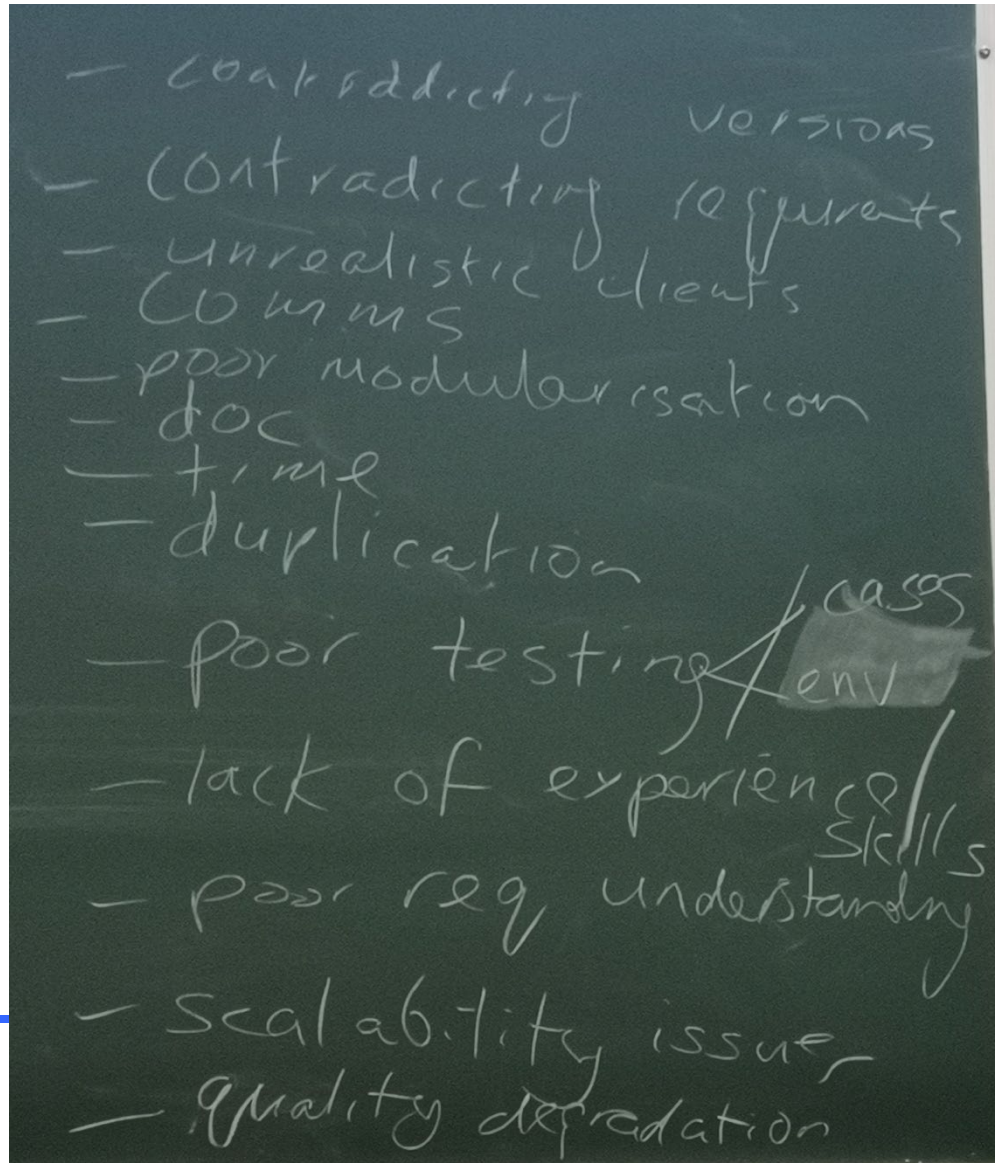
University of Dublin
Trinity College

# CS7CS3: Software Lifecycles

Prof. Siobhán Clarke *(she/her)*
Ext. 2224 – L2.15
www.scss.tcd.ie/Siobhan.Clarke/

# So, remember we pondered...

Whether software engineering is hard?



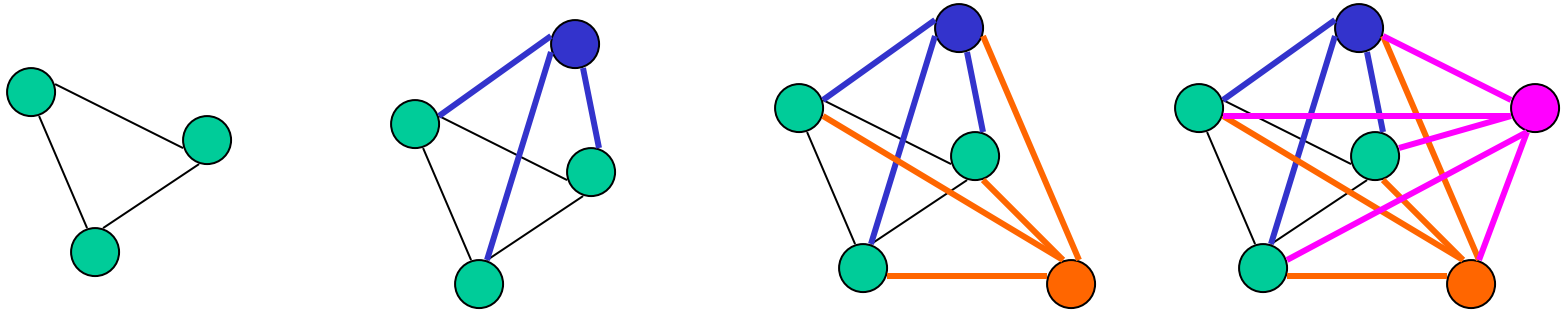
# So can we agree?

Software engineering is hard

i.e., for large systems, with good quality, that is easily maintainable and extensible over time

# The first basic problem

- …is one of communication – both personal and in the software



3 agents
3 channels

4 agents
6 channels

5 agents
10 channels

6 agents
15 channels

A system “twice as big” is actually much more than twice as complex

How do we master this complexity as systems grow?

# The second basic problem

- …concerns how systems evolve and goalposts move
- A system which doesn't change isn't being used
  - New platforms, new peripherals, new modes of working
  - The web changed every application – and the companies that didn't notice watched their businesses die
- But users want a certain amount of stability
  - Investment in training and support – may be more than in software
  - Resist moving to Linux because of the re-training – even if it offers a better technical solution to their problem
- How do we balance this trade-off as systems grow?

# Engineering systems

"The application of science to the design, building and use of machines, constructions etc"

Oxford Reference English dictionary (1998)

- Implications
  - Repeatable – should be able to re-use tools and techniques across projects
  - Responsible – should take account of best practices and ethics
  - Systematic – should be planned, documented and analysed
  - Measurable – should have objective support both for the products *and* for the process itself

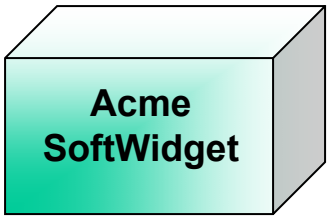
# Reminder of what's involved

## Requirements
- Functional requirements
- Non-functional requirements

## Specification
- What it must do
- What it mustn't do

## Design
- Architecture
- Functional components
- Algorithms and data structures

## Maintenance
- Bug fixes
- New features
- New platforms
- Versions




## Coding
- Individual components
- Synergy

## Testing
- Black-box testing
- White-box testing
- Acceptance testing

## Deployment
- Acceptance
- Packaging
- Marketing

## Documentation
- Requirements and specification
- Design decisions
- User documentation

## Integration
- OS variations
- Interworking
- Communication

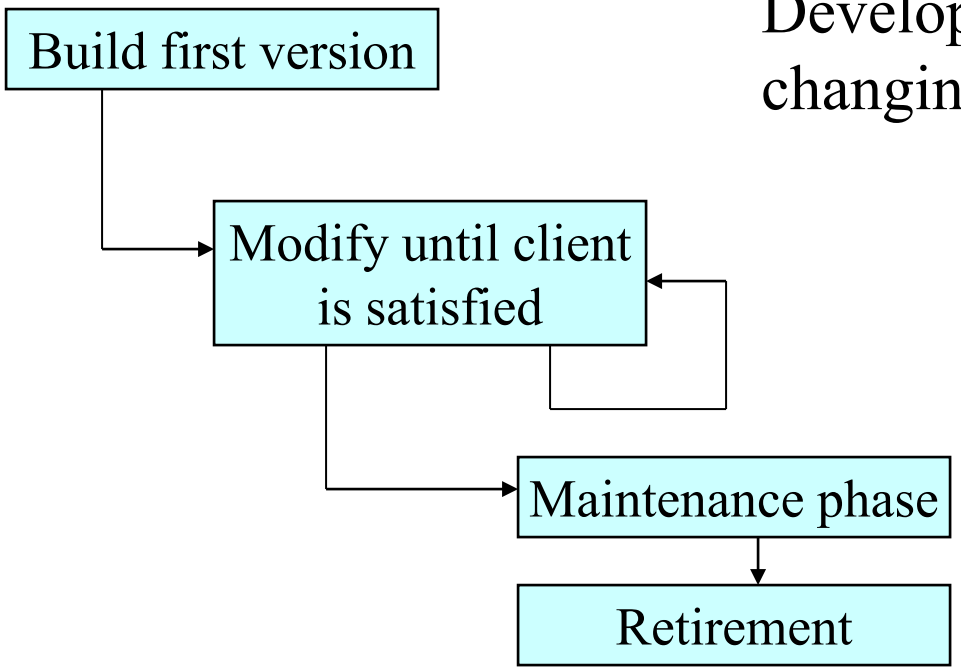
# When do we do each bit?

Series of steps through which product progresses is called the *life-cycle model*

Different kinds suit different situations – some examples:

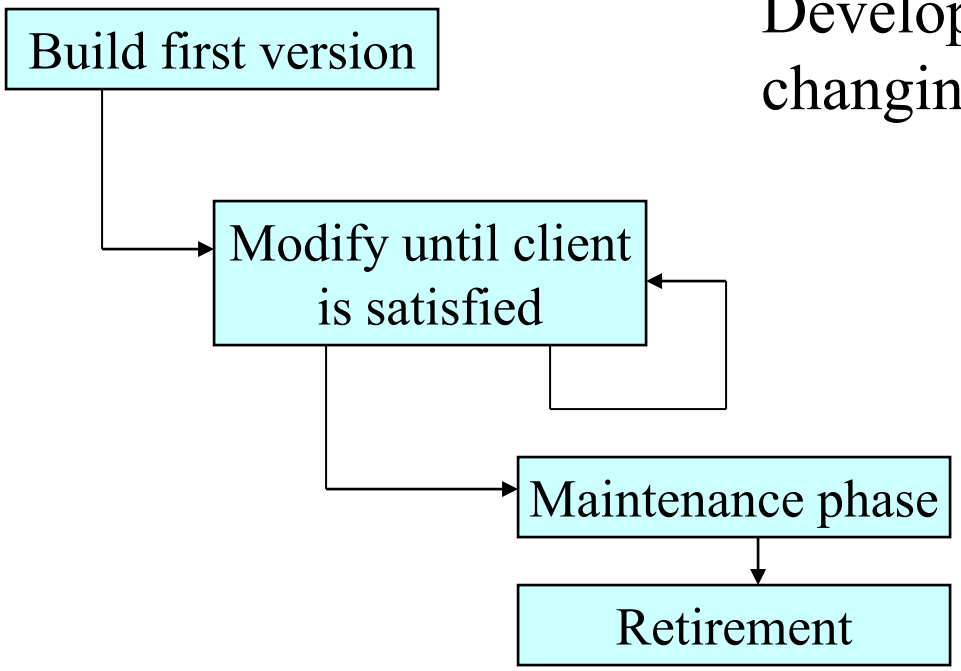
- Build-and-fix model
- Waterfall model
- Rapid prototyping model
- Incremental model
- eXtreme programming model

# Build-and-fix




Developer(s) just build it and keep changing it until it's "right"

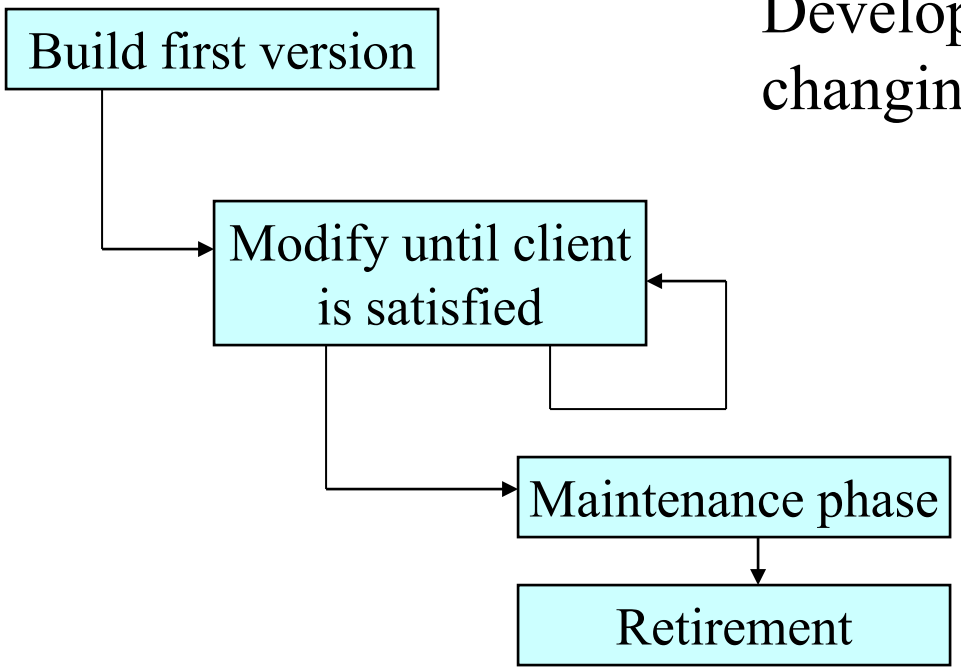
# Build-and-fix




Developer(s) just build it and keep changing it until it's "right"

What problems might arise?

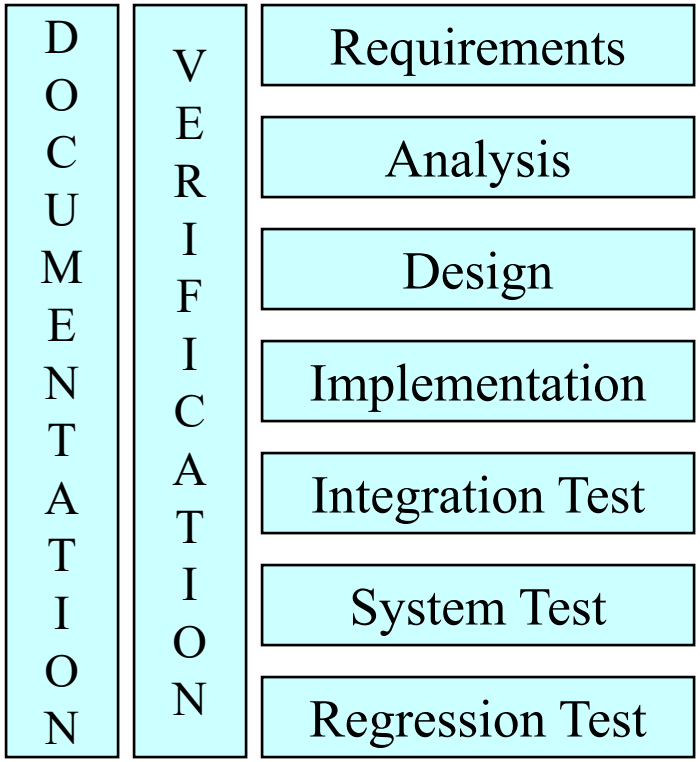
# Build-and-fix




Developer(s) just build it and keep changing it until it's "right"

**Problem:** Totally unsatisfactory for any project of reasonable size. Cost of change higher with code. Maintenance v. difficult with no design doc.

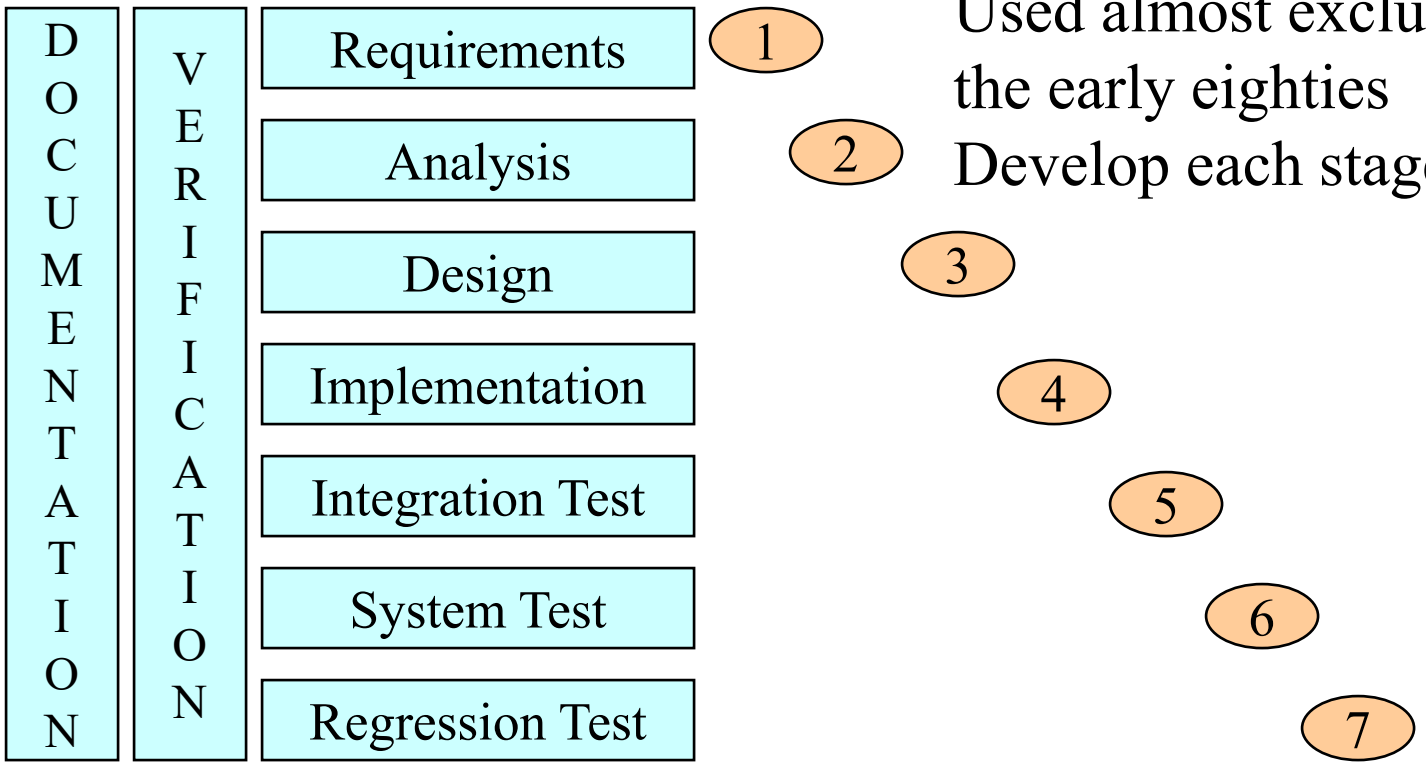
# Waterfall




Used almost exclusively up to the early eighties

Develop each stage in sequence

# Waterfall

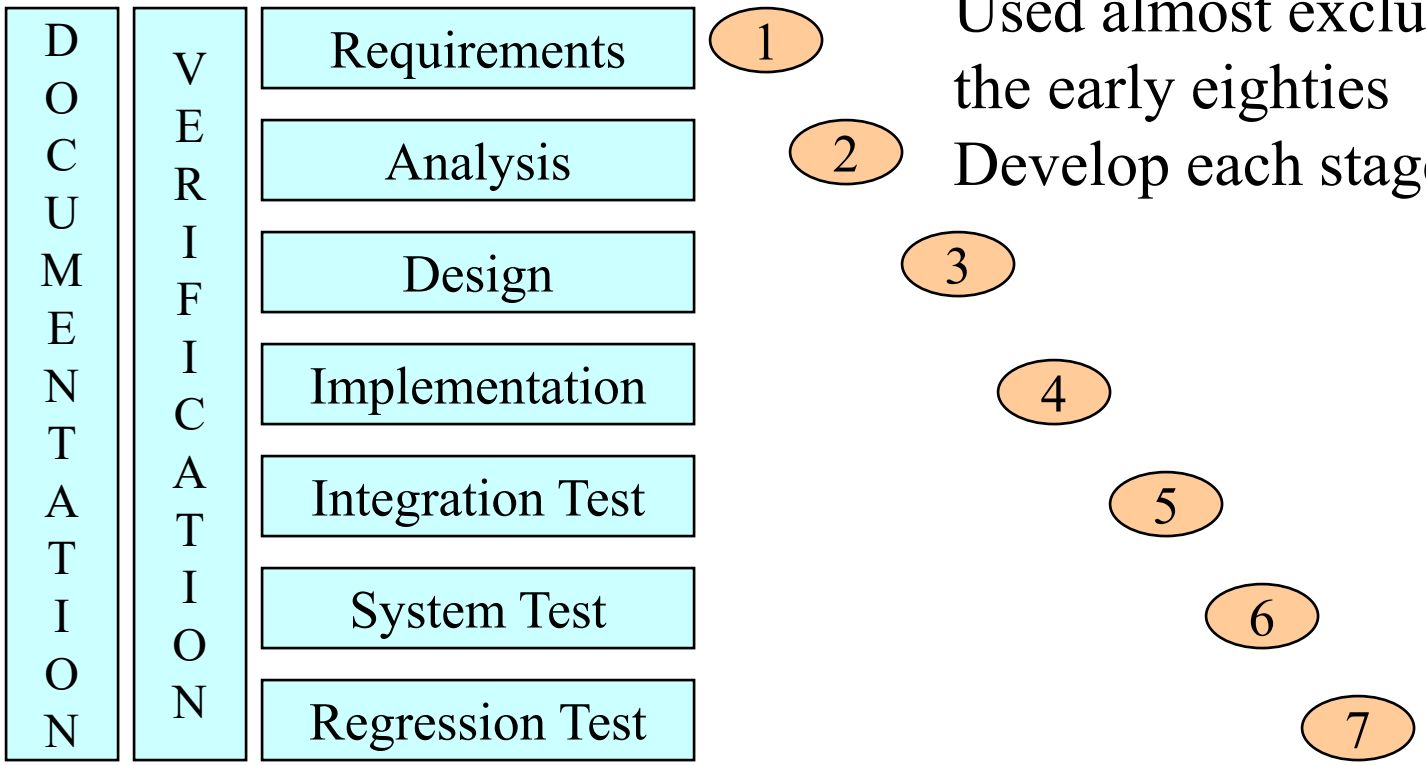



Used almost exclusively up to the early eighties

Develop each stage in sequence

What problems might arise?

# Waterfall

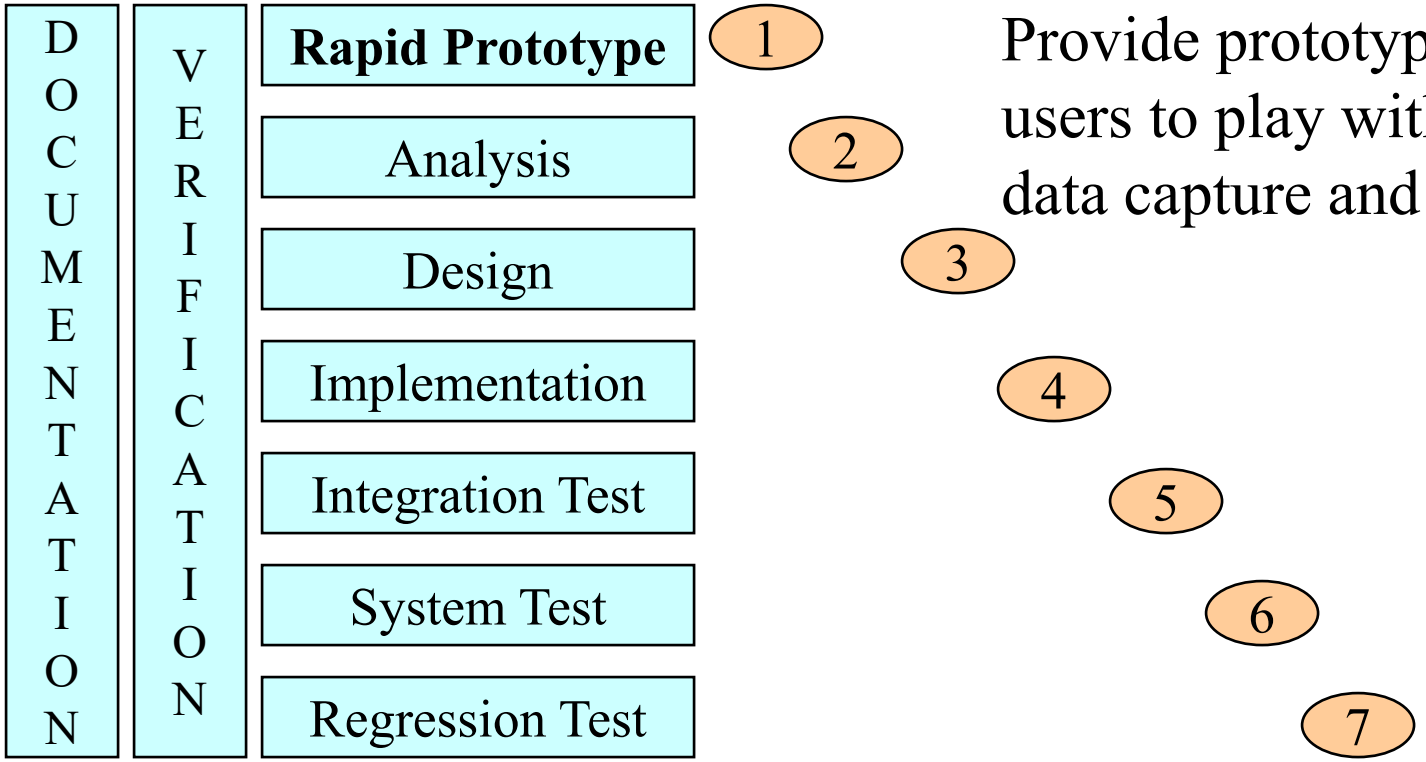



Used almost exclusively up to the early eighties

Develop each stage in sequence

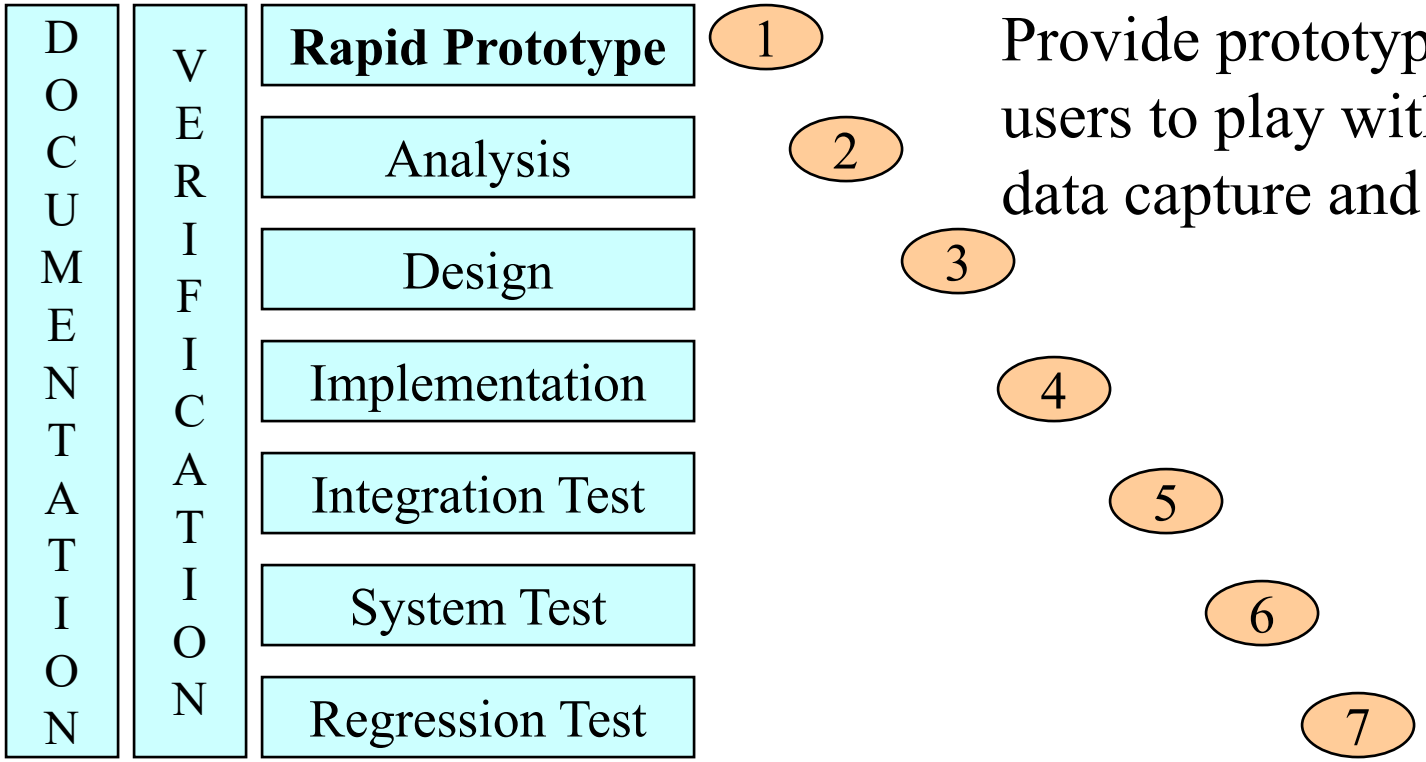
**Problem:** Issues/Difficulties/Defects are highlighted very late in the process, when they are more difficult/expensive to change

# Rapid Prototyping




Provide prototype of system to users to play with (e.g., UI for data capture and reports)

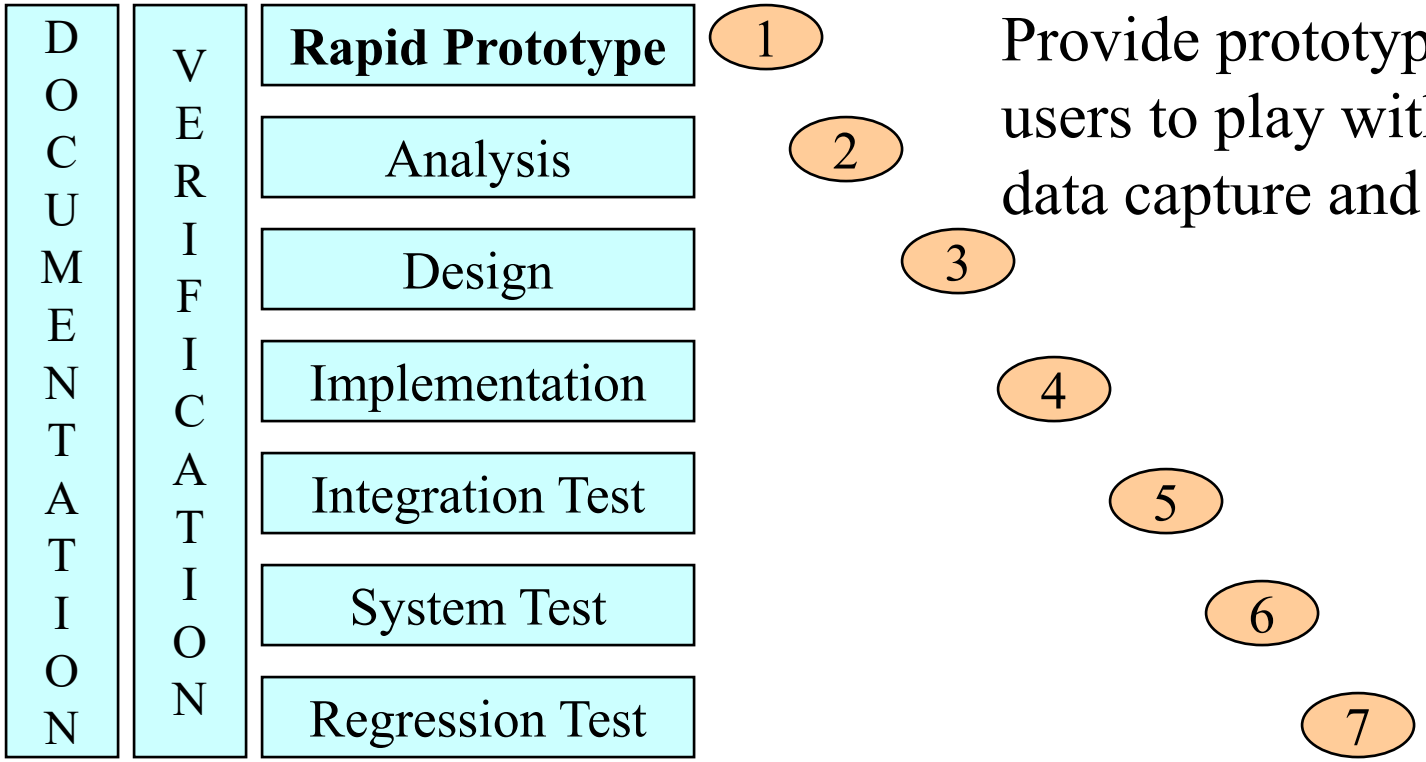
# Rapid Prototyping




Provide prototype of system to users to play with (e.g., UI for data capture and reports)

What problems might arise?

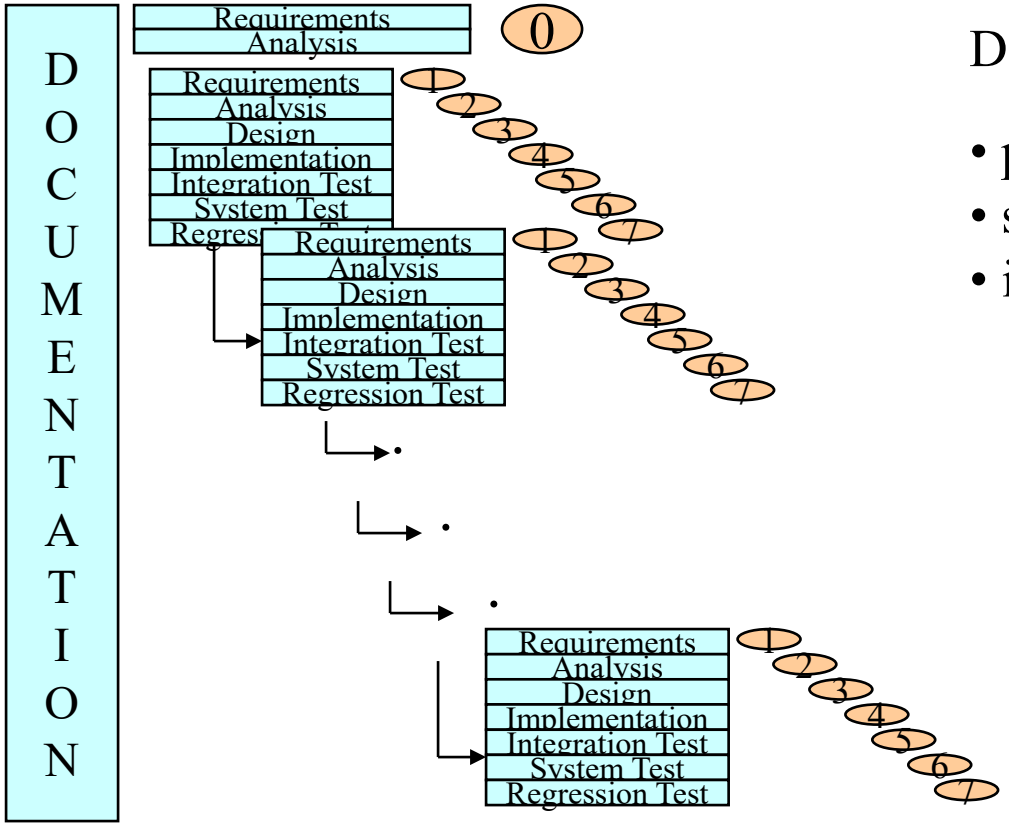
# Rapid Prototyping




Provide prototype of system to users to play with (e.g., UI for data capture and reports)

When used with the Waterfall model, can be more comfortable that you're providing the clients what they want… but they still won't get the real version until the "end".

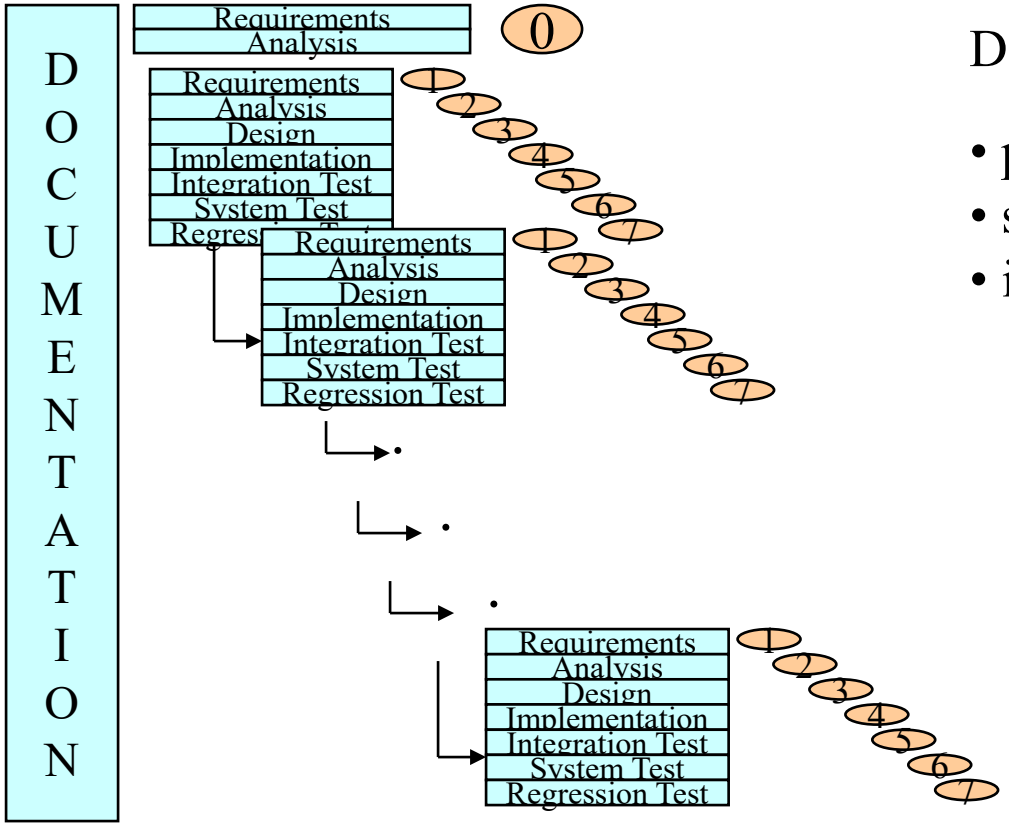
# Incremental




Develop in small steps:

- plan a little
- specify/design/implement a little
- integrate, test, and run each iteration

# Incremental



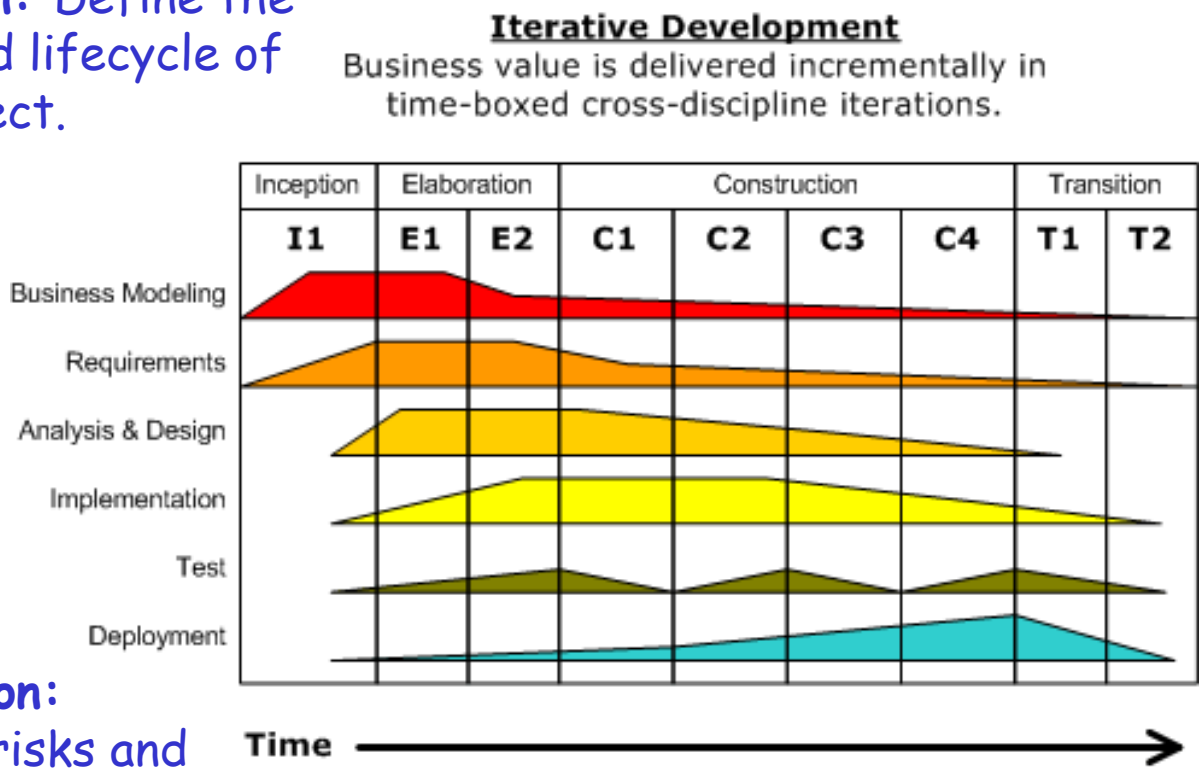

Develop in small steps:

- plan a little
- specify/design/implement a little
- integrate, test, and run each iteration

Issues/Difficulties/Defects are highlighted earlier in the process and solutions are fed in to the remaining iterations. Clients can use parts of system sooner.

# RUP Iterative Development

**Inception:** Define the scope and lifecycle of the project.




**Construction:** Develop the remainder of the system as efficiently as possible.

**Elaboration:** Mitigate risks and create a stable baseline architecture.

**Transition:** Train users to be self-sufficient; get customer acceptance of the product.

# Model-Driven Development

- Challenges in engineering software include multiplicity of technical platforms, product versions, ...

- Model driven software development is an approach that combines

  - Domain-specific modelling languages (DSMLs) that express structure, behaviour and requirements within particular domains. Abstracts complexities of concepts away from modeller.

    with

  - transformation engines & generators that analyse certain aspects of the models & then transform the DSMLs automatically to various software artifacts (e.g. source code, XML descriptors, alternative model representations, etc.). Hides technical complexities from programmer.

- Kind of orthogonal to “lifecycle”, but certainly an approach to developing software

# Agile Methods

- Lightweight development process

- Agile Manifesto, 2001
http://agilemanifesto.org/principles.html

- Methods include:
  - Agile Modeling
  - Agile Unified Process (AUP)
  - Dynamic Systems Development Method (DSDM)
  - Essential Unified Process (EssUP)
  - Extreme Programming (XP)
  - Feature Driven Development (FDD)
  - Kanban
  - Open Unified Process (OpenUP)
  - Scrum
  - Velocity tracking