# Software Engineering I CSC-382


**Lecture 1**



# Software Engineering I CS-382

- Instructor:
  - Dr. Michael Farmer
  - Office Hours:
    - 214a MSB, Mon. & Wed. 1:00-2:00, Tuesday 11:00-12:00.
  - Phone:
  - Email: farmerme@umflint.edu
- Book:
  - *Software Engineering: A Practitioner's Approach, 6/e, by Roger Pressman.*
- Tools:
  - *Microsoft Visio or some other drawing package (Smart Draw, etc.)*

## Software Engineering I CS-382

- Grading:
  - Homework (25%), Mid-term 1 (25%), Project (25%), Final (25%)
    - Note you probably cannot pass the class if you do not do the home works!!!!
  - Class participation is also an expected portion of the course work
- Attendance:
  - This course is taught in our new multi-media classroom. Therefore if you miss some classes you can easily review them online. Also if you wish to take the class completely online please see me to get permission
  - Otherwise regular class attendance is expected.



## You Can Watch the Video on:

- http://mediasitelx.csesp.umflint.edu/mediasite/viewer

## Approach to Class

- Format will be:
  - Lectures to explain techniques and methods
  - On-going in-class case study to try our hand at the techniques
    - This is where class participation is key
  - Best way to learn Software Engineering is by doing so there will be a significant project



## Project

- Project will be based on an application defined by the project team
  - Each team will consist of 3-4 people
    - Good number to divide work but simplify communications
  - Team will define project and write a high level customer needs statement as the proposal.
  - Team will then execute the stages of analysis and design defined in class to refine requirements and propose a solution to the problem.
  - The project write-up and final presentation will provide the details of the requirements and the corresponding design

## Comments on Homework Deliverables

1. Provide context to the reader
   - For any system you diagram, you should provide a paragraph describing the system and its objectives
   - Helps me understand your diagrams to know if they are representative and where they may be improved



## Comments on Homework Deliverables II

2. Use diagramming techniques and templates from class
   - Remember a picture is worth 1000 words
   - Also follow the formats, e.g. a Hatley-Pirbai system context diagram should have the background labeling of the I/O, GUI, etc.

## Comments on Homework Deliverables III

3. Provide a professional looking document
   - Typed in MS Word, diagrams generated in Word, Paint or Visio - NO HAND-WRITTEN ASSIGNMENTS WILL BE ACCEPTED
   - In 1-2 years you will be presenting to your boss instead of me and I want you to be ready to make a good impression (for yourself and for UM-Flint)
   - Rule of thumb for outside class prep time: minimum of 3 hours/credit hour

     => **9 hours of home preparation for a 3 credit class per week**



## How This Class is Different

- To date most of your classes have required you to write many lines of code
  - Many late nights spent searching for syntax errors, etc.
- In the algorithms classes you have learned the detailed design of some common algorithms
  - Learned the cost in terms of memory and speed of many standard tasks
- In general in each of these classes your approach was probably very coding-centric
  - Work on the PC trying to get the homeworks to function properly

## How This Class is Different II

- Now we will step far back and learn how to develop engineering models of software systems
  - These models will ultimately make the coding of the system much easier.
    - For very large systems coding would be impossible without this.
  - The goal of the class is to evolve you from programmers to software engineers.
  - We will learn the modeling techniques needed to accomplish this.
  - Our final products are graphical models and high level text descriptions



## Software Engineering CS-382/383

- The 2-class sequence of 382/383 provides you with a complete toolset for working on and managing a large software project
- Overview of CS-382:
  - CS-382 addresses the **methods and techniques** associated with the key stages of engineering a software system.
    - The objective is to provide the students with a set of *'tools'* to methodically and professionally engineer a software product.

## Software Engineering CS-382/383

- Overview of CS-383:
  - CS-383 addresses the aspects of **management and control** of the development of a software product.
    - The objective of this second class is to provide the student with the knowledge of a variety of process models within which to apply the tools and techniques developed in 382.
    - It also will provide the student with the skills to manage and document a software development.



## CS-382 Syllabus Topics/Chapters

- Overview
  - Overview of Software and Software Engineering (Ch. 1)
  - Concepts and Principles for Software Engineering (Ch. 5)
- Software Requirements Analysis
  - Systems Engineering (Software in a system context) (Ch. 6)
  - Requirements Engineering (Ch. 7)
  - Analysis Modeling (Ch. 8)

# CS-382 Syllabus Topics/Chapters II

- Software Design
  - Design Engineering (Ch. 9)
  - Architectural Design (Ch. 10)
  - Component-level Design (Ch. 11)
- Software Test
  - Software Testing Strategies (Ch. 13)
  - Software Testing Techniques (Ch. 14)



# Proposed Class Schedule

| Week | Monday | Wednesday |
|---|---|---|
| (1/7) | Overview & Introduction (Ch. 1 ) | Concepts and Principles of Software and Software Engineering (Ch. 1 & 5) |
| (1/14) | Systems Engineering (Ch. 6) | Requirements Engin: (Ch. 7.1-7.5) |
| (1/21) | *Martin Luther King Day (no class)* | Requirements Engin: (Ch. 7.6-7.9) |
| (1/28) | Analysis Modeling (Ch. 8.1, 8.2, & 8.6) | Analysis Mod. CRC Cards (Ch. 8.4 & 8.7) |
| (2/4) | Analysis Mod.: Class models (Ch. 8.7) | Analysis Mod.: Behavior models (Ch. 8.8) **Project Proposal Due** |
| (2/11) | Design Engin.: Overview (Ch. 9.1/9.2) | Design Engineering: OO-methods (Ch. 9.3/ 9.4) |
| (2/18) | *Review for Exam* | **Mid-term** |
| (2/25) | *Spring Break* | *Spring Break* |

Tentative and subject to change

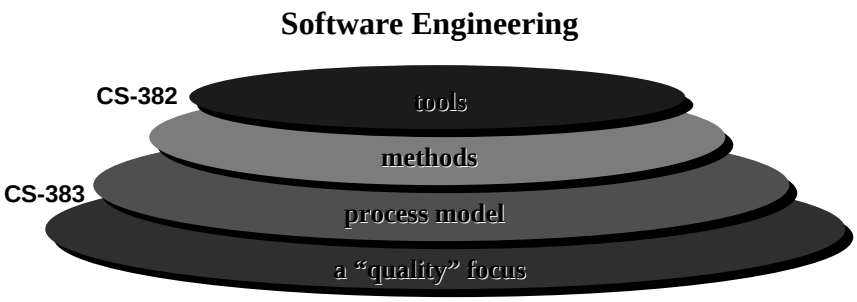
# Proposed Class Schedule II

| Week | Monday | Wednesday |
|---|---|---|
| (3/3) | Architectural Design (Ch. 10.1&10.2) | Architectural Design (Ch. 10.3 & 10.4) |
| (3/10) | Architectural Design (Ch. 10.5 & 10.6) | **Mid-term Project Reviews** |
| (3/17) | Component-level Design (Ch. 11) | Component-level Design (Ch. 11) |
| (3/24) | Component-level Design (Ch. 11) | Component-level Design (Ch. 11) |
| (3/31) | Software Testing Strategies (Ch. 13) | Software Testing Strategies (Ch. 13) |
| (4/7) | Software Testing Techs. (Ch. 14) | Software Testing Techs. (Ch. 14) |
| (4/14) | **Project Presentations** | **Project Presentations** |
| (4/21) | *Class Review for Final* | |

Tentative and subject to change



# Software Development is a Layered Technology



# The Five Critical Stages in Software Engineering

**Communication**
-Project initiation
-Requirements gathering

**Planning**
-Estimating
-Scheduling
-Tracking

**Modeling**
-Analysis
-Design

**Construction**
-Coding
-Testing

**Deployment**
-Delivery
-Support

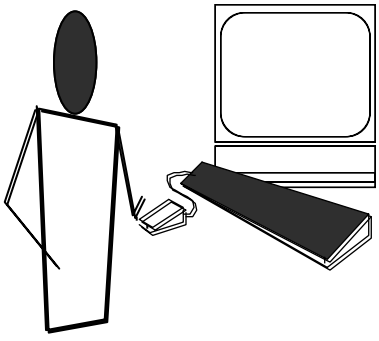
**Elements in yellow are covered in CS 383. CS 383 also covers discusses the variety of ways these core stages can be sequenced, managed, and documented.**
**Note: Coding is not covered here (already well covered in CS 175 & 275)**
**Coding is typically only 10% of a software engineering effort !!**



# What is Software?

Software is a set of items or objects that form a "configuration" that includes

- programs
- documents
- data ...



## Example Software Applications

- System software
- Application software
- Engineering/scientific software
- Embedded software (real-time software)
- Product-line software
- WebApps (Web applications)
- AI software
- New categories
  - Ubiquitous computing—wireless networks
  - Net-sourcing—the Web as a computing engine
  - Open source—"free" source code open to the computing community (a blessing, but also a potential curse!)



## Software's Dual Role

- Software is a product
  - Delivers computing potential
  - Produces, manages, acquires, modifies, displays, or transmits information
- Software is a vehicle for delivering a product
  - Supports or directly provides system functionality
  - Controls other programs (e.g., an operating system)
  - Effects communications (e.g., networking software)
  - Helps build other software (e.g., software tools)

## What is Unique about Software?

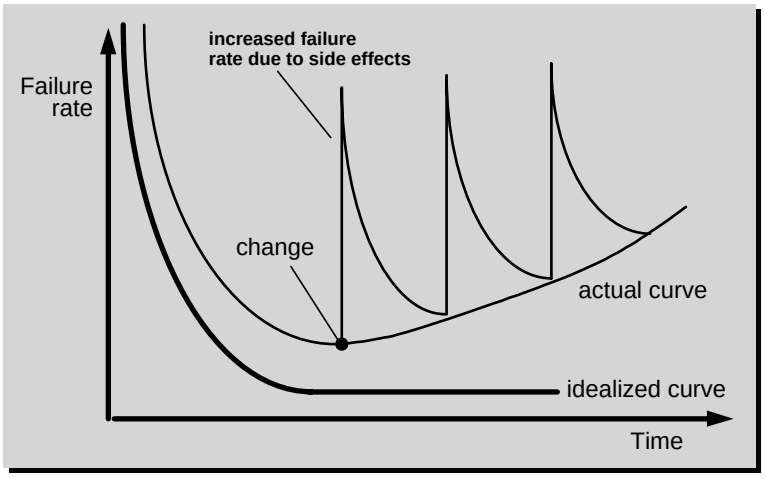
- Software is a labor-intensive,engineered product
  - Nearly the entire cost of software is due to the engineering labor required to develop it
  - Hardware systems have cost of materials, manufacturing facilities, etc. as part of the final product cost



## What is Unique about Software? II

- Software doesn't wear out
  - Hardware systems all have a finite expected life due to the natural wear-and-tear of the physical components
  - Software does tend to deteriorate due to continuous 'improvements'
- Ultimately Software is often custom-made
  - "Each new software project carries within its structure some intrinsic properties that make it unique…" – Ewusi-Mensah

## Wear vs. Deterioration






## Two Types of Software Development

- New software product development
  - May be new to the world product or a new means of providing an existing product need
  - Fewer of these encountered but provide the developer a relatively 'clean sheet of paper' for the development
- Maintenance/Modification/Enhancement of an existing software application
  - The bulk of the cost – 60-80% is in the ongoing lifecycle maintenance of software after it is delivered to the customer

## Legacy Software

***What is legacy software ?***

- Software that currently exists to solve a previously existing business application and now must be modified.



## Legacy Software II

***Why must it change?***

- Software must be ***adapted*** to meet the needs of new computing environments or technology.
- Software must be ***enhanced*** to implement new business requirements.
- Software must be ***extended to make it interoperable*** with other more modern systems or databases.
- Software must be ***re-architected*** to make it viable within a network environment.

## Software Evolution

- The Law of ***Continuing Change*** (1974): E-type systems must be continually adapted else they become progressively less satisfactory.
- The Law of ***Increasing Complexity*** (1974): As an E-type system evolves its complexity increases unless work is done to maintain or reduce it.
- The Law of ***Self Regulation*** (1974): The E-type system evolution process is self-regulating with distribution of product and process measures close to normal.

**Source: Lehman, M., et al, "Metrics and Laws of Software Evolution—The Nineties View,"** ***Proceedings of the 4th International Software Metrics Symposium (METRICS '97),*** **IEEE, 1997, can be downloaded from: http://www.ece.utexas.edu/~perry/work/papers/feast1.pdf**



## Software Evolution II

- The Law of ***Conservation of Organizational Stability*** (1980): The average effective global activity rate in an evolving E-type system is invariant over product lifetime.
- The Law of ***Conservation of Familiarity*** (1980): As an E-type system evolves all associated with it, developers, sales personnel, users, for example, must maintain mastery of its content and behavior to achieve satisfactory evolution.
- The Law of ***Continuing Growth*** (1980): The functional content of E-type systems must be continually increased to maintain user satisfaction over their lifetime.

## Software Evolution III

- The Law of ***Declining Quality*** (1996): The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.
- The ***Feedback System Law*** (1996): E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base.



## What is Software Engineering?

- Simply stated: **the application of engineering to software**
- Was born in 1968
  - At the NATO conference to address the growing 'software crisis' of late and over-budget large projects
- Parnas' definition (1978)
  - "multi-person construction of multi-version software"
- IEEE Std 610.12-1990 definition:
  - The application of a ***systematic***, ***disciplined***, and ***quantifiable*** approach to the development, operation, and maintenance of software.

## Why Study Software Engineering?

- In 1968 a crisis in late and over-budget projects lead to the birth of the domain. ***Has it improved?***
- In 1985 ***32%*** of all corporate IT projects in the US were abandoned before completion
  - Resulted in loss of $81 Billion (**1998 study had similar results** (32% of $250 Billion total expenditure on US IT developments
    - in some cases caused the bankruptcy of the company



## Why Study Software Engineering? II

- In 2000 the worldwide budget for software development was $800 Billion.
  - Clearly the need for developing means to contain this crisis will continue.
  - The high value-added tasks of systems analysis and design will always be in demand and not susceptible to out-sourcing
    - The methods taught in this class will differentiate you from the large supply of 'programmers'. You will be software systems engineers and analysts

## Software Myths

- Affect managers, customers (and other non-technical stakeholders) and practitioners
- Are believable because they often have elements of truth,

*but …*

- Invariably lead to bad decisions,

*therefore …*

- Insist on reality as you navigate your way through software engineering



## Some Favorite Software Myths

- ***If project is behind schedule, then add more developers***
  - Reality: Adding people late in a project actually makes the project later
- ***Project requirements continually change, but change is easily accommodated because software is flexible***
  - Reality: Changes identified early are easy, but changes late in the development can be catastrophic

## Some Favorite Software Myths II

- ***Once we write the program and get it to work our job is done***
  - Reality: The sooner you start coding the longer it will take since you do not yet fully understand the requirements
- ***The only deliverable work product is the code***
  - Reality: There are many levels of documentation that are critical to the ongoing and future success of a project



## For Next Class

- Read Chapter 1 and Chapter 5