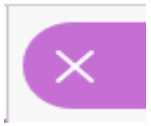
University of Dublin
Trinity College

CS7CS3: Software Lifecycles

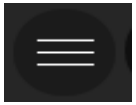
Prof. Siobhán Clarke
Ext. 2224 – L2.15
www.scss.tcd.ie/Siobhan.Clarke/

The Session Will Begin Shortly

1. Click on the pink Collaborate button (bottom right) to open the chat window and enter your message.



2. You can close the Collaborate panel at any time so you can see more of the current presentation.



3. Click on the menu icon at the top left when you want to exit the online lecture

→ A recording will be made available afterwards in case you get disconnected or have technical issues.

Student Online Teaching Advice Notice

The materials and content presented within this session are intended solely for use in a context of teaching and learning at Trinity.

Any session recorded for subsequent review is made available solely for the purpose of enhancing student learning.

Students should not edit or modify the recording in any way, nor disseminate it for use outside of a context of teaching and learning at Trinity.

Please be mindful of your physical environment and conscious of what may be captured by the device camera and microphone during videoconferencing calls.

Recorded materials will be handled in compliance with Trinity's statutory duties under the Universities Act, 1997 and in accordance with the University's policies and procedures.

Further information on data protection and best practice when using videoconferencing software is available at https://www.tcd.ie/info_compliance/data-protection/.

Questions?

So, remember we pondered...

Whether software engineering is hard?

No value for innovation. Focus only on implemntation.

Options up here! PLEASE USE TYPE NOT FREEHAND

Whiteboard

Shift responsibility

Unsure of how to estimate time for project

What are the factors that contributed to any **failures** in the projects you worked on?

lack of communication. Example tw
people picking up the same task.

too many meetings

demand issue

Delivery took priority over security

Requirement miss.

Tasks not described clearly enough

Miscommunication

Some people do not own responsibility.

inorrectly defined requirements

miscommunication

Lack of subject matter experts when the timelines were stringent

Work and time management

focusing on quantity of dev instead of quality

not having proper test cases

Disagreements between different sub-teams

poor documentation and compromising quality over deliverability

lack of following code practices

misunderstanding customer requirements

Time management

different culture

somebody went away

organizational

Lack of direction and information

lack of experience

politics

Lack of suitable and competent people for the job.

Lack of understanding of tech stack

Lack of support from the vendor

Ego management

Ego prevented proper delegation

Underestimation of loads

communication

ego clashes

Less number of testers

Under-estimation for the story

excessive documentation
work and less time to code

Not hiring the required expertise essential for the the project

people not contributing

Lack of concrete colloboration framework

Difference of opinions

infrequent standup meetings

nepotism

supportive peers and good team leader and an excellent manager

Options up here! PLEASE USE TYPE NOT FREEHAND

Whiteboard

Team members being honest on what they did/didn't know

weekly goal

Knowledge Sharing sessions with all the team members

enforcing correct code coverage

What are the factors that contributed to any **successes** in the projects you worked on?

Having close experience levels and areas in the team increases success in certain subjects

communication

Version control checks in place

A good manager

Comment your code

group discussions

agile methodology

Meaningful meetings

Good project Manager

peer reviews

Proper leave planning

clear milestones and frequent meetings

Proper Requirement Gathering

Team management

Taking the effort to understand the technology/systems from other teams

Granular specking of the task

your code will be interfacing with

excellent team leader and supportive peers

Time Management

Scoping the development into milestones.

having a

Daily 'Technical Office Hour'

defined set of requirements

Documentation on point

communication

Good in multitasking

Daily meetings

Good code style

continious testing of smaller modules

Communication

a good leader

sharing failure experience and the way to overcome it.

communication

teamwork

Communication

evryone achieved goals

Clear objectives/goals

Reasonable amount of team members

whiteboarding before coding.

Fear of failure

Proper Requirement Analysis

Working well under stress

deciding proper design pattern and abstractions

team willing to work overtime without complaints

Planning with the team and getting everyones feedback

Development is easier when you know business functionality

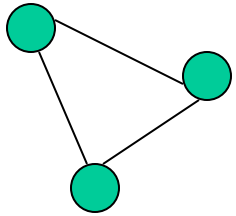
So can we agree?

Software engineering is hard

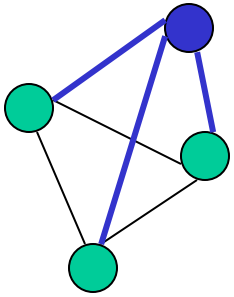
i.e., for large systems, with good quality, that is easily maintainable and extensible over time

The first basic problem

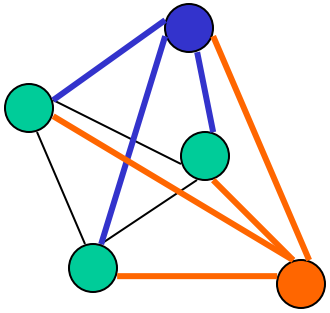
- …is one of communication – both personal and in the software



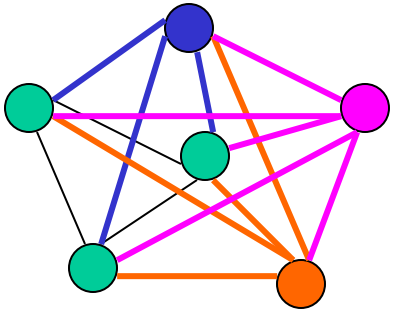
3 agents
3 channels



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

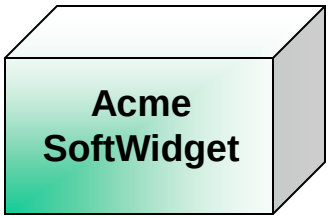
Reminder of what's involved

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

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

Deployment
- Acceptance
- Packaging
- Marketing

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

Documentation
- Requirements and specification
- Design decisions
- User documentation

Integration
- OS variations
- Interworking
- Communication

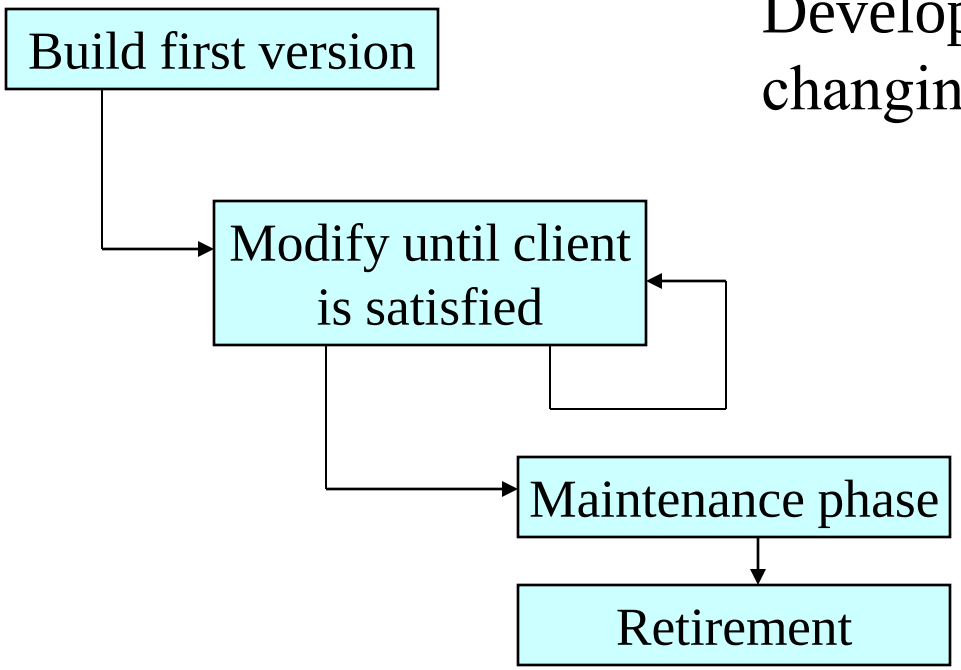
When do we do each bit?

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

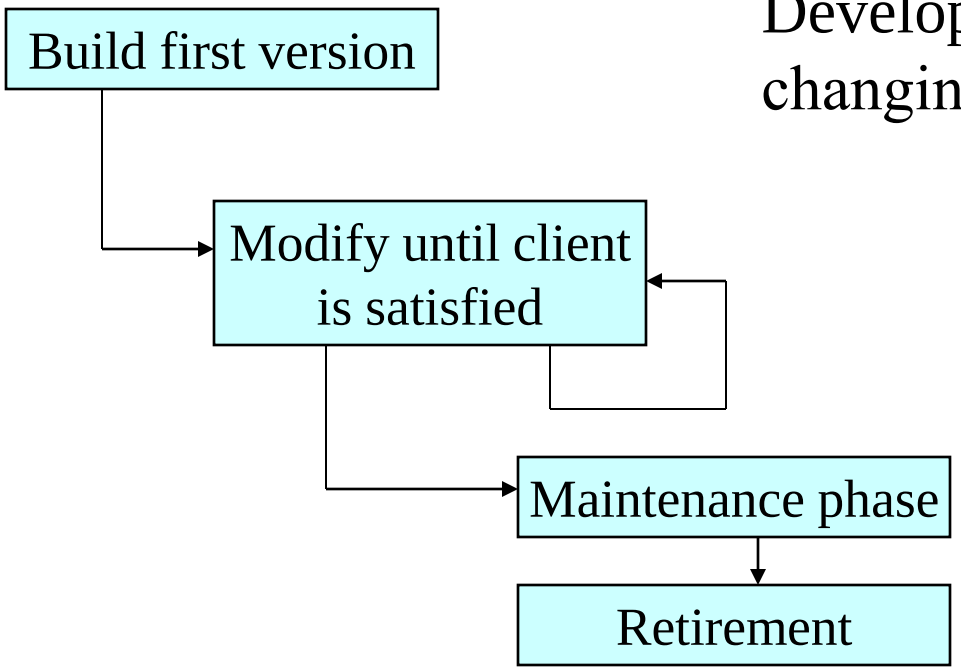
Build-and-fix



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

Options up here!

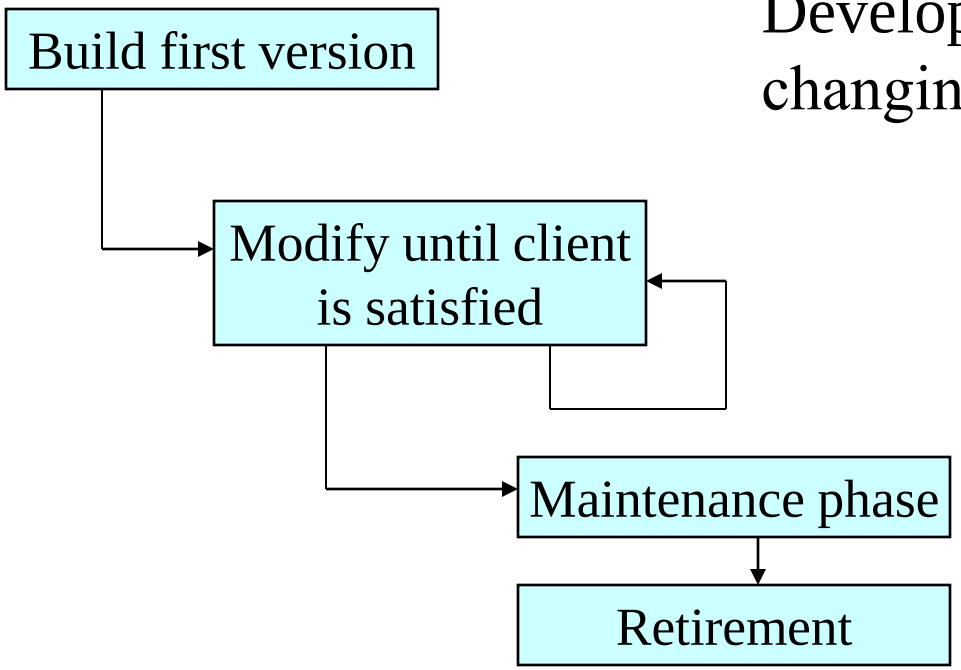
Build-and-fix



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

What problems might arise?

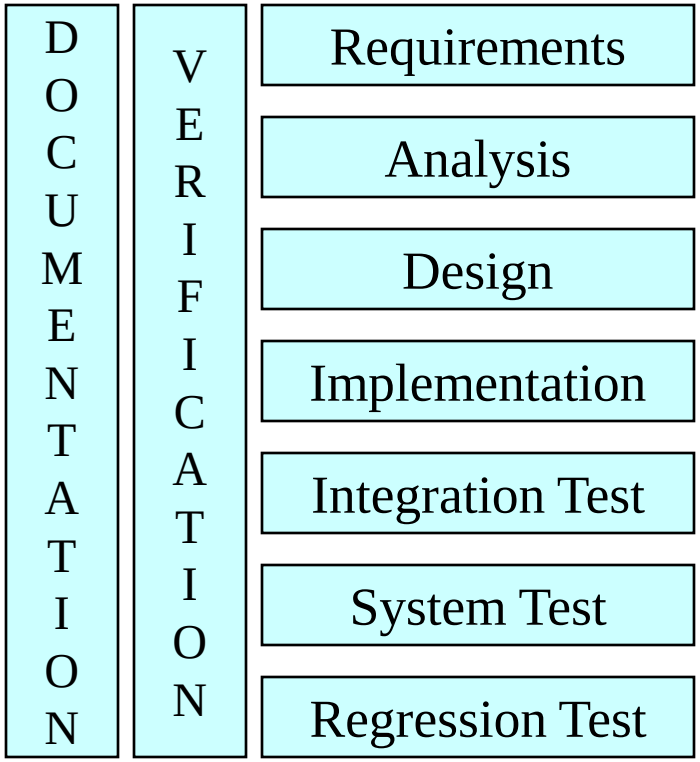
Build-and-fix



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

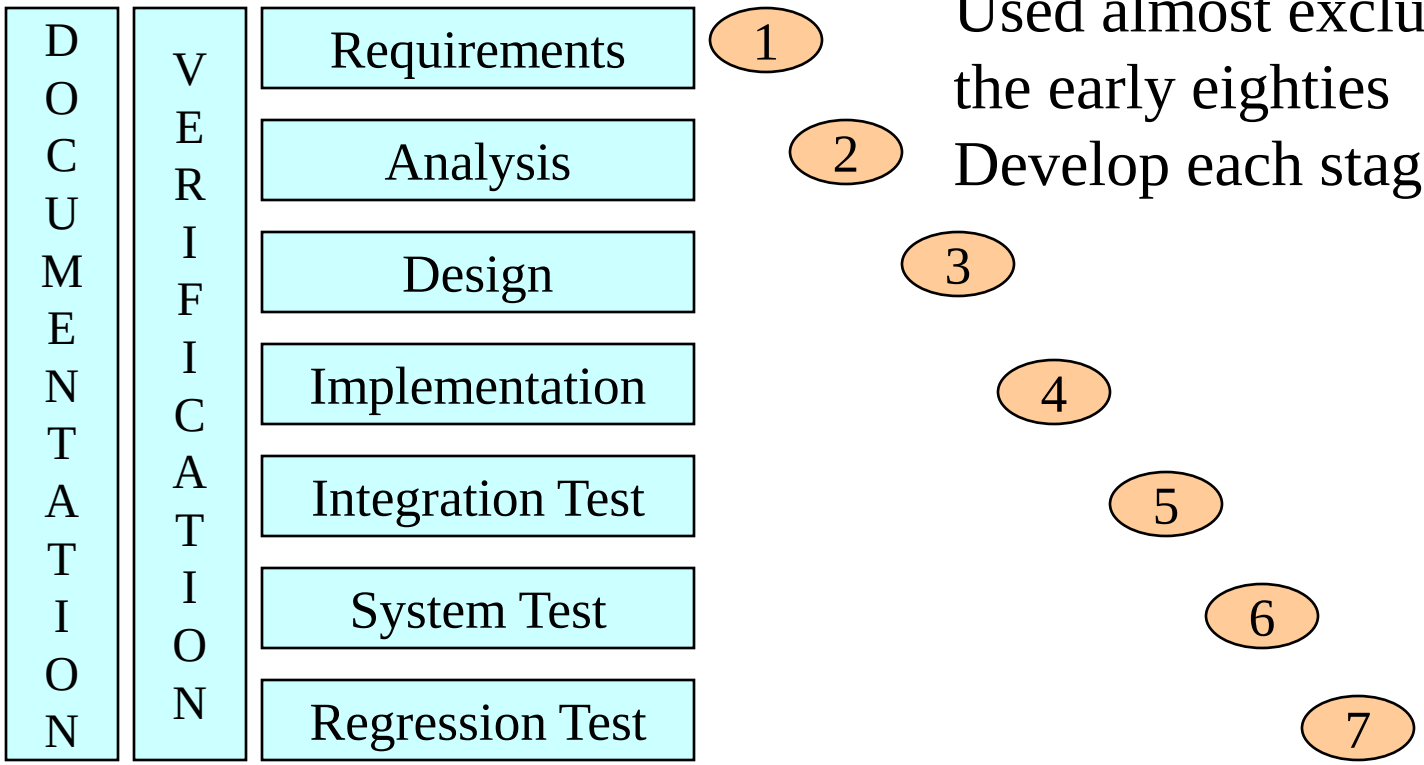


Used almost exclusively up to the early eighties

Develop each stage in sequence

Options up here!

Waterfall

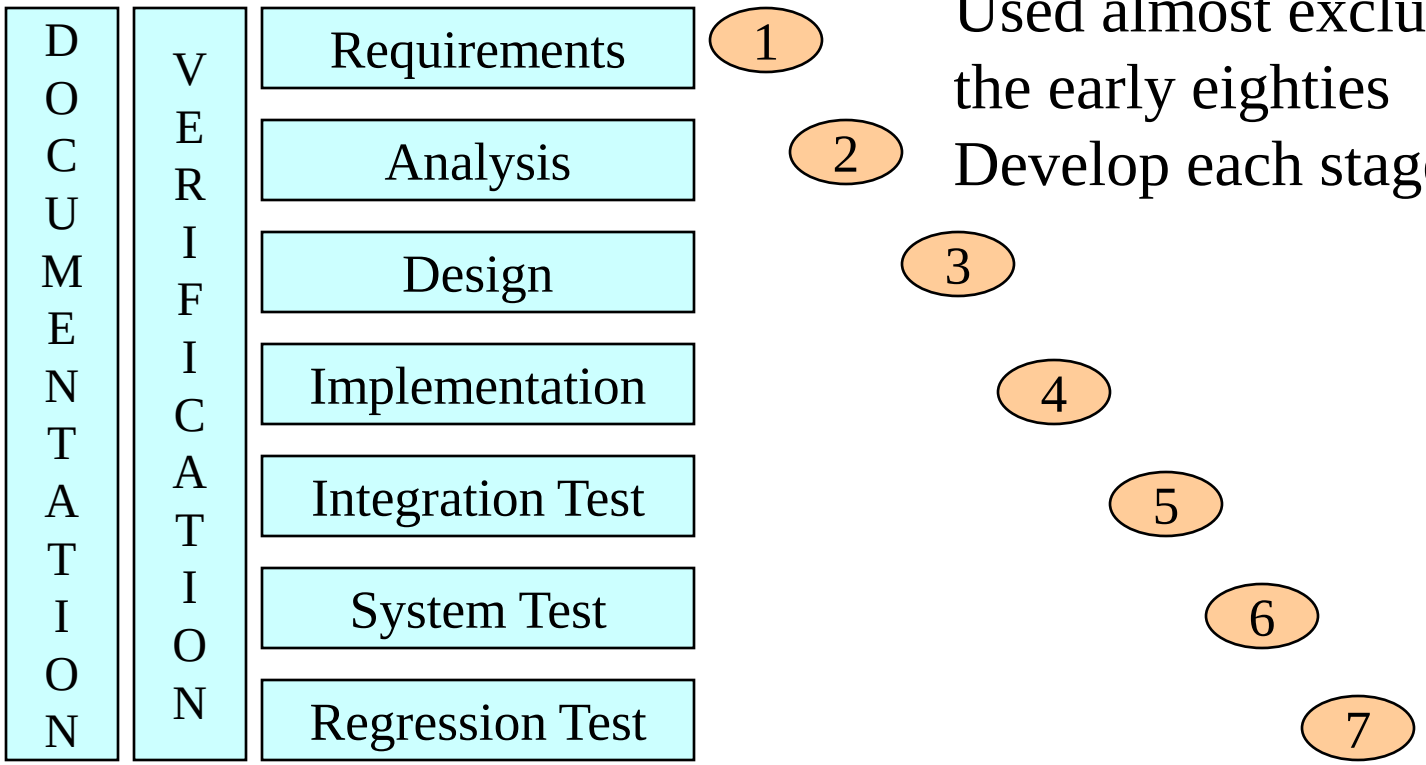


Used almost exclusively up to the early eighties

Develop each stage in sequence

What problems might arise?

Waterfall

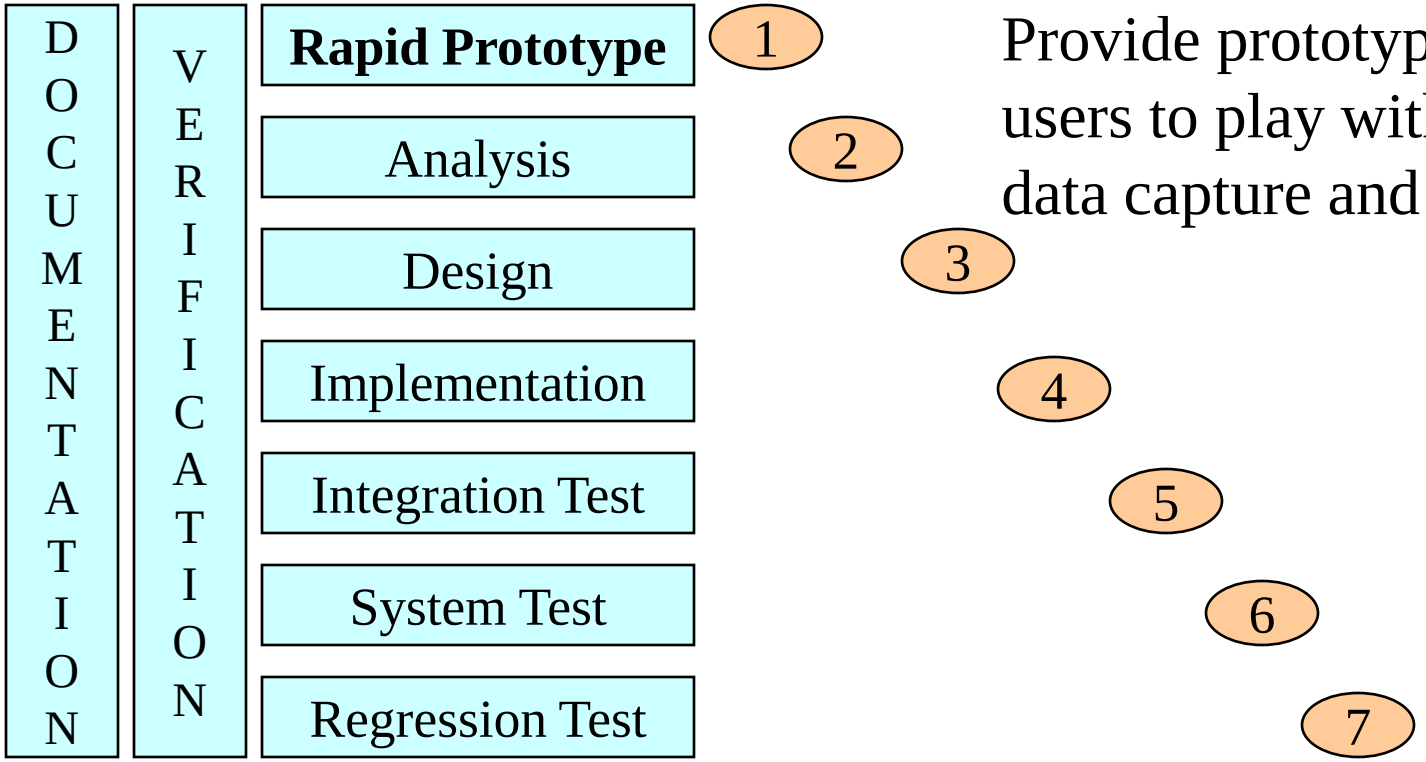


Used almost exclusively up to the early eighties

Develop each stage in sequence

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

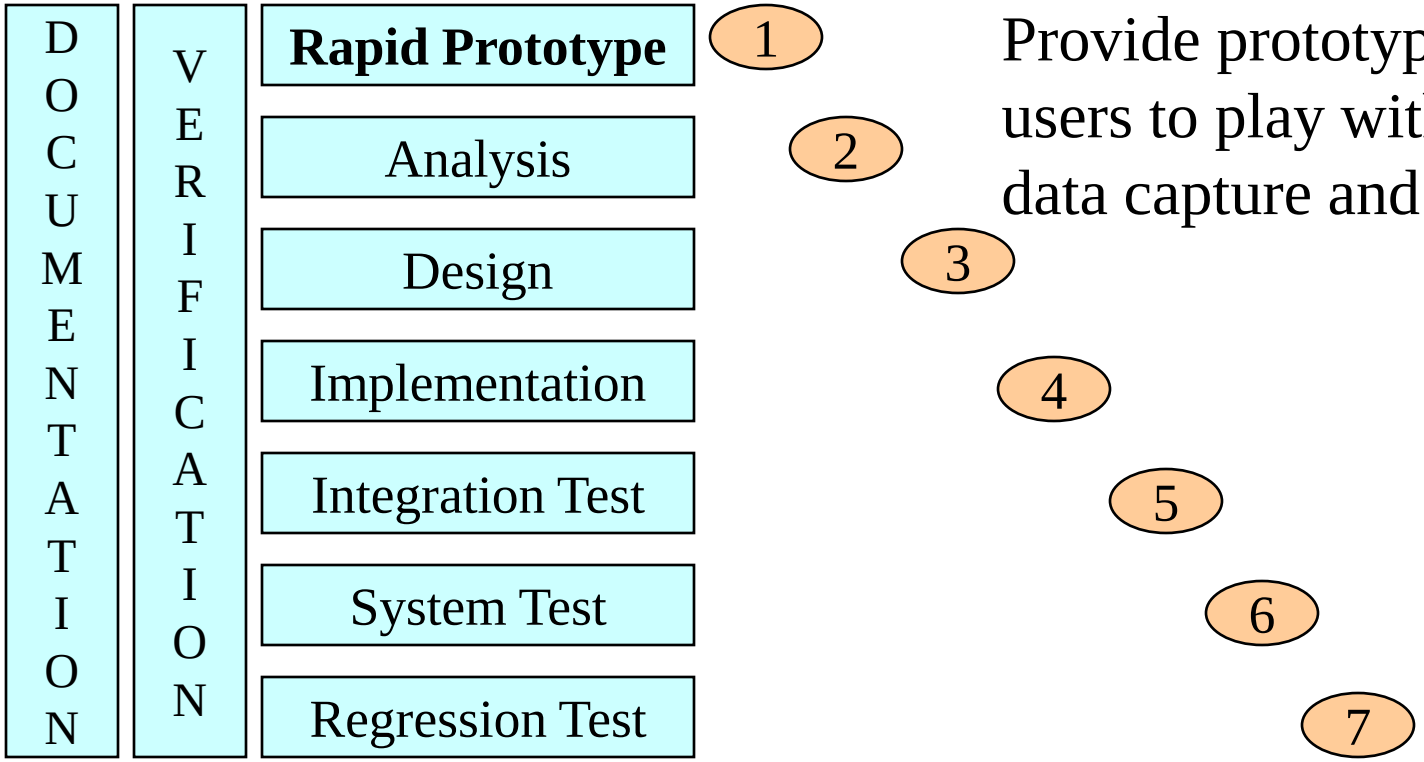
Rapid Prototyping



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

Options up here!

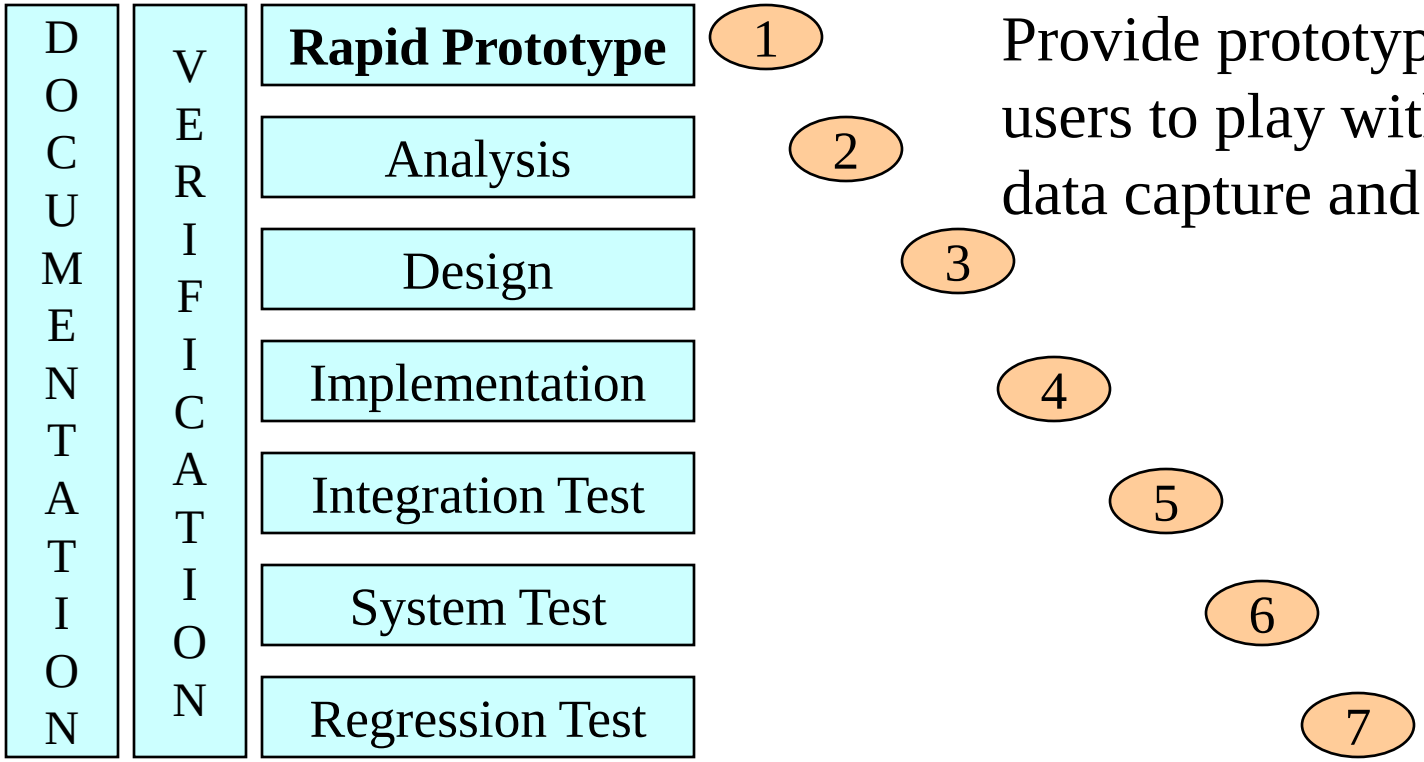
Rapid Prototyping



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

What problems might arise?

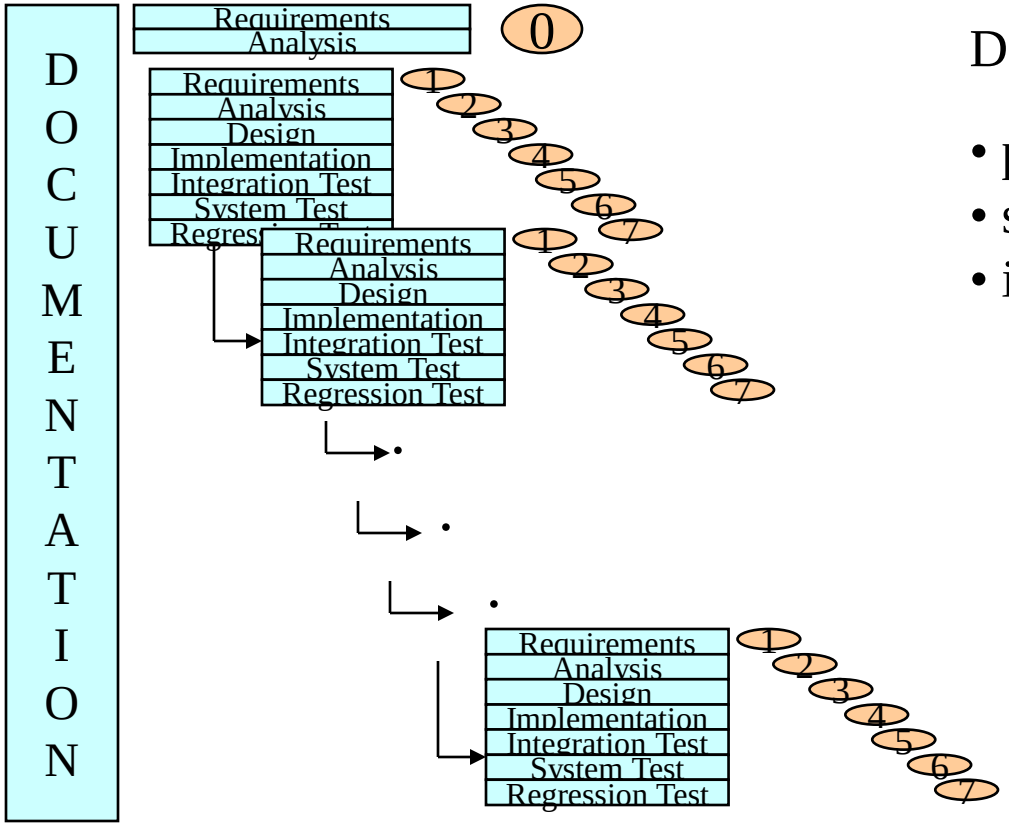
Rapid Prototyping



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".

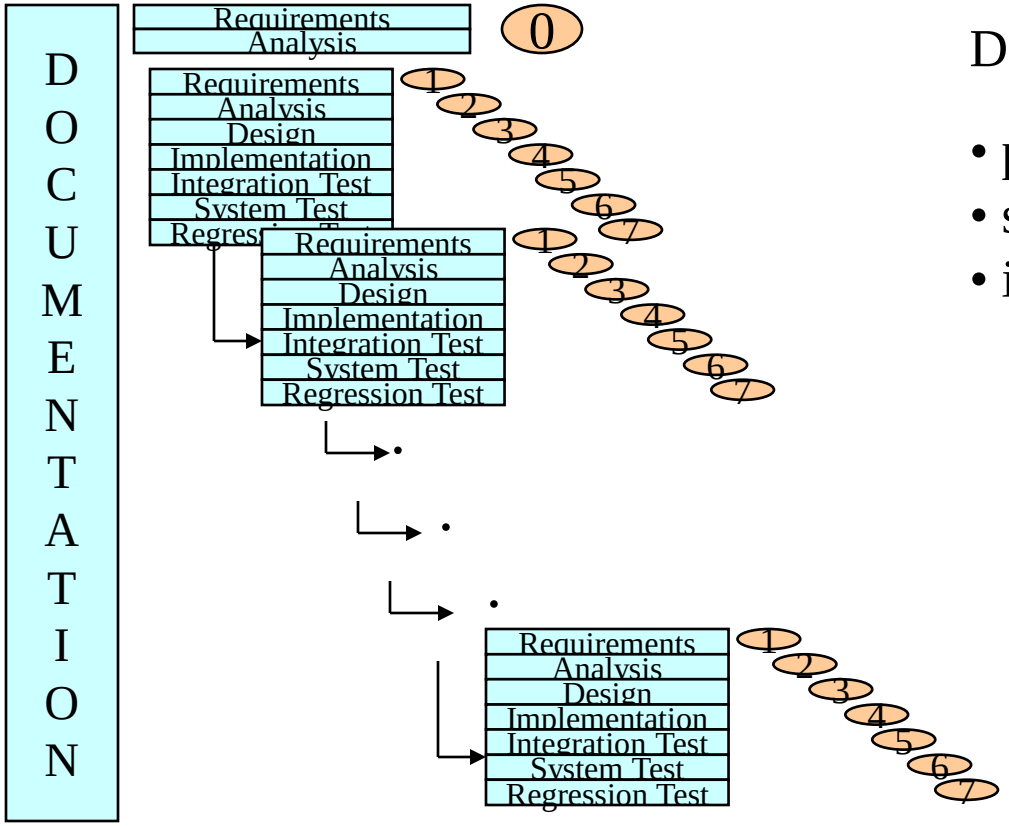
Incremental



Develop in small steps:

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

Incremental



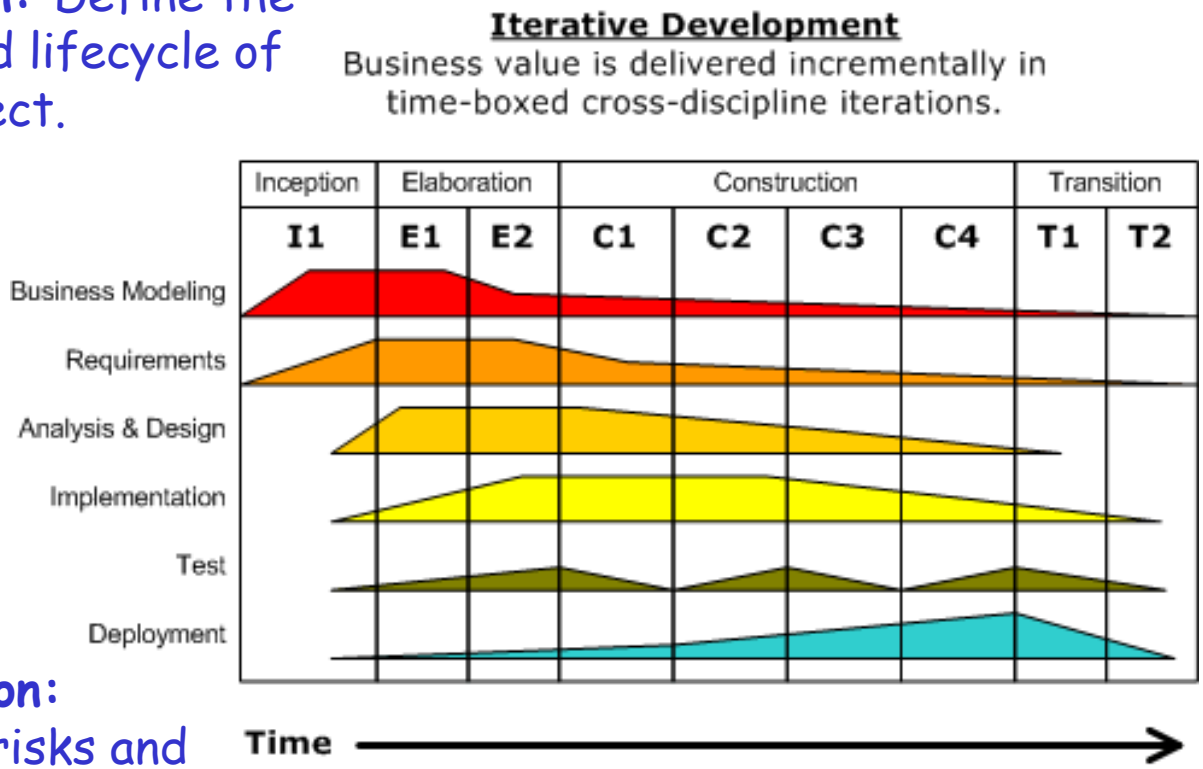
Develop in small steps:

- 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