CIS 422
Software Methodologies I

Software Engineering (van Vliet)

Chapter 3 – Software Life Cycle Rev.

Professor: Juan J. Flores

jflore10@uoregon.edu

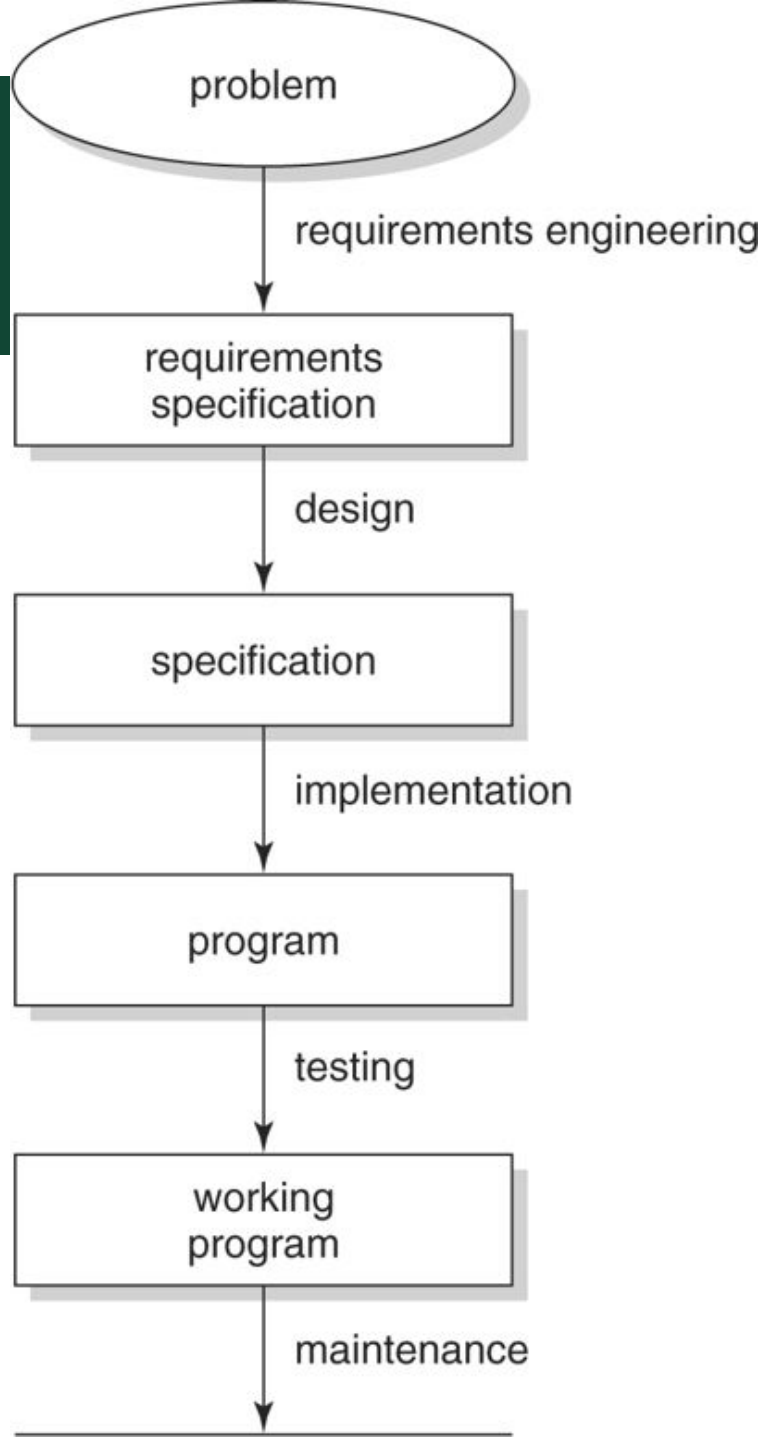
UNIVERSITY OF OREGON

Introduction

- Software development projects are large and complex
- Phased approach to control
- Traditional models ~ document-driven
- Evolutionary models ~ maintenance is inevitable
- Latest fashion: agile methods, eXtreme Programming
- Life cycle explicitly modeled – process modeling language

Simple life cycle model

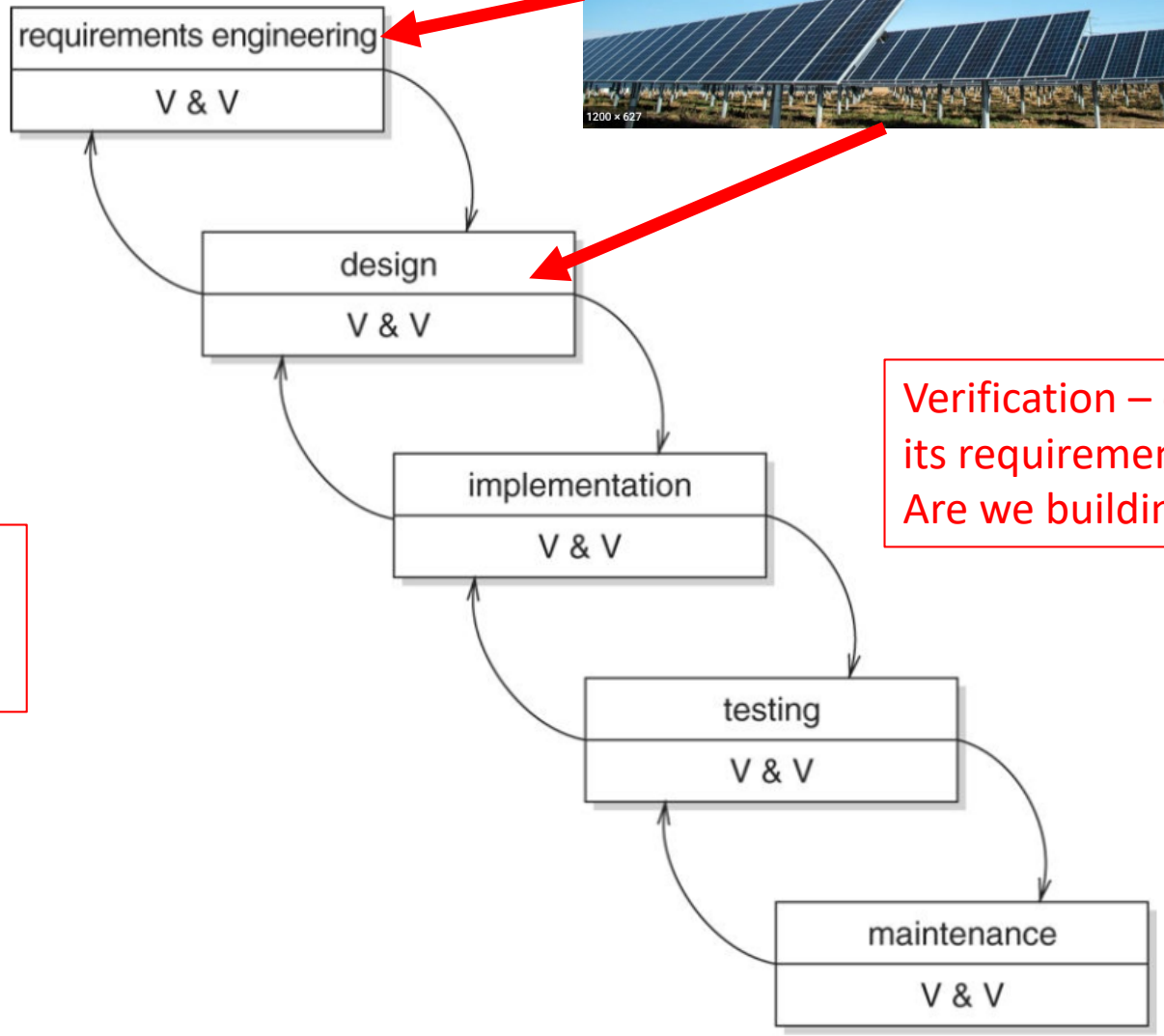
- Is this how we develop software?



Simple Life Cycle Model

- Document driven ~ planning driven
- Milestones reached if documentation is delivered
  - Requirements specification
  - Design specification
  - Program
  - Test document
- Planning upfront – contracts are signed
- Problems
  - feedback is not taken into account
  - maintenance does not imply evolution

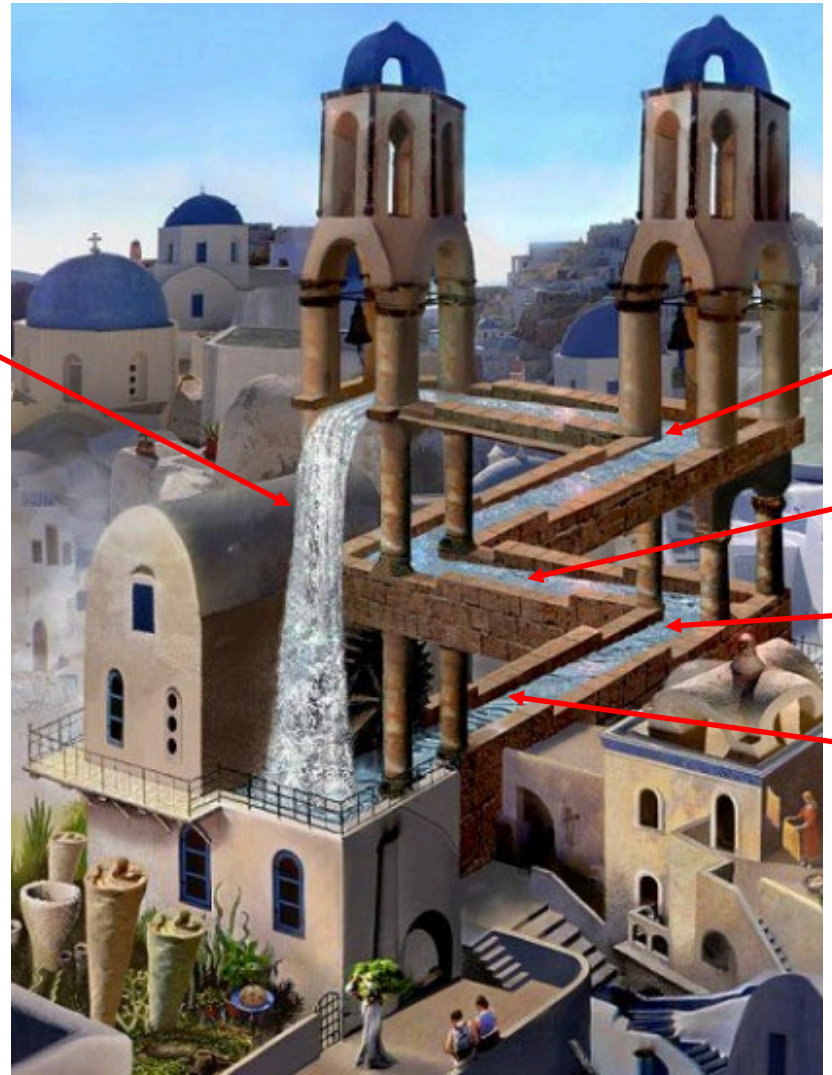
Waterfall Model



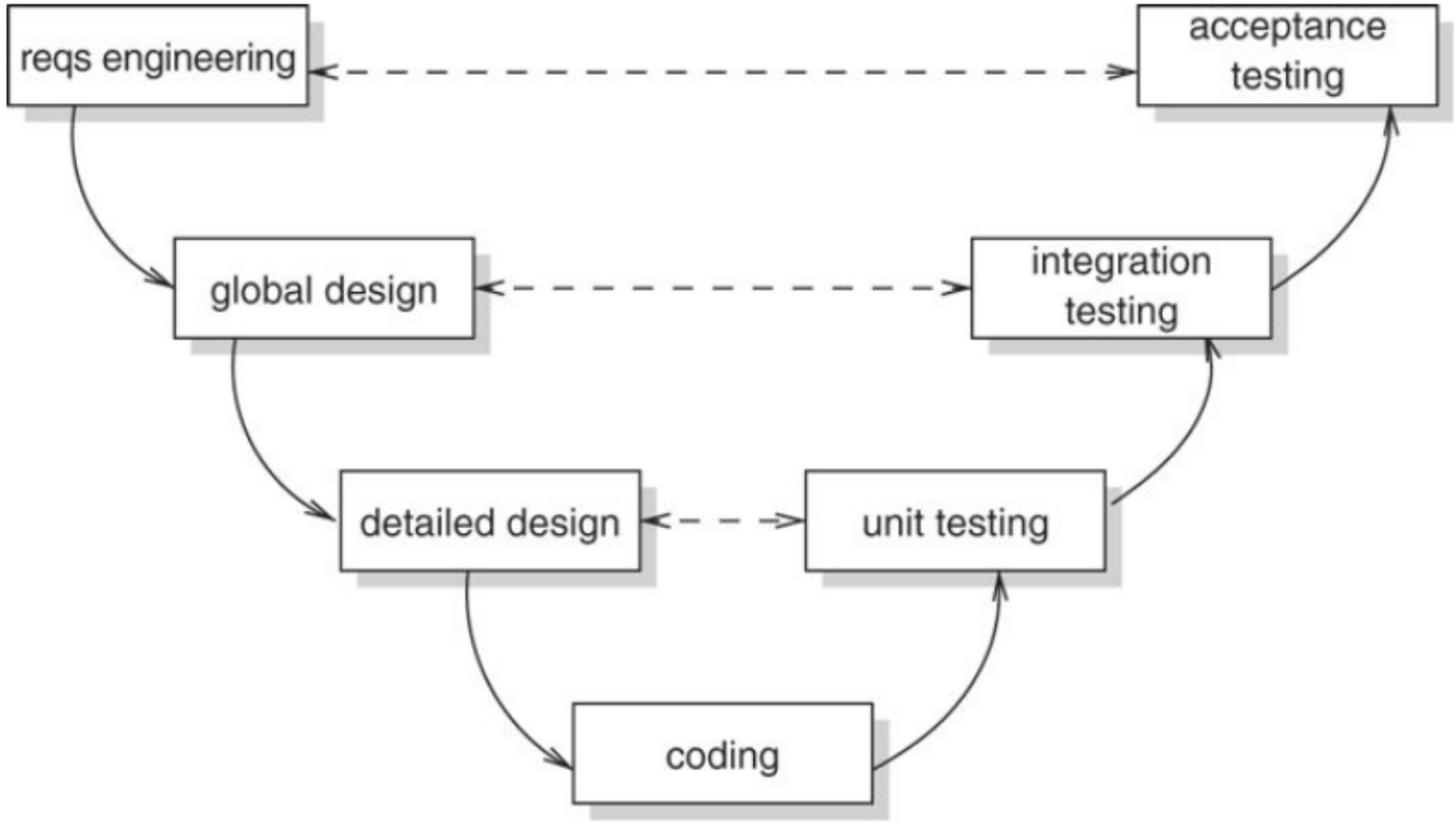
Verification – does the system meet its requirements?
Are we building the system right?

Validation – does the system meet the user's requirements?
Are we building the right system?

Another waterfall model



V-Model


reqs engineering
acceptance testing
global design
integration testing
detailed design
unit testing
coding

Waterfall Model

- Includes iteration and feedback
- V & V after each step
- User requirements are fixed as early as possible
- Problems
  - Too rigid
  - Developers cannot move between various abstraction levels

Activity versus phase

Activity \ *Phase*	Design	Implementation	Integration testing	Acceptance testing
Integration testing	**4.7**	**43.4**	**26.1**	**25.8**
Implementation (& unit testing)	**6.9**	**70.3**	**15.9**	**6.9**
Design	**49.2**	**34.1**	**10.3**	**6.4**

Effort distribution

Lightweight (agile) approaches

- Prototyping
- Incremental development
- Rapid Application Development (RAD)
- Dynamic systems Development Method (DSDM)
- Extreme Programming (XP)

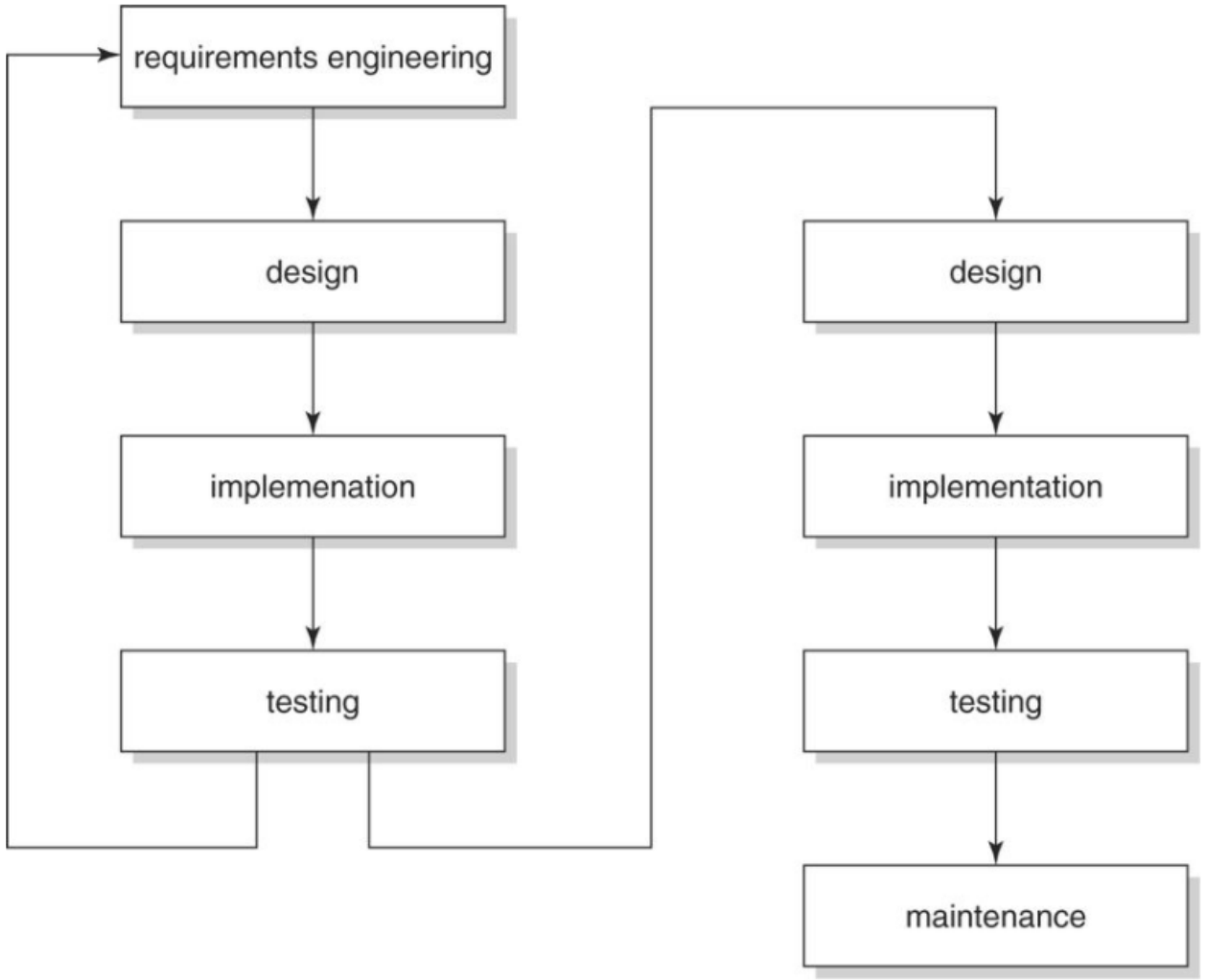
The Agile Manifesto

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

Prototyping

- Requirements elicitation is difficult
  - Software development ⇦ present situation is unsatisfactory
  - Desirable new situation – unknown
- Prototyping to obtain requirements of some aspects of the system
- Prototyping should be a relatively cheap process
  - Rapid prototyping languages and tools
  - Not all functionality needs to be implemented
  - Production quality is not required

Prototyping for requirements engineering



Prototyping

- *Throwaway prototyping*
  n-th prototype followed by a waterfall-like process

- *Evolutionary prototyping*:
  nth prototype is delivered

Prototyping, advantages

- The resulting system is easier to use
- User needs are better accommodated
- The resulting system has fewer features
- Problems are detected earlier
- The design is of higher quality
- The resulting system is easier to maintain
- The development incurs less effort

Prototyping, disadvantages

- The resulting system has more features
- The performance of the resulting system is worse
- The design is of less quality
- The resulting system is harder to maintain
- The prototyping approach requires more experienced team members

Prototyping, recommendations

- Users and designers must be well aware of issues and pitfalls
- Use prototyping when the requirements are unclear
- Prototyping needs to be planned and controlled as well

Incremental Development

- Software system delivered in small increments
  - Avoiding the Big Bang effect
- Waterfall model is employed in each phase
- Users closely involved in directing the next steps
- Incremental development prevents overfunctionality
  - Do you really need that?

RAD: Rapid Application Development

- Evolutionary development, with *time boxes*: fixed time frames within which activities are done
- Time frame is decided upon first, then one tries to realize as much as possible within that time frame
- Other elements: Joint Requirements Planning (JRP) and Joint Application Design (JAD), workshops in which users participate
- Requirements prioritization (*triage)*
- Development in a SWAT team: Skilled Workers with Advanced Tools

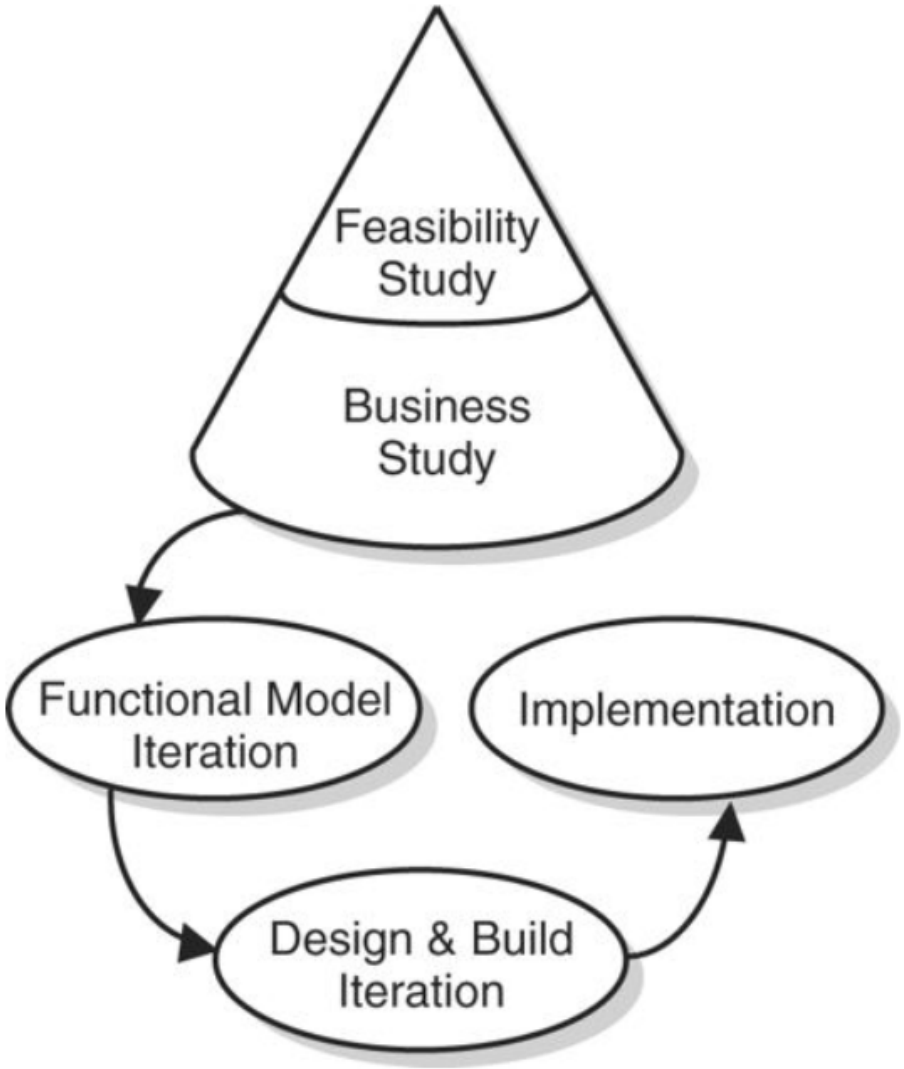
DSDM

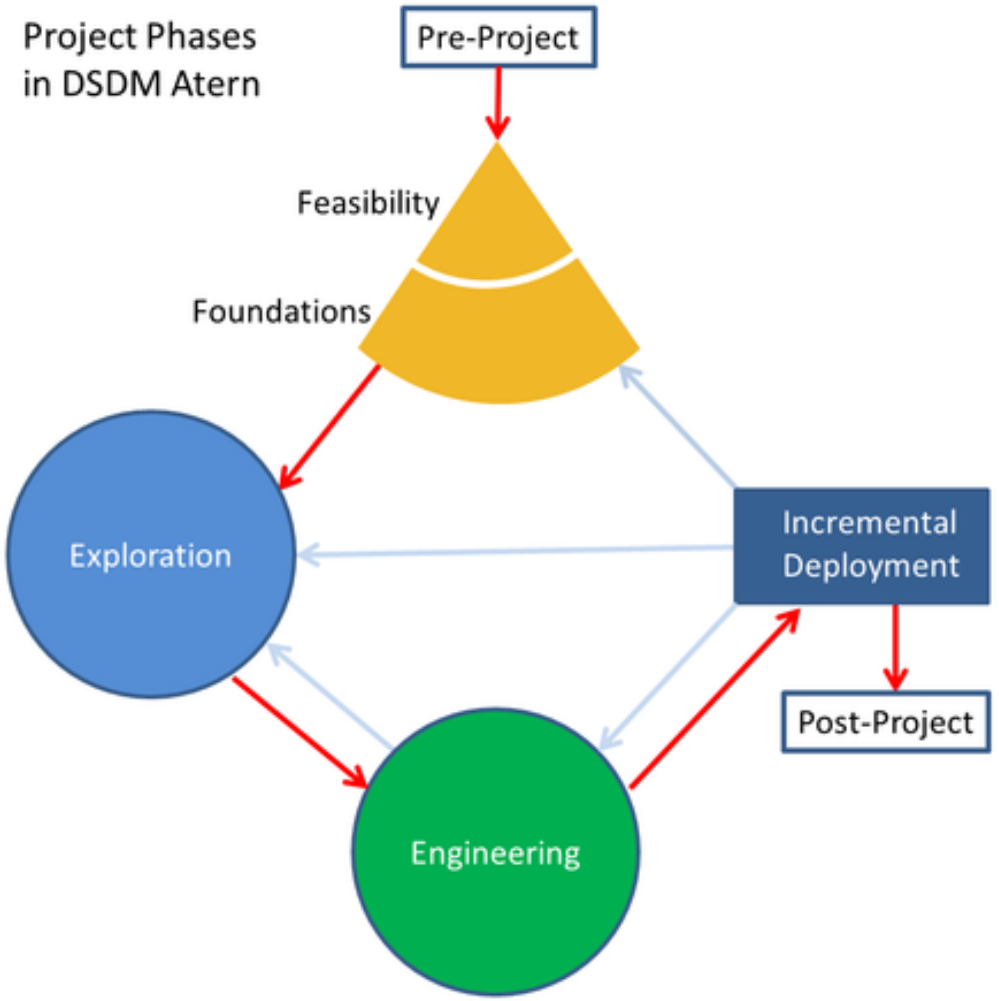
- Dynamic Systems Development Method
  - #1 RAD framework in UK
- Fix time and resources (*timebox*), adjust functionality accordingly
- The complete set of DSDM practices available to DSDM consortium's members

DSDM phases

- Feasibility: delivers feasibility report and outline plan, optionally fast prototype (few weeks)
- Business study: analyze characteristics of business and technology (in workshops), delivers Aspect-Oriented System Architecture Definition
- Functional model iteration: timeboxed iterative, incremental phase, yields requirements
- Design and build iteration
- Implementation: transfer to production environment

DSDM phases


Feasibility Study
Business Study
Functional Model Iteration
Implementation
Design & Build Iteration


Project Phases in DSDM Atern
Pre-Project
Feasibility
Foundations
Exploration
Incremental Deployment
Post-Project
Engineering

XP – eXtreme Programming

- Feedback is obtained quickly
- Design is kept as simple as possible
  - Everything is done in small steps
- The system always compiles, always runs
- Client as the center of development team
- Developers have same responsibility w.r.t. software and methodology

XP Practices

- The planning game, the scope of the next release is quickly determined
- Small releases (e.g. 2 weeks)
- Metaphor: XP programmers use metaphors to name each chunk of the code
- Simple design
- Refactoring – restructure maintaining behavior
- Pair programming
- Customer tests
- Collective code ownership
- Continuous integration: system always runs
- 40-hour week
- Whole team: the client is part of the team
- Test-driven development: tests developed first
- Coding standards

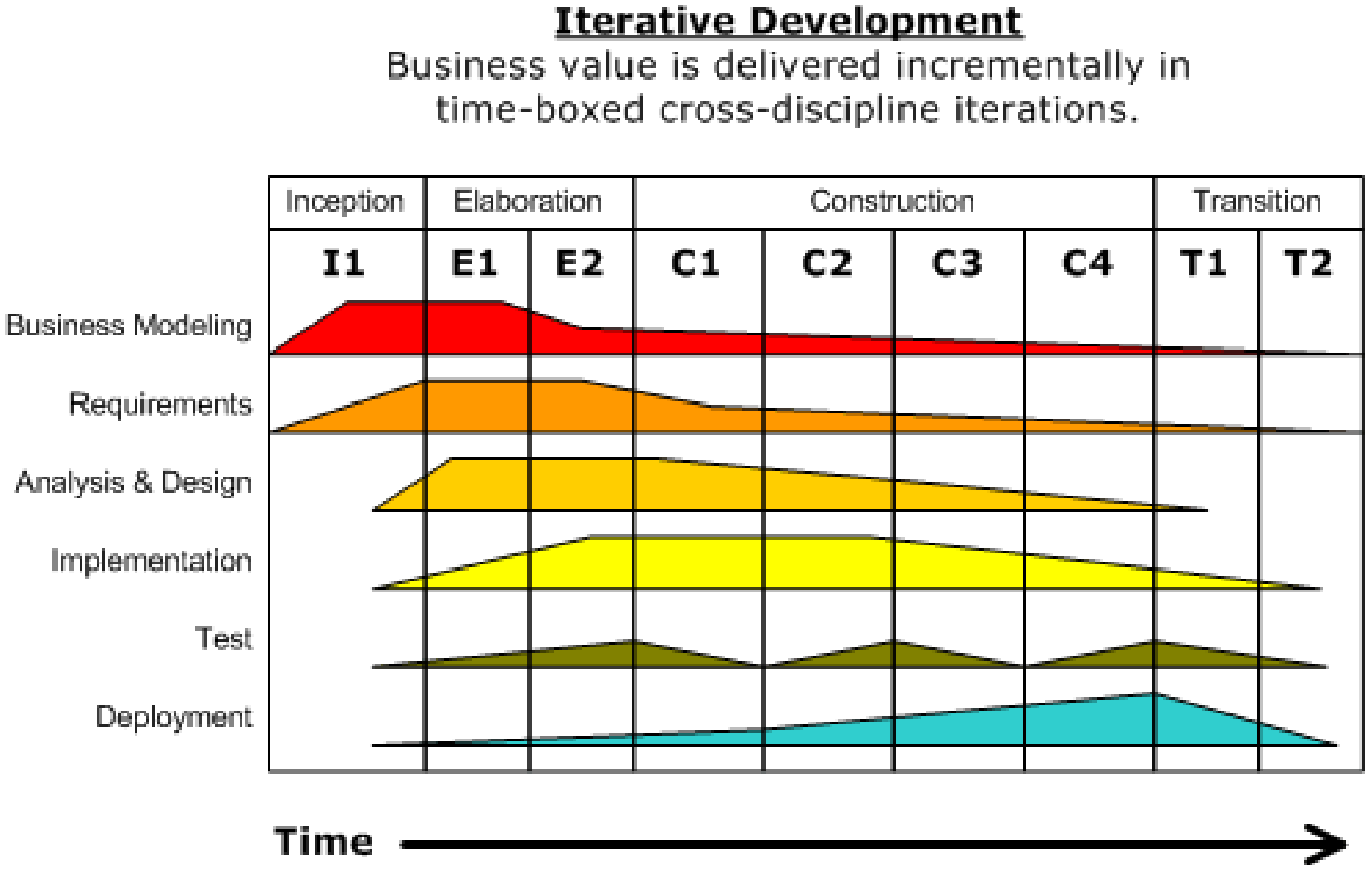
RUP

- Rational Unified Process
- Complement to UML (Unified Modeling Language)
- Iterative approach for object-oriented systems, strongly embraces use cases for modeling requirements
- Tool-supported (UML-tools, ClearCase)

RUP phases

- *Inception – Getting the objectives clear*
  - *E*stablish scope, boundaries, critical use cases, candidate architectures, schedule, and cost estimates
- *Elaboration*
  - Foundation of architecture, establish tool support, get all use cases
- *Construction*: manufacturing process, one or more releases
- *Transition*: release to user community, often several releases

RUP: Two-dimensional process



Differences for developers

- *Agile*: knowledgeable, collocated, collaborative

- *Heavyweight*: plan-driven, adequate skills, access to external knowledge

Differences for customers

- *Agile*: dedicated, knowledgeable, collocated, collaborative, representative, empowered

- *Heavyweight*: access to knowledgeable, collaborative, representative, empowered customers

Differences for requirements

- *Agile*: emergent, rapid change
- *Heavyweight*: knowable early, stable

Differences for architecture

- *Agile*: designed for current requirements

- *Heavyweight*: designed for current and foreseeable requirements

Differences for size

- *Agile*: smaller teams and products

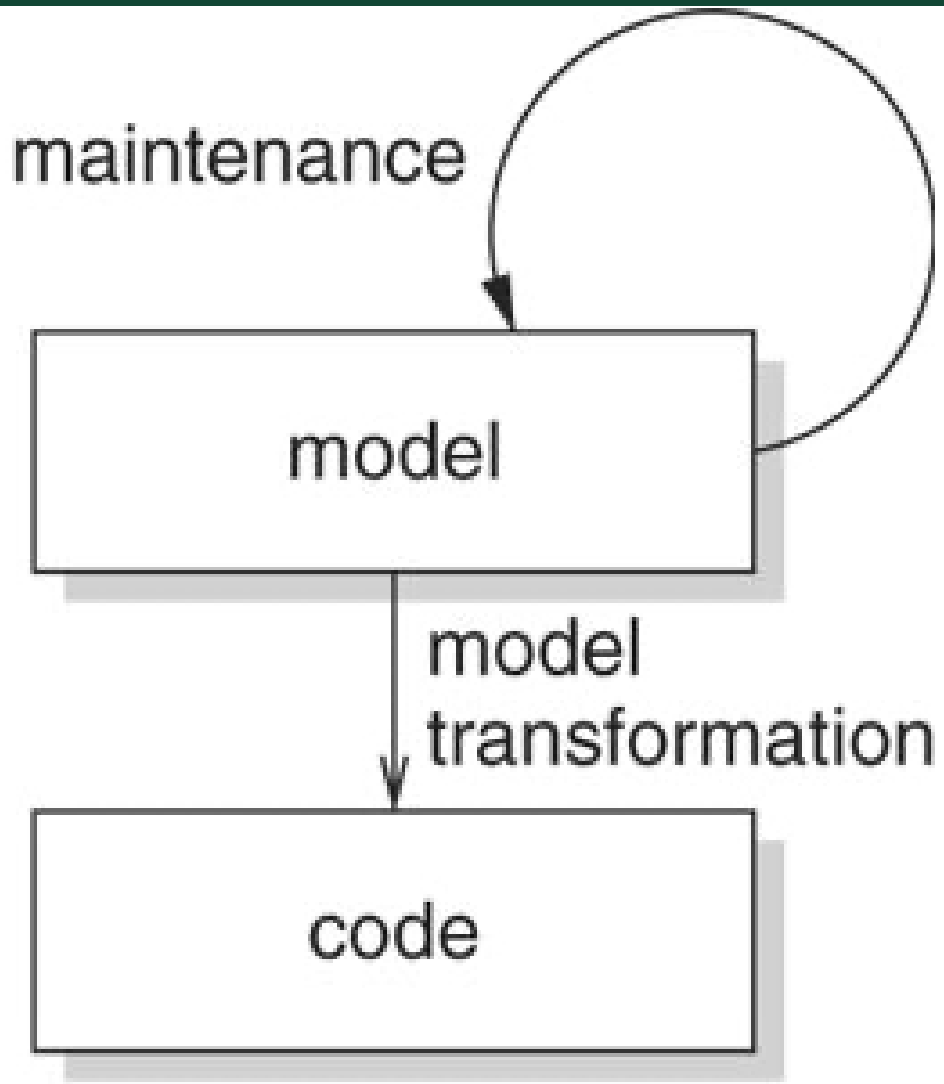
- *Heavyweight*: larger teams and products

Differences for primary objective

- *Agile*: rapid value

- *Heavyweight*: high assurance

MDA – Model Driven Architecture



Essence of MDA

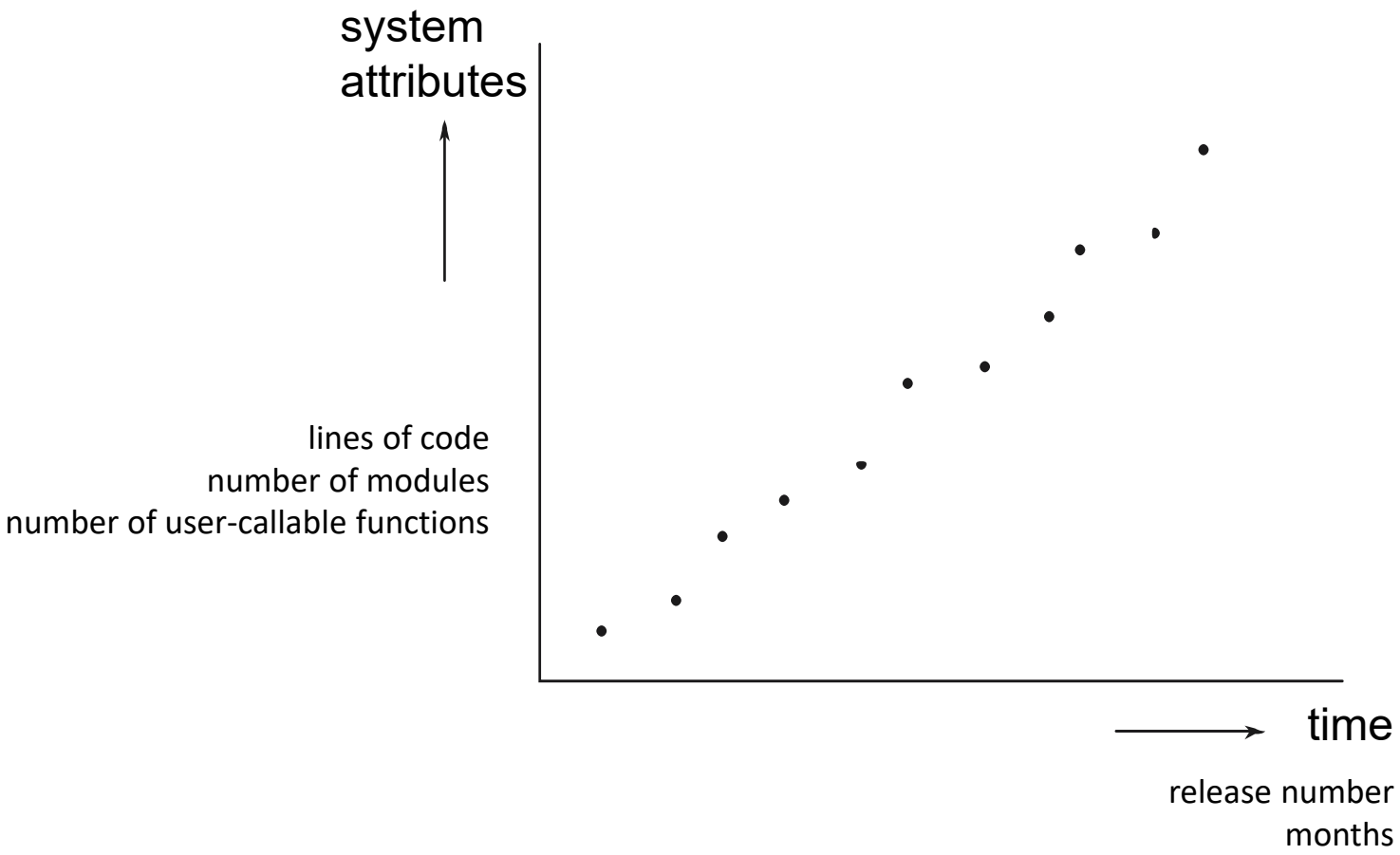
- Platform Independent Model (PIM)
- Model transformation and refinement
- Resulting in a Platform Specific Model (PSM)

Maintenance or Evolution

- Observations
  - Systems are not built from scratch
  - There is time pressure on maintenance

- Five laws of software evolution
  - Continuing change
  - Increasingly complexity
  - Program evolution
  - Incremental growth limit

Software Evolution



Software Product Lines

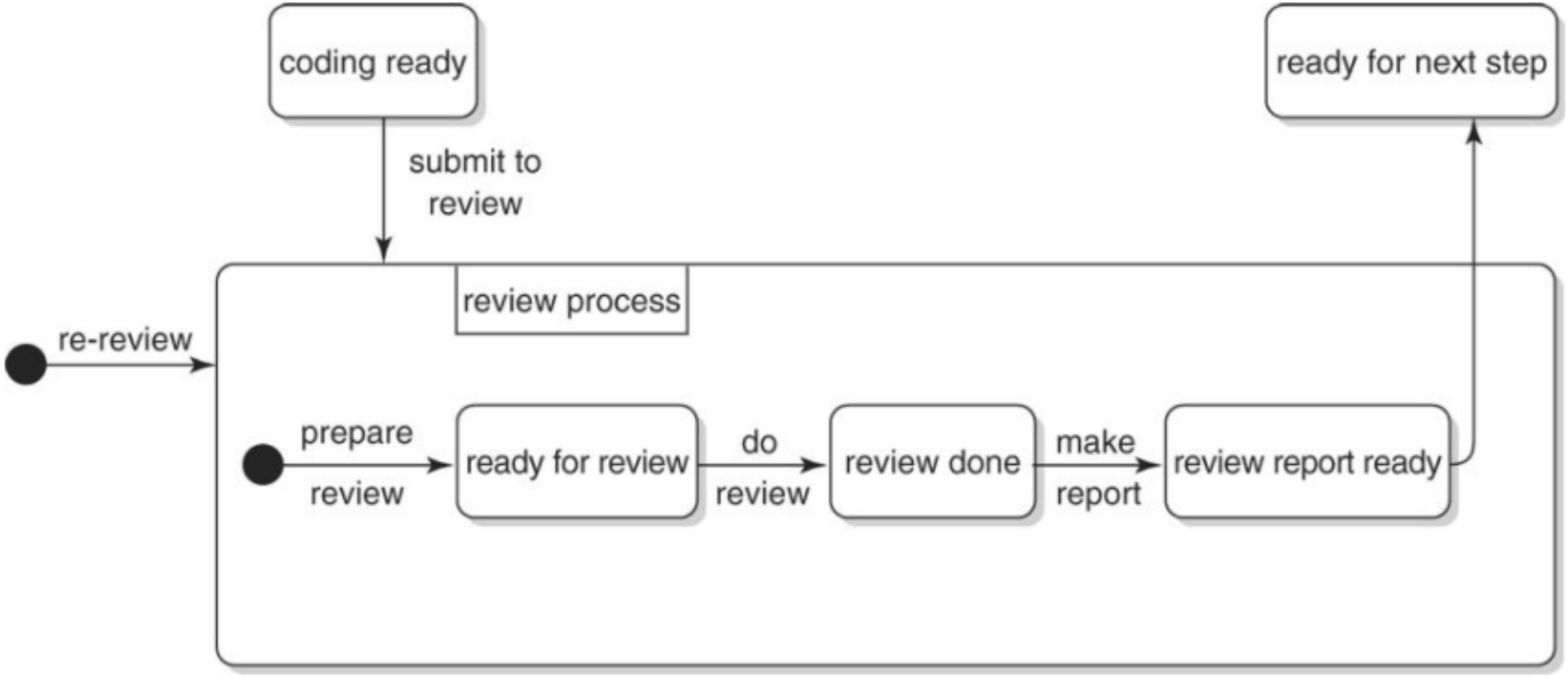
- Developers are not inclined to make a maintainable and reusable product ($$$)
- Not for the *product family* vs. single version product
- Reuse is planned, not accidental
- *Domain Engineering*, and *Application Engineering*

Process modeling

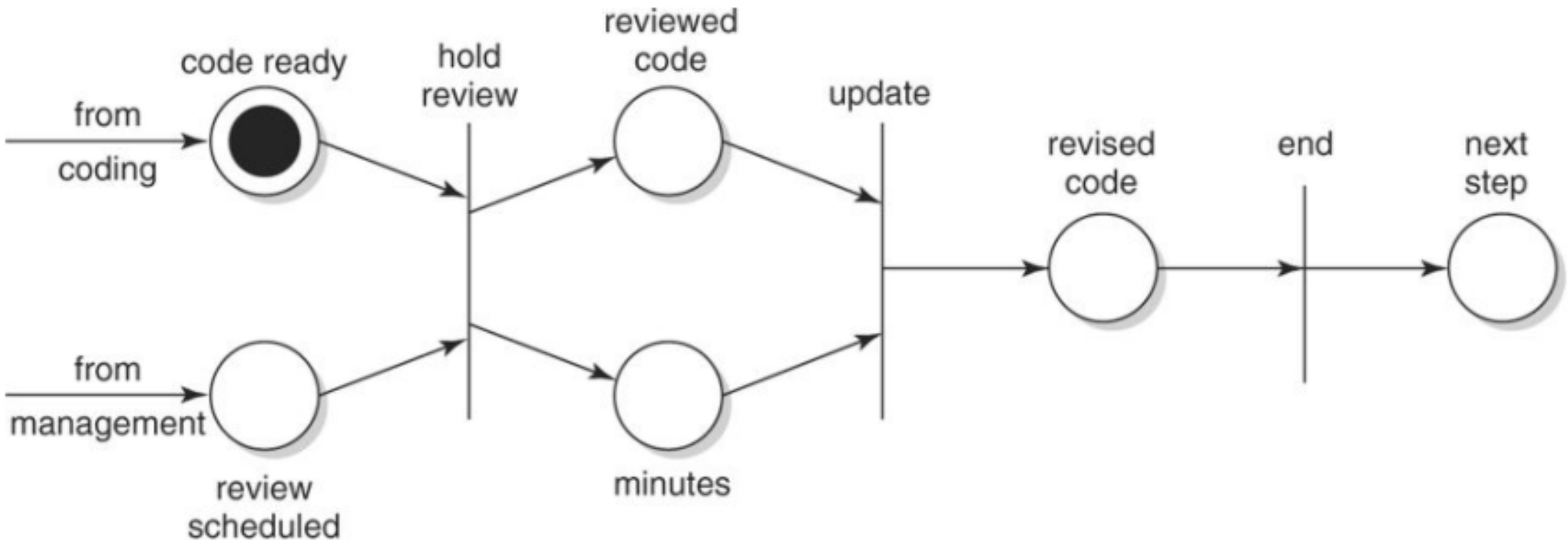
- Describe a software-development process in the form of a "program"

```
function review(document, threshold): boolean;
begin prepare-review;
   hold-review{document, no-of-problems);
   make-report;
   return no-of-problems < threshold
end review;
```

State Transition Diagram



Petri-net view of the review process



Purposes of process modeling

- Facilitates understanding and communication by providing a shared view of the process
- Supports management and improvement; it can be used to assign tasks, track progress, and identify trouble spots
- Serves as a basis for automated (management) support (usually not fully automatic)

Caveats of process modeling

- Not all aspects of software development can be caught in an algorithm
- A model is a simplification of reality
- Progression of stages differs from what is actually done
- Some processes (e.g. learning the domain) tend to be ignored
- No support for transfer across projects

Summary

- Traditional models focus on *control* of the process
- There is no one-size-fits-all model; each situation requires its own approach
- A pure project approach inhibits reuse and maintenance
- There has been quite some attention for process modeling, and tools based on such process models