DESIGN

as part of
SDLC
2022 W - April 2023

slides are taken from the textbook's web source and Internet for class purposes only

SW DESIGN

The Process

The Principles

Testing - The Big Picture

Design - The Process

- *Design* is a problem-solving process whose objective is to find and describe a way:
  - To implement the system's *functional requirements*...
  - While respecting the constraints imposed by the *non-functional requirements*...
    - including the budget
  - And while adhering to general principles of *good quality*

SW Design is actually Design of Design!

Design - A series of decisions

A designer is faced with a series of *design issues*

- These are sub-problems of the overall design problem.
- Each issue normally has several alternative solutions:
  - design *options*.
- The designer makes a *design decision* to resolve each issue.
  - This process involves choosing the best option from among the alternatives.

Making Decisions

The SW Designer needs to know about

— the requirements

— the design if done already

— the technology available

— software design principles and ‘best practices’

— what has worked well in the past

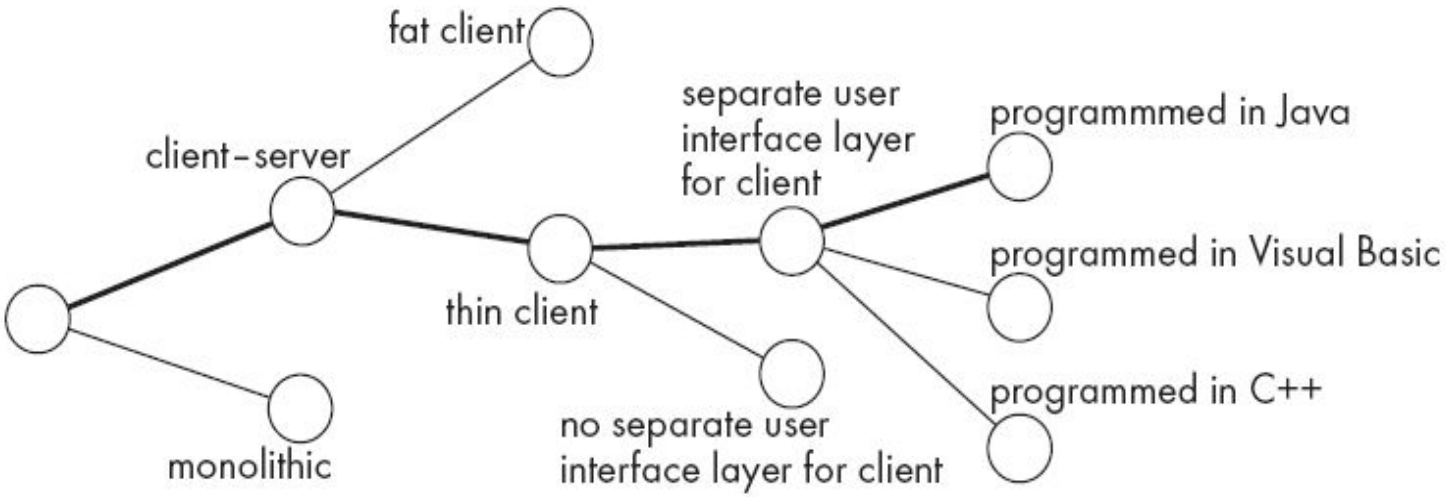
It was SAD to begin with - The difference?

Software Analysis has as output smaller problems to solve.

Design is a decision making process to find the best solution.

Design space

The space of possible designs that could be achieved by choosing different sets of alternatives is often called the *design space*



Some Terms *just like that!*

System Subsystem

Module Component

Package

Library Framework Pattern

Component

Any piece of software or hardware that has a clear role.

- A component can be isolated, allowing you to replace it with a different component that has equivalent functionality.
- Most components are designed to be reusable.
- Some perform special-purpose functions.

Module

A component that is defined at the programming language level

- For example, methods, classes and packages are modules in Java.
- Module Owners

System

A logical entity, having a set of definable responsibilities or objectives, and consisting of hardware, software or both.

- A system can have a specification which is then implemented by a collection of components.
- A system continues to exist, even if its components are changed or replaced.
- The goal of requirements analysis is to determine the responsibilities of a system.

System to Sub Systems

- **Sub System is a System**
- A system that is part of a larger system, and which has a definite interface

What about?

System Subsystem

Module Component

Pattern

Package vs Library

Library vs Framework [Hollywood Principle]

Design

— Bottom Up

— Top Down

Top-down design

- First design the very high level structure of the system.
- Then gradually work down to detailed decisions about low-level constructs.
- Finally arrive at detailed decisions such as:
  - the format of particular data items;
  - the individual algorithms that will be used.

Bottom-up design

- Make decisions about reusable low-level utilities.
- Then decide how these will be put together to create high-level constructs.

Which is better?

Programming Assignment vs SWE Project

SGPA vs Job
Job vs Career

And, Design is not just one; but has many aspects

Different aspects of design

- *Architecture design*: [Systems and Application Courses]
  - The division into subsystems and components,
    - How these will be connected.
    - How they will interact.
    - Their interfaces.
- *Class design*: [Object Oriented Courses]
  - The various features of classes.
- ***User interface design*** **[App Design]**
- *Algorithm design*: [DSA and DAA]
  - The design of computational mechanisms.
- *Protocol design*: [Networks, Communication]
  - The design of communications protocol.

SW DESIGN

The Process

The Principles

The Techniques

The Difficulties and Risks

Principles Leading to Good Design
From the Text

Goals of Good Design

- Reducing cost
- Avoid delay
- Conform with the requirements
- Increasing qualities such as
  - Usability
  - Efficiency
  - Reliability
  - **Maintainability [Process and Documents]**
  - Reusability

Let us code, why bother to “design”?

May be, the design is more focused / actually ensures meeting the non-functional requirements and taking the discipline / industry forward in a professional way.

If not for design, this will never occur!

Design Principle 1: Divide and Conquer

- Big into small parts.
- Separate people can work on each part.
- Specialization areas for Software Engineers
- Easier to understand.
- Parts can be replaced/changed/reused.

- Being a

Ways of dividing a software system

- Distributed System – Clients and Servers
- A system is divided up into subsystems
- A subsystem can be divided up into one or more packages
- A package is divided up into classes
- A class is divided up into methods

D&C in Algorithm Design

Design Principle 2: Increase cohesion where possible

A subsystem or module has high cohesion if it keeps together things that are related to each other, and keeps out other things

- The system is easier to understand and change
- Type of cohesion:

Functional cohesion

When all the code that computes a particular result is kept together - and everything else is kept out

- when a module only performs a *single* computation, and returns a result, *without having side-effects*.
- Benefits to the system:
  - Easier to understand
  - More reusable
  - Easier to replace
- Modules that update a database, create a new file or interact with the user are not functionally cohesive

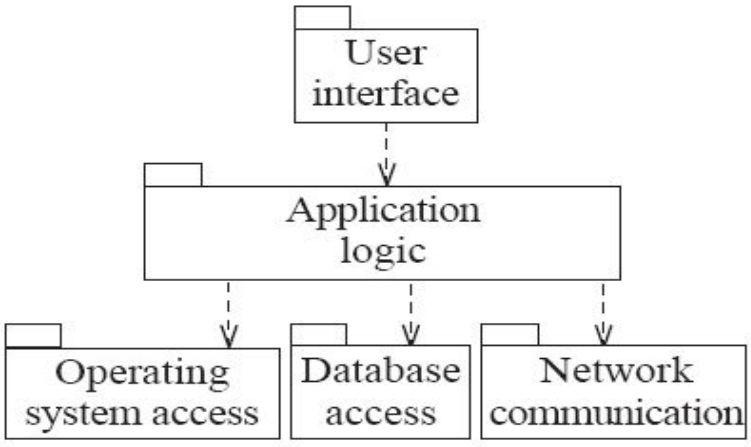
Layer cohesion

All the facilities for providing or accessing a set of related services are kept together, and everything else is kept out

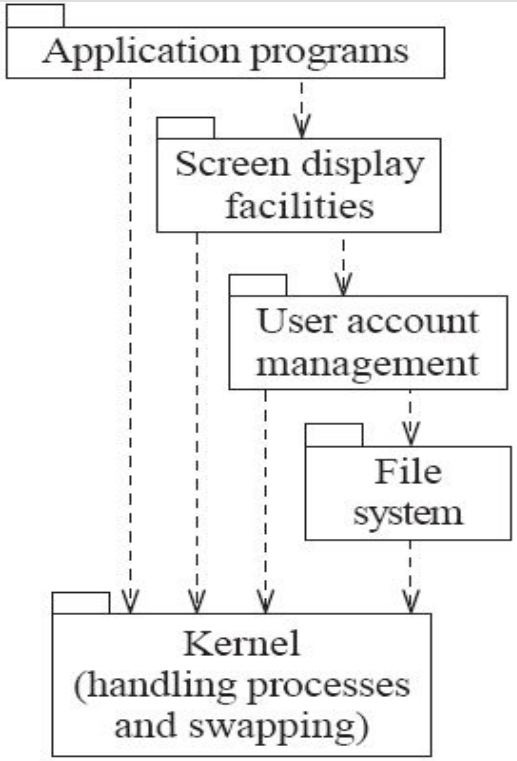
- The layers should form a hierarchy
  - Higher layers can access services of lower layers,
  - Lower layers do not access higher layers
- The set of procedures through which a layer provides its services is the *application programming interface (API)*

- UI, API, SDK

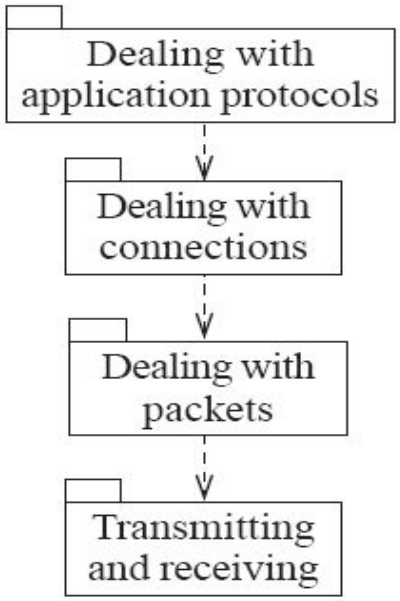
Example of the use of layers



(a) Typical layers in an application program

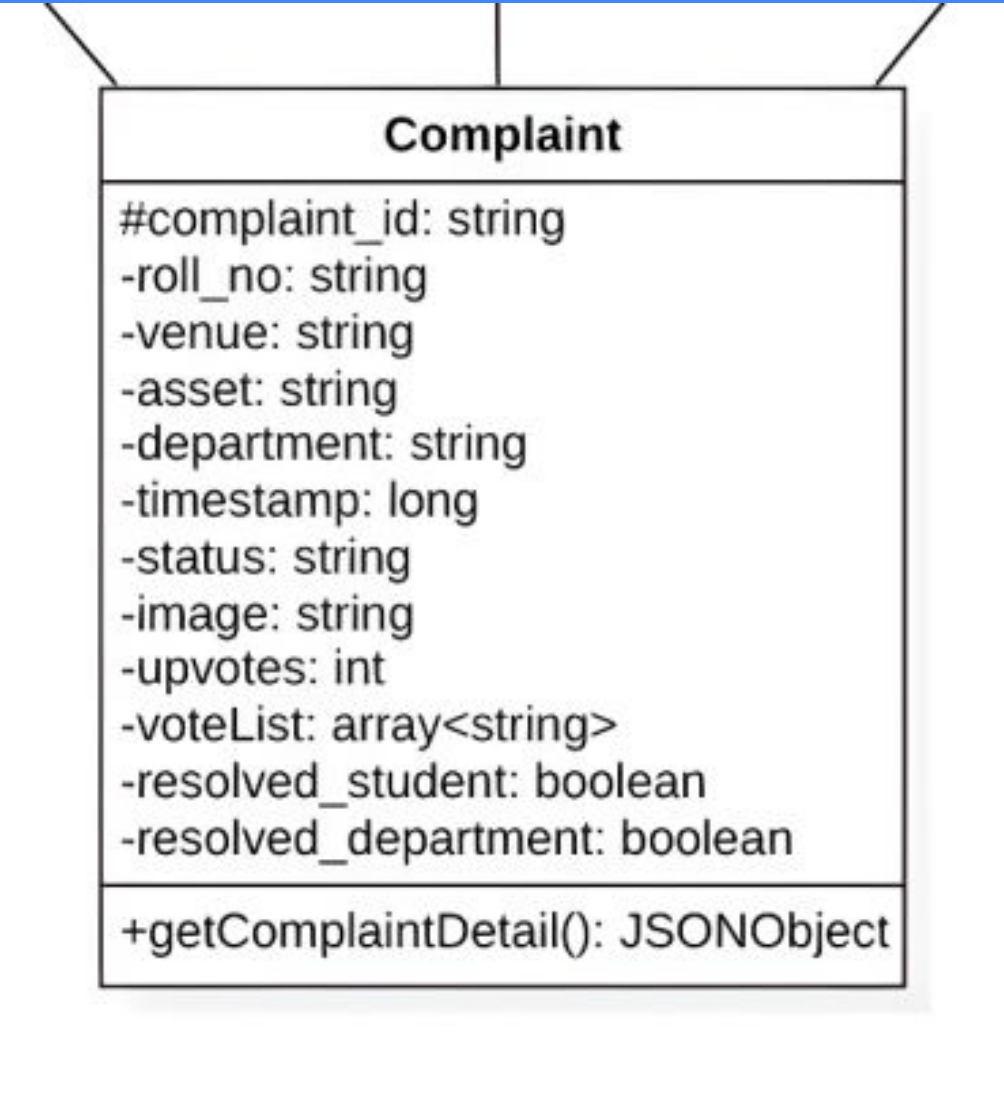


(b) Typical layers in an operating system



(c) Simplified view of layers in a communication system

In the context of OO software development, cohesion means relatedness of the public functionality of a class



Cohesion is a positive aspect

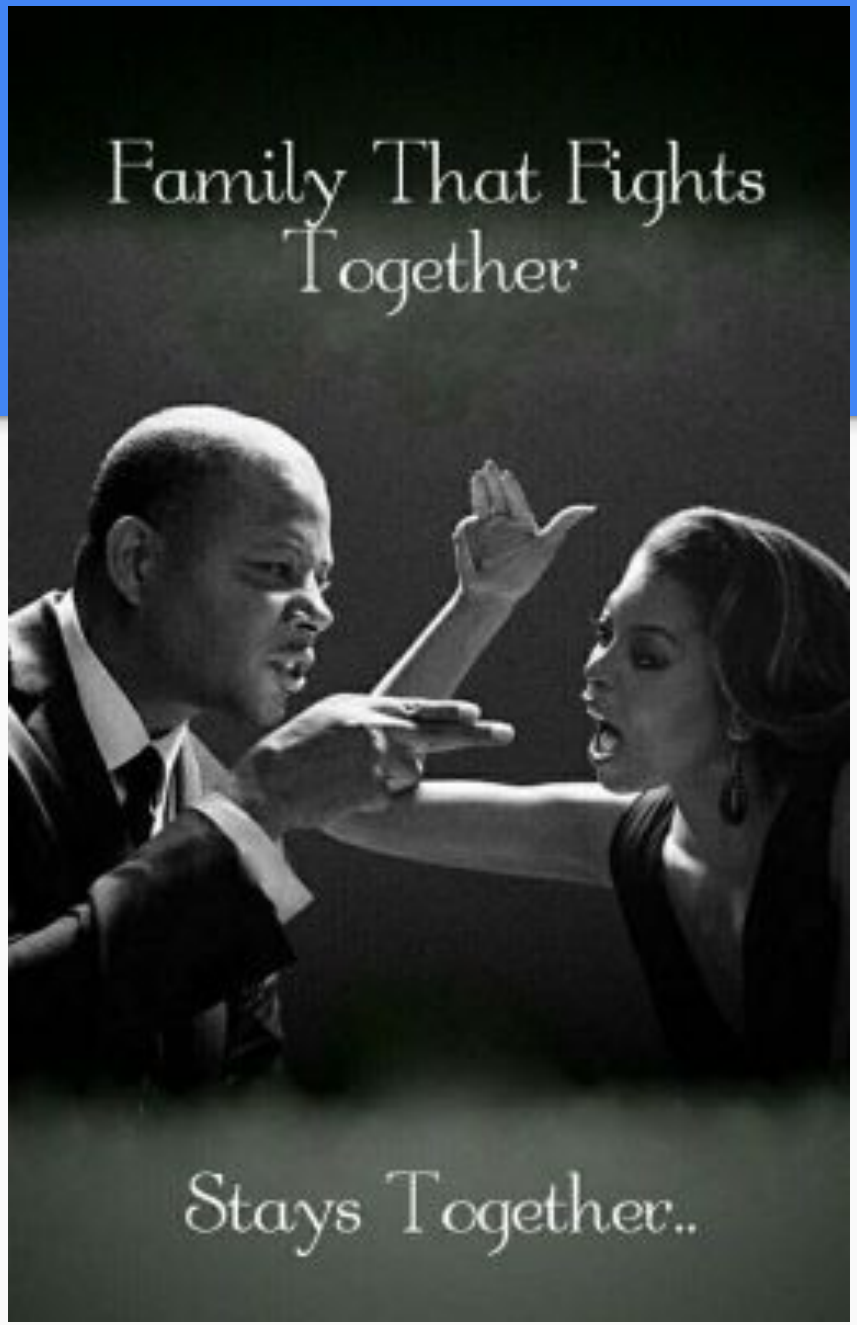
Remember

- Cohesion and Adhesion [Plus Two]
- Coherence [From -2 to forever]

- And now in Software Engineering

Is there cohesion in Indian Railways ?

And in families, ...


Family That Fights Together
Stays Together..

Design Principle 3: Reduce coupling where possible

Coupling occurs when there are *interdependencies* between one module and another

- When interdependencies exist, changes in one place will require changes somewhere else.
- Type of coupling:
  - Read Text for examples & The other source for types

Coupling

A **coupling** is a device used to connect two shafts together at their ends for the purpose of transmitting power.

Is there coupling in Indian Railways ?


Screw Coupling
Explain


SCHARFENBERG COUPLER

And BE PROUD!

Cohesion and Coupling

- Cohesion is a measure of:
  - functional strength of a module.
  - A cohesive module performs a single task or function.
- Coupling between two modules:
  - A measure of the degree of the interdependence or interaction between the two modules.

Cohesion and Coupling

- A module having high cohesion and low coupling:
  - functionally independent of other modules:
    - A functionally independent module has minimal interaction with other modules.

Advantages of Functional Independence

- Better understandability and good design:
- Complexity of design is reduced,
- Different modules easily understood in isolation:
  - Modules are independent

Advantages of Functional Independence

- Functional independence reduces error propagation.
  - Degree of interaction between modules is low.
  - An error existing in one module does not directly affect other modules.
- Reuse of modules is possible.

Advantages of Functional Independence

- A functionally independent module:
  - Can be easily taken out and reused in a different program.
    - Each module does some well-defined and precise function
    - The interfaces of a module with other modules is simple and minimal.

Mostly used well in

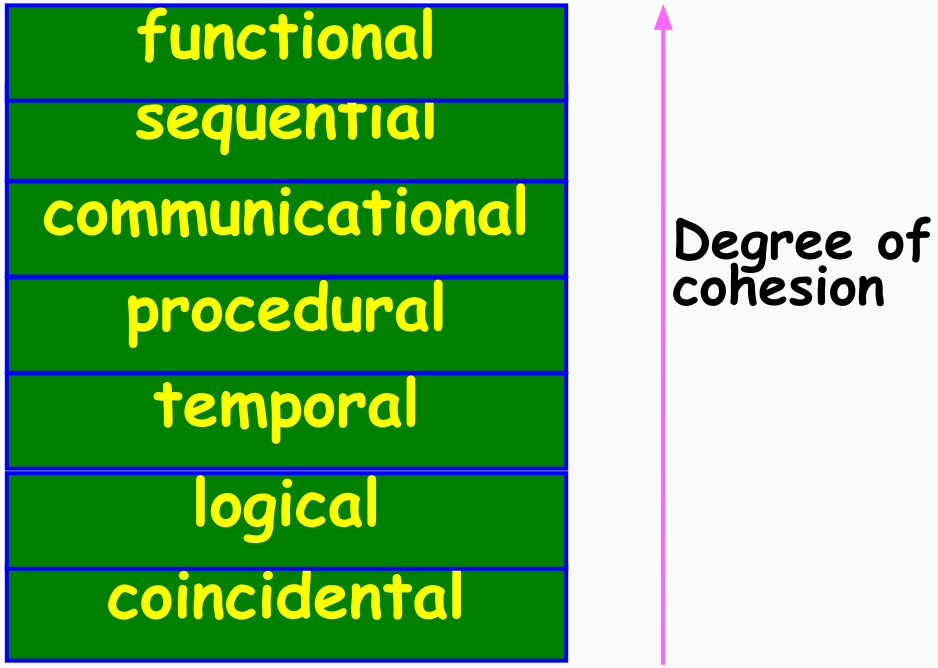
Functional Independence

- Unfortunately, there are no ways:
  - To quantitatively measure the degree of cohesion and coupling.
  - Classification of different kinds of cohesion and coupling:
    - Can give us some idea regarding the degree of cohesiveness of a module.

Classification of Cohesiveness

- Classification is often subjective:
  - Yet gives us some idea about cohesiveness of a module.
- By examining the type of cohesion exhibited by a module:
  - We can roughly tell whether it displays high cohesion or low cohesion.

Classification of Cohesiveness



Coincidental Cohesion

- The module performs a set of tasks:
  - Which relate to each other very loosely, if at all.
    - The module contains a random collection of functions.
    - Functions have been put in the module out of pure coincidence without any thought or design.

Logical Cohesion

- All elements of the module perform similar operations:
  - e.g. error handling, data input, data output, etc.

Temporal Cohesion

- The module contains tasks that are related by the fact:
  - All the tasks must be executed in the same time span.
-

Procedural Cohesion

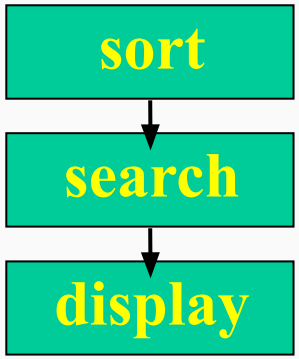
- The set of functions of the module:
  - All part of a procedure (algorithm)
  - Certain sequence of steps have to be carried out in a certain order for achieving an objective,
    - e.g. the algorithm for decoding a message.

Communicational Cohesion

- All functions of the module:
  - Reference or update the same data structure,
-

Sequential Cohesion

- Elements of a module form different parts of a sequence,
  - Output from one element of the sequence is input to the next.
  - Example:



Functional Cohesion

- Different elements of a module cooperate:
  - To achieve a single function,
  - e.g. managing an employee's pay-roll.
- When a module displays functional cohesion,
  - We can describe the function using a single sentence.

Determining Cohesiveness

- Write down a sentence to describe the function of the module
  - If the sentence is compound,
    - It has a sequential or communicational cohesion.
  - If it has words like "first", "next", "after", "then", etc.
    - It has sequential or temporal cohesion.
  - If it has words like initialize,
    - It probably has temporal cohesion.

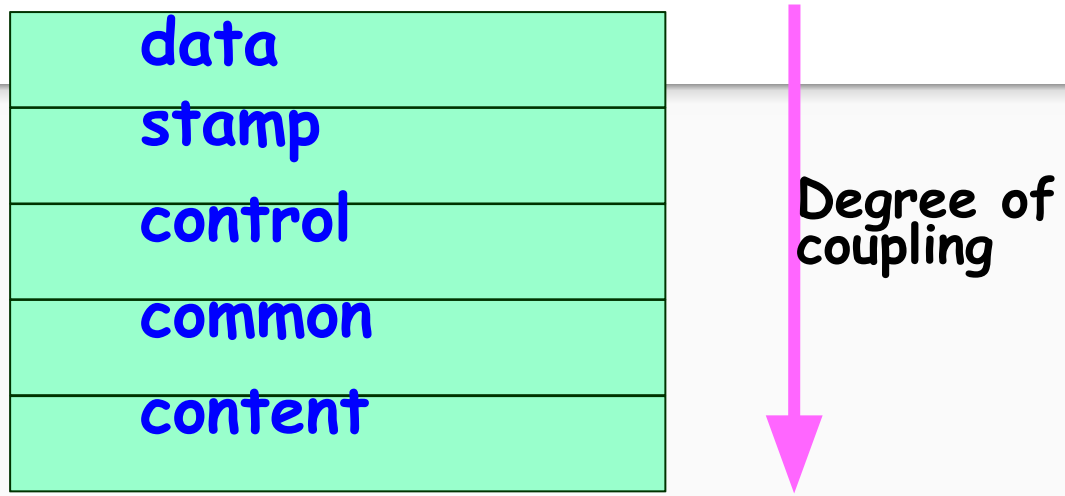
Coupling

- Coupling indicates:
  - How closely two modules interact or how interdependent they are.
  - The degree of coupling between two modules depends on their interface complexity.

Coupling

- There are no ways to precisely determine coupling between two modules:
  - Classification of different types of coupling will help us to approximately estimate the degree of coupling between two modules.
- Five types of coupling can exist between any two modules.

Classes of coupling



Data coupling

- Two modules are data coupled,
  - If they communicate via a parameter:
    - an elementary data item,
    - e.g an integer, a float, a character, etc.
  - The data item should be problem related:
    - Not used for control purpose.

Stamp Coupling

- Two modules are stamp coupled,
  - If they communicate via a composite data item
    - such as a record in PASCAL
    - or a structure in C.

Control Coupling

- Data from one module is used to direct:
  - Order of instruction execution in another.
- Example of control coupling:
  - A flag set in one module and tested in another module.

Common Coupling

- Two modules are common coupled,
  - If they share some global data.

Content Coupling

- Content coupling exists between two modules:
  - If they share code,
  - e.g, branching from one module into another module.
- The degree of coupling increases
  - from data coupling to content coupling.

Design Principle 4: Keep the level of abstraction as high as possible

Ensure that your designs allow you to hide or defer consideration of details, thus reducing complexity

- A good abstraction is said to provide *information hiding*
- Abstractions allow you to understand the essence of a subsystem without having to know unnecessary details
- Wishful Thinking

Abstraction and classes

Classes are data abstractions that contain procedural abstractions

- Abstraction is ? by defining all variables as private.
- The ? public methods in a class, the better the abstraction
- Superclasses and interfaces ? the level of abstraction
- Attributes and associations are also data abstractions.
- Methods are ? abstractions
  - Better abstractions are achieved by giving methods ? parameters

Design Principle 5: Increase reusability where possible

Design the various aspects of your system so that they can be used again in other contexts

- Generalize your design as much as possible [Abstract]
- Simplify your design as much as possible
  - Figma ?

Design Principle 6: Reuse existing designs and code where possible

Design with reuse is complementary to design for reusability

- Actively reusing designs or code allows you to take advantage of the investment you or others have made in reusable components
  - — *Are you suggesting that we should copy?*
  - — *What about academic integrity?*


ProgrammerHumor.io

the code copied from Stackoverflow

the rest of the code

You are unique, but not your problems :-)

Design Principle 6: Reuse existing designs and code where possible

Design with reuse is complementary to design for reusability

— *Are you suggesting that we should copy?*

— *What about academic integrity?*

Academia vs Industry

Academia - Either do not copy or copy anything but give due credits [cite]

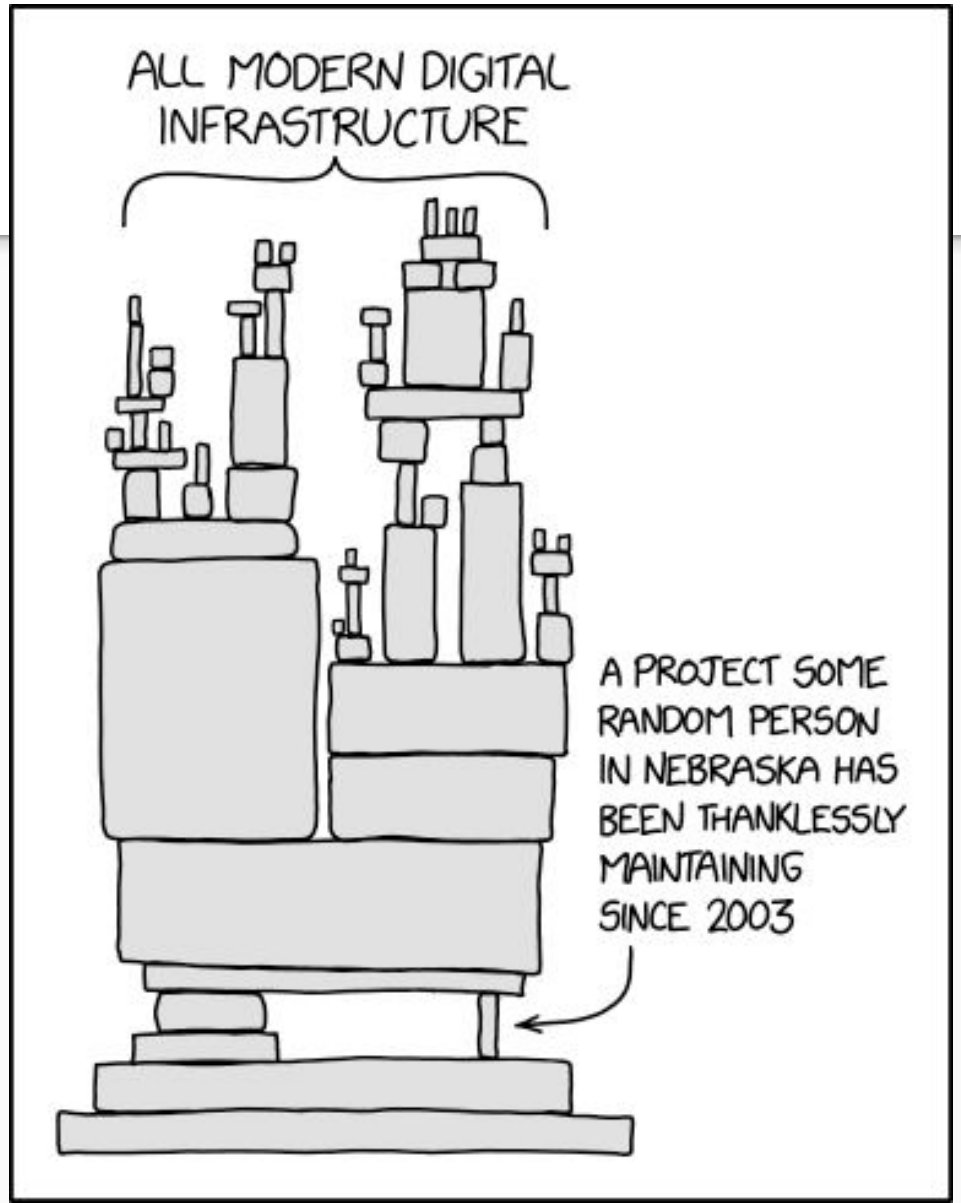
Industry - Extra careful and need to bother about licenses - GNU GPL, BSD etc.

Dart/Flutter? React js?

https://choosealicense.com/

Be careful!

Be thankful!

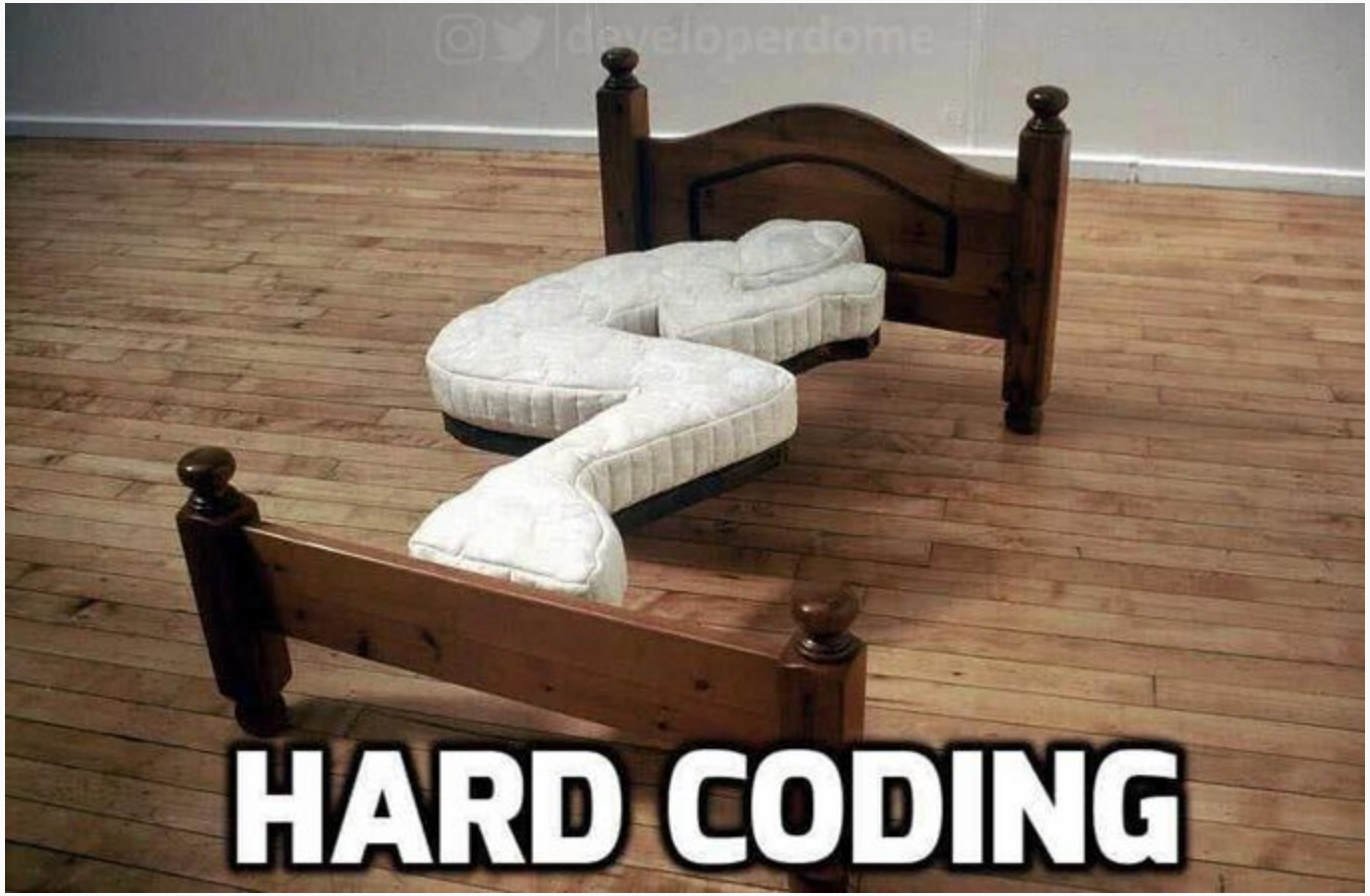

ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

Design Principle 7: Design for flexibility

Actively anticipate changes that a design may have to undergo in the future, and prepare for them

- Reduce coupling and increase cohesion
- Create abstractions
- Use reusable code and make code reusable
- And finally,

A “perfectly designed” cot and a matching bed, is it?



Do NOT hardcode!

Admin Login
NITC related stuff

When you hardcode in your part, it is a sure sign of integration issues, CI/CD will be a nightmare because the GIT merge will look like →

do we have time?

Design Principle 8: Anticipate obsolescence

Plan for changes in the technology or environment so the software will continue to run or can be easily changed

- Avoid using early releases of technology
- Avoid using software libraries that are specific to particular environments
- Avoid using software or special hardware from companies that are less likely to provide long-term support
- Use standard languages and technologies that are supported by multiple vendors

Design Principle 9: Design for Portability

Have the software run on as many platforms as possible

- Avoid the use of facilities that are specific to one particular environment

  E.g. a library only available in Microsoft Windows; only on Chrome browser

- RWD

CONTENT IS LIKE WATER

"You put water into a cup it becomes the cup.
You put water into a bottle it becomes the bottle.
You put it in a teapot, it becomes the teapot."

Josh Clark *(originally Bruce Lee)* - Seven deadly mobile myths

Illustration by Stéphanie Walter

Design Principle 10: Design for Testability

Take steps to make testing easier

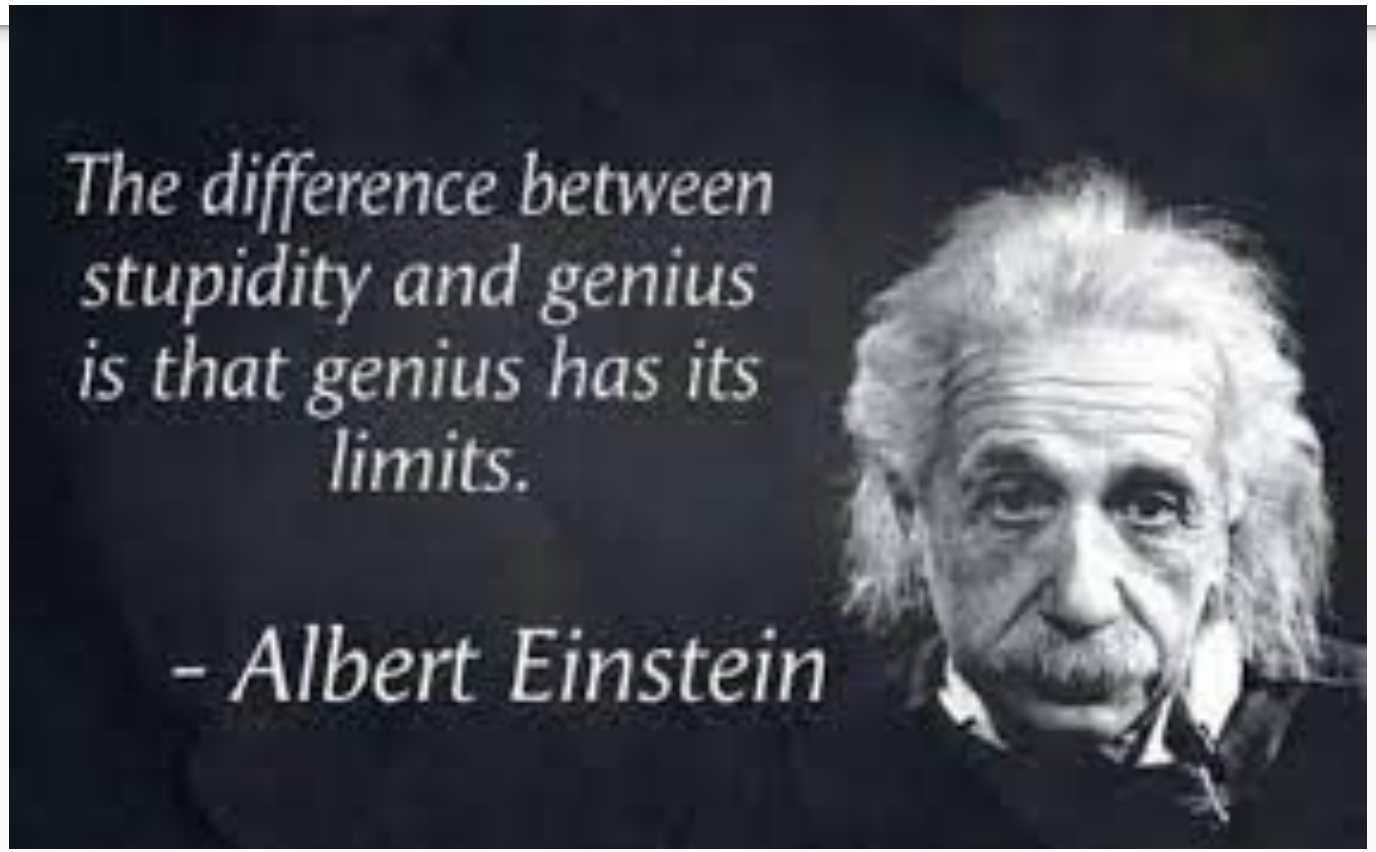
- Design a program to automatically test the software
  - — Ensure that all the functionality of the code can be driven by an external program, bypassing a graphical user interface

DevOps Tools - Docker/Jenkins/…

Design Principle 11: Design defensively

- Handle all cases where other code might attempt to use your component inappropriately
- Check that all of the inputs to your component are valid: the *preconditions*
  - — Unfortunately, over-zealous defensive design can result in unnecessarily repetitive checking
  - — Defensive Programming?
  - — Generative Programming?
- Never Trust how Others are going to use your software

Never trust how others will try to use a component you are designing

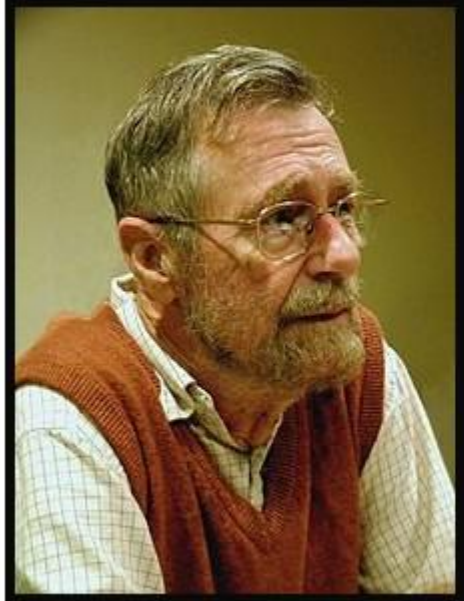


Design by contract

A technique that allows you to design defensively in an efficient and systematic way

- Key idea
  - each method has an explicit *contract* with its callers
- The contract has a set of assertions that state:
  - What *preconditions* the called method requires to be true when it starts executing
  - What *postconditions* the called method agrees to ensure are true when it finishes executing
  - What *invariants* the called method agrees will not change as it executes

Academia vs. Industry

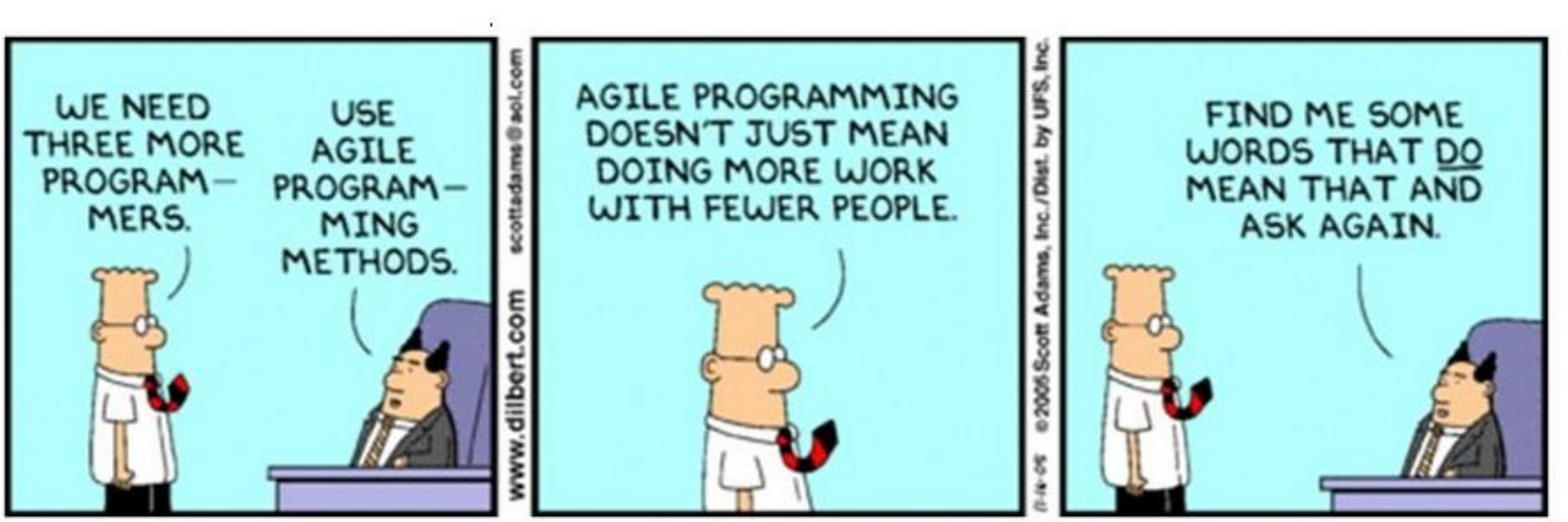


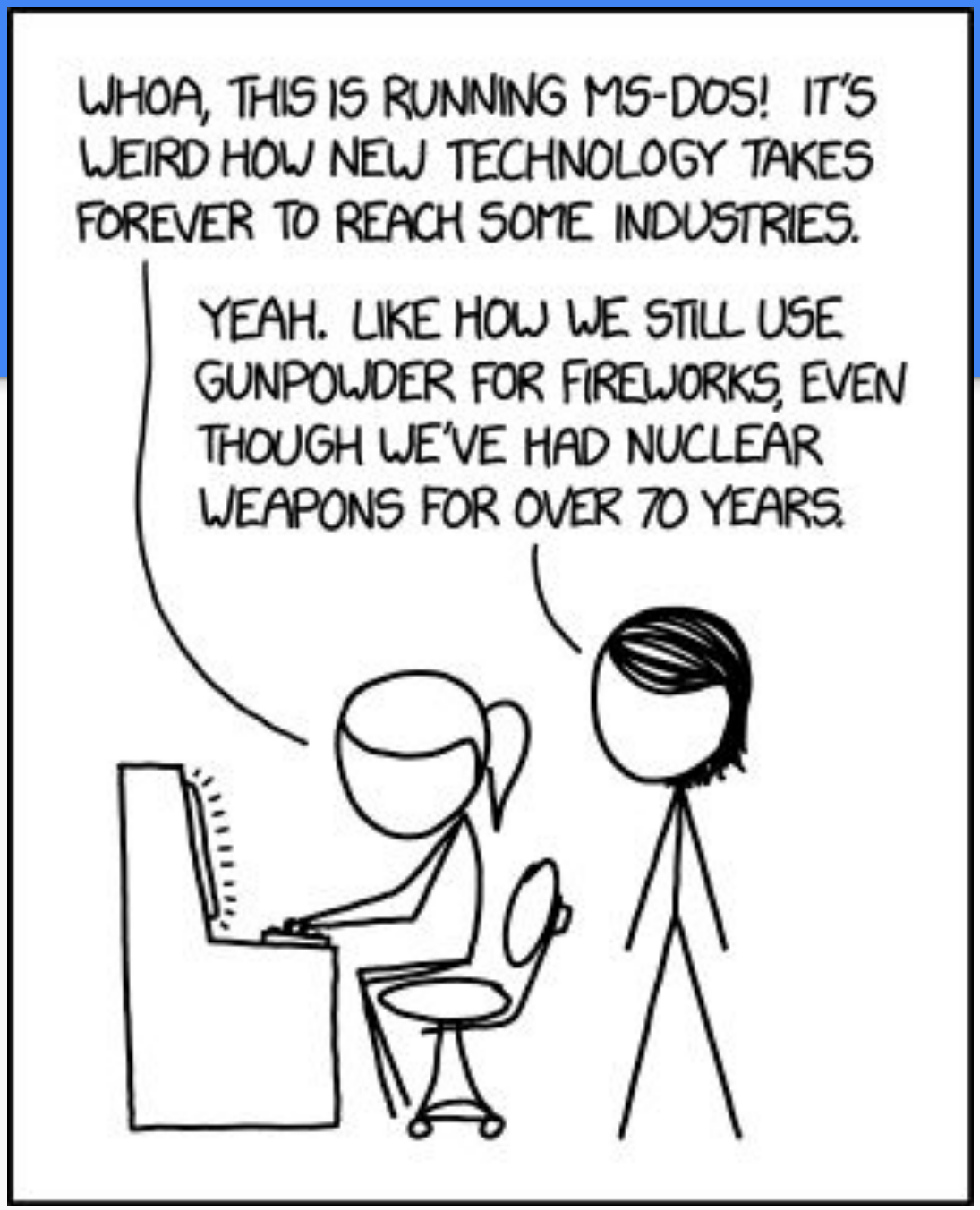
If 10 years from now, when you are doing something quick and dirty, you suddenly visualize that I am looking over your shoulders and say to yourself: 'Dijkstra would not have liked this', well that would be enough immortality for me.

(Edsger Dijkstra)

izquotes.com

Academia vs. Industry




WHOA, THIS IS RUNNING MS-DOS! IT'S WEIRD HOW NEW TECHNOLOGY TAKES FOREVER TO REACH SOME INDUSTRIES.
YEAH. LIKE HOW WE STILL USE GUNPOWDER FOR FIREWORKS, EVEN THOUGH WE'VE HAD NUCLEAR WEAPONS FOR OVER 70 YEARS.

A quick recap

Divide and Conquer

Increase Cohesion

Decrease Coupling

Keep high level of Abstraction

Design for Reusability

Reuse existing Designs

Design for Flexibility

Design for Portability

Design for Testability

Design for Stupidity (**Design Defensively**)

A quick recap

Divide and Conquer

Increase Cohesion

Decrease Coupling

Keep high level of Abstraction

Design for Reusability

Reuse existing Designs

Plan for Obsolescence

Design for Flexibility

Design for Portability

Design for Testability

Design for Stupidity (**Design Defensively**)

11 principles from Text
Many other sources too…
Never miss any article on this topic

Labor Intensive :-(

High Salary

Easily replaceable

Not like decommissioning a Ship, or a Furnace [or the technicians in it]

Just don't give hike, and if necessary pink slip.

—-----

Ability to interact, communicate, network

Read A Lot and Write A Bit (a few bytes, actually)

—-----

You need to (continue to) be VALUABLE to the org

Before I conclude and say good bye!

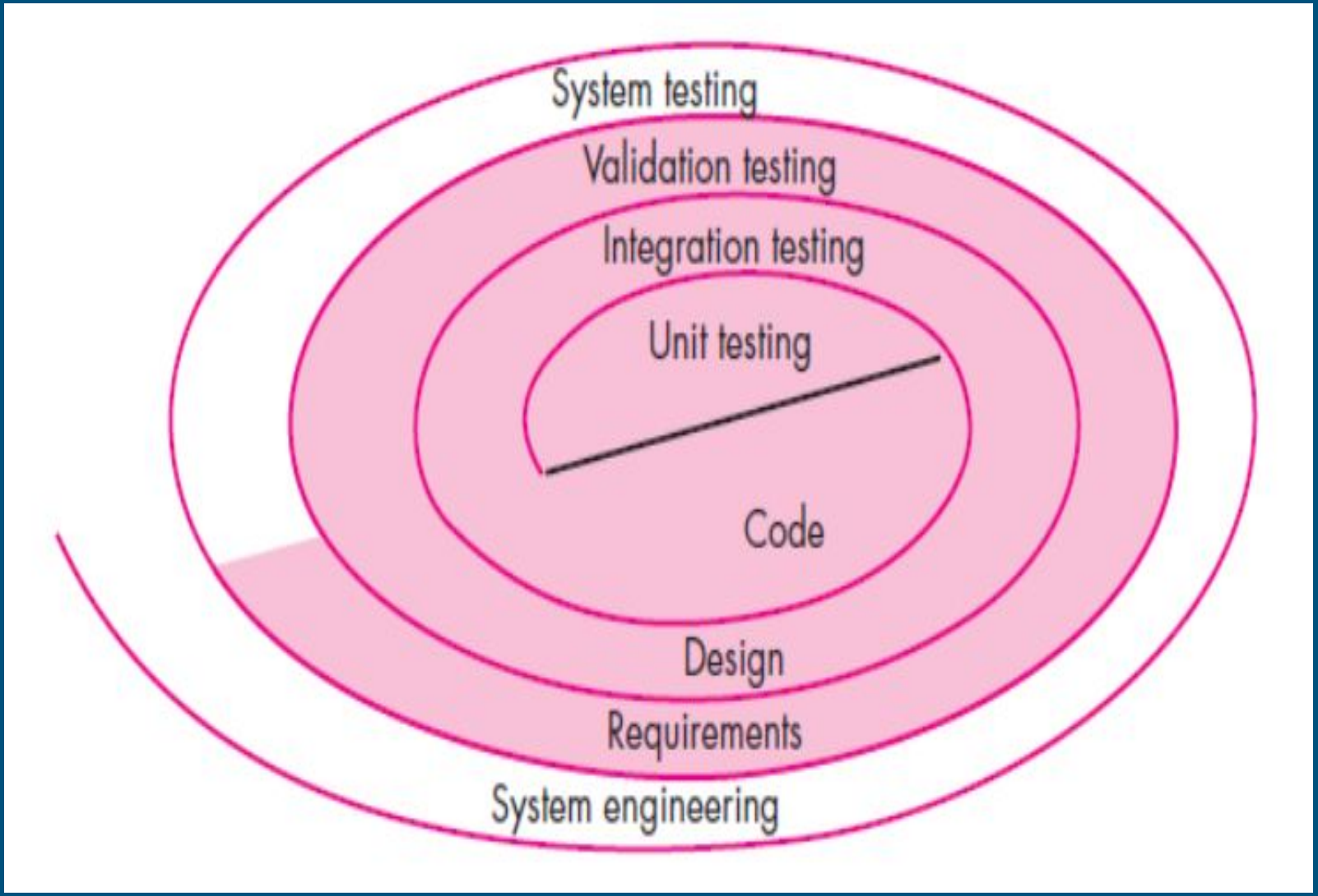
Three slides from Dr Manjusha

V & V

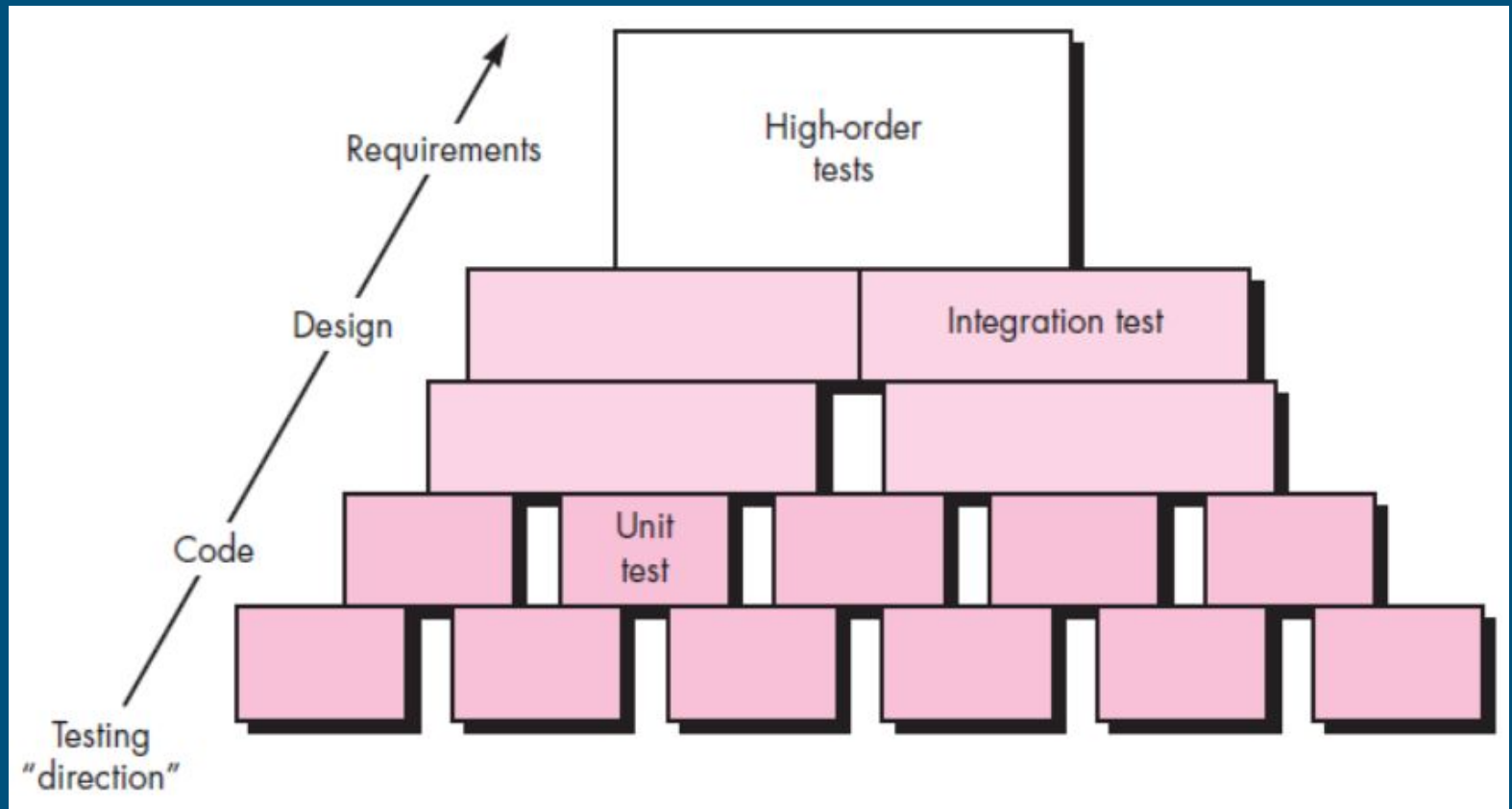
Verification: “Are we building the product right?”

Validation: “Are we building the right product?”

The BIG Picture



Remember GIT Merge



All the best from All 3 of us to BCS20!