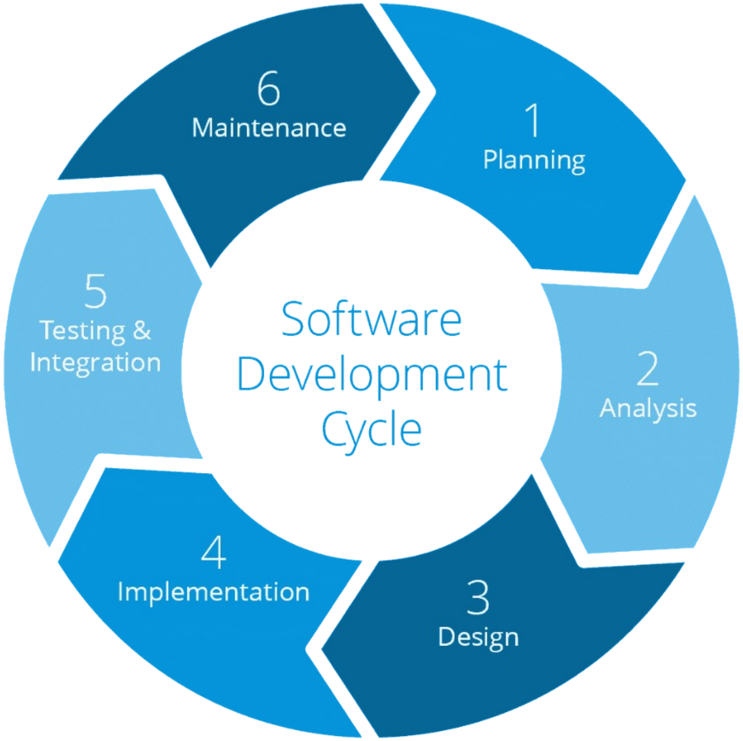
Software Lifecycle Models

A **software lifecycle model** is a standardised format for

- planning
- organising
- running

a new development project.



Hundreds of different kinds of models are known and used.

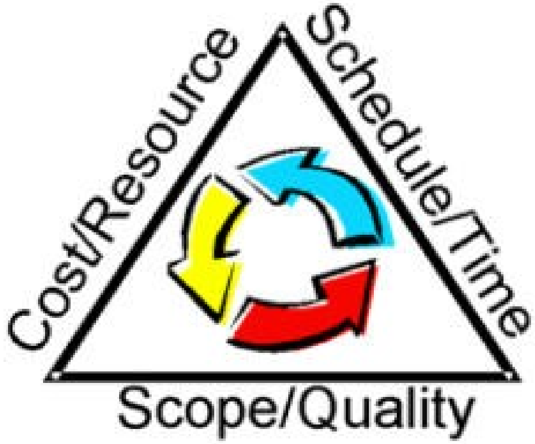
Many are minor variations on just a small number of basic models. In this section we:

- **survey** the main types of model, and
- consider **how to choose** between them.

Planning with Models

SE projects usually live with a **fixed financial budget**.

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



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

A project plan contains much information,
but must at least describe:

- **resources** needed (*people, money, equipment*)
- dependency & **timing** of work (*flow graph, work packages*)
- rate of **delivery** (*reports, code*)

It is impossible to measure the rate of **progress** without having a **plan** as reference

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)

What is a Lifecycle Model?

Definition.

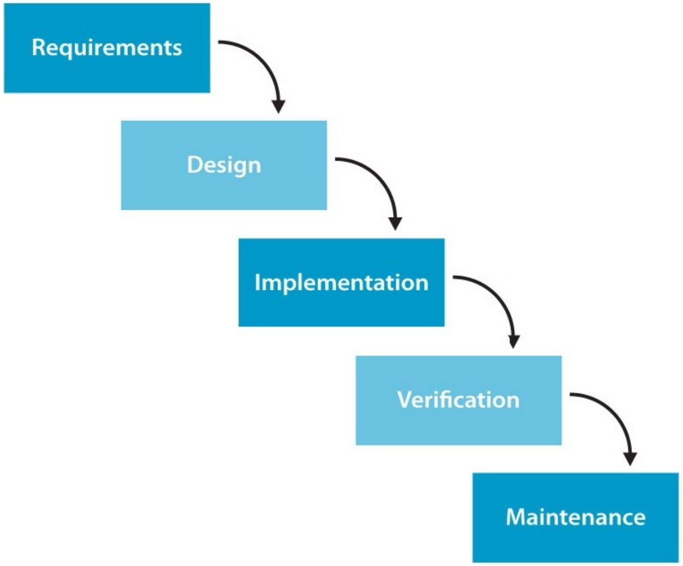
A (software/system) *lifecycle model* is a description of the sequence of activities carried out in an SE project, and the relative order of these activities.

By changing the lifecycle model, we can **improve** and/or **trade off**:

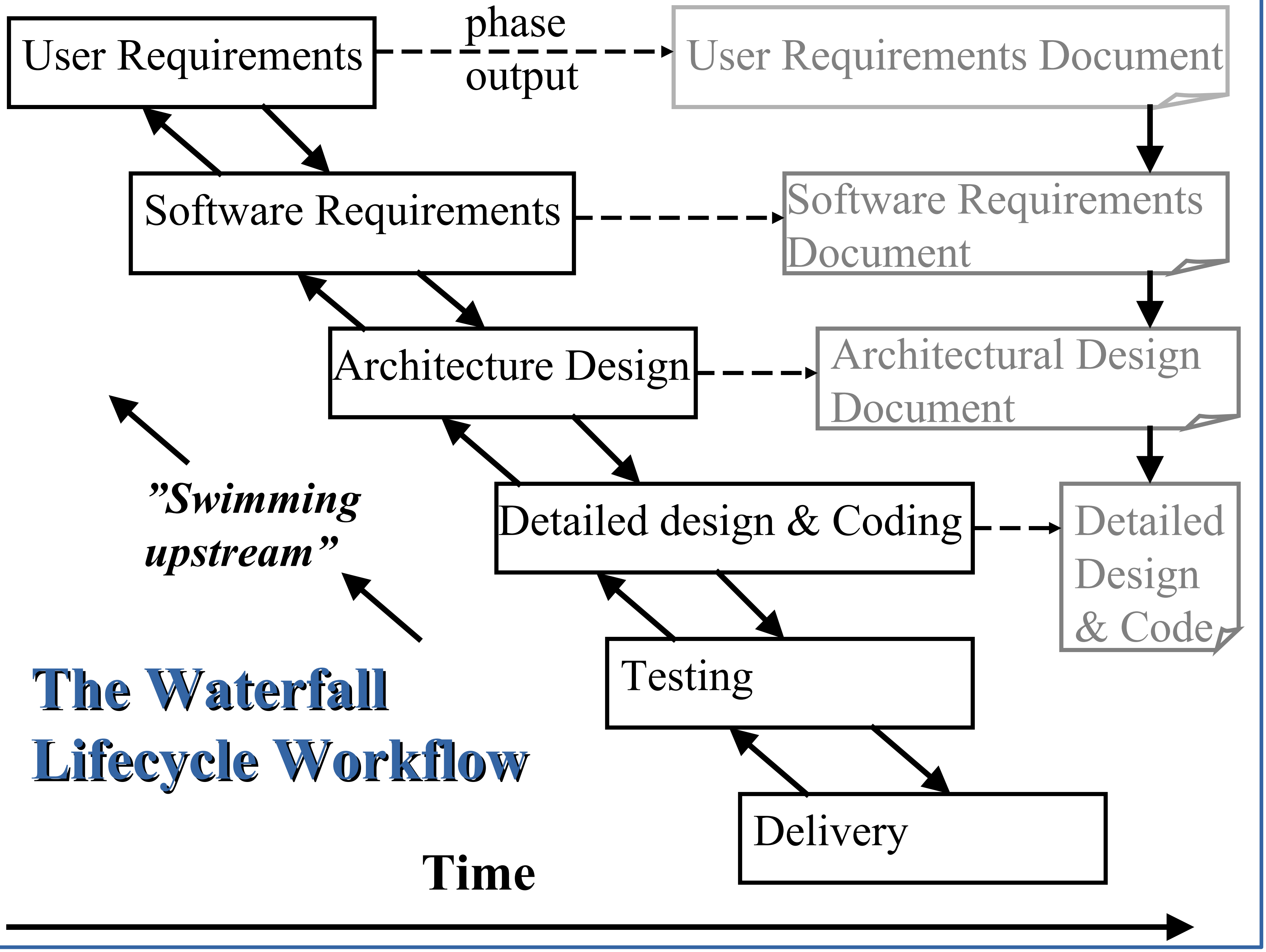
- Development **speed** (time to market)
- Product **quality**
- Project **visibility**
- **Risk** exposure
- Customer relations, etc.

Waterfall Model

- The waterfall model is the *classic* lifecycle model – it is widely known, understood and most commonly used.
- In an aspect, waterfall is the *”common sense” approach*.
- Introduced by Royce 1970.



The Waterfall Lifecycle Workflow



User Requirements

phase output

User Requirements Document

Software Requirements

Software Requirements Document

Architecture Design

Architectural Design Document

Detailed design & Coding

Detailed Design & Code

Testing

Delivery

"Swimming upstream"

Time

Advantages

1. Easy to understand and implement.
2. Widely used and known
3. Reinforces good habits: define-before-design, design-before-code
4. Identifies deliverables and milestones
5. Document driven, URD, SRD, … etc. Published documentation standards, e.g. PSS-05.
6. Works well on mature products and weak teams.

Disadvantages I

1. Idealised, doesn’t match reality well.
2. Doesn’t reflect iterative nature of exploratory development.
3. Unrealistic to expect accurate requirements so early in project
4. Software is delivered late in project, delays discovery of serious errors.

Disadvantages II

5. Difficult to integrate risk management
6. Difficult and expensive to make changes to documents, ”swimming upstream”.
7. Significant administrative overhead, costly for small teams and projects.

Spiral Model

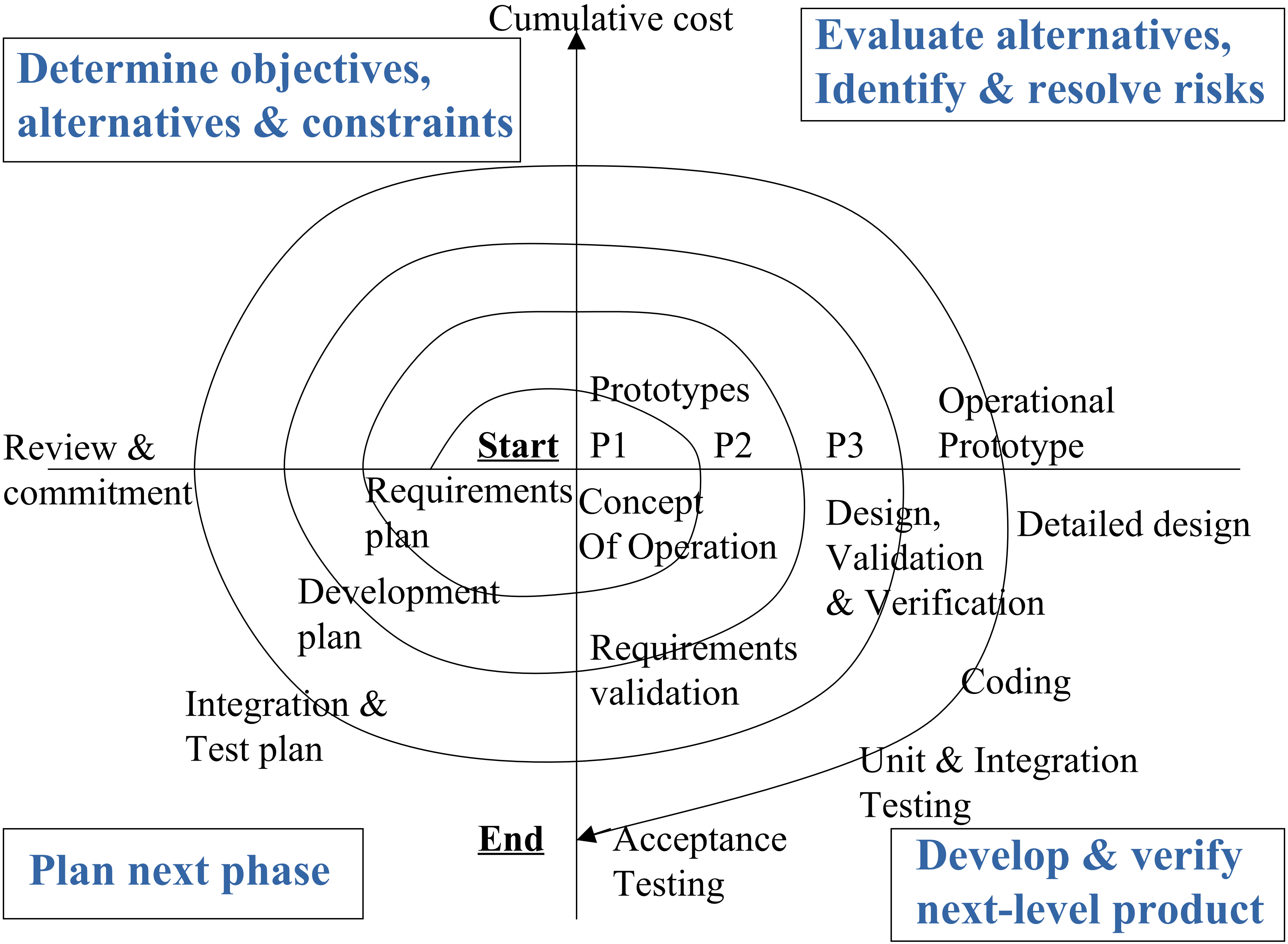
Since end-user requirements are hard to obtain/define, it is natural to develop software in an ***experimental*** way: e.g.

1. Build some software
2. See if it meets customer requirements
3. If not, go back to 1. Else, stop.

This loop approach gives rise to structured **iterative lifecycle models.**

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**.



Determine objectives, alternatives & constraints

Evaluate alternatives, Identify & resolve risks

Plan next phase

Develop & verify next-level product

Cumulative cost

Review & commitment

Start

Prototypes

P1

P2

P3

Operational Prototype

Requirements plan

Concept Of Operation

Design, Validation & Verification

Detailed design

Development plan

Requirements validation

Integration & Test plan

Coding

Unit & Integration Testing

End

Acceptance Testing

Each cycle follows a **waterfall model** by:

1. Determining objectives
2. Specifying constraints
3. Generating alternatives
4. Identifying risks
5. Resolving risks
6. Developing next-level product
7. Planning next cycle

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 waterfal and rapid prototyping methods
3. Comprehensive model decreases risk
4. Good project visibility.

Disadvantages

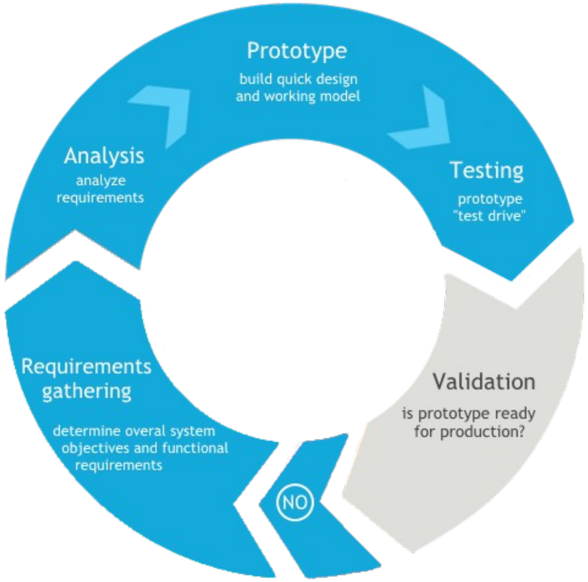
- Needs technical expertise in risk analysis to really work
- Model is poorly understood by non-technical management, hence not so widely used
- Complicated model, needs competent professional management. High administrative overhead.

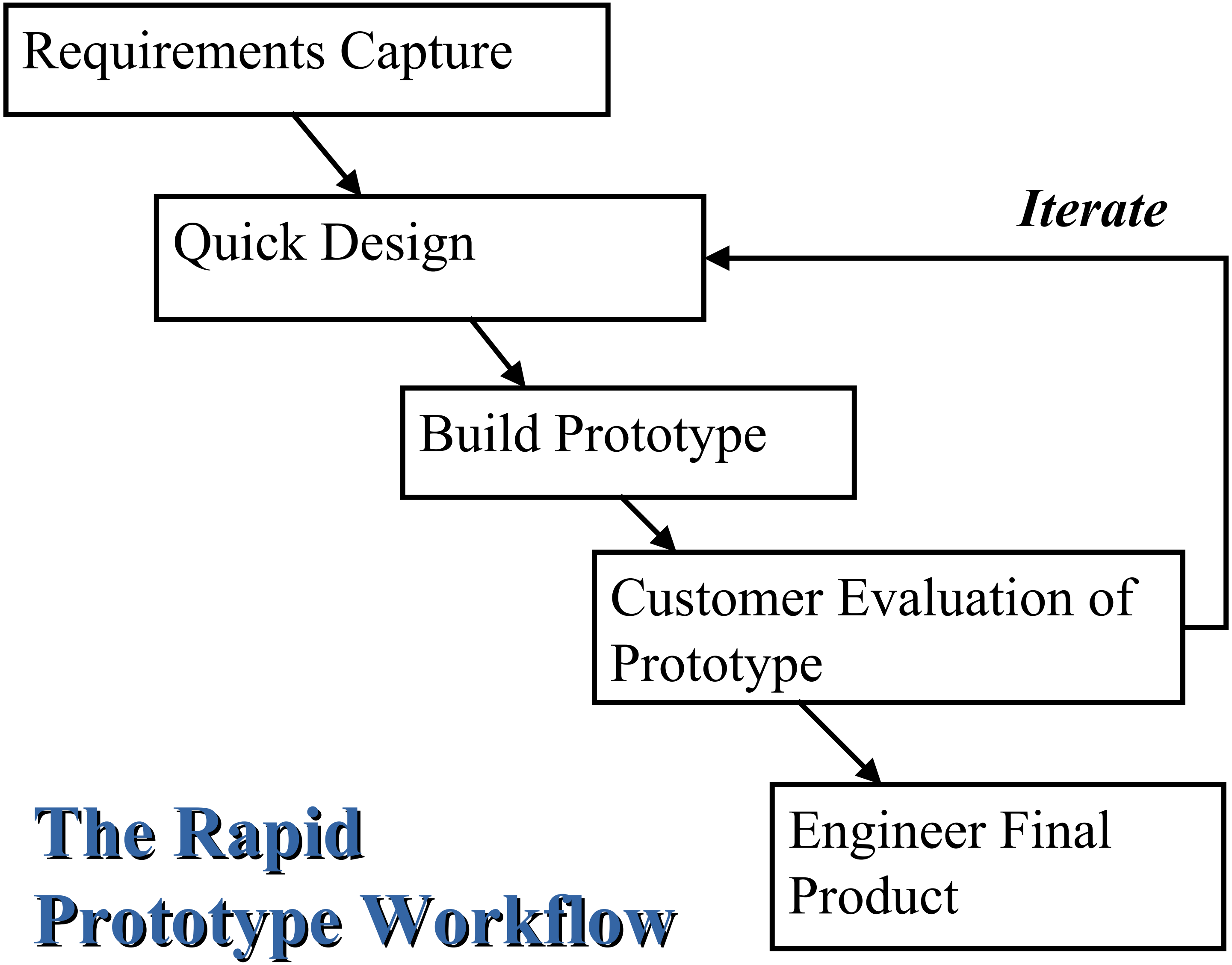
Rapid Prototyping

<u>Key idea</u>: Customers are non-technical and usually don't know what they want/can have.

Rapid prototyping <u>emphasises requirements analysis and validation</u>, also called:

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


Prototype
build quick design and working model
Testing
prototype "test drive"
Validation
is prototype ready for production?
NO
Requirements gathering
determine overal system objectives and functional requirements
Analysis
analyze requirements



Requirements Capture

Quick Design

Build Prototype

Customer Evaluation of Prototype

Iterate

Engineer Final Product

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.
2. Requires extensive customer collaboration
   - Costs customers **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.