI Development V1.3

Level		PA also exists at higher level	Practice Areas	Performance Capability
5 Optimizing			CAR, MPM (level 5 practices)	
4 Quantitatively Managed			CAR, PCM, SAM, MPM, PLAN, GOV (level 4 practices)	
3 Defined		✓	Causal Analysis and Resolution (CAR) Decision Analysis and Resolution (DAR) Risk and Opportunity Management (RSK) Organizational Training (OT) Process Management (PCM) Process Asset Development (PAD) Peer Reviews (PR) Verification and Validation (VV) Technical Solution (TS) Product Integration (PI)	
2 Managed		✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓	Supplier Agreement Management (SAM) Managing Performance and Measurement (MPM) Process Quality Assurance (PQA) Configuration Management (CM) Monitor and Control (MC) Planning (PLAN) Estimation (EST) Requirements Development & Management (RDM) Governance (GOV) Implementation Infrastructure (II)	Risk Rework
1 Initial / 0 Incomplete				

CMMI Development V2.0 – Simple View

# CMMI v2.0 Capability Areas





# CMMI v2.0 Capability Areas: Doing

- Includes Capability Areas for **producing, buying, and delivering quality solutions.**

Doing	Ensuring Quality	Ensuring Quality (ENQ)	Delivering and Managing Services
	Engineering & Developing Products	Requirements Development & Management (RDM) <small>ML2 ML3</small>	Service Delivery Management (SDM)
Delivering and Managing Services		Process Quality Assurance (PQA) <small>ML2 ML3</small>	Strategic Service Management (STSM)
	Selecting & Managing Suppliers	Verification & Validation (VV) <small>ML3</small> Peer Reviews (PR) <small>ML3</small>	Selecting & Managing Suppliers (SMS)
Selecting & Managing Suppliers		Engineering & Developing Products (EDP)	Supplier Source Selection (SSS)
		Technical Solution (TS) <small>ML3</small>	Supplier Agreement Management (SAM) <small>ML2 ML3</small>
		Product Integration (PI) <small>ML3</small>	

# **ENSURING QUALITY -helps to improve product and service quality**

- **REQUIREMENTS DEVELOPMENT AND MANAGEMENT (RDM)**
  - RDM helps to develop and update a joint understanding of the necessities and expectations of the solution.
  - Maturity Level 1=ML1, ML2, ML3
- **PROCESS QUALITY ASSURANCE (PQA)**
  - PQA helps to identify and prevent adverse effects, while ensuring the recurrence of positive results. Addressing the root problems quickly reduces the need for subsequent rework and improves productivity.
  - ML1, ML2, ML3
- **VERIFICATION AND VALIDATION (VV)**
  - VV ensures that requirements are fulfilled and the solution works as planned in the target environment.
  - ML1, ML2, ML3
- **PEER REVIEWS (PR)**
  - It helps to identify defects and issues in the solution.
  - ML1, ML2, ML3

# Ensuring Quality-> RDM

- Requirements Development and Management enables developing and keeping updated a common *understanding of needs and expectations for the solution.*

## Intent:

- Elicit requirements,
- ensure common **understanding by stakeholders**,
- align requirements, plans, and work products.

**Value:** Ensures that **customer's needs and expectations are satisfied**

# Practices of Requirements Engineering I

- **Maturity Level 1 - Initial**
  - RDM 1.1 Record requirements.

# CMMI-DEV 2.0:

## *Developing the Customer Requirements*

- Techniques to elicit the requirements:
  - Interview sessions
  - **Prototypes**
  - Brainstorming
  - Market surveys
  - **Use cases**
  - **User Story**
  - ....

# Practices of Requirements Engineering II

- **Maturity Level 2 - Managed**
  - RDM 2.1 Elicit stakeholder needs, expectations, constraints, and interfaces or connections.
  - RDM 2.2 **Transform stakeholder needs, expectations, constraints, and interfaces or connections into prioritized customer requirements.**
  - RDM 2.3 **Develop an understanding with the requirements providers on the meaning of the requirements.**

# CMMI-DEV 2.0:

## Transform the Stakeholder Needs into Customer Requirements

1. Check whether any **information is missing**, and update the missing part
2. Determine whether any **conflicts** exist in the requirements

# Practices of Requirements Engineering III

- **Maturity Level 2 - Managed**
  - RDM 2.4 Obtain commitment from project participants that they can implement the requirements.
  - RDM 2.5 **Develop, record, and maintain bidirectional traceability among requirements** and activities or work products.
  - RDM 2.6 Ensure that plans and activities or work products remain consistent with requirements.

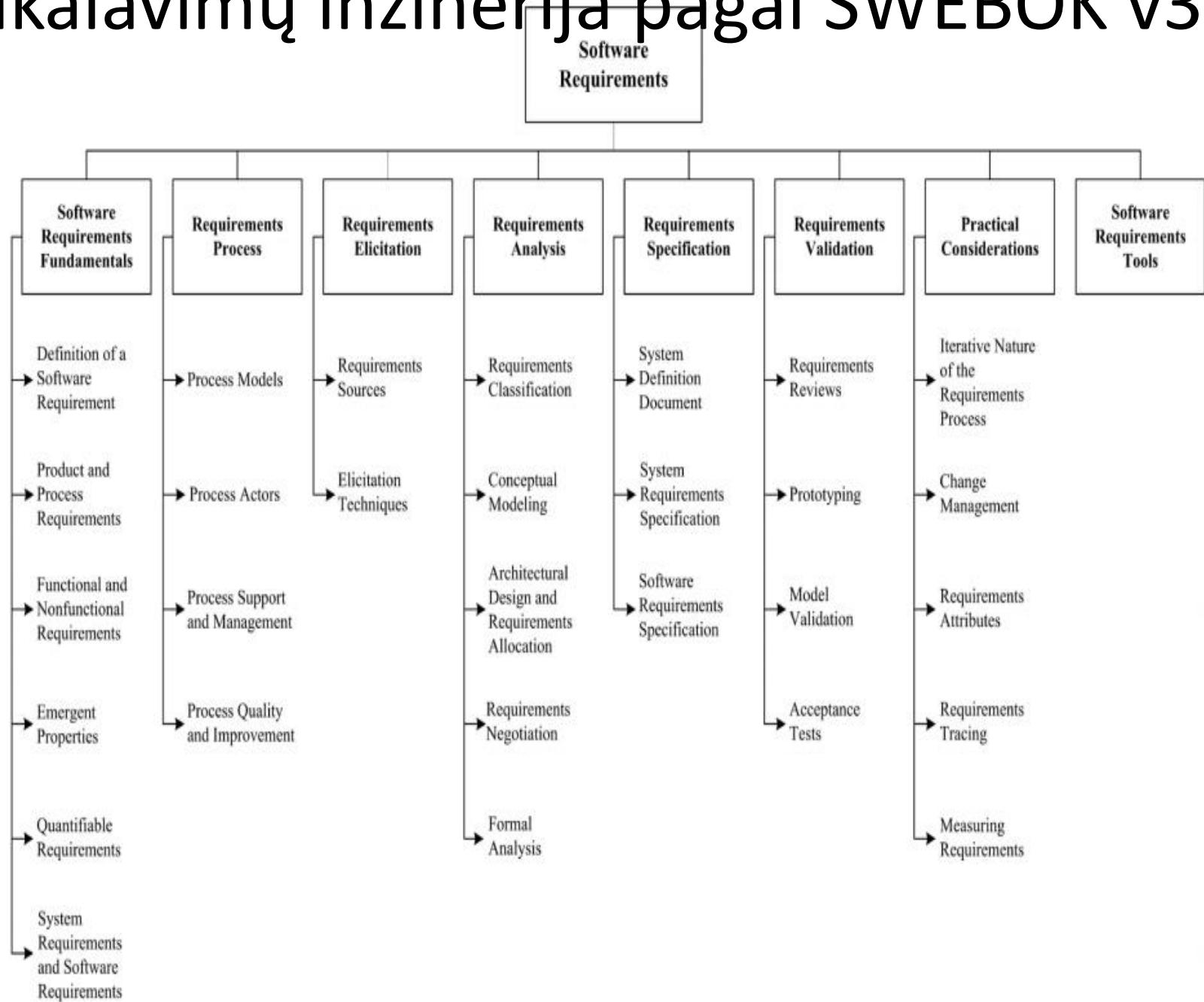
# Practices of Requirements Engineering IV

- **Maturity Level 3 - Defined**
- RDM 3.1 **Develop and keep requirements updated for the solution and its components.**  
RDM 3.2 **Develop operational concepts and scenarios.**  
RDM 3.3 Allocate the requirements to be implemented.

# Practices of Requirements Engineering V

- **Maturity Level 3 - Defined**
  - RDM 3.4 Identify, develop, and keep updated interface or connection requirements.
  - RDM 3.5 Ensure that requirements are necessary and sufficient.
  - RDM 3.6 Balance stakeholder needs and constraints.
  - RDM 3.7 Validate requirements to ensure the resulting solution will perform as intended in the target environment.

# Reikalavimų inžinerija pagal SWEBOK v3



# What is requirement?

- Software requirements express the **needs** and **constraints** placed on a software product that contribute to the solution of some **real-world problem**

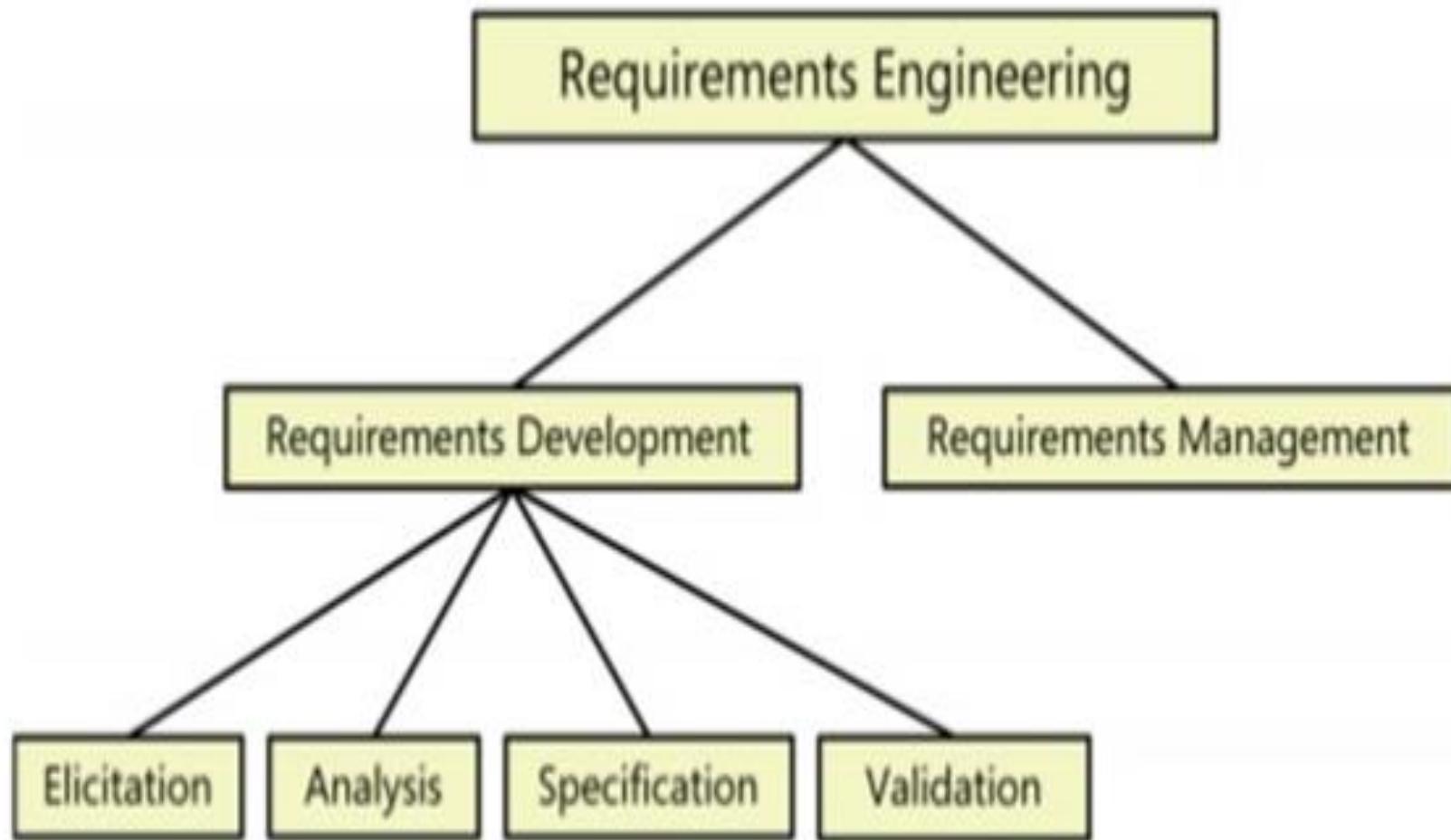


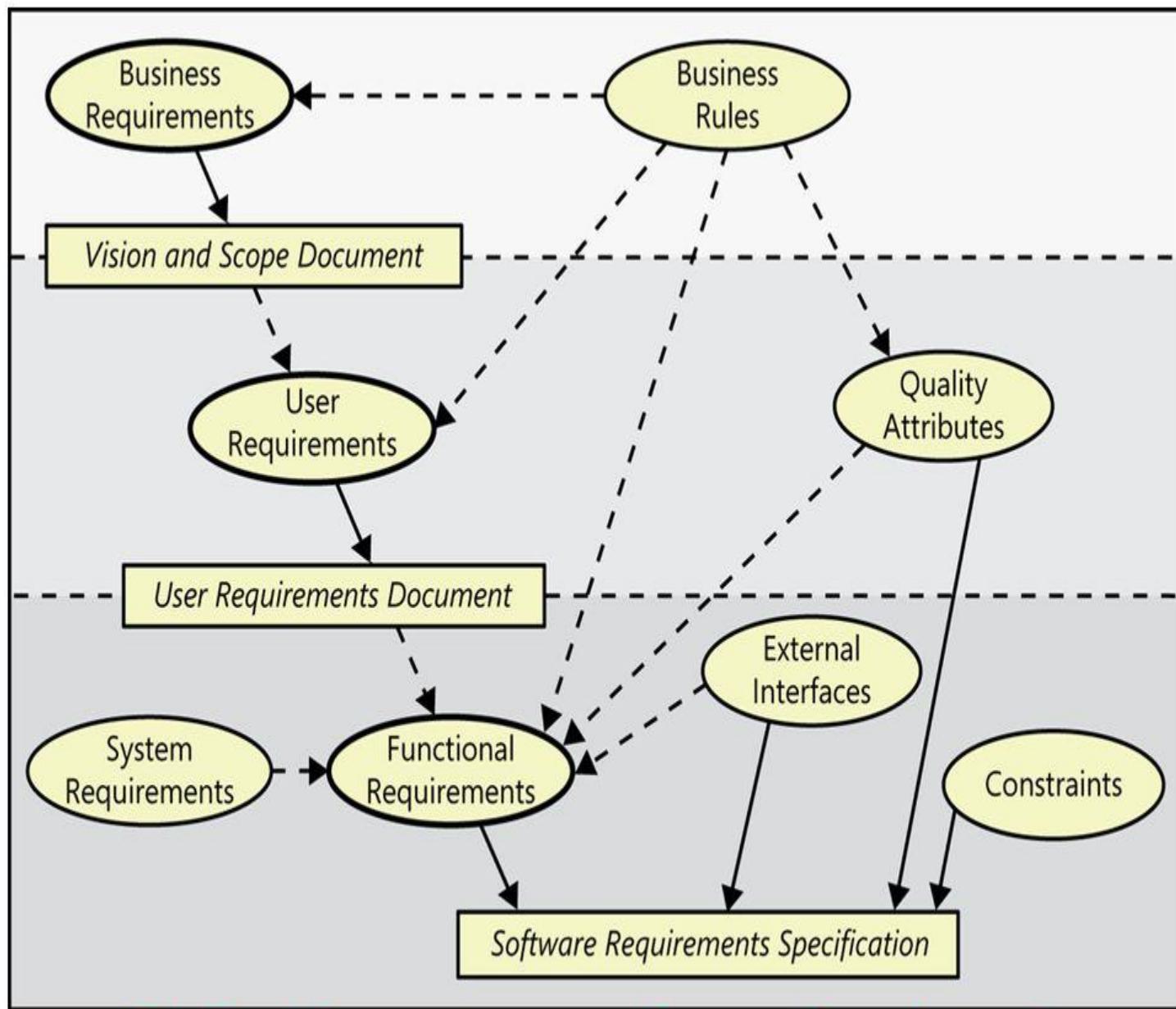
# Definition: Requirement (IEEE)

The IEEE Standard Glossary of Software Engineering Terminology defines a requirement as [IEEE 610.12-1990]:

- (1) A condition or capability needed by a user **to solve a problem or achieve an objective.**
- (2) A condition or capability that must be **met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.**
- (3) **A documented representation of a condition or capability as in (1) or (2).**

# Subset of Requirement Engineering





Relationships among several types of requirements information.

# Why are requirements so difficult to specify?

- Customer's **can't tell** you what they want
  - they don't know
  - they don't clearly articulate their needs
- Customer's have conflicting needs
- Customer's needs change or there understanding of their needs

# Why are Requirements Important?

- Causes of failed software projects:

- Incomplete requirements

- Lack of user involvement

- Lack of resources

- Unrealistic expectations

- Lack of executive support

- Changing requirements & specification

- Lack of planning

- System no longer needed

# Requirement Level Classification

Sommerville (2005) suggests organizing them into three levels of abstraction:

- User requirements
- System requirements
- Design specifications

# CMMI-DEV 2.0

- Three different types:
  - Customer Requirements
  - Product Requirements
  - Product Component Requirements

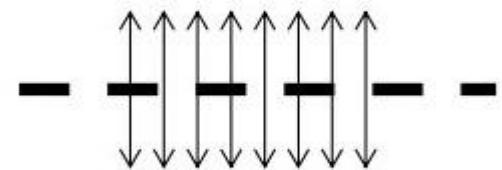
# Definition of UR

- **User requirements** describe goals or tasks the **users must be able to perform** with the product that will provide value to someone

# User Requirement

- User requirements are **abstract statements written in natural language** with accompanying informal diagrams.
- They specify what services (user functionality) the system is expected to provide and any constraints.
- In many situations user stories can play the role of user requirements.

Application  
Domain  
(User Requirements)



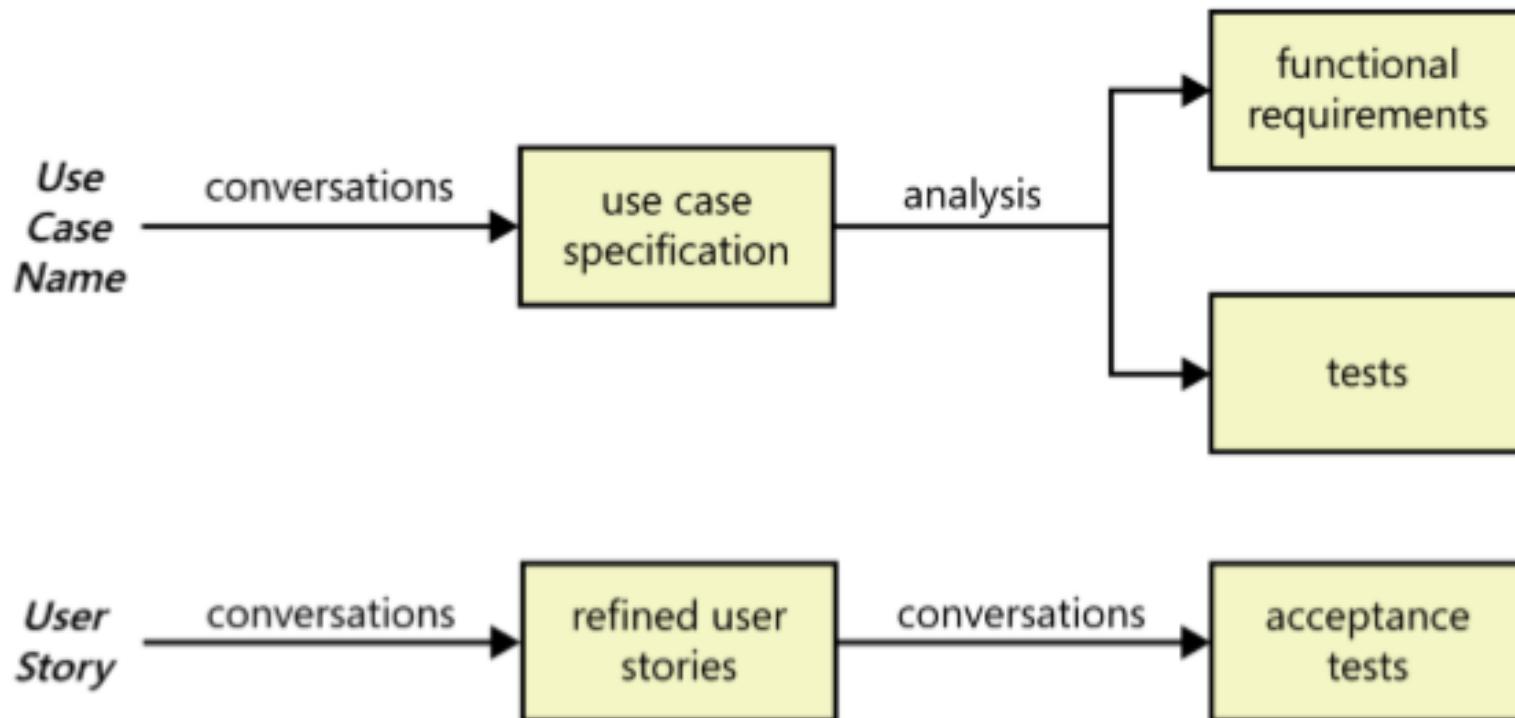
System  
Domain  
(System Requirements)

# User Requirement

- The usage-centered perspective was formalized:
  - into the **use case approach** to requirements modeling (Jacobson et al. 1992; Cockburn 2001; Kulak and Guiney 2004).
  - More recently, proponents of agile development introduced the concept of a “**user story**,” a concise statement that articulates a user need and serves as a starting point for conversations to flesh out the details (Cohn 2004).

# User Requirement:

## *User Story: Use Case*



# Example of use cases and corresponding user stories

- Airport check-in
  - UC: Check in for a Flight
  - US: As a traveler, I want to check in for a flight so that I can fly to my destination
- Online bookstore:
  - UC: Update Customer Profile
  - US: As a customer, I want to update my customer profile so that future purchases are billed to a new credit card number.

# Question of UR

- Both use cases and user stories shift from the **product-centric perspective of requirements** elicitation to discussing:

***What users need to accomplish,***  
***in contrast to asking users,***  
***hat they want the system to do?***

# *Product Requirements*

- A product requirement is a need or constraint on the software to be developed
- Functional requirement:
  - Functional requirements describe the functions that the software is to execute
  - These requirements describe **the business capabilities that the system must provide, as well as its behavior at run-time.**
  - Related to the functionality of the system.
  - Describe system services or actions.
  - **Can be used to distinguish correct output from incorrect.**

# *Product Requirements*

- Non-functional requirements:
  - These requirements describe **the “Quality Attributes” that the system must meet in delivering functional requirements.**
  - These requirements are qualifications of the functional requirements or **of the overall product**
  - Related to how functionality is performed, not whether the end result is correct.

# FR vs. NFR

*Functional requirements are usually well documented and carefully reviewed by the business stakeholders,*

***but Quality Attributes are documented in a much more succinct manner***

**Why?**

**Quality Attributes drive the architecture design**

# FR vs. NFR

## Functional requirements

## Non-functional requirements

### Qualitative requirements

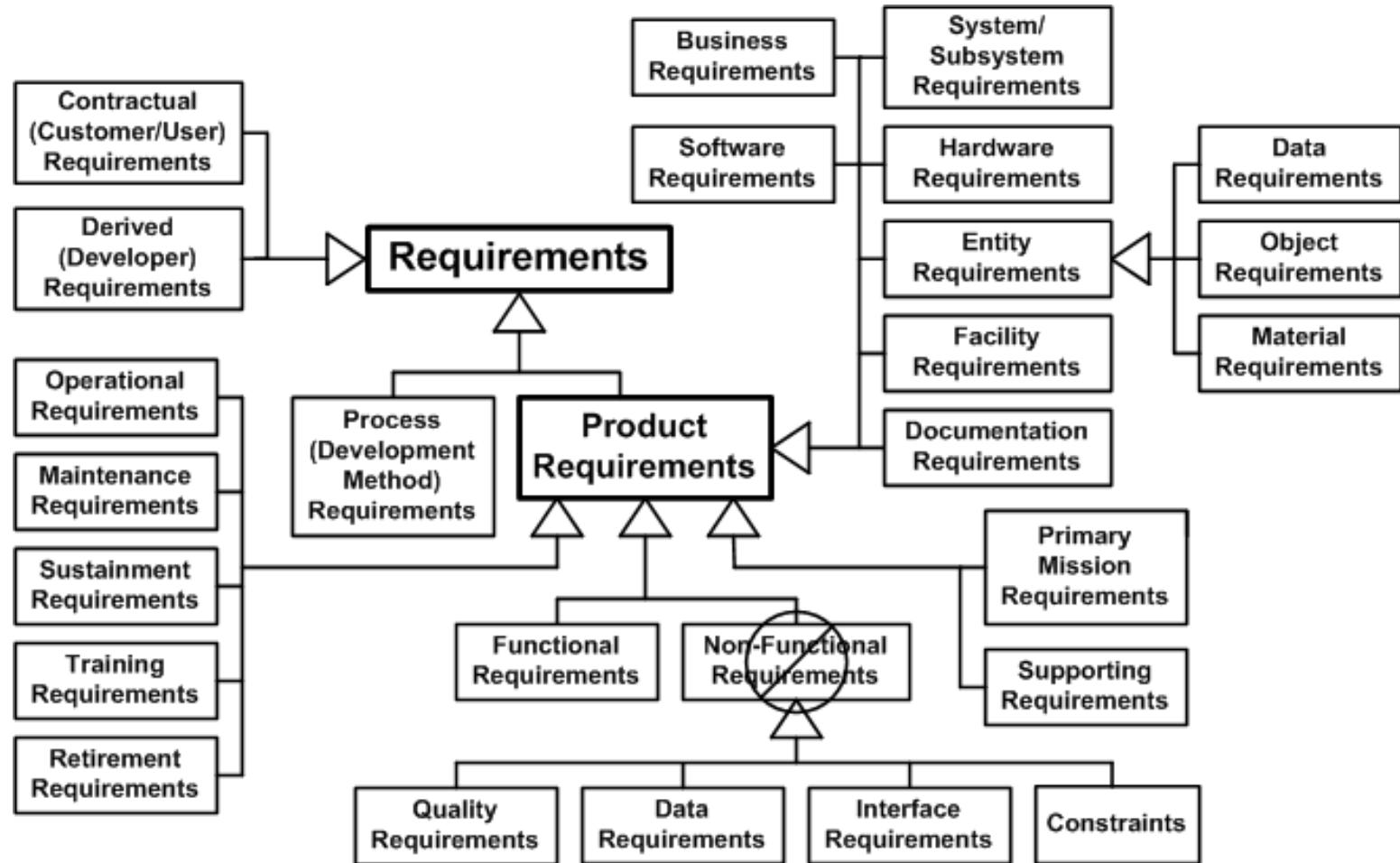
- Performance
- Security
- Reliability
- Usability
- Changeability/Maintainability
- Portability

### Boundary conditions

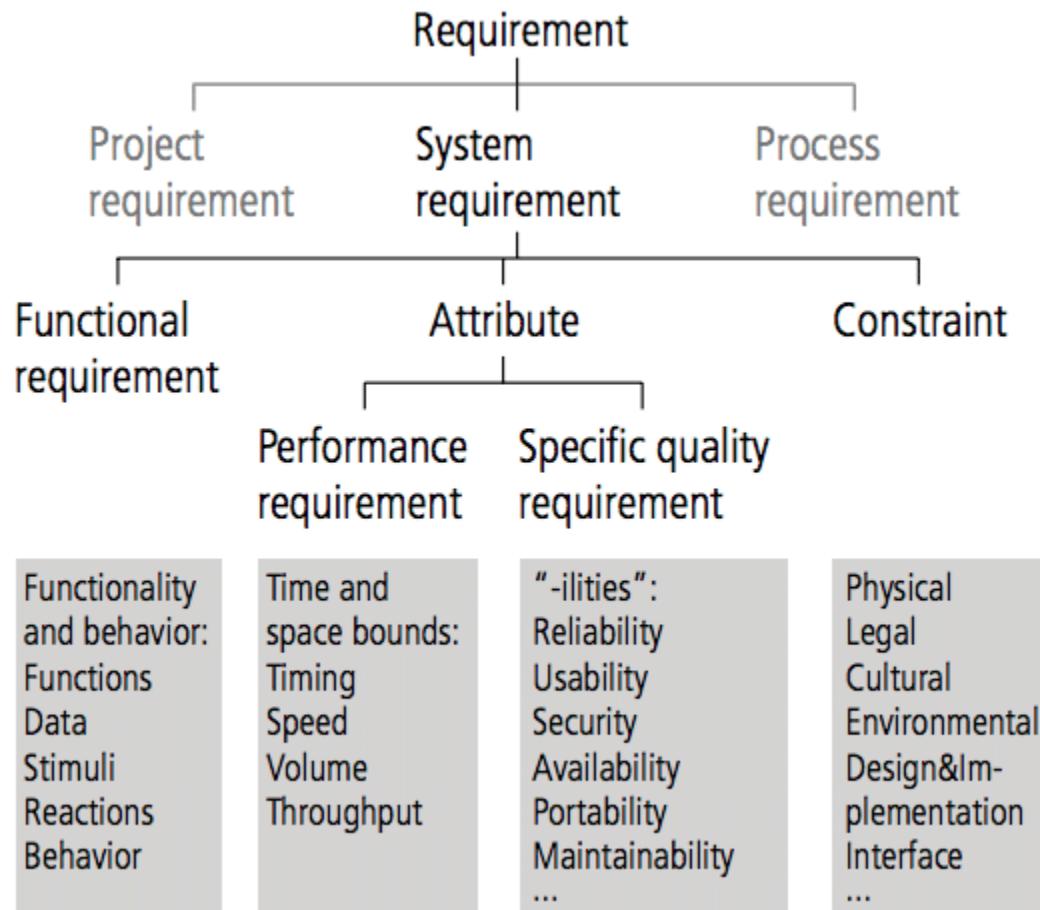
- Laws
- Standards
- Technology
- Process

For quantifying the qualitative requirements, a transformation into functional requirements is necessary

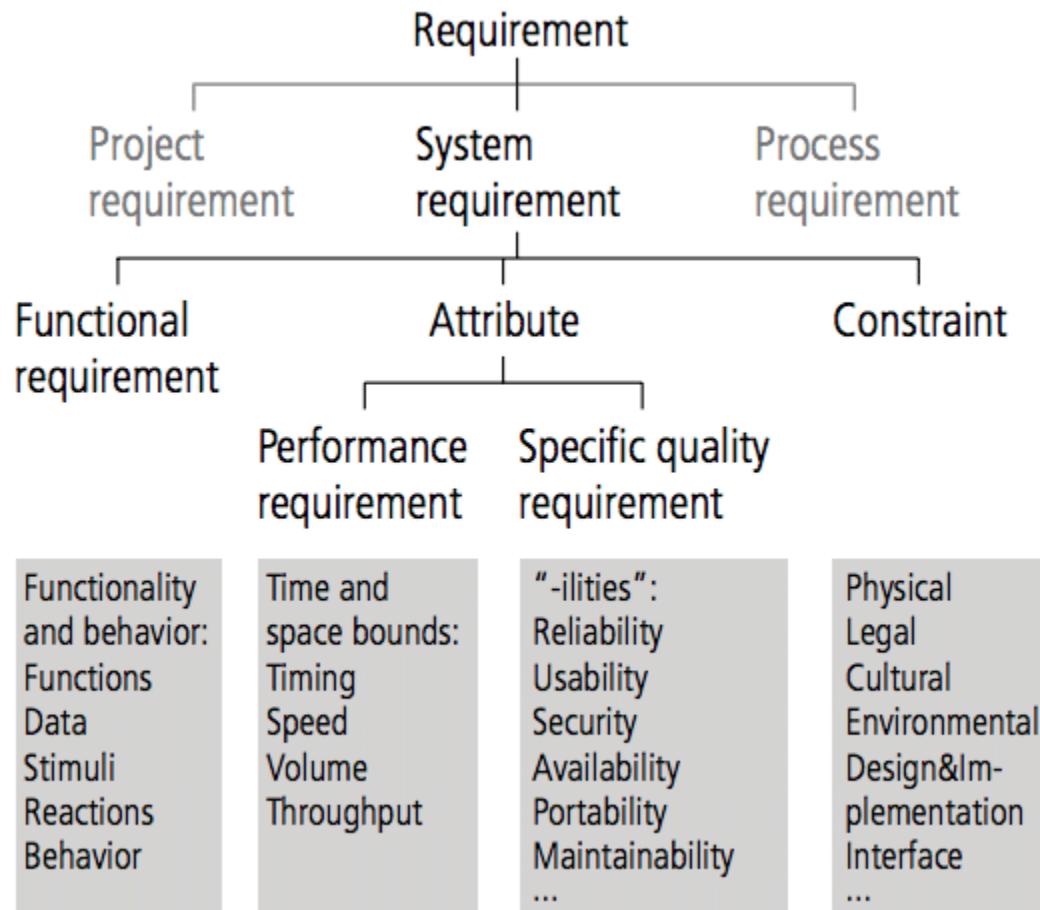
# Requirements taxonomy



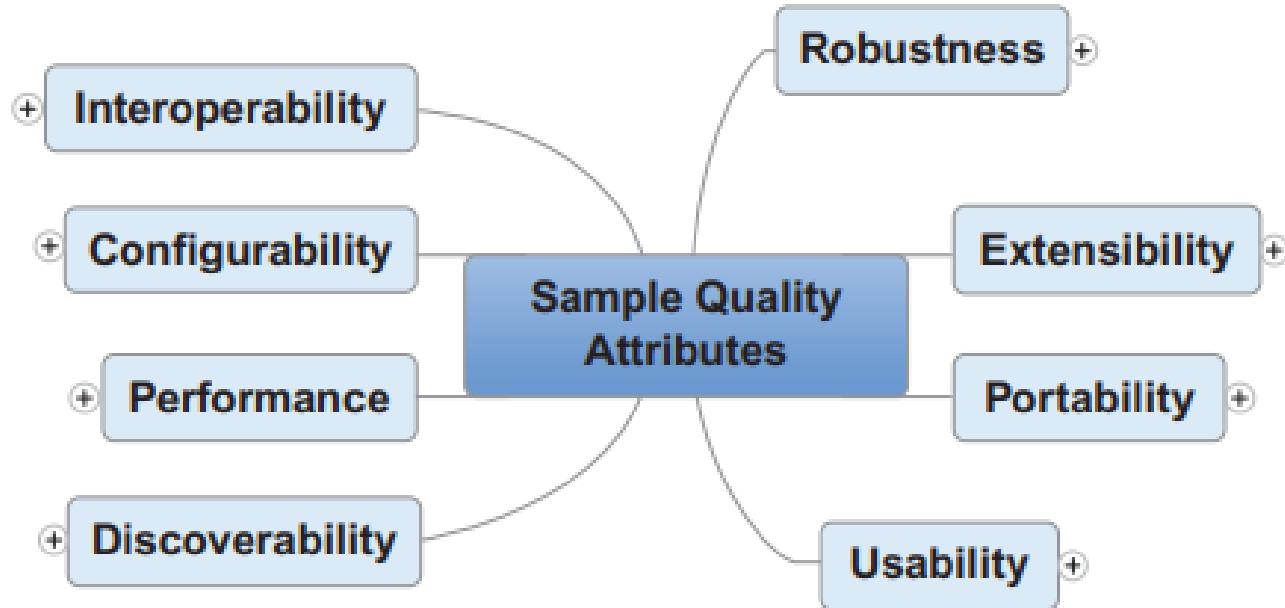
# Requirements taxonomy



# Requirements taxonomy



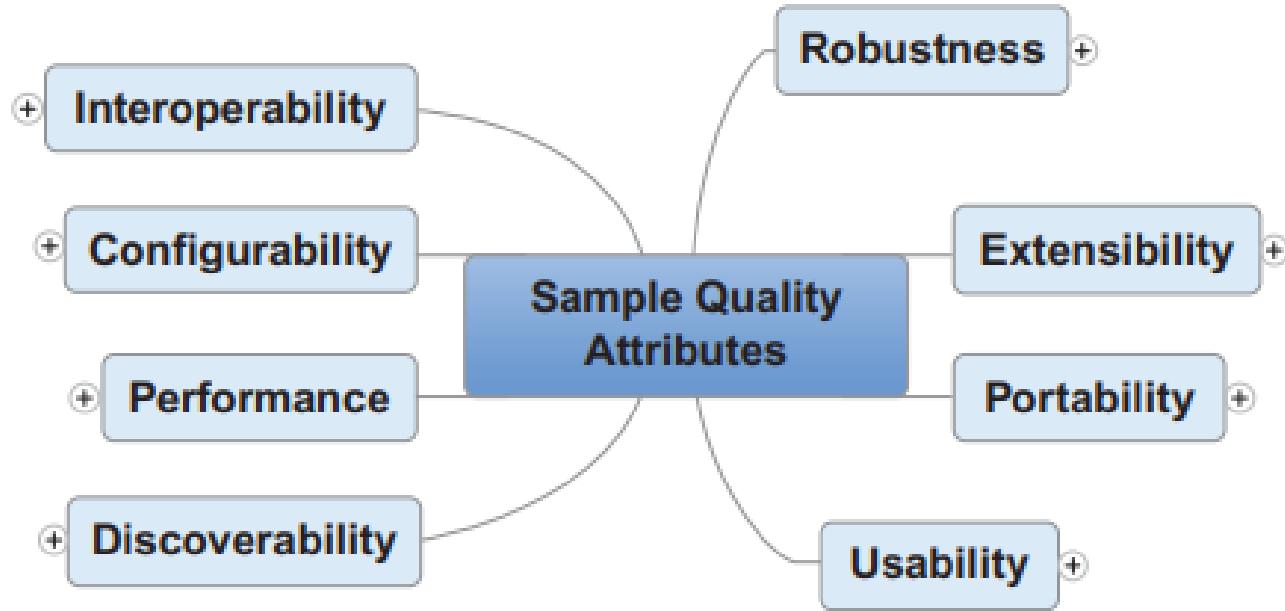
# Quality Attributes



*The system must be extremely user friendly ???*

*The system must be very fast ???*

# Quality Attributes



Functional Requirements define what the system must do  
Quality Attributes define how it does it  
**Clarifying Quality Attributes is important**

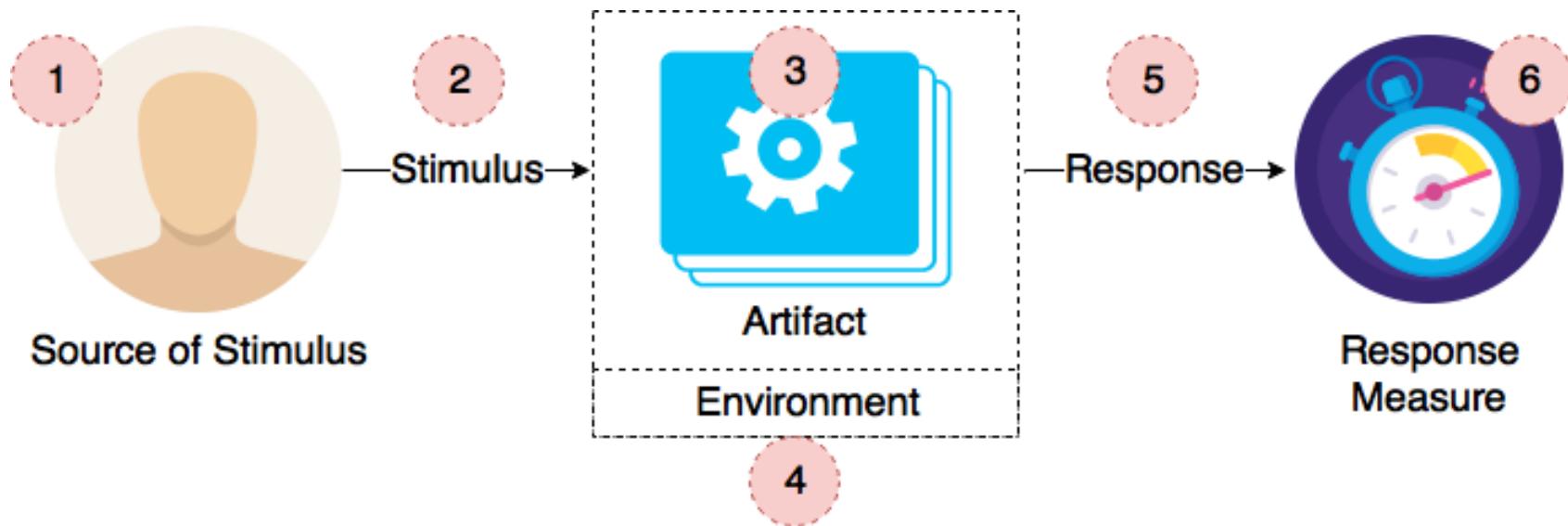
# Quality Attributes

- Operational categories:
  - **Availability**
  - **Interoperability**
  - **Usability**
  - **Performance**
  - **Security**
  - Reliability
  - Deployability
  - Scalability
  - Monitorability
  - Mobility
  - Compatibility
  - Safety

# Quality Attributes

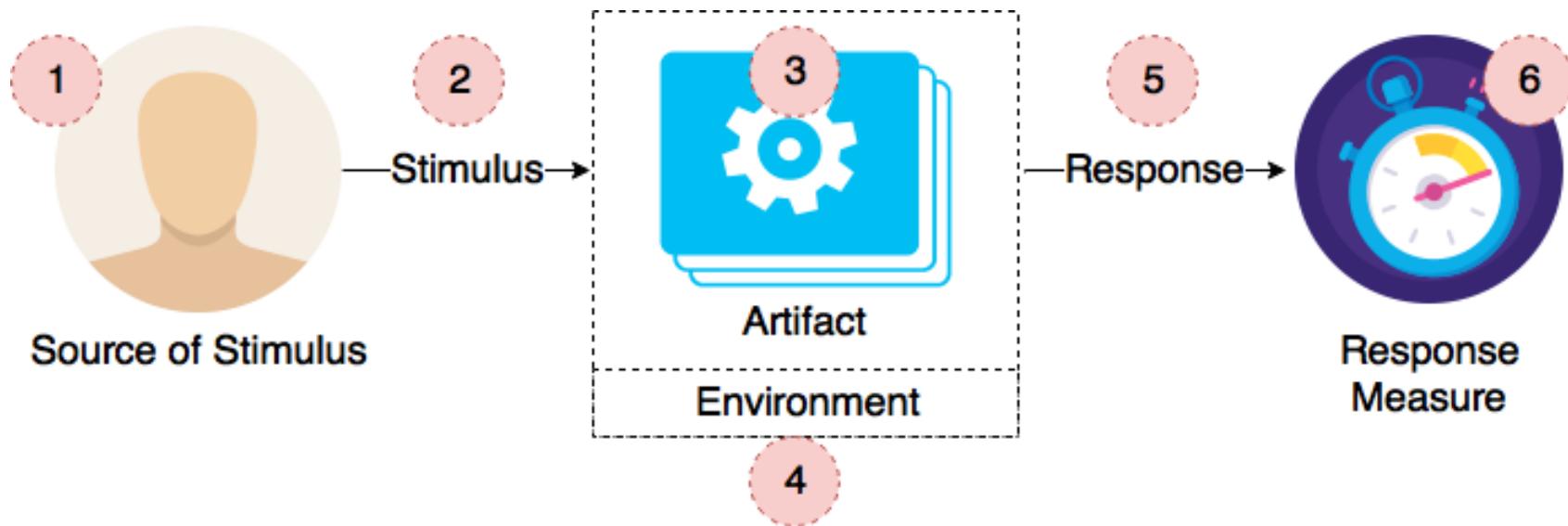
- Development categories:
  - **Modifiability**
  - Variability
  - Supportability
  - **Testability**
  - Maintainability
  - Portability
  - Localizability
  - Development distributability
  - Buildability

# Quality Attribute Scenarios



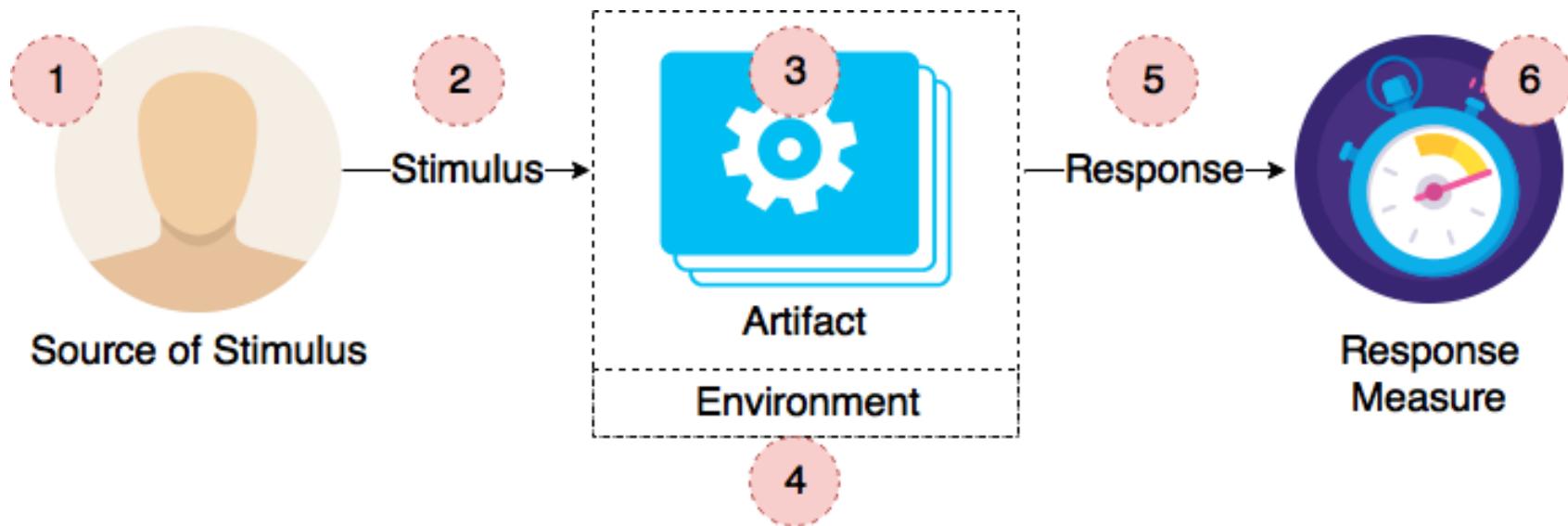
- Source of stimulus
  - An entity capable of creating stimulus (internal or external people, a computer system, etc)

# Quality Attribute Scenarios



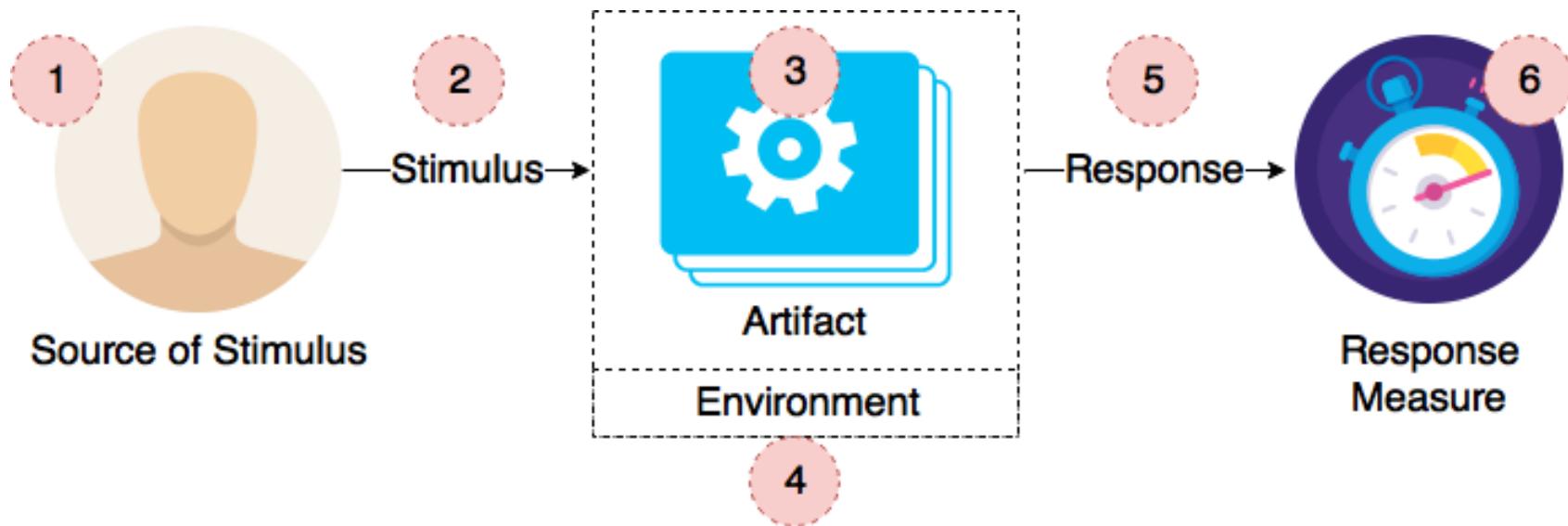
- **Stimulus:**
  - The stimulus is a condition that requires a response when it arrives at a system.

# Quality Attribute Scenarios



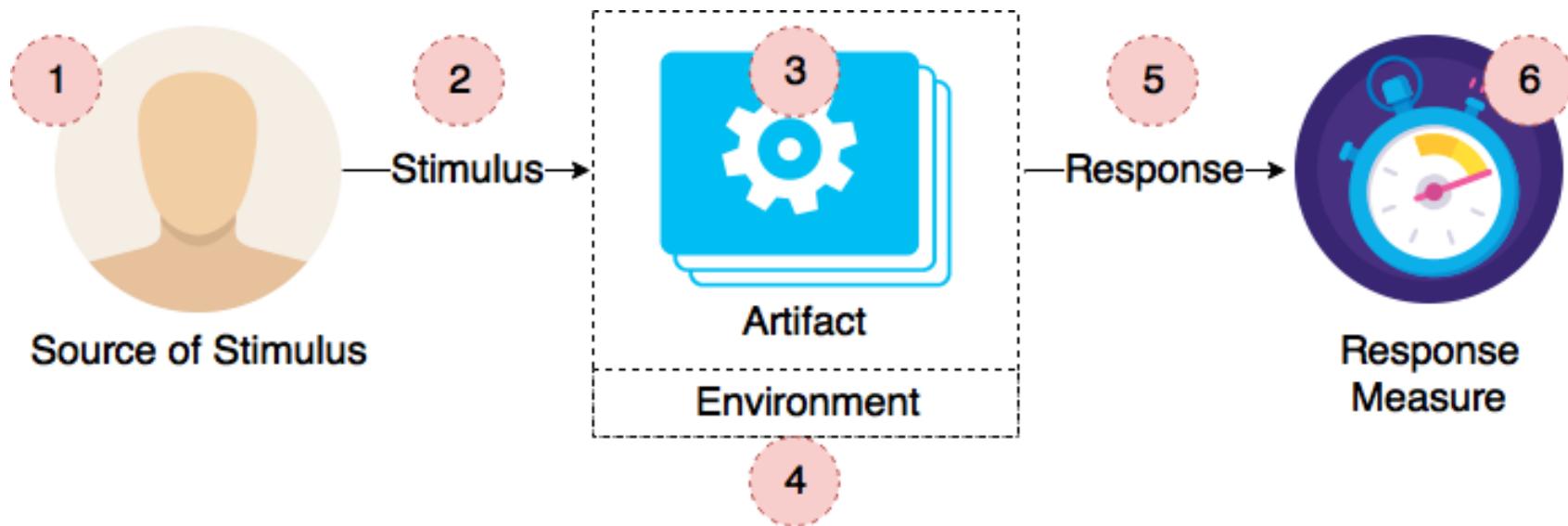
- **Environment:**
  - The environment where the stimulus occurs.
  - For instance, the system may be running in normal conditions, under heavy traffic, or with a high latency or any relevant state.

# Quality Attribute Scenarios



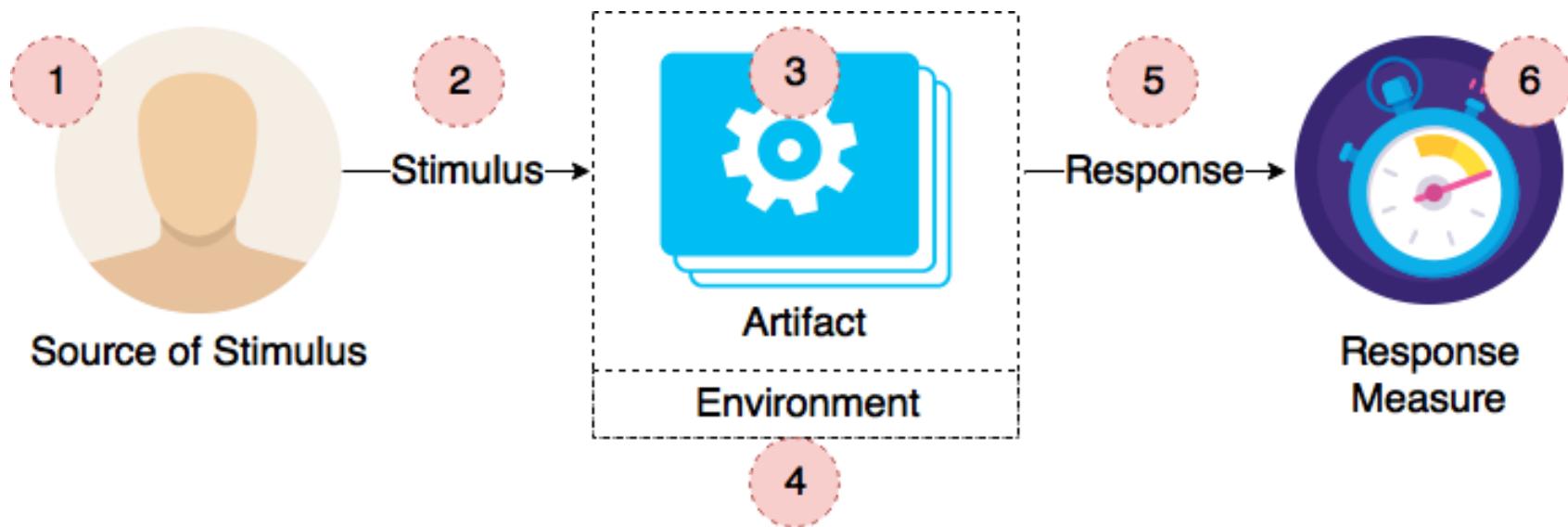
- **Artifact:**
  - The artifact that receives the stimulus.
  - This can be a component of the system, the whole system, or several systems.

# Quality Attribute Scenarios



- **Response:**
  - The is the response of the artifact according to the received stimulus.

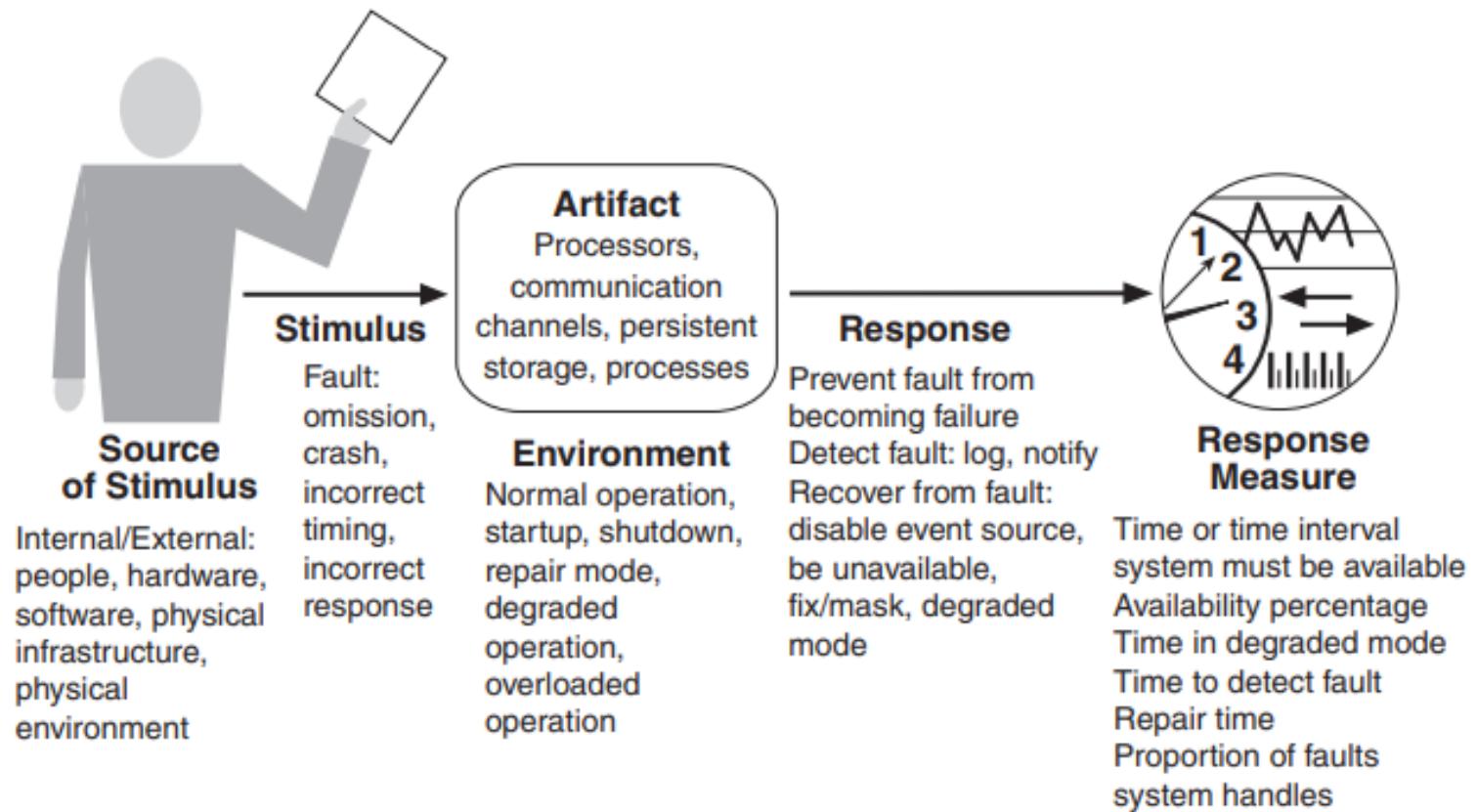
# Quality Attribute Scenarios



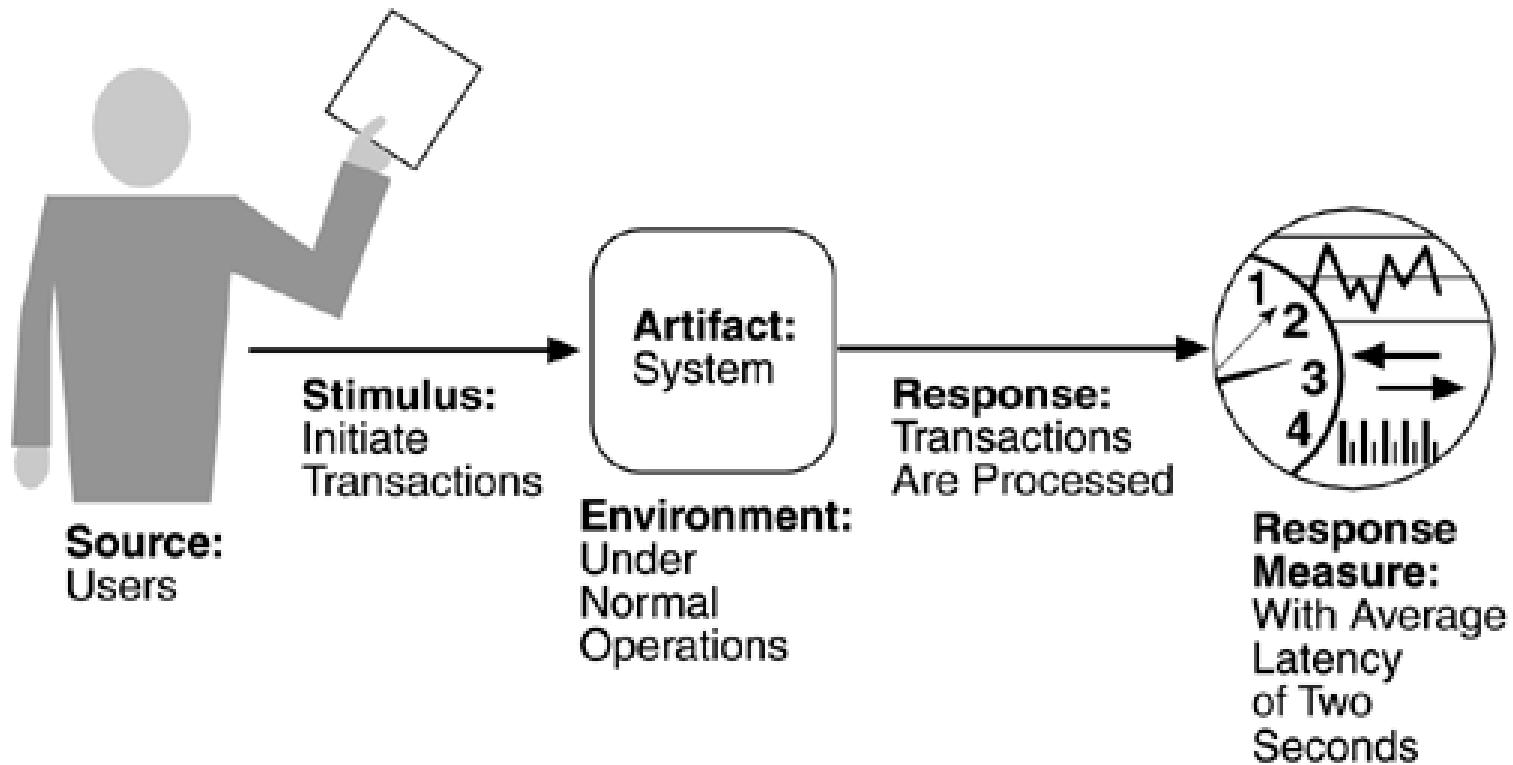
- **Response Measure:**
  - This is the measure that should be tested for the response to test if the requirement is well implemented.

# A general scenario for availability

- Software Architecture in Practice Third Edition



# Quality Attribute Scenarios: performance

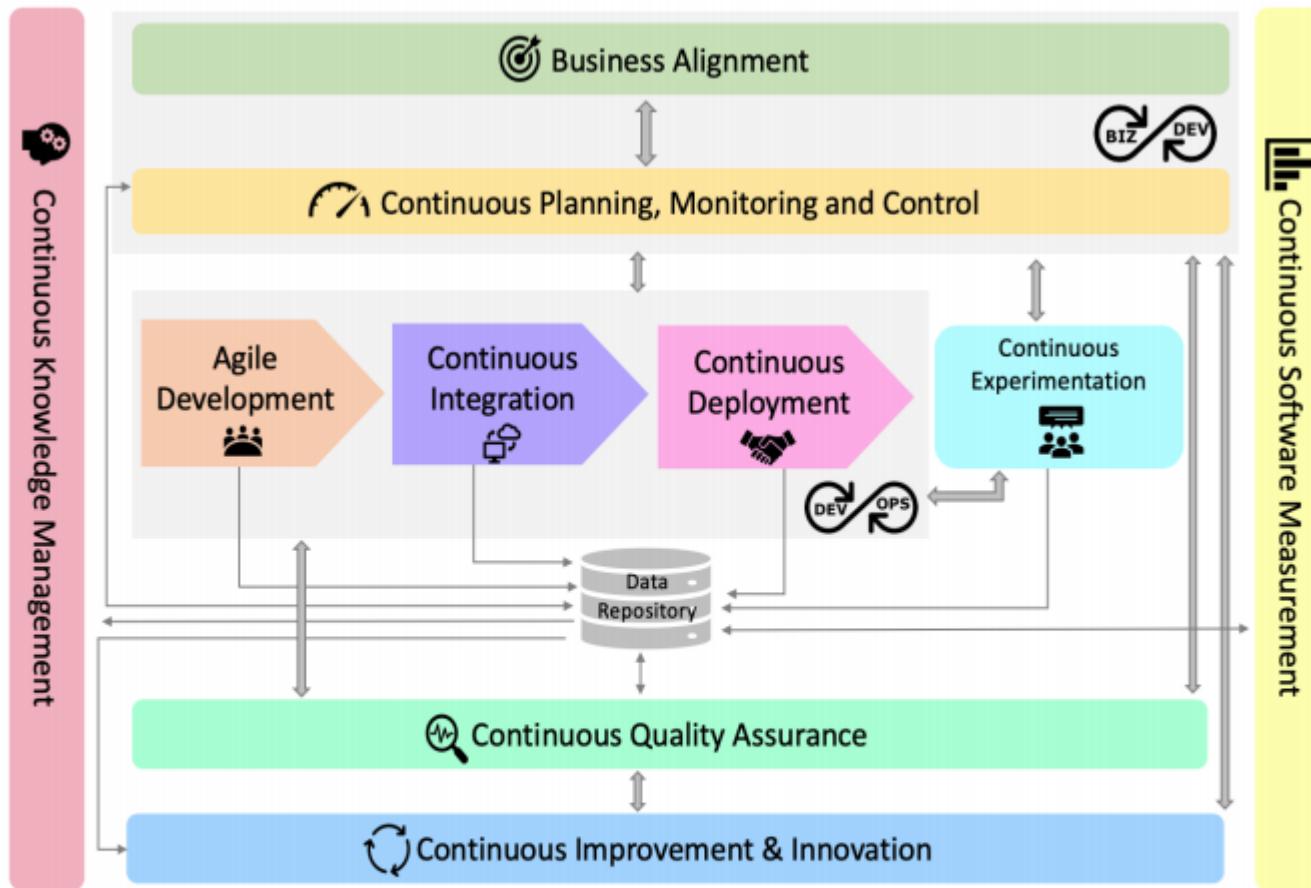


<http://www.ece.ubc.ca/~matei/EECE417/BASS/ch04lev1sec4.html>

<https://mperry.github.io/2012/02/15/quality-attributes.html>

<https://www.cs.unb.ca/~wdu/cs6075w10/sa2.htm>

# Framework for Continuous Software Engineering



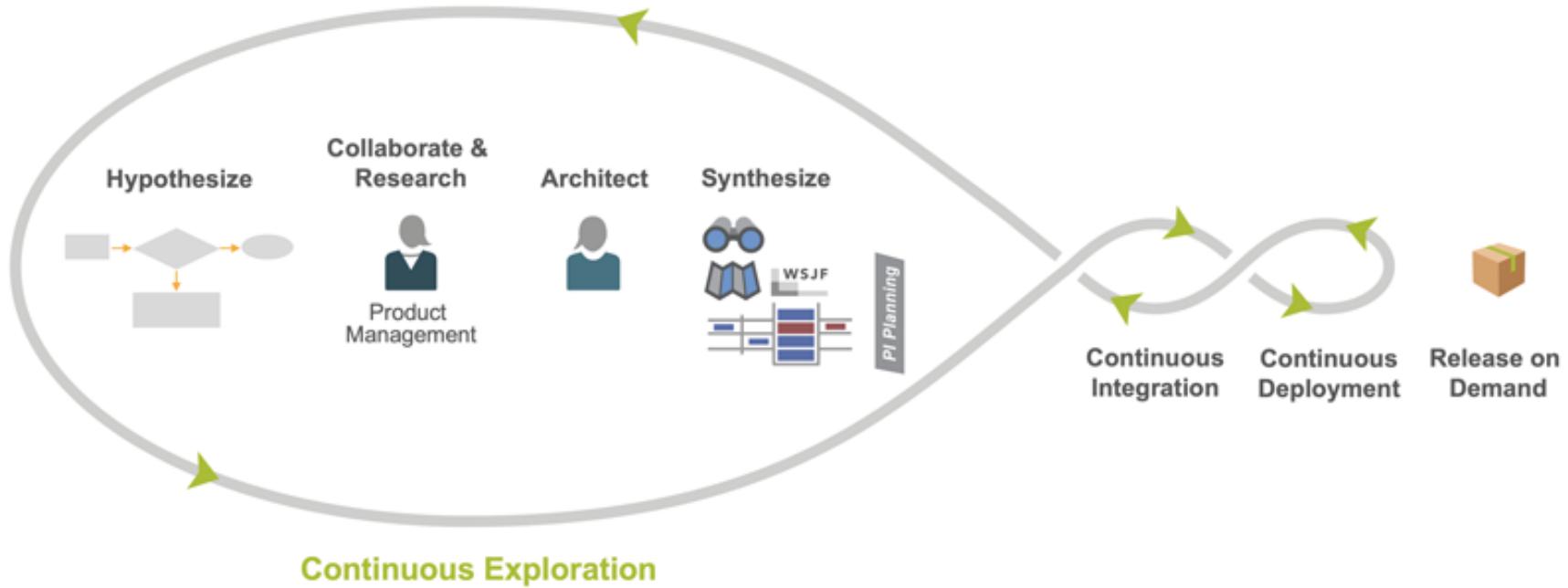
# Framework for Continuous Software Engineering

- CSE involves practices and tools that aims at establishing an end-to-end flow between customer demand and the fast delivery of a product or service
- **Continuous Software Engineering (CSE)** consists of a set of practices and tools that support a holistic view of software development with the purpose of making it **faster, iterative, integrated, continuous and aligned with business**.
- It understands that the software development process is **not a sequence of discrete activities**, performed by distinct and disconnected teams. It aims to establish a continuous flow between software-related activities, taking into consideration the entire software life cycle.

# BizDevOps

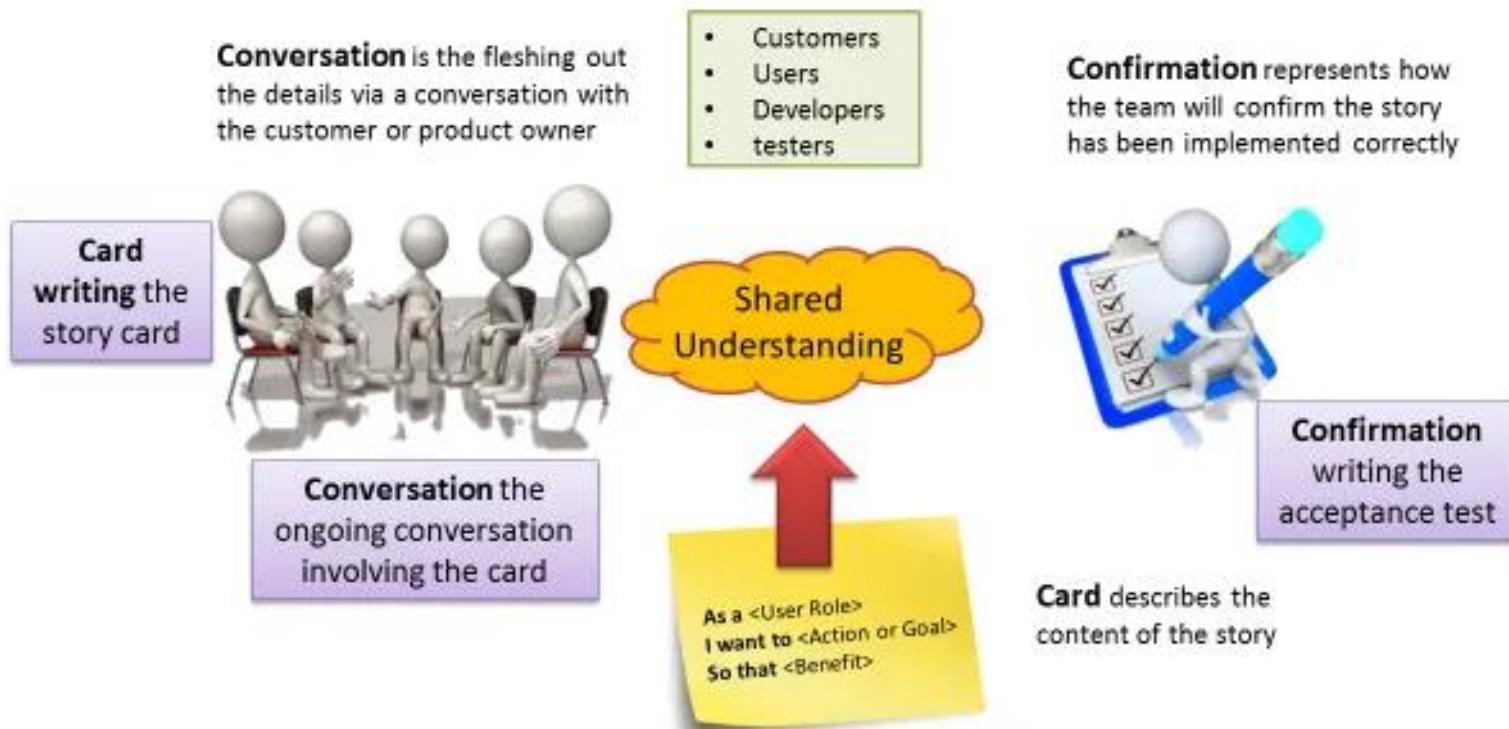


# Continuous Exploration

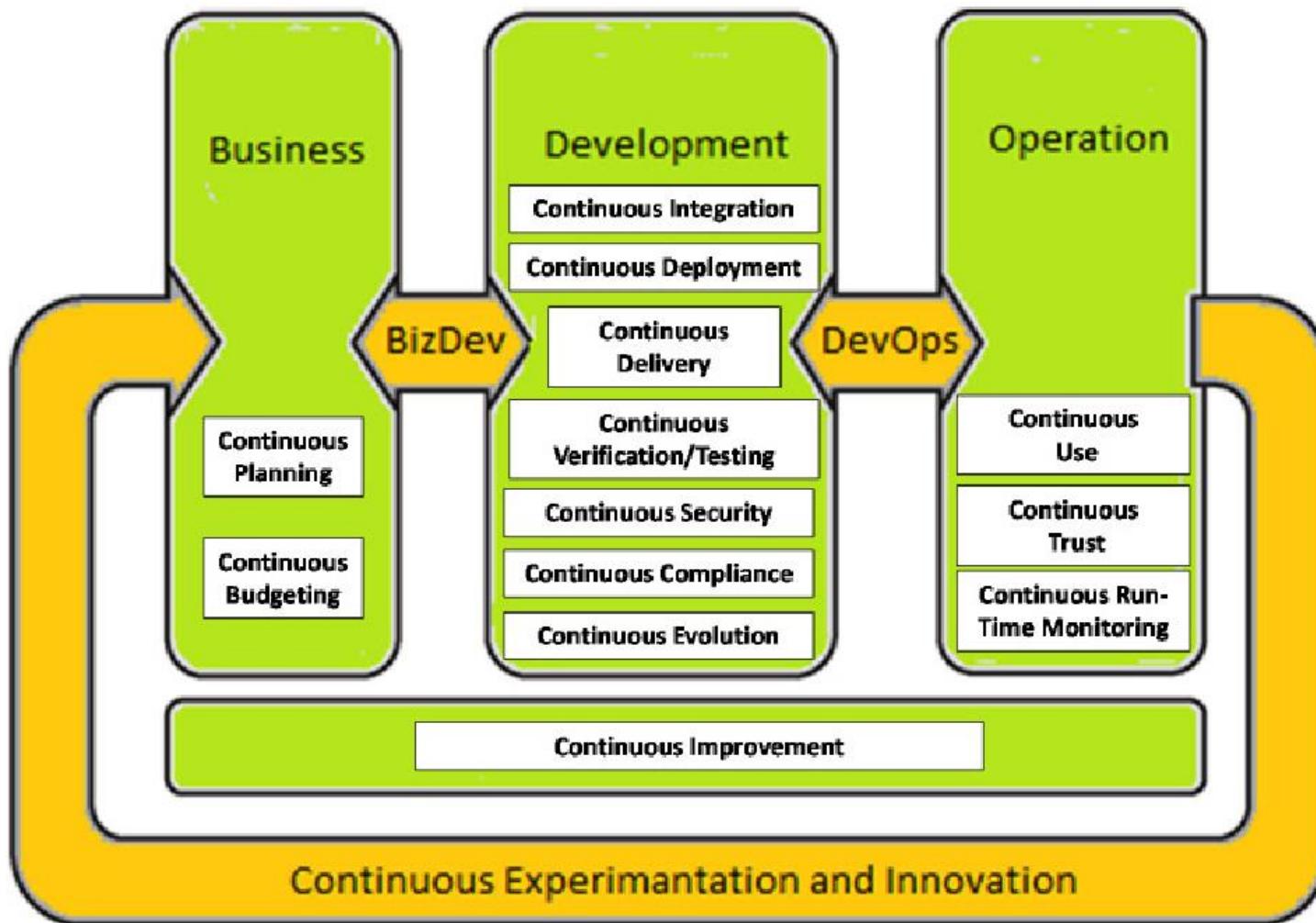


© Scaled Agile, Inc.

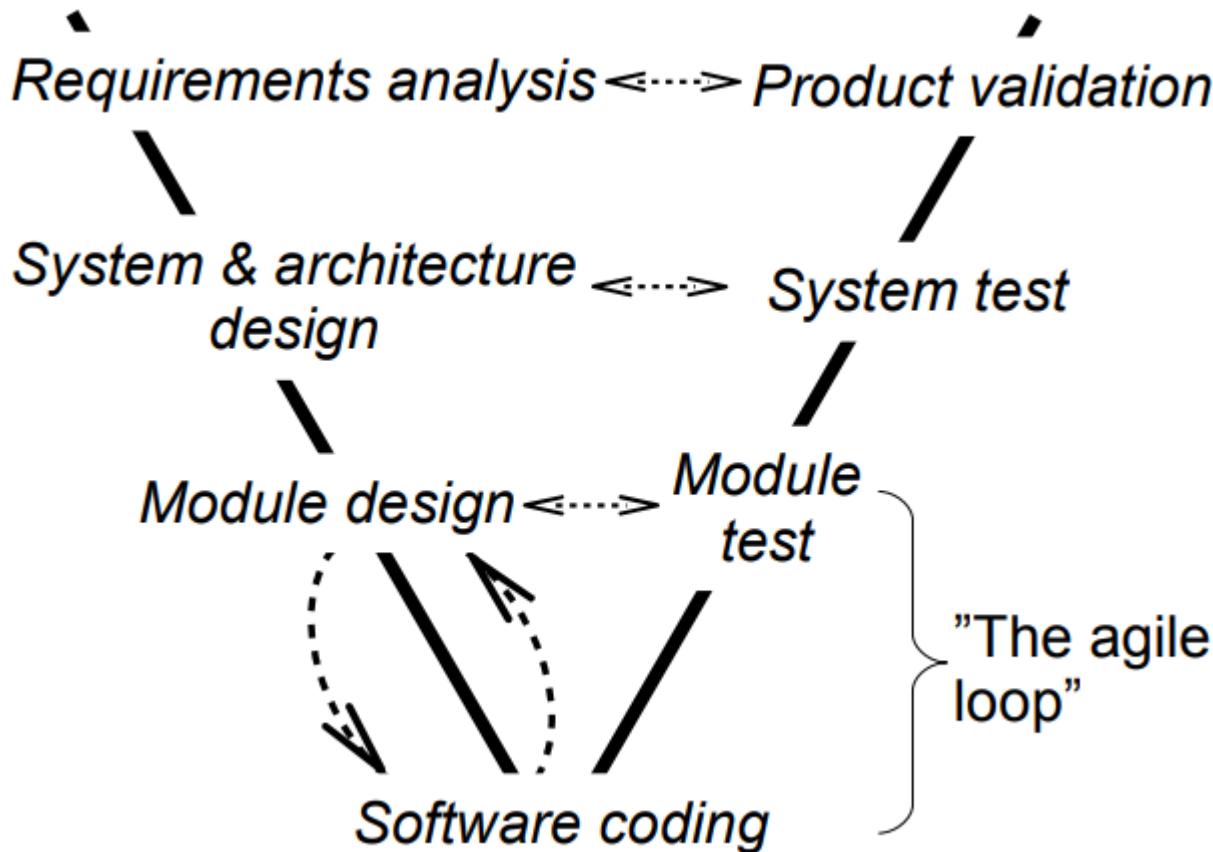
# Continues Requirements Engineering



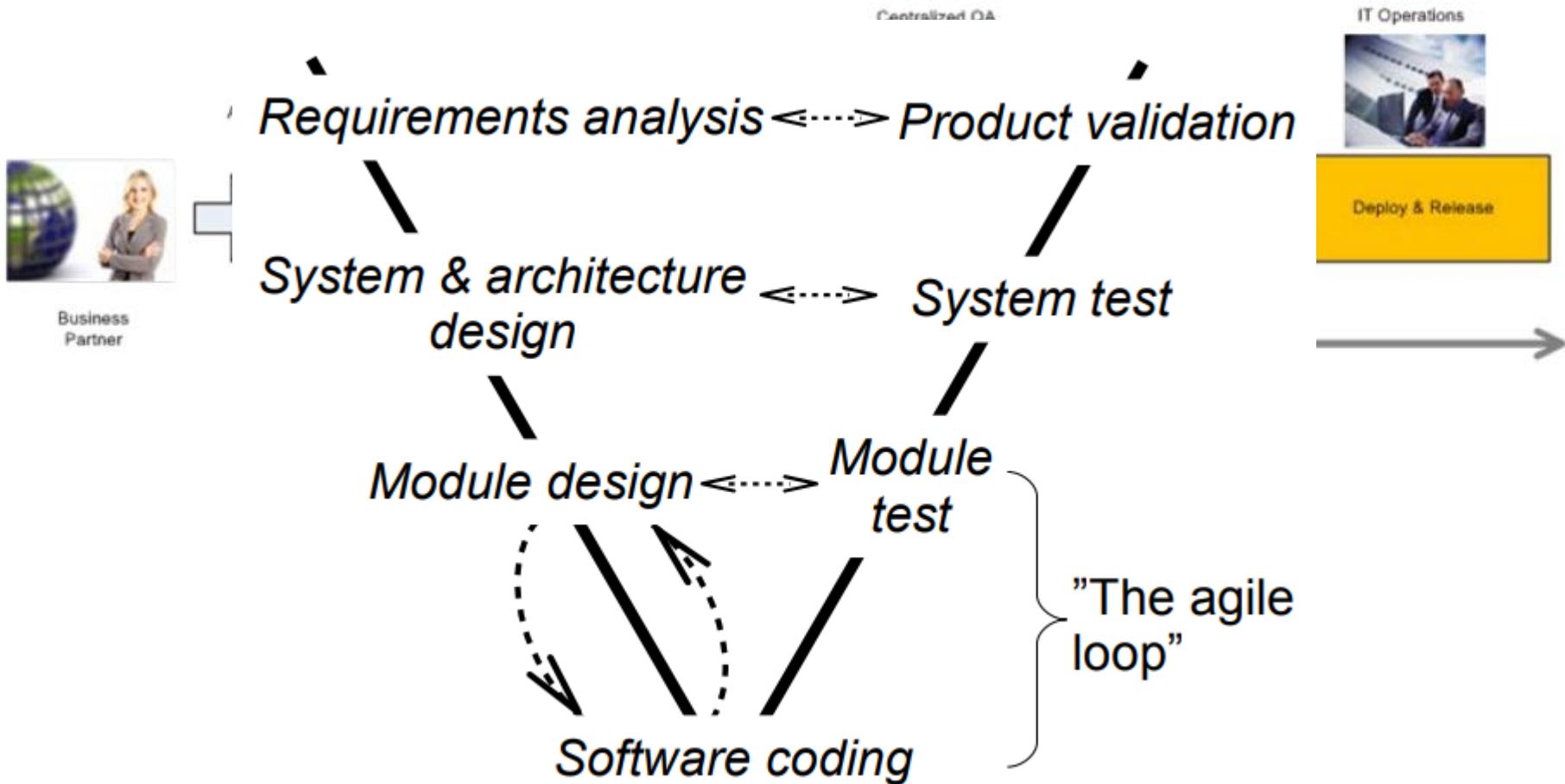
# Continuous Software Engineering



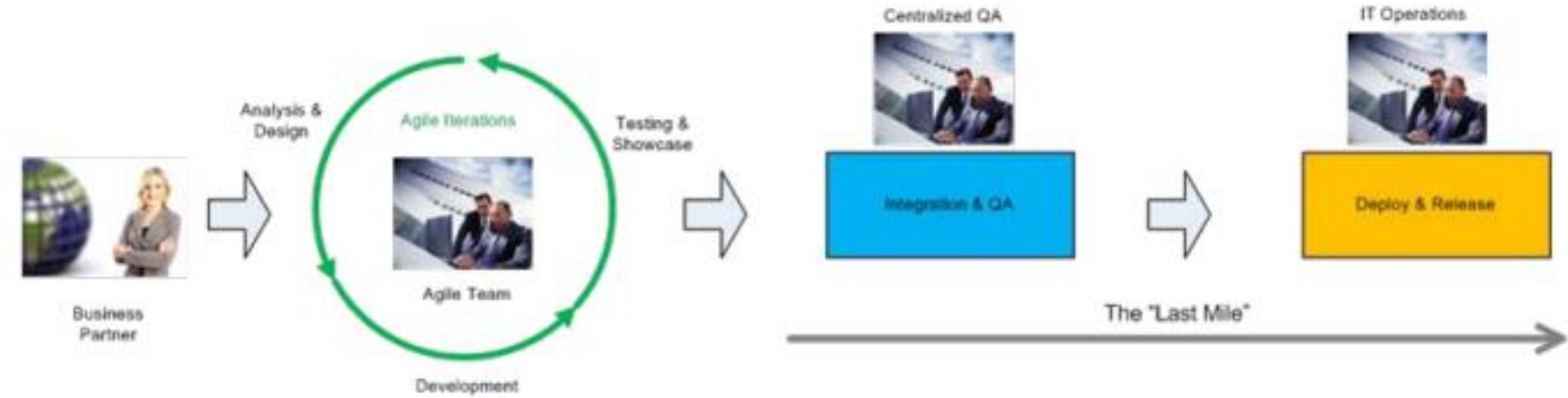
# Software development in practice...



# From this...



# From this



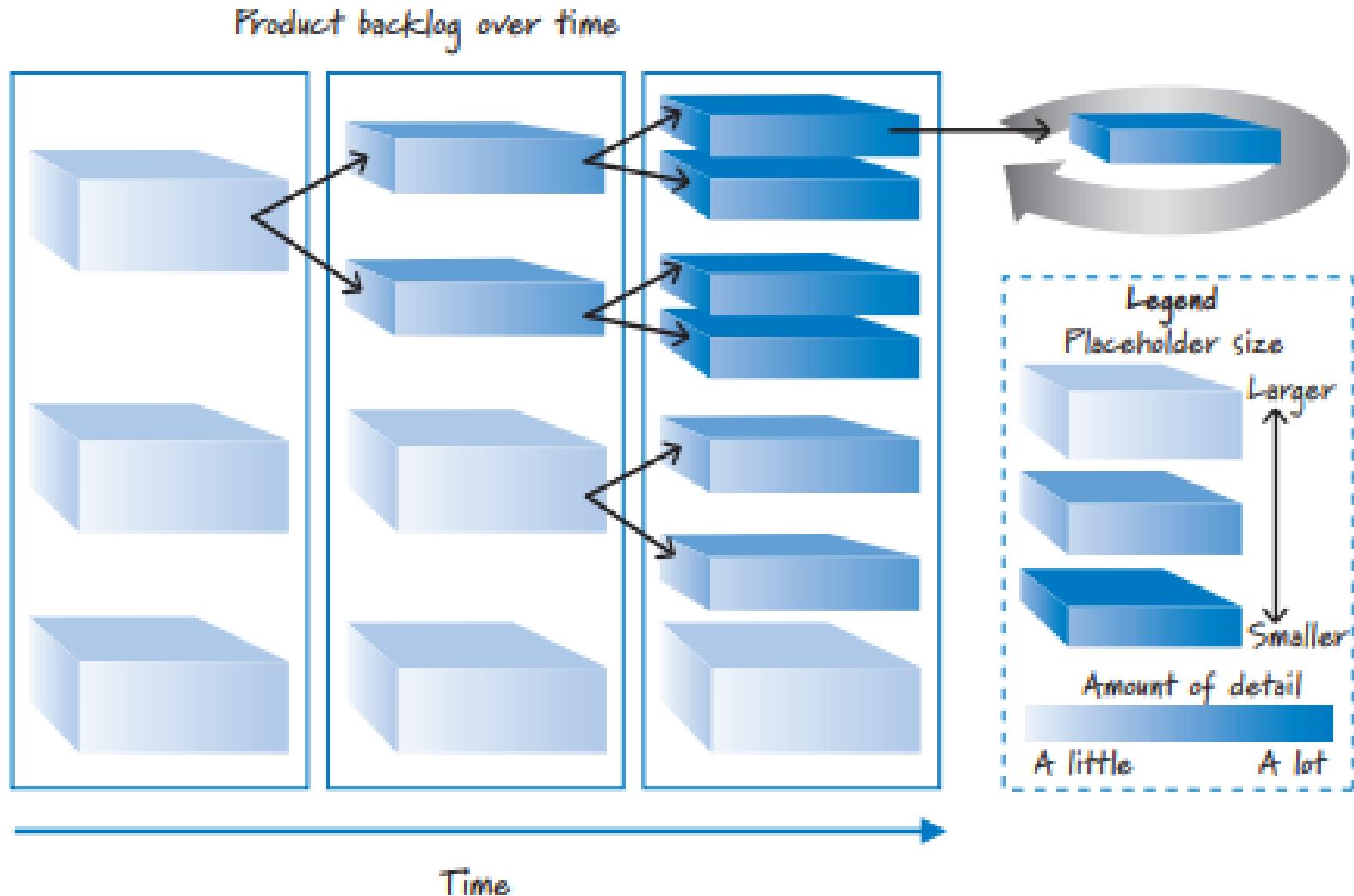
# To this



# Produkto reikalavimų inžinerijos procesas: agile RI

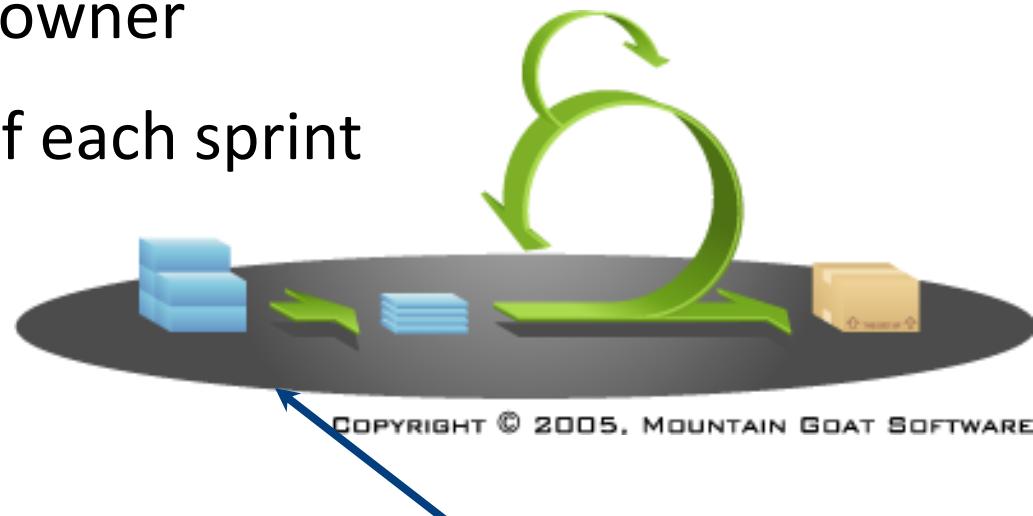
Dr. Asta Slotkienė

# Requirements hierarchy



# Product backlog

- The requirements
- A list of all desired work on the project
- Ideally expressed such that each item has value to the users or customers of the product
- Prioritized by the product owner
- Reprioritized at the start of each sprint



This is the  
product backlog

# Definition of Product backlog

- **The product backlog is a prioritized list of desired product functionality.**
- It provides a centralized and shared **understanding of what to build and the order in which to build it**

# Definition of Product backlog

- The product backlog is a master list of all desired functionality for the system under development
- A living document – stories are added, changed, and/or removed throughout project duration

# Definition of Product backlog

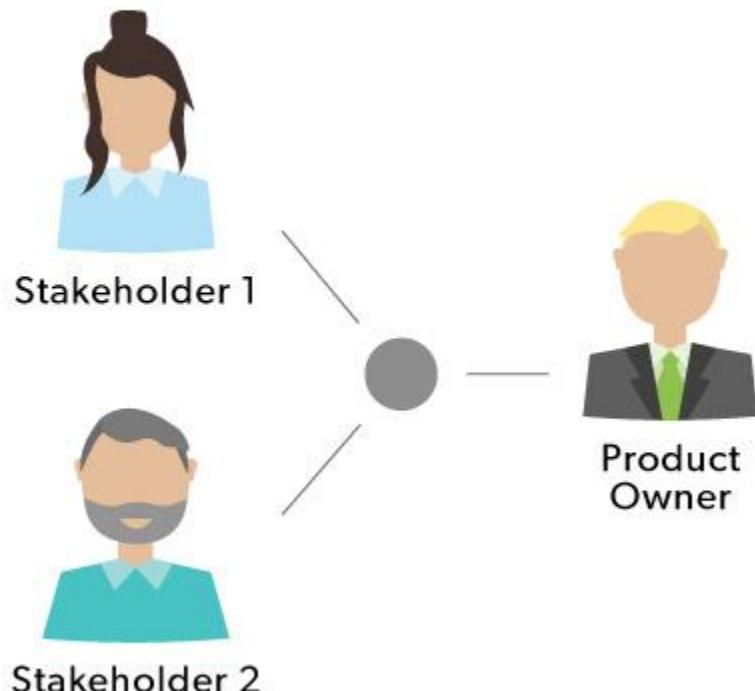
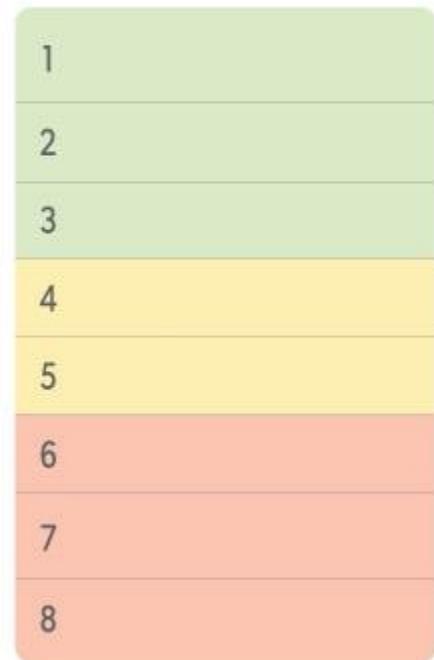
- Essentially the requirements, captured as a list of **user stories**
  - User stories are prioritized and assigned to sprints progressively
  - Stories are reprioritized at the start of each iteration

# Product backlog

Wish Lists

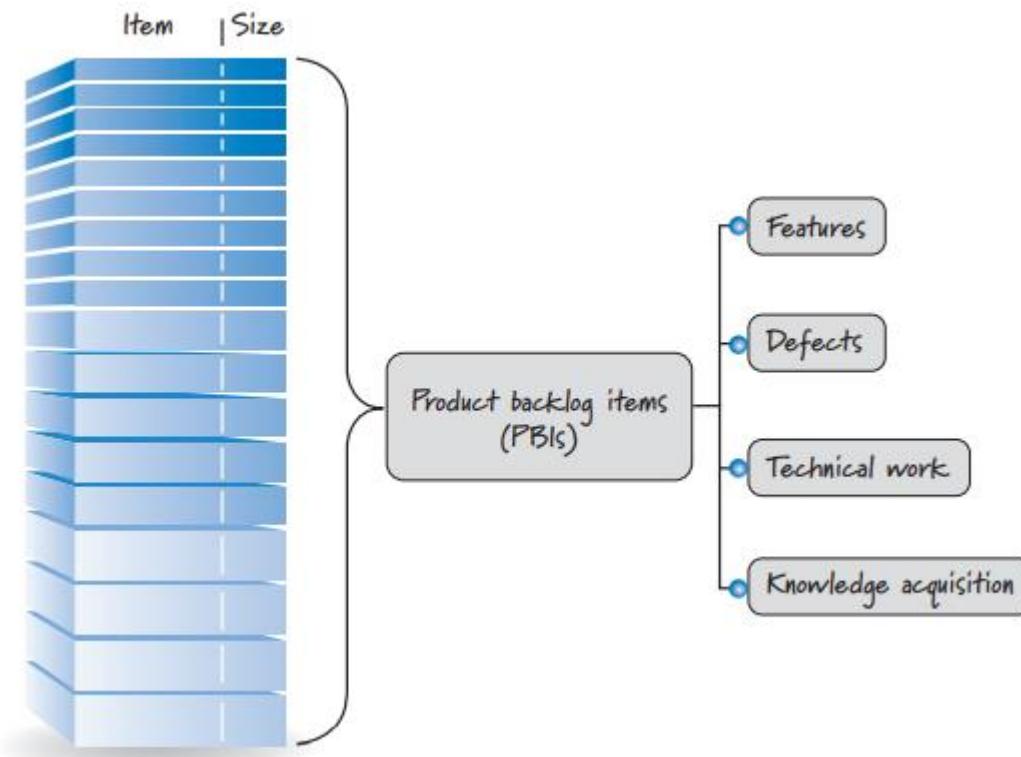


Product Backlog



# Product backlog items

- The product backlog is composed of backlog items, which I refer to as PBIs
- Most PBIs are features, items of functionality that will have **tangible value to the user or customer.**



# Version of product backlog: Goal

## ▼ Vision

### ▼ Goal 1

#### ▼ Feature 1 / Epic 1

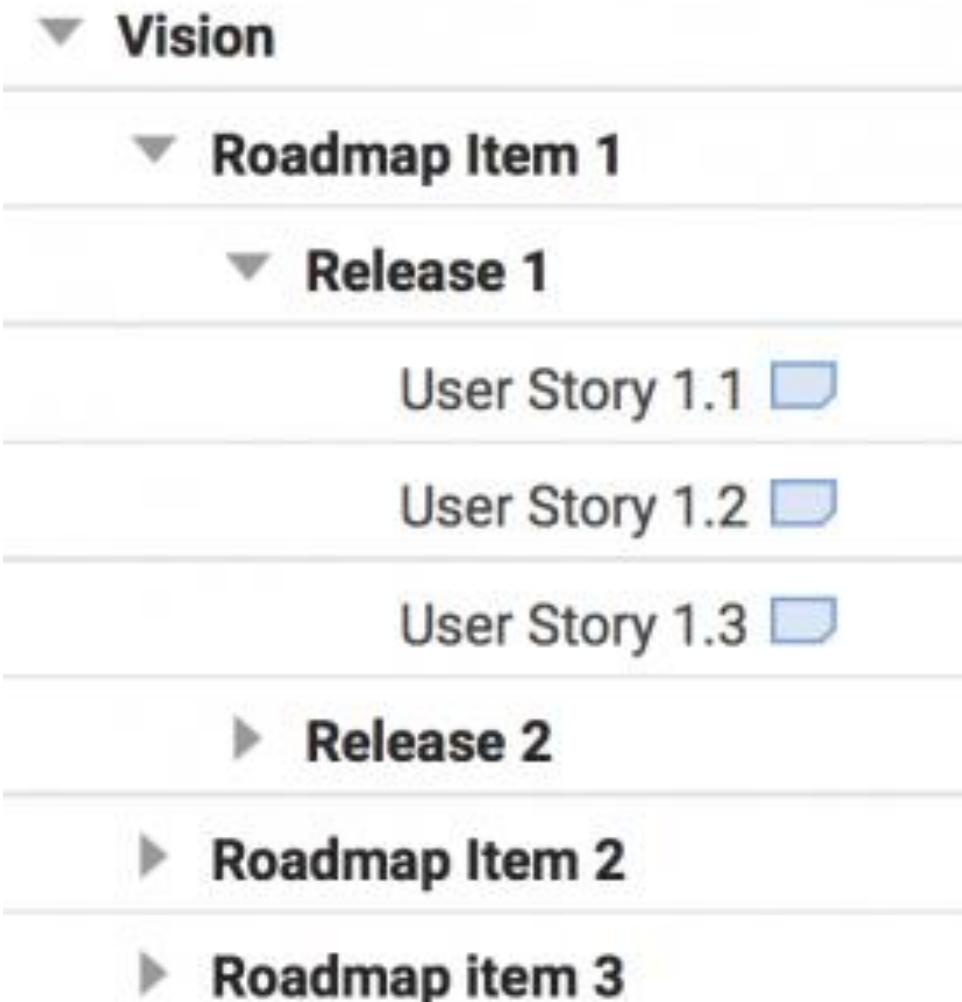
User story 1 

User story 2 

#### ► Feature 2 / Epic 2

#### ► Goal 2

# Version of product backlog: Planning



# Version of product backlog: component

- ▼ **Product 1**

- ▼ **Component 1**

- ▼ **Sub-system 1.1**

- Change 1

- Change 2

- **Sub-system 2.1**

- **Component 2**

- **Product 2**

# Version of product backlog: Team

▼ **Area 1** Delegated to: Product Owner

▼ **Team Yellow's Backlog** Delegated to: Team Yellow

User Story 1 

User Story 2 

User Story 3 

► **Team Blue's Backlog** Delegated to: Team Blue

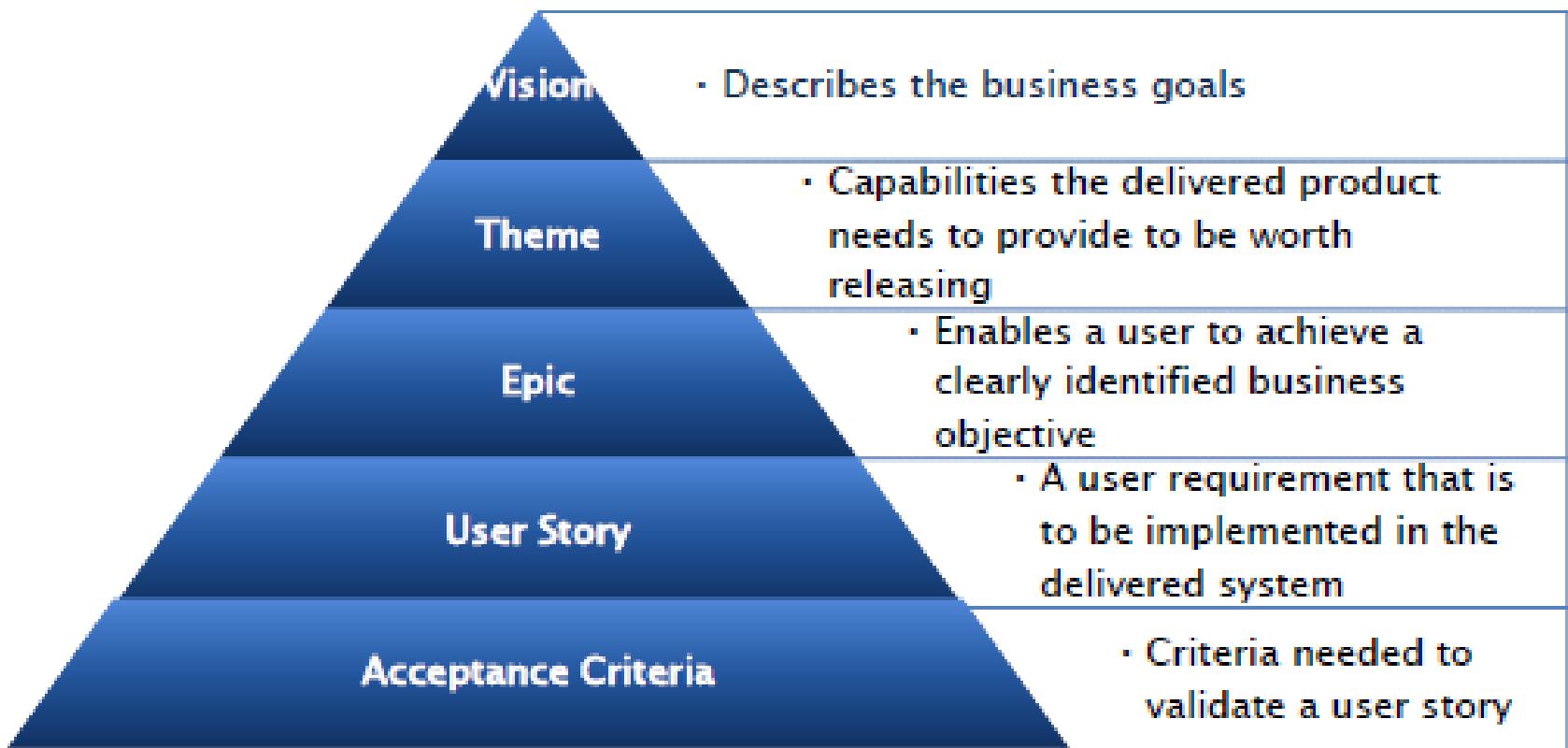
# Version of product backlog: Stakeholder and processes

## ▼ Funnel

---

- ▶ **Wish List 1** Delegated to: Stakeholder 1
- ▶ **Wish List 2** Delegated to: Stakeholder 2
- ▶ **Product Backlog** Delegated to: Product Owner
- ▶ **Verification / Release Ready**
- ▶ **Done** Archived

# Hierarchical requirement: User stories



# User story: Epic level

- **When a story is too large it is sometimes referred to as an *epic*.**
- **Epics can be split into two or more stories of smaller size.**

# User story: Epic level

- When a story is too large it is sometimes referred to as an *epic*.
- Epics can be split into two or more stories of smaller size.

# User story: Epic level

- An epic should be introduced by a short explanation.
  - *What is the problem we want to solve and why? (Problem statement)*
  - *Who are we doing it for? (Personas)*
  - *How do we want to improve it? (Goal/Solution)*

# Example of the epic

- Create products **Epic?**
- Create products online **Epic?**

# Example of the Epic

**As a customer , I want to be able to CRUD my products on my desktop PC, so that I have all product information in one place**

# Example of the epic

- User story:
  - *A user can search for a job*

could be split into these stories:

- *A user can search for jobs by attributes like location, salary range, job title, company name, and the date the job was posted.*
- *A user can view information about each job that is matched by a search.*
- *A user can view detailed information about a company that has posted a job.*

# Example of the epic: not splitting

- User story is a very reasonable and realistic story.
  - *A user can view information about each job that is matched by a search*”

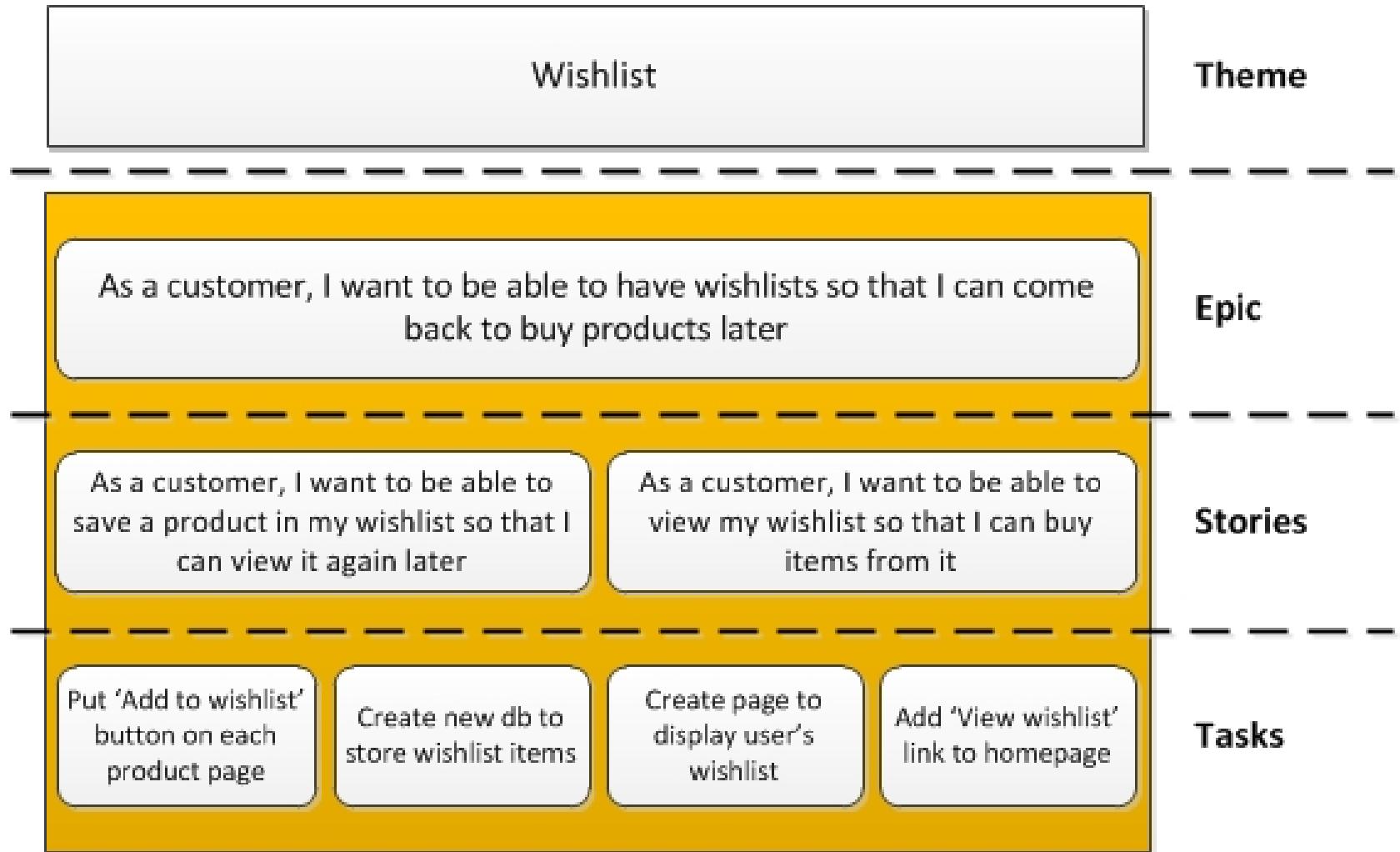
We do not need to divide it into smallest US:

- *A user can view a job description.*
- *A user can view a job's salary range.*
- *A user can view the location of a job.*

# Themes

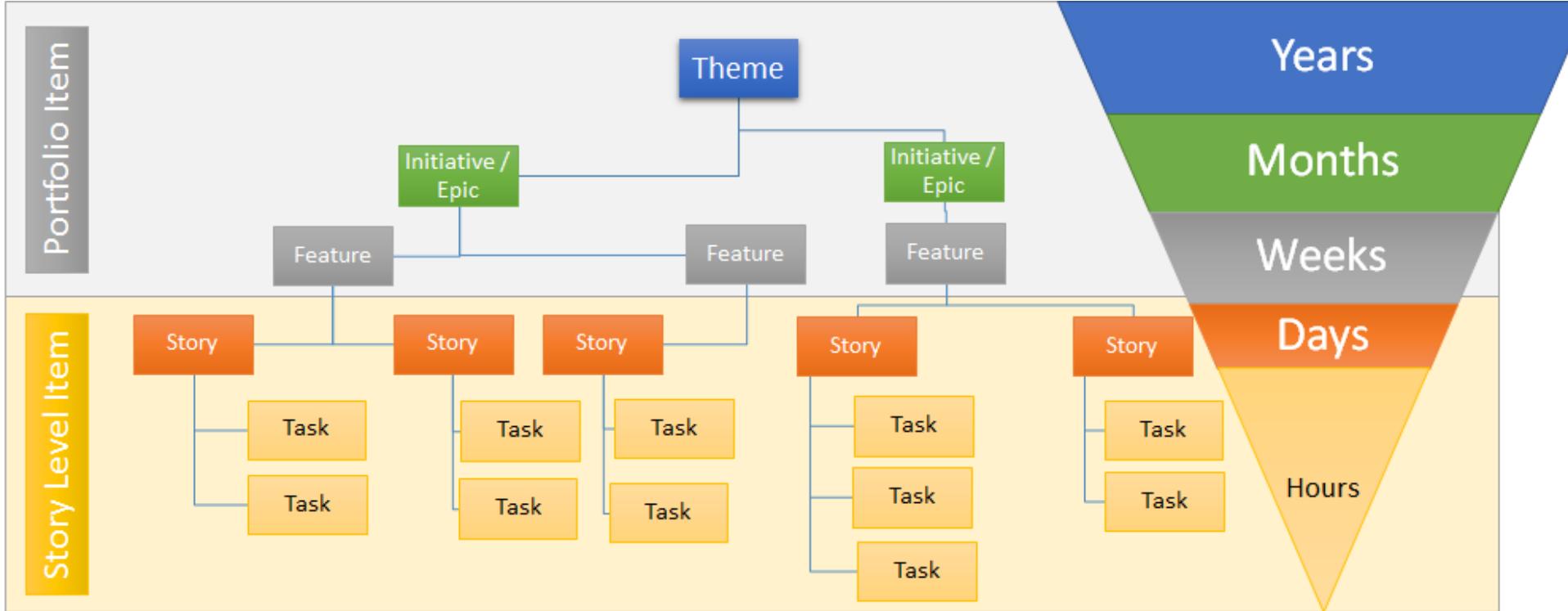
- A collection of related user stories.
- A theme provides a convenient way to indicate that a **set of stories have something in common**, such as being in the same functional area

# Example



# Feature

- **A distinguishing characteristic or capability of a software application or library:**
  - performance,
  - portability,
  - functionality and etc.



# Features vs. Epics

- A Feature is not a User Story.
- An Epic is a User Story.
- A User Story may be an Epic.
- A User Story can contain many Features.
- A Feature can fulfill 1 to many User Stories.

# Definition of User story

- A user story **describes functionality** that will be **valuable** to the a user or customer of a system or software
- Simple stories that describe **what the users world must look like**
- The basic user story template is simplistic, it **helps us remember a need while providing context.**
- A brief, simple **requirement statement from the perspective of the user**

# Definition of User story

- ...A user story is nothing more than an **agreement that the customer and developers will talk together about a feature.**

[Beck et al, 2001]

...understandable to customers and developers,  
testable, valuable to the customer and small  
enough so that the programmers can build half a  
dozen in an iteration

# User Story

As a <user role>

I want <goal>

so that <benefit>.

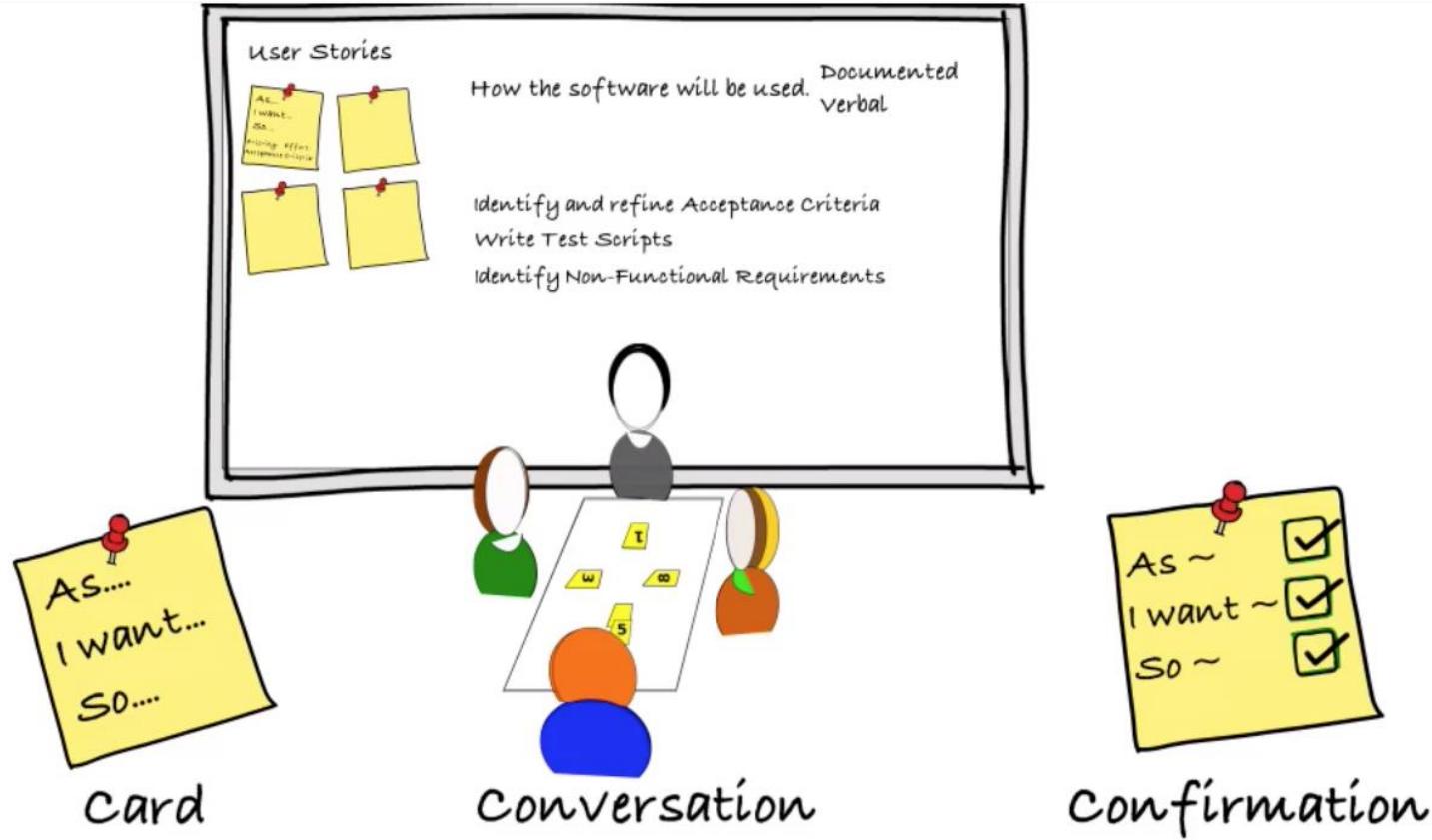
# User Story

<b>WHO</b> are we building it for? Who is the user?	As a <type of user>
<b>WHAT</b> are we building? What is the intention?	I want <some goal or objective>
<b>WHY</b> are we building it? What is the value for the customer?	So that <benefit/value>

# Purpose of US

- Communicate stakeholder need
- Describe product feature on a high level of abstraction
  - Can be understood by ALL stakeholders
- Reminder for future conversations
- Used for planning
- Provide value to the business

# The Three C's of User Stories



# The Three C's of User Stories

1. a written description of the story used for planning and as a reminder
2. conversations about the story that serve to flesh out the details of the story
3. tests that convey and document details and that can be used to determine when a story is complete

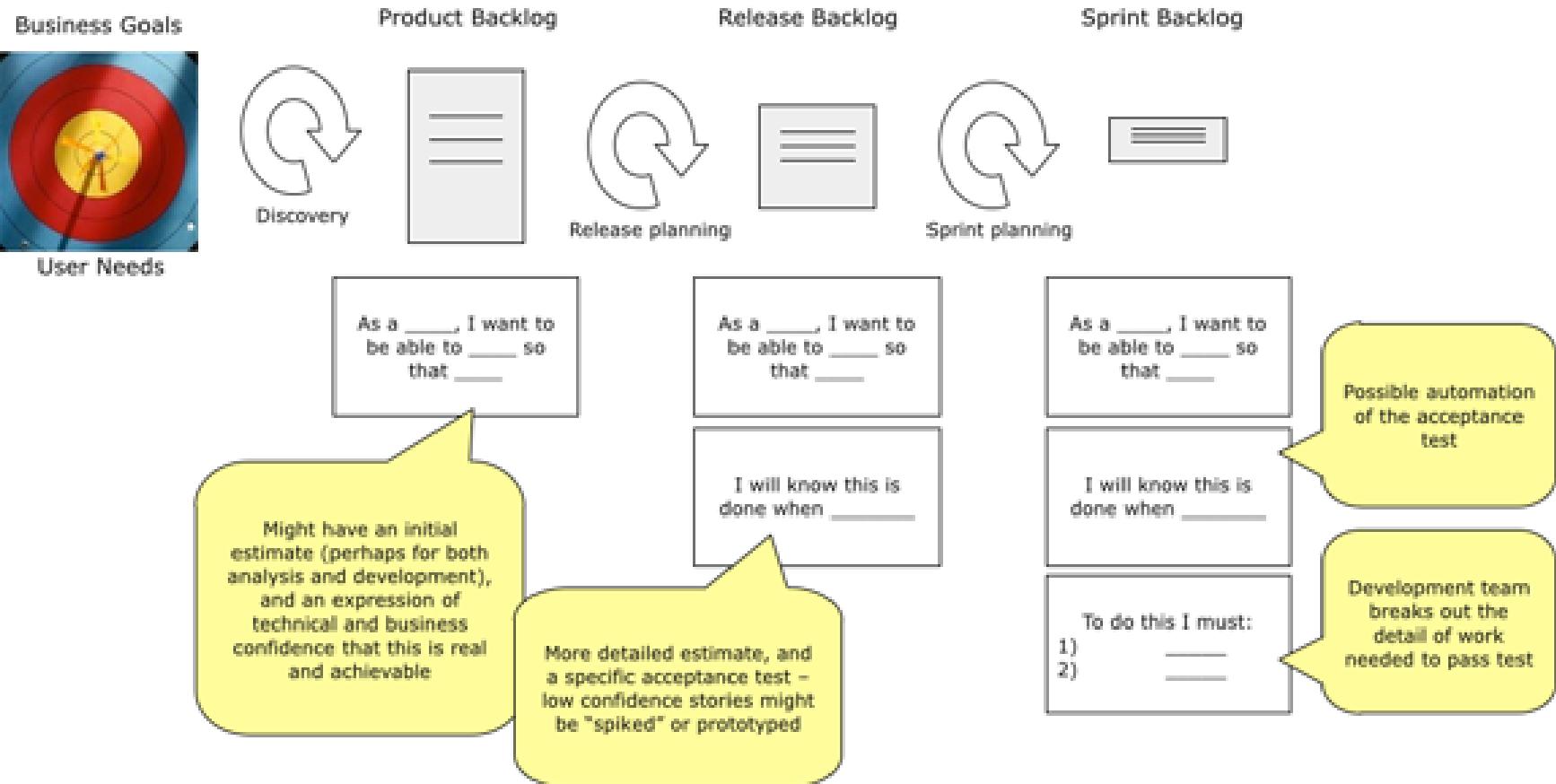
# US Refinement

- Refinement ensures the highest priority Stories meet the **Definition of Ready (DoR)**
- Product Owner and the Development Team collaborate to ensure that items on the Product Backlog:
  - **are understood the same way by all involved**
  - have a size estimate for the (relative) complexity and effort of their implementation,
  - are ordered according to their priority in terms of business value and effort required.

# US Refinement



# User Story Lifecycle



<https://agilefaqs.com/services/training/user-stories>

# Component of user stories

- Title— this is a short handle for the story.
  - A present tense verb in active voice is desirable in the title.
- Acceptance test—this is a unique identifier that will be the name of a method to test the story.
- Priority
- Story points—this is the estimated time to implement the user story.

# Task

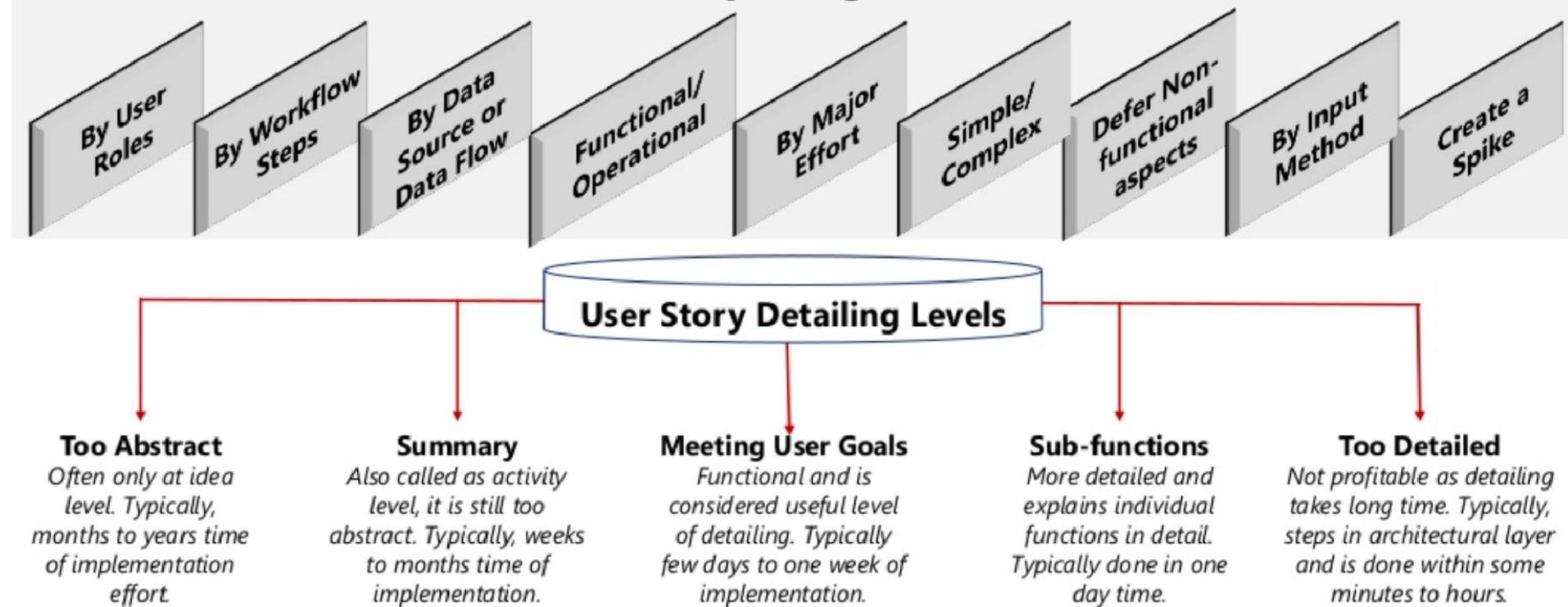
- What are the activities we need to perform in order to deliver outcomes (user stories)
- Tasks are individual pieces of work
- Main question – HOW?

# Writing user stories

- Developing user stories is an “iterative and interactive” process.
- The development team also manages the size of stories for uniformity:
  - too large—split, too small—combine).

# Splitting US

## Patterns of Splitting User Stories



# Example of US

- As a
  - *register user*
- I want
  - *change my password*
- So that
  - *keep my account secure*



# Example of US

- As a
  - *register user*
- I want
  - *remember my password*
- So that
  - *I have possibility to login in the system, when I don't remember it*

As a <role>  
I want <goal>  
So that <benefit>

Acceptance criteria:

...

# Example of US ?

- As a
  - *admin user* ?
  - *administrator of IS department* ??
- I want
  - *disable users* ??
  - *create rights of the users in the system* ?
- So that
  - *prevent unauthorized logins* ?
  - *ensure secure of my systems* ?



# Example of US ?

- As a
  - *user*
- I want
  - *view information about each job*
- So that
  - *is matched by a search ??*
  - *Is best possibilities for my future ??*



# Example of US

- As a
  - *guest*
- I want
  - *register in the system*
- So that
  - *admin has list of user*

Bad example, WHY?

As a <role>  
I want <goal>  
So that <benefit>

Acceptance criteria:

...

# Example of US

- As a
    - *register user*
  - I want
    - *to find a book about Java programming language*
  - So that
    - *has possibilities to improve my programming skills*
- Bad example, WHY?*



# Example of US

- As a
  - *student John*
- I want
  - *to get notifications about changes of timetable*
- So that
  - *I could arrange better my activities*

As a <role>  
I want <goal>  
So that <benefit>

Acceptance criteria:

...

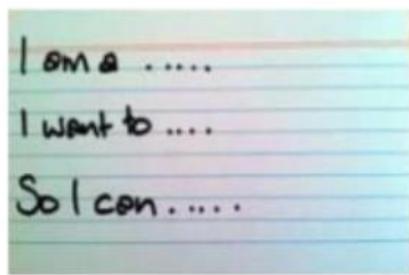
Bad example, WHY?

# Example of US

- *As a programmer I want to write the software code in C++ So that this language is OO*
- *As a developer I want use SQL database So I can store information without duplication*
- **Why these stories are wrong?**

# The Three C's of User Stories

## Card



## Conversation



## Confirmation



Stories are traditionally written on note cards

Details of a story come out during conversations with Product Owner

Acceptance tests confirm the results are what the requestor wants.

# User story template

<i>Story Title</i>	<b>Estimate:</b> ... story points
<b>As a</b> ... ← who? fill this in with a stakeholder type	
<b>I want</b> ... ← what? fill this in with the problem they want solved	
<b>so that</b> ... ← why? fill this in with the value they will gain	
<b>Additional detail:</b>	
... ← what else do we need to know to understand the story?	
<b>Acceptance criteria</b>	
We'll agree it's been completed when:	
<ul style="list-style-type: none"><li>• ... ← add a list of easy-to-verify true/false statements</li><li>• ... ← concrete examples are best</li><li>• ...</li><li>• ...</li><li>• ...</li></ul>	

# Acceptance test

- Acceptance testing is the process of verifying that **stories were developed** such that each works exactly the way the customer team expected it to work
- Tests should be **written as early** in an iteration as possible
- Acceptance tests also **provide basic criteria that can be used to determine if a story is fully implemented.**

# Acceptance test

- Valid for a specific user story
- Describes when value is realized for customer
- Should be testable
- Basis for estimations
- Can change over time
- Has to be fulfilled before story is moved to testing stage
- Provide basic criteria that can be used to determine if a story is fully implemented

# Acceptance Criteria Goals

- To clarify what
- To ensure everyone has a common understanding of the problem
- To help the team members know when the story is complete
- To help verify the story via automated tests

# User Story with Acceptance Criteria

## User Story:

- As an online banking customer, I want strong a strong password, so that my credit card information is secure

## Acceptance Criteria:

1. The password must be at least 8 characters
2. The password must contain at least 1 character from each of the following groups: lower case alphabet, upper case alphabet, numeric, special characters (!, @, #, \$, %, ^, &, \*)

# User Story with Acceptance tests

- User Story:
  - *A user can pay for the items in her shopping cart with a credit card*
- Acceptance tests:
  1. Test with Visa, MasterCard and American Express (pass).
  2. Test with Diner's Club (fail).
  3. Test with a Visa debit card (pass).
  4. Test with good, bad and missing card ID numbers from the back of the card.
  5. Test with expired cards.
  6. Test with different purchase amounts (including one over the card's limit).

# User story with Acceptance tests

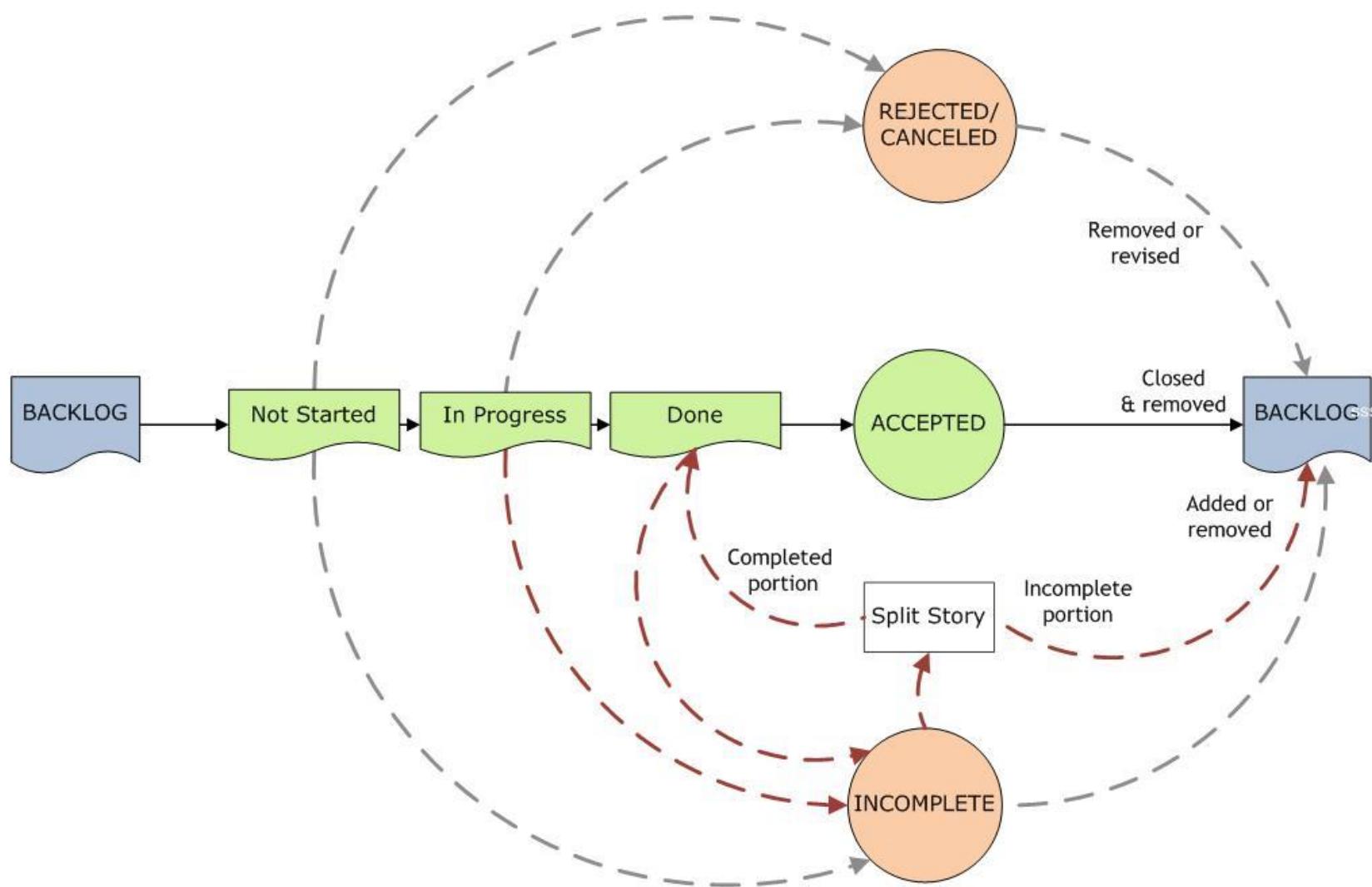
User story:

*As an employer I want to post a job so that others people can find it.*

Acceptance tests :

1. Verify that only an authorized user with a valid employer account can post a job.
2. Verify that a duplicate job posting cannot be entered.
3. Verify that the posting date is past today's date.
4. Verify that the posting expiration date within 90 days.
5. Verify that the screen fields pass our standard field format rules ([link here to doc](#)).
6. Verify that all required fields are entered (list them or link to UI Prototype).

State diagram for a user story



# **PRACTICAL VIEWPOINT: USER STORY**

Dr. Asta Slotkiene

# Epic level, examples

1. As a administrator, I want to administration users in e-bank system, so that I can do all actions with user data
2. As a administrator, I want to see user list, who are register/banned in the system
3. As a administrator, I want to generate various reports up to 10 s, so that I could get it quickly
4. As a administrator, I want to get data form database by Rest technology ?

# User stories level: examples

1. As a bank users, I want to find near ATM of my address, so that I can quickly to perform action with my deposit
2. As a administrator, I want to add new user, so that I can to fill true information about users

# Task level: examples

1. As a administrator, I want to change types of user, so that I can to have correct users list
2. As a administrator, I want to get user list by each criteria
3. As a administrator, I want to get user list by course of study
4. As a administrator I want to choose the role of user of the ComboBox ?

# Epic level/User story level/Task level ?

1. As a administrator, I want to generate log of action for 1 day, so that I can to find any mistake
2. As a register user, I want to buy, edit or cancelling my order, so that I can manage my order list
3. As a administrator, I want to make any change of user profile
4. As a register user, I want to change my address, So what my order to send by it

# INVEST model

Follow the INVEST  
guidelines for good  
user stories!



# User stories: INDEPENDENT

- **US should be self contained, in a way that there is no inherent depends on other user stories**
- For large systems this is nearly impossible, but, minimizing, identifying and prioritizing, **dependencies can result in a better backlog**
- **Dependencies between stories lead to prioritization and planning problems.**

# User stories: independent

Are these US independent?

1. *As a user I want to register, so that I could to perform test of knowledges*
2. *As a user I want to check my knowledge, that I would like to know my level of skills*
3. *As a register user I want to get report of my test*

# User stories: independent

1. *A company can pay for a job posting with a Visa card.*
  2. *A company can pay for a job posting with a MasterCard.*
  3. *A company can pay for a job posting with an American Express card.*
- 
- Why these stories are splitting not correctly?

# User stories: independent

1 solution: Combine the dependent stories into one larger, but independent story

- *A company can pay for a job posting with a credit card*

2 solution: Find a different way of splitting the stories

- *A customer can pay with one type of credit card.*
- *A customer can pay with two additional types of credit cards.*

# User story: Negotiable

- Stories are not written contracts or requirements that the software must implement.
- Story cards are short descriptions of functionality
- US are not explicit contract and should leave space for discussions

# User story: Negotiable

1. *As a driver I want to get shortly directions to my destination, so that I get there quickly*
  2. *As a driver I want to get directions to my destination on map, so that to use Google maps*
- Where isn't negotiable?

# User story: Valuable

- Each story must be valued by the users
- Many projects include stories that are not valued by users.
- Keeping in mind the distinction between
  - *user*: someone who uses the software
  - *purchaser*: someone who purchases the software
- The best way to ensure that each story is valuable to the customer or users is to have the customer write the stories

# User story: Valuable

1. *As a user I want to have my previous orders stored in the database So they will be there permanently*
2. *As a customer I want 75% off all purchases, So I can save money*

# User story: Small/Estimable

- User story should not be so big as to become impossible to plan/task/prioritize with certain level of certainty
- You must be always estimate the size of US

# User story: small? Estimate?

1. *As a customer I want to perform e-services of my bank account, so I can do all action with my account*
2. *As a customer I want to find an ATM near the address , so that I can make deposits outside of banking hours*

# User story: Small/Estimate

Divide to small US:

- *As a bank user I want to see my canceled operations online*
- *As a bank user I want to make the money transfer online*
- *As a bank user I want to plane an transfer which bank will do later*

# User story: Testable

- Description of US must be provide the necessary information to make test cases
- Solutions include adding acceptance criteria or better defining the story
- If the story cannot be tested, how can the developers know when they have finished coding?

# Untestable user stories

- They show up for nonfunctional requirements, which are requirements about the software but not directly about its functionality.
  - *A user must find the software easy to use.*
  - *A user must never have to wait long for any screen to appear.*

# Untestable ->Testable user stories

- *A user must find the software easy to use.*
- ***A novice user is able to complete common workflows without training***
- *A user must never have to wait long for any screen to appear.*
- **New screens appear within two seconds in 95% of all cases**

# User story: Testable 1

- *As a user I want to good GUI, so that quickly understood it*
- *As a user I want to simple GUI, so that quickly understood it*
- Acceptance criteria:
  - All text is dark colour on light background
  - Only two different fonts used in GUI

# User story: Testable 2

- *As a user I want to get confirmation of my order, so that I will be guaranteed*
- **Acceptance criteria:**
  - Copy of Confirmed order send by email through 1 hour
  - Decreasing selected items (of order) in database

# Requirement vs User Story

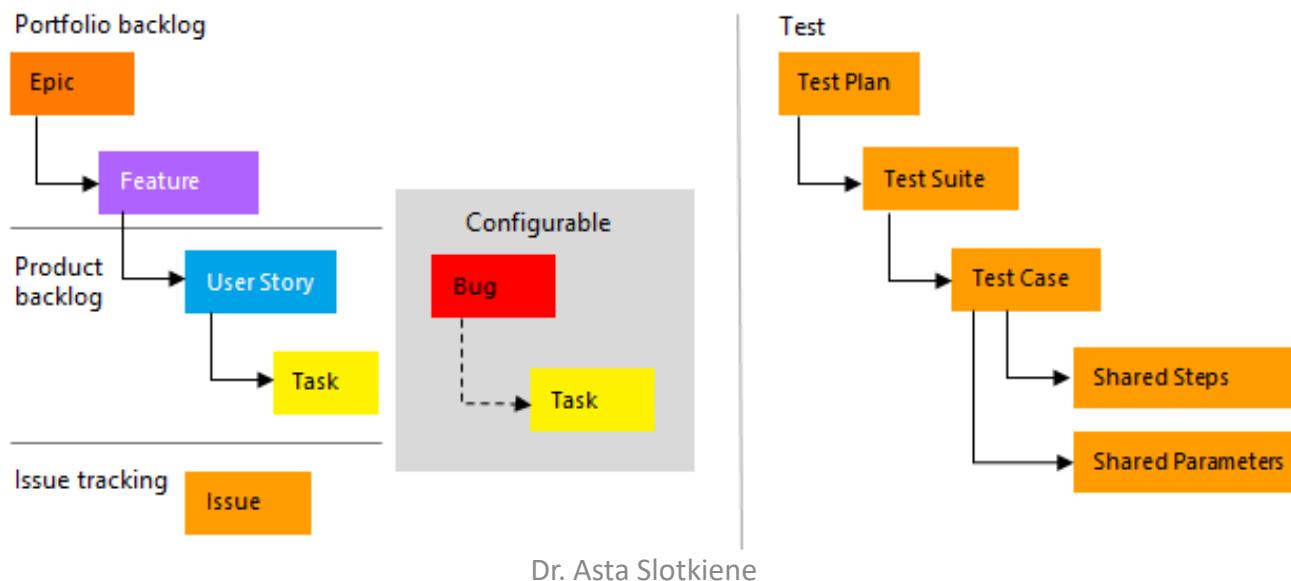
- REQUIREMENT =
  - User story (end users) +
  - Acceptance Criteria (non –functional requirements) +
  - Supporting/technical requirements

# Summary of US

- Ideally, stories are independent from one another. This isn't always possible but to the extent it is, stories should be written so that they can be developed in any order.
- The details of a story are negotiated between the user and the developers.
- Stories should be written so that their value to users or the customer is clear. The best way to achieve this is to have the customer write the stories.
- One of the best ways to annotate a story is to write test cases for the story
- Stories need to be testable.

# Requirement+ Test Plan

- REQUIREMENT =
  - User story (end users) +
  - Acceptance Criteria (non –functional requirements) +
  - Test cases +
  - Supporting/technical requirements



# Produkto reikalavimų inžinerijos procesas

Dr. Asta Slotkienė

# New and Changed Practices

- Technical Solution!

Level		Focus	Process Areas	Quality Productivity
5 Optimizing	Continuous Process Improvement	Causal Analysis and Resolution (CAR) Organizational Performance Management (OPM)		
4 Quantitatively Managed	Quantitative Management	Organizational Process Performance (OPP) Quantitative Project Management (QPM)		
3 Defined	Process Standardization	Integrated Project Management (IPM) Risk Management (RSKM) Decision Analysis and Resolution (DAR) Requirements Development (RD) Technical Solution (TS) Product Integration (PI) Verification (VER) Validation (VAL) Organizational Process Focus (OPF) Organizational Process Definition (OPD) Organizational Training (OT)		Risk Rework
2 Managed	Basic Project Management	Configuration Management (CM) Measurement and Analysis (MA) Project Monitoring and Control (PMC) Project Planning (PP) Process and Product Quality Assurance (PPQA) Requirements Management (REQM) Supplier Agreement Management (SAM)		
1 Initial				

CMMI Development V1.3

Level		PA also exists at higher level	Practice Areas	Performance Capability
5 Optimizing			CAR, MPM (level 5 practices)	
4 Quantitatively Managed			CAR, PCM, SAM, MPM, PLAN, GOV (level 4 practices)	
3 Defined		✓	Causal Analysis and Resolution (CAR) Decision Analysis and Resolution (DAR) Risk and Opportunity Management (RSK) Organizational Training (OT) Process Management (PCM) Process Asset Development (PAD) Peer Reviews (PR) Verification and Validation (VV) Technical Solution (TS) Product Integration (PI)	
2 Managed		✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓	Supplier Agreement Management (SAM) Managing Performance and Measurement (MPM) Process Quality Assurance (PQA) Configuration Management (CM) Monitor and Control (MC) Planning (PLAN) Estimation (EST) Requirements Development & Management (RDM) Governance (GOV) Implementation Infrastructure (II)	
1 Initial / 0 Incomplete				Risk Rework

CMMI Development V2.0 – Simple View

# Technical Solution (TS) (CMMI-DEV)

- The purpose of Technical Solution (TS) is to **select, design, and implement solutions** to requirements.
- Solutions, designs, and implementations encompass **products, product components**, and product related lifecycle processes either singly or in combination as appropriate.
- Its intention is **to ensure that designed and prepared products are in line with customer requirements**.
- TS value is providing effective design and solution that meets customer requirements while reducing costs and reworks. (ML1, ML2, ML3)

# Technical Solution (TS) (CMMI-DEV)

**TS.SG 1 Select Product Component Solutions** Product or product component solutions are selected from **alternative solutions**.

**TS.SG 2 Develop the Design** Product or product component designs are developed.

**TS.SG 3 Implement the Product Design** Product components, and associated support documentation, are implemented from their designs.

Technical Solution (TS) (CMMI-DEV): **TS.SG 1**

## **Select Product Component Solutions Product**

- Architectural choices
- Architectural patterns
  - that support achievement of quality attribute requirements
- The use commercial off-the-shelf (COTS) product components are considered relative to cost, schedule, performance, and risk.
  - COTS alternatives can be used with or without modification

# Alternative solutions and selection criteria (CMMI-DEV)

- Cost of development, manufacturing, procurement, maintenance, and support,
- Achievement of key **quality attribute requirements**, such as product timeliness, safety, reliability, and maintainability
- **Complexity of the product component and product-related lifecycle processes**
- **Robustness to product operating** and use conditions, operating modes, environments, and variations in product-related lifecycle processes
- **Product expansion and growth**
- **Technology limitations**
- Sensitivity to construction methods and materials
- Risk
- Evolution of requirements and technology
- Disposal
- Capabilities and limitations of end users and operators
- *Characteristics of COTS products*

# Software Architect vs. Software Design

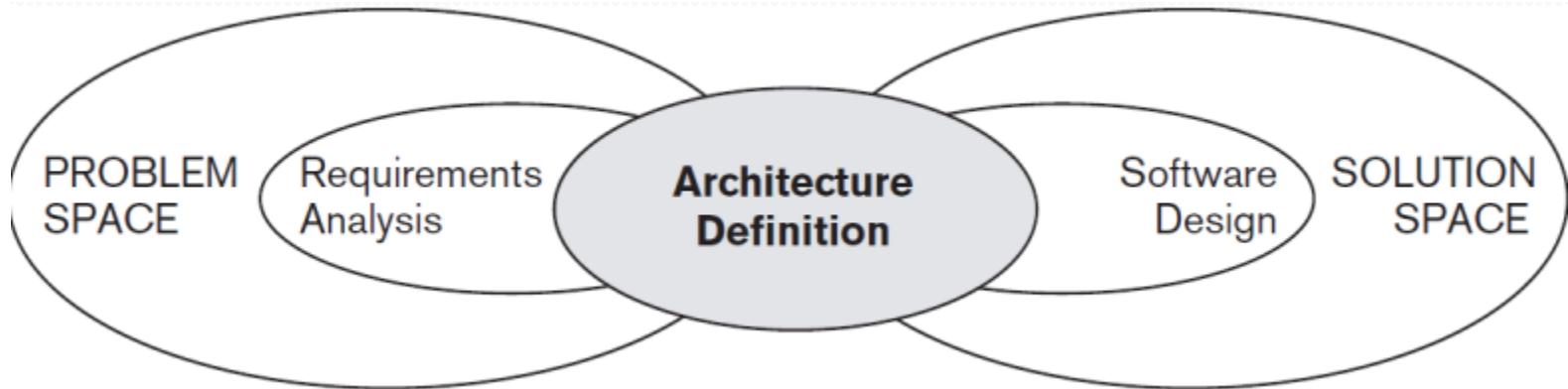
- Subsystems
- Systems
- Integration
- Code
- Classes
- Components
- Packages
- Modules

*All architecture is design but not all design is architecture.*

*Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change.*

*[Grady Booch]*

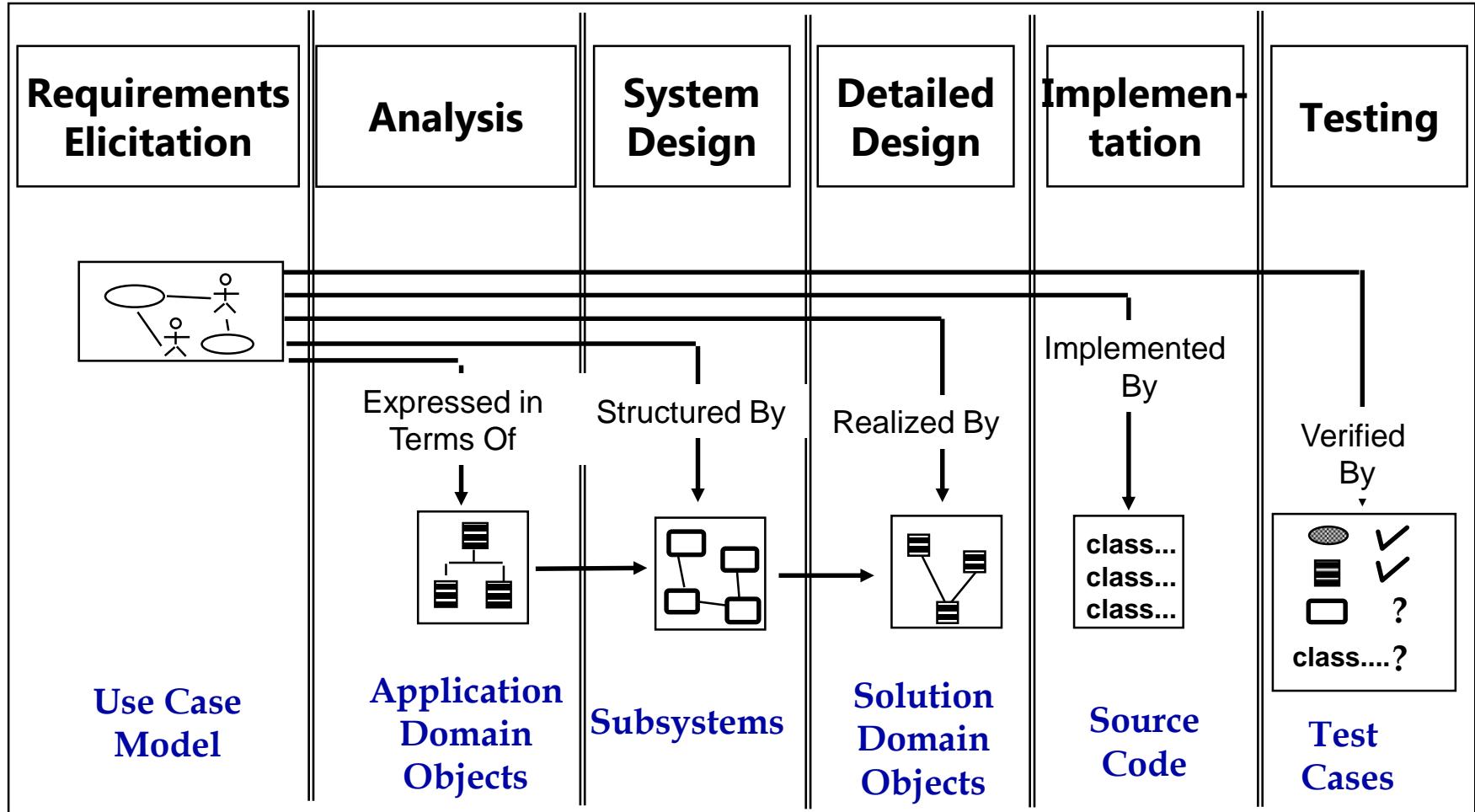
# Software Architect vs. Software Design



*All architecture is design but not all design is architecture.*

*Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change.  
[Grady Booch]*

# Activities of Software System Lifecycle



# From Requirements to Design

The requirements addressed the

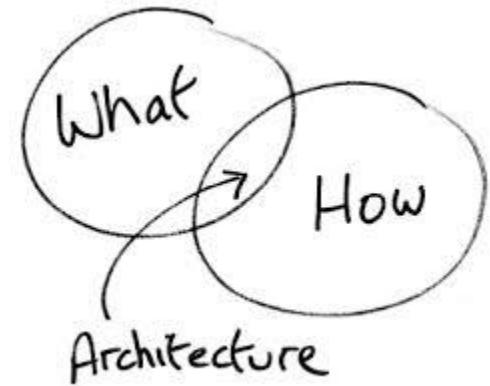
**“What?”**

*what the system is supposed to do, what are the constraints, etc.*

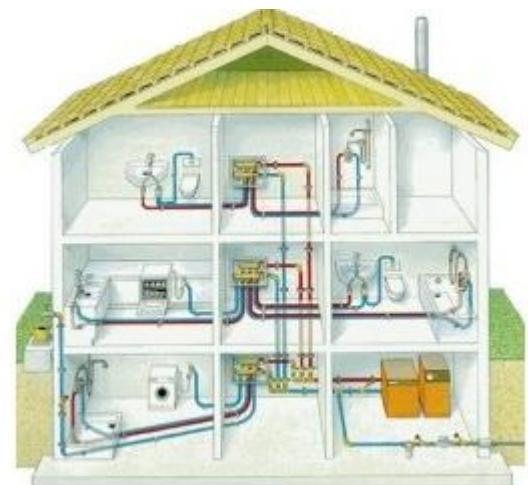
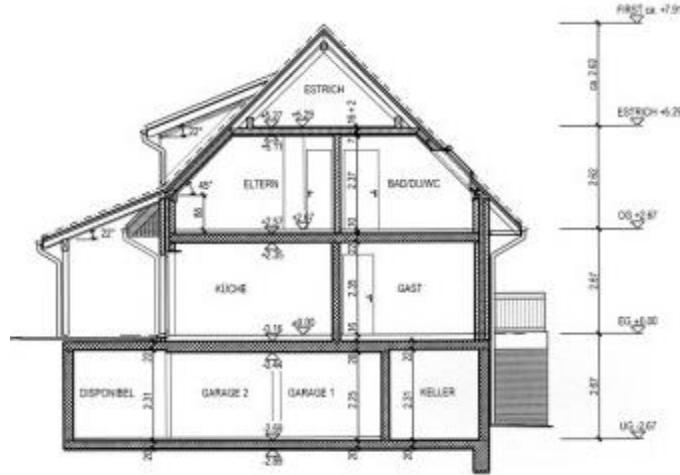
design addresses the

**“How?”**

*how the system is decomposed into components, how these components interface and interact, how each individual component works, etc.*



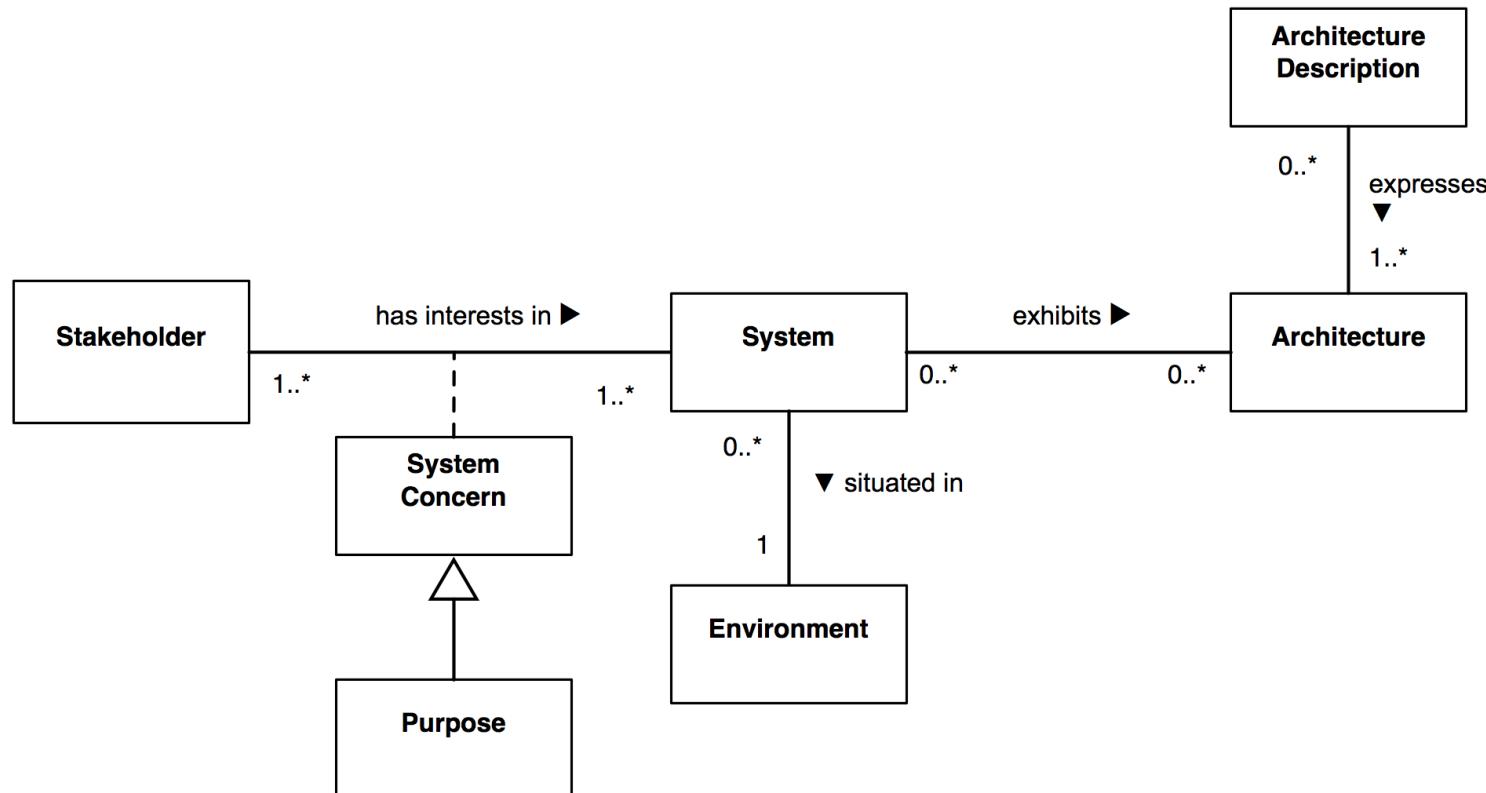
# What is architecture?



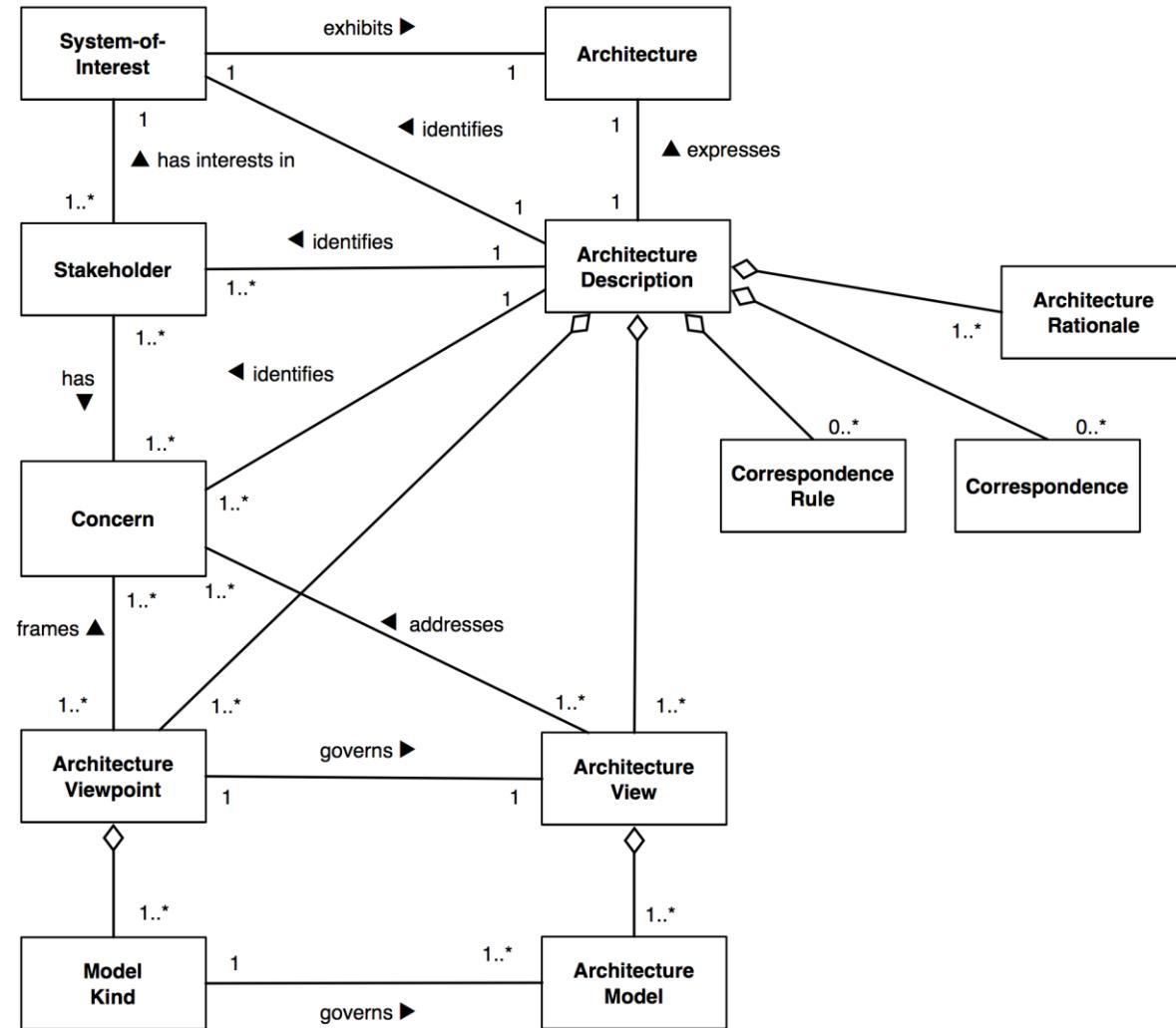
# Definition of Software Architecture

The fundamental organization of a system embodied in its **components, their relationships to each other**, and to the environment, and the principles guiding its design and evolution.

# A Conceptual Model of Architecture Description



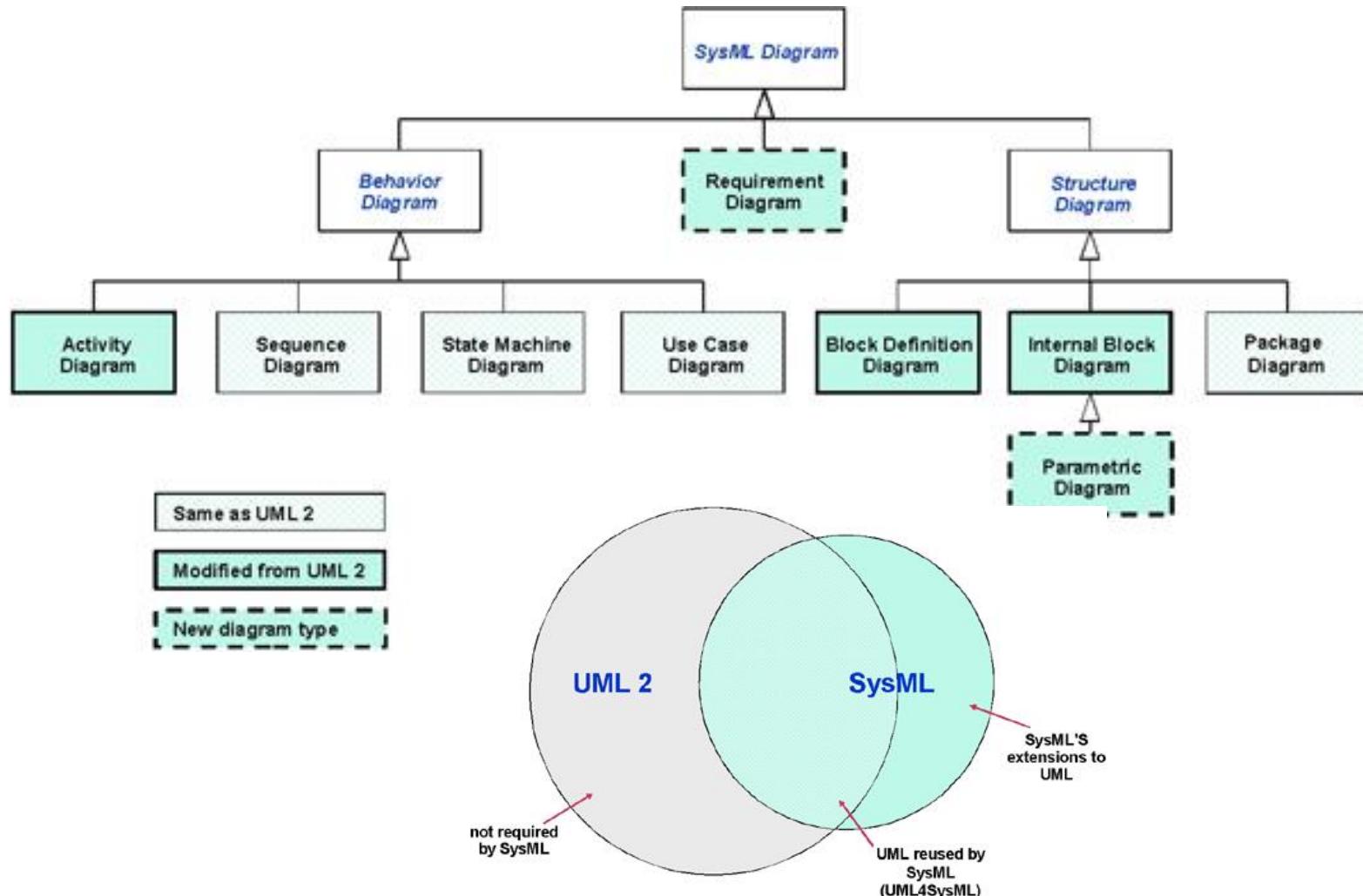
# Software Architecture Description



# Software architecture is driven by

- Functional needs
  - Your requirements.
- Quality needs
  - Non-functional requirements, e.g., performance, security.
- Constraints
  - Technical, legal, economic.

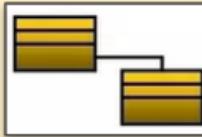
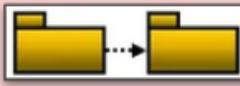
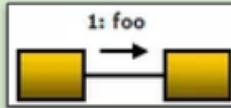
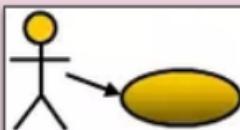
# OMG SysML/UML diagram



# 4+1 Architectural view model: UML diagrams

Functional

Non-functional

	Conceptual / Logical	Physical / Operational
Logical / Structural view	<p><b>Perspective:</b> Analysts, Designers <b>Stage:</b> Requirement analysis <b>Focus:</b> Object oriented decomposition <b>Concerns:</b> Functionality <b>Artefacts:</b></p> <ul style="list-style-type: none"><li>• Class diagram</li><li>• Object diagram</li><li>• Composite structure diagram</li></ul> 	<p><b>Perspective:</b> Developers, Proj. mngrs. <b>Stage:</b> Design <b>Focus:</b> Subsystem decomposition <b>Concerns:</b> Software management <b>Artefacts:</b></p> <ul style="list-style-type: none"><li>• Component diagram</li><li>• Package diagram</li></ul> 
Process / Behaviour view	<p><b>Perspective:</b> System Integrators <b>Stage:</b> Design <b>Focus:</b> Process decomposition <b>Concerns:</b> Performance, scalability, throughput <b>Artefacts:</b></p> <ul style="list-style-type: none"><li>• Sequence diagram</li><li>• Communication diagram</li><li>• Activity diagram</li><li>• State (machine) diagram</li><li>• Interaction overview diagram</li><li>• Timing diagram</li></ul> 	<p><b>Perspective:</b> End users <b>Stage:</b> Putting it alltogether <b>Concerns:</b> Understandability, usability <b>Focus:</b> Feature decomposition <b>Artefacts:</b></p> <ul style="list-style-type: none"><li>• Use-case diagram</li><li>• User stories</li></ul> 

# CMMI-Dev: Technical Solutions

- Purpose:
  - The focus of the technical solution **is to design and implement solutions according to the requirements.**
  - Develop solutions for design alternatives that:
    - feature current vs. new technologies,
    - COTS products, tools and so on

***Always are more than one possible solution how to implement a requirement***

[https://en.wikipedia.org/wiki/Commercial\\_off-the-shelf](https://en.wikipedia.org/wiki/Commercial_off-the-shelf)

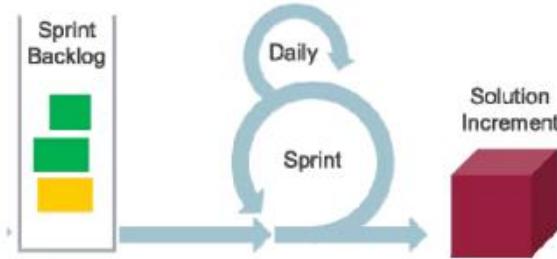
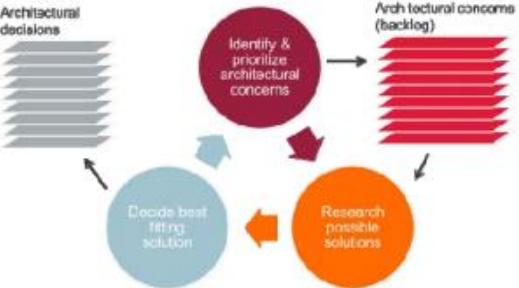
<https://searchdatacenter.techtarget.com/definition/COTS-MOTS-GOTS-and-NOTS>

# CMMI-Dev: Technical Solutions

A couple things to keep in mind when developing the design:

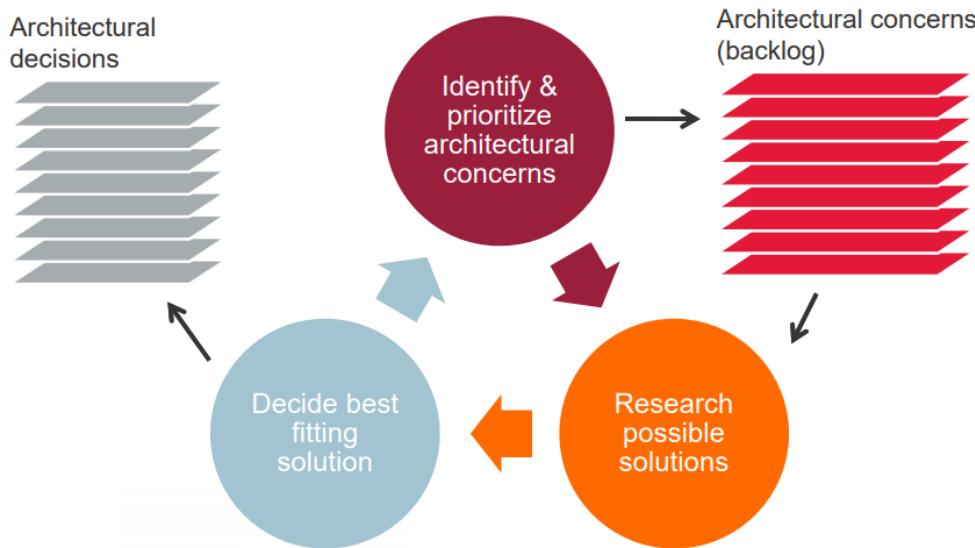
- Employ techniques and methods known for producing effective design (
  - prototypes,
  - structural models,
  - object oriented design,
  - entity relationship models)
- Ensure that design standards are adhered to while preparing the design:
  - ISO 42010, TOGAF, Archimate, 4 + 1, etc

# Architectural decisions

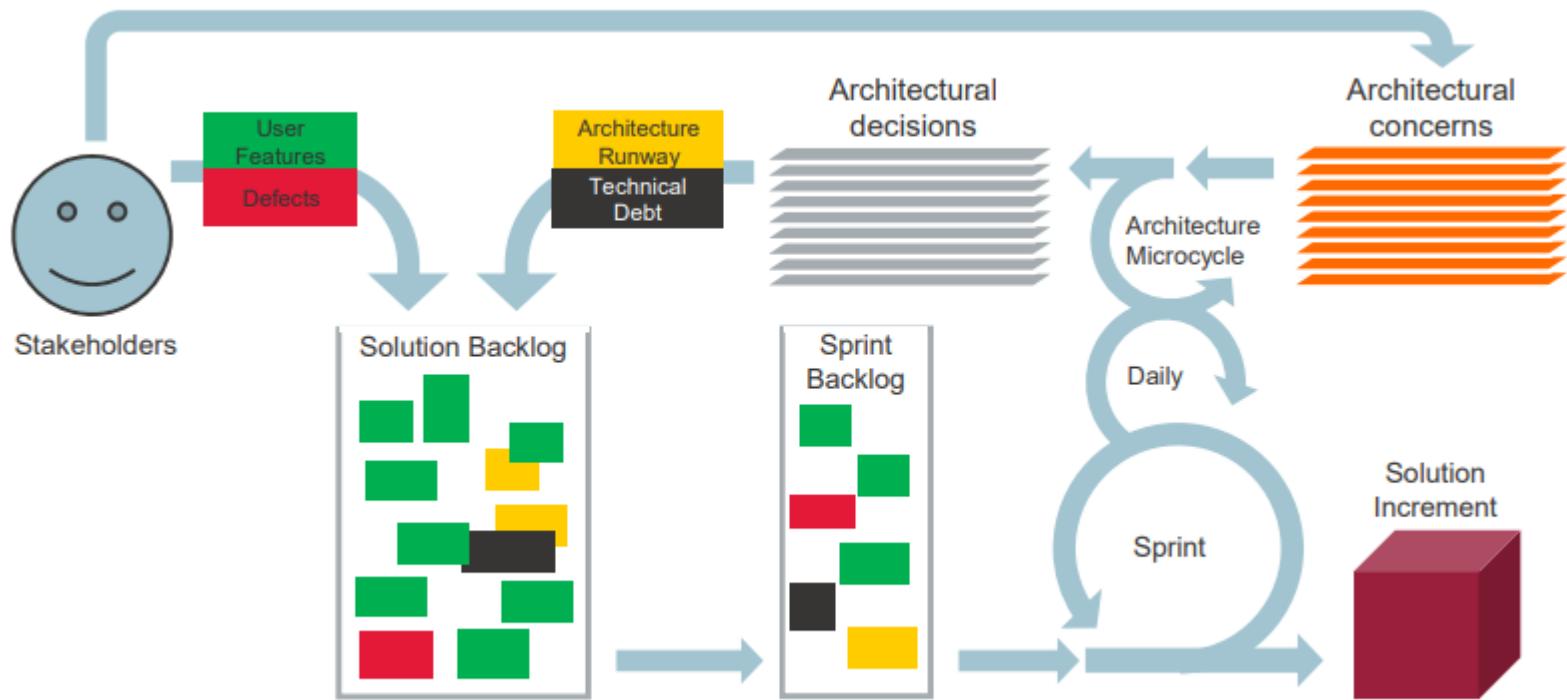
	Traditional	Agile
Development	 <p>Big Bang</p>	 <p>Continuous stream of improvements</p>
Architecture	 <p>Big Up-Front Design</p>	 <p>Continuous stream of Architectural Decisions</p>

# What is Continuous Architecture?

Continuous Architecture is about using the appropriate tools to make the **right decisions** and support **Continuous Delivery, Continuous Integration and Continuous Testing**



# SCRUM and the Architecture Microcycle



# Continuous Architecture Principles

1. Architect Products - Not Just Solutions for Projects.
2. Focus on Quality Attributes - not on Functional Requirements.
3. Delay design decisions until they are absolutely necessary.
4. Architect for Change – Leverage “The Power of Small”.
5. Architect for Build, Test and Deploy.
6. Model the organization of your teams after the design of the system you are working on.

# Software architecture evolution

## Evolution of Software Architectures

