Software Engineering VU [194.020]

Maria Christakis

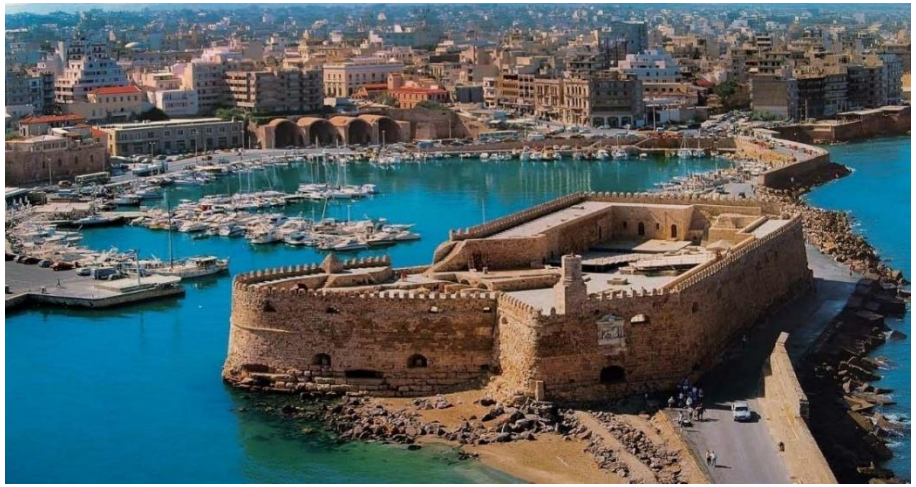
TU Wien

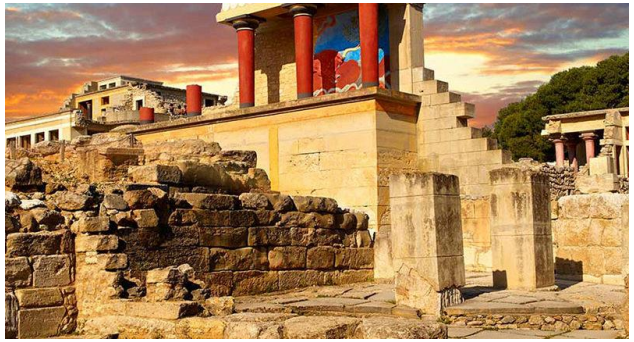
https://mariachris.github.io

About me

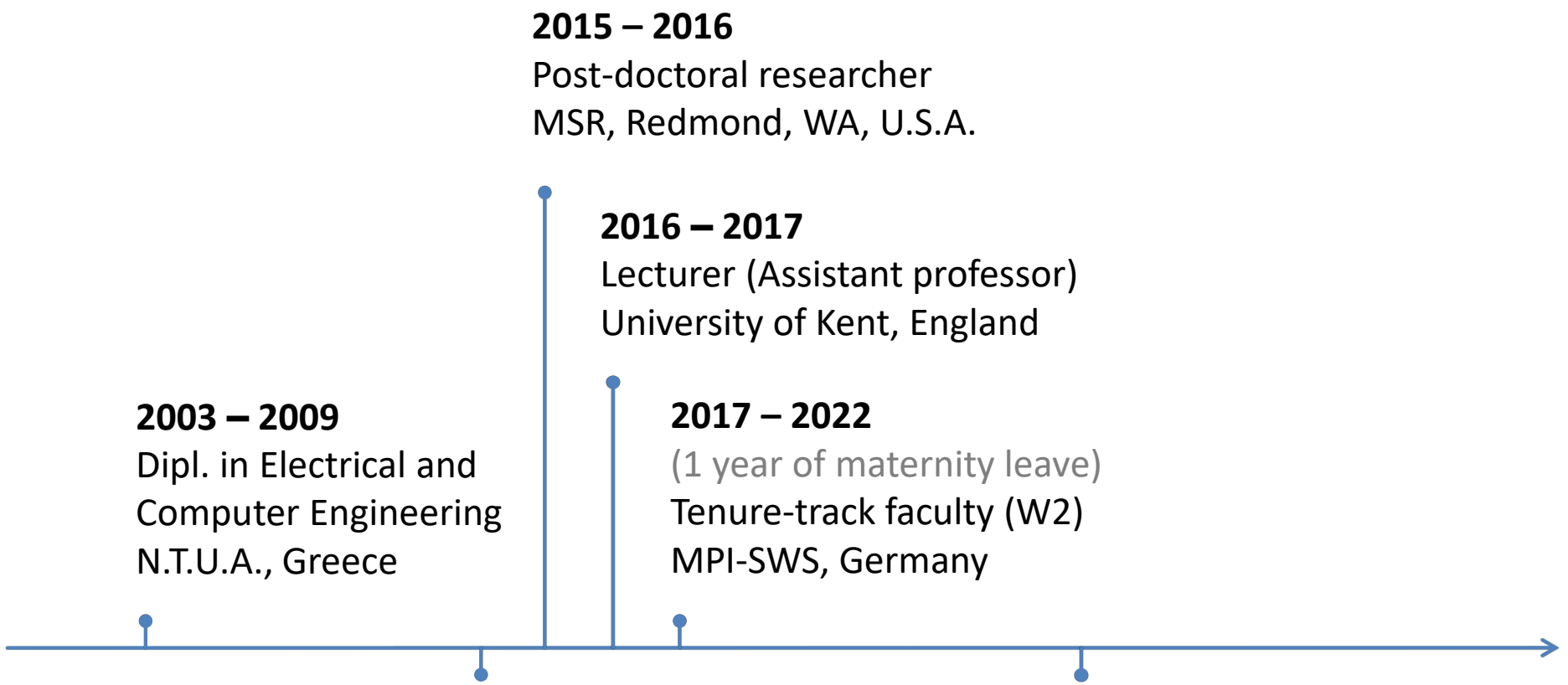
I am from Crete, Greece

I am from Crete, Greece





Where I have been until now



About my research

Software has bugs

Software is everywhere

but it is not reliable

Program analyzers find software bugs

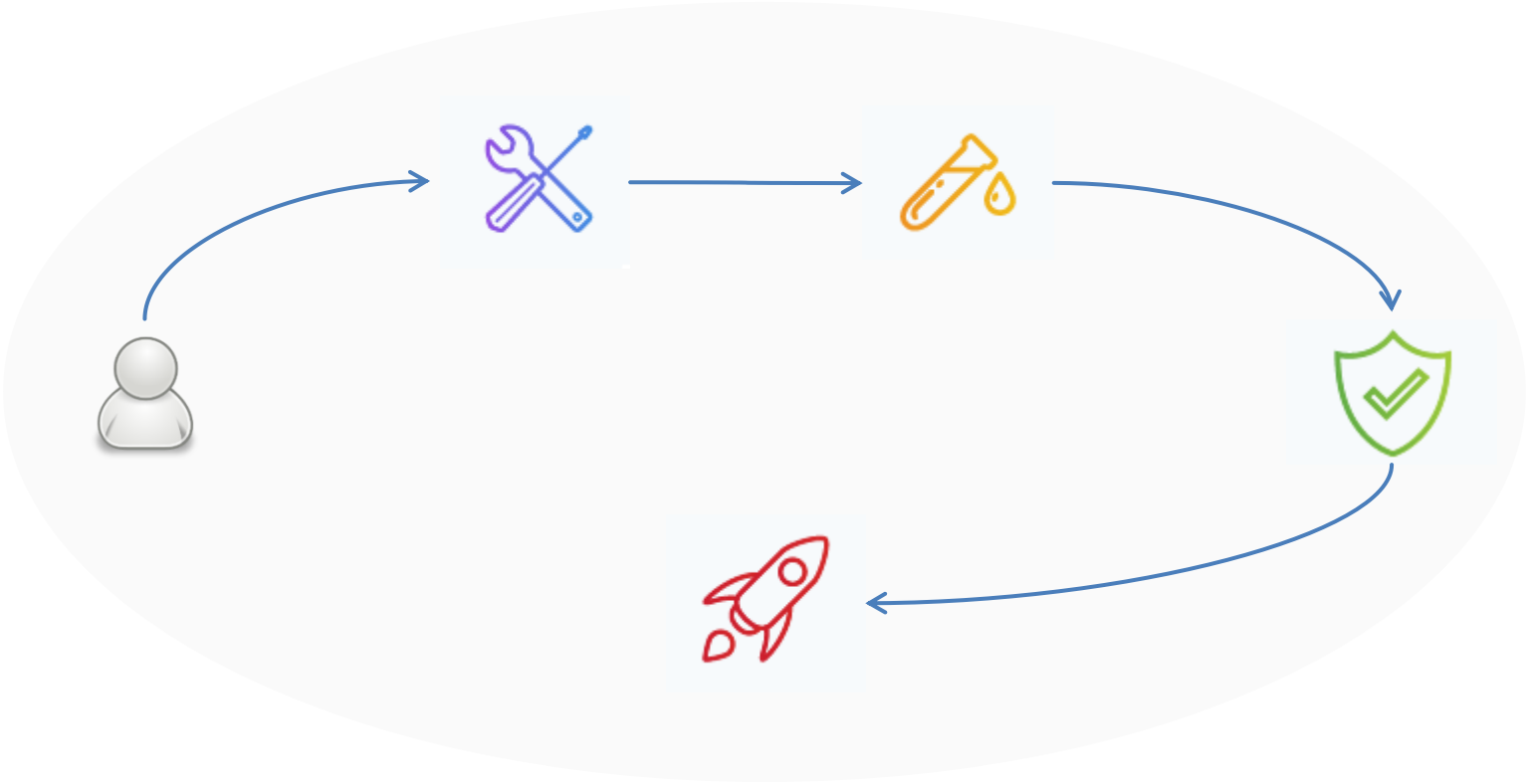
A program analyzer is software that analyzes other software to find bugs

Program analyzers find software bugs

A program analyzer is software that analyzes other software to find bugs

Analyzers can prevent catastrophic defects in safety-critical software

I work on program analysis



How to build more reliable software while increasing developer productivity

Software development models

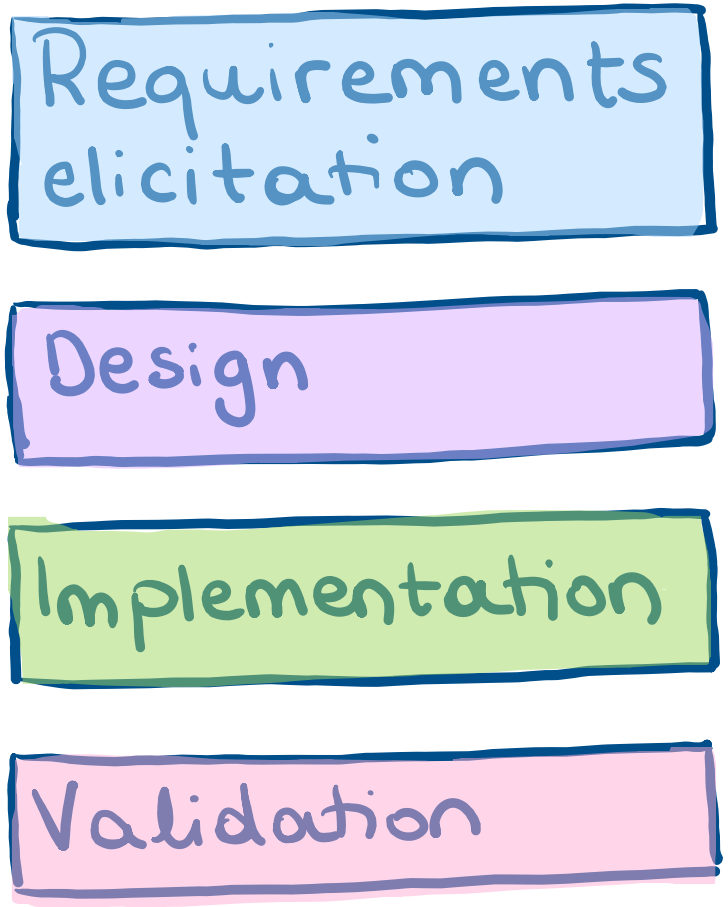
The software development life cycle (SDLC)

Definition:

The SDLC is the process of producing software through a series of stages

- It spans from conception to end-of-life
- Its duration varies from months to years

Life-cycle stages



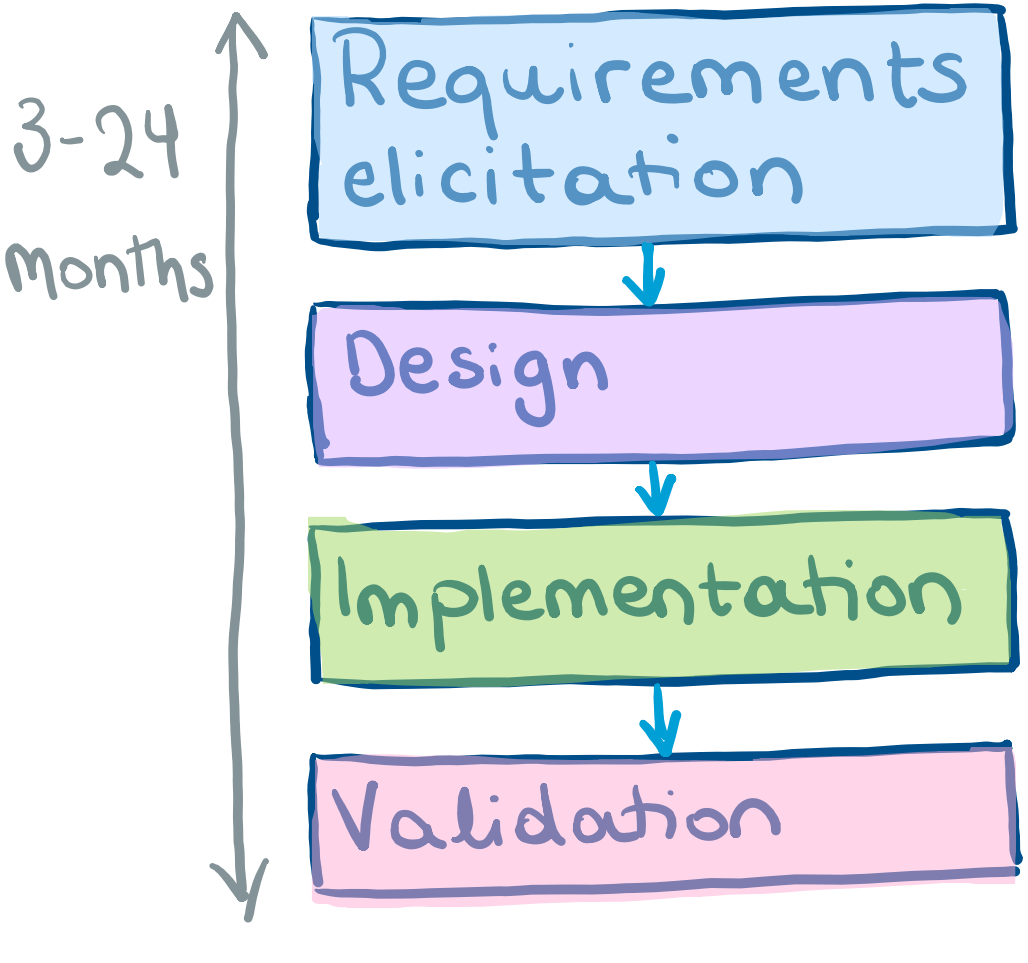
All SDLC models have these stages

- How to combine the stages?
- How quickly to go through the stages?

Main SDLC models

- Traditional models
  - Waterfall model
  - Spiral model
  - …
- Agile models
  - XP (Extreme Programming)
  - Scrum
  - …

Waterfall model



- Top-down approach
- Sequential, non-overlapping stages
- Each stage is finished and frozen
- No backsteps to correct mistakes

Waterfall model: Pros and cons

\+ Easy-to-follow, sequential model

\+ Works well for well-defined projects (requirements are clear)

\- Hard to do all the planning upfront

\- Final product may not match the client's needs

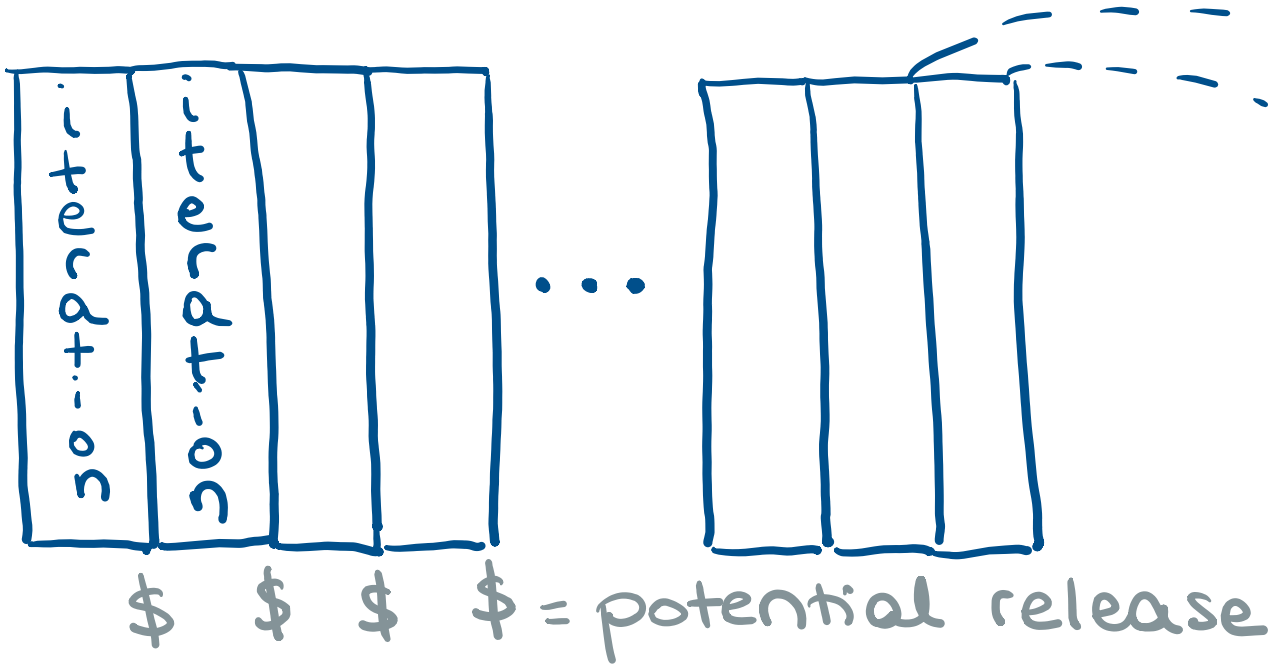
Agile models

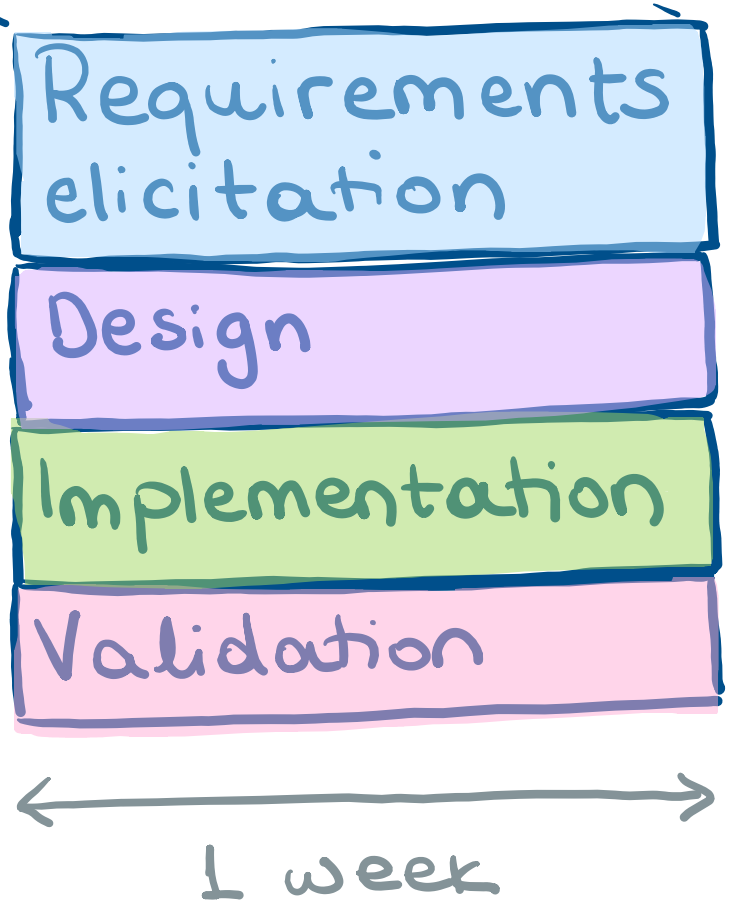
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

While there is value in the items on the right,
items on the left are valued more

From the Agile Manifesto
https://agilemanifesto.org

XP





- Stages happen simultaneously
- Adaptation and re-prioritization of requirements
- Roles are frequently changed

XP: Pros and cons

+ Flexibility (changes are expected)
+ Focus on quality (continuous testing)
+ Focus on communication (continuous customer involvement)

− Requires experienced managers and highly skilled engineers
− Prioritizing requirements can be difficult with multiple stakeholders
− Best for small to medium projects

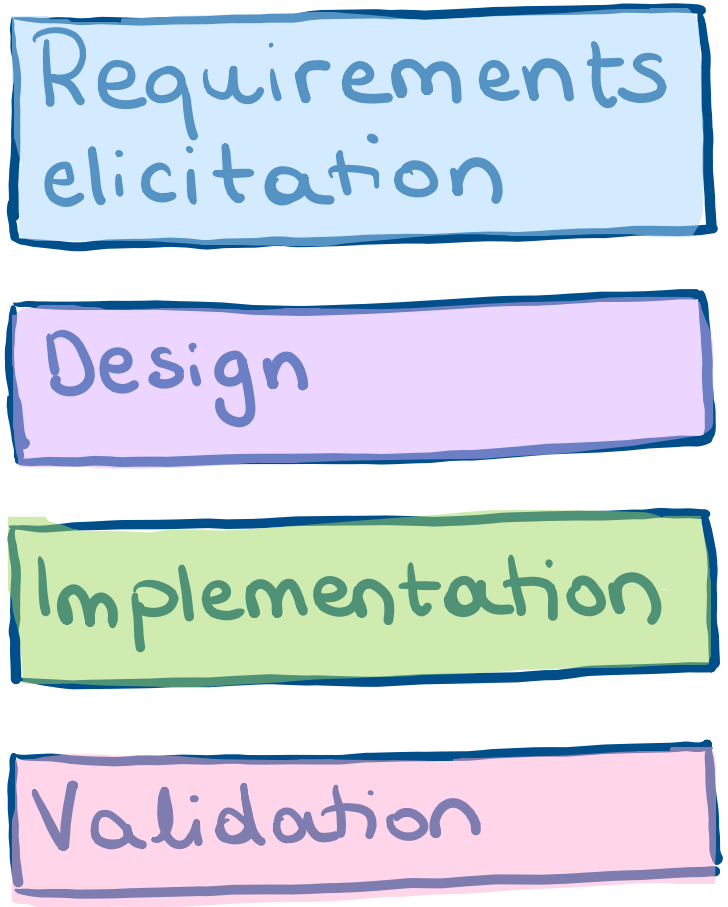
SDLC models: Summary

- All models have the same goals
  - Manage risks and produce high quality software
- All models basically involve the same stages
  - E.g., requirements elicitation, design, implementation, and validation
- All models have advantages and disadvantages

Requirements elicitation

- Requirements
- Activities

Life-cycle stages



All SDLC models have these stages

Requirements

Definition:

Features that the system must have or constraints that the system must satisfy to be accepted by the client

Requirements engineering defines the requirements of the system under construction

Requirements

- Describe the user's view of the system
- Identify the what of the system, not the how

PART OF REQUIREMENTS	NOT PART OF REQUIREMENTS
- Functionality - User interaction - Error handling - External interfaces	- System design - Implementation technology - Development methodology

Types of requirements

FUNCTIONAL REQUIREMENTS

- Functionality
  - What is the software supposed to do?
- External interfaces
  - Interaction with people, hardware, other software

Types of requirements

NON-FUNCTIONAL REQUIREMENTS

- Performance
  - Speed, availability, response time, recovery time
- Design constraints
  - Required standards, operating environment, etc.
- Quality criteria
  - Portability, correctness, maintainability, security

Types of requirements

FUNCTIONAL REQUIREMENTS

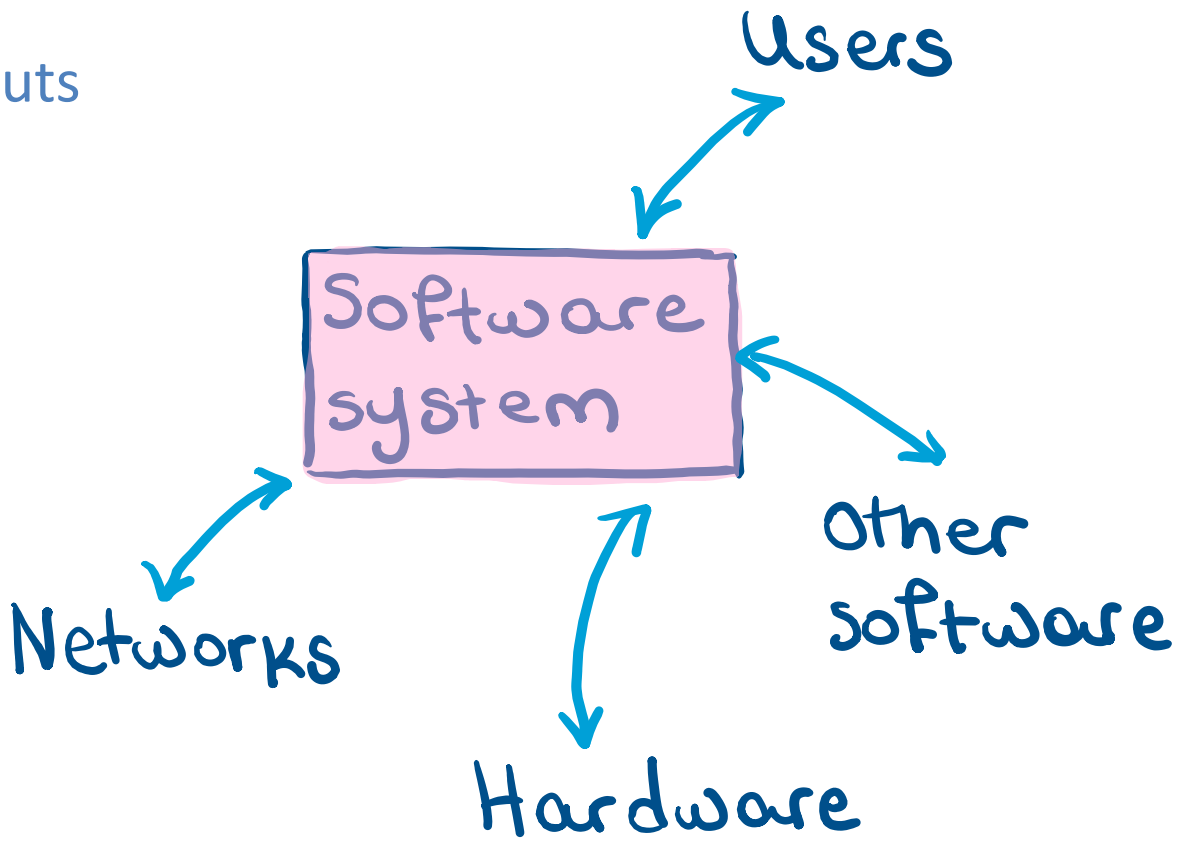
- Functionality
  - What is the software supposed to do?
- External interfaces
  - Interaction with people, hardware, other software

Functional requirements: Functionality

- Relationship of outputs to inputs
- Response to abnormal situations
- Exact sequence of operations
- Validity checks on the inputs
- Effect of parameters

Functional requirements: External interfaces

- Detailed description of all inputs and outputs
  - Description of purpose
  - Source of input, destination of output
  - Valid range, accuracy, tolerance
  - Units of measure
  - Relationships to other inputs and outputs
  - Screen and window formats
  - Data and command formats



Types of requirements

NON-FUNCTIONAL REQUIREMENTS

- Performance
  - Speed, availability, response time, recovery time
- Design constraints
  - Required standards, operating environment, etc.
- Quality criteria
  - Portability, correctness, maintainability, security

Non-functional requirements: Performance

- Static numerical requirements
  - Number of terminals supported
  - Number of simultaneous users supported
  - Amount of information handled

- Dynamic numerical requirements
  - Number of requests processed within certain time periods

  (average and peak workload)

  Example: 95% of the requests shall be processed in less than 1 second

Non-functional requirements: Design constraints

- Standard compliance
  - Report format, etc.
- Implementation requirements
  - Tools, programming languages, etc.
  - Development technology and methodology should not be constrained by the client
- Operation requirements
  - Administration and management of the system
- Legal requirements
  - Licensing, regulation, certification

Non-functional requirements: Quality criteria

Correctness

Requirements represent the client's view

Completeness

All possible scenarios are described including exceptional behavior

Consistency

Requirements do not contradict each other

Clarity (Un-ambiguity)

Requirements can be interpreted in only one way

Non-functional requirements: Quality criteria

Realism

Requirements can be implemented and delivered

Verifiability

Repeatable tests can be designed to show that the system fulfills the requirements

Traceability

Each feature can be traced to a set of functional requirements

Quality criteria: Examples

- “The system shall be usable by elderly people”
  - QUIZ

Quality criteria: Examples

- “The system shall be usable by elderly people”
  - Not verifiable, unclear
  - Solution: “Text shall appear in letters at least 1cm high”

Quality criteria: Examples

- “The system shall be usable by elderly people”
  - Not verifiable, unclear
  - Solution: “Text shall appear in letters at least 1cm high”

- “The product shall be error-free”
  - Not verifiable, not realistic
  - Solution: Specify test criteria

Quality criteria: Examples

- “The system shall be usable by elderly people”
  - Not verifiable, unclear
  - Solution: “Text shall appear in letters at least 1cm high”
- “The product shall be error-free”
  - Not verifiable, not realistic
  - Solution: Specify test criteria
- “The system shall provide real-time response”
  - QUIZ

Quality criteria: Examples

- “The system shall be usable by elderly people”
  - Not verifiable, unclear
  - Solution: “Text shall appear in letters at least 1cm high”
- “The product shall be error-free”
  - Not verifiable, not realistic
  - Solution: Specify test criteria
- “The system shall provide real-time response”
  - Unclear
  - Solution: “The system shall respond in less than 2s”

Requirements validation

- A quality assurance step, usually after requirements are gathered

- Reviews by clients and developers
  - Check all quality criteria
  - Discuss future validation (testing)

- Prototyping
  - Produce throw-away or evolutionary prototypes
  - Study feasibility
  - Give clients an impression of the future system (e.g., user interfaces)

Requirements elicitation

- Requirements
- Activities

Requirements-elicitation activities

Identifying actors

Identifying scenarios

Identifying use cases

Identifying non-functional requirements

Identifying actors

- Actors represent roles
  - Kind of user
  - External system
  - Physical environment

- Questions to ask
  - Which user groups are supported by the system?
  - Which user groups execute the system's main functions?
  - Which user groups perform secondary functions (e.g., maintenance, administration)?
  - With what external hardware and software will the system interact?

Scenarios and use cases

- Document the behavior of the system from the user's point of view
- Can be understood by client and users

Scenario

- Describes common cases
- Focus on understandability

Use case

- Generalizes scenarios to describe all possible cases
- Focus on completeness

Scenario

Definition:

A description of what people do and experience as they use the system

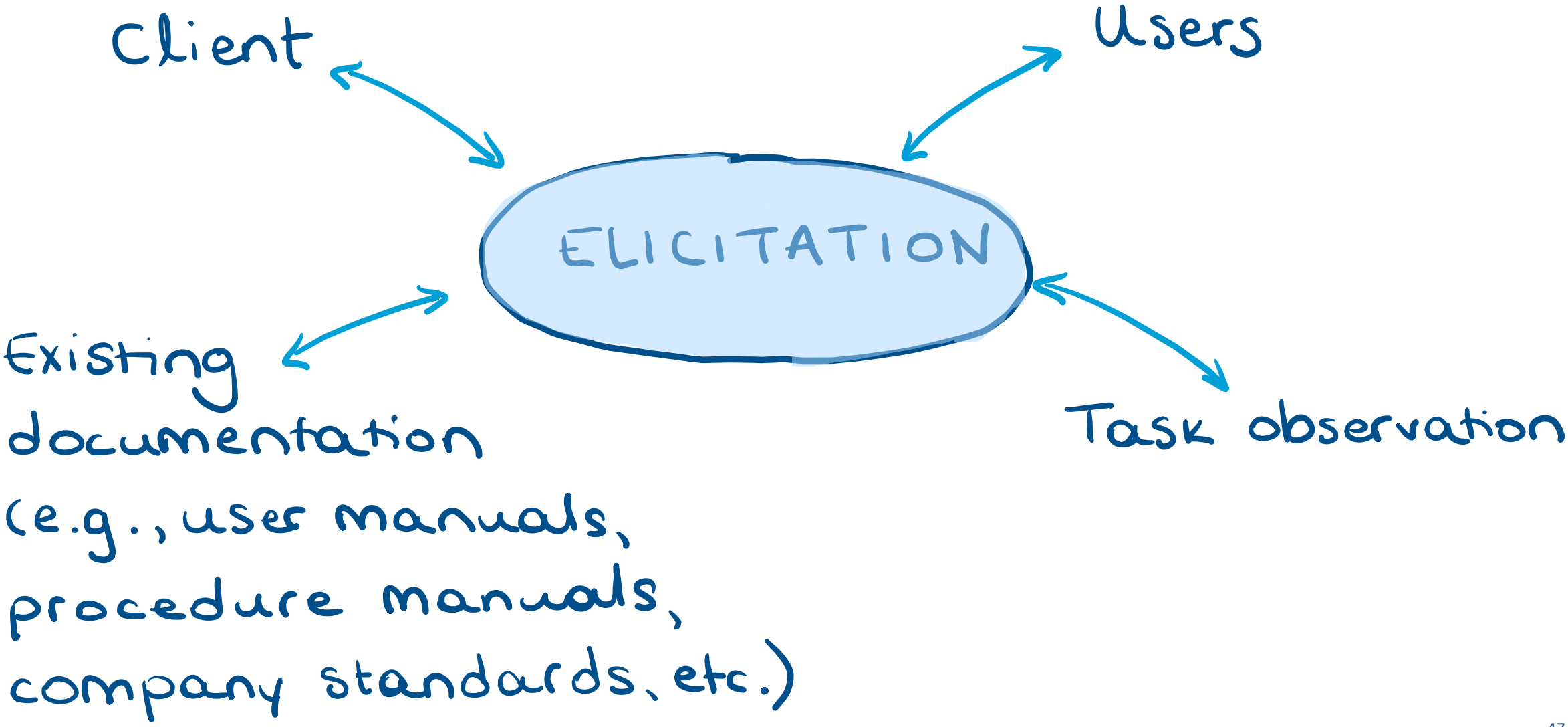
Example:

When Alice wants to borrow a book, she takes it to the checkout station. There, she first scans her library card. Then, she scans the barcode of the book. If she has no borrowed books that are overdue and the book is not reserved for another person, the system registers the book as being borrowed by Alice and…

Identifying scenarios: Questions to ask

- What are the tasks the actor wants the system to perform?
- What information does the actor access?
- Which external changes does the actor need to inform the system about?
- Which events does the system need to inform the actor about?

Sources of information



Use case

Definition:

A list of steps describing the interaction between an actor and the system,

to achieve a goal

Example:

Actor steps	System steps
1. Scans library card	2. Validates card; returns card; displays user data; displays 'Select function' dialog
3. Selects 'Borrow'	4. Displays 'Borrow' dialog
5. Scans barcode of book to be borrowed	6. Identifies book; records book as borrowed...

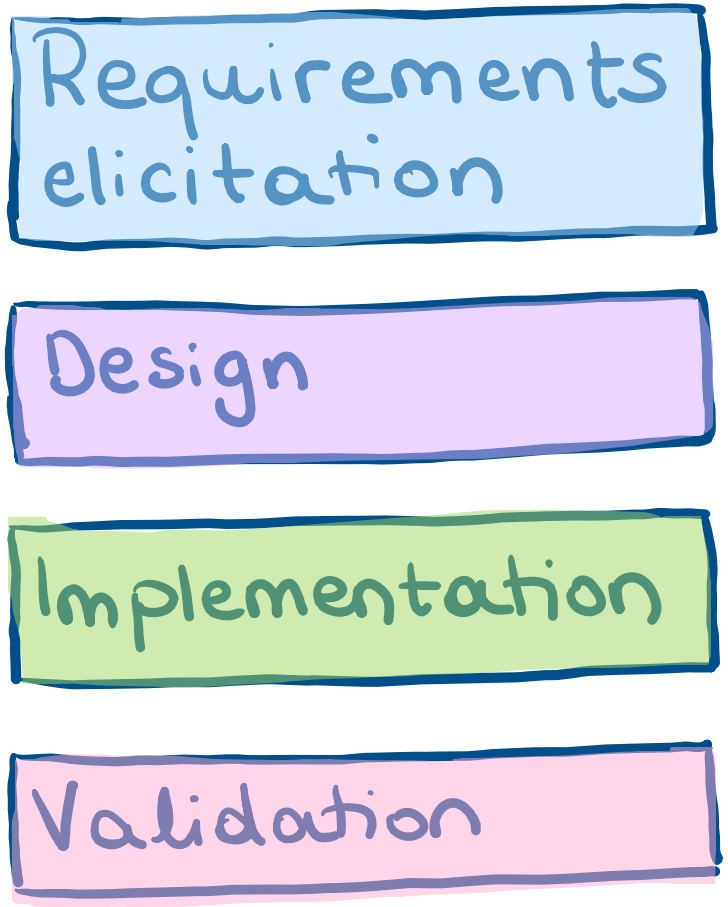
Identifying non-functional requirements

- Non-functional requirements are defined together with functional requirements because of dependencies
  - Example: Support for novice users requires help functionality
- Elicitation is typically done with check lists
- Resulting set of non-functional requirements typically contains conflicts
  - Real-time requirement suggests C implementation
  - Maintainability suggests OO implementation

Design

- UML models (next lecture)
- Formal models (in November)
- Design by contract (in November)

Life-cycle stages



All SDLC models have these stages

Design specifications

- Source code provides limited support for leaving design choices unspecified
  - Often because code is executable
- Some relevant design information is not represented in the program, or it is difficult to extract
  - Tools can extract some information like control or data flow graphs
  - Source code and documentation are too verbose
- Design specifications are models of the system providing suitable abstractions

What is modeling?

- Building an abstraction of reality
  - Abstractions from things, people, and processes
  - Relationships between these abstractions
- Abstractions are simplifications
  - They ignore irrelevant details
  - What is relevant depends on the purpose of the model
- Modeling is a means for dealing with complexity
  - They help to draw complicated conclusions in reality with simple steps in the model