COMP 3111
SOFTWARE ENGINEERING

INTRODUCTION

LEARNING OBJECTIVES

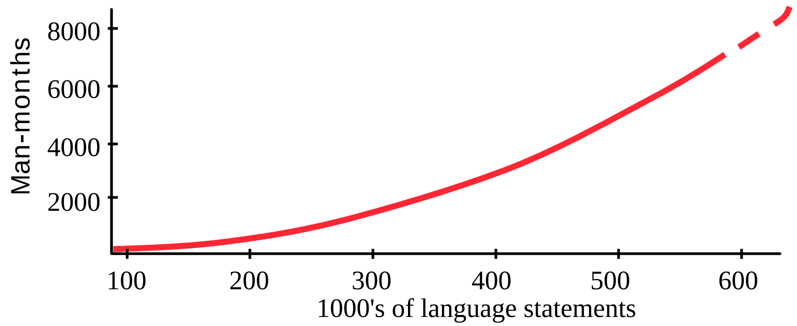
1. Appreciate that developing large software systems is a *complex process*.

2. Know some *techniques* for dealing with the complexity of software development.

3. Understand what is *software engineering* and why it is important in software development.

SOFTWARE IS COMPLEX TO DEVELOP

- Rome: Total War: 3 MLOC
- Boeing 787: 14 MLOC
- F-35 Fighter Jet: 24 MLOC
- Windows 7: 39 MLOC
- Windows Vista: 50 MLOC
- Facebook: 61 MLOC
- Mac OS X 10.4: 86 MLOC
- Luxury passenger car: 100 MLOC



Development effort is not linear with respect to amount of code!

☞ **Software evolution (maintenance) is also complex and usually takes up the majority of the time of software developers.**

Source: http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

SOFTWARE COMPLEXITY COMES FROM …

- **Application domain**
  - The problems are often *very complex*.
  - The developers are usually ________________
- **Communication among stakeholders (clients, developers)**
  - The stakeholders use different _________:
    domain experts ⇔ developers ⇔ developers.
  - The stakeholders have different background knowledge.
  - Human languages are inherently ambiguous.
- **Management of large software development projects**
  - Need to divide the project into pieces and reassemble the pieces.
  - Need to coordinate *many different* _____ and *many different* ______.
- **Coding software**
  - Creating useful software is a *complicated engineering process*.

SOFTWARE COMPLEXITY LEADS TO ...

1. Software ______ problems
   - unreliable → ARIANE 5 rocket
   - unsafe → London Ambulance
   - abandoned → London Stock Exchange
   - inflexible → hard to change/maintain

For large software projects:
- 17% company threatening
- 45% over budget
- 7% over time
- 56% deliver less value

Source: McKinsey & Company in conjunction with the University of Oxford (2012).

2. Software __________ problems
   - Over schedule and over budget *by an order of magnitude!*
   - Does not meet user requirements.
   - Development of *working code* is slower than expected.
   - Progress is *difficult to measure*.

SOFTWARE COMPLEXITY LEADS TO … (cont'd)

Ariane 5 Its maiden flight on June 4, 1996 ended in the launcher being blown up because of a chain of software failures [textbook section 3.1; https://en.wikipedia.org/wiki/Ariane_5#Notable_launches].

London Ambulance Because of a succession of software engineering failures, especially defects in project management, a system was introduced that failed twice in the autumn of 1992. Although the monetary cost, at “only” about £9m, was small by comparison with other examples, it is believed that people died who would not have died if ambulances had reached them as promptly as they would have done without the software failures [http://www.wired.com/2009/10/1026london-ambulance-computer-meltdown/].

London Stock Exchange Taurus was a planned automated transaction settlement system. The project was canceled in 1993 after having lasted more than five years and costing around £75m; the estimated loss to customers was around £450m; the damage to the reputation of the LSE was incalculable [http://www.wired.com/wired/archive/1.03/eword.html?pg=8].

SOFTWARE DEVELOPMENT PROBLEMS

- **Hershey** – SAP customer order system
  - could not fulfill customer orders for candies around Halloween
  - US$100 million inventory failed to ship
  - 8% fall in stock price
- **Nike** – enterprise resource planning (ERP) system
  - system manufactured wrong running shoe
  - US$100 million in lost sales
  - class action lawsuits
- **Government of Canada** – gun registry database
  - initial cost estimate CA$250 million; final cost over CA$1 billion
  - did not do what it was supposed to
  - communication of requirements a major issue

Source: https://vimeo.com/6478347

DEALING WITH COMPLEXITY: DESIGN GOALS

There are many desirable software ______ *characteristics*:

correct	efficient	evolvable	interoperable	maintainable
portable	productive	reliable	repairable	reusable
robust	timely	usable	verifiable	visible

It is often impossible (or unnecessary) to achieve all of them simultaneously (time / cost / conflicting / not important).

☞ **Need to clearly understand the client's design goals!**

☞ **Need to prioritize the design goals (qualities) for a given project and base the development around these.**

Having clear design goals reduces the complexity of designing the system!

DEALING WITH COMPLEXITY: DESIGN GOALS

correct - behaves according to the specifications of its functions (functionally correct)

efficient - in use of time, space, etc.

evolvable - if software can be easily changed over time

interoperable - if software can co-exist and cooperate with other hardware/software systems

maintainable - if software can be easily fixed/changed *after* implementation

portable - if software can run in different hardware/software environments

productive - efficiency of the software production process

reliable - probability that the software will work as expected over a specified time interval

repairable - if defects can be easily corrected with limited/reasonable effort

reusable - if we can reuse parts, perhaps with minor changes

robust - if it behaves "reasonably" in unanticipated circumstances

timely - ability to deliver a product on time

understandable - if it is easy to figure out what is going on/being done

usable - if human users find it easy to use (user interface aspect very important)

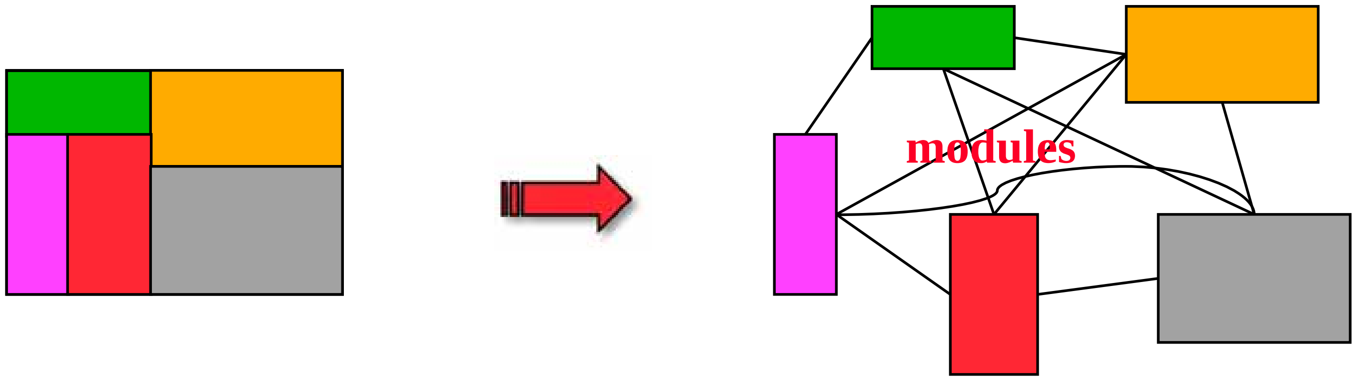
verifiable - if properties can be easily checked (e.g., correct, efficient, etc.)

visible - if all steps/current status are available and easily accessible for external examination

DEALING WITH COMPLEXITY:
MODULAR & INCREMENTAL DEVELOPMENT

There is a limit to human understanding.

DIVIDE AND CONQUER



module: A part of a system that it makes sense to consider separately.

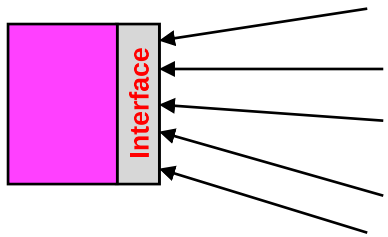
☞ **BUT modules need to _______ with each other.**

DEALING WITH COMPLEXITY: MODULAR & INCREMENTAL DEVELOPMENT (cont'd)

There is a limit to human understanding.

USE INFORMATION HIDING

☞ Allow modules to interact *only* via interfaces.



☞ An interface ________ and ____________ a module thereby providing information hiding.

DEALING WITH COMPLEXITY: MODULAR & INCREMENTAL DEVELOPMENT (cont'd)

- An interface abstracts a module so the developer does not have to know *how* the module is implemented to use it.

  ☞ **A module can be used by understanding *only* its interface.**

This reduces the complexity of understanding the system!

- An interface encapsulates a module so the developer cannot use knowledge about how the module is implemented.

  ☞ **A module can be changed (internally) without affecting the rest of the system.**

This reduces the complexity of maintaining the system!

DEALING WITH COMPLEXITY: MODULAR & INCREMENTAL DEVELOPMENT (cont'd)

- Modular and incremental development using interfaces allows for:
  - more productivity in team development
  - fewer bugs in system development
  - more maintainable software
  - more reusable software

This reduces the complexity of cost and time estimates for developing the system!

DEALING WITH COMPLEXITY:
TRAINING SOFTWARE ENGINEERS

What do engineers do?

DEALING WITH COMPLEXITY: TRAINING SOFTWARE ENGINEERS

"programming-in-the-small" → coding

"programming-in-the-large" → software engineering

A software engineer needs to be able to:

- talk with users in terms of the application.
- translate vague requirements into precise specifications.
- build models of a system at different levels of abstraction.
- use and apply several software development processes.
- choose among design alternatives (i.e., make design tradeoffs).
- work in well-defined roles as part of a team.

This reduces the complexity of building the system!

SOFTWARE ENGINEERING IS …

"The establishment and use of sound **engineering principles** in order to obtain **economically** software that is **reliable** and works **efficiently** on **real machines**."

— Frtiz Bauer

"… **multi-person** construction of **multi-version** software."

— Dave Parnas

- engineering principles → It is a __________ development effort.
- economically…reliable…efficiently → It has _____________.
- real machines → It solves a ________________ (implied).
- multi-person → It requires a __________.
- multi-version → It is _______________ development effort.

SOFTWARE ENGINEERING IS …

- engineering principles → It is a disciplined development effort that:
  - systematically uses appropriate methodologies, techniques and tools, and a store of relevant knowledge, architectures and components
  - applies software engineering techniques, such as system specification, design, modular development and evolution, to system development
- economically…reliable…efficiently → It has built-in quality that:
  - meets cost, time and other constraints using sound project management
  - has meaningful quality assurance (e.g., standards)
  - does formal testing of modules and the system as a whole
- real machines → It solves a real user problem (implied), therefore:
  - development is focused on meeting user requirements
- multi-person → It requires a team effort that:
  - considers team organization, dynamics, and management
- multi-version → It is not a "one-time" development effort, therefore, we:
  - need to plan for software evolution (maintenance)
  - have excellent documentation to facilitate software evolution

SOFTWARE ENGINEERING INVOLVES …

- **A modeling activity**
  - **requirements model** → models the user requirements
  - **solution model** → models the system to be built

Need to match!

- **A problem solving activity**
  - We search for an appropriate solution ***in the presence of change***.
  - Therefore, it is not algorithmic, but it should be systematic.

- **A knowledge acquisition activity**
  - Not a linear process → learn as you go, but ***may need to unlearn.***
  - Sometimes, you may even ***need to start over!***

- **A rationale management activity**
  - Our assumptions and solutions change constantly due to …
  - May need to revisit decisions → bugs, technology, etc.
  - We need to remember: ***Why did we make this choice?***

INTRODUCTION: SUMMARY

- Dealing with **software development complexity** requires:
  - having appropriate design goals.
  - using modular and incremental development techniques.
  - using effective software engineering techniques.

- From a technical viewpoint, **software development** consists of:
  - **software engineering** (i.e., modeling and documenting system requirements and solutions—**“programming-in-the-large”**), *and*
  - **coding** (i.e., building the system—**“programming-in-the-small”**).

While coding is an important software development activity,
software engineering is ***ESSENTIAL to help reduce complexity***
and build high quality, more maintainable software systems!