1. Software Process Models

(Sommerville Chapters 4, 17, 19, 12.4)

A software process model is a standardised format for

- planning
- organising, and
- running

a development project.

Hundreds of different models exist and are used, but many are minor variations on a small number of basic models.

In this section we:

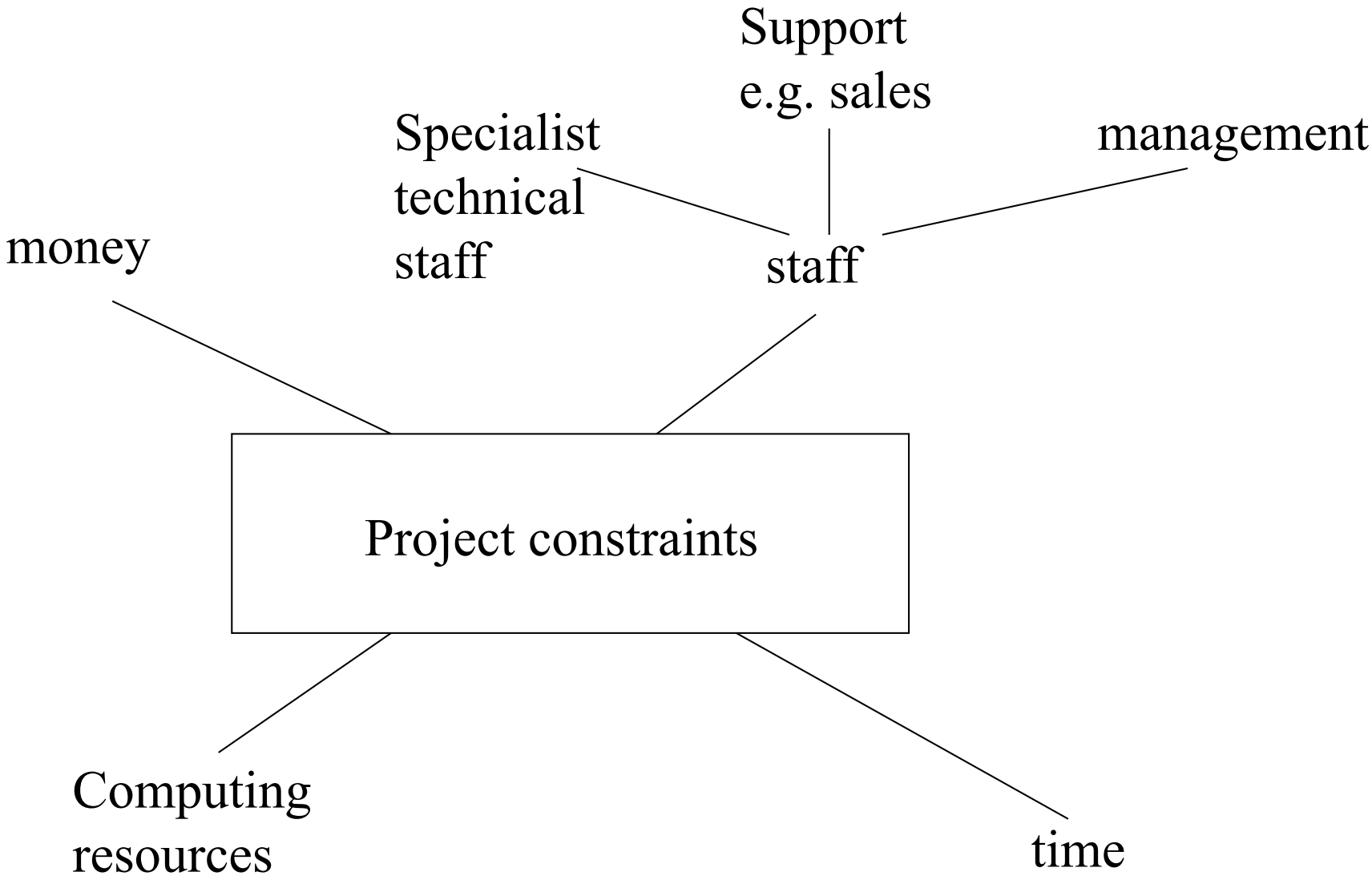
- **survey** the important basic models, and
- consider **how to choose** between them.

1.1. Planning with Models

SE projects usually live with a **fixed financial budget**. (An exception is maintainance?)

Additionally, time-to-market places a strong **time constraint**.

There will be other **project constraints** such as staff.



Examples of Project Constraints

Project planning is the art of scheduling/constraint solving the project parameters, along various dimensions:

time, money, staff …

in order to optimise:

- project risk [low] (see later)
- profit [high]
- customer satisfaction [high]
- worker satisfaction [high]
- long/short-term company goals

Questions:

1. What are project parameters?

2. Are there good patterns of organisation that we could copy?

Project parameters describe the whole project,

but we must at least describe:

- resources needed

  (***people, money, equipment, etc***)
- dependency & timing of work

  (***flow graph***, ***work packages***)
- rate of delivery (***reports, code, etc***)

It is impossible to measure **rate of progress**

except with reference to a plan.

In addition to project members, the following may need access to parts of the project plan:

- Management
- Customers
- Subcontractors (outsourcing)
- Suppliers (e.g. licenses, strategic partners)
- Investors (long term investment)
- Banks (short term cash)

1.2. Project Visibility

Unlike other engineers
(e.g. civil, electronic, chemical … etc.)
software engineers do not produce anything ***physical***.

It is inherently difficult to monitor an SE project due to **lack of visibility**.

This means that SE projects must produce

additional deliverables (***artifacts***)

which **are visible**, such as:

- ***Design documents/ prototypes***
- ***Reports***
- ***Project/status meetings***
- ***Client surveys* (*e.g. satisfaction level*)**

1.3. What is a Software Process Model?

Definition.

A (software/system) *process model* is a description of the sequence of activities carried out in an SE project, and the relative order of these activities.

It provides a fixed **generic framework** that can be tailored to a specific project.

Project specific ***parameters*** will include:

- Size, (person-years)
- Budget,
- Duration.

project plan =

process model + project parameters

There are hundreds of different process models to choose from, e.g:

- ***waterfall,***
- ***code-and-fix***
- ***spiral***
- ***rapid prototyping***
- ***unified process (UP)***
- ***agile methods, extreme programming (XP)***
- ***COTS …***

But most are minor variations on a small number of basic models.

By changing the process model, we can **improve and/or tradeoff**:

- ***Development speed (time to market)***
- ***Product quality***
- ***Project visibility***
- ***Administrative overhead***
- ***Risk exposure***
- ***Customer relations, etc, etc.***

Normally, a process model covers the entire **lifetime of a product**.

From ***birth of a commercial idea***
to ***final de-installation of last release***

i.e. The three main phases:

- ***design***,
- ***build***,
- ***maintain***. (50% of IT activity goes here!)

We can sometimes **combine** process models e.g.

1. waterfall inside evolutionary – onboard shuttle software

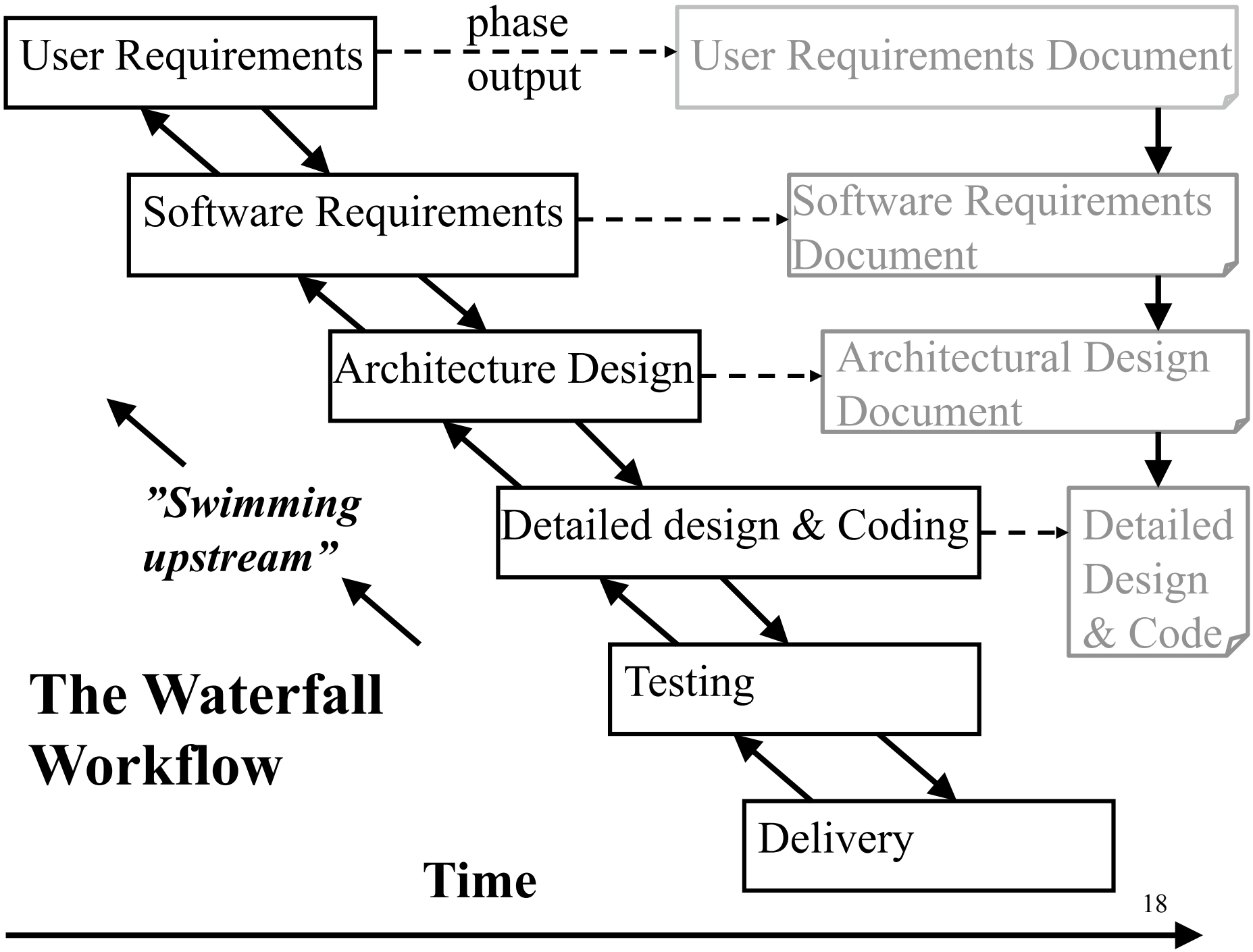
2. Evolutionary inside waterfall – e.g. GUI prototyping

We can also ***evolve the process model*** together with the product to account for product maturity,

e.g. *rapid prototyping → waterfall*

1.4. The Waterfall Model

- The waterfall model is the classic process model – it is widely known, understood and used.
- In some respect, waterfall is the ”common sense” approach.
- R.W. Royce, Managing the Development of Large Software Systems: Concepts and Techniques, Proc. IEEE Westcon, IEEE Press, 1970.



The Waterfall Workflow

Advantages

1. Easy to understand and implement.
2. Widely used and known (in theory!)
3. Fits other engineering process models: civil, mech etc.
4. Reinforces good habits: define-before- design, design-before-code
8. Identifies deliverables and milestones
9. Document driven: *People leave, documents don't*
   Published documentation standards: URD, SRD, … etc. , e.g. ESA PSS-05.
10. Works well on large/mature products and weak teams.

Disadvantages I

1. Doesn’t reflect iterative nature of exploratory development.
2. Sometimes unrealistic to expect accurate requirements early in a project
3. Software is delivered late, delays discovery of serious errors.
4. No inherent risk management
5. Difficult and expensive to change decisions, ”*swimming upstream*”.
6. Significant administrative overhead, costly for small teams and projects.

1.5. Code-and-Fix

This model starts with an informal general product idea and just develops code until a product is ”ready” (or money or time runs out). Work is in random order.

Corresponds with no plan! (**Hacking**!)

Advantages

1. No administrative overhead
2. Signs of progress (code) early.
3. Low expertise, anyone can use it!
4. Useful for small "***proof of concept***" projects, e.g. as part of risk reduction.

Disadvantages

1. Dangerous!
    1. No visibility/control
    2. No resource planning
    3. No deadlines
    4. Mistakes hard to detect/correct
2. Impossible for large projects, communication breakdown, chaos.

1.6. Evolutionary Development Types

Type 1: Exploratory Development: customer assisted development that ***evolves a product*** from ignorance to insight, starting from core, well understood components (e.g. GUI?)

Type 2: Throwaway Prototyping: customer assisted development that ***evolves requirements*** from ignorance to insight by means of lightweight disposable prototypes.

1.7. Type 1: Spiral Model

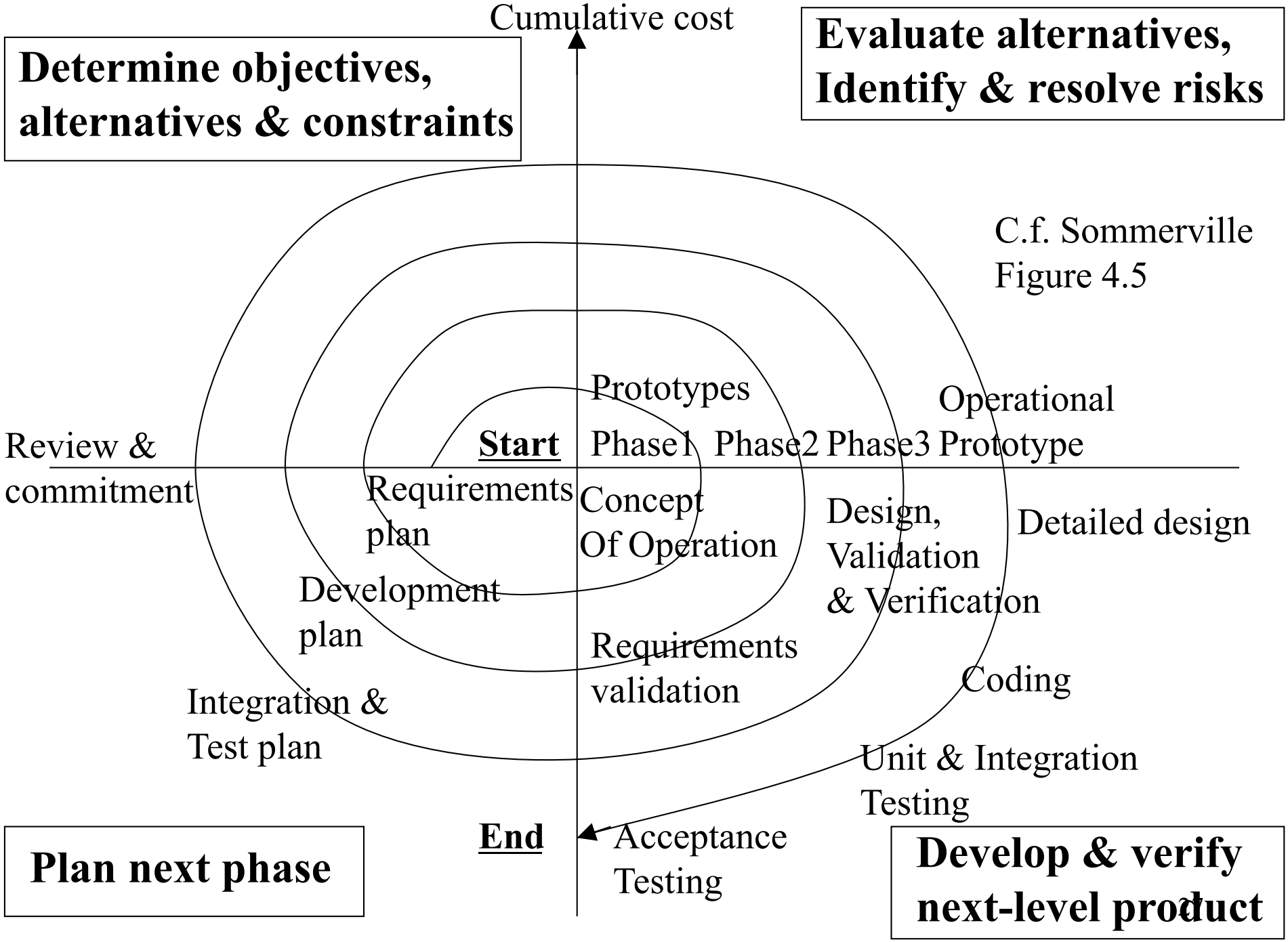
Extends waterfall model by **adding iteration to explore /manage risk**

Project risk is a moving target. Natural to progress a project cyclically in **four step phases**

1. Consider alternative scenarios, constraints
2. Identify and resolve risks
3. Execute the phase
4. Plan next phase: e.g. user req, software req, architecture … then goto 1

In 1988 Boehm developed the spiral model as an iterative model which includes ***risk analysis*** and ***risk management***.

Key idea: on each iteration identify and solve the sub-problems with the ***highest risk***.


Cumulative cost
Determine objectives, alternatives & constraints
Evaluate alternatives, Identify & resolve risks
C.f. Sommerville Figure 4.5
Prototypes
Operational Prototype
Review & commitment
Start
Phase1
Phase2
Phase3
Requirements plan
Concept Of Operation
Design, Validation & Verification
Detailed design
Development plan
Requirements validation
Coding
Integration & Test plan
Unit & Integration Testing
End
Acceptance Testing
Plan next phase
Develop & verify next-level product

Advantages

1. Realism: the model accurately reflects the iterative nature of software development on projects with unclear requirements
2. Flexible: incoporates the advantages of the waterfall and evolutionary methods
3. Comprehensive model decreases risk
4. Good project visibility.

Disadvantages

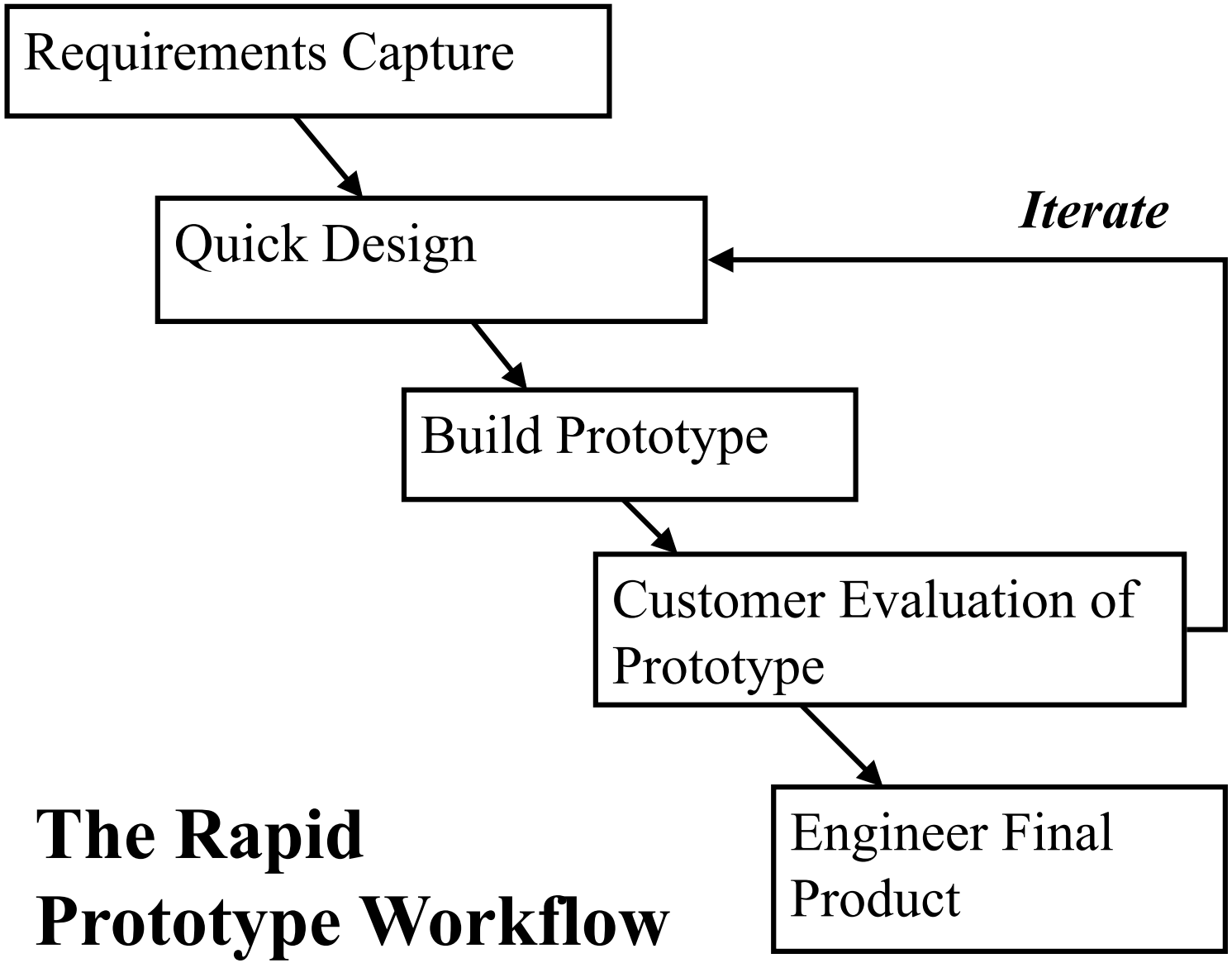
1. Needs technical expertise in risk analysis and risk management to work well.
2. Model is poorly understood by non-technical management, hence not so widely used
3. Complicated model, needs competent professional management. High administrative overhead.

1.8. Type 2: Rapid Prototyping

Key idea: *Customers are non-technical and usually don't know what they want.*

Rapid prototyping emphasises requirements analysis and validation, also called:

- ***customer oriented development,***
- ***evolutionary prototyping***



The Rapid Prototype Workflow

Advantages

1. Reduces risk of incorrect user requirements
2. Good where requirements are changing/ uncommitted
3. Regular visible progress aids management
4. Supports early product marketing

Disadvantages I

1. An unstable/badly implemented prototype often becomes the final product. (Migration to a type 1 process!)
2. Requires extensive customer collaboration
   - Costs customers time/money
   - Needs committed customers
   - Difficult to finish if customer withdraws
   - May be too customer specific, no broad market

Disadvantages II

3. Difficult to know how long project will last
4. Easy to fall back into code-and-fix without proper requirements analysis, design, customer evaluation and feedback.

1.9. Type 1: Agile Software Processes

Need for an adaptive process model suited to changes in:

- User requirements
- Customer business models
- Technology
- In-house environment

De-emphasise documentation, esp. URD!

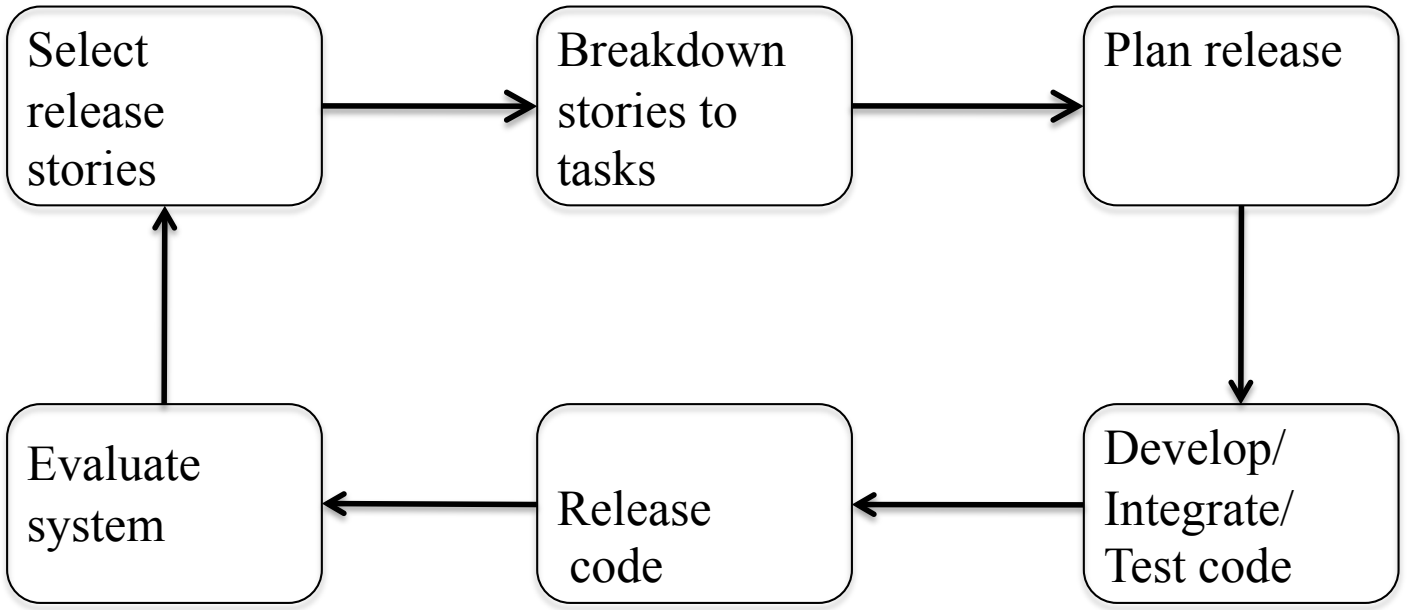
Emphasise change management e.g. reverse engineering design!

Examples include **XP**, Scrum, Agile modeling etc

1.9.1. Agile Principles
(C.f Sommerville Fig 17.3)

- Incremental delivery of software
- Continuous collaboration with customer
- Embrace change
- Value participants and their interaction
- Simplicity in code,

1.9.2. XP Release Cycle



For a sample story see Sommerville Figure 17.6
Same as use-case?

1.9.3. XP Practices (Summary)

1. Incremental planning
2. Small releases
3. Simple design
4. Programming in pairs (egoless programming, see 7.)
5. Test-driven development
6. Software refactoring (needs UML?)
7. Collective ownership: metaphors, standards, code
8. Continuous integration
9. Sustainable pace (No overtime!)
10. On-site customer!

Advantages

1. Lightweight methods suit small-medium size projects
2. Produces good team cohesion
3. Emphasises final product
4. Iterative
5. Test-based approach to requirements and quality assurance

Disadvantages

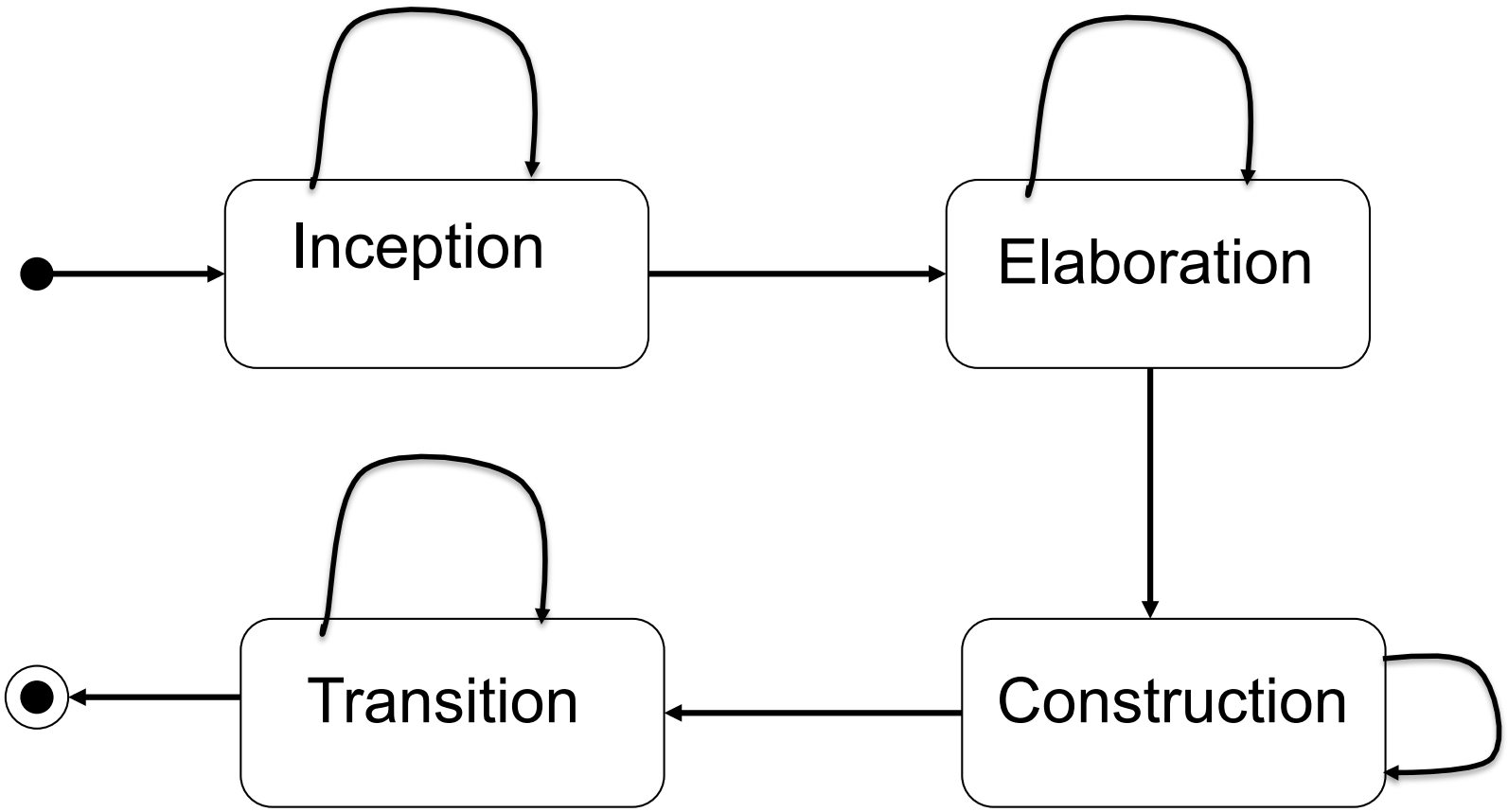
1. Difficult to scale up to large projects where documentation is essential
2. Needs experience and skill if not to degenerate into code-and-fix
3. Programming pairs is costly (but see productivity literature)
4. Test case construction is a difficult and specialised skill.

1.10. Rational Unified Process (RUP)

- Hybrid model inspired by UML and Unified Software Development Process.
- A generic component-based process?
- Three views on the process
  - **Dynamic view**: RUP phases
  - **Static view**: RUP activities
  - **Practise view**: RUP best-practise

Details

- Lifetime of a software product in **cycles**:
- ***Birth, childhood, adulthood, old-age, death***.
- Identify product maturity stages
- Each project iteration cycle is a phase, culiminating in a new release (c.f. Spiral model)



UP process – RUP phase workflow (drawn as a UML Statechart!)

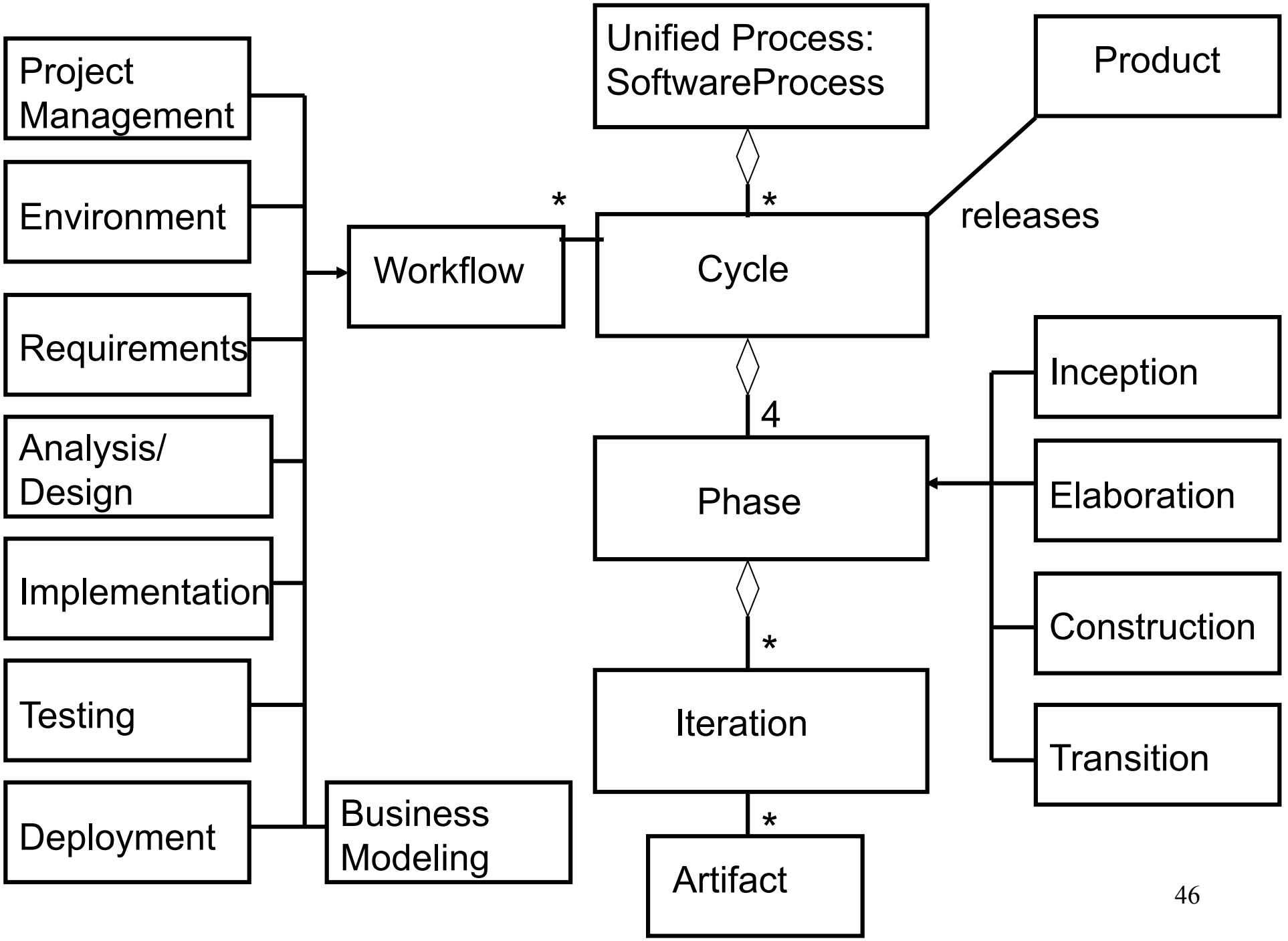
RUP Phases

Phases are goal directed and yield deliverables:

- **Inception** – Establish the business case. Identify external entities (actors, systems). Estimate ROI.
- **Elaboration** – Understand problem domain. Establish architecture, and consider design tradeoffs. Identify project risks. Estimate and schedule project. Decide on build vs. buy.
- **Construction** – Design, program and test. Components are bought and integrated.
- **Transition** – release a mature version and deploy in real world.

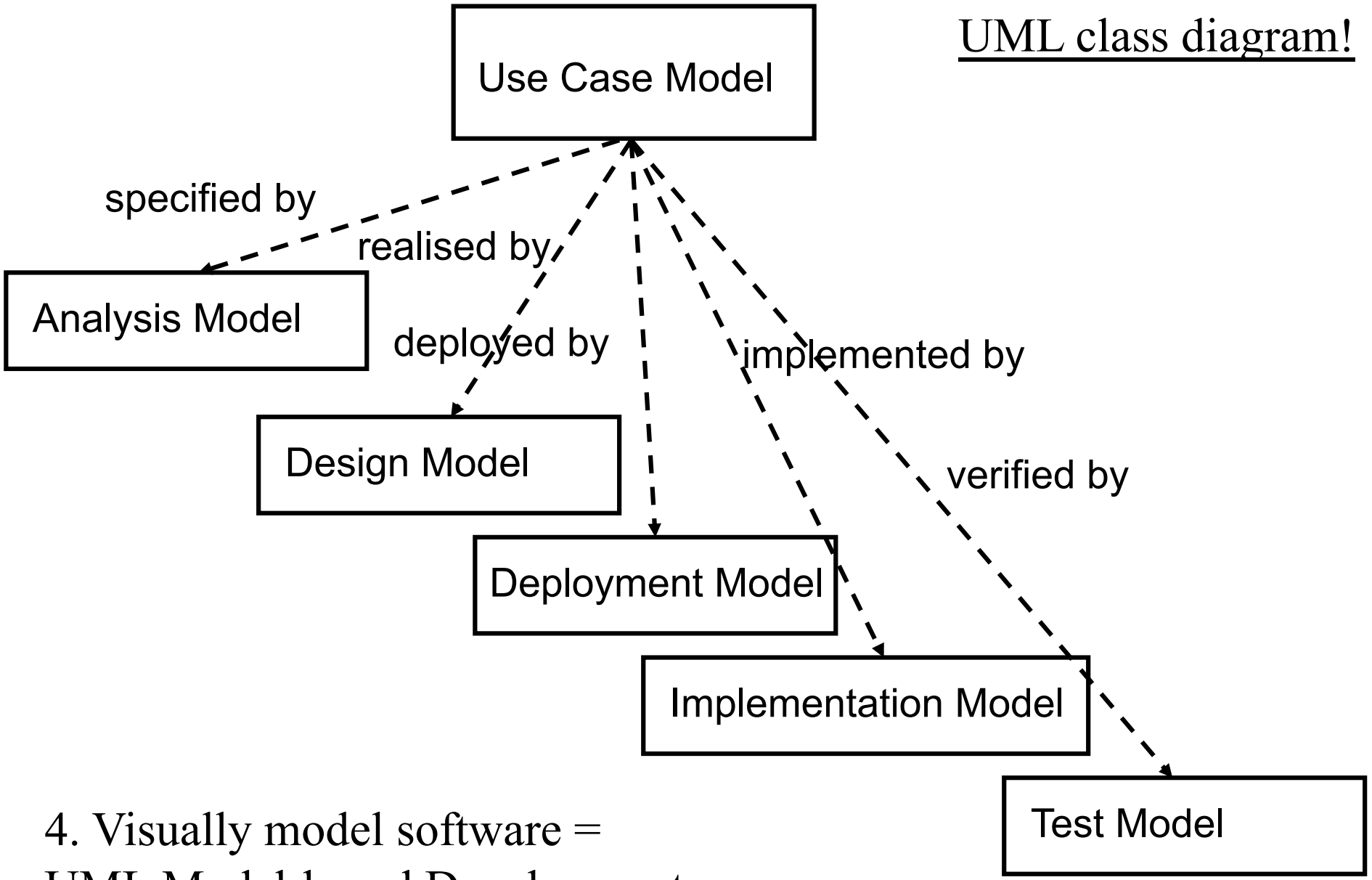
RUP Workflows

- RUP separates *what* and *when* into two orthogonal process views.
- *When* modeled by phases
- *What* modeled by workflows (c.f. Sommerville Figure 4.13)
- Any workflows can be active in any phases.
- Anything that instantiates the following diagram is an instance of RUP!!
- An agile instantiation exists (Larman 2002)


Project Management
Environment
Requirements
Analysis/ Design
Implementation
Testing
Deployment
Business Modeling
Workflow
Unified Process: SoftwareProcess
Product
releases
*
*
Cycle
4
Phase
*
Iteration
*
Artifact
Inception
Elaboration
Construction
Transition

RUP Practise View

1. Develop software iteratively
2. Manage requirements
3. Use component-based architectures
4. Visually model software
5. Verify software quality
6. Control software changes



UML class diagram!

4. Visually model software =
UML Model-based Development

Advantages/ Disadvantages

- Difficult to judge without knowing the actual chosen instantiation of the RUP
- Unique use of the UML philosophy of SE.
- Can be as good/better than any other process
- Also as bad/worse than …

1.11. COTS

- **COTS =**

  Commercial Off-The-Shelf software
- Integrate a solution from existing commercial software components using minimal software plumbing
- All projects seek some software re-use – *Holy Grail of SE*
- See also Sommerville Chapter 19.

Possible Approaches

1. **Third-party vendors**: component libraries, Java beans,
2. **Integration solutions**: CORBA, MS COM+, Enterprise Java Beans, software frameworks …
3. **Good software engineering**: application generators (Yacc, Lex … Sommerville 18.3), generic programming (types, polymorphism etc)
4. **Commercial finished packages**: databases, spread sheets, word processors, web browsers, etc.
5. **Design to open source interfaces**: e.g. XML, ISO standards, etc.

Distributed COTS: Web Service Providers

- Integration of web-services: third party ***service providers*** using (XML-based?) ***service models***:
- JINI – extension of Java for service discovery
- SOAP (Simple Object Access Protocol)
- WSDL (Web Services Description Language)
- UDDI (Universal Description, Discovery and Integration)

Advantages

1. Fast, cheap solution
2. Explore solutions with existing products (c.f. your project work!)
3. May give all the basic functionality
4. Well defined **integration project**, easy to run.
5. Open to outsourcing (c.f. Point 4)
6. Build strategic supplier partnerships

Disadvantages

1. Limited functionality/ compromise/ requirements drift.
2. Component identification can be tricky – mistakes occur!
3. Licensing problems: freeware, shareware, etc.
4. Customer lock-in: license fees, maintainance fees, upgrades …
5. Compatibility issues/loss of control:
    1. Between components (doesn't start!)
    2. During maintenance (suddenly stops!)