CMPSC 483W
Software Design Methods

Catalog Data:  Software Design Methods (3)
Applications of scientific knowledge and methods in the design and construction of computer software using engineering concepts. Prerequisite: CMPSC 221; CMPSC 465; ENGL 202C.

Typical Textbook:  Frank, Marriot & Warzusen, CSQE Primer, 3rd edition, Quality Council of Indiana, 2002
OR

Course Objectives:  CMPSC 483W introduces the student to modern processes of developing medium-to-large scale software. The student develops an understanding of the procedures and terminology of modern software development, and gets practical experience in the process of developing software in teams. Given high-level requirements, each student will be the member of (an approximately 40-person) team which will develop, implement, test and deploy a solution to the problem, the goal of which is to instill in the student the appropriate “mind-set” for developing in teams as opposed to doing individual projects.

Primary Course Outcomes:  Upon completion of the course, students should possess the following skills:

- Life cycle models: The student can identify and appropriately use various models of team-based software development, including waterfall, rapid prototyping, spiral, incremental, V, object oriented and extreme.
- Terminology: The student can identify and appropriately define the terminology of modern software.
- Algorithm and code analysis: The student will demonstrate his/her ability to analyze algorithms and code for correctness and efficiency through the performance of a detailed design step and reviews of other students’ code.
- Development: The student can develop a modest (on the order of 1,000 LOC) computer program which appropriately fits into the entire design and environment of the class project, including appropriate documentation and testing. This development is performed under appropriate time and budget constraints.
- Writing: The student will be able to write clear and effective technical prose, as demonstrated by the writing of requirements, design documents, release notes and user manual entries.
- Public speaking: The student will be able to speak clearly and persuasively about technical subjects in large and/or small group settings, and use supporting materials effectively. This ability will be demonstrated through one each of a small group and a large group presentation.
- Independent learning: The student will demonstrate the ability to use unfamiliar computer systems, test equipment and software tools by utilizing a software production platform which is not previously known to the student.
- Current trends and research: The student will demonstrate an ability to discuss current trends and research in the industry through participation in the large and small group presentations as both presenter and audience member.
- Team-work: The student will demonstrate an ability to work effectively in multi-disciplinary teams through his/her participation in various group sub-projects which will be incorporated into the large-scale project.
- Ethics: The student will demonstrate an understanding of professional codes of ethics through his/her actions and a written examination.

Relationship to Undergraduate Program Outcomes:  CMPSC 483W is a capstone course which introduces and reinforces the following program outcomes:
• Analyze algorithms or computer code for correctness and efficiency.
• Develop a modest (on the order of 1000 lines of code) software application using appropriate data structures and algorithms.
• Given specifications, design and implement a computer under time and budget constraints.
• Write clear and effective technical prose.
• Speak clearly and persuasively about technical subjects in large and/or small group settings, and use supporting materials effectively.
• Demonstrate independent learning by using unfamiliar computer systems, test equipment, and software tools to solve technical problems.
• Be able to discuss major trends in industry and current research activities within the discipline.
• Demonstrate the ability to work effectively in multi-disciplinary teams. The term multidisciplinary is used here in the broader sense to include teams of computer professionals having different skills; e.g., one team member might be familiar with web development, whereas another might have experience with microprocessor systems.
• Be able to state a code of professional ethics and to identify issues in engineering case studies.

Required Topics:

1) Introduction to software engineering:
   a) Quality philosophies (Deming, Juran, etc.)
   b) Prevention versus detection
   c) Benchmarking
   d) ISO 9000 / Q9000
   e) IEEE Software engineering standards
   f) Capability maturity model
   g) Leadership tools and skills
   h) SWOT analysis
   i) Change management
   j) Knowledge management
   k) Motivation
   l) Team management
   m) Facilitation
   n) Communication
   o) Ethics
   p) Intellectual property
   q) Liability
   r) Training and development

2) Software quality assurance:
   a) Quality principles
   b) Quality policies
   c) Quality tasks: Documentation, review, audit, monitoring, inspecting, participating, corrective action, assisting, testing.
   d) Outsourcing
   e) Subcontractor management
   f) Quality tasks within the development lifecycle
   g) Planning
   h) Tracking
   i) Requirements writing
   j) QA tools
   k) Change management
   l) Software assessment
   m) Cost of quality
   n) Corrective action procedures
   o) Quality improvement processes
p) Defect detection and removal

3) Software audits
   a) Audit objectives
   b) Responsibilities
   c) Auditing standards (ANSI/ISO/ASQC Q10011-1,2-1991)
   d) Audit types
      i) First party, second party, third party, internal, external
      ii) System, process, product, compliance
   e) Audit schedule
   f) Audit tools and procedures
   g) Corrective action follow-up phase

4) Software engineering processes
   a) Phases: analysis, design, code, test, maintain.
   b) Lifecycle models: waterfall, rapid prototyping, spiral, incremental, V, object oriented and extreme
   c) Types of project: adaptive, perfective, reengineering
   d) System architectures
   e) Requirements
   f) Software design methods
   g) Defect prevention and removal
   h) Software development tools
   i) Maintenance
      i) Types
      ii) Maintenance models
      iii) Measures of maintainability
      iv) Maintenance process flow

5) Project management
   a) Planning
      i) Initial requirements
      ii) Scope management
      iii) Estimation and forecasting
      iv) Project definition
      v) Software management plan
      vi) Planning tools
      vii) Cost and value
   b) Tracking and controlling the project
      i) Scheduling
      ii) Budgeting
      iii) Gantt charts
      iv) Cost metrics
   c) Risk management
      i) Estimation
      ii) Planning
      iii) Analysis
      iv) Contingency plans
   d) Release decisions
   e) Security and hazard analysis
      i) Trade-offs
      ii) Code security
      iii) Authentication
      iv) Safety-critical software
      v) Hazard analysis

6) Software measurement
   a) Measurement theory
b) Reliability
c) Validity
d) Measurement error
e) Defect/error/failure
f) Measurement scales
g) Statistics
h) Human factors in software measurement
i) Commonly used metrics
j) Quality attributes
k) Types of software testing
   i) Unit testing
   ii) Integration test
   iii) System test
   iv) Regression test
l) Software reliability models
m) Defect removal effectiveness
n) Program performance analysis
o) Data quality analysis
p) Quality analysis tools
   i) Check sheet
   ii) Pareto diagram
   iii) Histograms
   iv) Graphs
   v) Scatter diagrams
   vi) Control charts
   vii) Cause-and-effect diagrams
   viii) Flow charts
q) Problem-solving approaches
   i) Root cause analysis
   ii) Plan-do-check-act
   iii) Interrelationship diagram
   iv) Tree diagram
   v) Prioritization matrices
   vi) Process decision program charts
   vii) Activity network diagrams
r) Sampling theory
7) Verification and validation
   a) Theory
   b) Procedures
      i) Structural analysis
      ii) Dynamic tests
      iii) Mathematical proof
      iv) Simulation
c) V&V planning
d) V&V evaluation
   i) Traceability
   ii) Documentation
   iii) Source code
   iv) Test results
   v) Audit results
e) Review and inspection methods
   i) Desk checking
   ii) Walkthroughs
   iii) Inspections
f) Defect classification
g) Types of tests
i) Functional
ii) Performance
iii) Regression
iv) Environmental load
v) Worst case
vi) Perfective
vii) Exploratory
viii) Random-input
ix) Certification
x) Stress
xi) Usability
xii) Real-time response

h) Approaches
i) Top-down
ii) Bottom-up
iii) Black box
iv) White box
v) Simulation
vi) I/O
vii) Beta
viii) Fault insertion
ix) Fault error-handling
x) Equivalence-class partitioning
xi) Boundary value analysis
xii) Cause-effect
xiii) Customer defect reports

i) Testing environments
i) Stubs/drivers
ii) Harnesses
iii) Equipment compatibility test labs
iv) Simulators

j) Test plans
i) System test plans
ii) Acceptance test plans
iii) Validation test plans

k) Scheduling
i) Prioritization
ii) Freezing

l) Documentation
i) Defect recording
ii) Defect tracking
iii) Test report completion metrics
iv) Trouble reports
v) Logs
vi) Test designs
vii) Test cases
viii) Test procedures
ix) Test reports

m) Code coverage metrics

n) Severity measures

8) Configuration management
a) Rationale
i) Simultaneous update
ii) Double maintenance
iii) Shared code
iv) Version control
b) Functional areas of SCM
   i) Management and planning
   ii) Configuration identification
   iii) Configuration control
   iv) Configuration status accounting
   v) Configuration verification & audit

c) Library/repository processes
   i) Version control
   ii) Configuration support
   iii) Security
   iv) Reporting
   v) Build support
   vi) Release management
   vii) Customization support
d) Build processes
e) Release processes
f) Change processes
   i) Baseline
   ii) Impact assessment
   iii) Initiation
   iv) Classification
   v) Evaluation
   vi) Implementation
   vii) Verification
g) Models of concurrent development
h) Traceability
   i) Software configuration audit
   j) Control of software libraries
   k) Documentation control
   l) Configuration audits
   m) Packaging

Class Format: Two 75-minute lecture/labs per week.

Professional Component: The student will learn the concepts and nomenclature of a professional software engineering environment, how to work and communicate with other software professionals, how to write software to specification, how to test the quality of software, and how to document and release software products.

Evaluation: Students should be evaluated both on their knowledge of the topic areas (50%) and their ability to work within a simulated professional environment (50%).

Author: Steve Shaffer
Last Revised: July 25, 2007