SYSTEMS DEVELOPMENT LIFE CYCLE

Intro

Building Software

Software processes

INTRO

WHY USE A SOFTWARE ENGINEERING PROCESS?

- Experience from classic engineering disciplines:
  - Process standardization is useful!
- Software development processes allow …
  - a better structuring of the development approach.
  - reliable planning of milestones, partial goals and deadlines.
  - comparison and appraisal of projects

Basis for the certification of companies

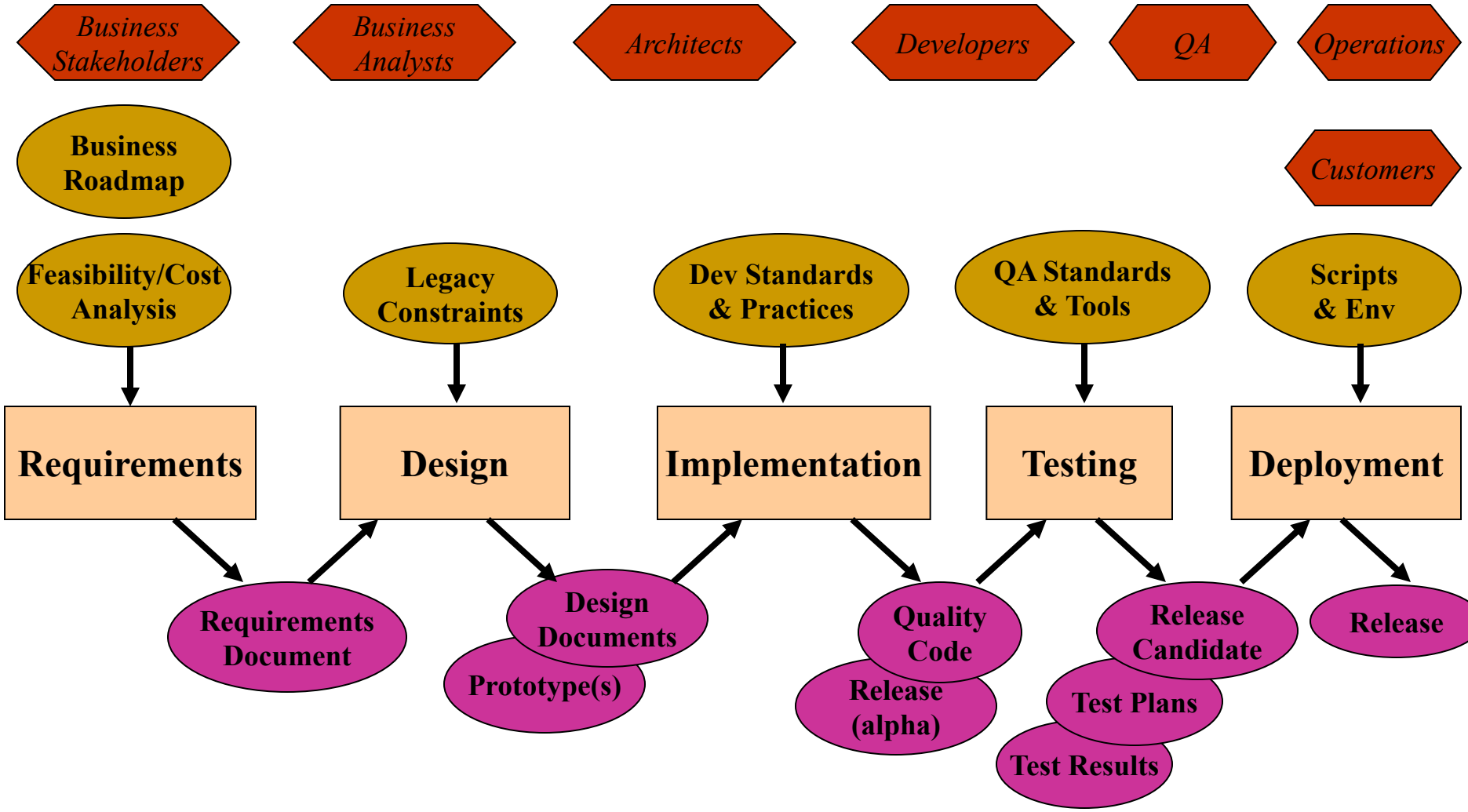
- Proof of process experience for the customers
- e.g. using Capability Maturity Model Integration (CMMI)

REVIEW


Define the Problem
Identify Criteria and Constraints
Generate Ideas
Explore Possibilities
Select a Design
Develop a Detailed Design
Create Models and Prototypes
Test and Evaluate
Refine the Design
Implementation
Communicate Process and Results

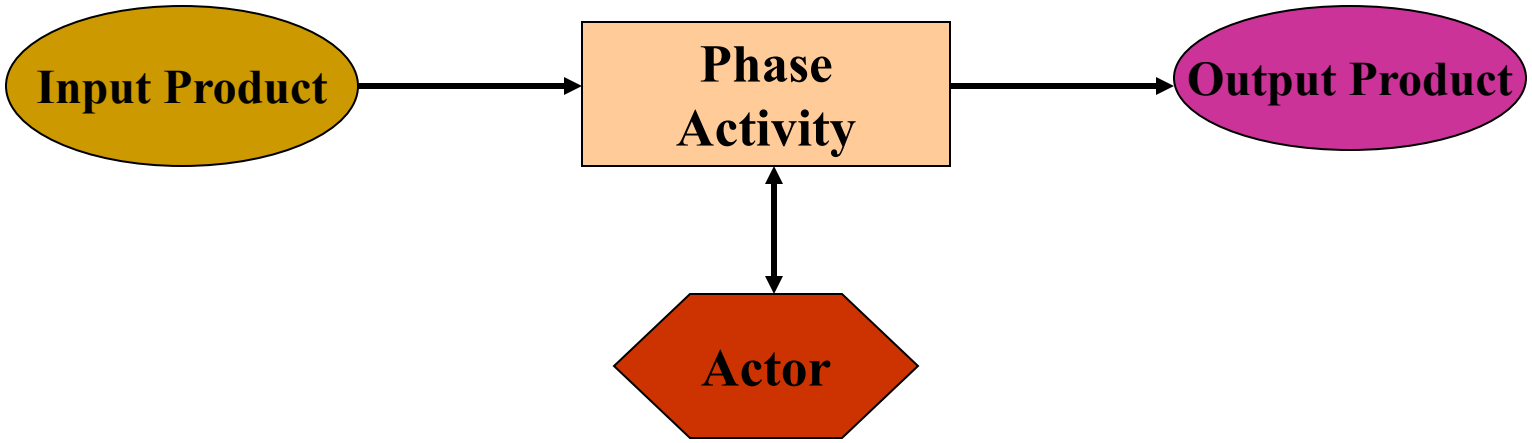
SOFTWARE PROCESS FLOW

An Over-simplified View of the Software Development Lifecycle (SDLC) Process Phases



SOFTWARE PROCESS PHASES

- How are the phases connected?
  - Now that we have talked about each phase, what transitions exist between each phase?
  - What data is needed by each phase?
  - What data is produced by each phase?
  - Who are the players (the actors) involved in each phase?
- Given what we have covered in each phase, what is the *lifecycle model* that captures software process?



BUILDING SOFTWARE

HOW DO YOU BUILD SOFTWARE?

SOFTWARE PROCESSES

A software process is a structured set of activities followed to develop a software system.

There are four generic activities or **phases**:

- **specification:** identifying the required services and the constraints on system operation & development
- **development:** converting the system specification into an executable system
- **validation:** showing that a system conforms to its specification and meets the user requirements
- **evolution:** adapting software to changes in circumstances

SOFTWARE PROCESSES

also called software life cycle

A software process is a structured set of activities followed to develop a software system.

what the system does

how the system works

that the system works

keep the system working

There are four g or **phases**:

- **specification**: ed services and the constraints on s evelopment
- **development**: c ecification into an executable syste
- **validation**: sh ms to its specification an nts
- **evolution**: adapting software to changes in circumstances

SOFTWARE PROCESSES

Different software processes differ based on

- how the phases are arranged
- how much attention is paid to each phases
  - whether and how the phases are broken down
- what must be done in each phase
- when and how to transition between phase
  - what must be delivered by each phase

SPECIFICATION

also called requirements engineering (SER415)

specification: identifying the required services and the constraints on system operation & development

Main concern: **what** should the system do?

Main result: **requirements** == description of desired system functionality

- **user** requirements: abstract, natural-language
  - aimed at users, customer, and contractor
- **system** requirements: specific, semi-formal
  - aimed at developers and contractor

REQUIREMENTS ENGINEERING TASKS

- **feasibility study**: check whether a new system is needed, and whether it can be build it within the given time and budget
- **requirements elicitation**: gather requirements by
  - user interviews and questionnaires
  - user observation (ethnographic studies)
  - brainstorming and role playing
  - use cases and prototyping.
- **requirements analysis**: classify, organize, and prioritize requirements
  - can prompt more elicitation

REQUIREMENTS ENGINEERING TASKS

- **requirements specification**: translate chosen requirements into coherent document(s)
  - different specification methods
- **requirements validation**: check requirements for realism, consistency, and completeness
  - interleaved with analysis and specification
  - can prompt more elicitation
  - will inevitably lead to the discovery of errors that must be fixed in the user and system requirements

REQUIREMENTS ENGINEERING IS HARD...

development

development: converting the system specification into an executable system

Main concern: **how** should the system work?

DEVELOPMENT = DESIGN + IMPLEMENTATION

development: converting the system specification into an executable system

Traditionally broken down into several stages:

- architectural design
- interface design
- abstract specification
- coding
- component design
- data structure design
- algorithm design
- debugging

- development is an iterative process with feedback between the stages
- design and implementation are typically interleaved

VALIDATION

validation: showing that a system conforms to its specification and meets the user requirements

Main concern: **does** the system work?

Verification and Validation

validation: showing that a system conforms to its specification and meets the user requirements

Traditionally broken down into two separate activities:

- verification: do we build the system right?
- validation: do we build the right system?

V&V techniques:

- reviews (code and documents)
- prototyping and simulation (documents)
- testing
- static analysis (e.g., model checking, proof)

Verification and Validation

Testing works at different levels:

- **unit testing**: individual components
  - methods; test functional behavior (pre/post)
- **module testing**: groups of related components
  - classes; test class invariants
- **system testing**: whole (sub-) system(s)
  - test emergent properties (e.g., security, reliability,...)
- **acceptance testing**: use customer data
  - test that system meets user expectations

EVOLUTION

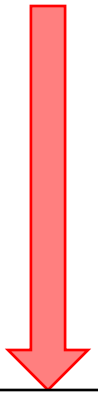
also called maintenance

evolution: adapting software to changes in circumstances

Main concern: **keep** the system working

Traditionally broken down into several categories:

- **corrective**: fixing bugs in the field
- **adaptive**: respond to changes in environment
- **perfective**: respond to changes in user requirements
- **preventive**: improve or refactor system to prevent future problems

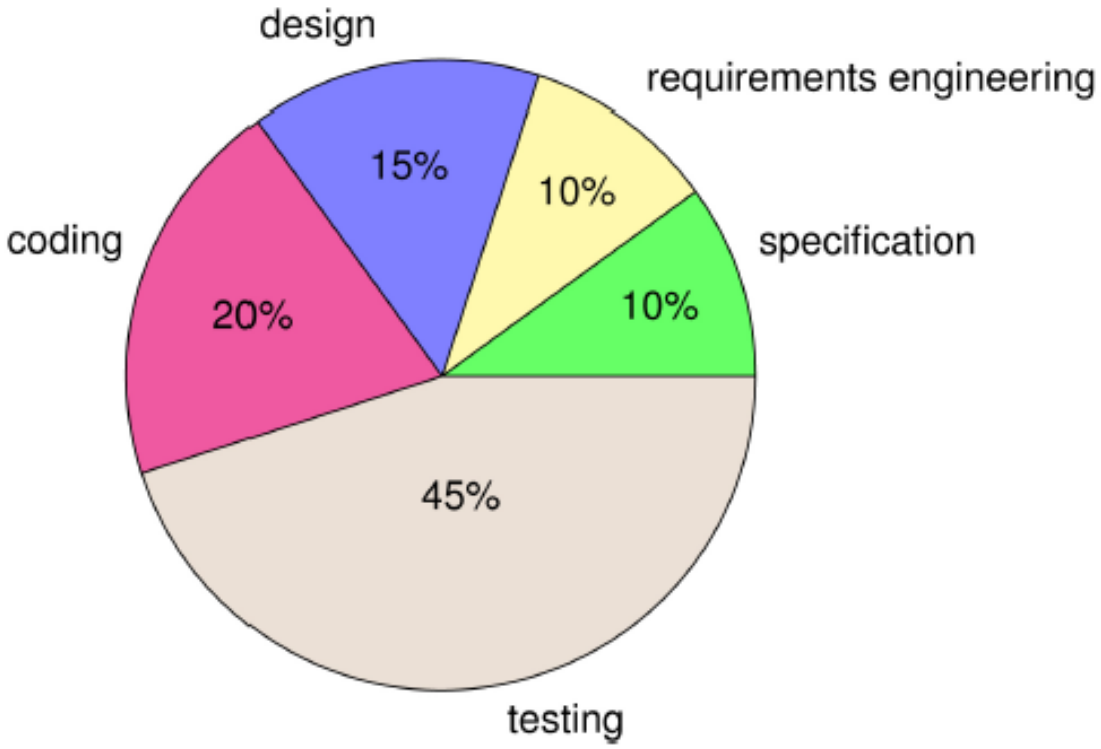


perceived as less urgent

More information:

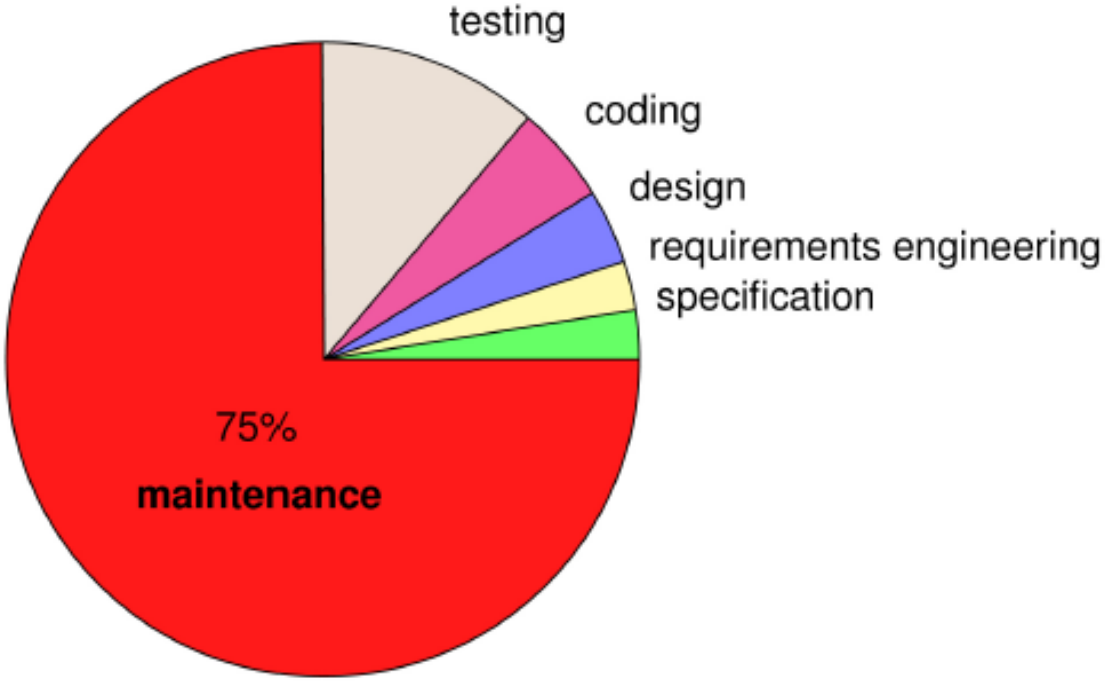
- ISO/IEC 14764:2006 Software Engineering — Software Life Cycle Processes — Maintenance.

DISTRIBUTION OF EFFORTS



Verification effort dominates both specification and development efforts!

DISTRIBUTION OF EFFORTS



... but maintenance (50-75%)
dominates the total life cycle effort!

SUMMARY

- software development process is split into phases:
  - Sspecification
  - Development
  - Validation
  - Maintenance
- maintenance dominates total life cycle effort
- software process models describe how the phases are arranged and what must be done in each phase

SOFTWARE PROCESSES

DIMENSIONS OF SOFTWARE PROCESSES

- **linear** vs. **iterative**
  - phases are executed in order
  - phases are repeated regularly
- **monolithic** vs. **incremental**
  - single system is delivered at end
  - system is delivered in several working versions with increasing functionality
- **document**-based vs. **code**-based
  - most deliverables (except final system) are text
  - almost all deliverables are code (prototypes, test cases, ...)

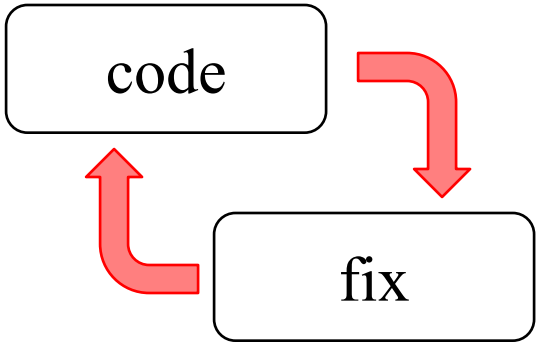
DIMENSIONS OF SOFTWARE PROCESSES

- **heavy**-weight vs. **light**-weight
  - strict rules and detailed plans, formal models and methods
  - flexible and reactive, prototyping
  - also called plan-driven and agile
- **component**-driven vs. **model**-driven vs. **test**-driven
  - requirements are modified to fit existing components
  - requirements are formulated as (formal) models
  - requirements are formulated as test cases

Usually combinations are used but not all combinations possible or feasible...

CODE-AND-FIX

- No specification, no design
- Just dive in and starting coding



Advantages

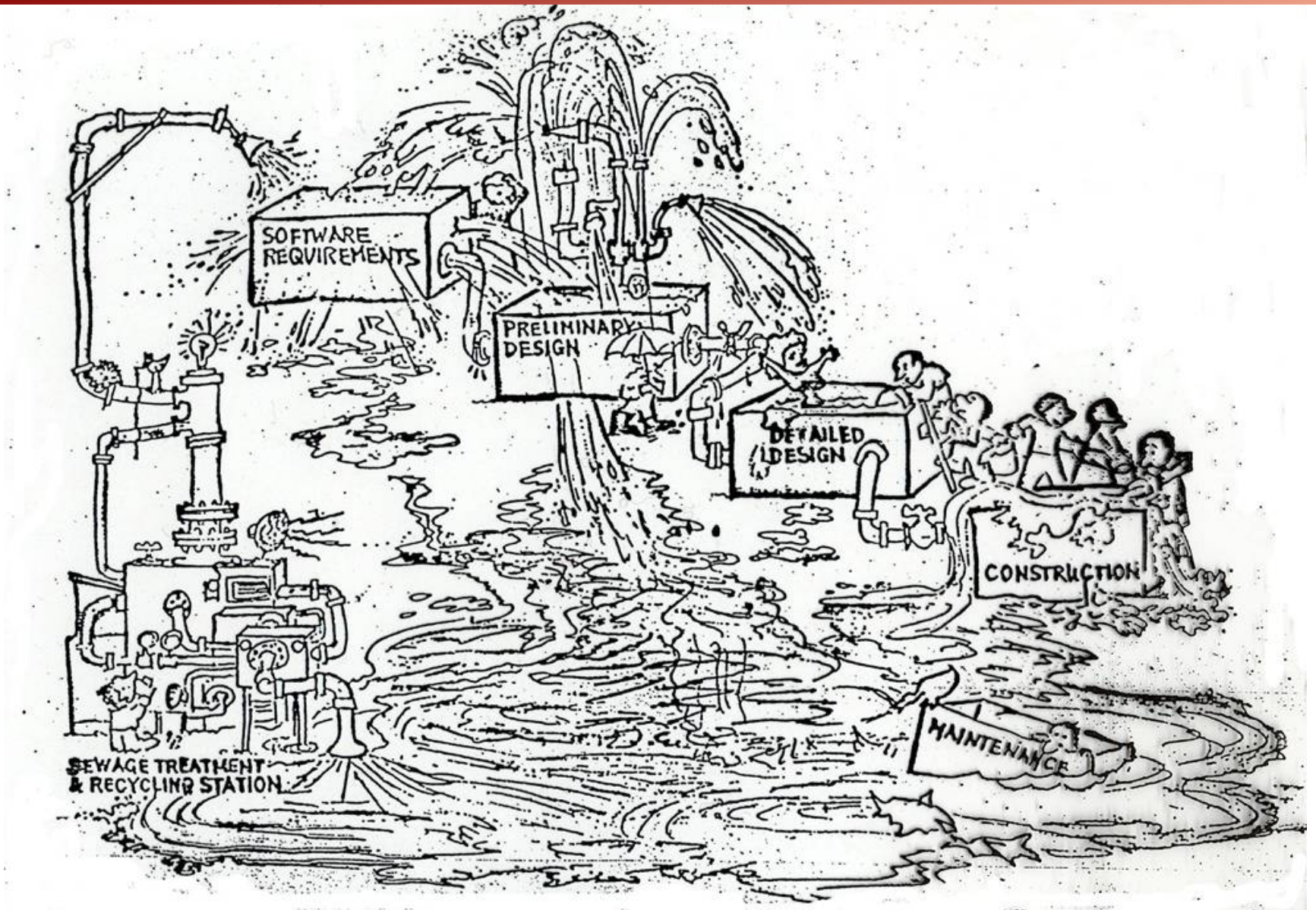
- no overhead
- see progress quickly
- suitable for small, short-lived programs (<200 lines)

Disadvantages

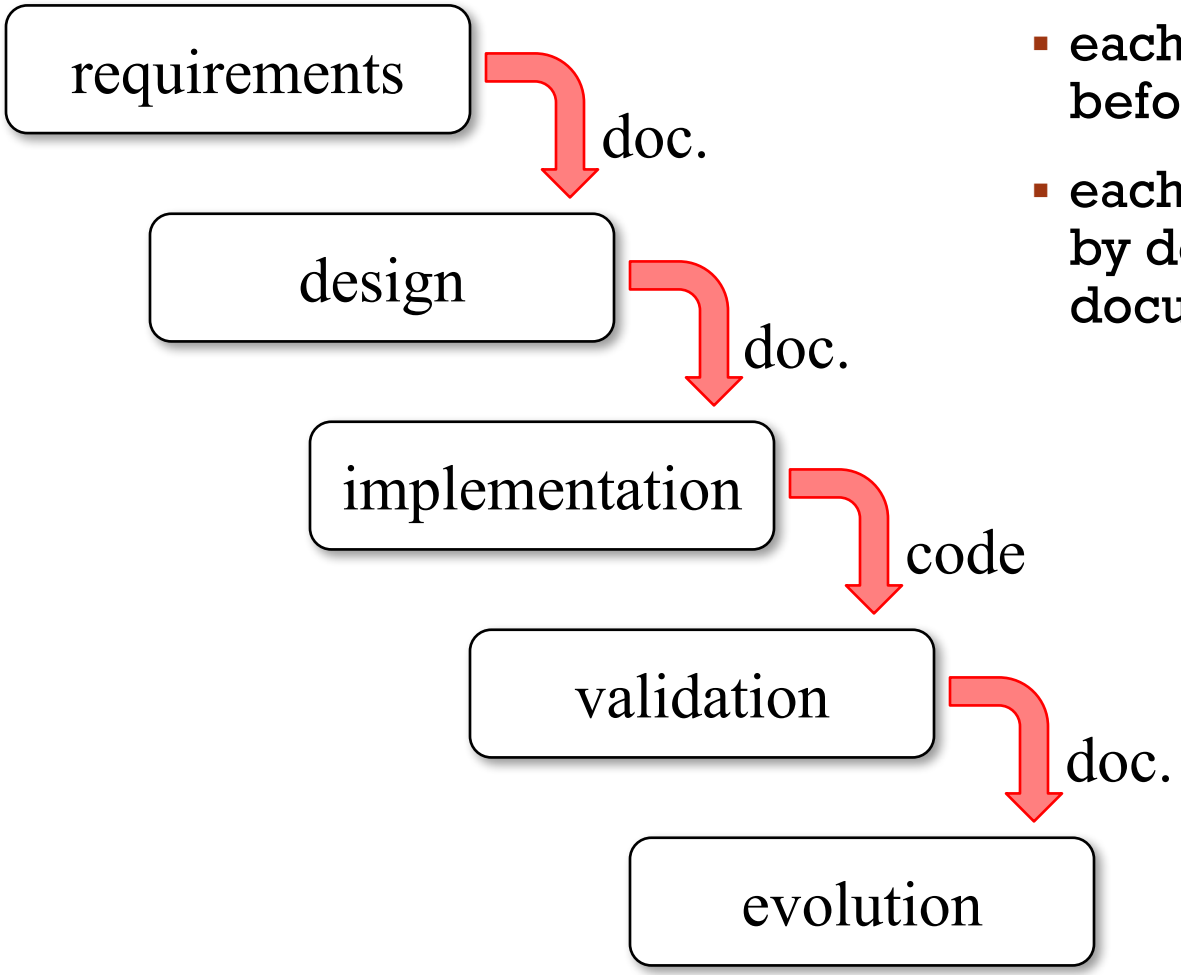
- no way to assess progress, quality, or risks
- changes likely to require major design overhaul

Dangerous for all projects, use only in very small programs!!

WATERFALL MODEL



CLASSIC WATERFALL MODEL



- each phase **must** be completed before next is started
- each phase transition is marked by delivery of corresponding documents

More information:

- W. Royce: Managing the Development of Large Software Systems, Proc.IEEE WESCON 26:1–9, 1970.
- DOD-STD-2167, MILITARY STANDARD: DEFENSE SYSTEM SOFTWARE DEVELOPMENT. US Department of Defense,1988.

CLASSIC WATERFALL MODEL: ADVANTAGES

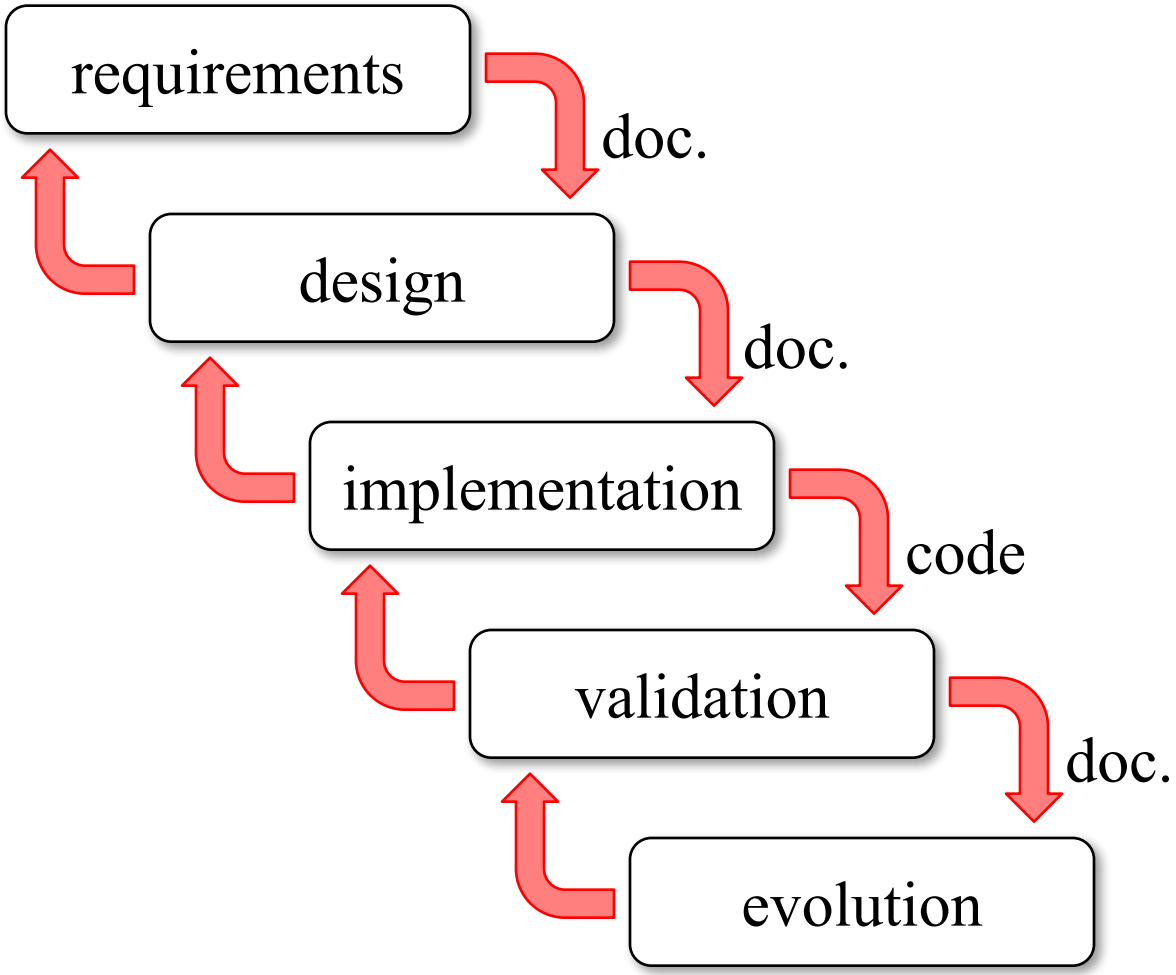
- phase distinction and order encourages discipline
  - can provide support for inexperienced development teams
- emphasis on communication through documents
  - promotes documentation
  - support for off-shoring
  - allows automation (if documents are formal models)
- implementation postponed until project objectives are well-understood
  - minimizes risk of waste

CLASSIC WATERFALL MODEL: DISADVANTAGES

- inflexible partitioning of project into distinct stages
  - no feedback, but specification problems often show only during coding or integration
  - validation too late
- first working version available only very late
  - little feedback from users until then
  - increases risk of obsolescence
- requires accurate and complete requirements right from the start
  - difficult to respond to changing requirements
- over-reliance on documents, heavy-weight process
  - documents get out-of-sync

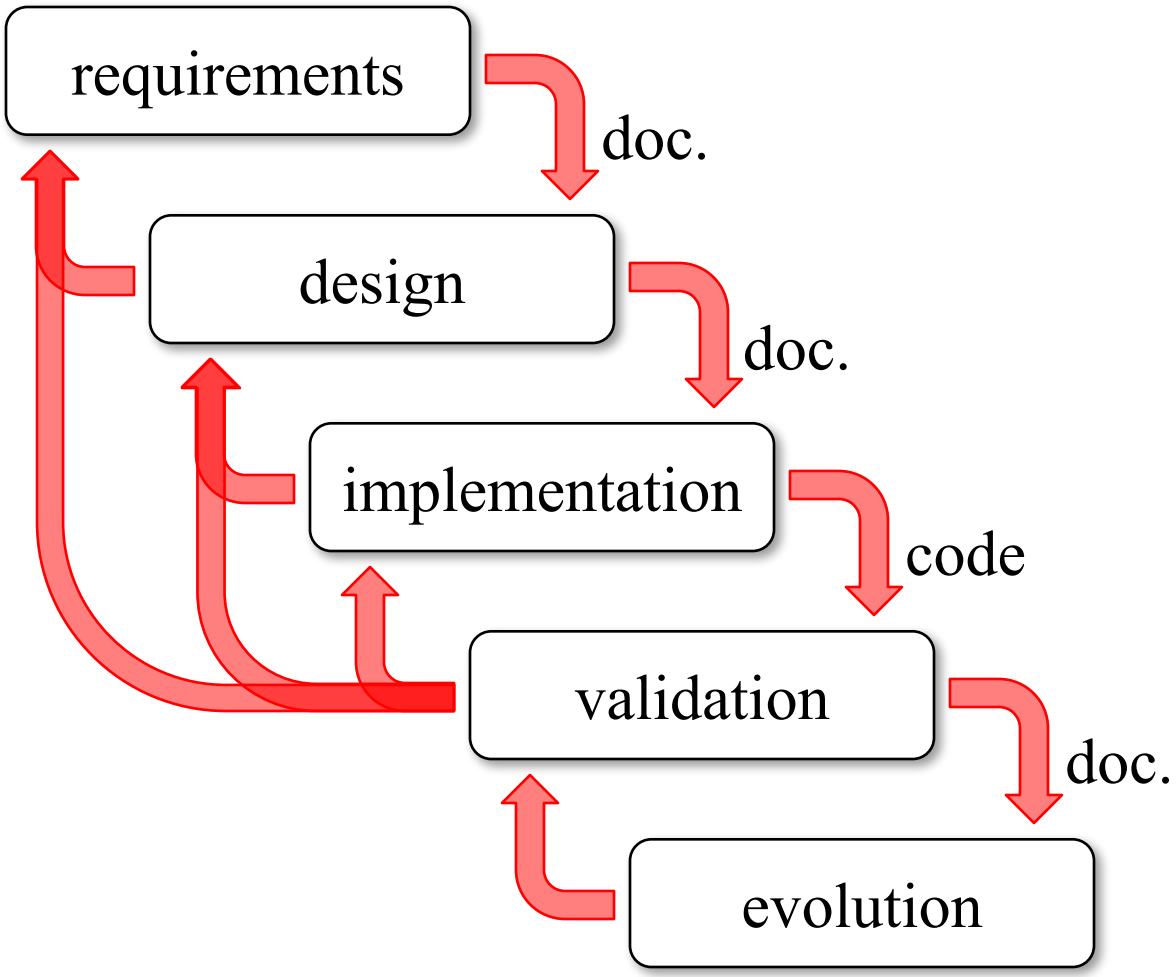
Phases never meant to be followed strictly linearly!!

MODIFIED WATERFALL MODEL



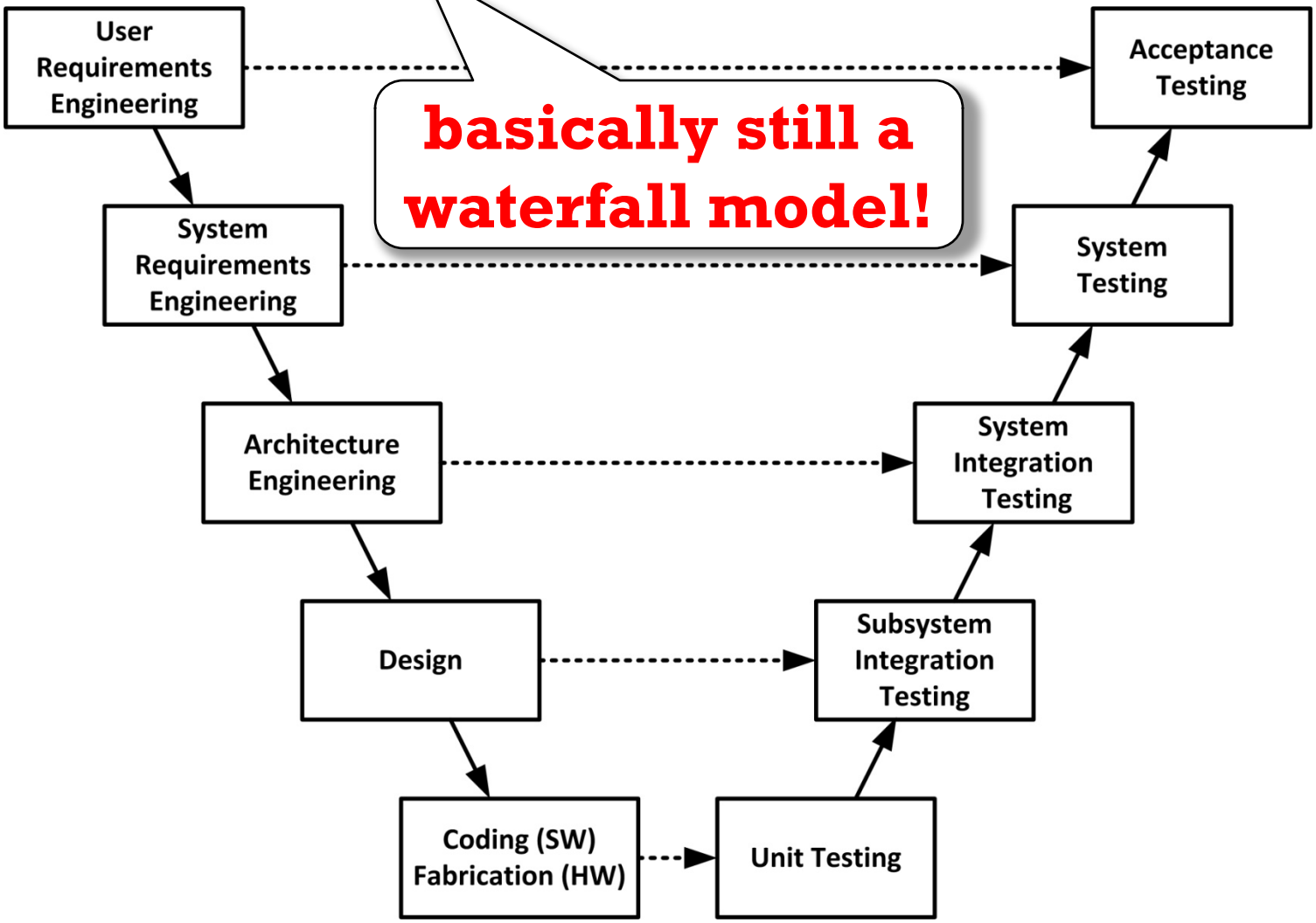
- previous phase can be revisited when problems occur
- increases flexibility
- … but requires more synchronization of documents

MODIFIED WATERFALL MODEL



- previous phase can be revisited when problems occur
- increases flexibility
- … but requires more synchronization of documents
- …and often more than one earlier phase is affected
- … and validation is still late

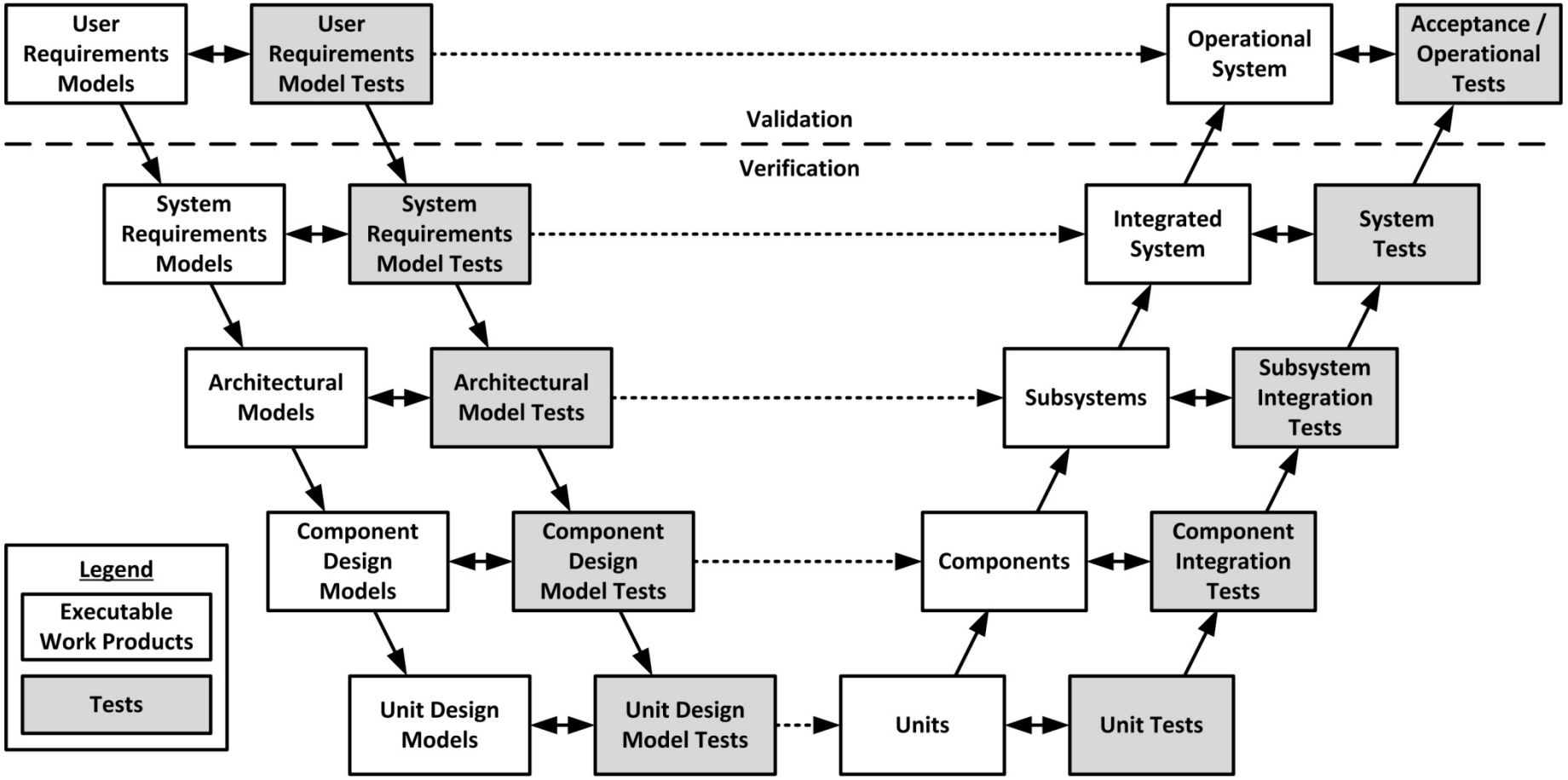
V-Model: Mirror all specification and development steps by validation steps.



Source:

- D. Firesmith: Using V models for testing, `http://blog.sei.cmu.edu/post.cfm/using-v-models-testing-315`

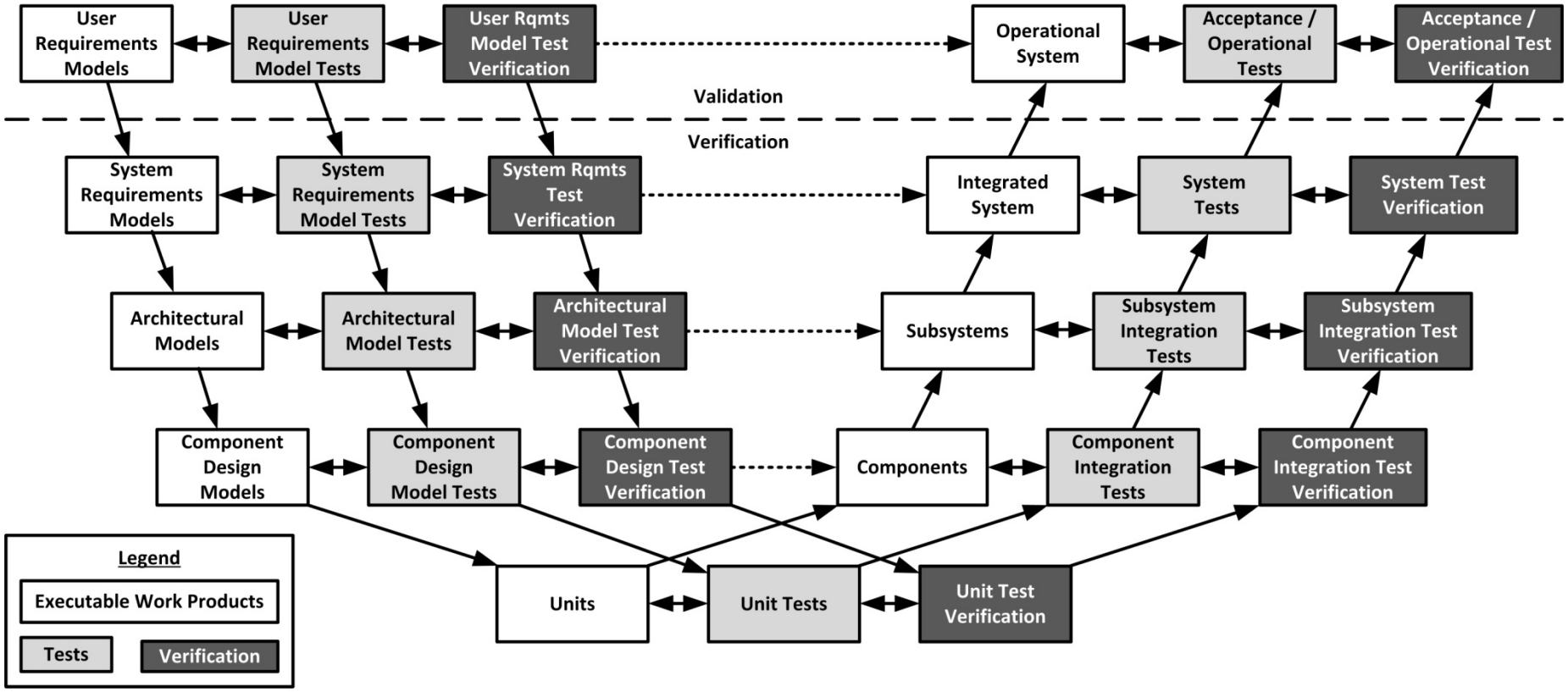
DOUBLE-VEE: USE AND TEST MODELS.



Source:

- D. Firesmith: Using V models for testing, `http://blog.sei.cmu.edu/post.cfm/using-v-models-testing-315`

Triple-Vee: Verify Tests.



Source:

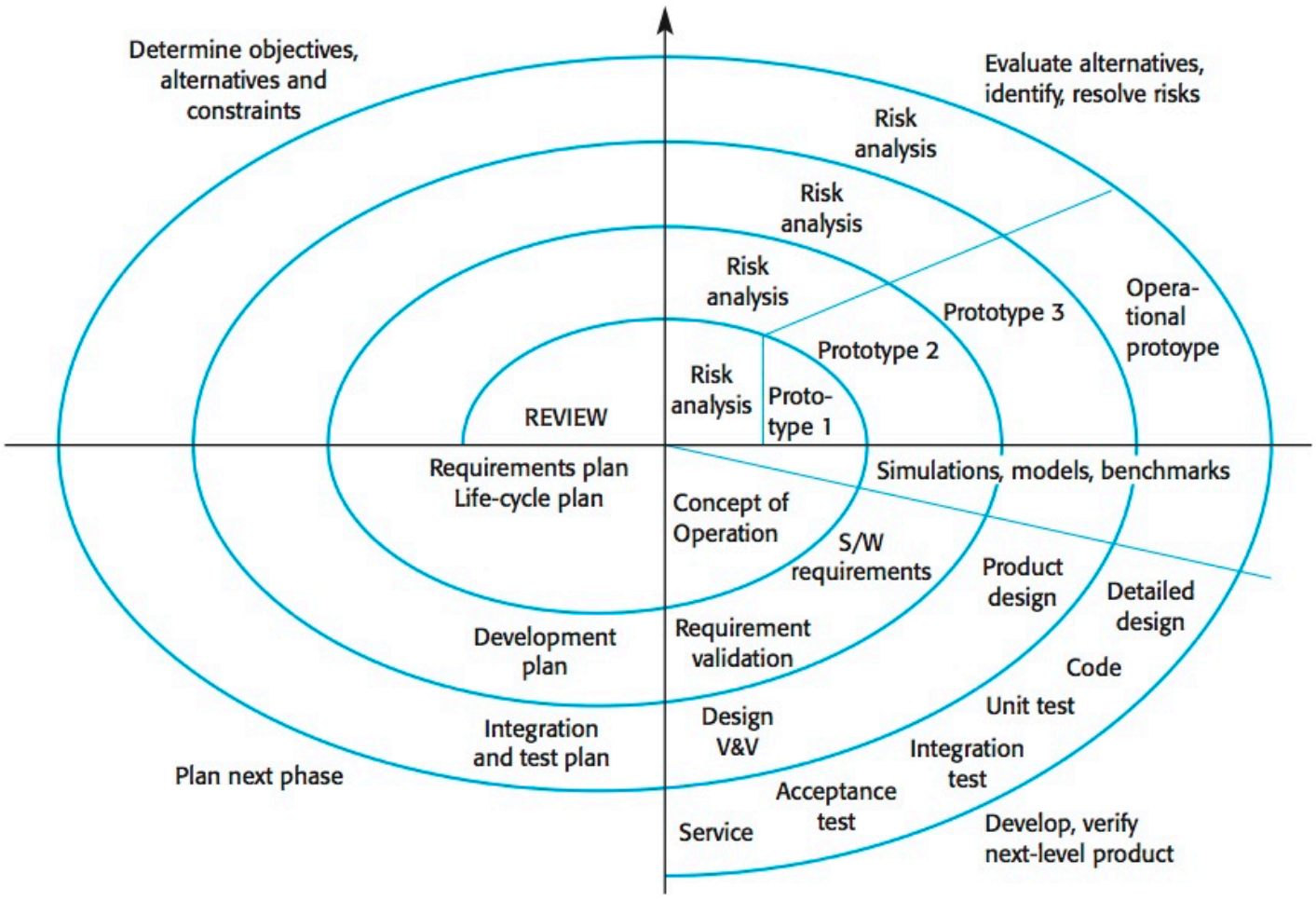
- D. Firesmith: Using V models for testing, `http://blog.sei.cmu.edu/post.cfm/using-v-models-testing-315`

BIG DESIGN UP-FRONT MODELS (BDUF)

BDUF denotes any model where the design is completed and perfected before the implementation is started.

- derived from waterfall model
- suitable for
  - well-understood domains with stable requirements
  - high-reliability systems
- unsuitable for
  - highly complex systems in ill-understood domains
  - changing requirements

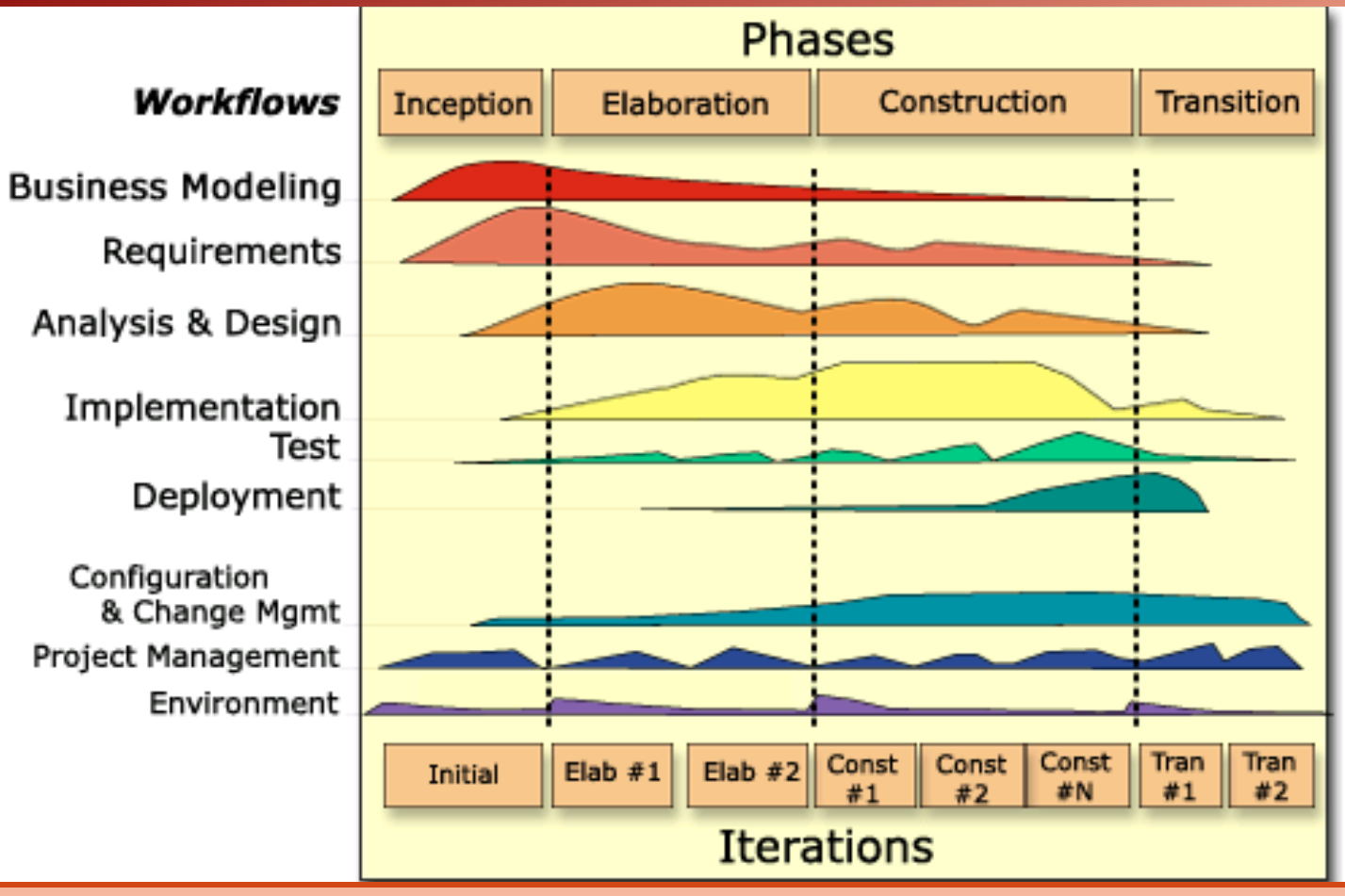
SPIRAL MODEL OF THE SOFTWARE PROCESS



- Risk-driven
- Prototyping
- Expand and Converges

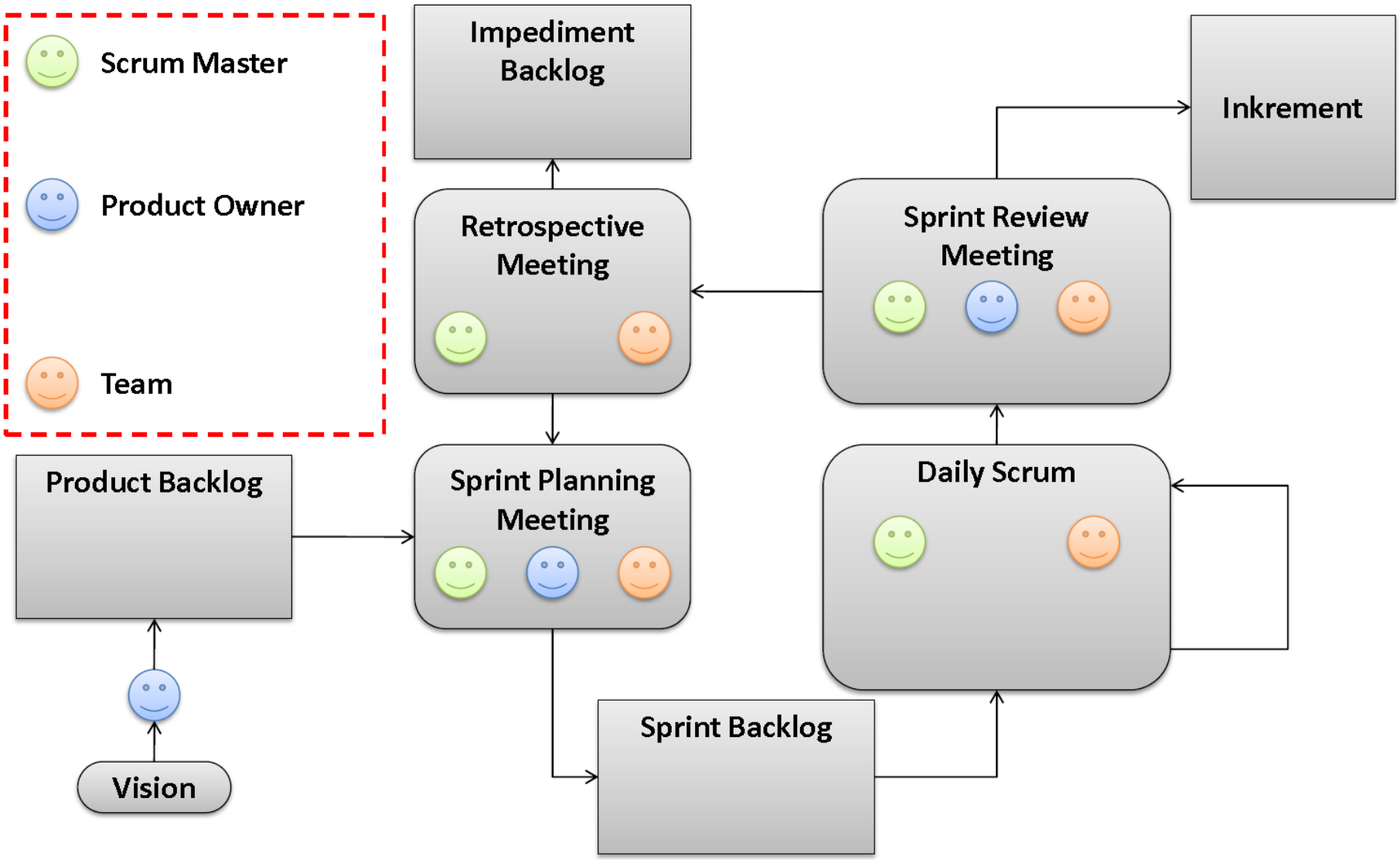
Problem: Some say you “spiral out of control”!

RATIONAL UNIFIED PROCESS (RUP)

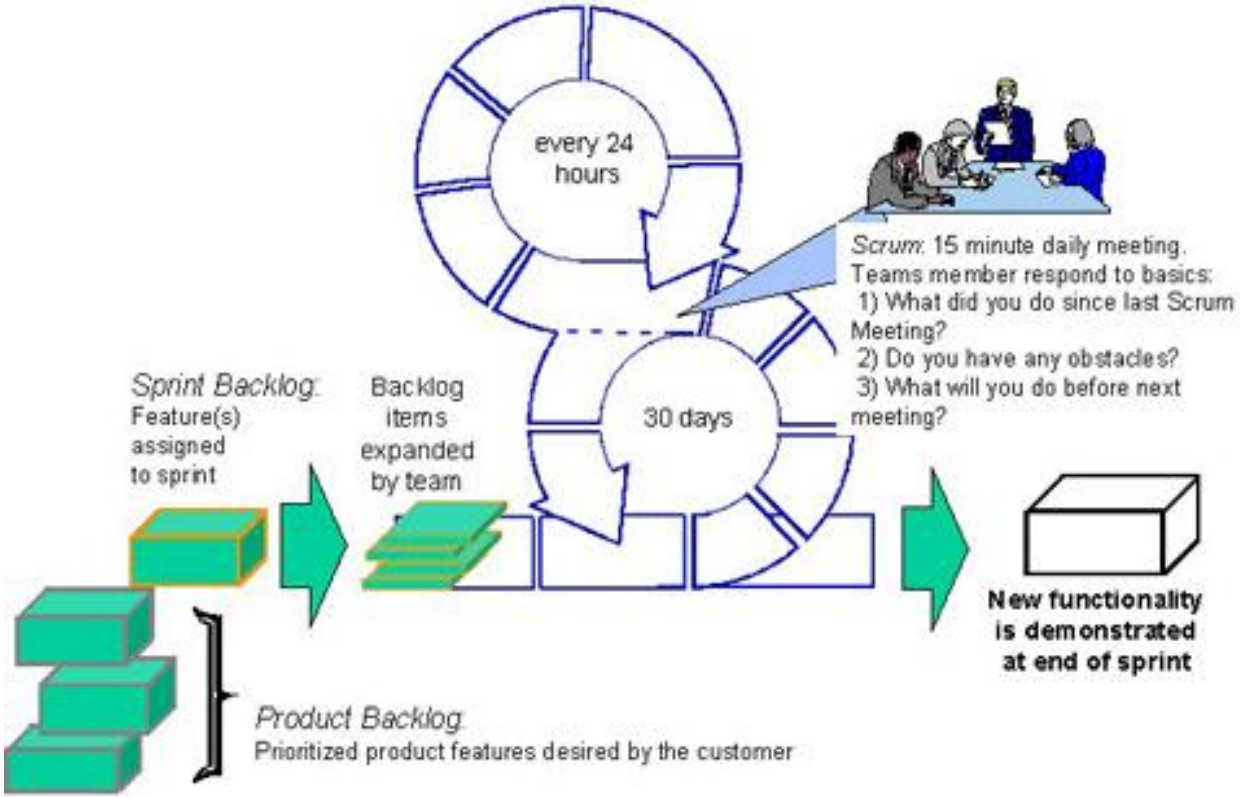


- Pre-Agile popular model, still is widely adopted
- Explicitly identifies Workflows and Transition Phase
- Over-commercialized; accused of being waterfall-ish

AGILE EXAMPLES: SCRUM, XP



AGILE EXAMPLES: SCRUM, XP



We will use Scrum & XP in the Spring!

Pros:

- Adapts to change
- People-centric
- Fast & light
- Fosters dev buy-in

Cons:

- Requires good people
- Best small teams

SUMMARY

- Different distincions between processes
- Main distinction: **linear** vs. **iterative**
- The linear **waterfall model** is the traditional model
  - good for stable requirements, bad for change
- Adaptions form the Waterfall model to make it more flexible are now often uesed
- **iterative**, **incremental** Agile methods are often used in new projects nowadays

We will use a RUP like process (but adjusted to our needs). Agile will be a main topic in SER316