SOFTWARE DEVELOPMENT PARADIGMS – CT 216 SOFTWARE ENGINEERING I

Dr. Enda Barrett

OÉ Gaillimh
NUI Galway

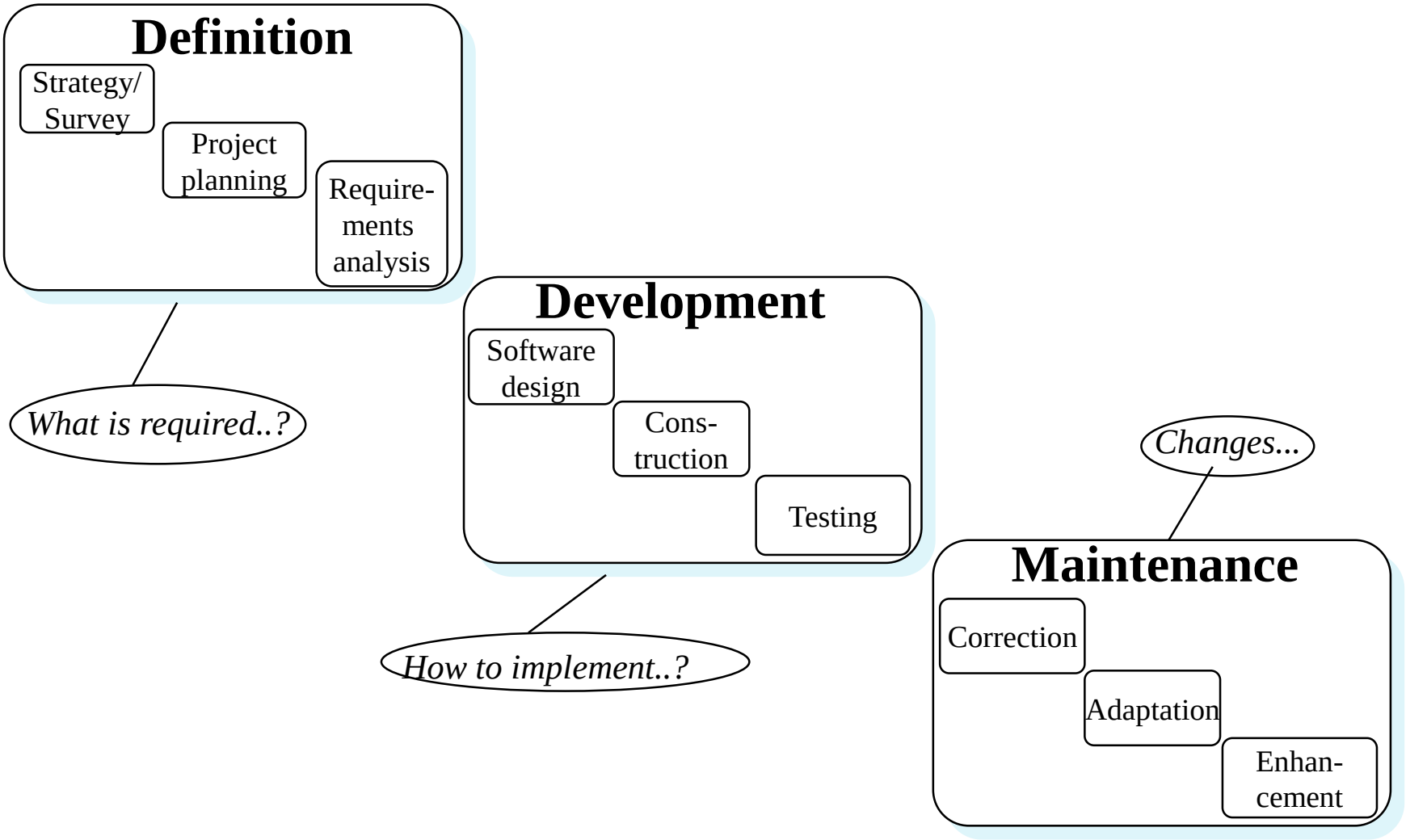
Learning objectives

- Paradigms of software development
  - Software life cycle
- A few flavours of the waterfall model
  - Traditional Waterfall
  - Waterfall with feedback
  - Waterfall with feedback and overlaps
- Prototyping

Paradigms of software development

- Most software process models are of the following form
  - The waterfall approach
    - Separate process phases, requirements, design, implementation…
  - Iterative/evolutionary development
    - Initial system developed, then customer redefines and iteratively the final system is built
  - Component Based Software Engineering (CBSE)
    - Parts of the system already built are integrated

SDLC


Definition
Strategy/ Survey
Project planning
Require-ments analysis
What is required..?
Development
Software design
Cons-truction
Testing
How to implement..?
Changes...
Maintenance
Correction
Adaptation
Enhan-cement

Software Development Lifecycle

- The software lifecycle is an abstract representation of a software process. It defines the steps, methods, tools, activities and deliverables of a software development projects. The following **lifecycle phases** are considered:
  - 1. requirements analysis
  - 2. system design
  - 3. implementation
  - 4. integration and deployment
  - 5. operation and maintenance

1. Requirements Analysis

- User requirements are statements in natural language plus diagrams of what services the system is expected to provide and the constraints under which it must operate (Sommerville 2001)
- Activity of determining and specifying requirements
  - Interview users, try to make communication clear
  - Questionnaires to users
  - Observations of users performing their tasks
  - Study existing system documents
  - Study similar software systems to learn about domain knowledge
  - Prototypes to confirm requirements
- Output is a **requirements document**

2. System Design

- A 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 the system components and the algorithms used (Sommerville 2001)
  - Algorithms are not always concretely defined in this phase as it is necessary to give some freedom to the developers.
  - Design begins where analysis ends.
  - Theoretically : Analysis is modelling unconstrained by any hardware/software considerations
  - Design is modelling that takes into consideration the platform upon which it is to be deployed.

- The output of this phase should be a **design document**

3. Implementation

- Implementation is mostly, programming, testing
  - A programmer is a "*component engineer*"
  - The programmer will attempt to re-use code where ever they can. However, it can often require quite a bit of modification to suit the specific needs.
  - Expand upon the design, i.e. algorithms will only be partially specified, engineers will have to thrash out the specifics of the design.
  - Turn this design into code
  - Debugging
  - Testing
- Output is **code**

4. Integration and Deployment

- **Integration** assembles the application from the set of components previously implemented and tested.
- ***Deployment*** is the handing over of the system to the customer for production use.
- Integration can be sometimes difficult to disassociate from the implementation phase nowadays, especially with continuous integration tooling so readily used. But still a stage in it own right.
- Software is deployed in releases i.e. version 1.0, 1.1 …
- Deployment releases
  - Alpha
  - Beta

5. Operation and Maintenance

- The new software product is used in day-to-day operations while the previous system is phased out.
  - Often systems will be run in parallel during the phase out
- **Maintenance** (Maciaszek)
  - *Corrective* – fixing defects and errors discovered in operation (*patches*)
  - *Adaptive* – modifying the software in response to changes in the computing and business environment (*new release 7.6 to 7.7*)
  - *Perfective* – evolving the product by adding new features (depending on the features but maybe a whole new release *i.e. go from 7.6 to 8.0*)
- Finally the system only becomes a ***legacy system*** and are only retired when it is not technically possible to support them anymore
  - Enterprises will flog them till their death

Waterfall model

- When these phases are all linearly placed, we get the traditional waterfall model
- Came out in the 70's as a solution monolithic type business applications (Cobol), fixed number of subroutines
- The completion of each phase results in some deliverable.
- Today it is used much less frequently as trying to maintain strictly within the phases is difficult
- May take a long time to see the final product

Traditional Waterfall model

User

Requirements Analysis

System Design

Implementation

Integration and Deployment

Product release

Operation and Maintenance

Question

- Which of following statements about the Waterfall Model is **false**

  - A) Documentation is produced at each phase
  - B) Inflexible with changing requirements
  - C) Phases are linearly placed
  - D) Customers are involved in all stages of the process

https://pollev.com/endabarrett504

Waterfall Model

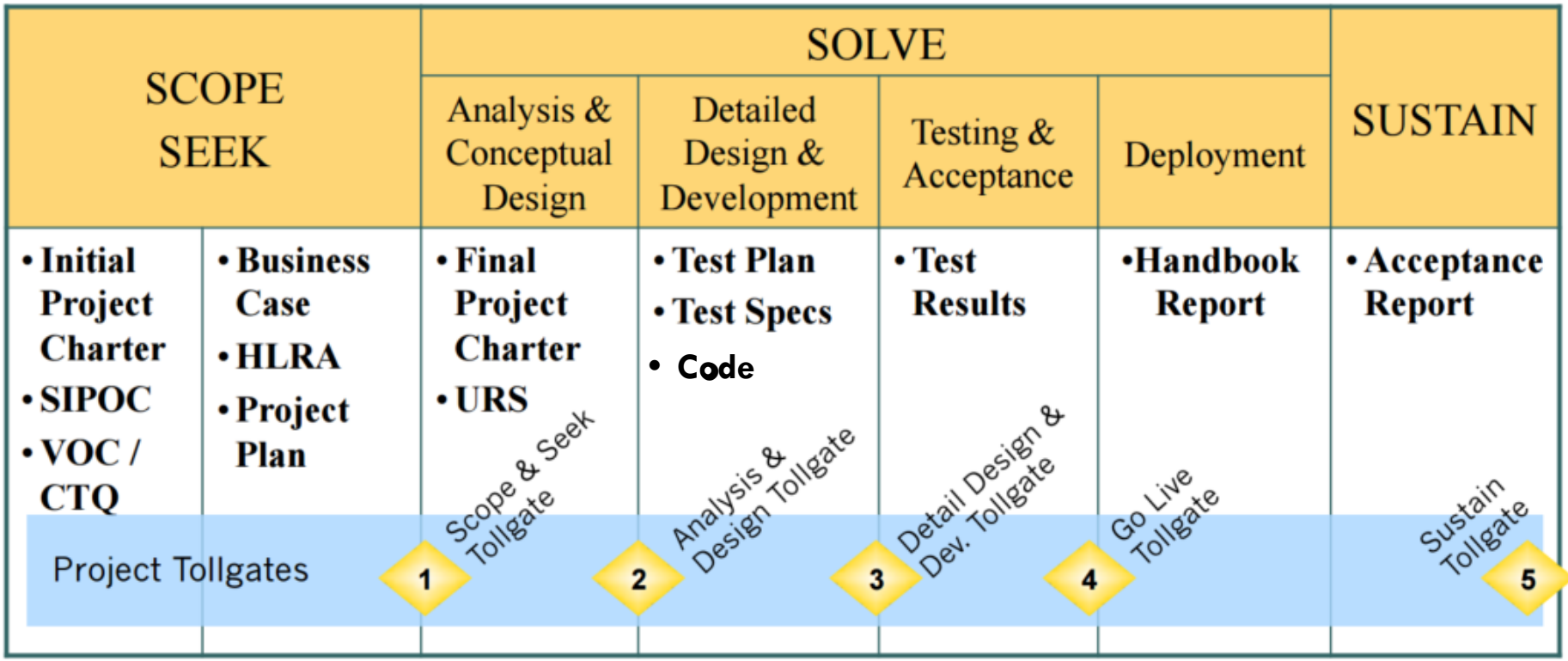
- Advantages
  - Documentation is produced at each phase
  - It fits in with other engineering process models
  - Easier to project manage
  - Sign-off phase clarifies legal position

- Disadvantages
  - Inflexible partitioning of the project into distinct phases
  - Difficult to adjust to changes in requirements
  - Delivered project may need re-work
  - Documentation can give false sense of the progress

Waterfall model

- An organisation may elect to use a particular lifecycle model to develop its software.
- However specifics of the lifecycle model i.e. how the work is done varies from org to org, from project to project.
- Remember a software product is not manufactured, it's "developed"
  - Every time you repeat the process you could easily get different results
  - This is why there are still so many engineering jobs and will continue to be for the foreseeable future ☺

Sample: Software Development Process



SIPOC – Suppliers, Inputs, Process, Outputs and Customers
VOC – Voice of Customer
CTQ – Critical to Quality
HLRA – High Level Risk Assessment
URS – User Requirements Specification

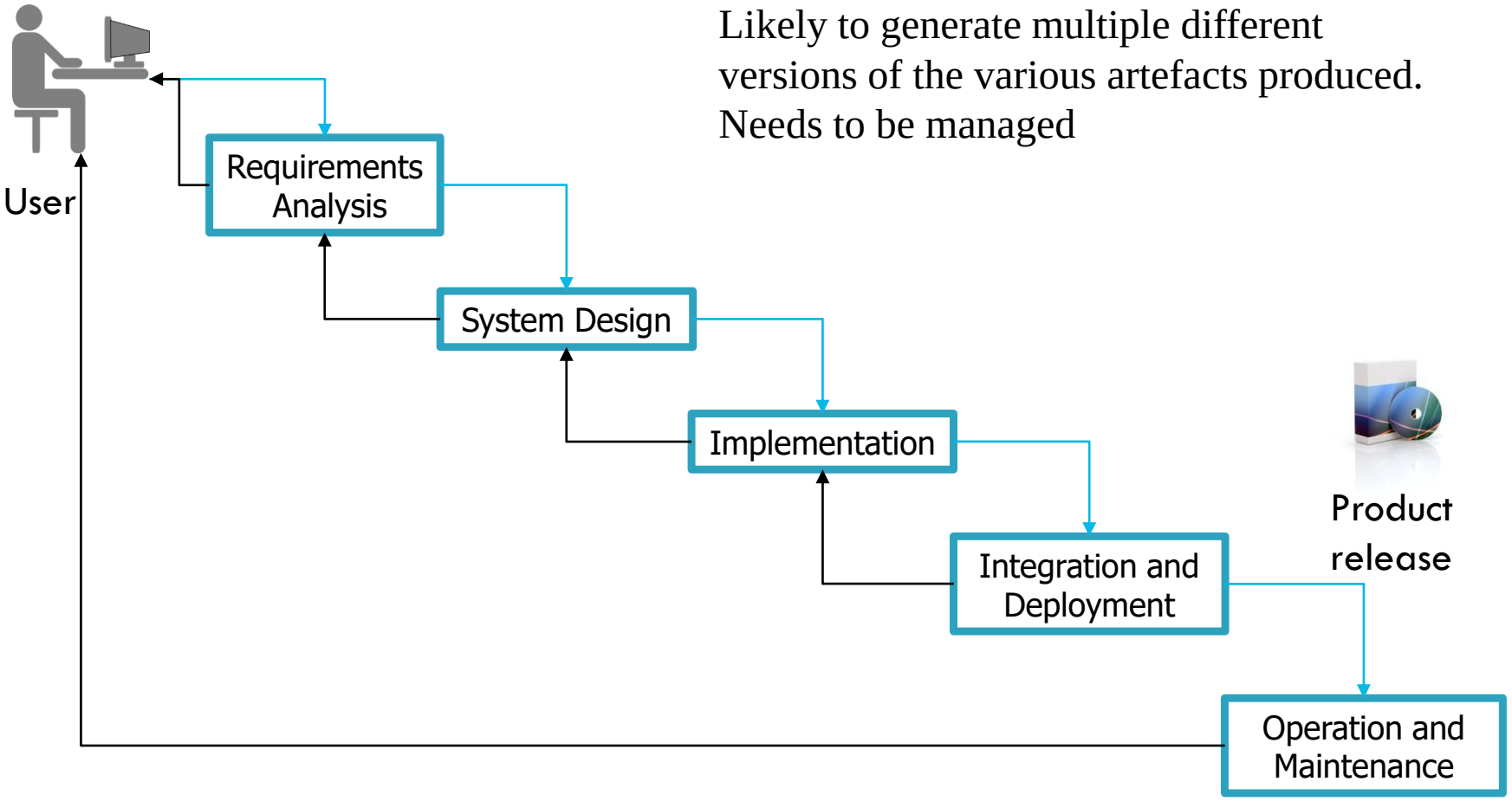
Waterfall Model with feedback

- Disadvantages
  - **Inflexible partitioning of the project into distinct phases**
  - **Difficult to adjust to changes in requirements**
  - Delivered project may need re-work
  - Documentation can give false sense of the progress

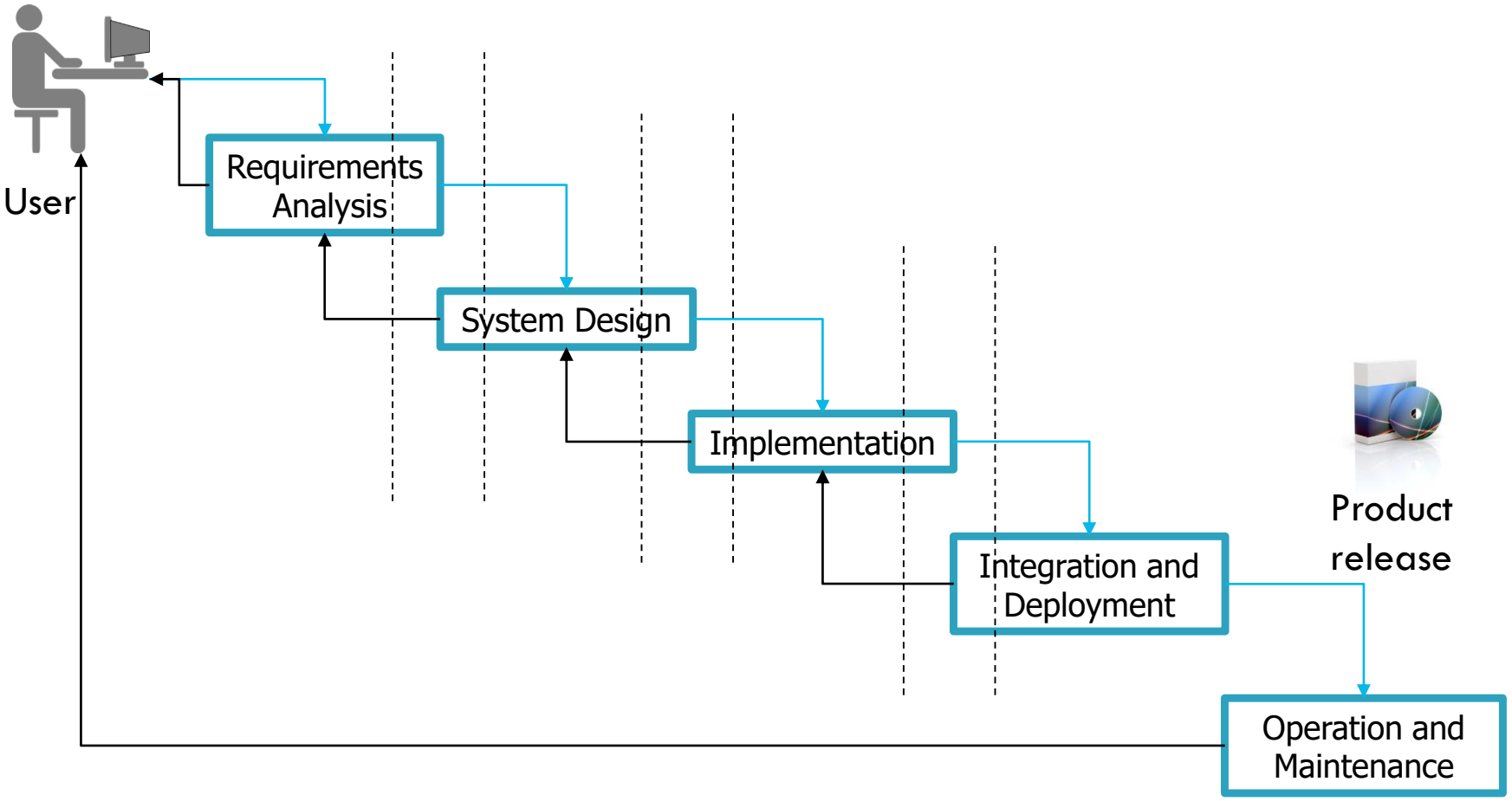
- Simplest way of addressing these two disadvantages is to introduce feedback paths to the development process

Waterfall model with feedback

Likely to generate multiple different versions of the various artefacts produced. Needs to be managed



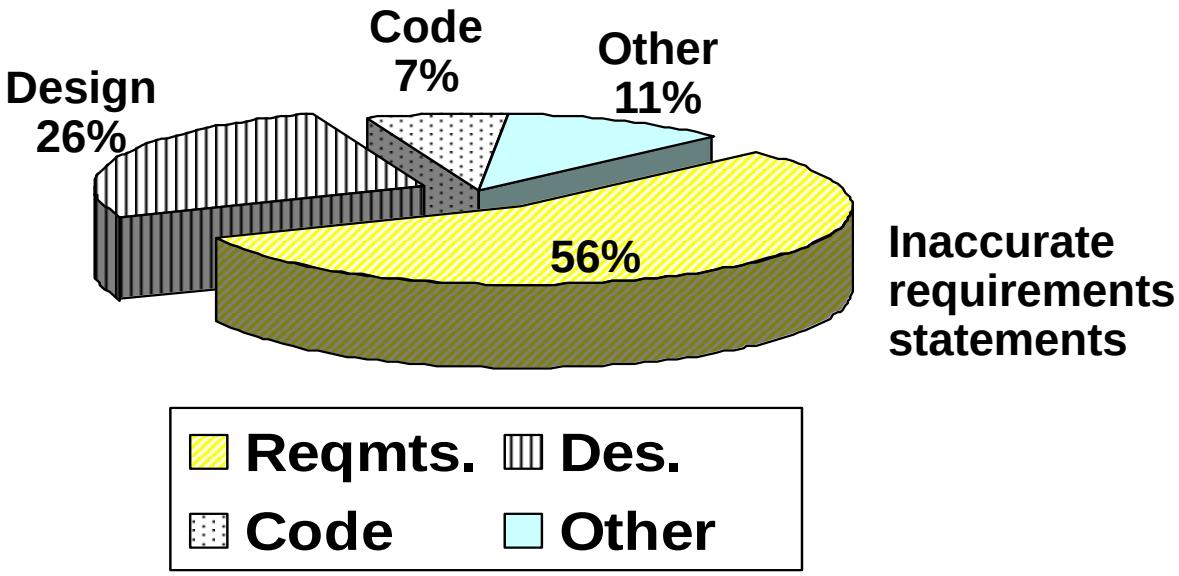
Waterfall model with feedback and overlaps



Deliverables from the SDLC

Phase	Deliverables
Requirements Analysis	Requirements specification Functional specification Acceptance test specification
System Design	Software architecture specification System test specification Design specification Sub-system test specification Unit test specification
Implementation & Testing	Program code Unit test report Sub-system test report System test report Acceptance test report Completed system
Integration & Deployment	Installed system
Operation & Maintenance	Change requests Change request report

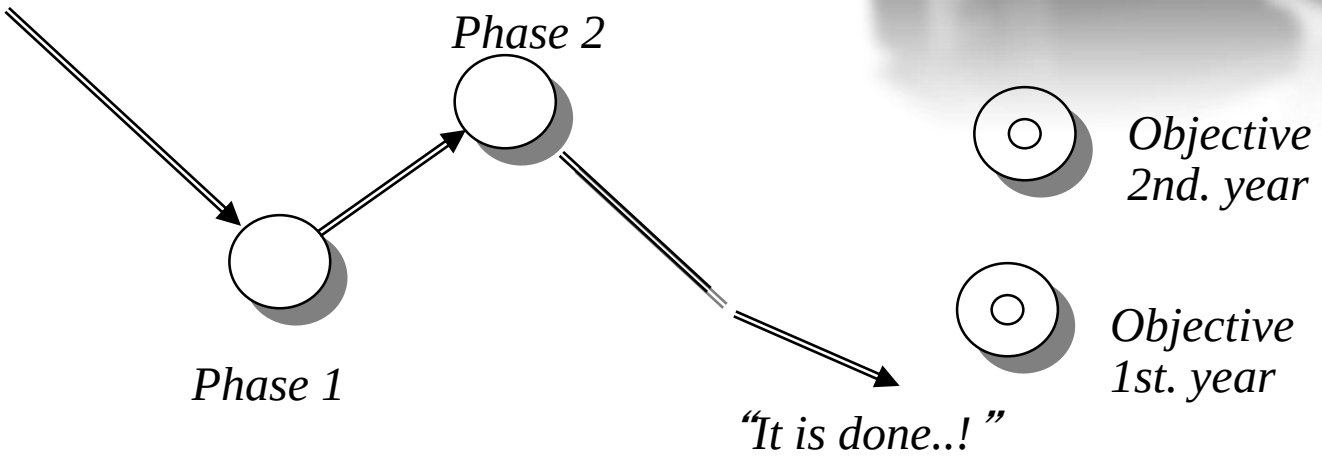
Sources of Error in the SDLC



Limitations of the SDLC

"The water isn't flowing....":

- Rarely delivers what is wanted
- Specialists needed for development
- Difficult to execute -- e.g. freezing specs.
- It takes too long
- Too much documentation



Prototyping

An alternative approach to requirements definition is to capture an initial set of needs and to implement quickly those needs with the stated intent of iteratively expanding and refining them as mutual user/developer understanding of the system grows.

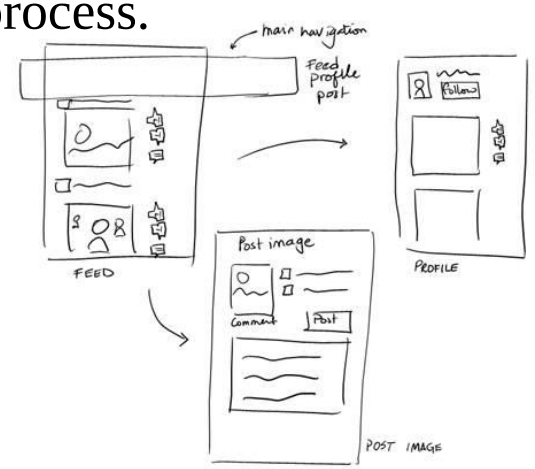
Definition of the system grows through gradual and evolutionary discovery as opposed to omniscient foresight.

How it works

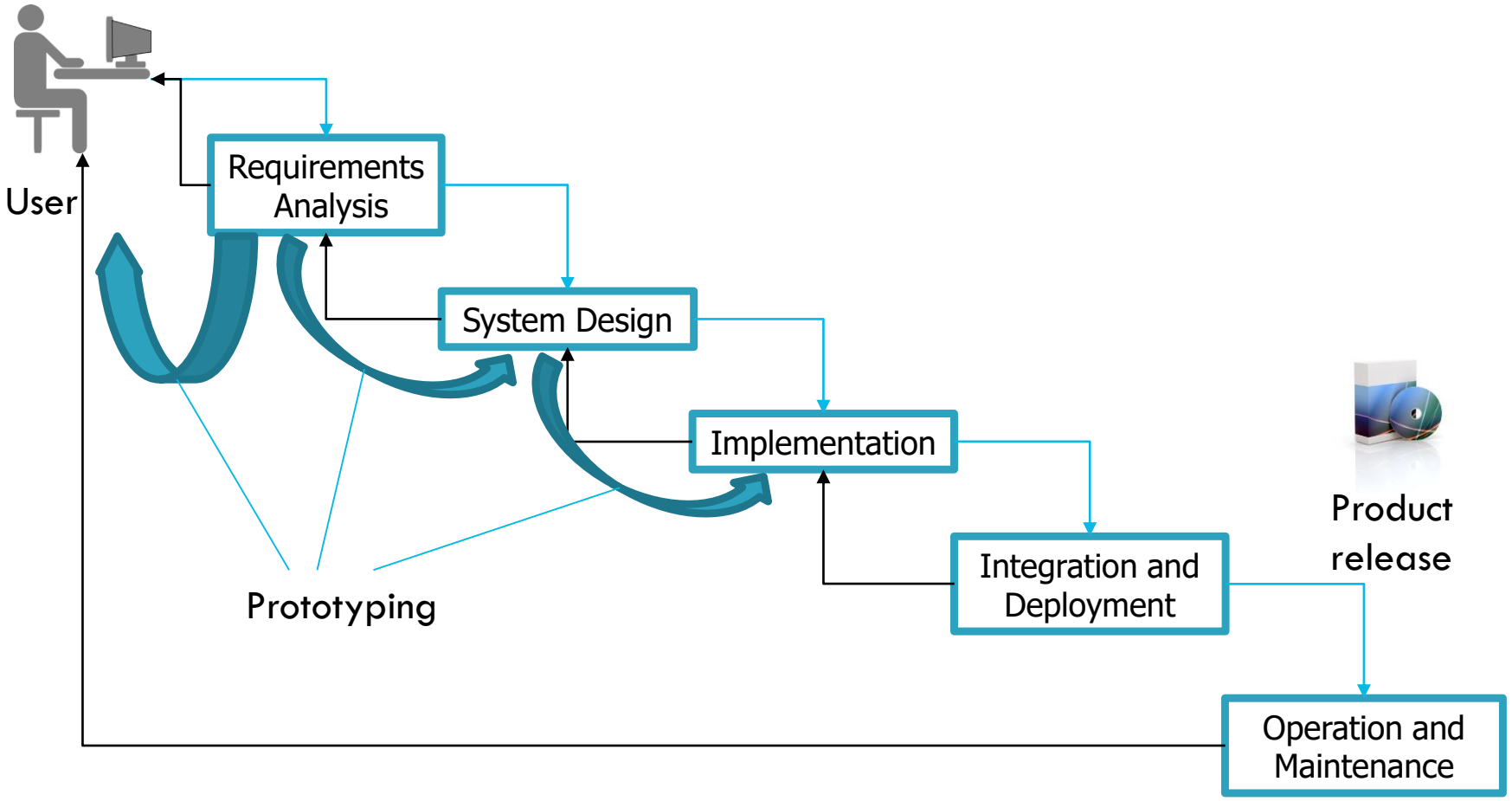
This approach assumes that the model of the system will be a working model, that is, a collection of computer programs that will simulate some or all of the functions that the user wants.
Mitigate against the risk of delivering something that the customer does not want by offering a “quick and dirty” solution.

With *Rapid Prototyping* these programs will be thrown away and eventually replaced by the REAL programs. – **My old job**

With *Evolutionary Prototyping* a subset of a system is fully developed and then the rest of the system is developed using the same development process.



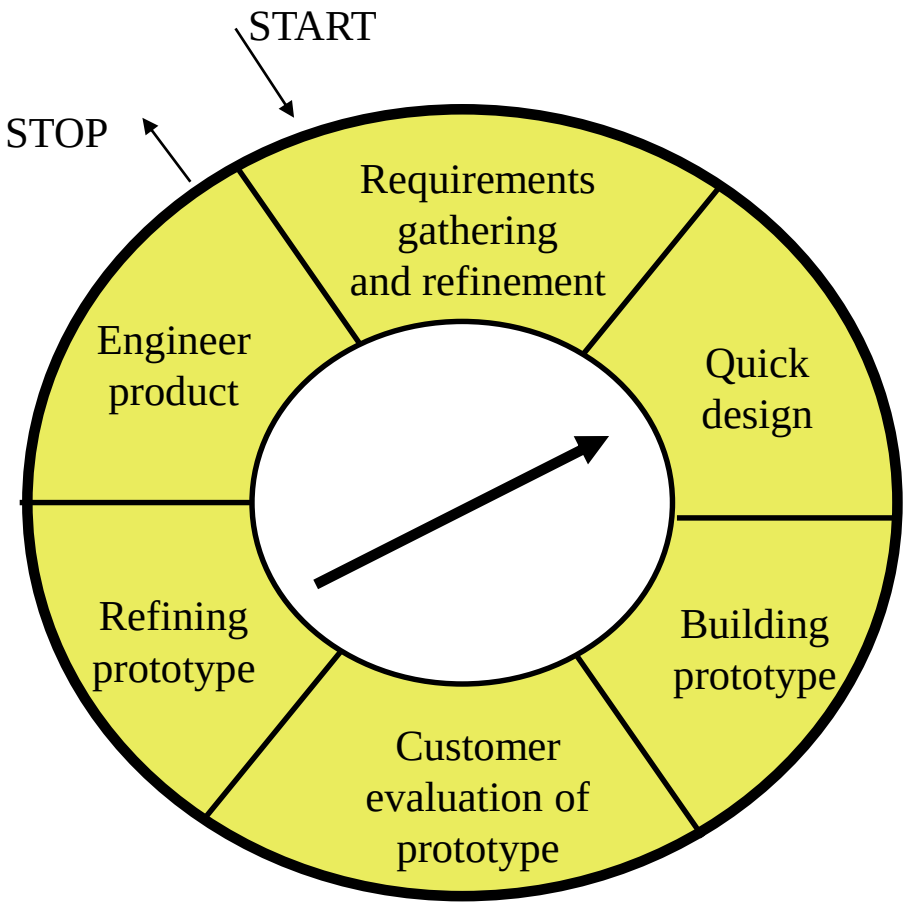
Waterfall model with prototyping



Prototyping per phase

- Requirements
  - Prototype can help with the elicitation and validation of system requirements
- System design
  - Explore particular design solutions and **support UI design**
- System testing
  - Run initial testing on the system that will be delivered to the customer

Prototyping approach



Summary

- Paradigms of software development
  - Software life cycle
- Waterfall model
  - Traditional Waterfall
  - Waterfall with feedback
  - Waterfall with feedback and overlaps
- Prototyping