Models and Methodologies

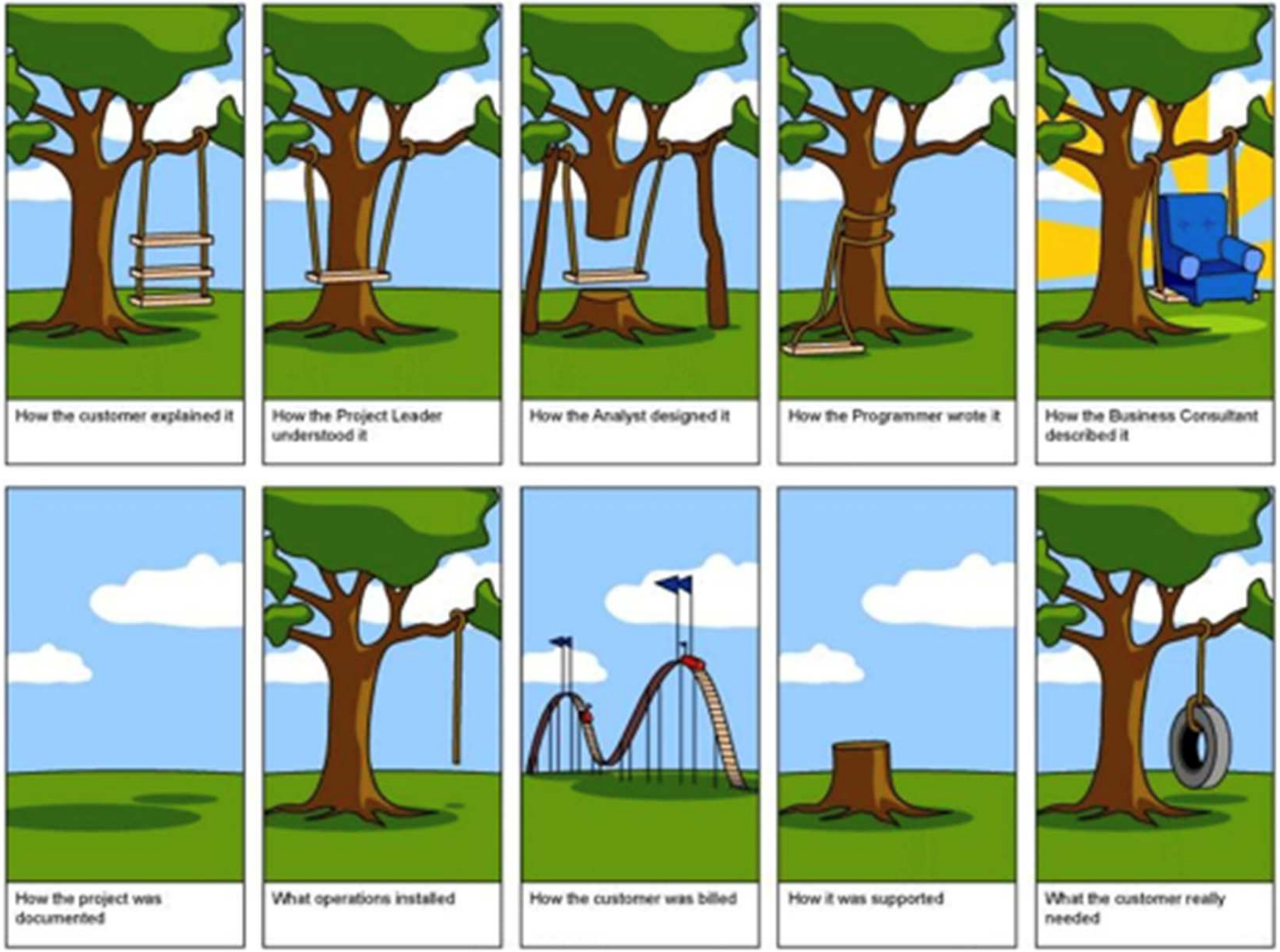
Methodologies models and examples of methodologies

Models

Model of development process

- A **development process** is a structured set of **activities** (phases) needed to develop a software system
  - Specification
  - Design
  - Implementation
  - Test and validation
  - Maintenance and evolution
- A **model** of the development process is an **abstract representation** of the process

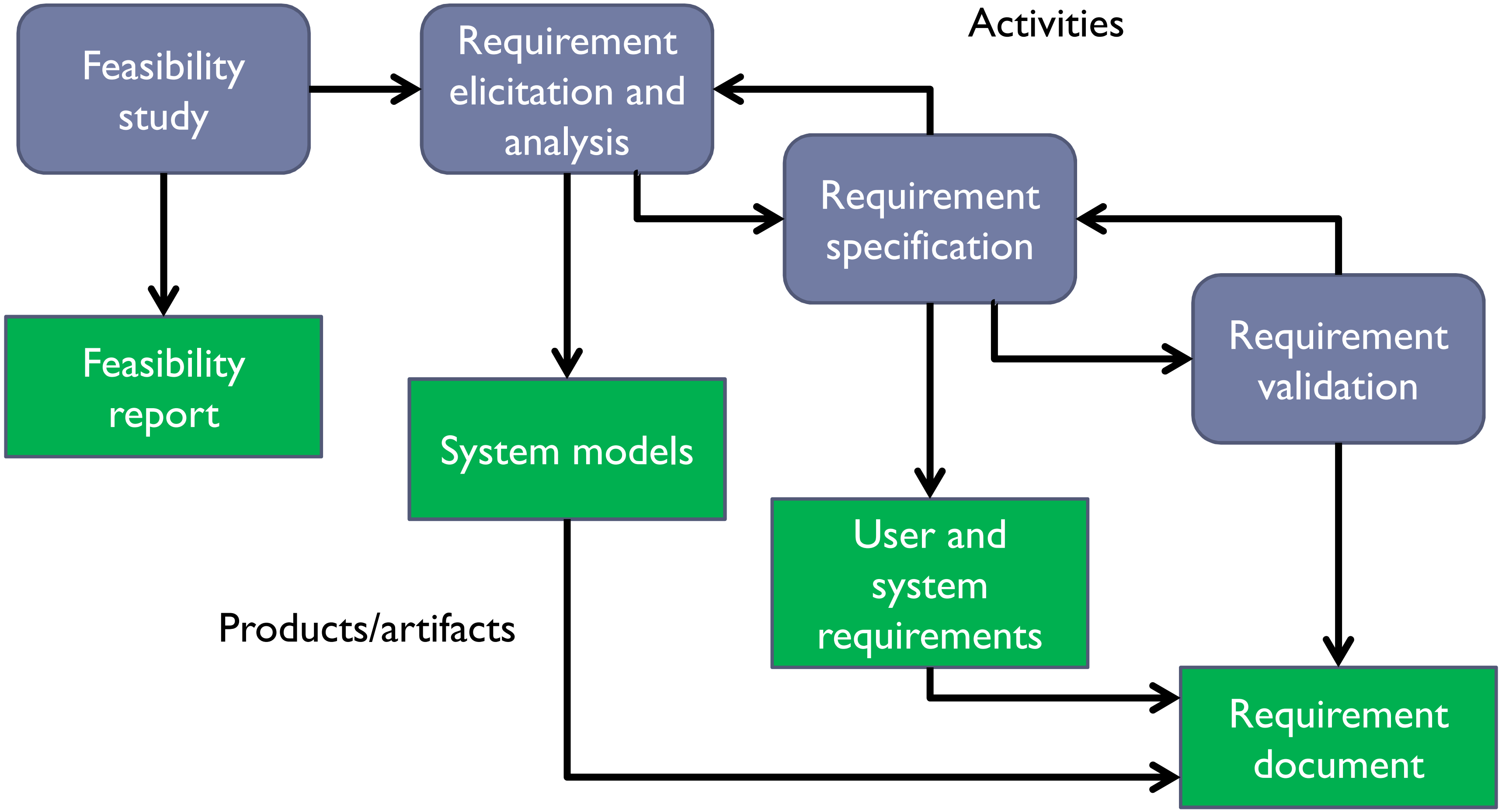
Developing is not easy…


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Specification

- Specification is an activity that aims at defining:
  - The **requirements** (explicit or implicit) of the customer
  - The **constraints** (explicit or implicit) of the system and of its development
- Requirement engineering process
  - **Feasibility** study
  - **Elicitation** (extraction) and **analysis** of the requirements
  - **Specification** of the requirements
  - **Validation** of the requirements

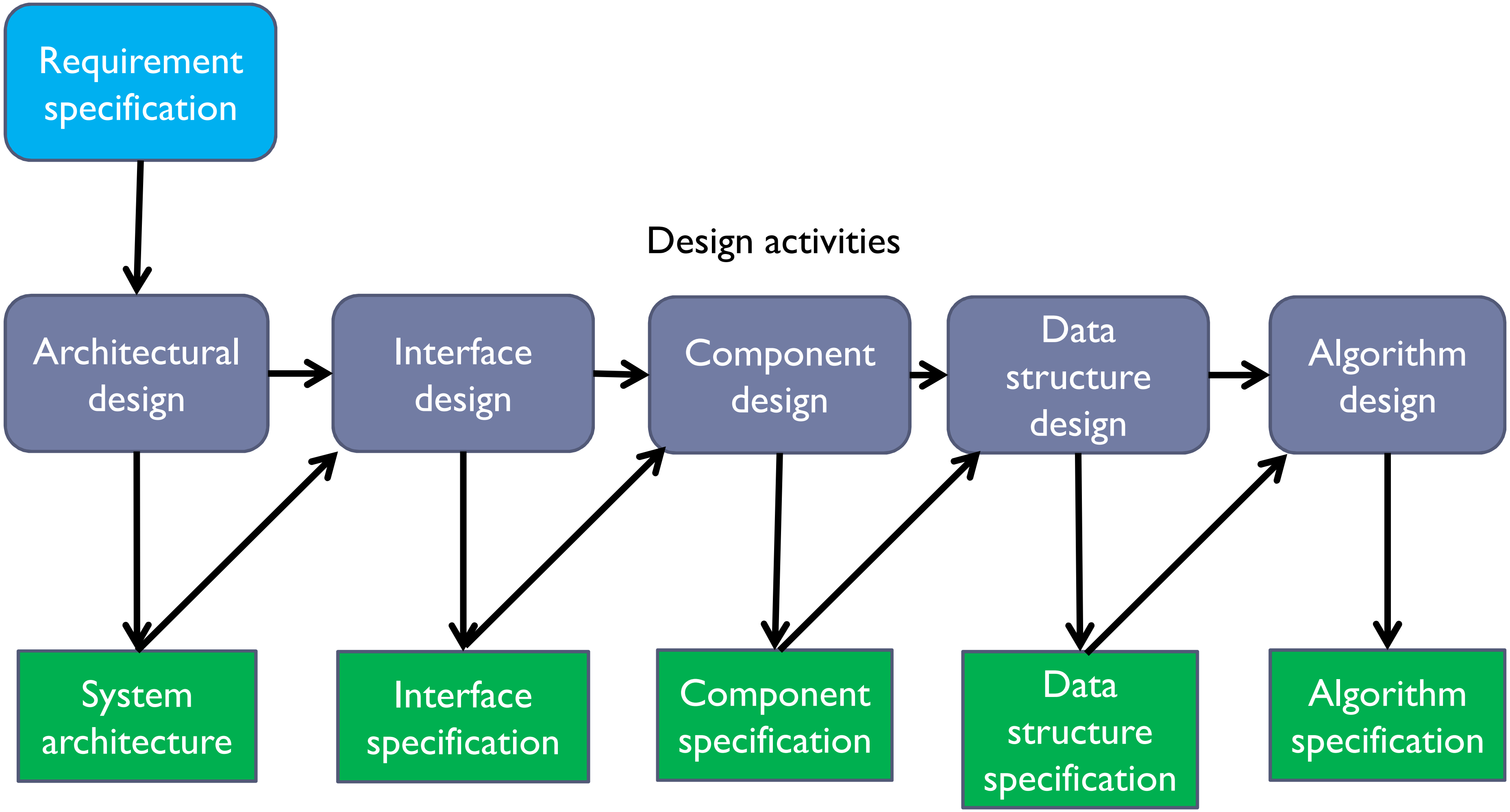
Requirement engineering process



Design and Implementation

- Process of converting from the system specification to a working system
- Design
  - Design of the structure (**architecture**) that realizes the specification at different levels
- Implementation
  - Translation of the design products into executable code
- Design and implementation are strongly **related** and (some time) **interleaved**
- But we must spend an effort to keep them **separated**

Design process



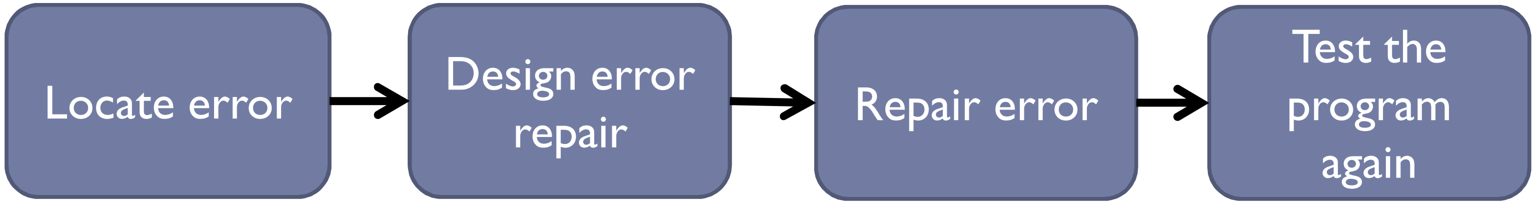
Design products/artifacts

Design methodologies

- **Systematic** approach to the development of a software project
- The project is typically documented by means of **graphic models**
- Examples
  - Data-Flow Diagram (DFD)
  - Entity-Relation diagram (ER)
  - UML diagrams
    - class diagram
    - interaction diagram

Implementation and testing (debugging)

- Transformation of a design into a **program** and cleaning from the coding **errors**
- Each programmer **tests** the **units** she is developing
  - Or, better, the tests are in charge of testers different from the programmers



Verification and validation

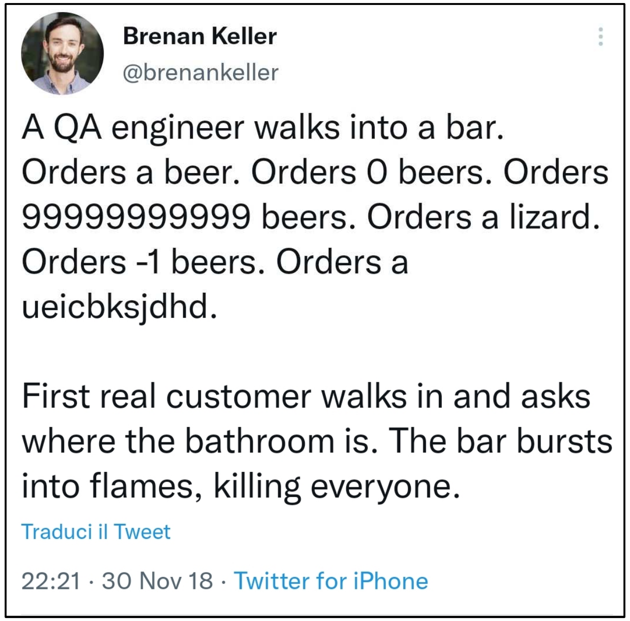
- The verification and validation aim at showing that the developed system:
  - Is **compliant** to the specification
  - Satisfies the customer **requirements**
- **Review** and **testing** of the system
- The testing requires to execute the system on some **test cases** derived from the specifications

Testing

- What is testing useful for?
  - *Testing shows the presence, not the absence of bugs* [E. Dijkstra]
  - Testing is useful to verify the behavior of the system in a **set of cases** enough wide to make it plausible that its behavior is analogous in the **other** situations
- The testing operation can be divided into:
  - Testing in the **small**, addressing single modules/units
  - Testing in the **large**, addressing the whole system

Testing: be careful!

- *A QA engineer walks into a bar. Orders a beer. Orders 0 beers. Orders 99999999999 beers. Orders a lizard. Orders -1 beers. Orders a ueicbksjdhd. First real customer walks in and asks where the bathroom is. The bar bursts into flames, killing everyone.* [Brenan Keller]



Brenan Keller
@brenankeller

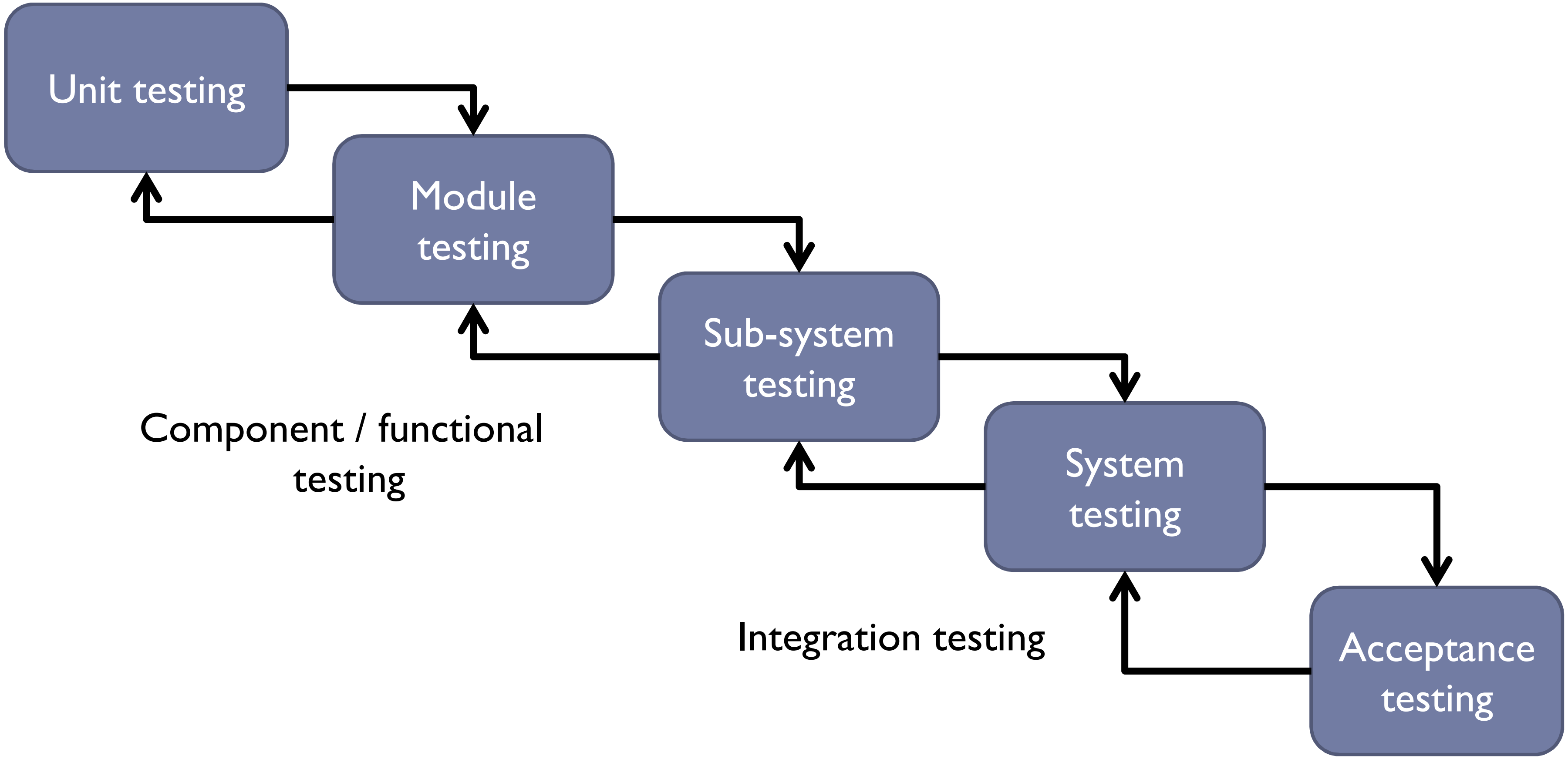
A QA engineer walks into a bar. Orders a beer. Orders 0 beers. Orders 99999999999 beers. Orders a lizard. Orders -1 beers. Orders a ueicbksjdhd.

First real customer walks in and asks where the bathroom is. The bar bursts into flames, killing everyone.

Traduci il Tweet

22:21 · 30 Nov 18 · Twitter for iPhone

Testing process



Component / functional testing

Integration testing

User testing

Testing activities

- Unit testing
  - testing of a **single** component
- Module testing
  - testing of each module, **set** of inter-dependent components

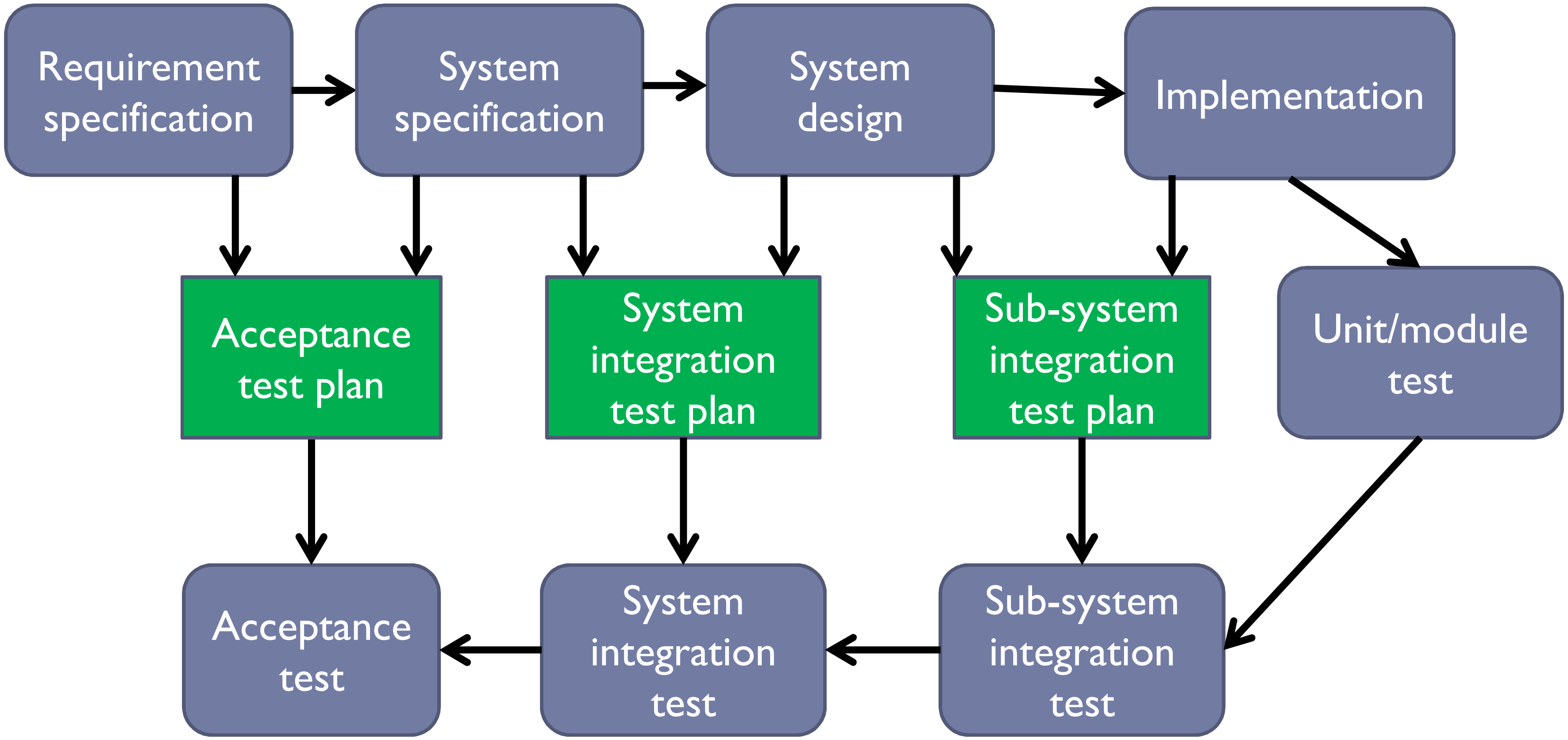
Component / functional test

- Sub-system testing
  - testing of the integration among modules into **sub systems.** The aim is to find **interface** problems
- System testing
  - testing of the system as a **whole.** Testing of possible **emerging** properties of the system

Integration test

- Acceptance testing
  - testing with the **customers** to verify the **acceptance** of the system
- Other tests
  - **regression** test, **performance** test

Testing activities (2)



Testing in the small

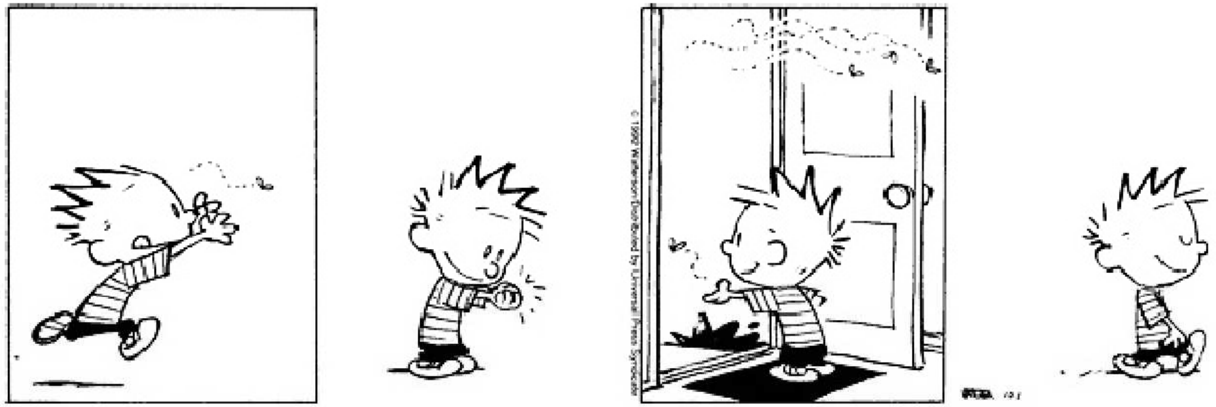
- These tests aims at verify the correctness of **code**
- They address **classes** and **methods**
- Given some relevant **input**, the test checks whether the **output** is correct
- All code parts must be covered
  - **Coverage** test
  - Address **branches**!
- **White-box** testing: the test "knows" what is inside a unit
- Code inspection
  - The code is analyzed to understand the properties and the functionalities

Testing in the large

- When components and modules are integrated, it is **impossible** to perform a test on the single code instructions
- So, the behavior of the system is tested against the **specifications**
- **Black-box** testing: the test "does not know" what is inside a unit

Regression test

- When a bug is fixed, the modifications can impact other parts of the code
- New bugs can be introduced
- The regression test concerns testing functionalities not modified and already tested



checkpoint
TECHNOLOGIES
Software Quality. Assured.

Process models

Generic development process models

- **Waterfall** model
  - Sharply separates the different phases
- **Component-based** model
  - The system is developed assembling available components
- **Evolutionary** model
  - Specifications and development are interleaved
- **Incremental** model
  - The system is developed by incremental steps

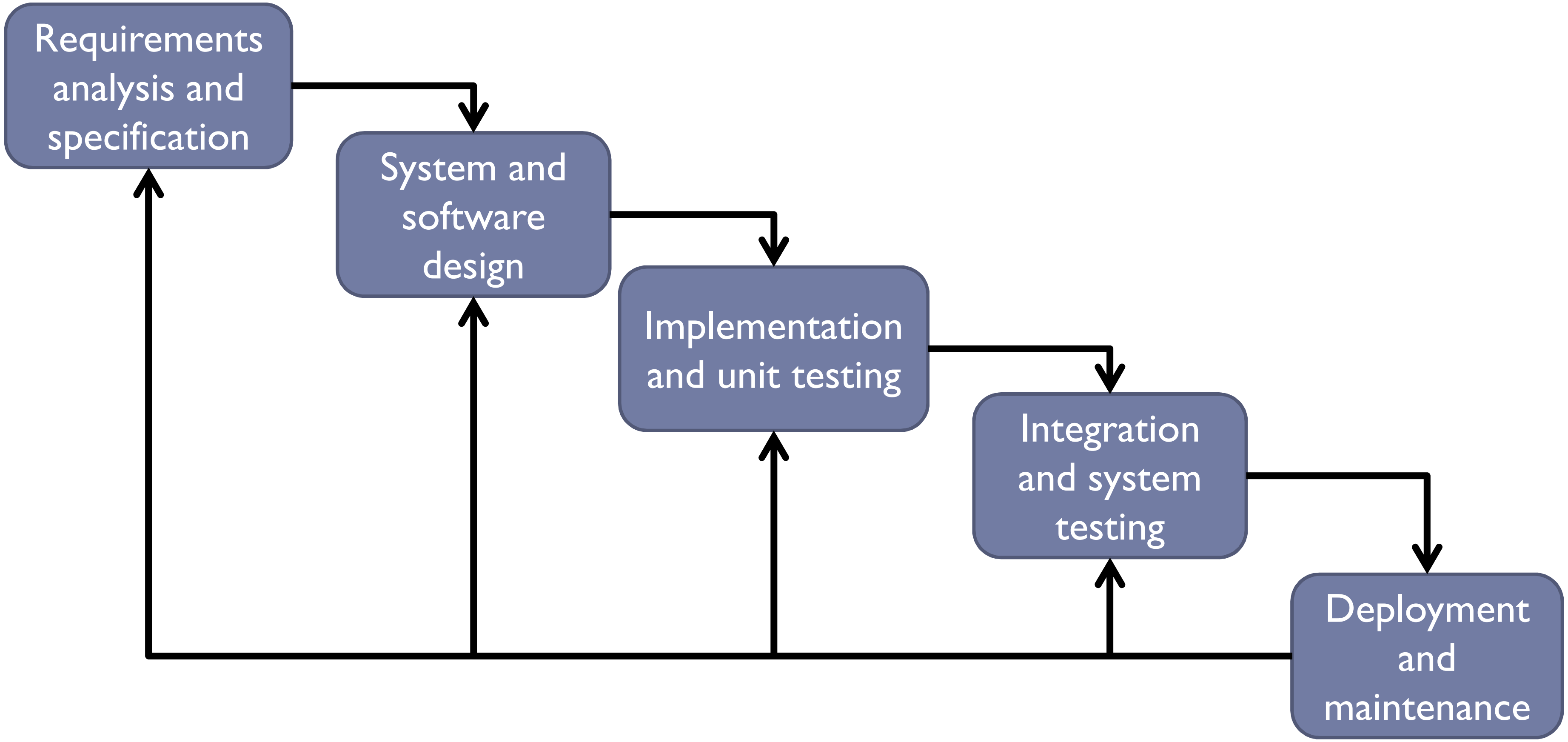
Iterative models

- **Transformational** model
  - The steps of the development transform the formal specifications into implementation

Code and fix

- Two phases: "write the code" and "adjust it"
  - Phases are iterated
- Widely exploited
- But **not** a real model
- Programmers **seem** to save time
  - Actually, it is very **difficult** to satisfy the customer
  - Each (small) modification requires a **lot of work**
- Suitable for **very small** projects
- Important note: this is very **different** from the agile processes presented later
  - No design, no plan, no schedule, no collaboration, …

Waterfall model



Requirements analysis and specification

System and software design

Implementation and unit testing

Integration and system testing

Deployment and maintenance

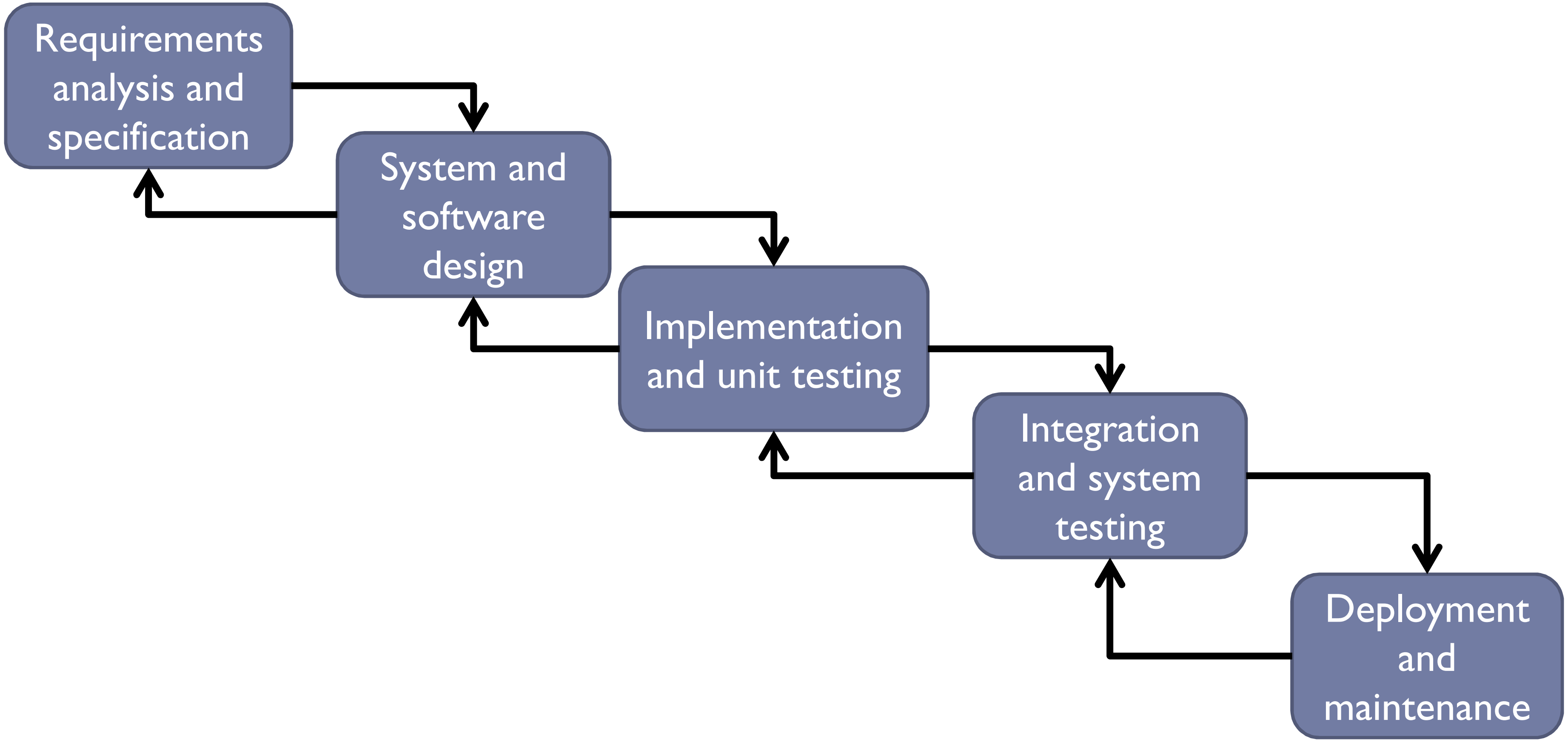
Phases in the waterfall model

- Requirements analysis and specification
  - Understanding of what the customers want
  - Formalization of the requirements
- System and software design
  - Construction of a model of the system to be developed
- Implementation and unit testing
  - Coding of the components and their test
- Integration and system testing
  - Integration of the developed components and testing of the system
- Deployment and maintenance
  - The system is made operative
  - The needed modifications are performed

Pros and cons of the waterfall model

- Pros
  - Well-defined process
  - Time can be estimated with good precision
  - The process is well-documented
  - Good for known kinds of systems
- Cons
  - Rigid process
  - Difficult to manage changes
    - In particular, of requirements
  - No flexibility in the phases

Waterfall model with feedback



Requirements analysis and specification

System and software design

Implementation and unit testing

Integration and system testing

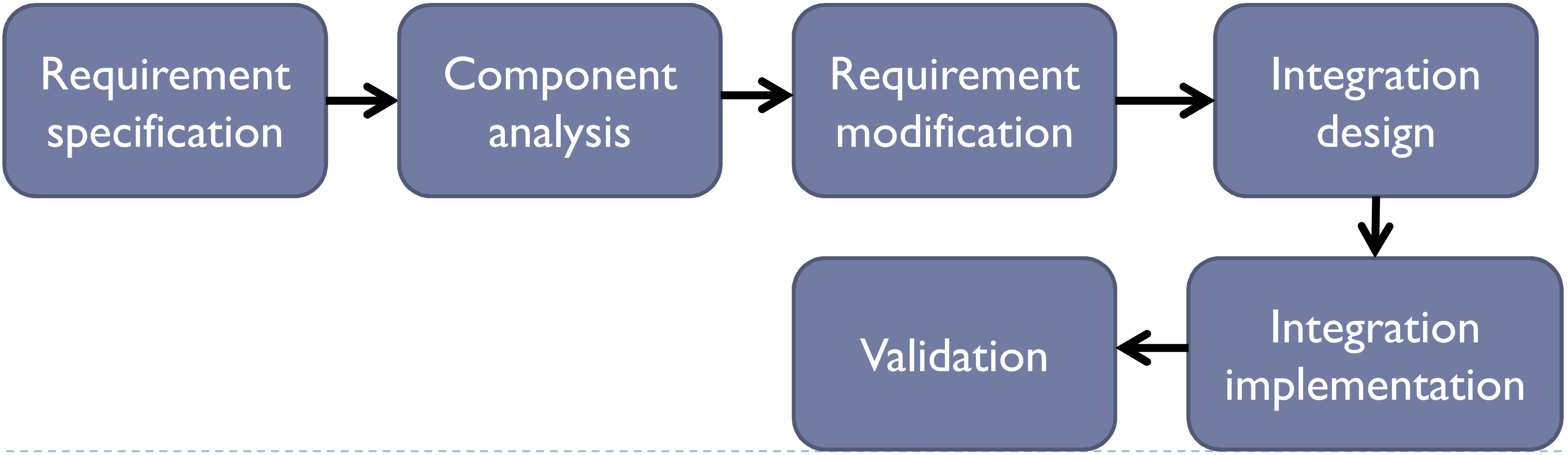
Deployment and maintenance

Component-based model

- Based on the **reuse** of software components
  - Developed **in-house**
  - **Commercial-Off-The-Shelf** (COTS)
- The system is built by assembling the **available** components
  - **Glue** code
- Exploits **OOP**
- Many components **models** exist
  - E.g., SOM, OLE, COM, DCOM, CORBA, EJB
- This approach is very **interesting** even if it is **not** so exploited

Component-based phases

- Requirements specification
- Analysis of the available components
- Requirements modification to fit available components
- Design of the component integration
- Implementation of the integration
- Validation



Pros and cons of the component model

- Pros
  - No need for reinventing the wheel
  - Tested components

- Cons
  - Requirements must be adapted to available components
  - Truly reusable components are difficult to be developed
  - “not invented here” syndrome
  - Sometimes, glue code is harder than starting from scratch

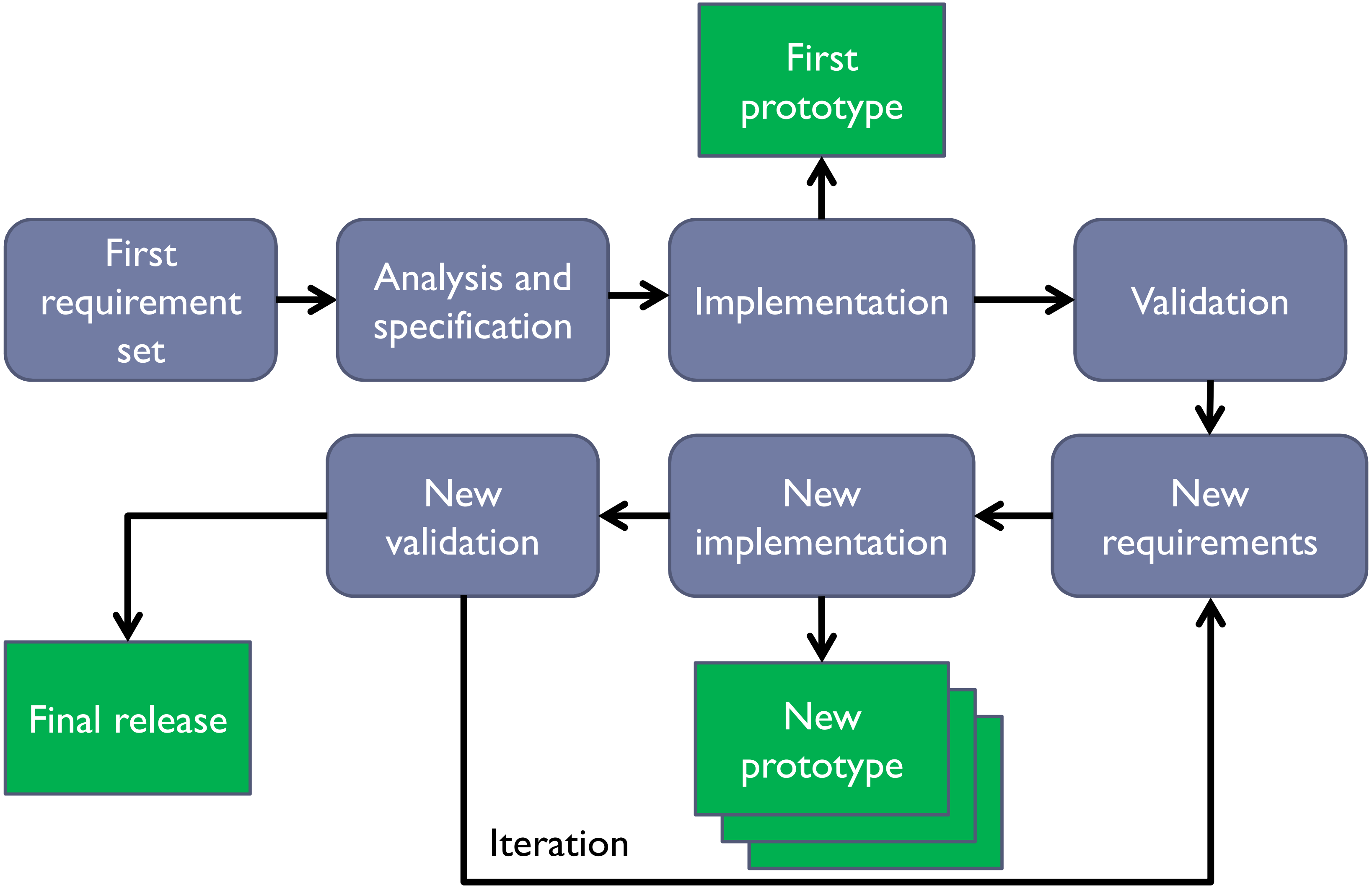
Iterative models

- In the previous models the phases were **sequential**
- This make them **rigid**
- Often, the development environment **changes**
  - Change of **requirements**
  - Change of **understandings**
  - Change of **conditions**
- In this case, it is useful to perform **again** one or more process phases to review what have been done
  - Considering the new situations
- Iterative models "**iterate**" some/all phases
- We will see 2 iterative models

Evolutionary model

- The system “**evolves**” during the development
- The steps are as follows
  1. A first set of (reasonable) requirements is asked to the customer
  2. A prototype is developed
  3. The prototype is shown to the customer
  4. A new set of requirements (or changes) is asked
  5. A new prototype is developed
  6. The new prototype is shown to the customer
  7. Steps 4 to 6 are iterated until the final version is released
- Requirements can change
- **Throw-away** prototyping
  - Requirements can be discarded during the development

Evolutionary model (2)



First prototype

First requirement set

Analysis and specification

Implementation

Validation

New validation

New implementation

New requirements

Final release

New prototype

Iteration

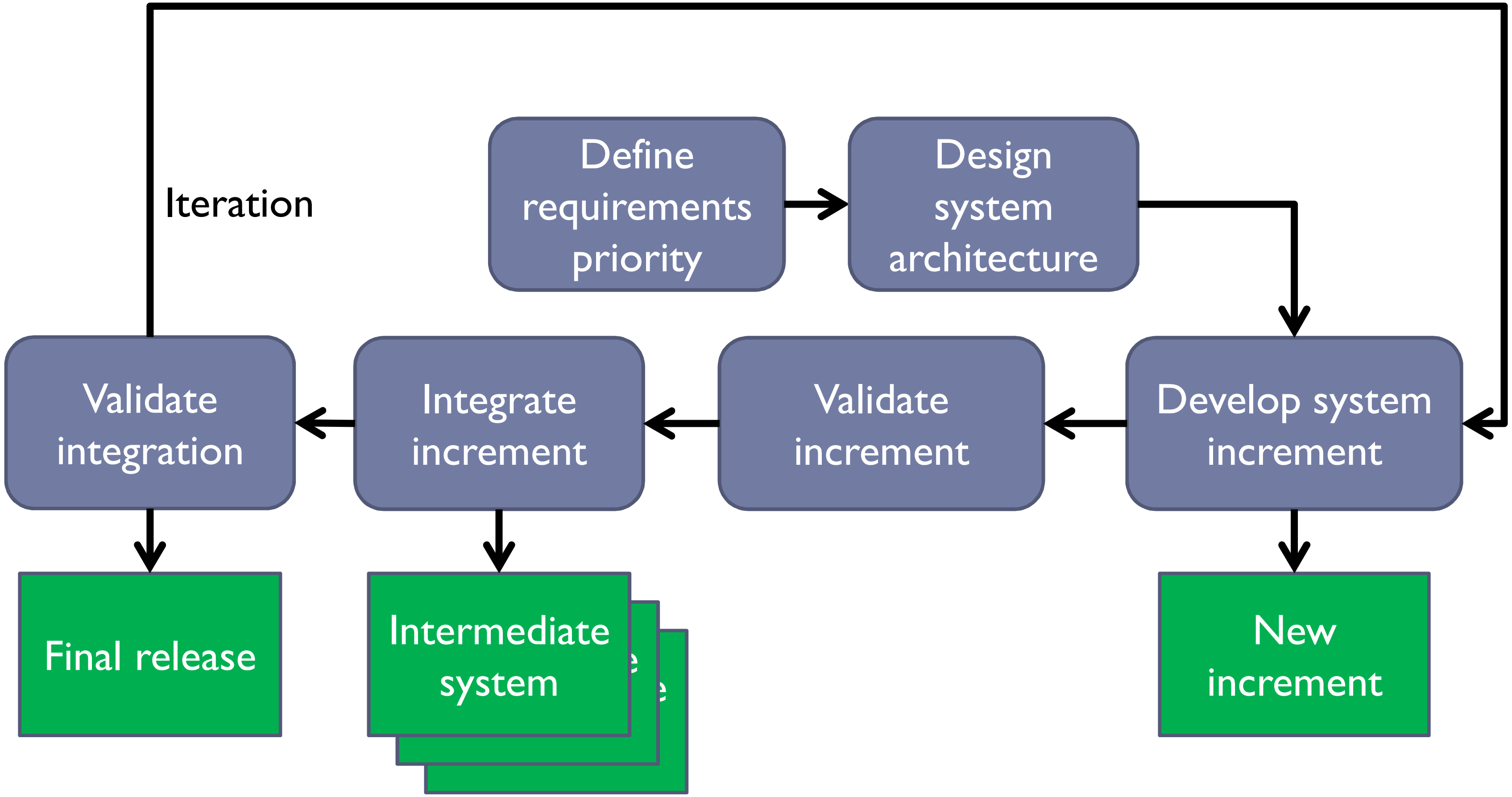
Pros and cons of the evolutionary model

- Pros
  - Flexibility
  - Can adapt to requirement **changes**
- Cons
  - The development is hard to **inspect**
  - It is difficult to estimate the **time** of the delivery of the final release
  - Resulting systems could be **little structured**
- Applicability
  - Parts of large systems
  - Short-life systems
  - Unknown systems
- Requires tools, languages and techniques for rapid prototyping

Incremental model

- The development of the whole system is divided into developments of **increments**
- The steps are as follows
  1. The requirements are **prioritized**
  2. The system architecture is designed
  3. An increment is developed following the priority
  4. The increment is validated
  5. The increment is integrated
  6. The integration is validated
  7. Steps 3 to 6 are iterated until the final version is released
- The developed part of the system is **not** modified
  - Only **increments** are added

Incremental model (2)



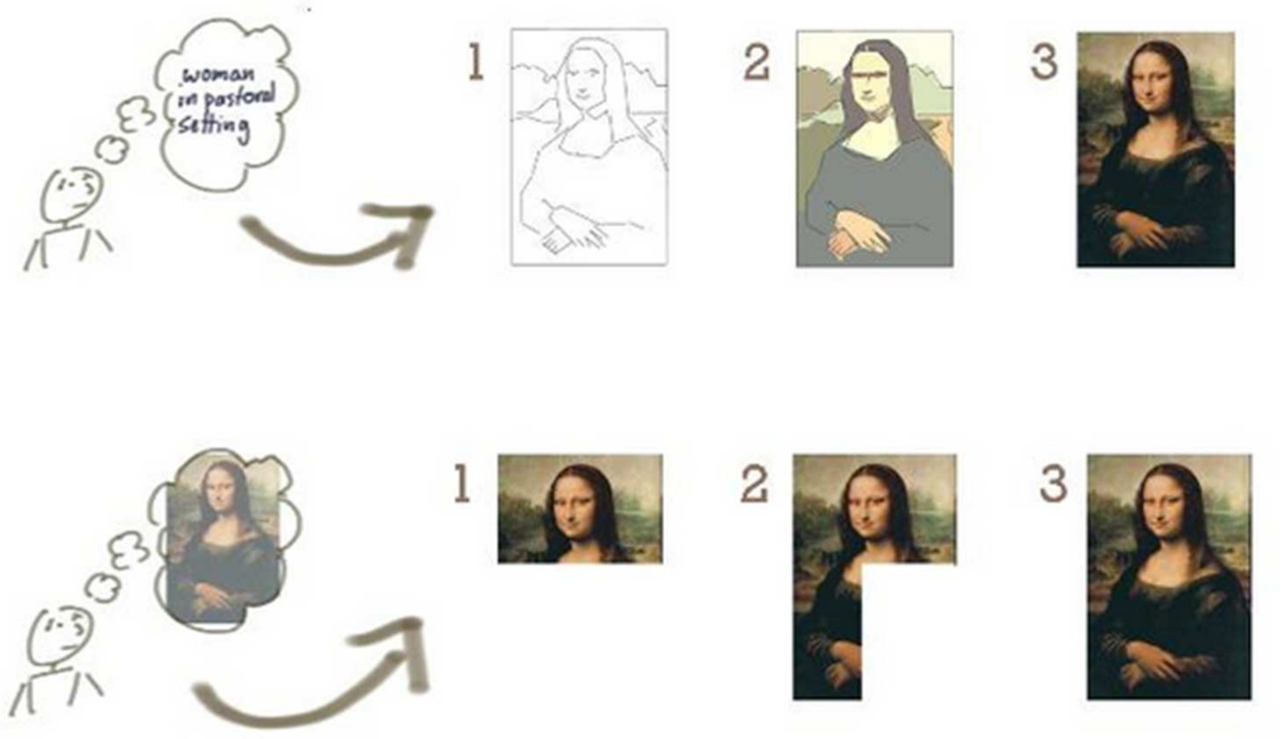
Pros and cons of the incremental model

- Pros
  - The core functionalities are available to the customer **soon**
  - First releases can be exploited as prototypes to let **other** requirements emerge
  - The failure risk is **low**
  - The core functionalities are **tested** more times
    - Because are in the first release and are not changed

- Cons
  - It is difficult to estimate the **number** of increments
  - It is difficult to estimate the **time** of the delivery of the final release

Difference between evolutionary and incremental

- By Jeff Patton



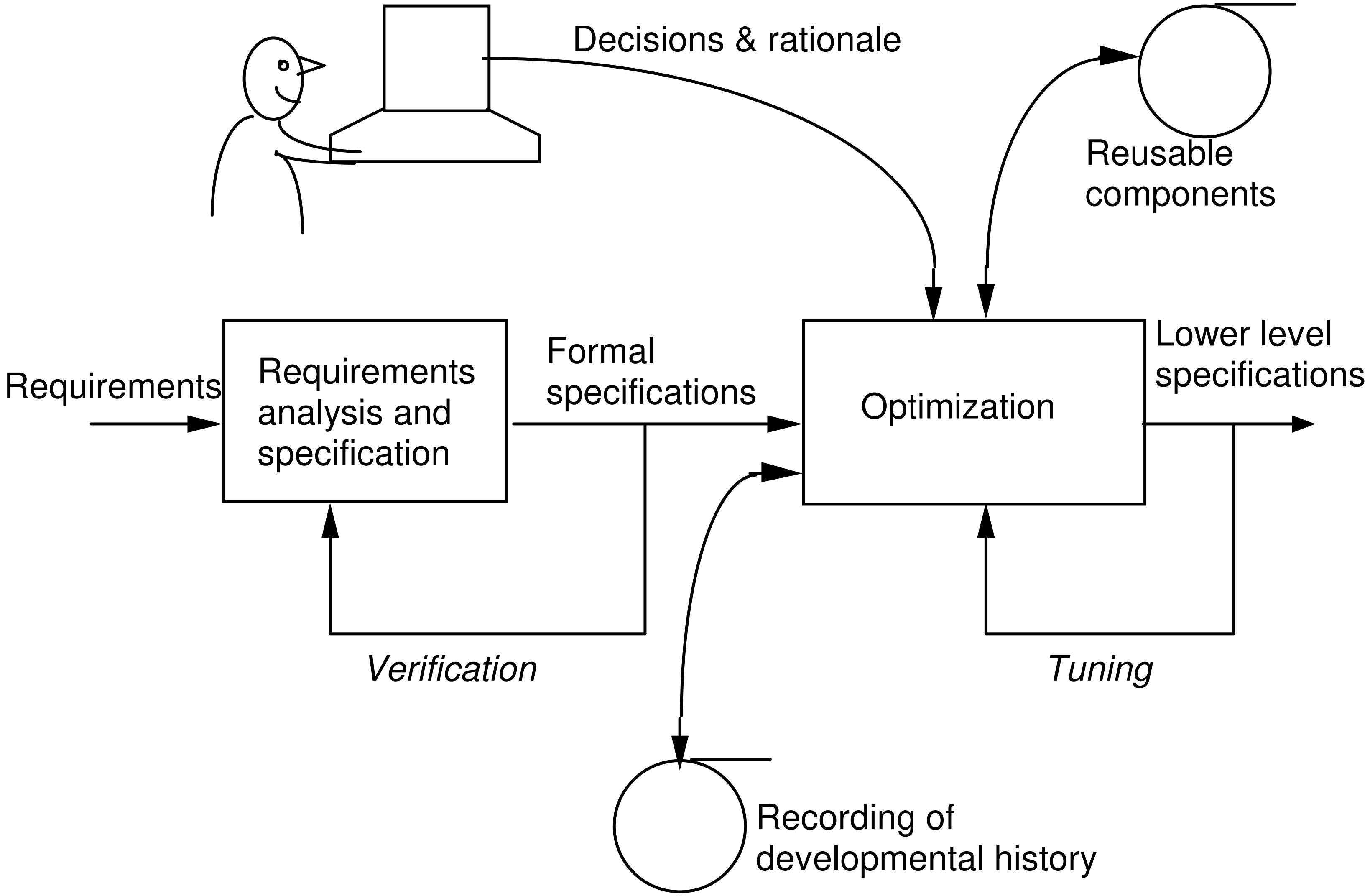
Pros and cons of the iterative models

- Pros
  - Flexibility
    - Changes are dealt with
  - Software prototypes can be presented to the customer early
- Cons
  - Less deterministic about the delivery time
  - More complex to manage
    - Require expertise and experience

Transformational model

- Based on **formal** methods
  - To specify development **information**
  - To transform them into **code**
- From **specification** to **implementation**
- From abstract description of the system to running implementation
  - By means of **formal transformation**
- Still a **theoretical** approach

Transformational model



Pros and cons of the transformational model

- Pros
  - The transformation process is performed in a (almost) **automatic** way
    - **Human** intervention is limited but still needed
  - A **modification** to the specification is automatically enacted on the implementation
  - A priori **correctness checks** can be performed
    - Differently from testing **after** implementation
- Cons
  - Formal methods are very **difficult** to manage
  - Formal methods require **more time** in the first development

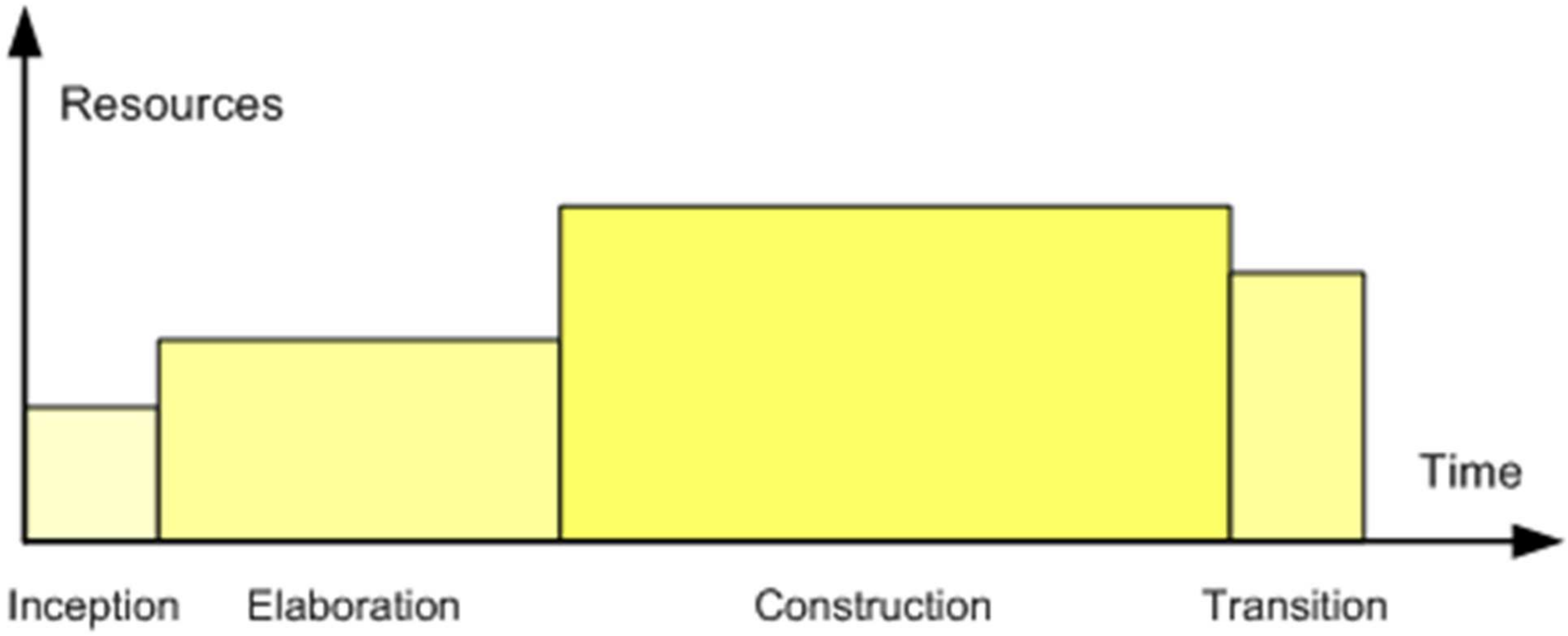
Models comparison

- Waterfall
  - Driven by artifacts and documentation
- Components
  - Driven by components
- Iterative (evolutionary and incremental)
  - Driven by increments
- Transformational
  - Driven by specifications

Models comparison (2)

Model	Pros	Cons
Waterfall	Linear, easy to inspect, good documentation	Rigid
Components	Reuse of existing software	Difficult to adapt components
Iterative	Flexible, early code	Complex, less deterministic
Transformational	Automatic, correctness checks	Difficult, longer

Unified process



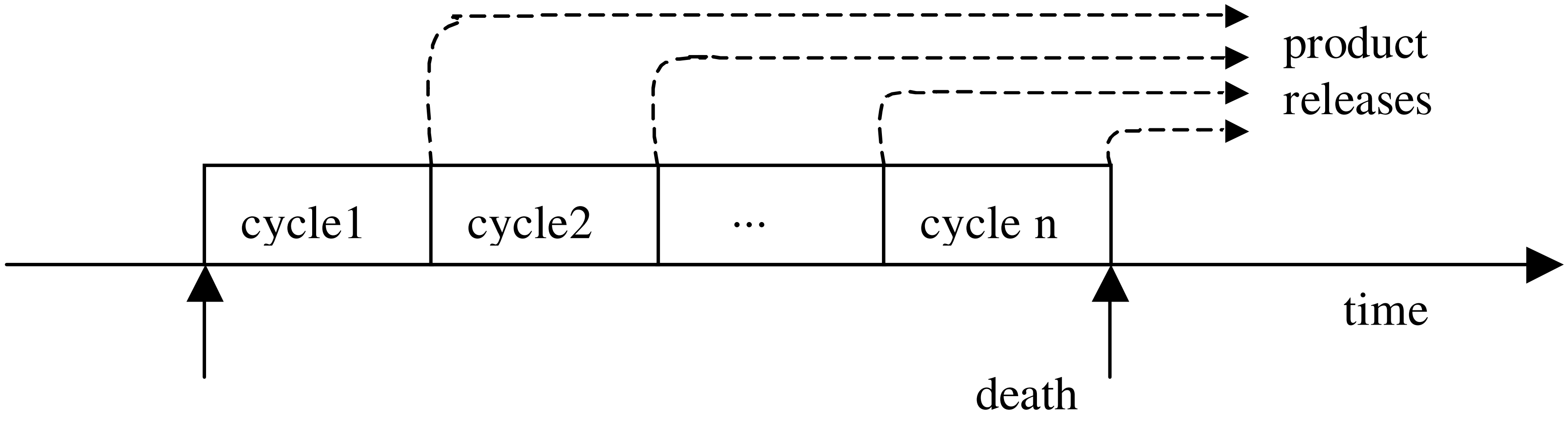
Unified process

- UP
- Born in '60 at Ericsson (telecom company)
- Industrial standard
- Exploits SRS, UML and risk management
- The forerunner of other two methodologies:
  - Objectory
  - Rational

Unified process model

- Iterative and in particular **incremental**
- In each iteration, the product is increased
- Increment factors
  - Improvement of **usability**
  - Better **identification** of the product
- Increments
  - Addictive
  - Perfective
- After each iteration (called cycle), a **product release** is produced
  - Running
  - With documentation

UP development cycles



UP cycles and phases

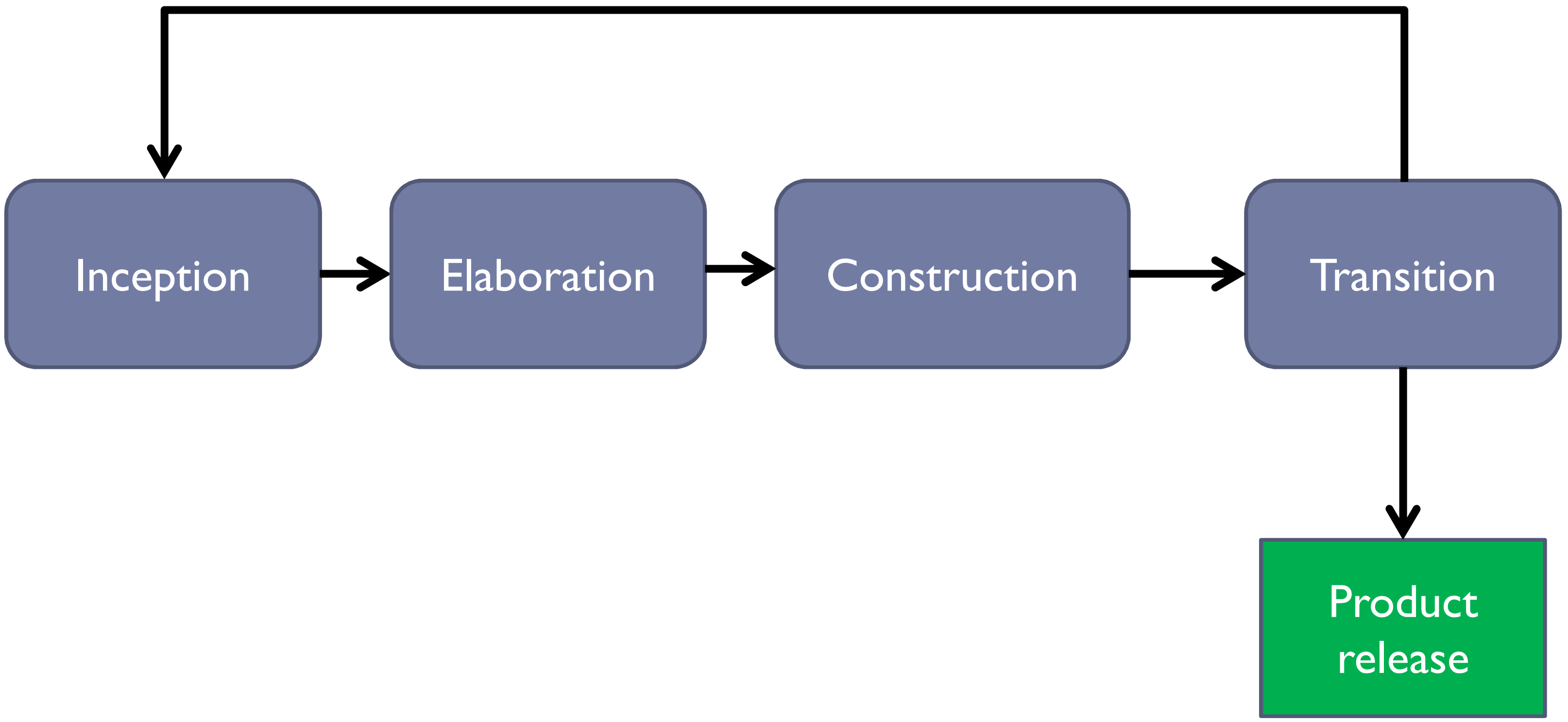
- Each cycle is divided into 4 phases

1. Inception (1/8 of the time)
    - Feasibility study
    - Requirements collection
    - Business case definition
    - Requirement prioritization
2. Elaboration (1/4 of the time)
    - Definition of the architecture
        - System, subsystems, components, component interfaces
    - Requirement selection (10-20% requirements are selected)

UP cycles and phases (2)

3. Construction (1/2 of the time)
   - Implementation of the components
   - Test of the components
   - Integration
   - Quality check
4. Transition (1/8 of the time)
   - Documentation
   - Software release
   - Acceptance test

UP phases


Inception
Elaboration
Construction
Transition
Product release

UP and agile methods

- UP laid the foundations of the agile methods
- And their acceptance in industrial environments