WHIRLWIND RE-CAP OF SOFTWARE ENGINEERING

DR. VIVEK NALLUR

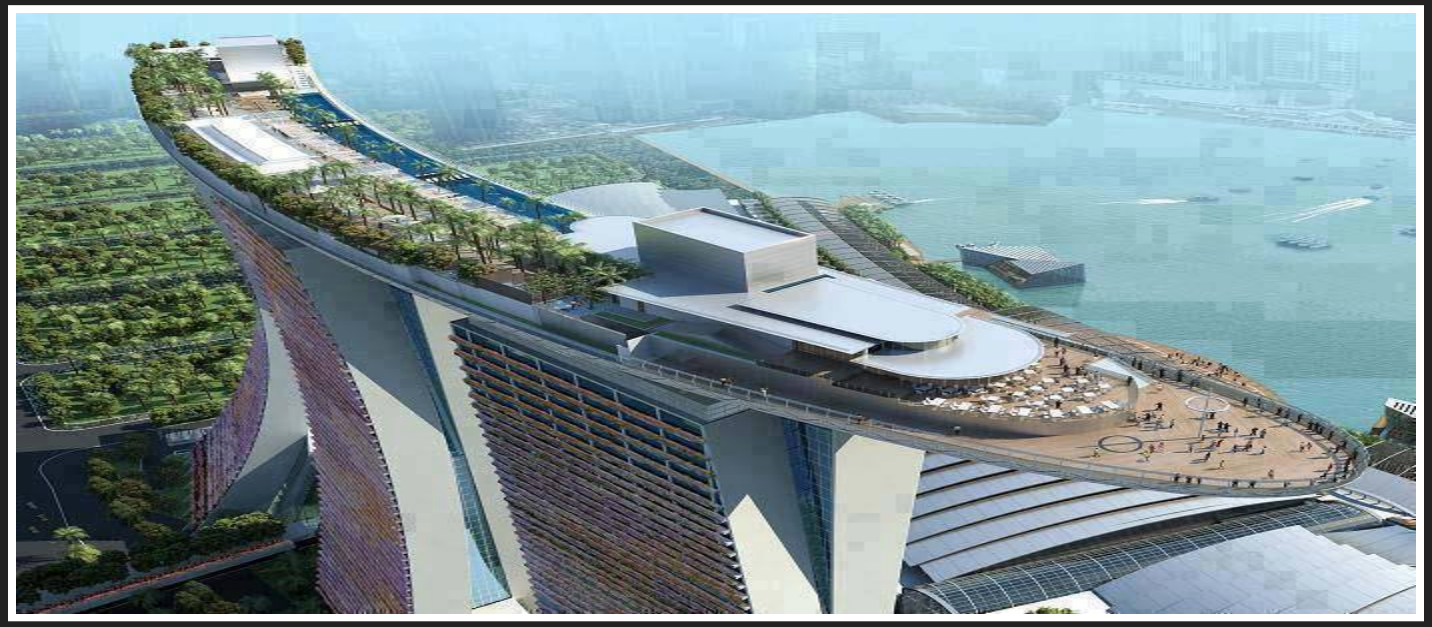
VIVEK.NALLUR@SCSS.TCD.IE

WHAT IS SOFTWARE ENGINEERING?

Engineering principles and practice that convert the art of programming into reliable software products/projects

IF YOU'RE ENGINEERING IT

- You need to know WHAT to build
- You need to know HOW to build it
- You need to build it
- You need to DOCUMENT what you're doing
- You need to do it on time!



WHAT IS A 'REQUIREMENT' ANYWAY?

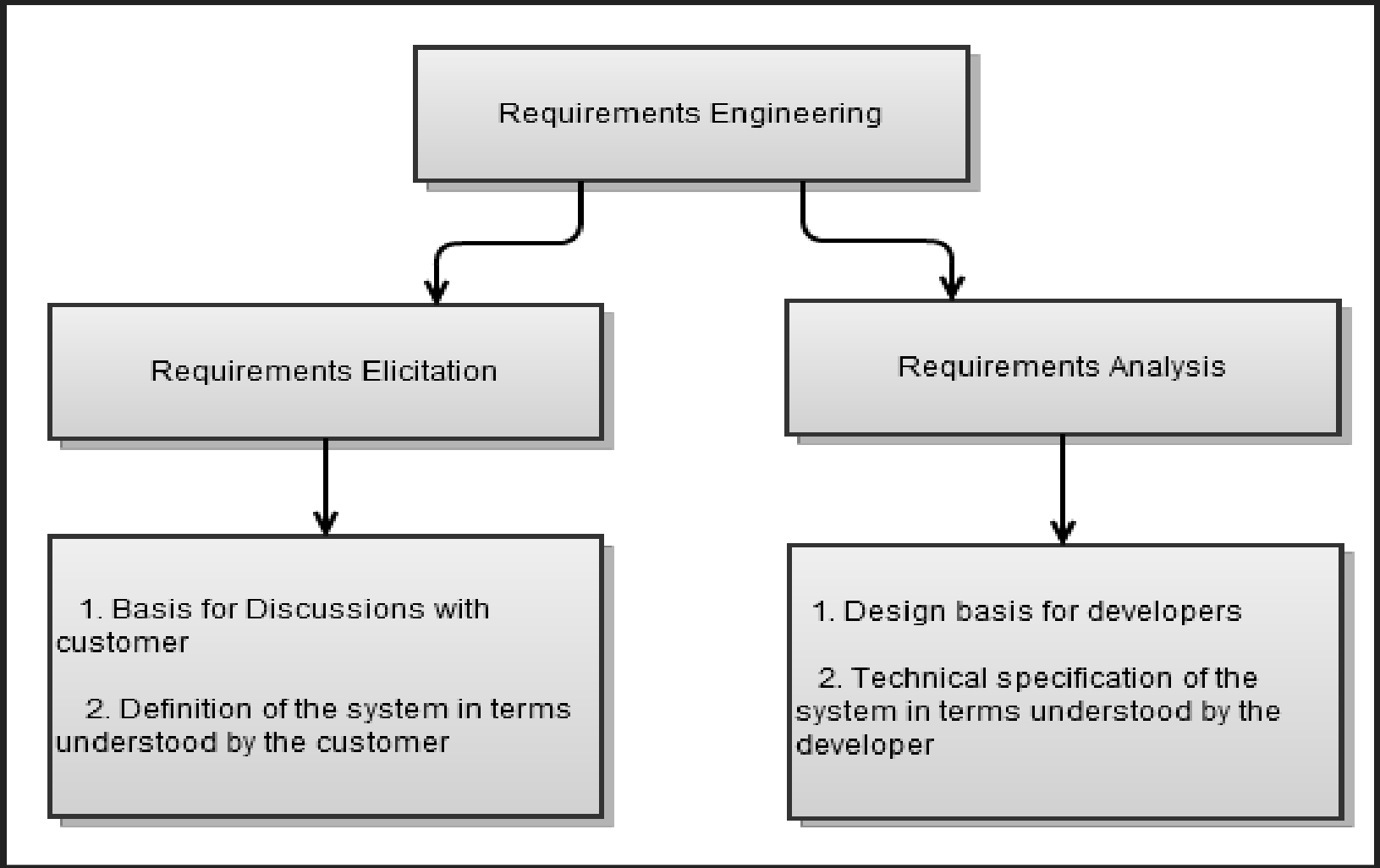
According to the *IEEE*:

- A condition or capability needed by a user to solve a problem or achieve an objective.
- A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formal document

Requirements cover not only the desired functionality of a system or software product, but also address:

- *Non-functional issues* (*e.g.*, performance, reliability, etc.)
- *Constraints on design* (*e.g.*, must operate with existing hardware/software)
- *Constraints on implementation* (*e.g.*, must be written in Java)

TWO DIFFERENT SET OF STAKEHOLDERS



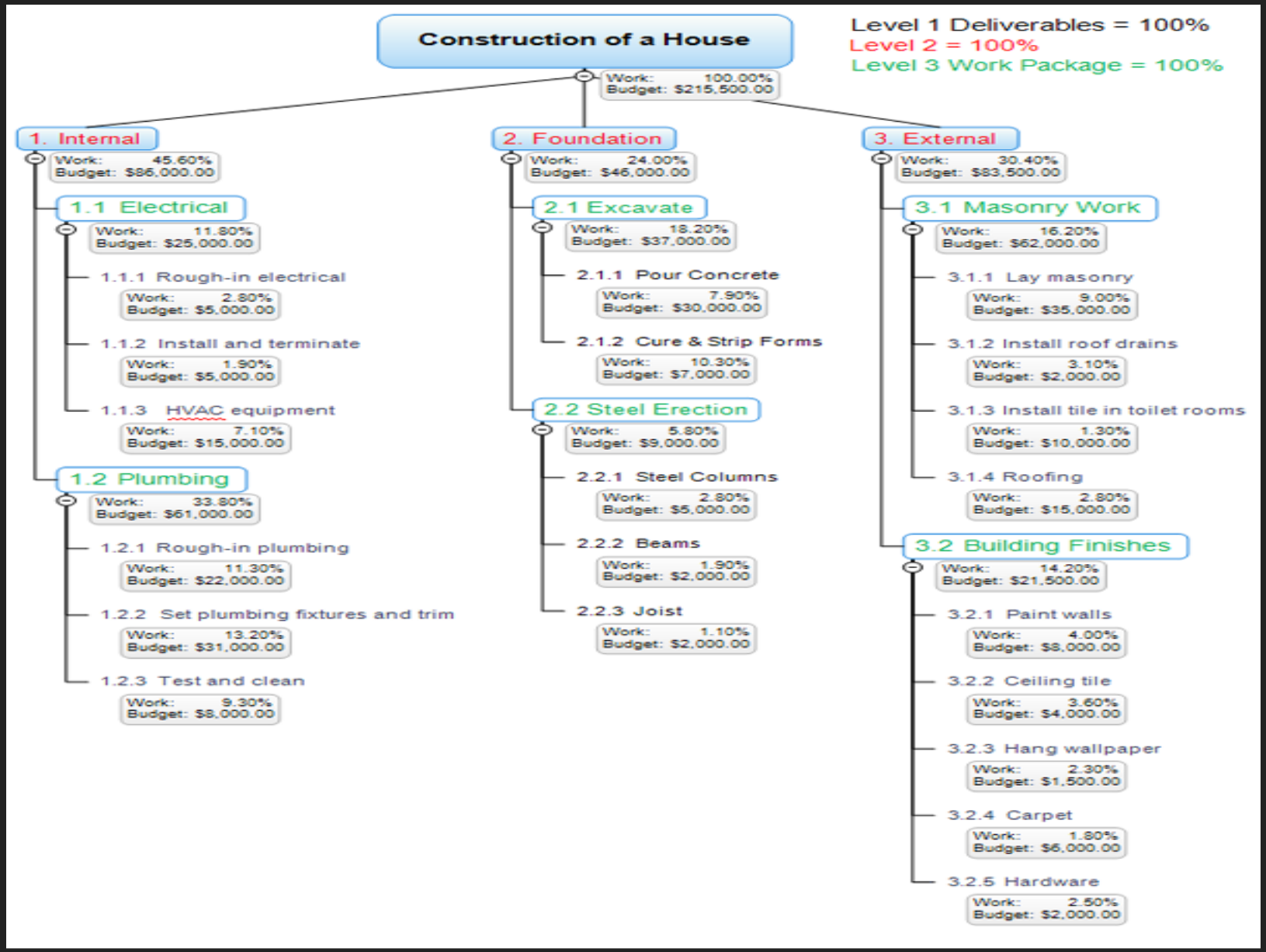
REQUIREMENTS OF REQUIREMENTS

- Clear
- Measurable
- Concise
- Feasible
- Necessary
- Prioritized

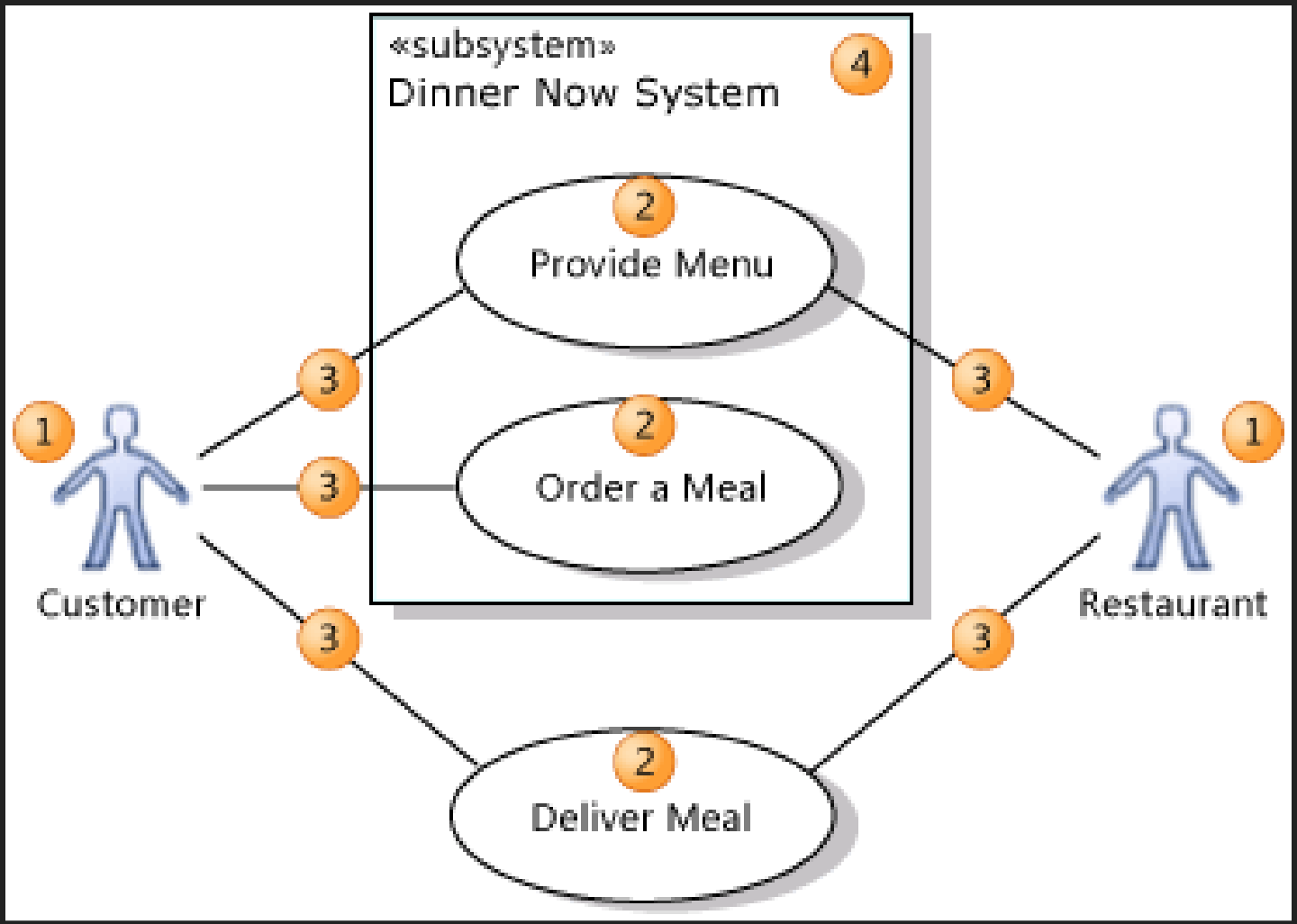
TECHNIQUES OF WRITING DOWN REQUIREMENTS

- Work-breakdown Structure
- Use cases
- Data Flow Diagrams, Activity Diagrams, Petri Nets, ...
- User stories

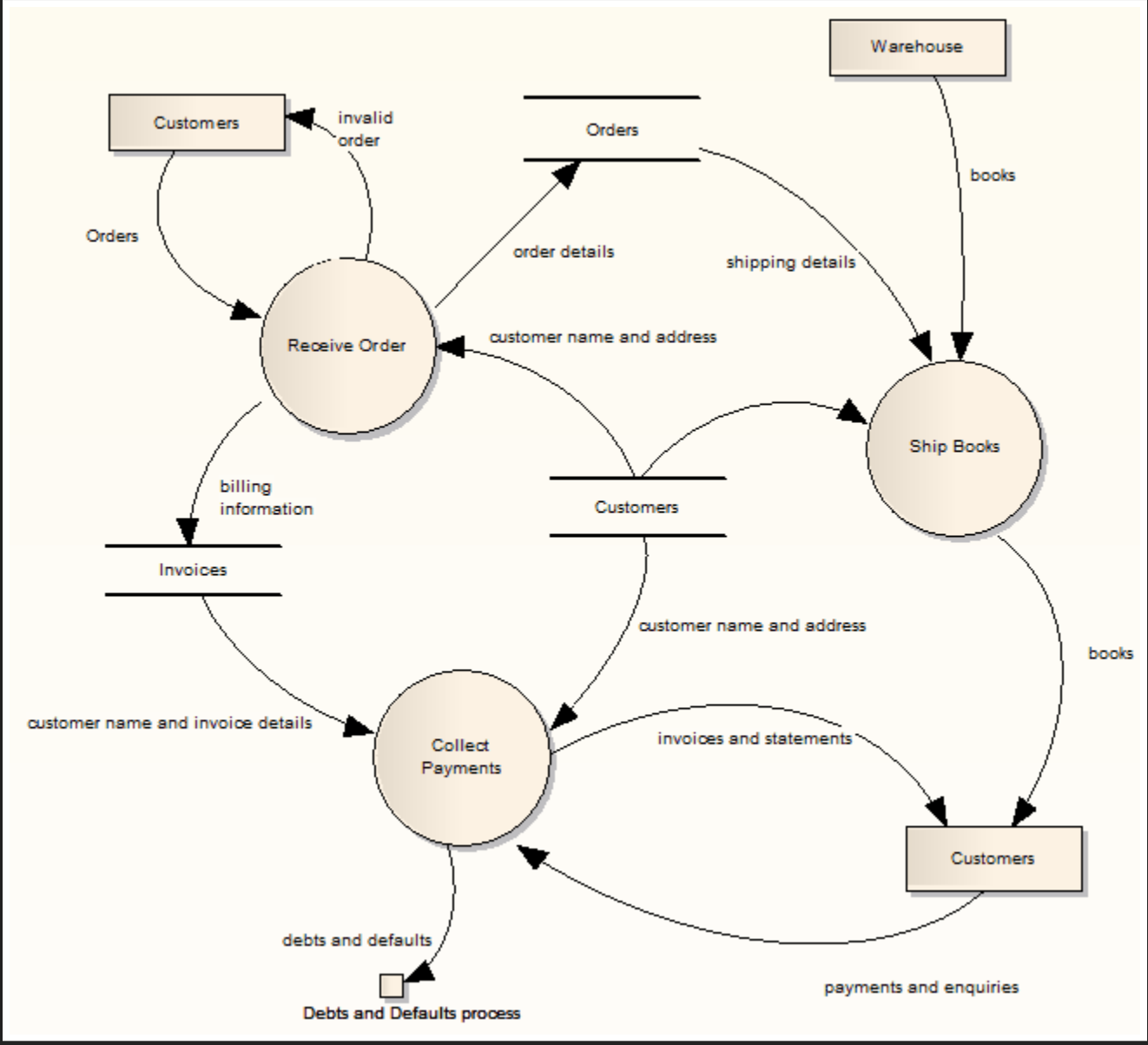
WORK-BREAKDOWN STRUCTURE



USE CASES


«subsystem»
Dinner Now System
4
2
Provide Menu
2
Order a Meal
2
Deliver Meal
1
Customer
1
Restaurant
3
3
3
3
3

DATA FLOW DIAGRAMS



WHAT IS AN ESTIMATE?

A tentative evaluation or rough calculation

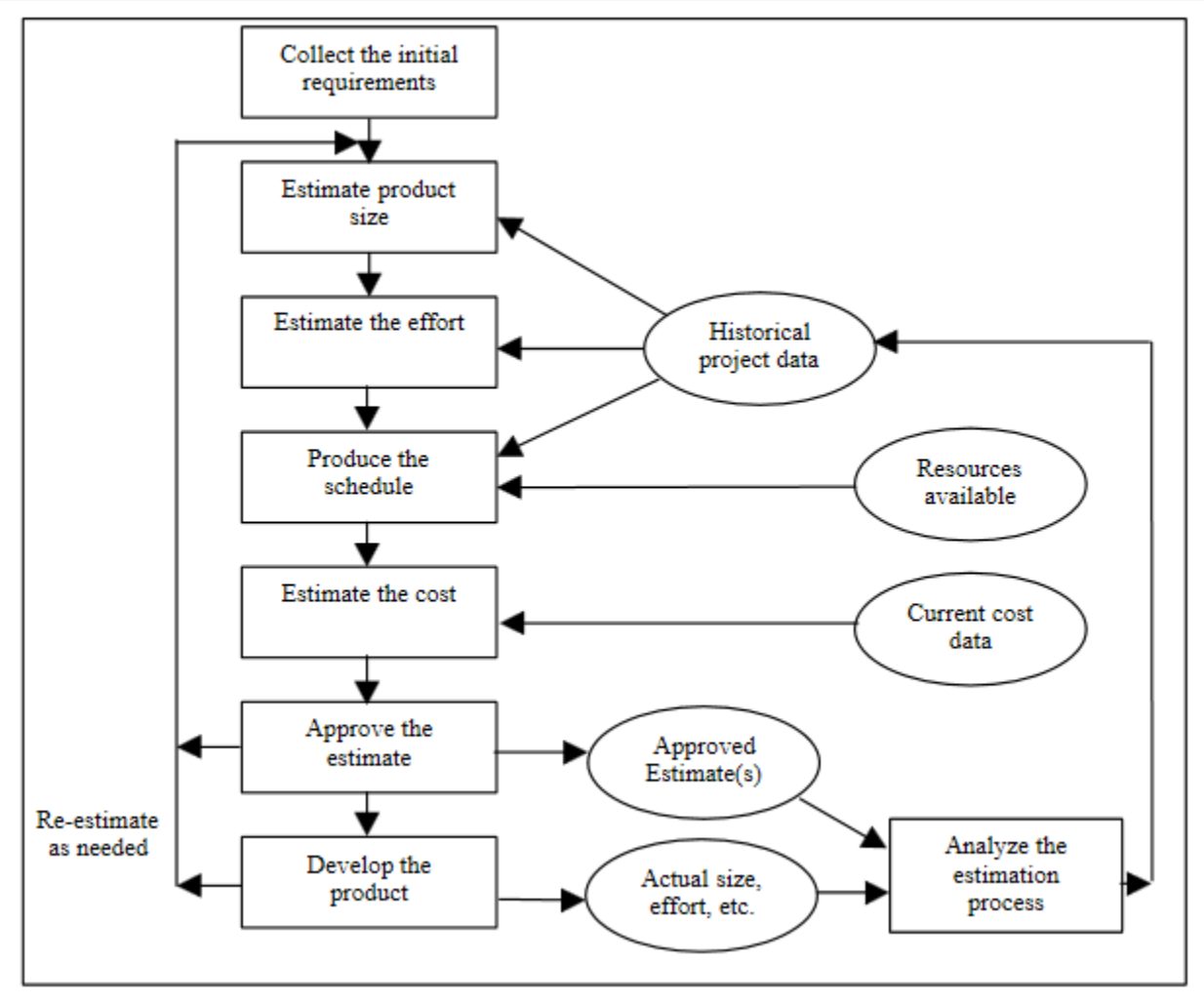
A preliminary calculation of the cost of a project

A judgement based on one's impressions; opinion

A GOOD ESTIMATE

A good estimate is an estimate that provides a clear enough view of the project reality to allow the project leadership to make good decisions about how to control the project to hit its targets

BASIC ESTIMATION PROCESS



OTHER ESTIMATION PROCEDURES

- COCOMO & COCOMO-II
- Function Points
- Wideband Delphi

RECOMMENDED ESTIMATION PROCESS

- Use multiple types of estimation approaches
- Use experts to converge on a range with probability distribution
- Revise at least twice, as more detail is available

WHAT IS ARCHITECTURE?

> *Fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.*

ISO/IEC/IEEE 42010

HOW DOES IT RELATE TO DESIGN?

> *All architecture is design but not all design is architecture. Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change.*

Grady Booch in *Pattern-Oriented Software Architecture, On Patterns and Pattern Languages*

SIGNIFICANT COST OF CHANGE

- Security
- Scalability
- Availability
- Privacy

WHAT IS AN ARCHITECTURAL STYLE?

> *A family of systems in terms of a pattern of structural organization. More specifically, a style determines the vocabulary of components and connectors that can be combined.*

David Garlan and Mary Shaw, *An Introduction to Software Architecture, 1996*

CATEGORIZATION

Category	Architectural Styles
Communication	Service-Oriented, Message Bus, ...
Deployment	Client/Server, N-Tier, 3-Tier, ...
Domain	Domain driven design
Structure	Component-based, Object-oriented, Layered, ...

KEY PRINCIPLES OF ALL STYLES

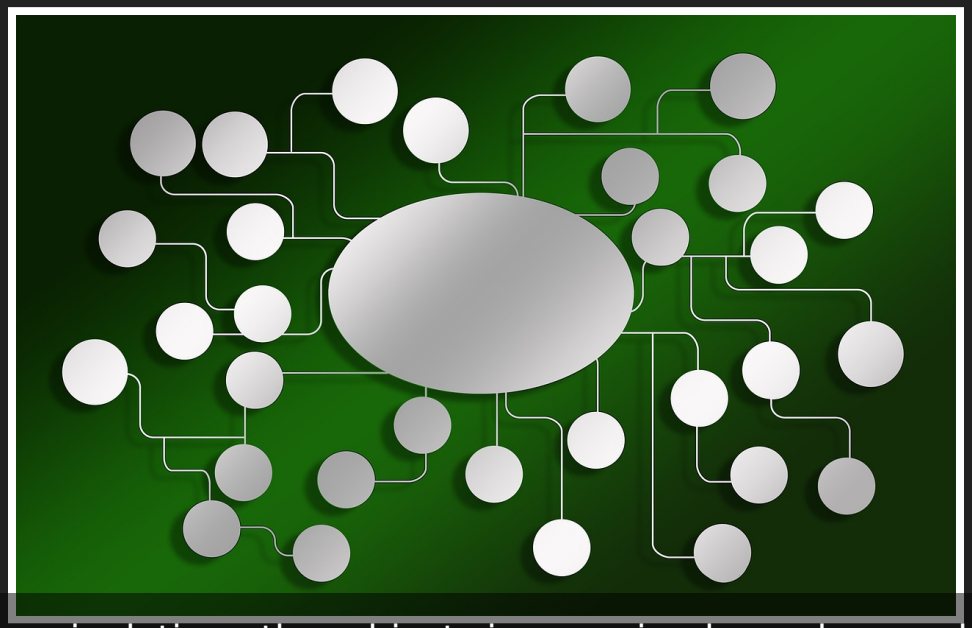
Separation-of-concerns: Functionality should not be spread amongst components unnecessarily

Single Responsibility: Components should do only one thing, and do it well

Principle of least knowledge: Talk to the interface!

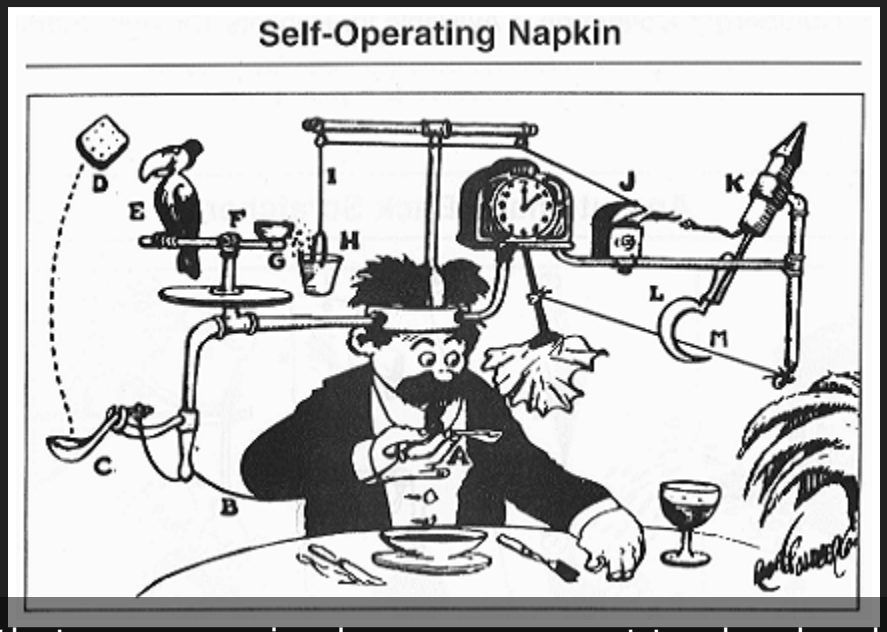
OBJECT-ORIENTED DESIGN

Object-oriented programs work by making objects collaborate with each other, manipulating state and behaviour, in some particular order.



While the concept of objects manipulating other objects is very simple, and grounded in everyday experience, communicating this is very hard

COMMUNICATING DESIGN IS HARD



One reason for this hardness is that programming languages are at too low-level abstraction to communicate fundamental ideas. Like trying to design a car, while talking about the individual nuts and bolts.

UML: UNIFIED MODELLING LANGUAGE

A common (mostly diagrammatic) language to describe a system:

- Structure
- Behaviour

There have been many object-oriented modelling languages in the past. Through a process of software evolution, UML has emerged as the de-facto standard modelling language used to describe object-oriented design

CLASS DIAGRAMS

The most commonly used structural view is *class*

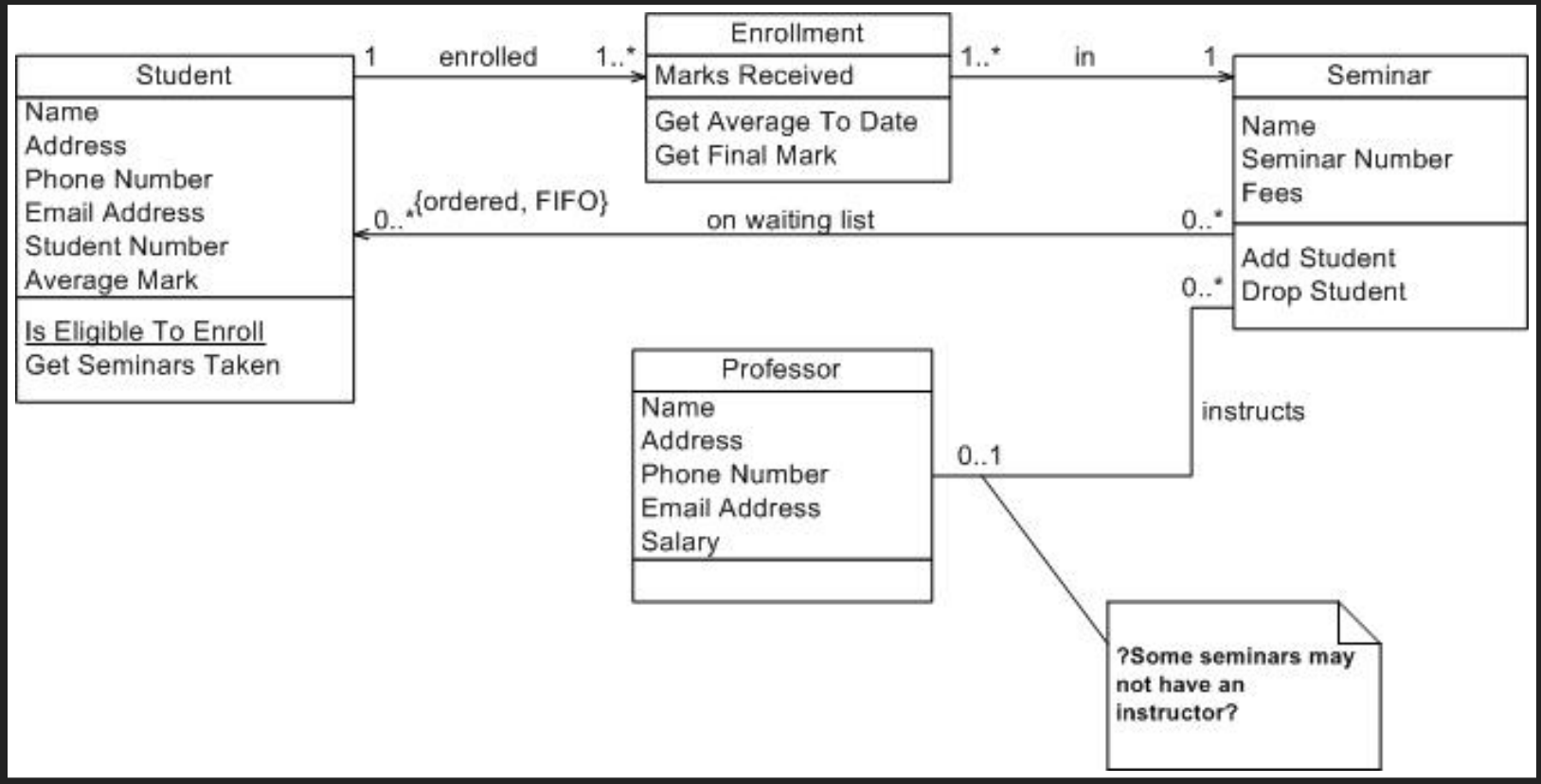


Image source: Agile Modelling

HOW TO CREATE A CONCEPTUAL CLASS DIAGRAM

- List all the classes in your domain (person, place, thing, concept, event, screen, report, ...)
- Each class is typically modelled as a rectangle with three sections:
  - Name
  - Attributes
  - Methods
- Who/what does it associate with?

WHAT IS AN SEQUENCE DIAGRAM?

- Most common form of documenting interactions in UML
- Typically captures the behaviour of a single scenario/usecase
- Shows participating objects and messages passed between them

NOTION OF DESIGN PATTERNS

- Evolved from 'real building terminology'
- Allows certain elegant solutions, in the presence of:
  - Common problems
  - Common tools
  - Common forces of change

COMMON DESIGN PATTERNS

- Strategy Pattern
- Observer Pattern
- Decorator Pattern
- Adapter Pattern
- ... (many more)

CHARACTERISTICS OF USABILITY

Effective	Completeness and accuracy with which users achieve their goals
Efficient	Speed with which users complete their tasks
Error Tolerant	Ability of an interface to prevent errors or recover from errors

IMPORTANCE OF USABILITY

- Usability makes the user come back to the system
- Users are choosier than ever before
- User-perception can make or break your business

MEASURING USABILITY

- *Task Time*: Measures efficiency, ease of learning, mental processing time
- *Number of Problems*: Think-aloud and record list of problems.
- *Emotional state of users*: If the user is upset, something is wrong!

SOFTWARE TESTING

Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end-user

WHAT CAN TESTING SHOW?

- Errors
- Conformance with requirements
- Performance
- An indication of quality

OPACITY OF TESTING

Depending on how much visibility into the code is available to the tester:

- *Whitebox Testing*: The tester can see all the code, and tries to test all possible paths through program logic
- *Blackbox Testing*: The tester only knows what the expected outputs are, to a particular set of inputs

SCALE OF TESTING

We begin by *testing-in-the-small* and move towards *testing-in-the-large*

- Unit Test (Class)
  - Data structures
  - Boundary conditions
  - Independent paths
  - Error-handling paths

Opacity: Whitebox

TESTING-IN-THE-LARGE

- Integration Testing
- System Testing
- Acceptance Testing

OTHER TESTING

- *Regression Testing*: Selective re-testing of a system/component to verify modifications have not caused unintended effects
- *Beta Testing*: Ad-hoc testing by potential users in a production environment.
  Not very systematic, but can reveal unexpected errors

WHAT IS A SOFTWARE DEVELOPMENT PROCESS?

Also known as Software Lifecycle, the development process is a structure imposed on the development of a software product/project.

ACTIVITIES

Requirements	Extracting and specifying the requirements of the desired software
Architecture	An abstract representation of the structural and communication patterns of the software
Implementation	Translating the design into code
Testing	Verification and validation of the software
Maintenance	Fixing discovered bugs, enhancing the software for new requirements

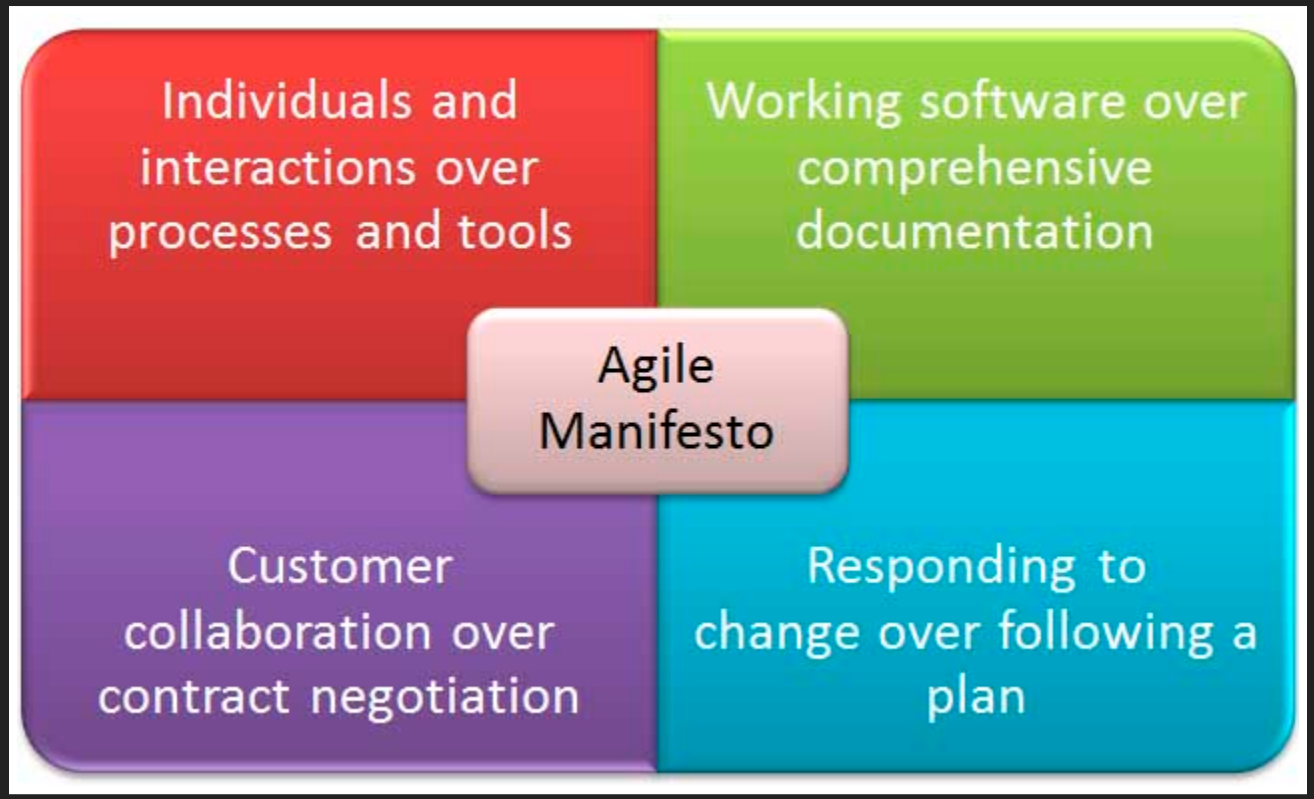
WATERFALL: CHARACTERISTICS

- *Sequential*: Each phase *must* be completed before the next phase can begin
- *Easy*: Ease of understanding, use, and management
- *Well-specified*: Each phase has specific deliverables, which can be reviewed for completeness

WATERFALL: PROS / CONS

Works well when well-specified	Can't go back from Testing to Architecture/Design
Easy to implement	Difficult to deal with ambiguity
Works for short-projects	Long, on-going projects not well-suited

AGILE MANIFESTO



AGILE: CHARACTERISTICS

- Iterative and incremental development
- Self-organizing, cross-functional teams
- Adaptive velocity-based planning
- Repeatable, time-boxed iterations

WHY DO SOFTWARE PROJECTS FAIL?

- *Bad Software Engineering*
  - Late
  - Over-budget
  - Incomplete functionality, bugs, performance ...
- *Environmental Changes*
  - Markets change
  - Technological obsolescence
  - Change in management

CLASSIFICATION OF RISK

A software risk can be classified into:

- *Internal Risk*: Under the control of the project manager
  - Resources available
  - Technological in-experience
- *External Risk*: Beyond the control of the project manager
  - Change of management
  - Change of markets

RISK MANAGEMENT

Processes Involved	What it does
Risk Identification	List precisely all risk events possible in a project
Risk Analysis	Define probability of occurrence, potential for loss
Risk Planning	Preventive measures to reduce likelihood or impact of risk event
Risk Monitoring	Match measures to re-calculated probabilities

STRATEGIC DECISIONS

Big software projects are strategically important for organizations

Incumbent upon senior management to understand the costs, options, risks, and strategic implications of software projects

Required: Someone who is familiar with both, computing aspects of the project and the strategic direction of the company

BUSINESS DECISIONS WITH REGARD TO DEVELOPMENT

- Software Customization
  - Bespoke software
  - Off-the-shelf packages
  - Packages with modifications
- Development Team
  - In-house
  - Out-sourced
  - Retail

LEGAL ASPECTS

- Retail / Commercial licenses
- Open-source licenses

Even if software is free, packaging/distribution/customizations/other-services can be a fairly good business. E.g., *RedHat*, *Canonical*, etc.

OBLIGATIONS

- Ethical Obligation
  - If a software house creating bespoke software, inform clients about possible effects of licenses of software you use.
  - E.g., GPL v3 is aggresively viral. BSD is not
- Legal Obligation
  - Packaged Software (either as binary or incl. src code): Full disclosure about licenses of sub-components used

CLOSING COMMENTS

- Final (written) exam in summer
- Sample paper will be available in January

THAT'S ALL, FOLKS!

Questions? Comments?