CS4507 Advanced Software Engineering

Lectures 2 & 3: Software Development Lifecycle Models

A O’Riordan, 2015

Some diagrams from Sommerville, some notes from Maciaszek/Liong

Lifecycle Model

- *Software development* consists of a number of *phases* such as design and implementation.

- For a *project* these are organized into a *process model* or *lifecycle model* indicating the order and frequency of each phase as well as defining the *quality attributes* and *deliverables* of the project.

- Some lifecycles like the waterfall execute each phase only once, others such as with iterative, incremental development execute phases repeatedly.

Software Process

- Process forms the basis of management control, and establishes the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.,) are produced, milestones are established, quality ensured, and change is properly managed. (Pressman)

- Process can generally be characterised as either *heavy* (requiring large effort) or *light/agile*.
  - advantages of a heavy process, *e.g.* RUP, include greater control and repeatability
  - disadvantages of a heavy process are that they can be time consuming, inflexible and misunderstood

Software Development Phases

- *Inception*: initial vision and formulation of plan;
- *Requirements specification*: defining what the system should do;
  - *functional requirements* relate the features or services the software provides
  - non-functional requirements indicate required qualities
- *Design*: defining the organization of the system and how it will be constructed;
  - "Software design is a description of the structure of the software to be implemented, the data which is part of the system, the interfaces between system components and sometimes the algorithms used" (Sommerville);
  - design can be divided into high-level or *architectural design* and low-level or *detailed design;*
  - architectural design is responsible for the overall organization the design. How the software is broken into subsystems and how they interrelate.

Software Development Phases *continued*

- *Implementation*: implementing the system;
  - software design is realized as a set of programs and data stores

- *Integration and testing:* individual parts of systems are integrated and tested;

- *Evolution*: changing the system in response to changing customer needs.
  - includes the repair of software defects (*bug fixes*)
  - *new releases* and enhancements
  - *operations and maintenance*

Some Lifecycle Models

There are many lifecycle models, some are *plan-driven* such as the Waterfall model, others are agile where planning is incremental

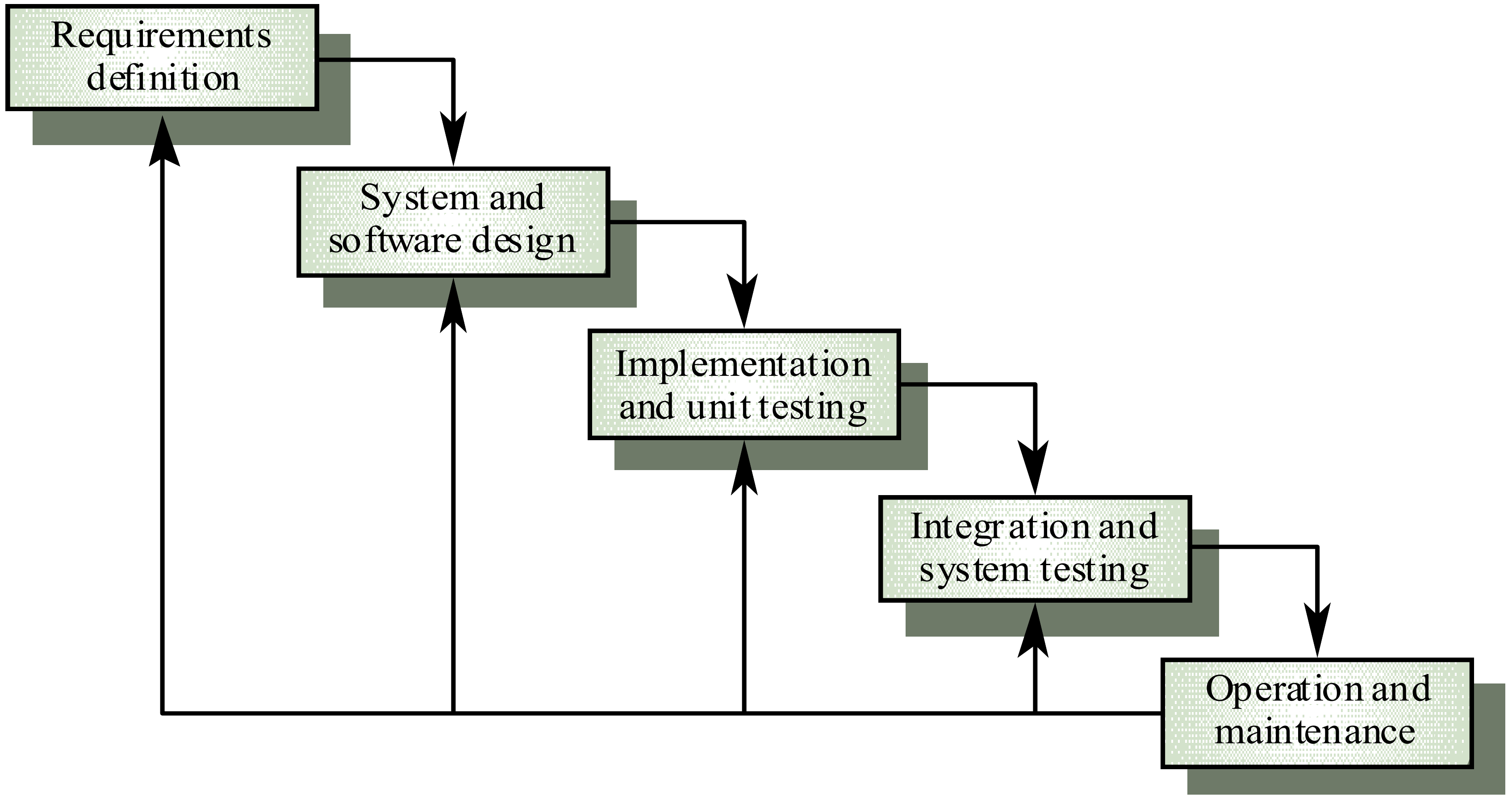
Here are some popular lifecycle models:

- Waterfall model
- V-model
- Evolutionary/Prototyping model
- Incremental model
- Spiral model
- RUP ®
- MDA ®
- Agile models (Scrum, XP) covered in separate lectures

and there are more! not covered

- Dual Vee Model
- WinWin Spiral Model
- Cleanroom
- Unified Software Development Process
- ICONIX Process
- MBASE
- Eclipse Modelling Framework
- DevOps
- more agile methods – Crystal Clear, ASD, FDD, DSDM

Waterfall Model



Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

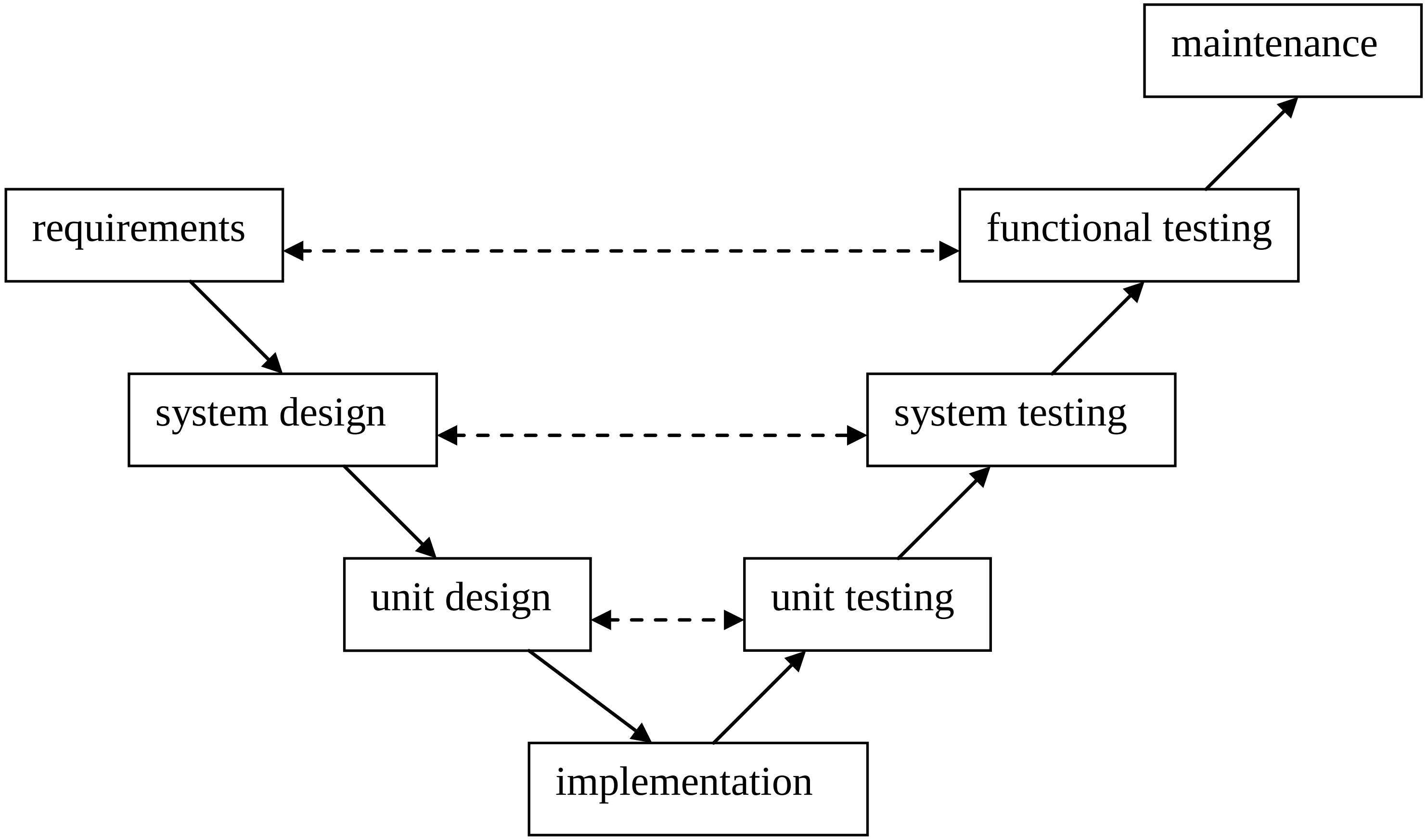
Characteristics of Waterfall Model

- First described by Royce in 1970.
- Consists of a set of phases that a project progresses through in a sequential order.
- Each phase must be completed before the project can progress to the next phase.
- At the end of each phase is some form of gateway, usually a formal review where that decision is made.
- There is no overlap between phases.

Characteristics of Waterfall Model *continued*

- It is straightforward, simple to understand and use.
- Deliverables are frozen at the end of each phase and serve as a baseline for the following phases.
- You do not see the software until the end of the project (big bang software development).
- Changes are not supposed to happen or are limited or are tightly controlled.

V-model

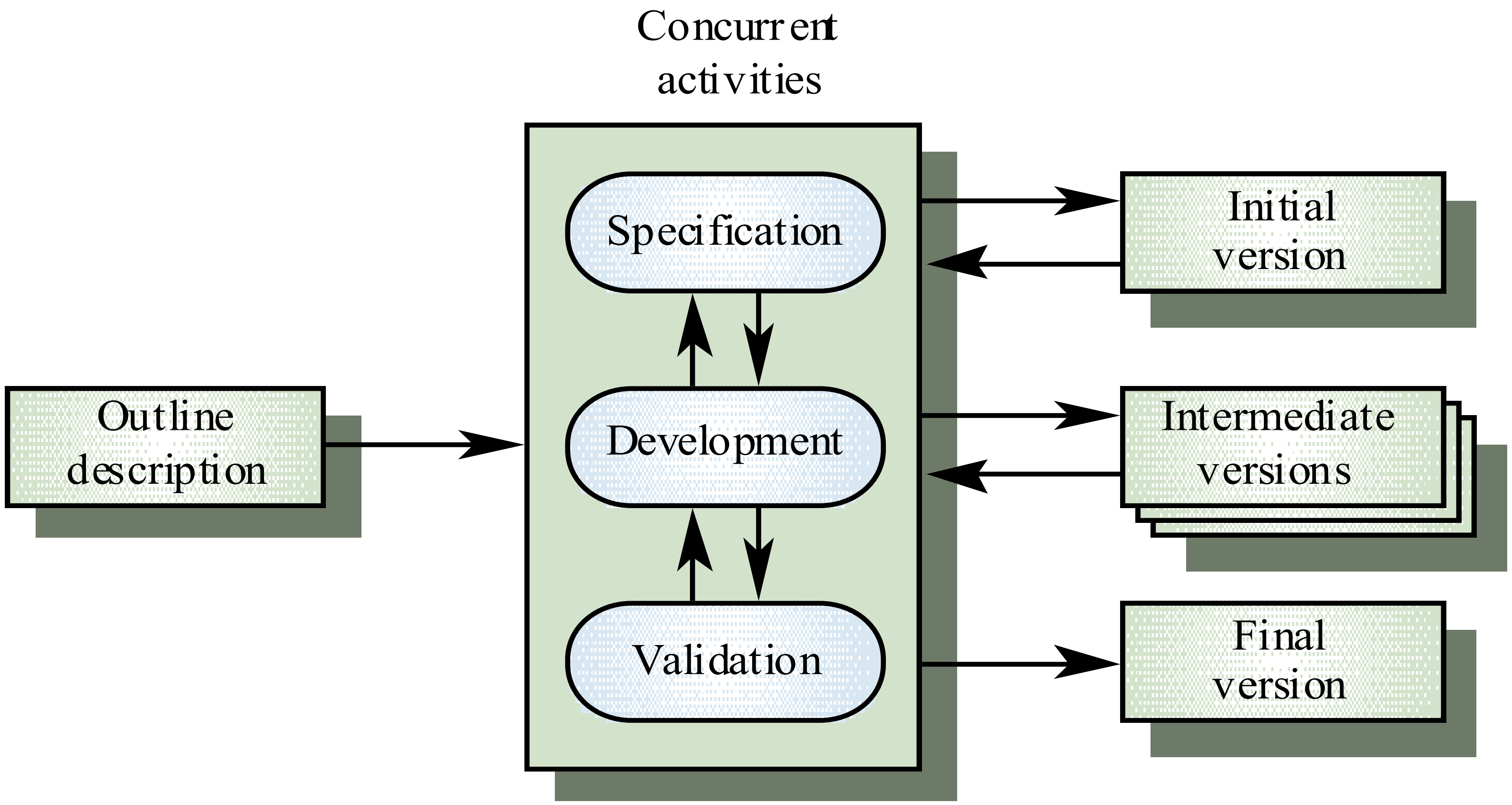

maintenance
requirements
functional testing
system design
system testing
unit design
unit testing
implementation

V-model: A refinement of Waterfall

- V-model is refinement of the Waterfall model that makes explicit the relationship between development activities and V&V activities. Equal billing is given to building the software and validating it.

- Described by Jensen and Tonies, 1979.

Evolutionary Model



Evolutionary Model Characteristics

- In exploratory prototyping the objective is to work with customers and to evolve the software.
- Allows you to put functional software into the hands of the customer much earlier than either the waterfall.
- Requires careful planning at both the project management level and the technical level.

Prototyping

- A prototype is an initial version of a software system that is used to demonstrate concepts, try out design options, find out more about the problem or elicit feedback from the customer.

- Prototyping is also a risk management technique

- A prototype can be *throw-away*, for example created early to assess risk but not used in the software design
  - in contrast a prototype can be *evolutionary* where it evolves into the final software system
  - an *experimental prototype* investigates some aspect of a design solution

Should we prototype it?

Pros of prototyping

- Customer needs are better accommodated
- Helps clarify requirements
- Problems are detected earlier
- Prototype gives you *working code* which can evolve in final product

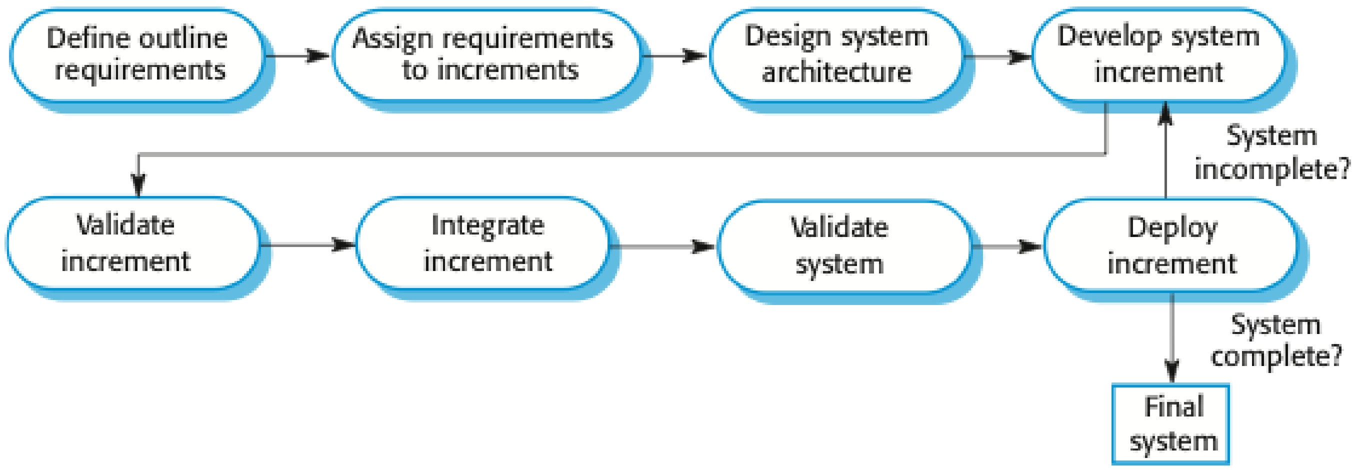
Cons

- Extra time and effort
- Prototype may not capture non-functional requirements
- If evolutionary prototyping is used the performance of the resulting system may be worse

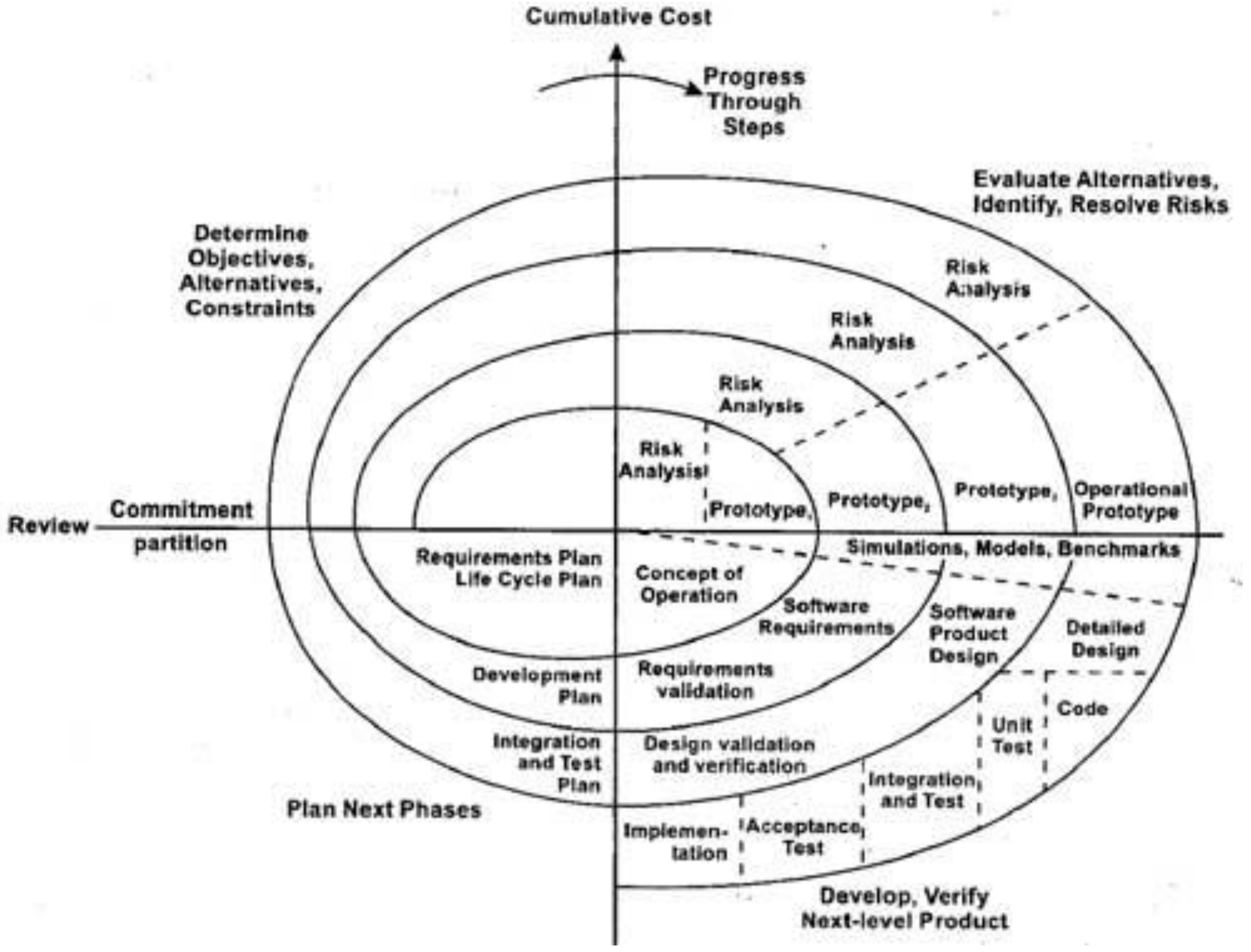
Iterative lifecycle model

- A iterative lifecycle entails all or some phases being repeated.
- Most modern software process models are iterative.
- An Iterative lifecycle leads to *increments*, improved or extended version of the software at each iteration.
- An iterative lifecycle assumes *short iterations* between increments, this can be as short as two weeks in the case of some agile approaches.
- Iterative lifecycles lead to frequent *builds*, executable code that is the main output of the iteration.

Integrating increments



Spiral model



Spiral model characteristics

- The spiral process model was originally proposed by Boehm in 1988.
- It is a *risk-based* model that makes risk assessment explicit
  - risks can internal to project or external
- No fixed phases such as specification or design; loops in the spiral are chosen depending on what is required
- Risks are explicitly assessed and resolved throughout the process
  - Risk analysis involves cost-benefit and threats-opportunities analyses
- Spiral model has been described as a meta-model or reference model for models, rather than a lifecycle model itself (Ghezzi)

Spiral model phases

- Each loop in spiral represents a phase of development
- Starting on inside with initial planning
- Each loop is divided into four quadrants:
  - Objective setting: objectives for the phase are identified
  - Risk assessment: risks are assessed and activities put in place to reduce the key risks
  - Development and V&V: encompasses all development activities
  - Planning: project is reviewed and the next phase of the spiral is planned
- *WinWin Spiral Model* (not covered) extends Spiral Model with Theory W, a management theory (Boehm, 1998)

RUP

- RUP $^{®}$ (Rational Unified Process $^{®}$) by IBM is a modern software process (Krutchen, 2003).

- RUP is based on the Unified Software Development Process (USDP) defined by the creators of the UML as a *unified process* based on earlier process models (Booch method, Objectory, OMT).

- RUP is a mature widely used commercial product supported by the *RUP platform* and a number of tools.

- RUP is *use-case driven*, *architecture-centric*, *iterative and incremental*.

- RUP accounts for most aspects of software engineering including planning, project management, V&V as well as development.

RUP Cycles and Phases

- The software lifecycle is broken into *cycles*, each cycle working on new generation of the product.
- RUP has different perspectives: *static* (showing activities), *dynamic* (showing phases) and *practice* (showing good practices).
- RUP divides one development cycle in four consecutive *phases*:
  - Inception phase
  - Elaboration phase
  - Construction phase
  - Transition phase
- Each phase consists of one or more iterations.

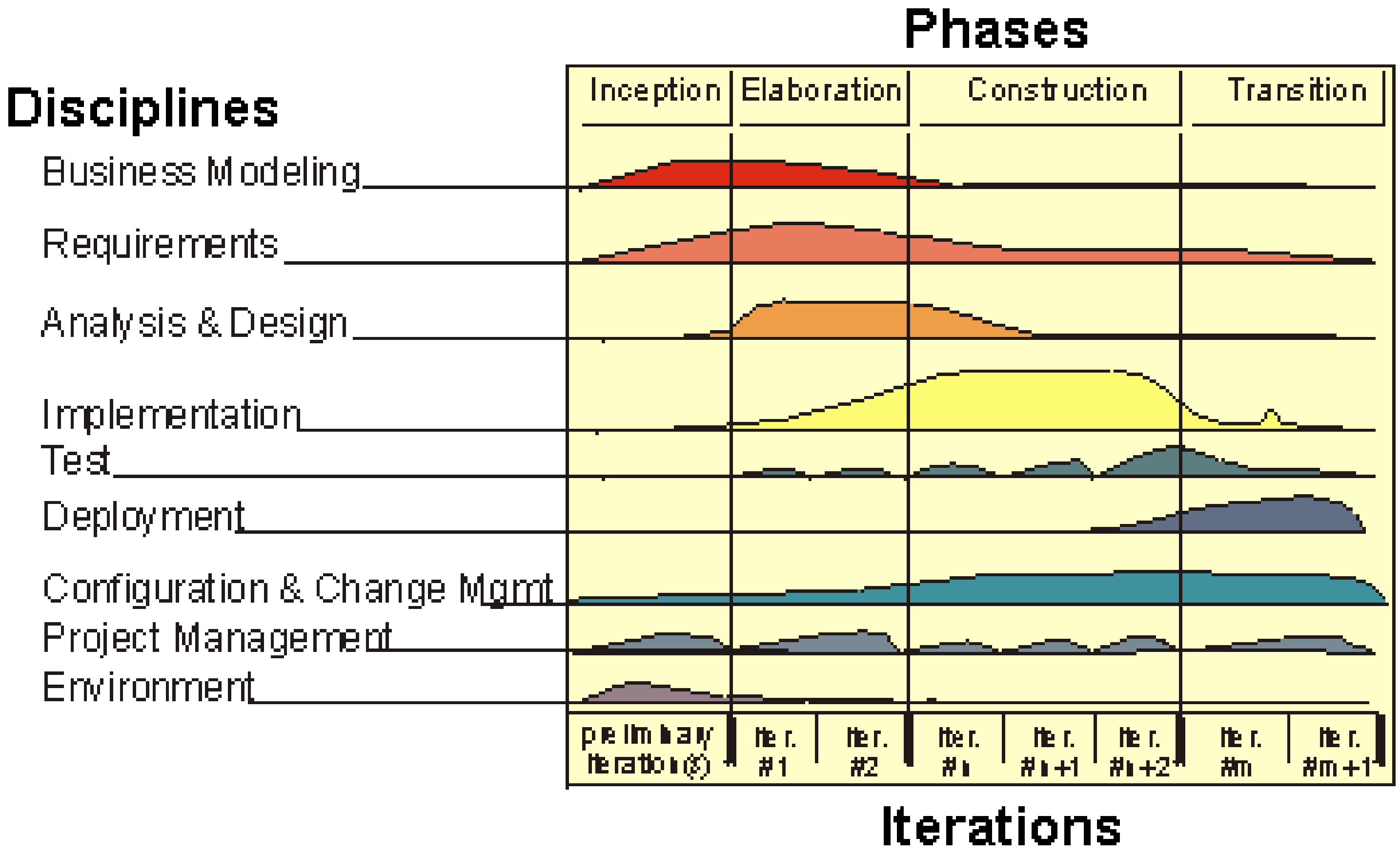
RUP Phases and Objectives

- Inception
  - establish the business case for the system.
  - *Lifecycle Objectives*
- Elaboration
  - develop an understanding of the problem domain and the system architecture.
  - *Lifecycle Architecture*
- Construction
  - system design, programming and testing.
  - *Initial Operational Capability*
- Transition
  - deploy the system in its operating environment.
  - *Product Release*

RUP Disciplines

- Iterations should be short and the output a tested, integrated and executable (partial) system.
- Each iteration involves some of 9 disciplines (shown on left of diagram)
- The “humps” in the diagram give a rough indication of the relative effort expended on each discipline during each phase/iteration.

RUP in a single diagram



Benefits of RUP

- Assesses risks: early and continuous documentation of most urgent and the most probable risks.
- Includes Planning and Follow up
- Uses UML, a common modelling language
  - Use cases may be used as test cases, end-user documentation and design description
- Keeps Glossary: A clearly defined terminology makes everybody in a project aware of the exact meaning of a term. At every time.
- Includes V&V activities

Risks

- Both RUP and Spiral model include *risk analysis* (=risk identification + resolution)

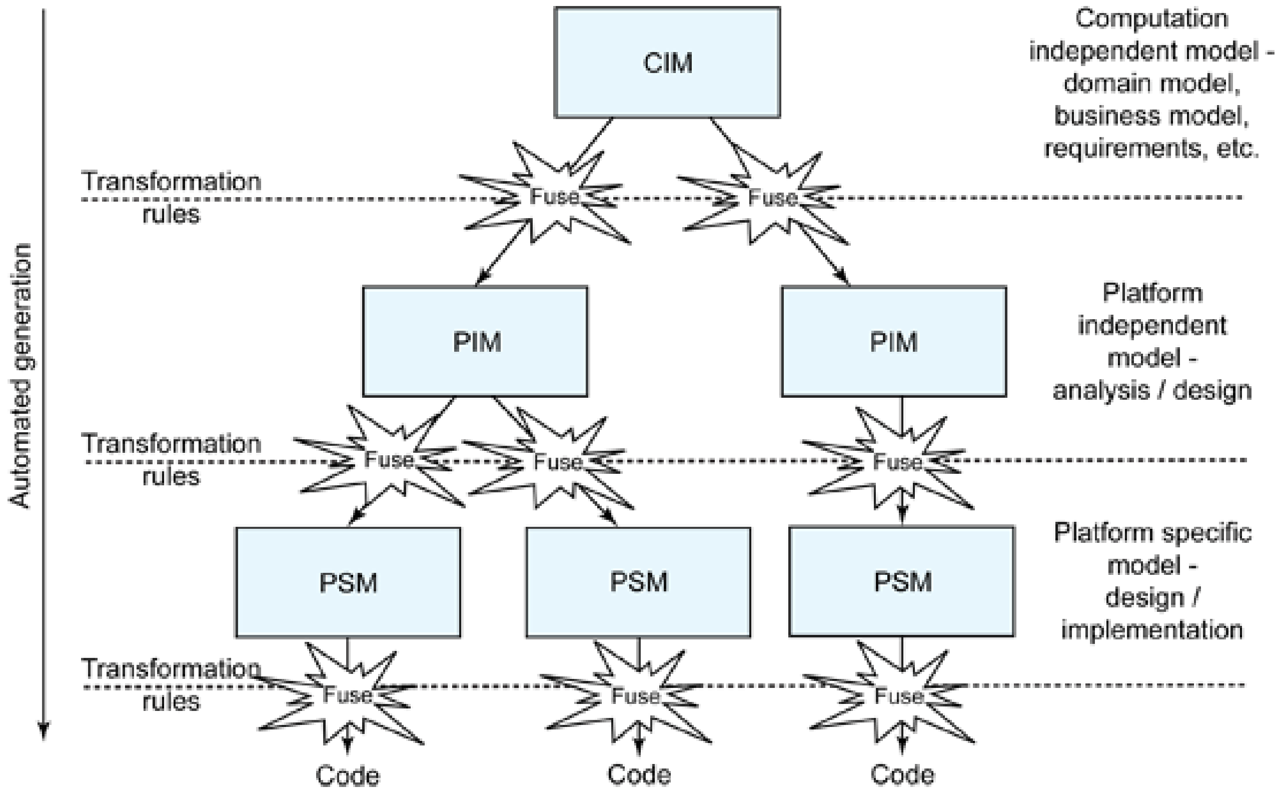
- What risks? Examples include
  - Customer relations risks, *e.g.* too high expectations
  - Staff risks, *e.g.* personnel shortfalls
  - Unrealistic schedules and budgets
  - Immature process
  - Project too large
  - Developing the wrong functions
  - Developing the wrong user interfaces
  - Requirements changes
  - Technology risks, *e.g.* use of new unproven technology

MDA

- MDA ® (Model-driven Architecture ®) is a modern, *open*, vendor-neutral approach to systems development by the OMG (Object Management Group)
  - described by (Kleppe, 2003) MDA is just one example of *Model-driven engineering (MDE)*

- MDA is a *transformation model*
  - systems development is seen as a series of transformations, from the requirements specification, to design stages to *executable code*.
  - extensive *automation* of the transformations making use of CASE tools
  - each transformation is carefully evaluated
  - output of requirements analysis is a *Platform Independent Model (PIM)*
  - output of design is a *Platform Specific Model (PSM)* each describing how the base model is implemented on a different middleware platform.

MDA Model Transformations


CIM
Computation independent model - domain model, business model, requirements, etc.
Transformation rules
Fuse
Fuse
PIM
PIM
Platform independent model - analysis / design
Transformation rules
Fuse
Fuse
Fuse
PSM
PSM
PSM
Platform specific model - design / implementation
Transformation rules
Fuse
Fuse
Fuse
Code
Code
Code
Automated generation

More about MDA

- MDA enables application interoperability and portability
- MDA encompasses other OMG technologies such as repository services, transaction processing, event handling, and security.
- MDA addresses the complete life cycle of designing, deploying, integrating, and managing applications as well as data using open standards.
- MDA supports evolving standards in application domains