CSIT314 Software Development Methodologies

Introduction to Software Development and its Lifecyle

Software Engineering

- Engineering vs. Science
- Software Engineering is

  "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software" (IEEE standard 610.12-1990).

Components of Software Engineering

2 main components –

- PRODUCT
  - The actual software product or system that is built and put into operation
- PROCESS
  - A framework for the tasks that are required to build high-quality software.

What is Engineering?

- A body of knowledge used when building things
  - Scheduling
  - Costing
  - Estimating
  - Building
  - Testing
  - Communicating
  - Organising

It is easy to build something if you have unlimited money and time. A professional differs from an amateur in that they can contain costs and time.

How software is different?

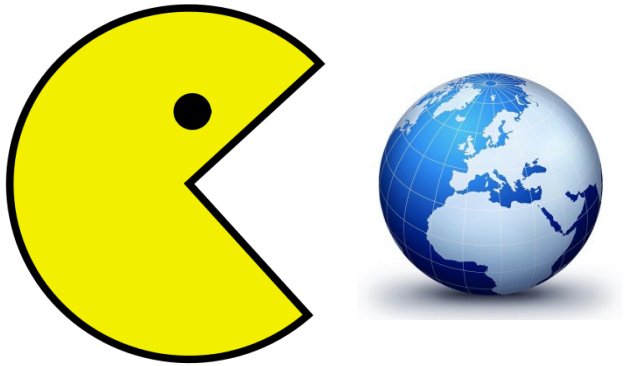
- Software is soft and intangible
- There are no physical laws underlying software behaviour
- Software are never wears out
  - traditional reliability measures don’t apply
- Software is not mass produced
- The specification for software continuously changes

"Software eats the World"

- "*We are in the middle of a dramatic and broad technological and economic shift in which software companies are poised to take over large swathes of the economy*"

  (Marc Andreessen, "Why Software is Eating the World", **The Wall Street Journal**, http://online.wsj.com/article/SB10001424053111903480904576512250915629460.html)

- More and more major businesses and industries are being run on software and delivered online services.



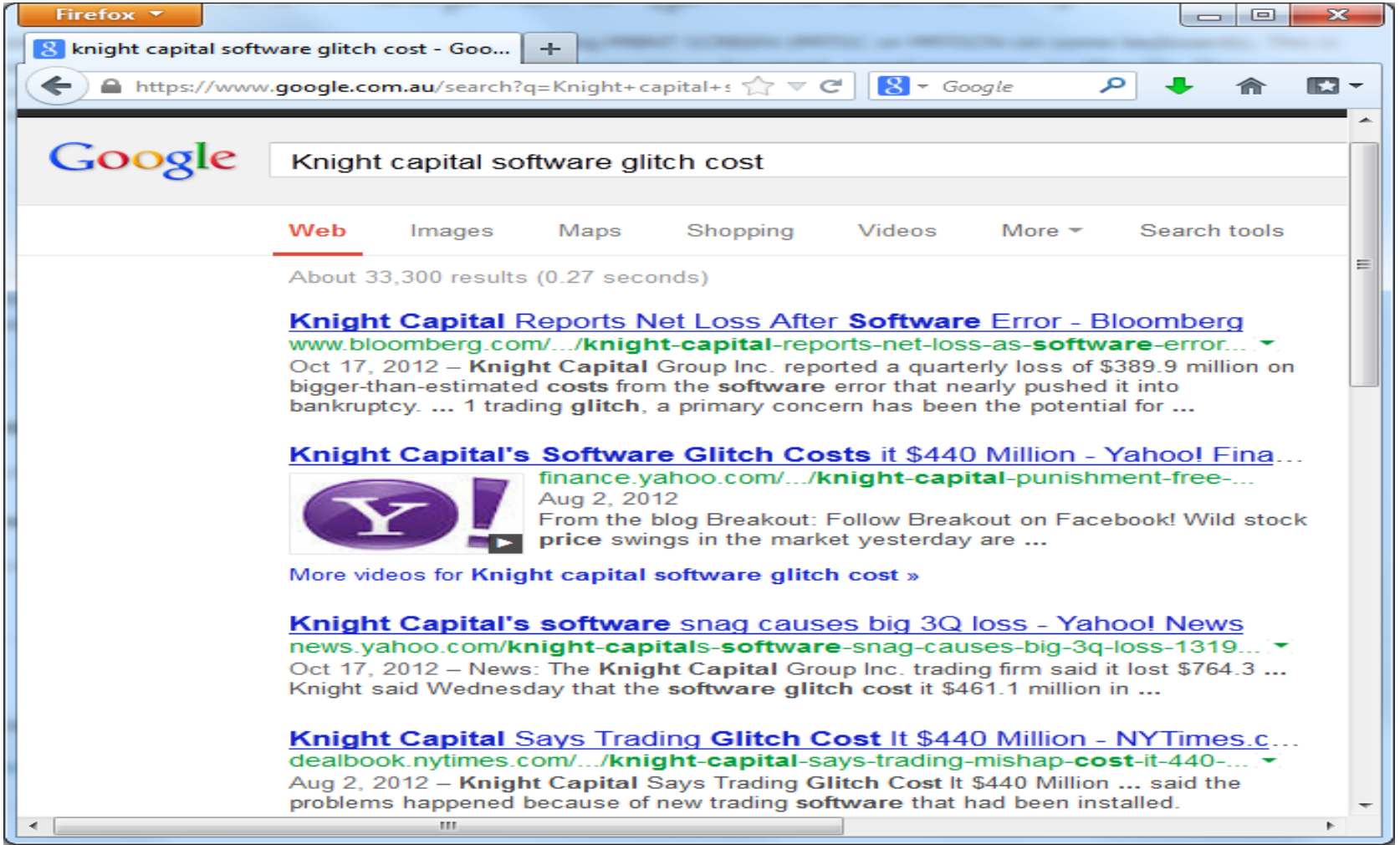
“Software eats the World” (cont.)

Good news for us ….

But “*With great power there must also come great responsibility*” …

Low-quality software costs jobs …

https://www.youtube.com/watch?v=7BKNnpJfWII



Low-quality software costs money ...

On the 4th June 1996 at 1233 GMT (UTC) the European Space Agency launched a new rocket, **Ariane 5**, on its maiden unmanned flight,

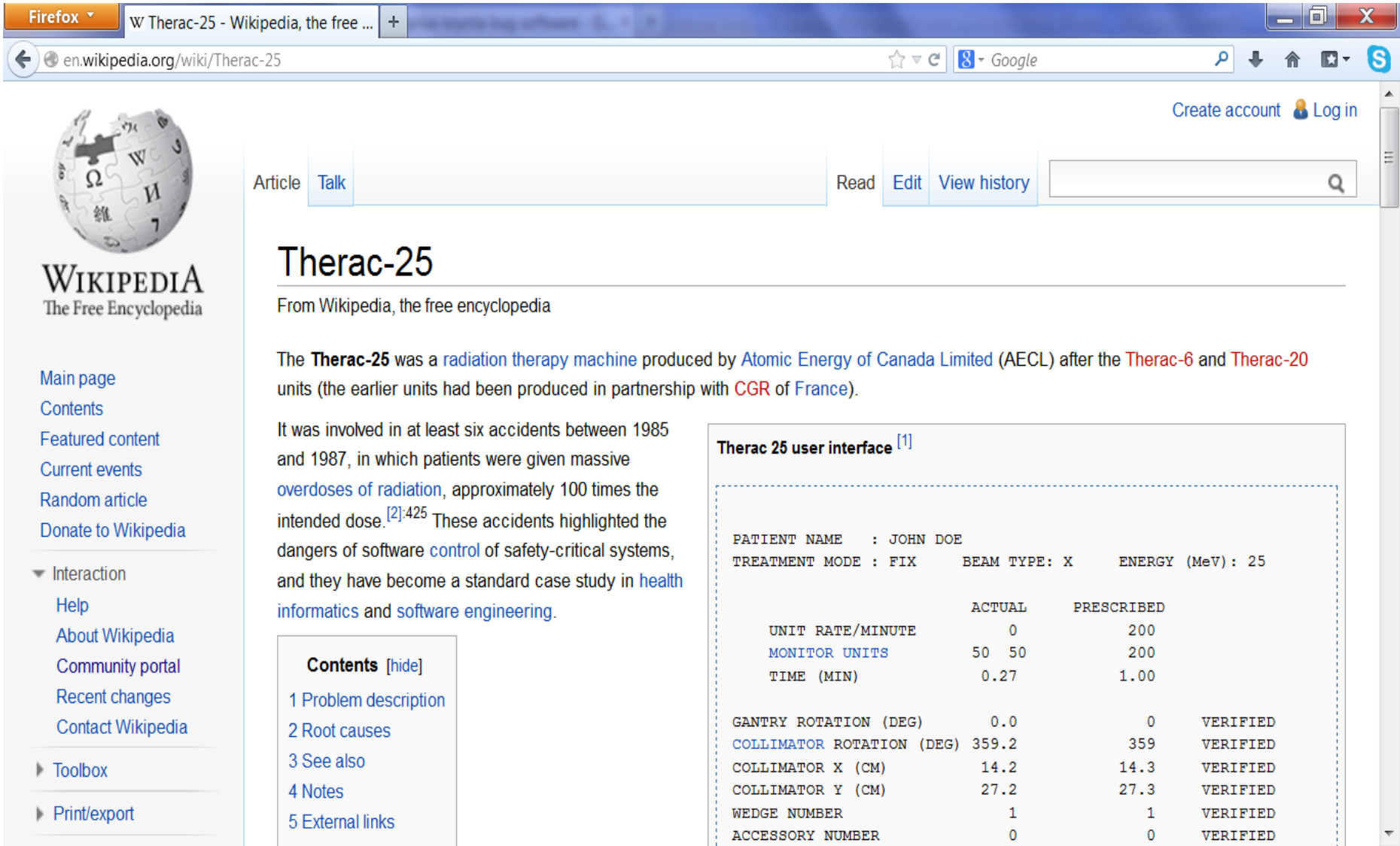
https://www.youtube.com/watch?v=kYUrqdUyEpI

"It took the European Space Agency **10 years** and **$7 billion** to produce **Ariane 5**, a giant rocket capable of hurling a pair of three-ton satellites into orbit with each launch and intended to give Europe overwhelming supremacy in the commercial space business.

All it took to explode that rocket less than a minute into its maiden voyage last June, scattering fiery rubble across the mangrove swamps of French Guiana, was a **small computer program** trying to stuff a 64-bit number into a 16-bit space"

Source: http://www.around.com/ariane.html

Low-quality software costs lives



Software

The Airbus A380 uses a substantial amount of software to create a "paperless" cockpit

Q: How many lines of code constituting the plane's software?

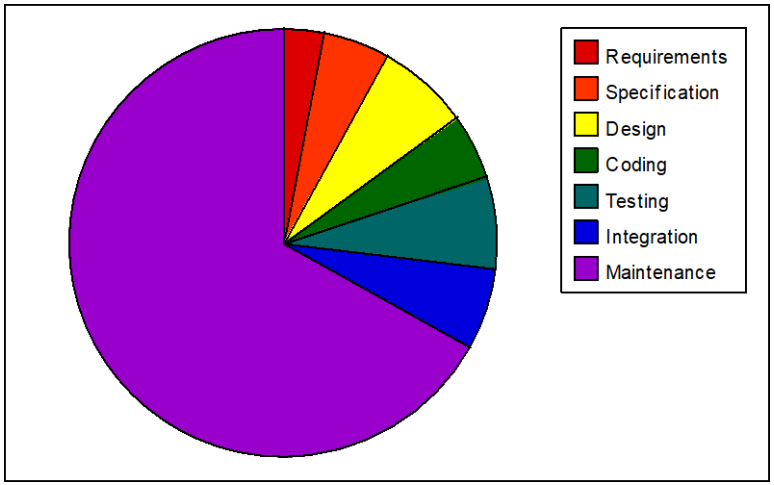
A: **Millions** of lines of code

Android OS has 15 million lines of code. Moodle have 1 million lines of code. Windows XP has 45 million lines of code.

Software Engineering plays a critical part in the development of such a large-scale software.

Software Development Activities

- Planning
- Requirements analysis
- Design
- Implementation
- Verification & Validation
- Maintenance and evolution



Planning

- Identify business value
- Analyse feasibility
- Develop work plan
- Staff the project
- Estimate
- Identify risk

You have learnt these in CSIT214

Software Engineering Development Activities

- Planning
- Requirements analysis
- Design
- Implementation
- Verification & Validation
- Maintenance and evolution

Requirements analysis

- The process of establishing what services are required and the constraints on the system's operation and development.
- There are two major activities:
  - Requirements elicitation
    - Who are the stakeholders of this project?
    - What do the system stakeholders require or expect from the system?
    - Interviews, questionnaires, meeting, etc.
  - Requirements specification
    - Defining the requirements in detail
    - Use cases are the major part of a requirements specification

Software Engineering Development Activities

- Planning
- Requirements analysis
- Design
- Implementation
- Verification & Validation
- Maintenance and evolution

Design

- What is Design?
  - Design is concerned with HOW we build a system
- How is Design different from Requirements Analysis?
  - Analysis is concerned with WHAT is to be built
- There are many different aspects to design

Design (cont.)

Design comprises of different aspects:

1. Architectural design
2. Sub system design
3. Detailed design
4. Persistent data design
5. GUI design

Architecture Design

Concerned with major structure of the software

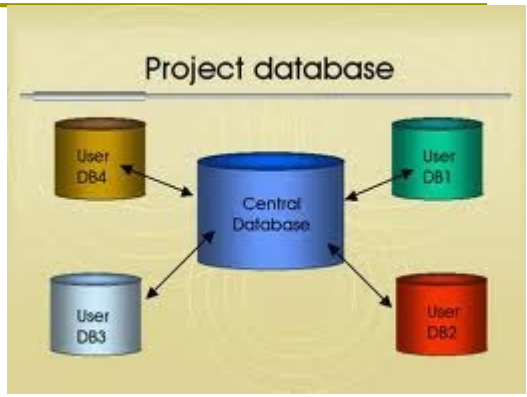
- Similar to building architecture (e.g. roof, walls, foundations)
- Contains sub-systems (air-conditioning unit, water pump)
- Uses external services (sewerage, water, gas, electric)
- The most important part of design – if the architecture is wrong, the house will fall down (or software will fall down!)

Design

- Sub-system design
  - Describes interfaces/protocols of each sub-system
  - Structure of each sub-system
  - Structure can describe how each sub-system sub-divides into packages, components
- Detailed design
  - Describes each class, methods, attributes/data-types

Design (cont.)

- Persistent data design
  - Describes choice of database, tables, primary and foreign keys



- User interface design
  - Describes:
  - how the GUI will look like
  - what tools will be used to build it
  - the structure of the GUI
  - what design guidelines will be followed
  - what style guidelines will be followed

Flashback Quiz

- Software deteriorates rather than wears out because
  - **A)** Software suffers from exposure to hostile environments
  - **B)** Defects are more likely to arise after software has been used often
  - **C)** Multiple change requests introduce errors in component interactions
  - **D)** Software spare parts become harder to order

Software Engineering Development Activities

- Planning
- Requirements analysis
- Design
- Implementation
- Verification & Validation
- Maintenance and evolution

Implementation

- Construction
  - Write the code for the project.
- Installation (or deployment)

Software Engineering Development Activities

- Planning
- Requirements analysis
- Design
- Implementation
- Verification & Validation
- Maintenance and evolution

Software verification and validation

- Verification and validation (V & V) is intended to show that:
  - a system conforms to its specification and
  - meets the requirements of the system customer.
- Testing is the most commonly used V & V activity.
- System testing involves **executing** the system with **test cases** that are derived from the specification of the **real data** to be processed by the system.

Verification

- It works!
- Final check through all those **use cases** from the specification document
  - Does the system perform exactly as specified?

and Validation

- Bring in the users and demonstrate

You: It is working. Here let me demonstrate how you would …

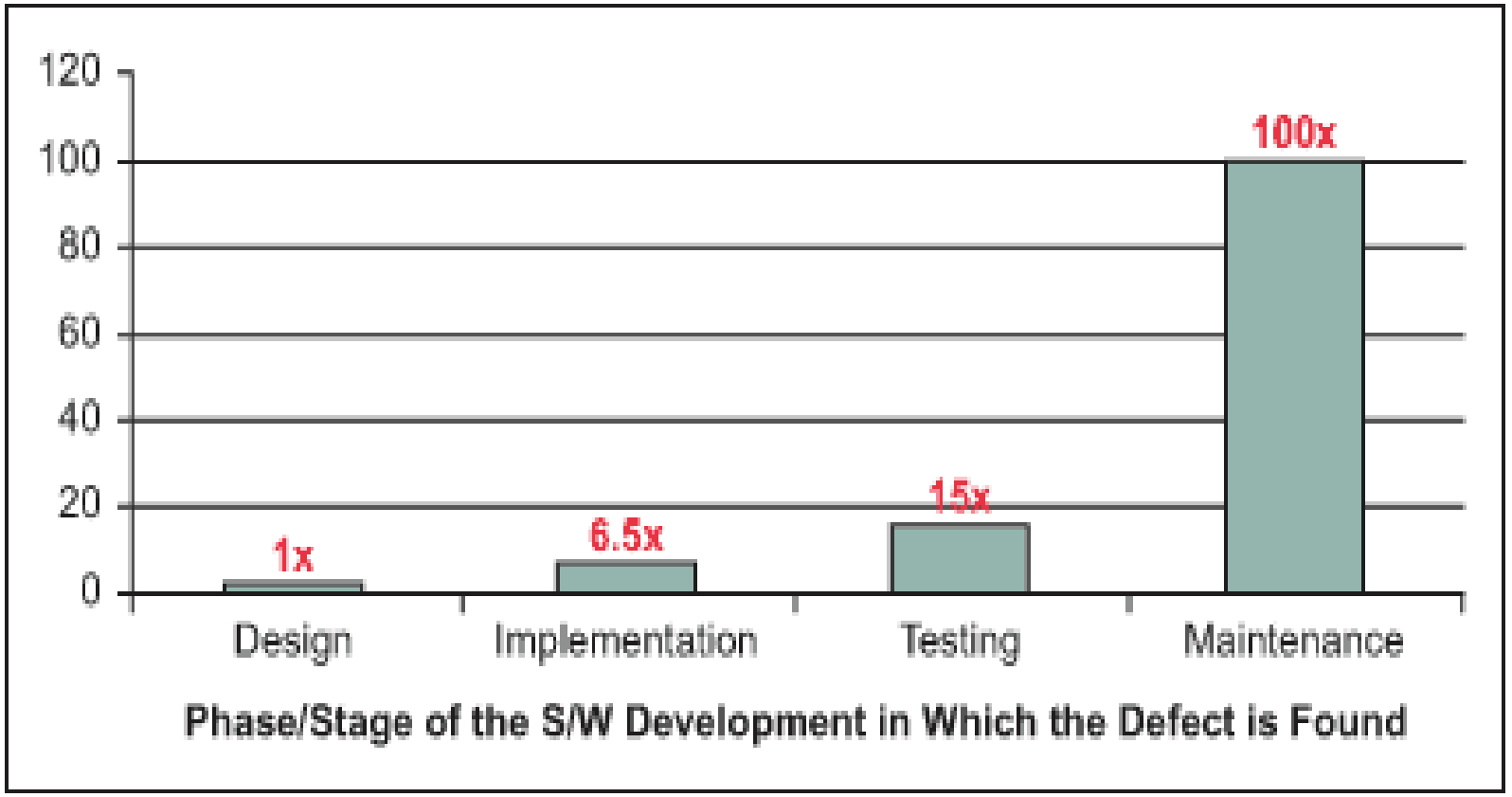
User: That's not what I wanted!

I wanted …

You: It is exactly what is in the agreed specification! You said you wanted …

User: You should have implemented what I meant, not what I said.

Cost of defects …



- Cost of correcting an error in requirement specifications increases as we move through lifecycle phases

Software Engineering Development Activities

- Planning
- Requirements analysis
- Design
- Implementation
- Verification & Validation
- Maintenance and evolution

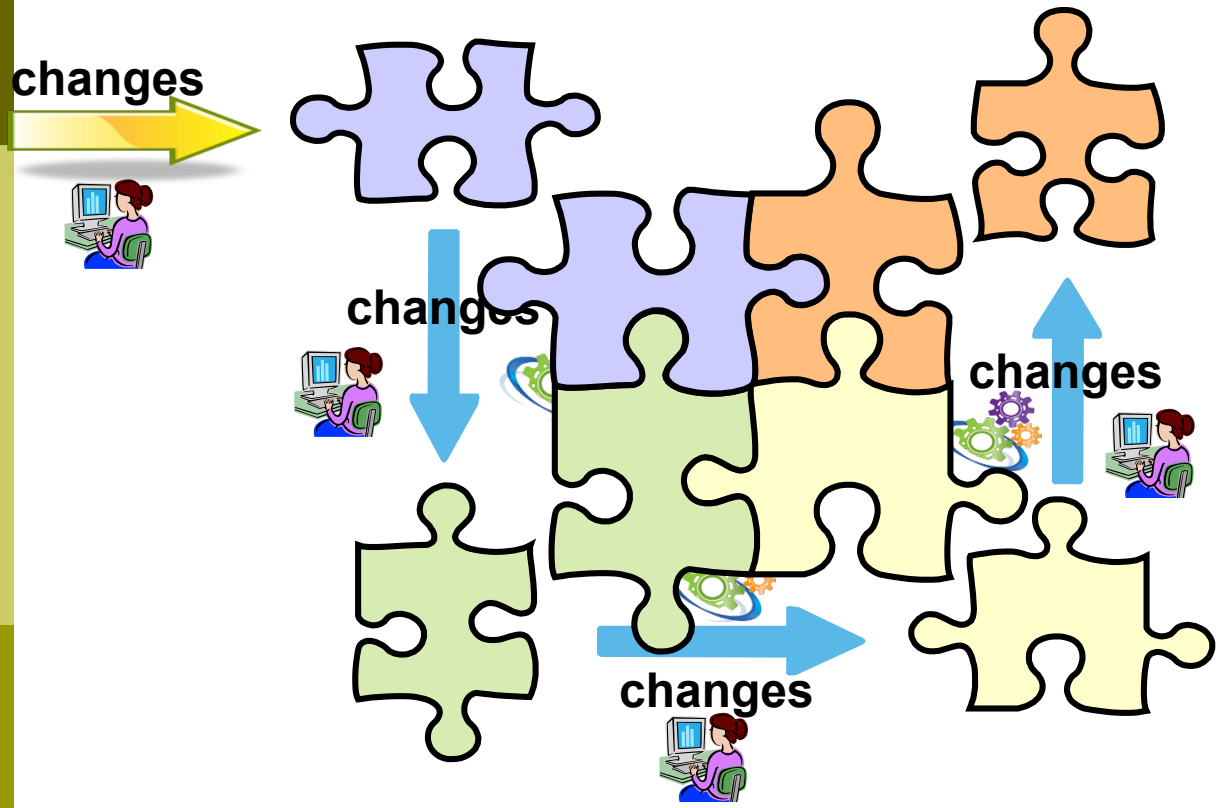
Software maintenance and evolution

- Changes are inevitable in software development – WHY?:
  - New requirements emerged at any time in the software development lifecycle.
    - E.g. new functionalities
  - Changes in business environments
    - E.g. competition, laws, new markets, new customers, etc.
  - Changes in infrastructure environments
    - E.g. new servers, new equipments, etc.
  - New technology arriving
    - E.g. New version of OS, new standards, etc.
  - Bugs need fixing
  - Performance needs improvement.

Types of software maintenance

- Adaptive maintenance
  - Changing the system in response to changes in its environment so it continues to function
- Corrective maintenance
  - Fixing errors & bugs
- Perfective maintenance
  - Changing the system's functionality to meet changing needs

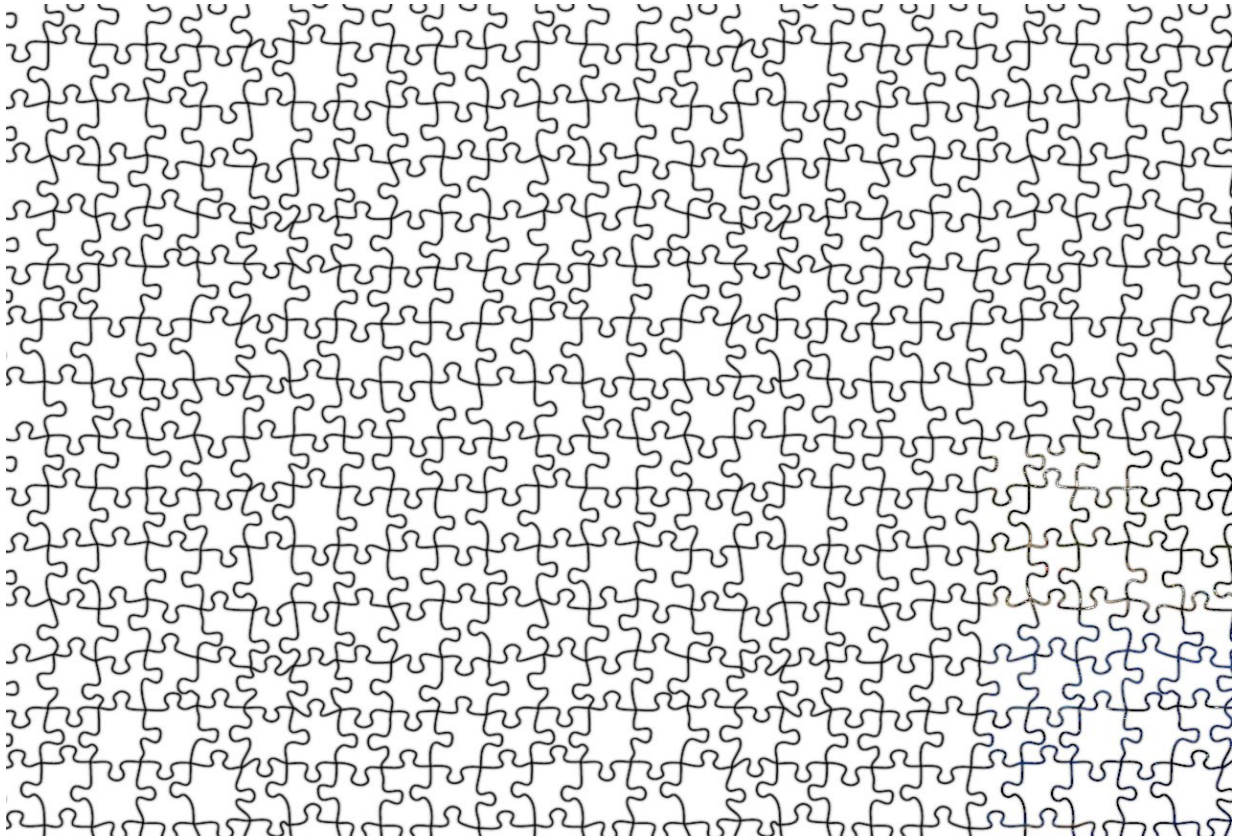
Change propagation



The ripple effect

As a change is started on a software system, other coordinated changes are often needed at the same time in other parts of the software.

Change propagation (cont.)



These other required changes, if ***not*** completed immediately, incur a kind of ***debt*** that must be paid at some point in the future.

Technical Debt

- "*Technical debt is a concept in software development that reflects the implied cost of additional rework caused by choosing an easy (limited) solution now instead of using a better approach that would take longer*"
- If technical debt is not repaid, it can accumulate 'interest', making it harder to implement changes.
- How to repay technical debt?

Source: https://en.wikipedia.org/wiki/Technical_debt

The eight laws of software evolution

- **Law 1: Continuing change**
- **Law 2: Increasing complexity**
- Law 3: Self regulation
- Law 4: Conservation of organisational stability
- Law 5: Conservation of familiarity
- **Law 6: Continuing growth**
- **Law 7: Declining quality**
- Law 8: Feedback system

Source: M. M. Lehman. *Laws of Software Evolution Revisited*. Lecture Notes in Computer Science 1149, pp. 108-124, Springer Verlag, 1997

The eight laws of software evolution

- **Law 1: Continuing change**
  - *A program that is used in a real-world environment must be continually adapted, or else become progressively less satisfactory*

- Why?
  - New requirements emerged constantly
  - Evolution of the environment => increasing mismatch between the system and its environment.

The eight laws of software evolution

- **Law 2: Increasing complexity**
  - *As a program is evolved, its complexity increases with time, unless specific work is done to maintain or reduce it.*

- Why?
  - Unpaid technical debts increases **software entropy**
  - Accumulated technical debt makes it more difficult to change.
  - Refactoring and restructuring may be needed.

The eight laws of software evolution

- **Law 6: Continuing growth**
  - *Functional content of a program must be continually increased to maintain user satisfaction over its lifetime.*

- Why?
  - If the users find they can get work done with the new system, then they will soon identify additional tasks for it to do and require changed forms for interaction with the system.

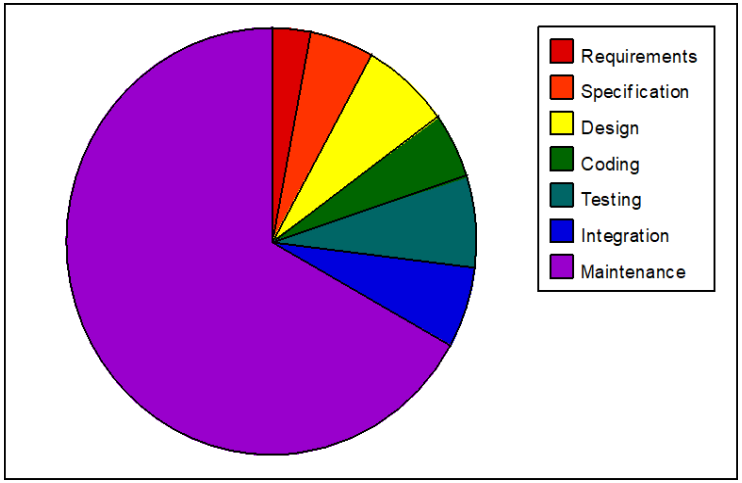
The eight laws of software evolution

- **Law 7: Declining quality**
  - *Evolving programs will be perceived as of declining quality unless rigorously maintained and adapted to a changing operational environment.*

- Why?
  - Similar to Law 1 but this law emphasizes on reliability and performance.

Review

- Planning
- Requirements analysis
- Design
- Implementation
- Verification & Validation
- Maintenance and evolution



Development costs

Video

- The evolution of Eclipse through a visualization
  - https://vimeo.com/1130828