Queen Mary
University of Lo

ECS505U
SOFTWARE ENGINEERING

MUSTAFA BOZKURT & LORENZO JAMONE

LECTURER IN SOFTWARE ENGINEERING

Queen Mary
University of London

Week 8

SOFTWARE LIFE-CYCLE MODELS

From Waterfall to Extreme Programming

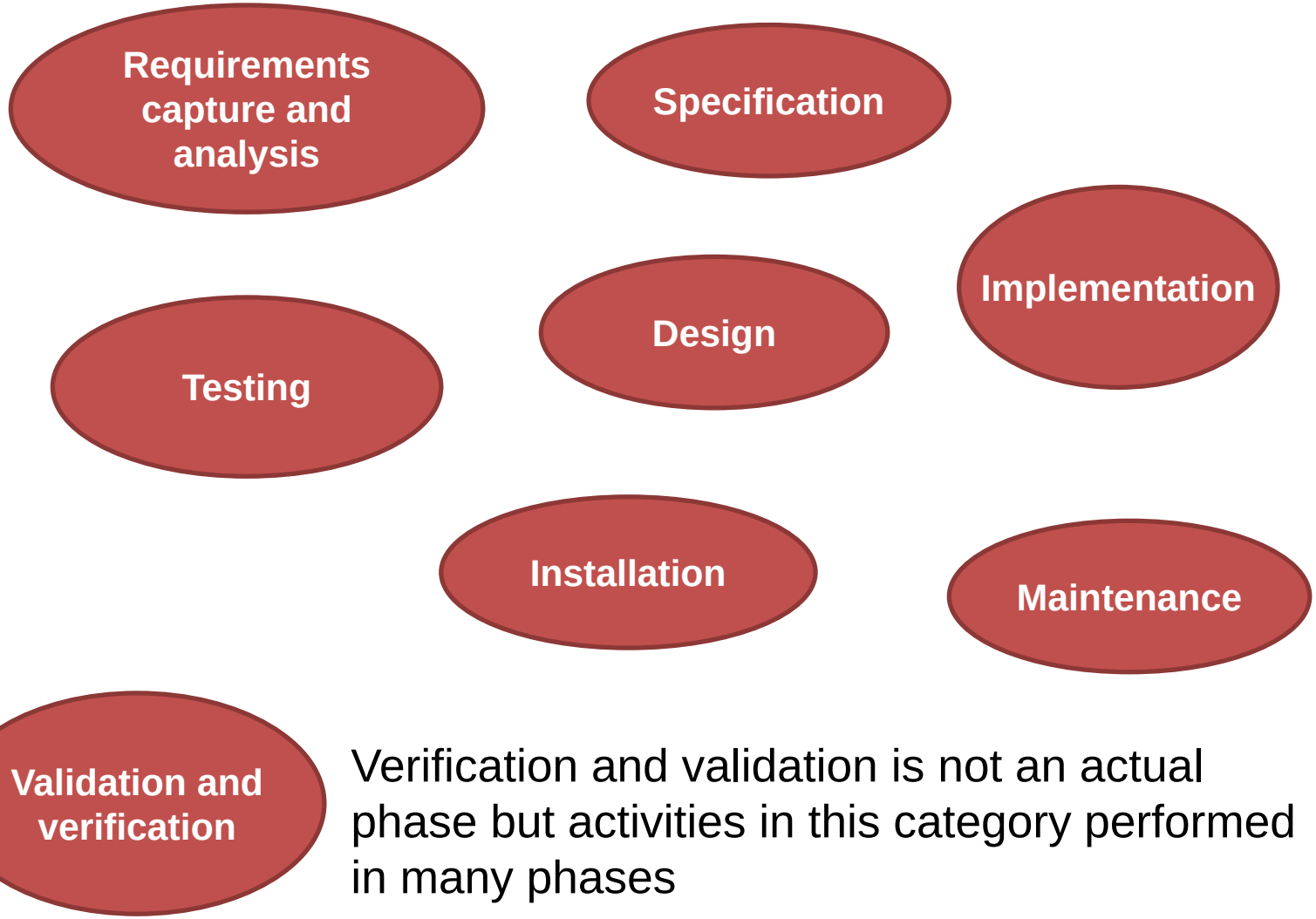
Queen Mary
University of London

LESSON OBJECTIVES

- Understand major activities of software projects
- Understand the place of these in different life-cycle models
- Understand the pros and cons of different life-cycle models
- Know enough about the incremental delivery and extreme programming to use relevant parts

SOFTWARE LIFE-CYCLE PHASES

Queen Mary University of London



Verification and validation is not an actual phase but activities in this category performed in many phases

REQUIREMENTS CAPTURE AND ANALYSIS

Queen Mary University of London

- Identify and agree general functional and non-functional requirements
- Prioritise requirements where possible
- Determine risk factors, cost analysis, development schedule
- Identify how customer might test completed system
- Identify likely changes and how to accommodate them

Queen Mary
University of London

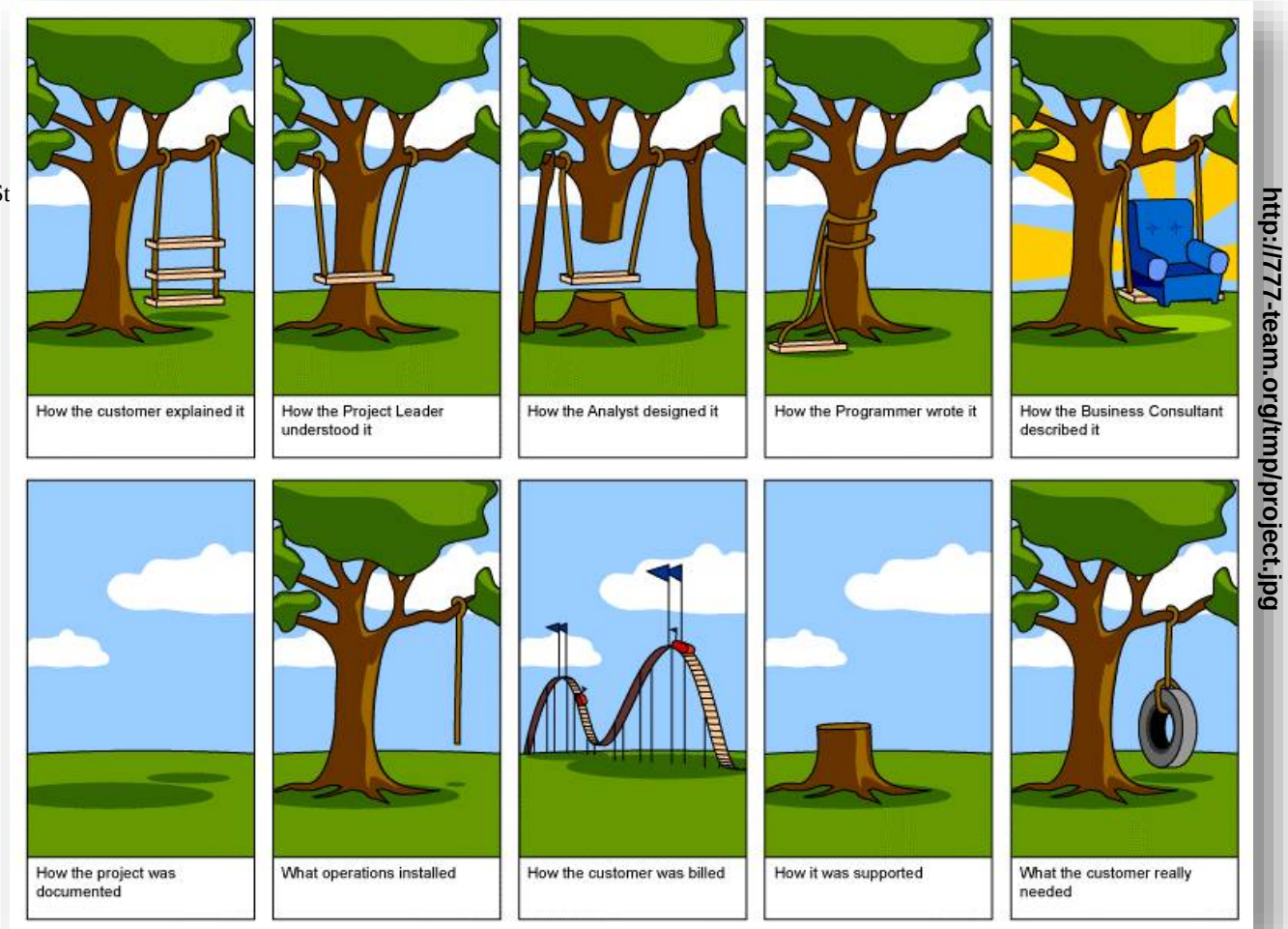
SPECIFICATION

- Says what the system does not how it does it
- Formal version of requirements specification, written by developer
- Developer identifies errors, omissions, or impracticalities in the requirements
- Crystallises precise functional and performance requirements
- Basis for formal contract, hence should contain sufficient explanation for customer to understand
- Should include detailed plan for acceptance test

THE PROBLEM WITH SPECIFICATION

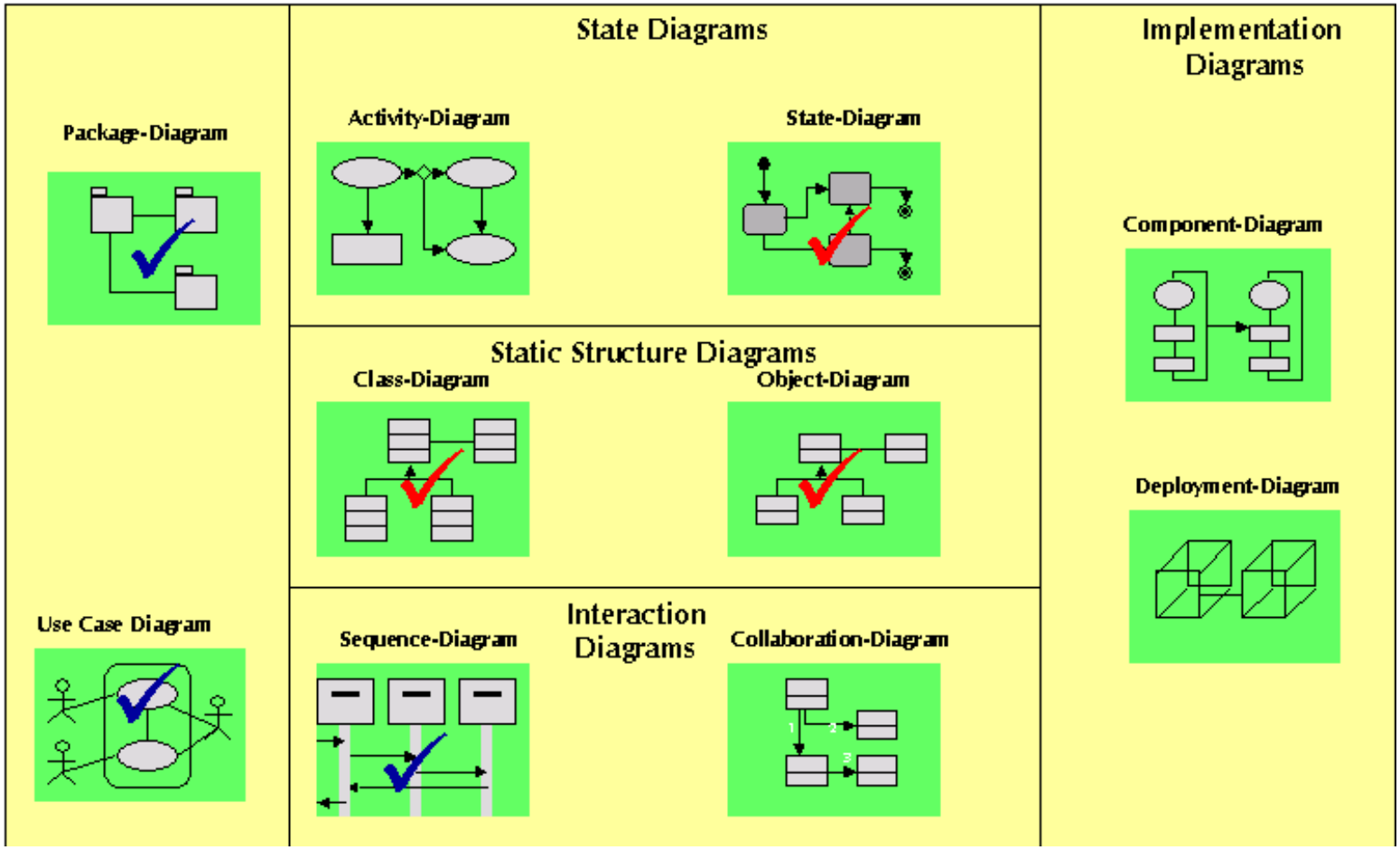
Queen Mary University of London

St



http://777-team.org/tmp/project.jpg

DESIGN

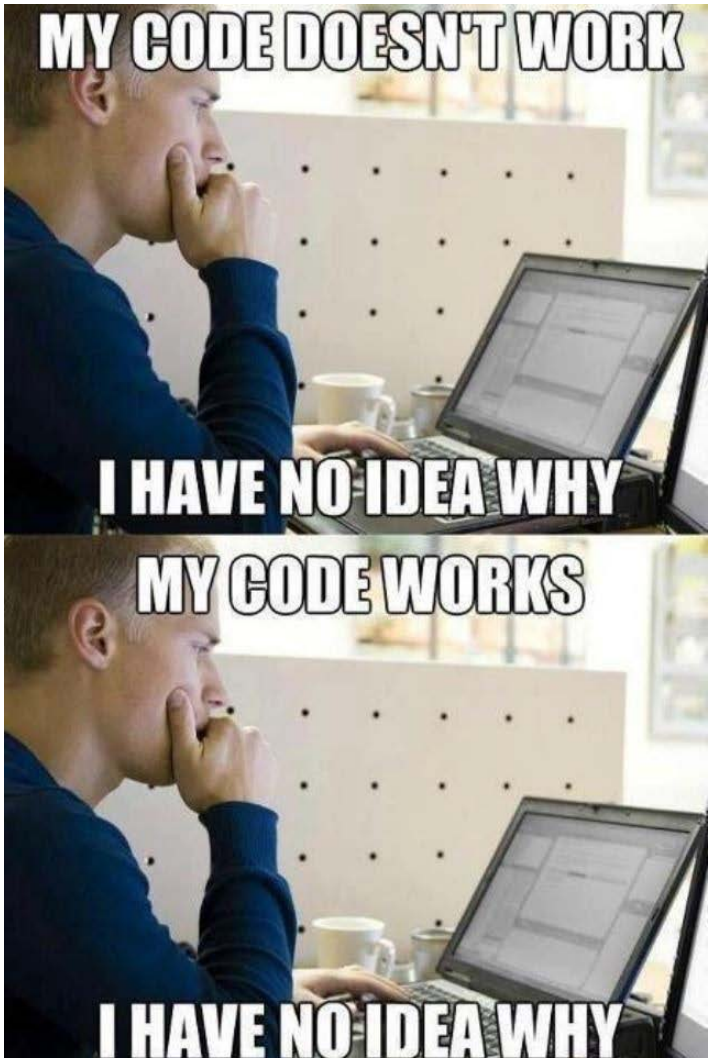


We will focus on object-oriented design using UML

IMPLEMENTATION

Process of expressing the detailed design in a programming language so that it will run on the target computer.

TESTING


MY CODE DOESN'T WORK
I HAVE NO IDEA WHY
MY CODE WORKS
I HAVE NO IDEA WHY

TESTING

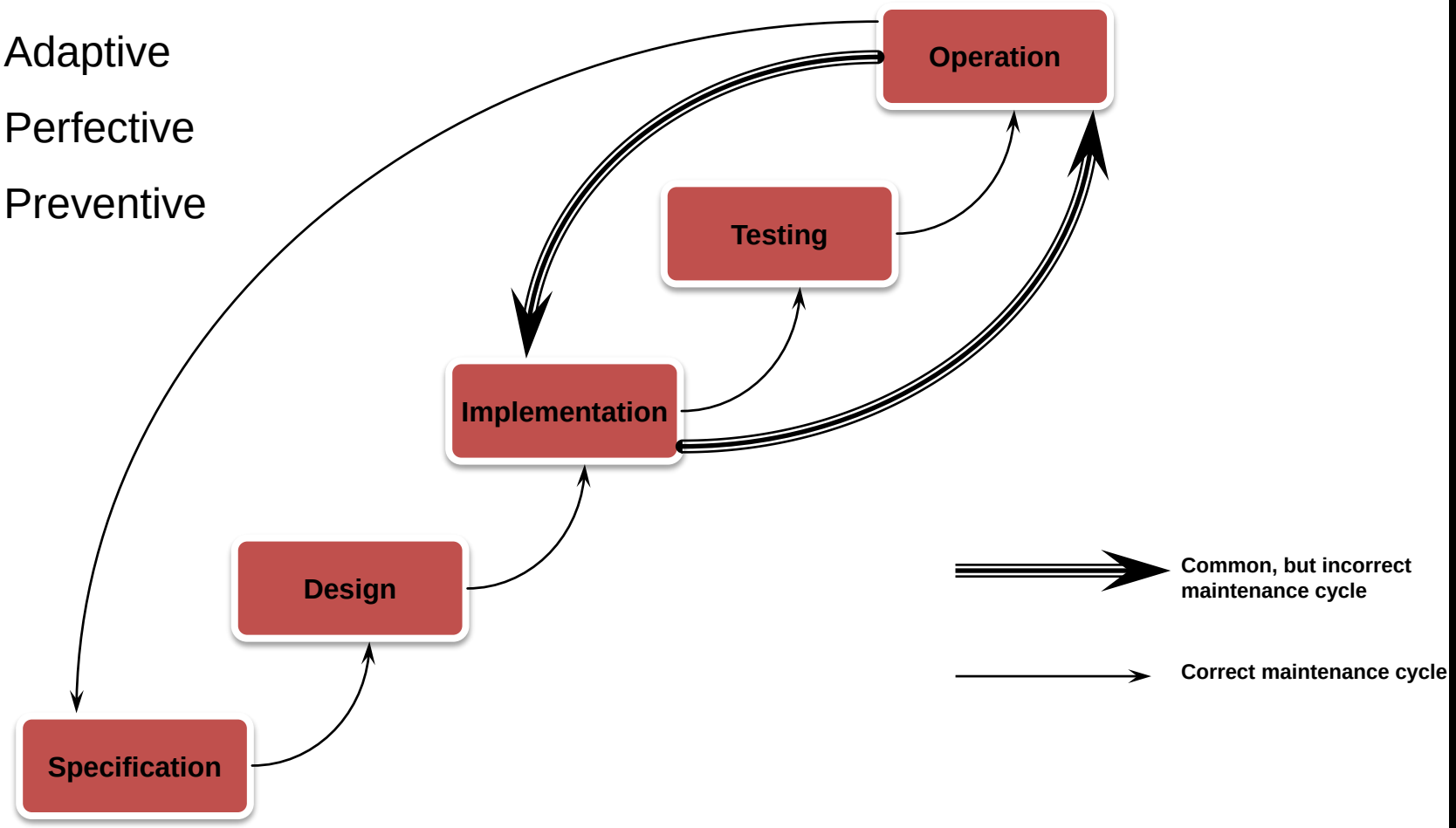
- Unit tests
- Integration tests
- System and acceptance tests
- Regression tests

INSTALLATION

- Create working version on target machine
- Make available all support files and manuals
- Coordinate with hardware
- Rerun acceptance test
- Training and on-line support
- Evaluation of project

MAINTENANCE AND SUPPORT

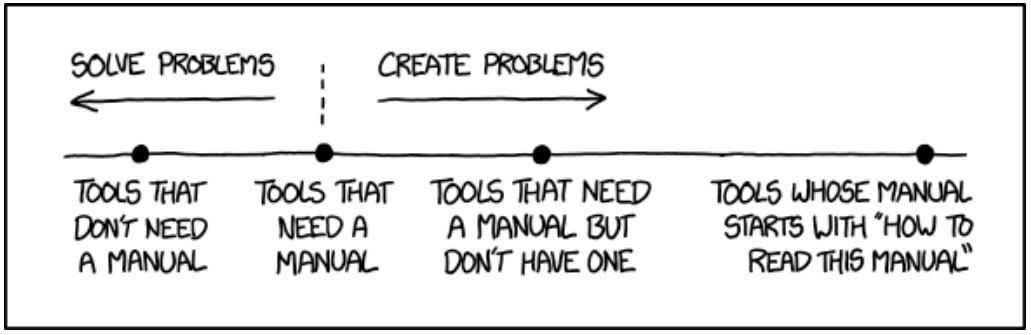
- Corrective
- Adaptive
- Perfective
- Preventive

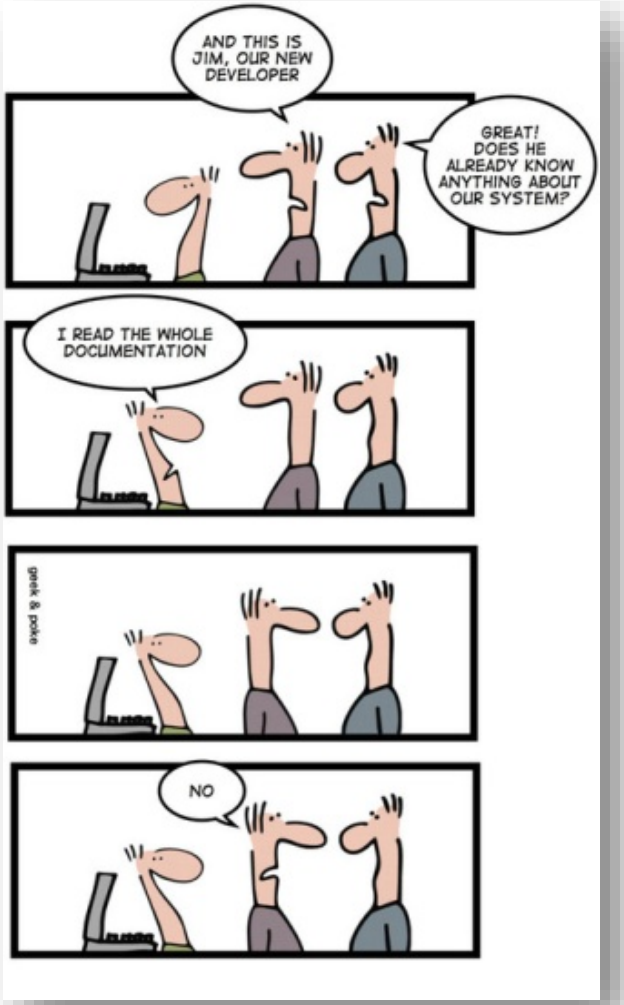


SOFTWARE DOCUMENTATION

Types of documentation:

- Requirements
- Architecture/Design
- Technical
- End user
- Marketing





http://media.tumblr.com/tumblr_m8pz5fKnq21rps1w0.jpg

SOFTWARE QUALITY CONTROL

Queen Mary University of London

Verification:

Are we building the product right?

Validation:

Are we building the right product?

VERIFICATION & VALIDATION

Queen Mary University of London

According to the Capability Maturity Model (CMMI-SW v1.1),

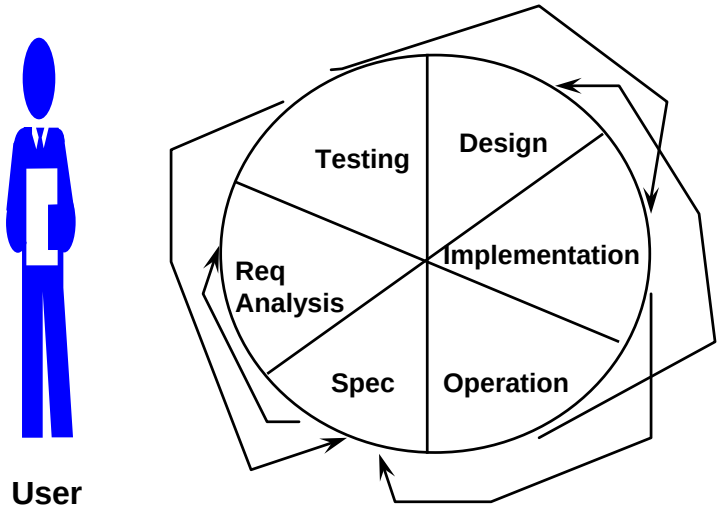
Software Validation: The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements. [IEEE-STD-610]

Software Verification: The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. [IEEE-STD-610]

VERIFICATION & VALIDATION

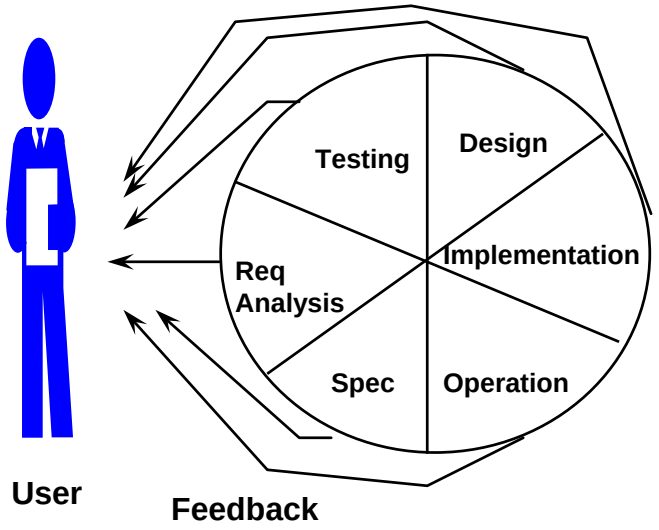
Queen Mary University of London

Verification



Are we building the product right?

Validation



Are we building the right product?

VERIFICATION & VALIDATION

Queen Mary University of London

Verification activities are concerned with ensuring that the development is internally consistent and correct.

- For example, that development standards are adhered to, that formal reasoning demonstrates the design/code is a faithful implementation of the specification. Unit testing is a method of verification.

Validation activities are concerned with demonstrating to the outside world that the system is satisfactory.

- System testing against customer requirements and expectations is a form of validation.

Software inspection techniques (like Fagan inspections) cover both verification and validation.

Queen Mary University of London

LIFE-CYCLE MODELS

- Build-and-fix
- Waterfall
- Formal transformations model
- Spiral model
- Unified process (UP)
- Extreme programming (XP)

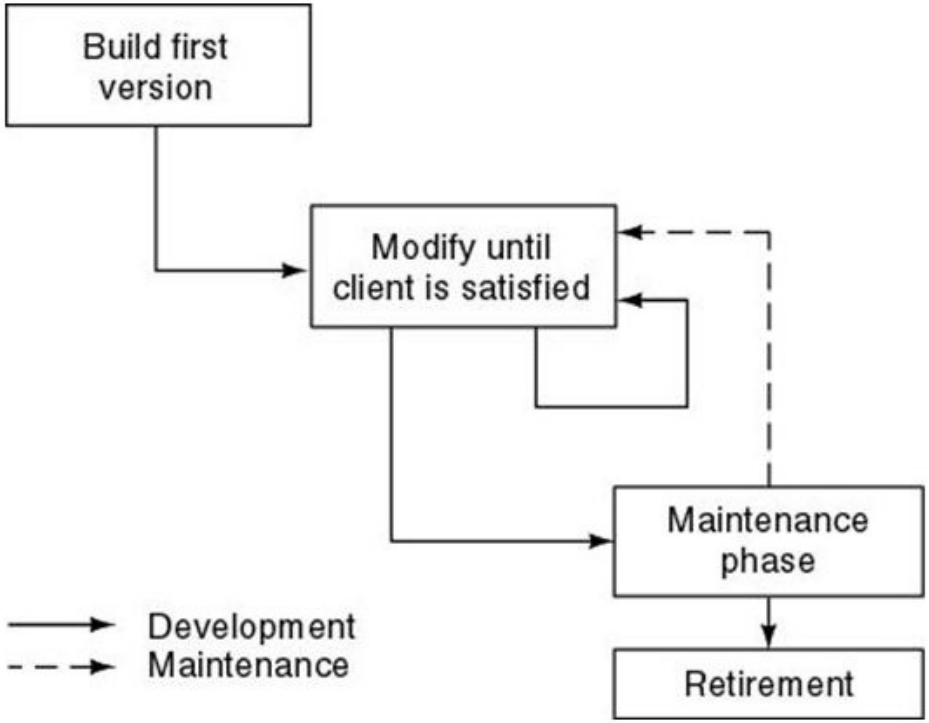
BUILD-AND-FIX MODEL

Problems

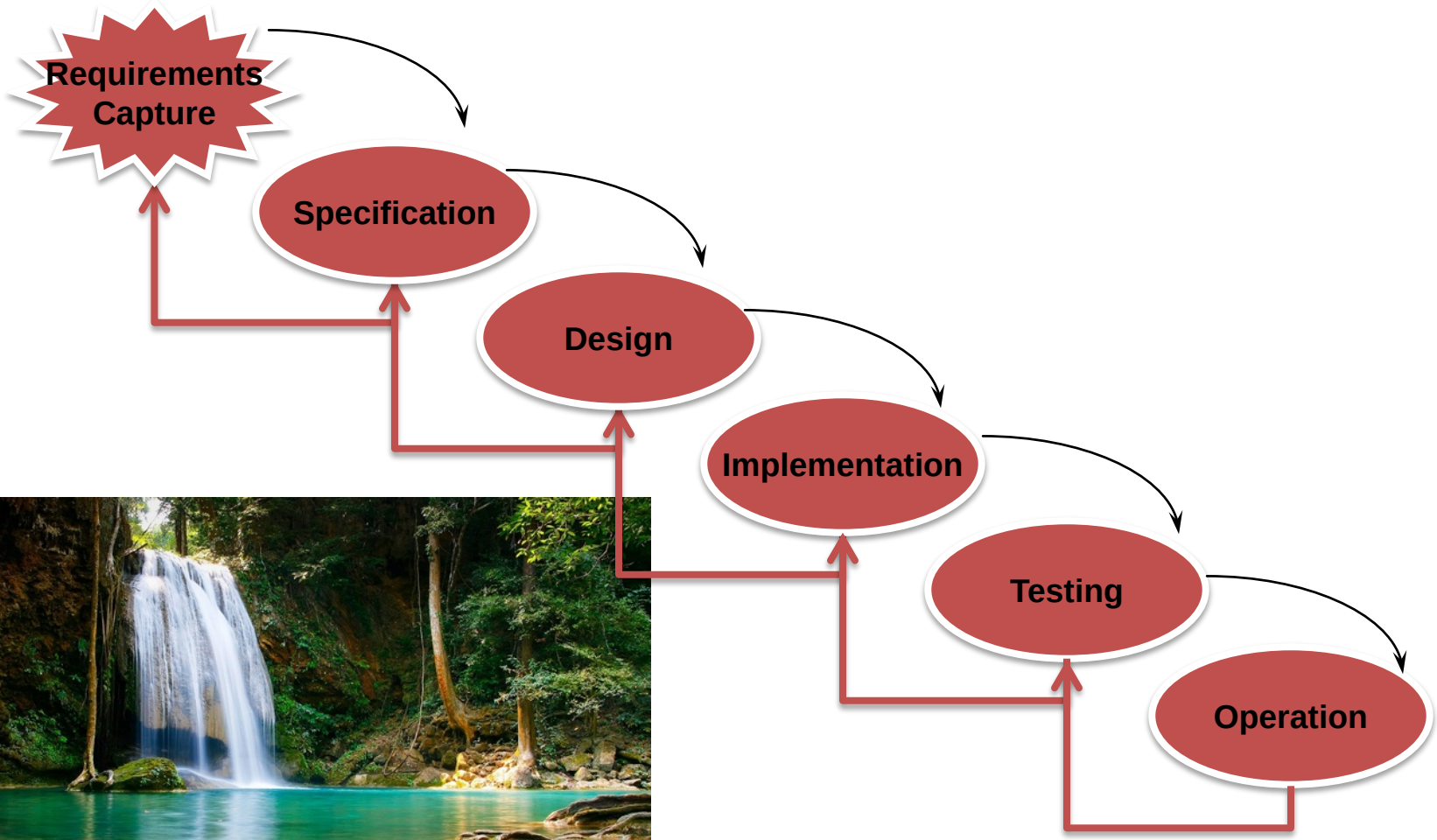
- No specifications
- No design

Totally unsatisfactory

- High cost
- Difficult maintenance



WATERFALL MODEL



WATERFALL MODEL

Advantages:

Enforces disciplined approach

- Documentation for each phase
- Products of each phase checked for QA

Maintenance is easier

- Every change reflected in the relevant documentation

WATERFALL MODEL

Disadvantages:

- Working version of the software will not be available until late in the project time-span
- Specifications are long, detailed
- “Blocking states” – some project team members must wait for other team members to complete dependent tasks
- No prototyping – makes it hard to gather requirements

WATERFALL MODEL

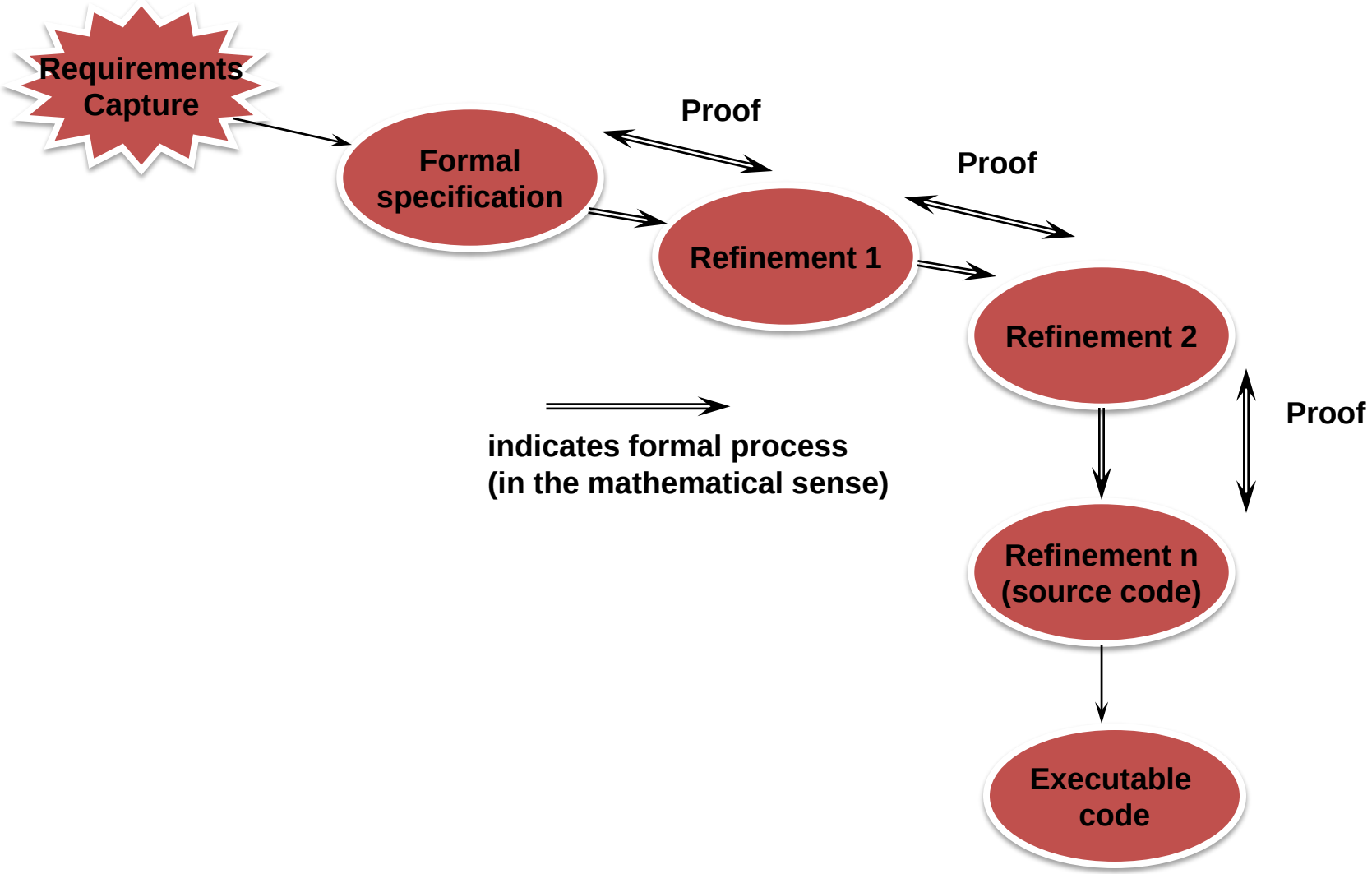
A common feature of waterfall projects is that they tend to fail.

18% cancelled, 53% late, over budget or descoped[1].

1. CHAOS Demographics and Project Resolution 2004, Standish Group

FORMAL TRANSFORMATION MODEL

Queen Mary University of London



FORMAL TRANSFORMATION MODEL

Queen Mary University of London

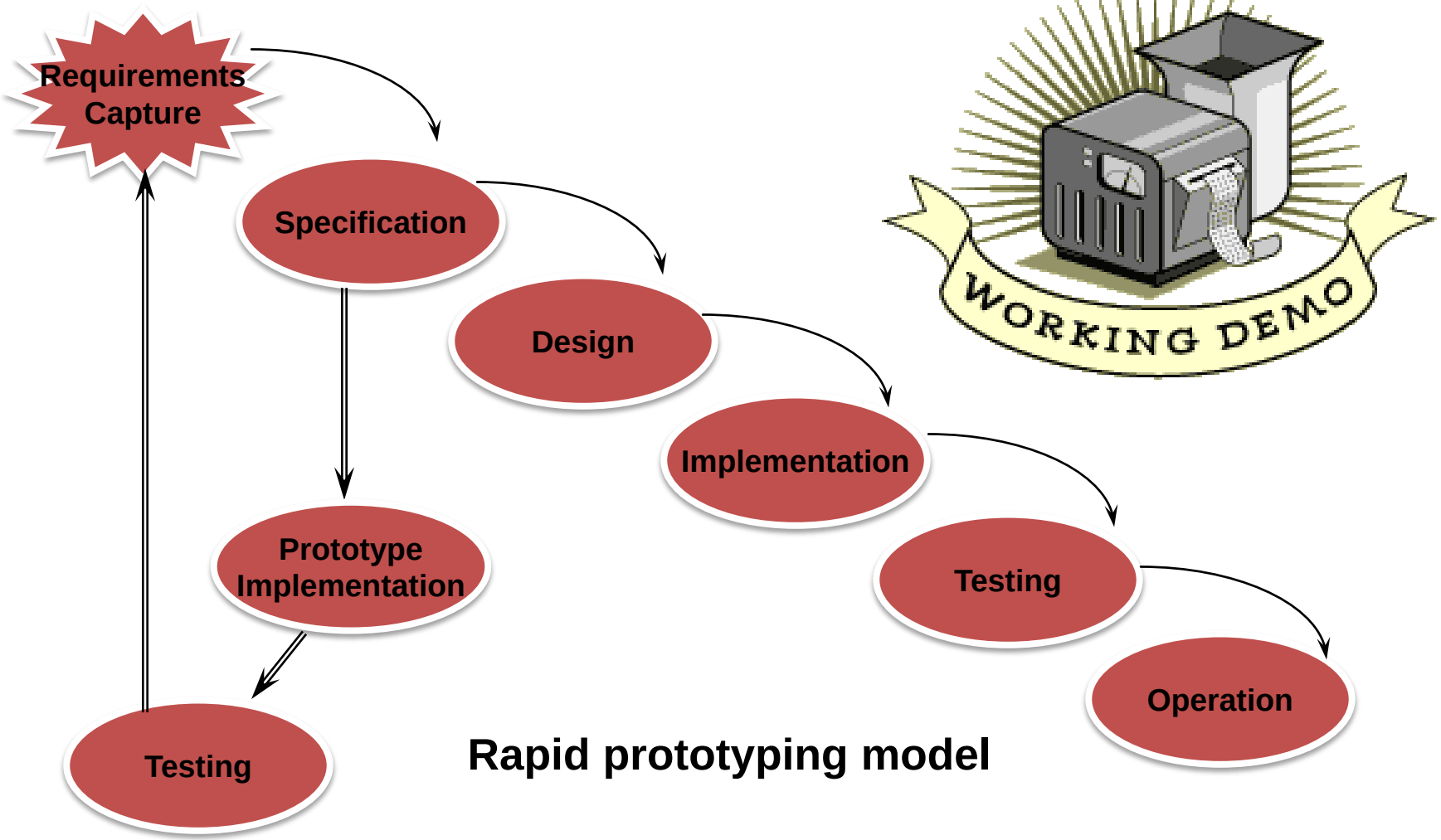
Advantages:

- Very precise
- Often used in safety critical systems.

Disadvantages:

- Considerable expertise is required.
- Extra level of complexity!
- It is very costly and might become error prone with excessive verification (due to errors in transformation).

PROTOTYPING



Rapid prototyping model

PROTOTYPING

Throwaway/Rapid prototyping refers to the creation of a model that will eventually be discarded rather than becoming part of the final delivered software. After preliminary requirements gathering is accomplished, a simple working model of the system is constructed to visually show the users what their requirements may look like when they are implemented into a finished system.

Evolutionary prototyping: The main goal of Evolutionary Prototyping is to build a very robust prototype in a structured manner and constantly refine it. The reason for this is that the Evolutionary prototype, when built, forms the heart of the new system, and the improvements and further requirements will be built.

Incremental prototyping: The final product is built as separate prototypes. At the end the separate prototypes are merged in an overall design.

PROTOTYPING

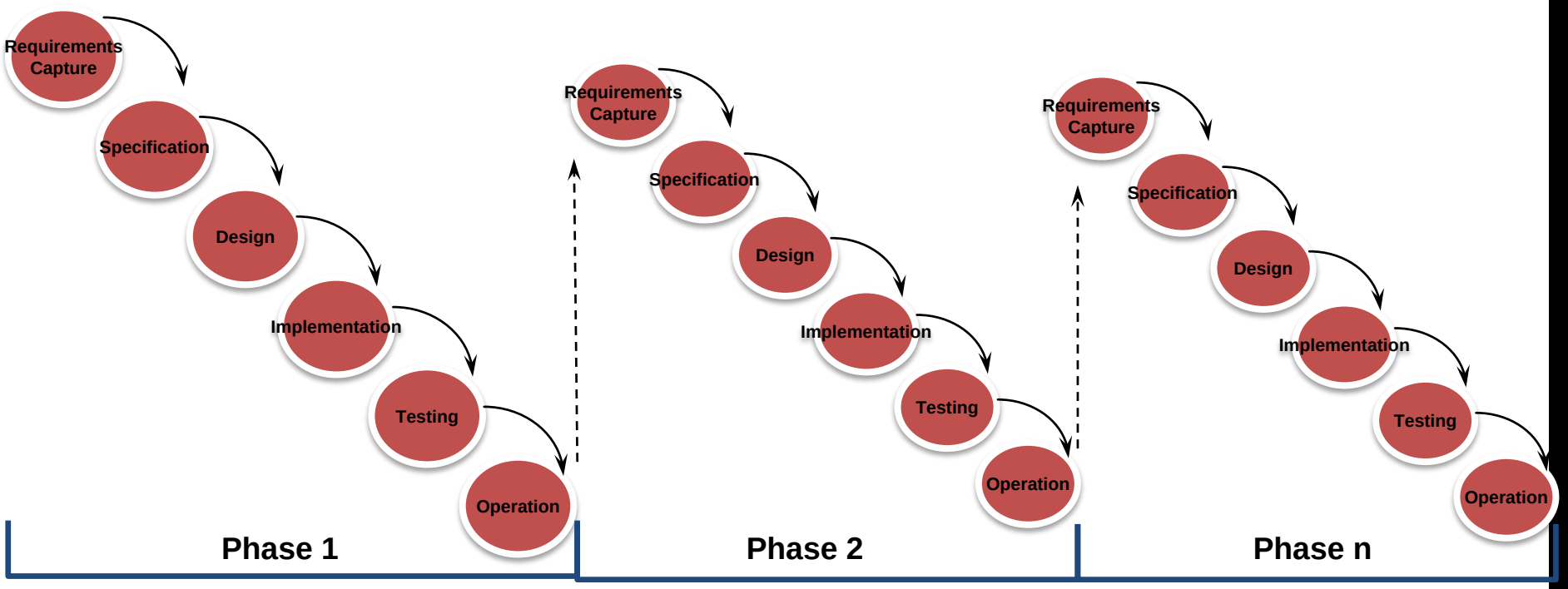
Advantages:

- Reduced time and costs!
- Speed and accuracy (due to user involvement)

Disadvantages:

- Client disappointment!
- Hard to focus developers to implement a prototype. (developer attachment!)
- Excessive time and effort (expense).
- Legal issues might arise.

ITERATIVE DEVELOPMENT



“You should use iterative development only on projects that you want to succeed”

Martin Fowler, 2000

ITERATIVE DEVELOPMENT

Advantages:

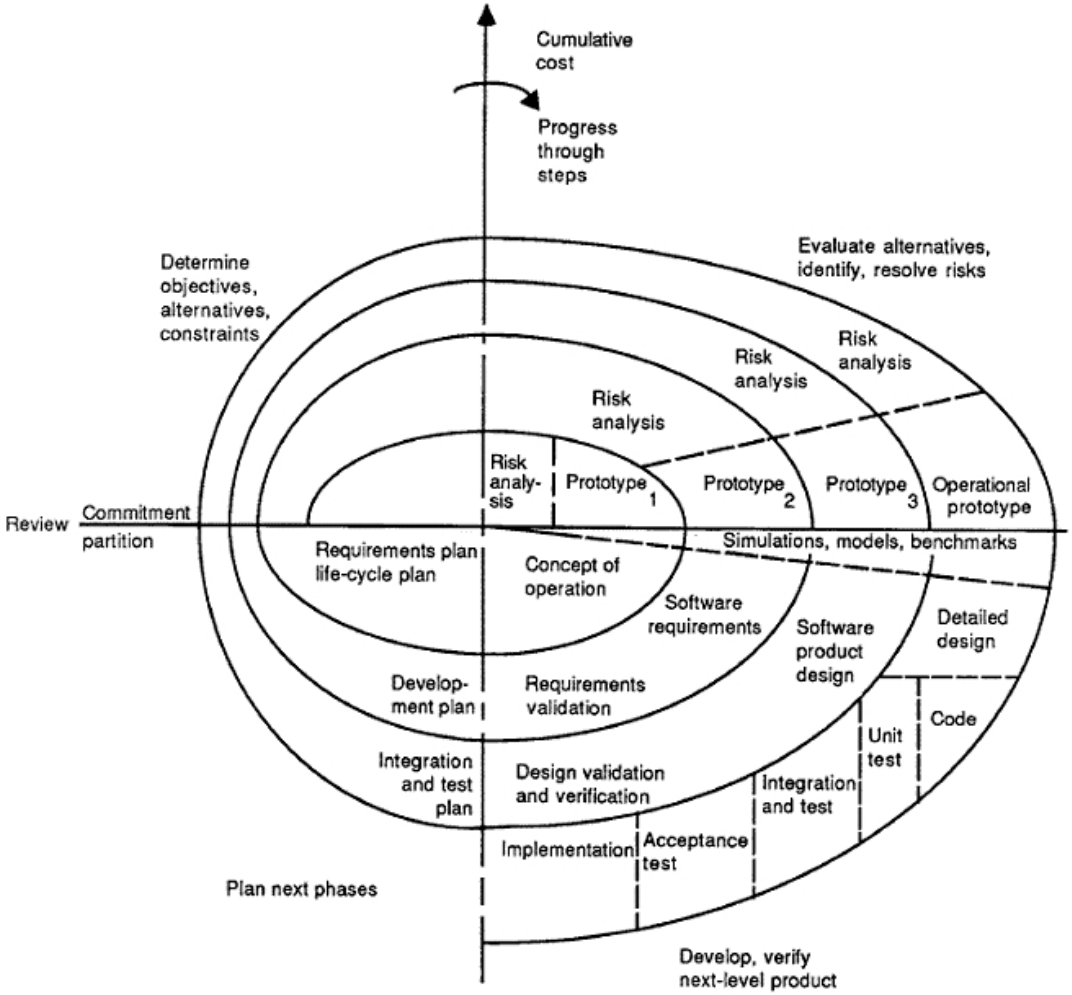
- Customer feedback due to early working software.
- Requirements and specifications in phases.
- Step-by-step improvement makes easy to track defects.

ITERATIVE DEVELOPMENT

Disadvantages:

- Without careful planning early design decisions can cripple future evolutions.
- Difficult to switch methodologies and languages.
- Not having all requirements might cause problems.

SPIRAL MODEL



SPIRAL MODEL

Advantages:

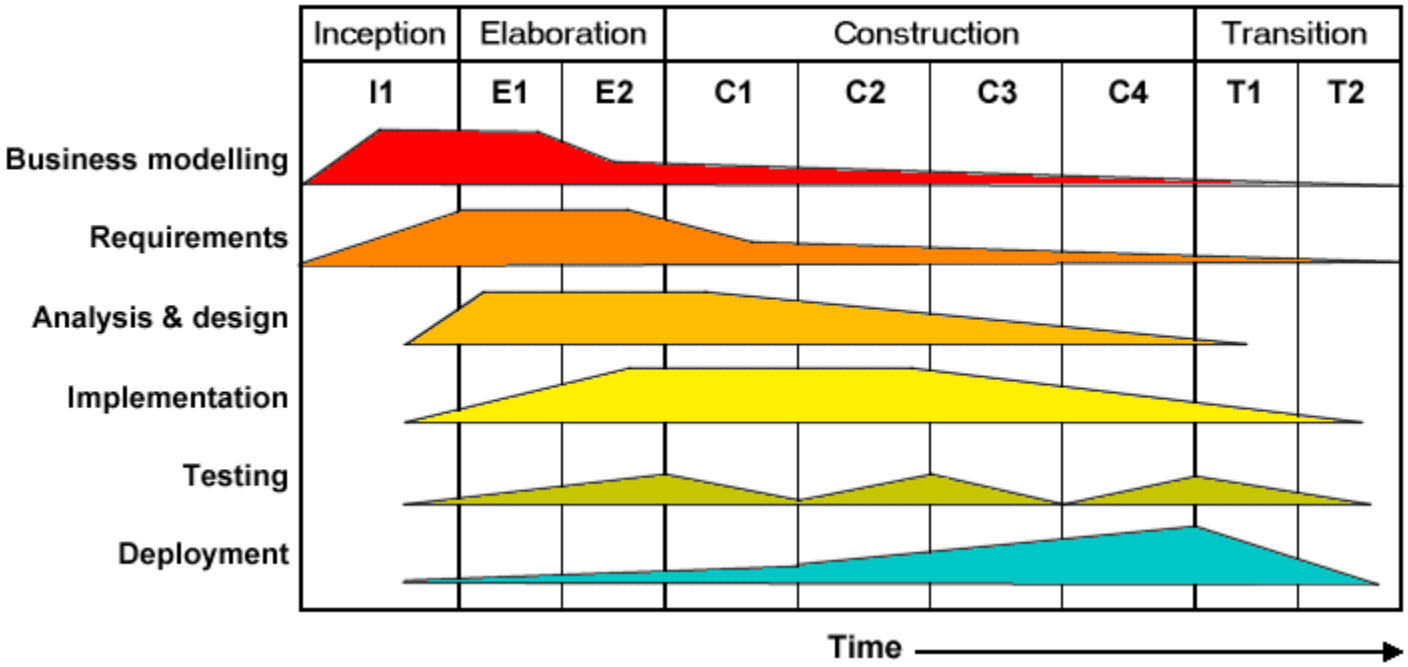
- Realistic approach for large-scale projects.
- Prototyping to reduce risk.
- Iterative and incremental approach.
- Reduced risk.

SPIRAL MODEL

Disadvantages:

- Requires considerable expertise in risk-assessment.
- Not proven enough due to less employment!

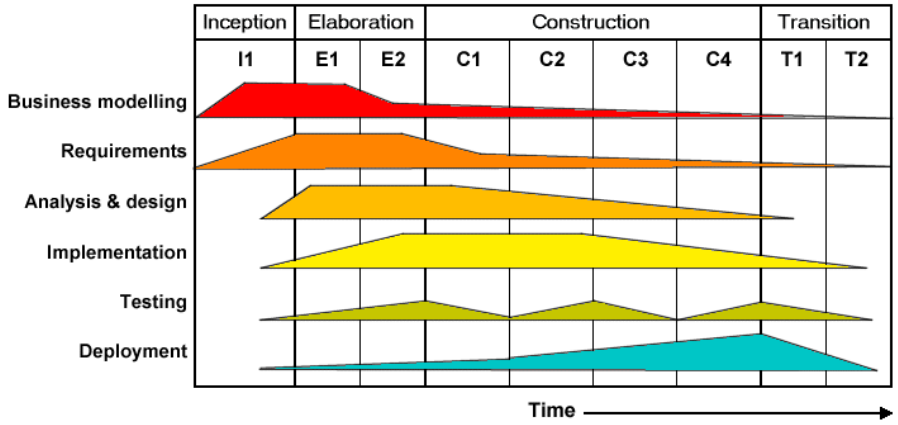
UNIFIED PROCESS OUTLINE



The balance between workflows is different in different phases

UNIFIED PROCESS (UP)

- Current state-of-the-art methodology
- Initially developed by designers of UML
- Structures project as a number of phases
- Each phase contains several iterations
- Different workflows (activities) are performed in each iteration



UNIFIED PROCESS

Inception:

- Establish the **project scope** and **boundary** conditions
- Outline the **use cases** and key requirements for design tradeoffs
- Outline one or more candidate architectures
- Identify risks
- Prepare a preliminary project schedule and cost estimate

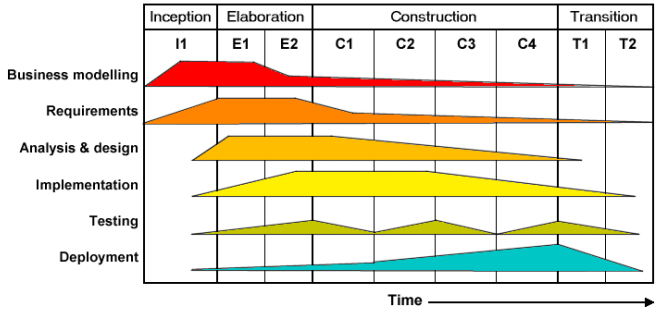
Elaboration:

- Capture requirements
- Design software architecture
- Plan for construction phase

Construction (build using UML diagrams)

Transition

- Deployment of initial release
- Feedback reports defects and changes



UNIFIED PROCESS

Advantages:

- Well-documented and complete methodology
- Adapts easily to changing requirements
- Reduced integration time and effort
- Higher level of reuse

UNIFIED PROCESS

Disadvantages:

- The process is too complex
- Not natural way of developing code
- Disorganised development
- Continuous integration might be challenging

UML AND THE UNIFIED PROCESS

Queen Mary University of London

UML diagrams can be used in conjunction with many different processes (or even in the absence of a formal process) however there is a close fit between UML and the UP.

AGILE MODELS

Why would you want a destroyer when you have an aircraft carrier?

AGILE MODELS

- Agile movement proposes alternatives to traditional project management.
- Agile approaches are typically used in software development to help businesses respond to unpredictability.

AGILE MODELS

Philosophy

- Encourages customer satisfaction and early incremental delivery of the software
- Small highly motivated project teams
- Informal methods
- Minimal software engineering work products
- Overall development simplicity

Development guidelines

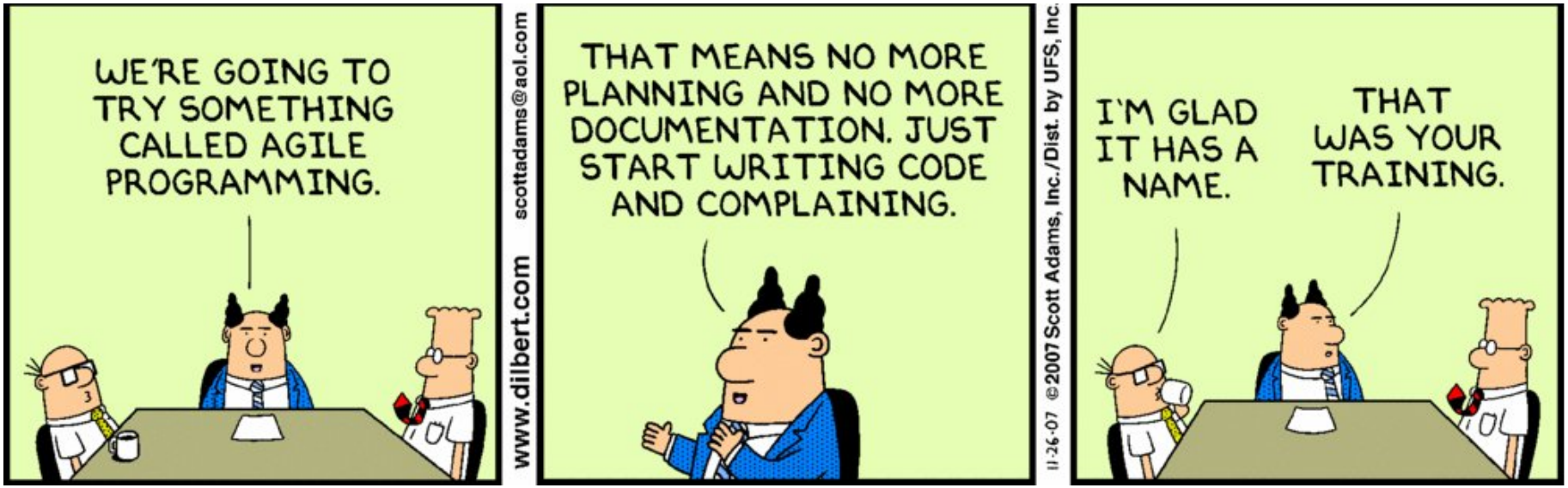
- Stress delivery over analysis and design
- Active and continuous communication between developers and customers

AGILE MODELS

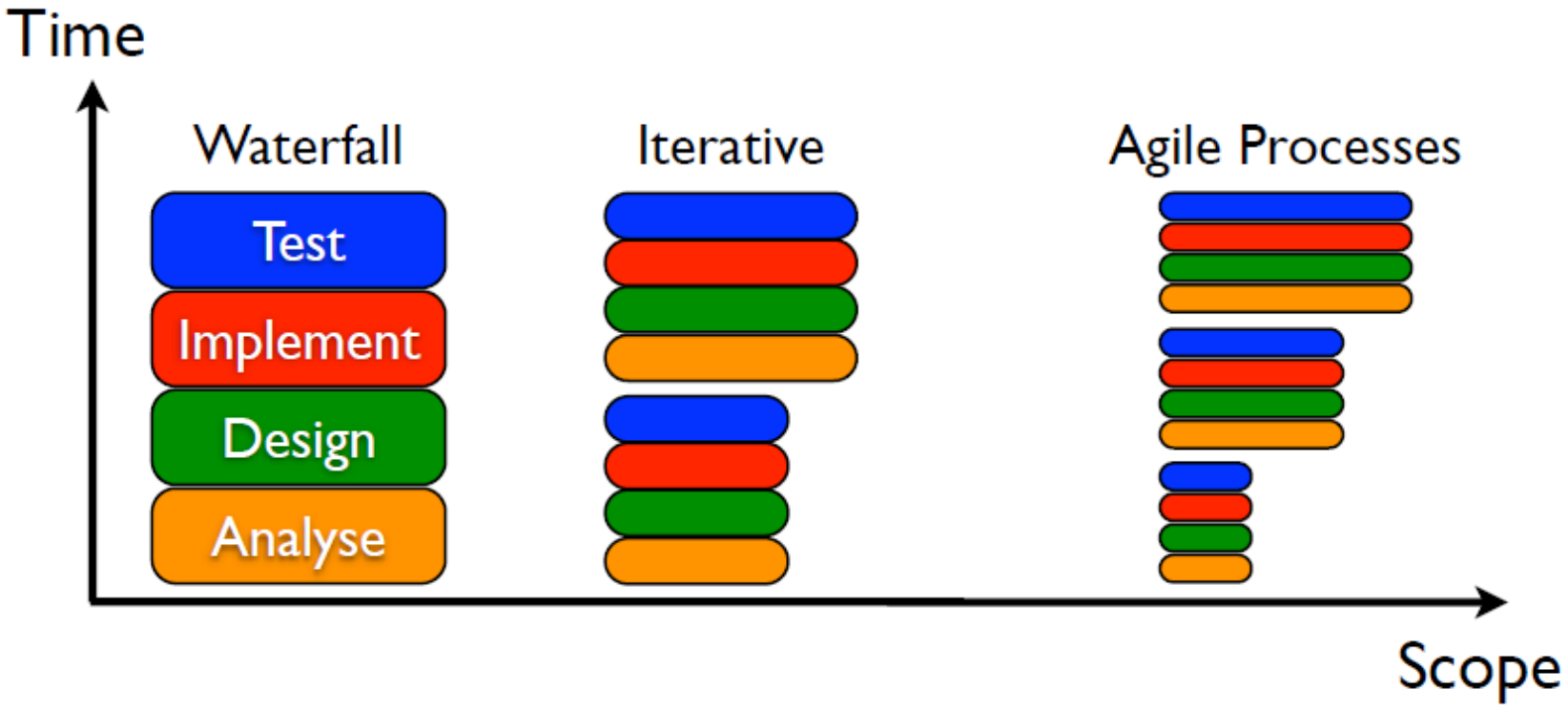
The Manifesto for Agile Software Development

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

AGILE MODELS

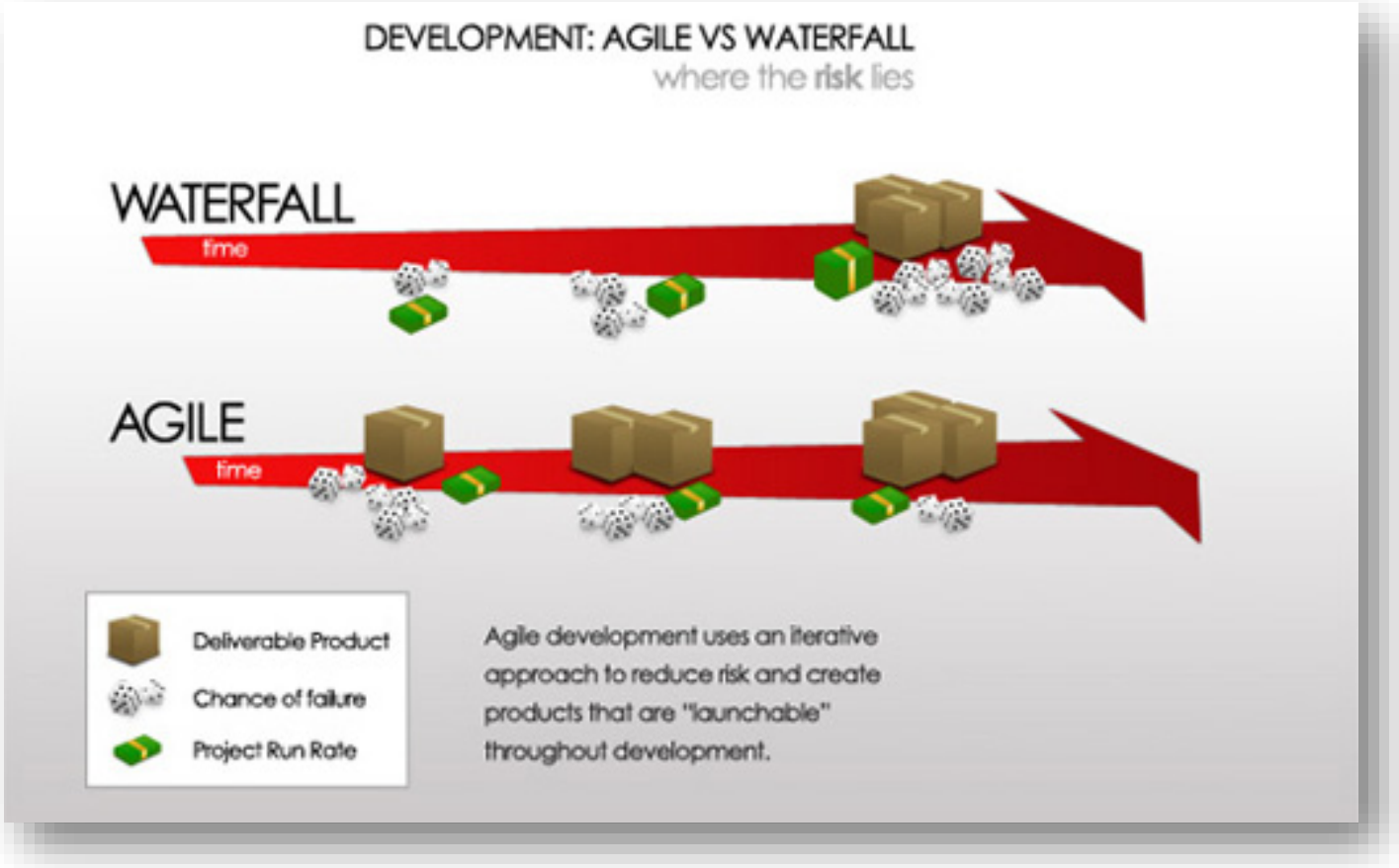


AGILE MODELS



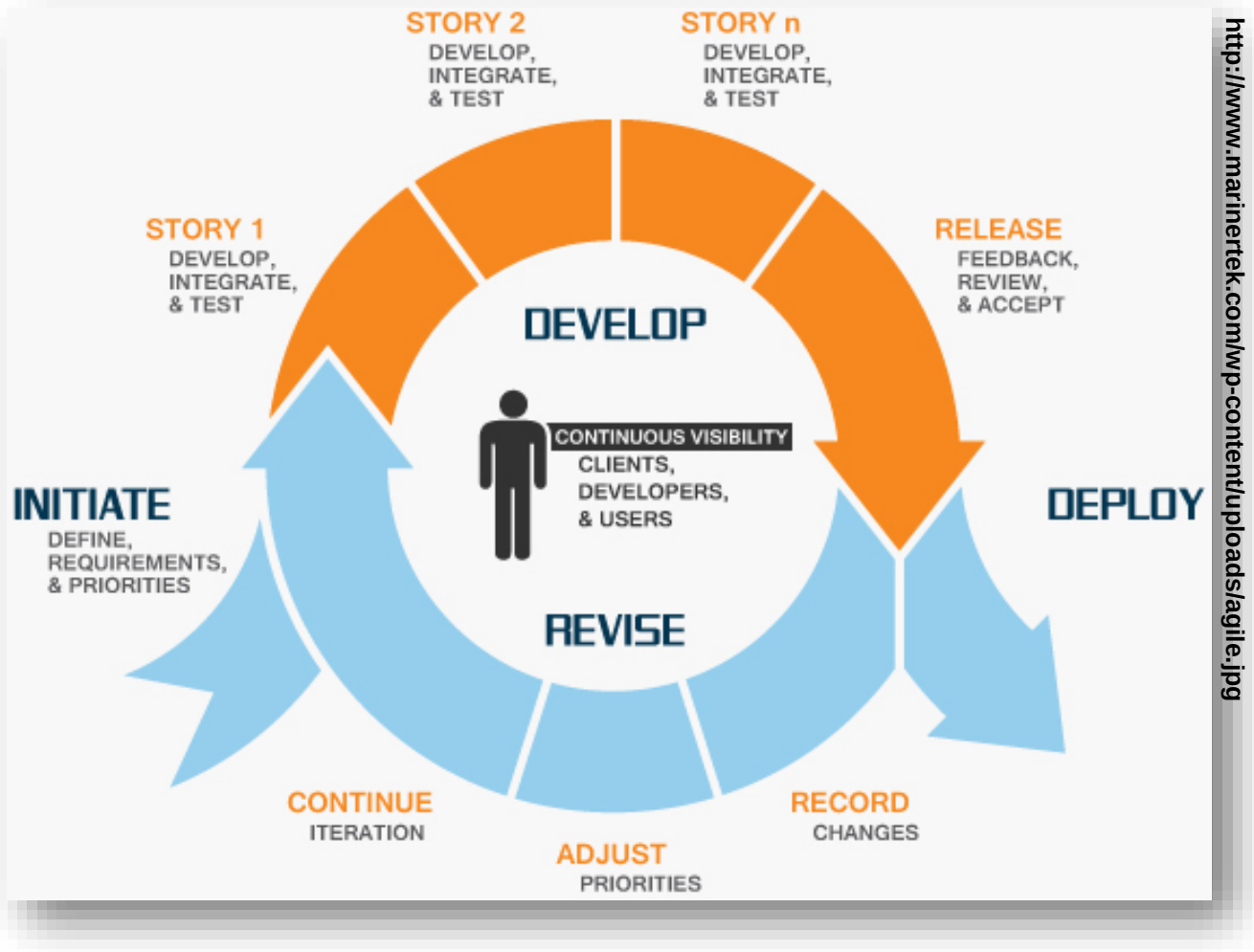
Taken from Andreas Zeller's Software Engineering notes

AGILE MODELS



Agile delivers early working prototype, small increments and user-centric. Waterfall late delivery, less user feedback and more documentation

AGILE MODELS



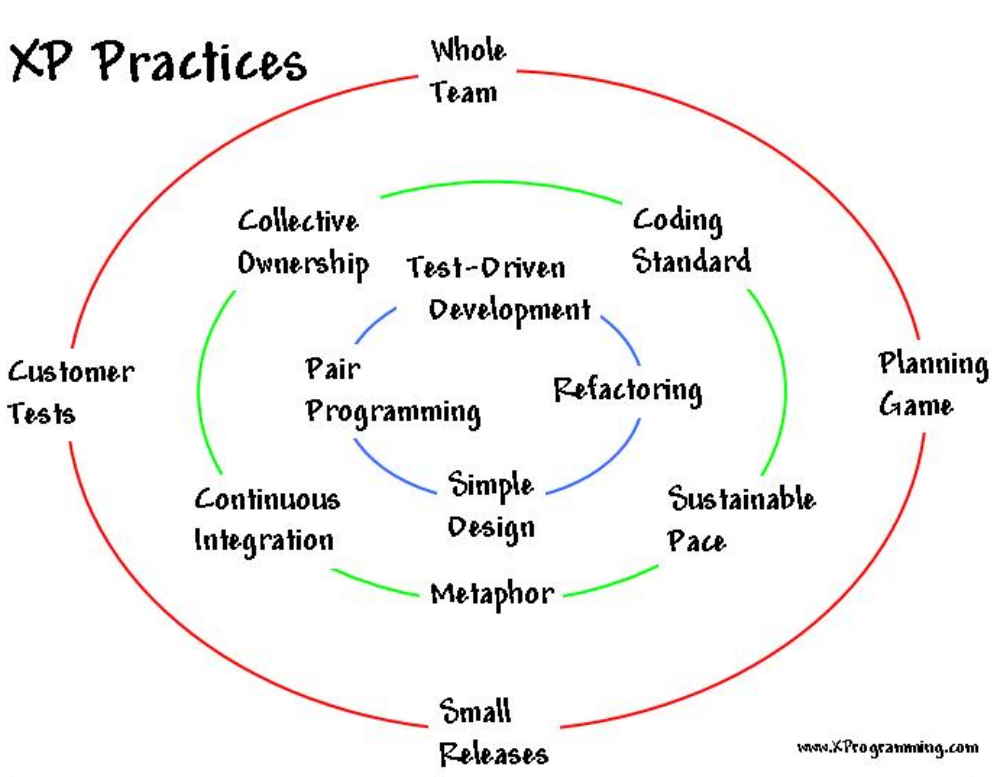
AGILE MODELS

There are many agile process models

- Extreme Programming (XP)
- Adaptive Software Development (ASD)
- Dynamic System Development Method (DSDM)
- Scrum
- Crystal
- Feature Driven Development (FDD)
- Agile Modeling (AM)

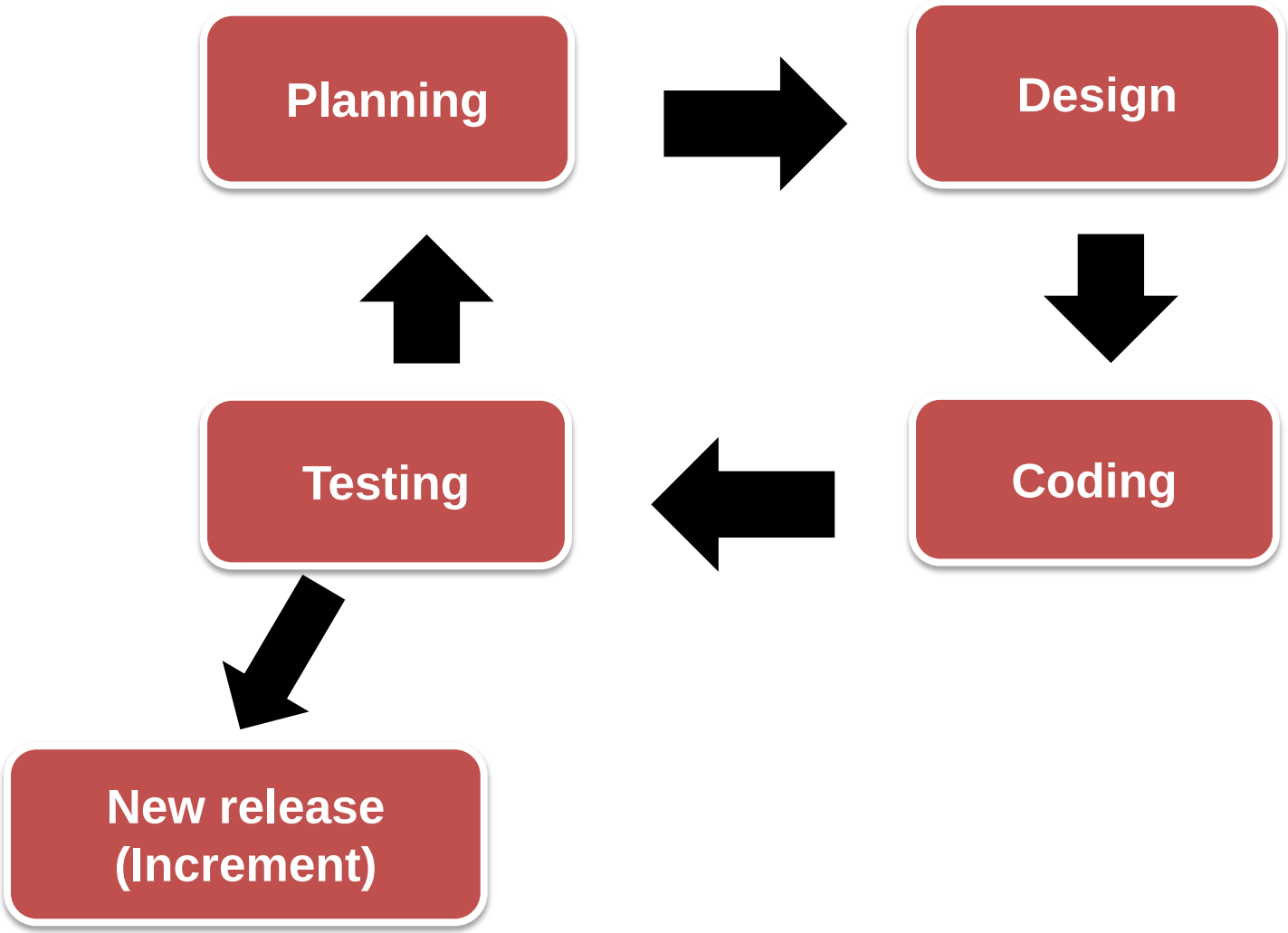
EXTREME PROGRAMMING: PRINCIPLES

- Plan to release in small increments
- Test first
- Keep it simple
- Own it collectively
- Code to standards
- Integrate continuously
- Refactor
- Program in pairs



EXTREME PROGRAMMING: ACTIVITIES

Queen Mary University of London



EXTREME PROGRAMMING: USER STORIES

Queen Mary University of London

In software development and product management, a user story is a description consisting of one or more sentences in the everyday or business language of the end user or user of a system that captures what a user does or needs to do as part of his or her job function.

- Project planning is based on the user stories
- The stakeholders work together to determine which stories will be released in each iteration

EXTREME PROGRAMMING: USER STORIES

Queen Mary University of London

Every team (company) has their own way

- "As a <*role*>, I can <*action with system*> so that <*external benefit*>"
- "As <persona> ,I want <what?> so that <why?>“

As a power user, I can specify files or folders to backup based on file size, date created and date modified.

What is the difference between use cases and user stories?

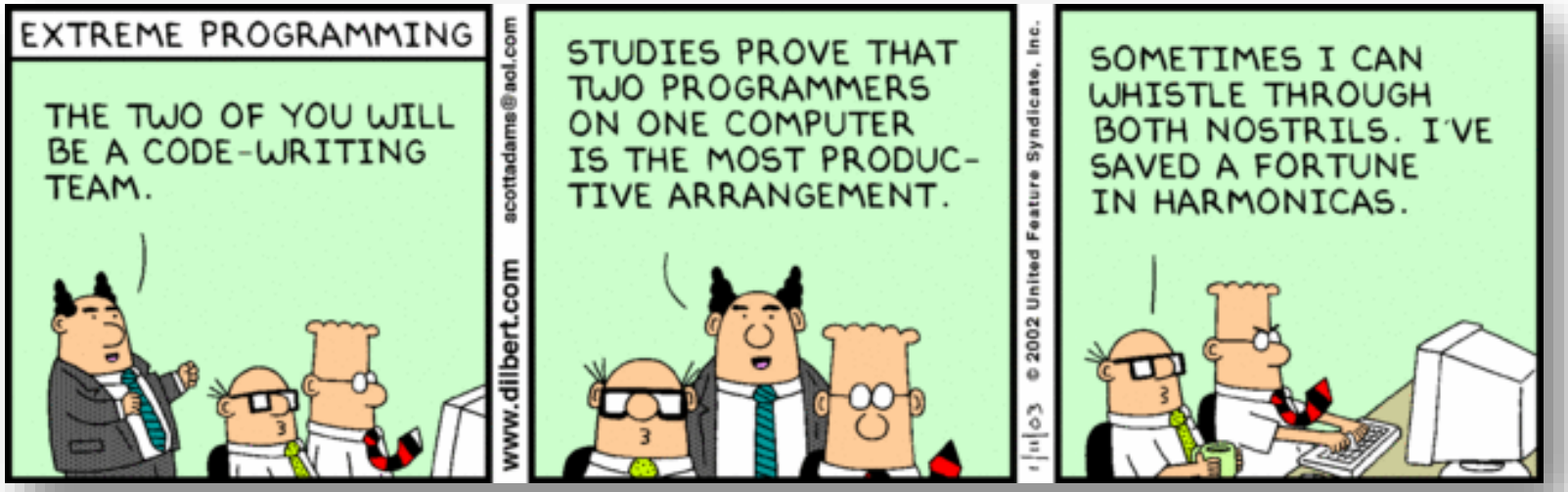
EXTREME PROGRAMMING: PLANNING

Queen Mary University of London

- Initial brief ‘prototype’ phase
- Quickly determine scope of next release
- Put simple system into production quickly, then release new versions on a short cycle.
- Keep meetings short but frequent
- Involve the customer throughout
- Don’t burn out
- Embrace change – it will happen anyway

EXTREME PROGRAMMING: CODING

- All production code written by two programmers at one machine
- Anyone can change any code anywhere in the system at any time
- All code written to agreed standard that emphasizes communication throughout



EXTREME PROGRAMMING: TESTING

Queen Mary University of London

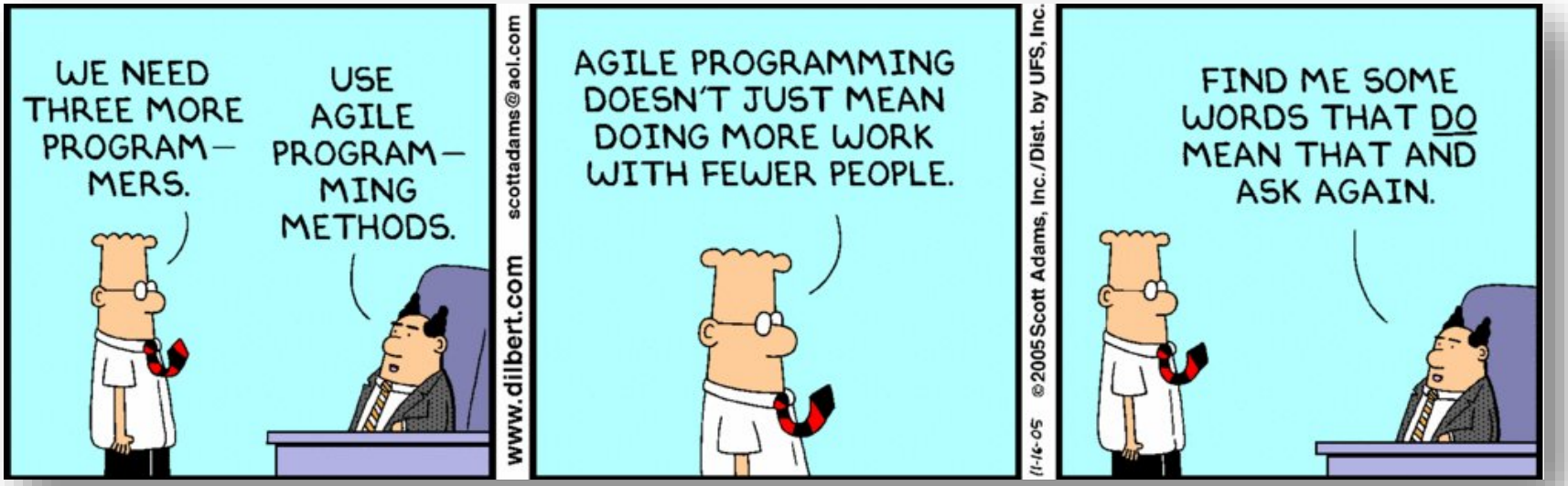
- Write unit tests for each method even before you start coding (Test driven development)
- Tests provide a definition and documentation of the required behaviour
- Integrate and build the system every time a task is completed

EXTREME PROGRAMMING: KEEPING IT SIMPLE

Queen Mary University of London

- System should be designed as simply as possible at any given moment
- Extra complexity is removed as soon as it is discovered
- Look for well known design patterns
- Refactor

EXTREME PROGRAMMING



...REAL SOFTWARE LIFE-CYCLE?

Queen Mary University of London

Requirements capture	Lone genius has bright idea
Coding	Friends drop by with more bright ideas
Test and trial	Someone in accounts wants to play with it
User documentation	Someone in accounts can't remember how to drive it
Architectural design	You must be joking
Detailed design	" "
Functional specification	'Manager wants to know what you've been doing for the past year'
Requirements definition	'Manager wants to know what xxxxx use it is'
Maintenance	Impossible

Queen Mary University of London

LESSON SUMMARY

- Some kind of defined life-cycle needed
- Classic waterfall is dead (or not?)
- Different life-cycle models contain common key activities.
- Prototyping and evolutionary development approaches allow more user feedback
- Extreme programming is suitable for your group project