

# Software Engineering

## Introduction

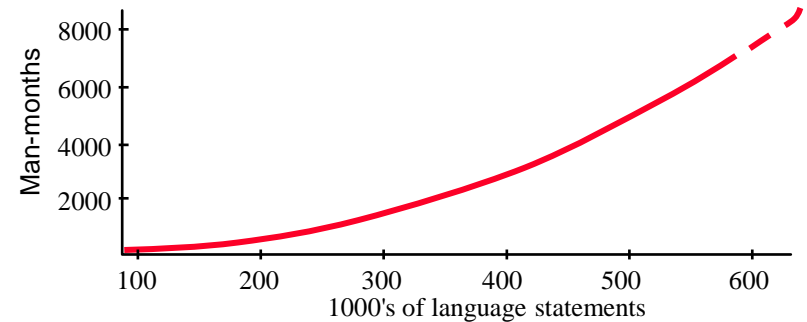


# LEARNING OBJECTIVES

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: ~40MLOC
- Windows 10: ~50 MLOC
- Facebook: 61 MLOC
- Mac OS X: ~90 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 *not domain experts*.
- **Communication among stakeholders (clients, developers)**
  - The stakeholders use different **vocabulary**:  
domain experts  $\Leftrightarrow$  developers  $\Leftrightarrow$  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 parts* and *many different people*.
- **Coding software**
  - Creating **useful** software is a *complicated engineering process*.

# SOFTWARE COMPLEXITY LEADS TO ...

## 1. Software quality 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 development 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*.

# DEALING WITH COMPLEXITY: DESIGN GOALS

There are many desirable software quality 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!

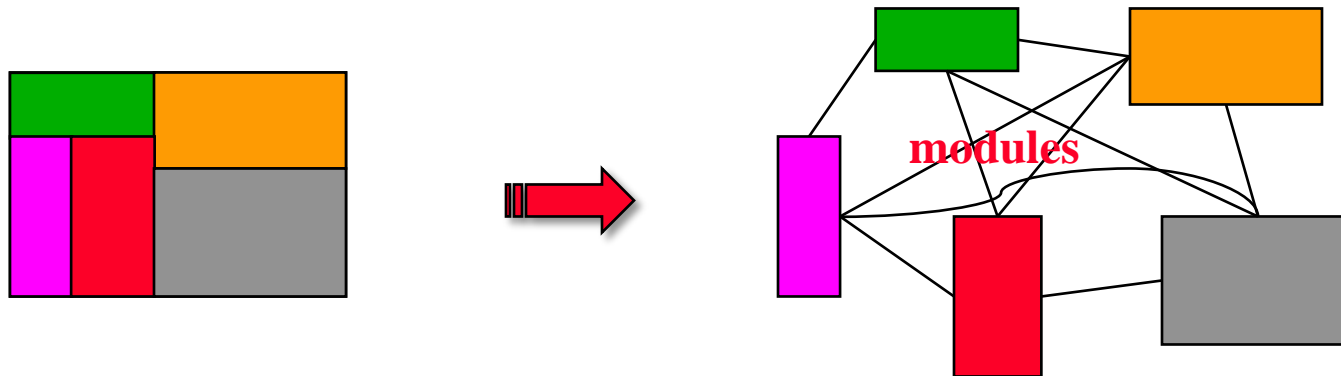
👉 Need to prioritize the design goals (qualities) for a given project and base the development around these.

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

# DEALING WITH COMPLEXITY: 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.

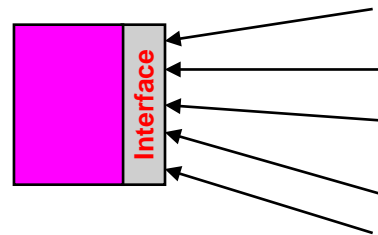
👉 **BUT** modules need to **interact** with each other.

# 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 abstracts and encapsulates 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
- resulting in more predictable software development.

**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!**

# WHAT IS SOFTWARE ENGINEERING ANYWAY?



**I'm Chris**

# 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.”

— Fritz Bauer

“... multi-person construction of multi-version software.”

— Dave Parnas

- engineering principles → It is a disciplined development effort.
- economically...reliable...efficiently → It has built-in quality.
- real machines → It solves a real user problem (implied).
- multi-person → It requires a team effort.
- multi-version → It is not a “one-time” development effort.

# 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 → bugs, technology, etc.
- Thus, we may need to revisit decisions.
- Important 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!**

# COMP 3111 SYLLABUS

- ✓ **1. Introduction**
- 2. Modeling Software Systems using UML
- 3. Software Development
- 4. System Requirements Capture
- 5. Implementation
- 6. Testing
- 7. System Analysis and Design
- 8. Software Quality Assurance
- 9. Managing Software Development