CSE 435: Software Engineering

Dr. B. Cheng

chengb at msu dot edu

Office Hours: M, W: 4:20-5:00 pm, or by appointment

TAs: Kira Chan; Nick Polanco,

Tues, Thurs: 5-6 pm; Fri: 2-3 pm, or by appt.

{chanken1, polanco3} at msu dot edu

FYI

- Professor in Computer Science and Engineering

- Here at MSU for > 20 years
  - Software Engineering and Network Systems (SENS) Lab
  - Digital Evolution (DEVOLab)
  - BEACON: NSF Science and Technology Center ("Evolution in Action")
  - MSU/NSU Sociomobility REU

- Research and Instruction areas:
  - High-assurance systems
  - Model-driven engineering
  - Autonomic (self-adaptive) systems
  - Automotive Cybersecurity
  - Evolutionary-based computing
  - Recently, also working in following areas:
    - Trusted AI/Machine Learning
    - Model-Driven Engineering for Sustainable Systems (e.g., smart grid)
  - Work extensively with industrial collaborators (e.g., Ford, GM, Aerospace Corp., Continental Automotive, Motorola, Dataspeed, Groundspeed, BAE Systems, Siemens); NASA
  - International collaborations (sustainability, uncertainty interaction, SE4AI)
High-Assurance Autonomic Computing

- **Autonomic computing [2001]:** Promises self-managed and long-running systems with limited human guidance.
- Systems must continue to **operate correctly** during exceptional situations, upgrades, and evolution under **uncertain** conditions.
- Need for assurance:
  - hardware component failures
  - network outages
  - software faults
  - security attacks

New Scale

High-Assurance Cyberphysical Systems

*Intelligent Transportation and Vehicle Systems*

Requires increasingly complex systems:
- Thousands of platforms, sensors, decision nodes, complex systems
- Connected through heterogeneous wired and wireless networks.
The future...

Now...
Advanced Driver-Assistance Systems

- Onboard Autonomous Features
  - Safety
  - Convenience
Reality Check


Reality Check

NYTimes, Aug. 18, 2023
Objectives of this course

Introduce **industrial-strength** software development:
- formal processes/artifacts for planning, specifying, designing, implementing, and verifying
- Individual and team-based development
- life-cycle issues and “umbrella” activities

Introduce key foundations underlying these activities
- E.g., requirements engineering
- E.g., software modeling
- E.g., assurance

https://youtu.be/TuY6SUPd5Vk?si=gqVJDPE0_DCsNg22
Acknowledgements

- L. Dillon, M. Heimdahl, M. Langford
- Numerous SE texts and articles
- Industrial collaborators
- Former students who have provided feedback

Overview of Course

- Emphasis on requirements engineering and design
- Learn/apply new techniques for software development
- Learn to work with a group
- Improve technical writing skills
- Become up to date on current trends in SE
- Explore presentation media and techniques
Structure of Course

- (Short) assignments over readings
- In-class assignments (interactive exercises)
- Group projects with industrial customers (requirements document, prototype): modeling and documentation
- Two exams (middle and final)
- Presentations: oral presentations, prototype demos

How “different” is this course from other CSE courses?

Quite!
- Not a “programming course”
- Exercises aim to facilitate problem understanding, solutions, tradeoffs, and sensitivity to challenges that affect industrial software development
- Written and oral communication skills will be exercised, improved, and assessed
- Team work is critical and will be assessed
What is Software Engineering ?

- The study of **systematic** and effective processes and technologies for supporting software development and maintenance activities
  - Improve quality
  - Reduce costs

What is Software Engineering?

- A **broader perspective** on software development
- From **theory to practice**, how do we best realize our software?
  - Personal-use software vs. Systems deployed in the real world
- **Multiple dimensions** of consideration:
  - Algorithmic correctness
  - Computational efficiency
  - Cost of development
  - Maintenance
  - Usability
  - Ethical, privacy concerns

“Software Engineering is the branch of computer science that creates practical, cost-effective solutions to computing and information-processing problems, preferentially by applying scientific knowledge, developing software systems in the service of mankind.”

---

Half a Century of Software Engineering Education: The CMU Exemplar, 2018
Mead, Garlan, and Shaw
**Historical Perspective**

- **1940s**: computers invented
- **1950s**: assembly language, Fortran
- **1960s**: COBOL, ALGOL, PL/1, operating systems
  
  *1969: First conference on Software Eng*
- **1970s**: multi-user systems, databases, structured programming

**Historical Perspective (cont.)**

- **1980s**: networking, personal computing, embedded systems, parallel architectures
- **1990s**: information superhighway, distributed systems, OO in widespread use.
- **2000s**: virtual reality, voice recognition, video conferencing, global computing, pervasive computing...
- **2010s**: EMRs, automated features for vehicles, new security awareness, ...
- **2020s**: self-driving cars, smart grids, AI/LLMs, ...
“The Software Crisis”

- NATO Software Engineering Conferences (1968/1969)
  - A recognition of major software issues
  - Software is often inefficient and of poor quality
  - Projects are running over-budget/over-time

"The major cause of the software crisis is that the machines have become several orders more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem."

E. Dijkstra, 1972 Turing Award

Software Engineering is an effort to manage the software crisis.

Hardware Costs vs Software Costs (% of overall costs)
Why is software so expensive?

- Hardware has made great advances
- But, software has made great advances ...
- We do the least understood tasks in software.
  - When task is simple & understood, encode it in hardware
  - Why?
- Demand more and more of software
  - Consider your cell phone

Size of programs continues to grow

- **Trivial**: 1 month, 1 programmer, 500 LOC,
  - Intro programming assignments

- **Very small**: 4 months, 1 programmer, 2000 LOC
  - Course project

- **Small**: 2 years, 3 programmers, 50K LOC
  - Nuclear power plant, pace maker

- **Medium**: 3 years, 10s of programmers, 100K LOC
  - Optimizing compiler
Size of programs continues to grow

- **Large**: 5 years, 100s of programmers, 1M LOC
  - MS Word, Excel

- **Very large**: 10 years, 1000s of programmers, 10M LOC
  - Air traffic control,
  - Telecommunications, space shuttle

- **Very, Very Large**: 15+ years, 1000s programmers, 35M LOC
  - W2K

- **Ultra-Large Scale**: ? years, ? developers distributed,
  - 1000s of sensors, decision units,
  - heterogeneous platforms, decentralized control
  - Intelligent transportation systems; healthcare systems

Comparison of Software Complexity

- **Codebases**: Millions of lines of code
  (informationisbeautiful.net)

<table>
<thead>
<tr>
<th>Codebase</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple iPhone App</td>
<td>1,000</td>
</tr>
<tr>
<td>Space Shuttle</td>
<td>40,000</td>
</tr>
<tr>
<td>Unreal Engine 3</td>
<td>2,000,000</td>
</tr>
<tr>
<td>US Military Drone</td>
<td>3,500,000</td>
</tr>
<tr>
<td>Google Chrome</td>
<td>6,700,000</td>
</tr>
<tr>
<td>Android</td>
<td>12,000,000</td>
</tr>
<tr>
<td>Facebook</td>
<td>62,000,000</td>
</tr>
<tr>
<td>Modern High-end Automobile</td>
<td>100,000,000</td>
</tr>
<tr>
<td>Google (all internet services)</td>
<td>2,000,000,000</td>
</tr>
</tbody>
</table>

How do we manage millions of lines of code?
High Assurance Systems

- **Safety-critical systems** (power plants, factories, automobiles)
  - **Safety concerns**, prevent events leading to death, injury, or damage property
  - **Fault-tolerance**, must guarantee services even with hardware/software faults
  - **Security needs**, cyberattacks, like ransomware, are a rising concern

> "(Lansing) BWL cyberattack bills reach nearly $2M"  
> **Lansing State Journal, 2016**

How much do you trust software?

Software Lifespans

- **Average software life span** approx. 10 years  
  > **Tamai & Torimitsu, 1990**

- **Geriatric software** still in use:
  - **MOCAS** (1958), Tracks contracts and payments for US DoD
  - **Voyager 1/2** (1977), In continuous operation and still transmitting data!

> "Banks scramble to fix old systems as IT 'cowboys' ride into sunset"  
> **CNBC, 2017**

You will encounter legacy code in your career.
Legacy Software Systems

- Why do they exist?
  - Some systems require *constant availability* (banks, utilities, military, etc.)
  - Can be *expensive to redesign and retrain* personnel
  - "If it ain’t broke, don’t fix it”

- Problems to consider
  - It becomes increasingly *difficult to find hardware and personnel* to support it
  - Can be *vulnerable to new security threats*
  - Can be *difficult to integrate* with newer systems

Can we instead create software that is easier to support continuous improvement?

A Business Perspective

- **Product Life Cycle**
  - Most products follow a common pattern (not just software)
    - Stage 1: *Initial release*
    - Stage 2: *Takeoff*
    - Stage 3: *Waning demand*
    - Stage 4: *Loss of interest*

- **Planned obsolescence** *(Kramer 2012)*
  - A planned expiration date to spur consumer spending
  - *Very common in electronics and software*

What are your ethical responsibilities?
Psychology of Software Engineering

- **Behavior Model** for Software Engineering
  - At the *individual* level, focus is on a specific problem to be solved
  - At the *team/project* levels, group dynamics are the focus (communication/collaboration)
  - At the *company/business* levels, marketing and profitability are the focus
  - Each level has its own understanding of what the problem is and what sort of software should be delivered

> How do we know if we’re delivering the correct solution?

The Stakeholders

- A **stakeholder** is anyone who has an interest in the final product
  - *End Users*: actually interact with the software
  - *Customers*: request and pay for the software
  - *Practitioners*: create and test the software
  - *Technical Managers*: coordinate creation of software
  - *Senior Managers*: manage business affairs

> A successful software product satisfies **all** stakeholders.
Case Study: MiDAS Unemployment System [2018]

- Michigan UIA replaced a 30 year system running COBOL
- $44.4 million and 26 months of effort
- After deployment, accusations of fraud grew 5x
- Fines grew from $3 million to $69 million within a year
- Falsely accused 34,000 unemployed of fraud

"The fiasco is all too familiar: A government agency wants to replace a legacy IT system to gain cost and operational efficiencies, but alas, the effort goes horribly wrong because of gross risk mismanagement.


Poor software processes have the potential to ruin lives.

What are Your Responsibilities?

- Creating software is hard/difficult.
- End users have blind trust in software.
- Businesses often rely too much on “crunch time.”

"Any sufficiently advanced technology is indistinguishable from magic.

As a software engineer, it is your responsibility to educate both end users and management about the realities of software development."
The Importance of a Process

- Well-defined processes help to ensure products satisfy all stakeholders

- **Major activities**
  - **Communication** – actions to determine software requirements
  - **Planning** – actions to determine required resources and work schedule
  - **Modeling** – actions to understand and validated requirements
  - **Construction** – actions to create and test the final product
  - **Deployment** – actions to deliver the product and receive feedback

Software Engineering is a science of refining this process to deliver the best software products.

Summary

- Software development is difficult in practice.
- Stakeholders have an interest in the final product.
- Software engineering is a systematic approach to create high-quality software in a timely manner.
SE: Challenges

Craft vs Engineering

New Scale
Ultra-Large Scale SW-Intensive Systems
The ULS Ecosystem

- Key elements:
  - Computing devices
  - Business and organizational policies
  - Environment (including people)

- Forces:
  - Competition for resources
  - Unexpected environmental changes
  - Decentralized control
  - Demand for assurance
What’s the problem?

- Software cannot be built fast enough to keep up with
  - H/W advances
  - Rising expectations
  - Feature explosion

- Increasing need for high reliability software

What’s the problem?

- Software is difficult to maintain
  - “aging software”

- Difficult to estimate software costs and schedules

- Too many projects fail
  - Arianne Missile
  - Denver Airport Baggage System
  - Therac
Why is software engineering needed?

- To predict time, effort, and cost
- To improve software quality
- To improve maintainability
- To meet increasing demands
- To lower software costs
- To successfully build large, complex software systems
- To facilitate group effort in developing software

Shaw’s model of engineering evolution [Shaw-IEEE-Computer90]
Characteristics: Craft

- Virtuosos and talented amateurs
- Intuition and brute force
- Haphazard progress
- Casual transmission of knowledge
- Extravagant use of available materials
- Manufacture for use rather than sale

- Examples: woodworking, artists

Characteristics: Commercial production

- Skilled crafts
- Established procedure
- Pragmatic refinement
- Training in specific domain (e.g., mechanics-- automotive technicians, structures-- construction worker, electricians)
- Economic concern for cost and supply of materials
- Manufacture for sale

Examples: automotive parts, chip manufacturing
Characteristics: Professional engineering

- Educated professionals
- Analysis and theory
- Progress relies on science
- Educated professional class
- New applications enabled through analysis
- Market segmentation by product variety

Examples: civil engineering (bridges, buildings), automotive engineers (electronics, mechanical engineering)

Evolution of Civil Engineering

1700: Statics, strength of materials

1st century: Romans

1750: Material Properties
1850: Bridge Analysis
**Civil Engineering**

Basis in theory.
- Actually two theories:
  - Statics: composition of forces.
  - Material strength: bending of a beam.
- Theories preceded real CE by 150 years!

Underlying science emerged 1700 years after commercial production evolved!

---

**Software Engineering Evolution**

(circa 1990)

Science: algorithms, logic, databases, languages

1980’s: Development Methodologies

Isolated Examples (5ESS, Shuttle)
Two “pillars” of SE education

Basis in:
- production processes and process frameworks
- rigorous theories addressing design problems that attend to the various phases of these processes

This course:
- organized around first pillar
- structured so that process issues will motivate introduction of theoretical content

Software Engineering Phases

- Definition: What?
- Development: How?
- Maintenance: Managing change
- Umbrella Activities: Throughout lifecycle
Definition

- Requirements definition and analysis
  - Developer must understand
    - Application domain
    - Required functionality
    - Required performance
    - User interface

Definition (cont.)

- Project planning
  - Allocate resources
  - Estimate costs
  - Define work tasks
  - Define schedule

- System analysis
  - Allocate system resources to
    - Hardware
    - Software
    - Users
Development

- Software design
  - User interface design
  - High-level design
    - Define modular components
    - Define major data structures
  - Detailed design
    - Define algorithms and procedural detail

Development (cont.)

- Coding
  - Develop code for each module
  - Unit testing

- Integration
  - Combine modules
  - System testing
Maintenance

- **Correction** - Fix software defects
- **Adaptation** - Accommodate changes
  - New hardware
  - New company policies
- **Prevention** - make more maintainable
- **Enhancement** - Add functionality

Umbrella Activities

- **Reviews** - assure quality
- **Documentation** - improve maintainability
- **Version control** - track changes
- **Configuration management** - integrity of collection of components
Software Engineering Costs

Relative Costs to Fix Errors

This is why software process pays off