CSE 435: Software Engineering

Dr. James Daly
3110 Engineering Building
Tues, Fri: 1:30-2:30 pm or by appt
dalyjame at msu dot edu

TA: Ian Whalen
Mon, Wed: 6:30-7:30 pm or by appt.
whalenia at msu dot edu
• Freshly minted Ph.D.

• Previous courses taught
  ▪ Discrete Mathematics (CSE 260)
  ▪ Data Structures and Algorithms (CSE 331)

• Research areas:
  ▪ Computer network algorithms
    o Packet classification

• Industry Experience
  ▪ MIT Lincoln Laboratory
  ▪ TechSmith
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

![Diagram showing various advanced driver-assistance systems and features](image-url)
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
Overview of Course

- Emphasis on analysis 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 lab assignments (various SE tools)
- Group projects (prototype, analysis, design): modeling and documentation
- Two exams (middle and final)
- Presentations: oral presentations, prototype demos
How “different” is this course?

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
Relation to other courses?

Not a design/programming course (ala CSE 335)
- Much “higher-level” coverage of notations
- More emphasis on process than design methods

Not a capstone design experience (ala CSE 498)
- Smaller, more constrained project
- Smaller teams
- Projects will be industry-based

Ideal pre-capstone course:
CSE 335  ☐  CSE 435  ☐  CSE 498
(coding, design)  ➔  (design, reqts, process)  ➔  (synthesis)

CSE 435: Software Engineering
PAUSE

• Syllabus
• HW1 due next Wednesday
• Survey due Sept 7
What is Software Engineering ???

- The study of *systematic* and effective processes and technologies for supporting software development and maintenance activities
  - Improve quality
  - Reduce costs
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, autonomous vehicles, new security awareness, ...
Hardware Costs vs Software Costs (% of overall costs)

- **h/w costs**
- **s/w costs**

Time
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
New Scale
Ultra-Large Scale SW-Intensive Systems

Healthcare Infrastructure
New Scale

Intelligent Transportation and Vehicle 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
Context: “Sufficient” System Health

High-level Objective:

- How to design a safe adaptive system with incomplete information and evolving environmental conditions

- **Execution environment**
  - How to model environment
  - How to effectively monitor changing conditions
  - Adaptive monitoring

- **Decision-making for dynamic adaptation**
  - Decentralized control
  - Assurance guarantees (functional and non-functional constraints)

- **Adaptation mechanisms:**
  - Application level
  - Middleware level
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
  - Ariane-5 Missile
  - Denver Airport Baggage System
  - Therac-25
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)

Production

Commercial

Craft

Science

Engineering
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

- Maintenance
- Development
- Definition
Relative Costs to Fix Errors

This is why software process pays off