CSS 360: Software engineering – A Survey Course

Dr. Frank Cioch (Sigh-oach)

Email: [cioch@u.washington.edu](mailto:cioch@u.washington.edu)

Office: UW2-310

Phone:

2 midterm exams, 1 final exam

Final exam is comprehensive

Course structure:

Intro

SDLCs

Requirements engineering

Quality Assurance

Project Management

Agenda:

Introduction (1-2 weeks): Chapters 1, 5

Process (2 weeks): Chapters 2, 3, 4

Exam 1

Requirements (2 weeks): Chapters 6,7,8

Design (2 weeks): Chapters 9,10,11

Exam 2

Quality: Chapters 13, 14, 15

Management: Chapters 21, 22

Wrap-up: Chapter 32

Assignment:

Write a 3-5 single-space page (1-inch margins Times New Roman 11pt font) paper describing your current understanding of the process of software engineering and the value it brings. Do not use endnote citations religiously; most of the content should be a paraphrase. Write this as a student, not a professional; explain clearly the subjects and make sure they are comprehensible to the layman.

Title “What is software engineering?”

Due: April 4th, 2007

Rubric:

Looking for a sense that the writer internalized the material

Correct content (high points; good outline)

Subject is explained simply and clearly with good examples.

Clear and professional formatting (use lists)

Use 4 sources below:

Required sources:

Textbook (Pressman)

Wikipedia: software engineering

Software Engineering Body of Knowledge: <http://www.swebok.org>

Choose one more source from the list below (Google the name and ‘software engineering’):

Pressman

Ian Sommerville

SEI – Software Engineering Institute at CMU

IEEE Software

Steve McConnell

Barry Boehm

**April 2, 2007**

Chapter 5: Practice: A Generic View

Practice, the software development lifecycle:

the “how-tos” of the software development process; techniques for building high-quality software

* Effective communication (collaboration between developers and other stakeholders (other developers, business/systems analysts, customers, QA folk/testers, project managers))
* Planning (management)
* Modeling (analysis, design, process)
* Construction (coding and testing)
* Deployment (delivering, support, feedback)

Key Concepts (OOP/information-hiding)

Principles:

Methods:

Pair-programming

Unified process

Agile modeling

Process models

SDL

Tools:

Languages

IDEs, etc.

**April 4, 2007**

Take-home exam, due April 16.

Covers chapters 2-4 (pages 46-94).

Questions similar to handout.

Chapter 2: Assessing process

* How do you measure an organization’s competence in building quality software?
  + Product has high value to customers
  + Product fulfills functional and quality specification requirements
  + Product is delivered on time
  + Product is delivered within budget
* How can an organization improve their competence?

Chapter 3: SDL models

* Waterfall
* Incremental/iterative (IID)
  + Agile
  + Rapid Application Development (RAD) James Martin
* Evolutionary
  + Evolutionary prototypes
  + Spiral
* Unifed Process
  + Rational Unified Process/IBM Unified Process

Chapter 4: Agile Processes: different from plan-based

* What?
* How?
* Why?
* Agile Process Models:
  + Extreme Programming (XP)
  + Scrum
  + Agile Modeling

New! Not covered in the textbook! Hybrid models

* Agile Unified Process
* Open Unified Process

**April 9, 2007**

Watts Humphrey – Capability Maturity Model (CMM0 (mission-critical software)

The results of software failure:

<http://catless.ncl.ac.uk/Risks/>

Models:

Generic (can be mapped to any other model):

* Communication
* Planning
* Modeling
* Construction
* Deployment

Waterfall (big-design up-front), phased, sequential, milestones, document-driven, customer feedback at milestones

1. Requirements (output: Requirements spec)
2. Design (output: Design spec)
3. Implementation (output: code)
4. Verification (output: test pass results)
5. Maintenance (output: )

* Sequential model.
* Each phase feeds a work product (output) into the next phase, which receives it as input
* 1 milestone/phase
* Customer gets feedback at the end of every milestone
* Document-driven
* Big-design up-front
* Does not handle change well

Improvements (modifications):

* Throwaway prototype in requirements phase to reduce number of customer design change requests
* Overlap the phases (sashimi). Start one phase while the other is not yet completed.
* Break the project up into large functional parts and iterate ‘n’ waterfalls.
* Additional information: “Rapid Development: Taming wild Software Schedules”. Microsoft Press. ISBN 1-55615-900-5. Steve McConnell

Unified Process (Iterative and Incremental Development, not Sequential; produces code + documents (work artifacts):

* Iterative: all development activities are repeated in phases, with the weights of the activities changing in each phase.
* Use-case-driven and iterative
* Guided by customer “use cases”
* “Draws in the net” around the featureset in multiple iterations
* Vision/preliminary requirements based on key use cases/prototypes
* Big-design up-front
* Requirements up-front
* Document-heavy
* Use key “use cases” to enumerate and identify risks (customer feedback, technical risks, process risks)
* Craig Larman – Valtech

Spiral: cyclical, incremental, and evolutionary:

* Cyclical instead of iterative: different activities are performed at different points in the cycle (instead of repeated iterations of the same development activities).
* Different SDLs can be uses in each cycle
* Each cycle features a prototype on which the next version is built
* Risk-driven (can abort at the end of any spiral)
  + Actual (as opposed to perceived) customer requirements can increase risk
  + Technical risks (missing tech, not accounting for future hardware)
  + Business risks (Building the wrong product for emerging customer base)
* Determine objectives, alternatives, and constraints
* Evaluate alternatives, identify, resolve risks
* Develop, verify next-level product
* Plan next phase

1. Define new system requirements (interviews)
2. Create preliminary system design
3. Create preliminary prototype (approximation of currently-known requirements)
4. Create second prototype from first; evaluate first, define requirements, plan and design, construct and test
5. Abort project if necessary (risk analysis)
6. Third prototype if necessary
7. Repeat steps 1-6 until prototype displays all customer requirements
8. Implement (construct)
9. Test

Barry Boehm, “A Spiral Model of Softwae Development and Enhancement”, *Computer*, May 1988, 61-72. A pdf download is available from the UW library IEEE Xplore database.

Agile: incremental and iterative development

<http://www.agilealliance.org>

<http://agilemanifest.org>

Craig Larman and Victor R. Basili, “Iterative and Incremental Development: A Brief History”, IEEE Computer June 2003, pp. 48-56. A pdf download is available from the UW Library IEEE Xplore database.

* Rapid and adaptive response to customer change
  + Refactoring and tools to enable it
* Effective communication among all stakeholders
* Drawing the customer onto the team
* Organizing a team so that it is in control of the work product
* Satisfy customer through early and continuous software delivery
* Accommodating change increases customer’s competitive advantage
* Frequent releases (2-3 weeks)
* Business and dev folk work together daily
* Trust your motivated people
* Face-to-face communication
* Progress is measured by working software
* Sustainable development – should be continuable forever
* Technical excellence and good designs enhance agility
* Simplicity is essential
* Best architectures, requirements and design emerge from self-organizing teams
* Teams self-calibrate on regular intervals for efficiency

Driven by customer scenarios (stories)

It is difficult to predict customer requirements and priority levels (and they change)

Plans are short-lived

It is difficult to predict required time for analysis, design, and testing

Iterative development with heavy construction emphasis

Construction proves the design

XP: Kent Beck, Ward Cunningham

Scrum: Jeff Sutherland; Ken Schwaber and Mike Beedle

Crystal: Alistair Cockburn

Dynamic Systems Development Method (DSDM): <http://www.dsdm.org>

Feature-Driven Development (FDD): Peter Coad

Adaptive Software Development (ASD): Jim Highsmith

Agile Modeling (AM); at the core of Agile Unified Process, or OpenUP: Scott Ambler

Customer feedback required after every sprint (2-4 weeks)

Extreme Programming

Scrum

Agile Modeling

Hybrid

Monday, April 16, 2007

Next 5 classes, chapters 6-9 (Requirements analysis and design)

Assignment 2 – Requirements Modeling. Pressman, chapters 7 & 8

Due Wednesday, May 2

Worth 25% of grade

2 approaches to requirements modeling

Used to specify requirements and define the system, early in the lifecycle model.

Imagine you are a CSS 497 intern. The project has a CMMI level 0 in requirement engineering. Your job is to write a whitepaper, educating stakeholders about their options for improving their requirements engineering (not elicitation, but refining requirements) using mature technology. Use a textbook as the source for this.

1. Structured Analysis
   1. Flow-oriented models
      1. Data flow diagrams
   2. Control spec (CSPEC) state diagrams
   3. Process Spec (PSPEC) “mini-spec”
2. Object-oriented analysis
   1. Scenario-based models
      1. Use-cases
      2. UML Activity Diagrams
      3. UML Swimlane diagrams
      4. User stories (agile)
   2. Class-based static model
      1. UML Class diagrams
      2. Class responsibility collaborator CRC models
   3. Behavioral models
      1. UML State diagrams
      2. UML Sequence diagrams

Scrum and Extreme Programming (XP) – most widely used agile methods

Scrum: management-oriented

Ken Schwaber and Mike Beedle:

Mike Cohn – Scrum Overview (good place for readings and diagrams):

<http://www.mountaingoatsoftware.com/scrum/>

Daily stand-up meeting w/ small team (5-7) people

What did you do since yesterday?

What are you going to do for tomorrow?

What roadblocks are in your way?

Roadblock-resolver is the “scrum master”, project manager. Does not assign work (that’s purely voluntary).

Iterative process, but small work artifact output compared to UP

Measure of progress is working code

2-4 week iterations (“sprint”), customer feedback incorporated after every iteration. Output is a “potentially shippable product increment”

Features for next sprint are kept in a “sprint backlog”; contains “user stories” (use-case for lazy folk)

Complete featureset is “product backlog” (complete requirements for lazy folk)

Customer demos within a “timebox” (short demo)

<http://www.xp123.com/xplor/xp0507/Scrum-dev.doc>

<http://alistair.cockburn.us/index.php/Main_Page/>

Alistair Cockburn, *Agile Software Development*, Addison-Wesley, 2002

P. 31-38 The Cooperative Game Principle

“Software development is a (resource-limited) cooperative game of invention and commlunication. The primary goal of the game is to developer useful, working software. The secondary goal, the residue of the game, is to set up for the next game. The next game may be to alter or replace the system or to create a neighboring system.

“Intermediate work products help…as reminders for our reflection. They provide shared experiences to refer to or serve as support structures for new idea. These mock-ups are not second-clas items, used only due to some accidental absence of technology. Rather, they are a fundamental technique used to help people construct…”

<http://www.ambysoft.com/scottAmbler.html>

<http://www.agilemodeling.com/>

<http://www.agilemodeling.com/essays/agileDocumentation.htm>.

Critical Points:

Fundamental issue is communication, not documentation.

Document stable things, not speculative things.

Prefer executable work products such as customer tests and developer tests over static work products such as POD.

You should understand the TCO for a document, and someone must explicitly choose to make that investment.

Agile Methods and the CMMI

Mark C. Paulk, “Agile Methodoliges and Process Discipline”, Crosstalk: The Journal of Defense Software Engineering, October 2002.

<http://www.stsc.hill.af.mil/crosstalk/2002/10/paulk.html>

Kent Beck and Barry Boehm, “Agility Through Discipline: A Debate” IEEE Computer, June 2003, pp. 44-46. A .pdf download is available from the UW Library IEEE Xplore database.

XP: Engineering practices-oriented

Pair-programming

Test-driven development

Refactoring – for fixing mistakes introduced by your lesser half

User stories

Highly-iterative (time-boxed)

Velocity – feature rate (estimating tool)

Design uses CRC cards

Spike solutions – design prototypes

KISS

Unit testing (tests drive the design; TDD)

William Wake – XP on a page

<http://xp123.com/xplor/xp0202/xp-one-page.PDF>

Man iterations in a release

Many releases in a product

Design is emergent – pay as you go

Hybrid models:

Agile Unified Process Models

Agile UP (Scott Ambler)

Open UP (Eclipse Project)

Agile methods and the CMMI

April 18, 2007

System Engineering:

Systems: a hierarchy of macro-elements

* Software
* Hardware
* People
* Database
* Documentation
* Procedures

Why?

Software interacts with other systems; understanding interrelationships is important.

Software systems don’t “succeed” unless you understand your customer (the people in the system)

Consider all system elements before focusing on software.

Good engineering begins with a clear understanding of the world view and progressively narrows to technical detail.

Consider alternative solutions and trade-offs

Business process engineering:

* Uses an integrated set of procedures, methods and tools to identify how information systems can best meet the strategic goals of the enterprise.
  + Each of these must work together
* Focuses first on the enterprise and then on the business area
* Creates enterprise models, data models and process models
* Creates a framework for better information management distribution, and control

Information strategy planning (world view/company philosophy), Business area analysis (domain view), Business system design (element view), Construction and integration detailed view.

Systems analysts understand the entire system. Software engineers don’t go there, but they need to undersand the “results of business considerations that set the stage for the requirements definitions”.

Validation: meets customer needs

Verification: meets specification

Requirements engineering:

Inception (come up with a shared vision):

1. Waterfall
   1. Planning
      1. High-level analysis
      2. Business areas
   2. Requirements
2. UP
   1. Inception and Elaboration
      1. Initial use-cases
      2. Define architectural scope
3. Scrum/XP
   1. Vision statement

Elicitation (ask the customer what their requirements are):

1. Waterfall
   1. Requirements
      1. Get specifics from customer
      2. Write a complete spec
2. UP
   1. Inception
   2. Elaboration
      1. Use-cases
      2. Prototype iterations
3. Scrum/XP
   1. User-stories
   2. CRC cards
   3. Spikes
   4. Daily meeting with on-site customer

Validation (iterative; elicitation happens at the same time as validation):

1. Waterfall
   1. Customer feedback at the end of every phase
   2. Based on documents
2. UP
   1. Customer feedback at the end of every iteration of every phase
   2. Construction
3. Scrum/XP
   1. Customer feedback every day

April 23rd, 2007

7.1 Overview

Requirements engineering helps software engineers better understand the problesmt hey are trying to solve.

* It is essential that the development team understand the requirements of a problem before the team tries to solve it.
* Building an elegant computer solution that ignores the customer’s needs helps no one.
* It is important to understand the customers wants and needs before beginning development of a computer-based system.
* The requirements engineering process begins with inception, moves on to elicitation, elaboration, negotiation, problem specification, and ends with review or validation of the specification.
* The intent of requirements engineering is for all stakeholders to develop a shared understanding of the customer’s problem.
* Must be adapted to the needs of a specific process, project, product, or people doing the work.
* Several different work products might be used to communicate this understanding (user scenarios, function and feature lists, analysis models, or specifications; change request process; other processes, etc.)
* In some cases, requirements engineering may be abbreviated, but it is never abandoned.

Requirements Engineering Tasks

* Inception: use context-free questions to establish
  + A basic understanding of the problem
  + The people who want a solution
  + The nature of the solution
  + The effectiveness of the collaboration between customers and developers
  + (Output: Vision statements, System metaphors, Success criteria)
* Elicitation: Find out from customers:
  + What the product objectives are
  + What is to be done
  + How the product fits into business needs
  + How the product is used on the day-to-day basis
  + Fine-tune the inception direction
  + (Output: Onsite visits, surveys, historical data, usability studies)
* Elaboration
  + Develop a refined technical analysis model of software functions, features, and constraints
* Negotiation - agree on a deliverable system that is realistic for developers and customers. This includes:
  + Categorizing and organizing requirements into subsets
  + Identifying relations among requirements
  + Reviewing requirements for correctness
  + Prioritizing requirements based on customer needs (priority points, p. 151. A variation is ‘dot-voting’, see <http://www.innovationtools.com/Articles/ArticleDetails.asp?a=141>
* Specification - Produce written work products describing the function, performance, and development constraints. This could be one or more of the following:

Validation

Formal technical reviews to examine the specification work products to ensure that all work products conform to decisions about the process, project, and products. The review mechanism needs to look for:

* Errors in content or interpretation
* Areas where clarification may be required

Requirements management - set of activities that help project team to identify, control, and track requirements and changes as project proceeds.

* Many of these activities are identical to those that make up the software configuration management process.
* Requirements are tagged with a unique identifier and classified by type (functional, data, behavioral, interface, or output)
* Database systems are invaluable in helping software teams track requirement changes
* Traceability tables (e.g., features, source, dependency, subsystem, interface) are developed and updated any time a requirement is modified). See Figure 7.1: Generic Traceability Table – each row is a requirement and each column is a specific aspect of the system or its environment. The table serves as a checklist.
* Work artifact – traceability table
  + Features traceability table – shows how requirements relate to customer observable features
  + Source traceability table – identifies source of each requirement
  + Dependency traceability table – indicate relations among requirements
  + Subsystem traceability table – requirements categorized by subsystem.

Initiating the Requirements Engineering process (Inception)

Identify stakeholders

Recognize the existence of multiple stakeholder viewpoints

Use context-free questions to focus on customer, stakeholders, overall goals, and benefits of the system

Who is behind the request for work?

Who will use the solution?

What will be the economic benefit of a successful solution?

Is there another source for the solution needed?

The next set of questions enable developers to better understand the problem

Work towards stakeholder collaboration.

Picture the software in the environment in which it will be used.

User-Centered Analysis and Design – Grounds the process on users through the planning, design and development of a product.

<http://www.upassoc.org.usability_resources/about_usability/what_is_ucd.html> (Usability Processional’s Organization)

<http://www.uiaccess.com/accessucd/background.html> (excerpt from Just Ask: Integrating Accessibility Throughout Design by Shawn Lawton Henry.)

Eliciting Requirements

Collaborate Requirements Gathering

Meetings attended by both developers and customers

Rules for preparation for participation are established

Flexibile agenda is used

Facilitator controls the meeting

Definition mechanism (stickers, flip sheets, electronic bulletin board) used to gauge group consensus

JAD: Joint Application Development – A task force of users, managers and IT professionals work together to gather information, discuss business needs and define new system requirements (see Wikipedia)

QFD: Quality function deployment

* Uses customer interviews, surveys, on-site observation and examination of historical data (bug reports)
* Identifies three types of requirements (normal, expected, exciting)
* In customer meetings, function deployment I sused to determine value of each function that is required for the system
* Information deployment identifies both data objects and events that the system must consume or produce (these are linked to functions)
* Task deployment examines the system behavior in the context of its environment
* Value analysis is conducted to determine relative priority of each requirement generated by the deployment activities

Elicitation work products:

Statement of need and feasibility

Bounded statement of scope for system or product

List of customer, users and other stakeholders involved in requirements elicitation

Description of system’s technical environment

List of requirements

Set of usage scenarios

7.5

Requirements models are not design models. They represent what the system must do, not how it works. They use structured, formal, natural language. Functional/services/capability view.

Dataflow diagram – the kind of services that a system has to provide

Assignment: explain the nature of the differences between the different modeling viewpoints.

April 25, 2007

Modeling techniques

Scenario-based modeling:

What tests in the user domain are going to be performed?

Page 190 – 191

April 30, 2007

Next time: Chapter 9

Then: QA, 13-15 (in-class exam), Monday, May 21st, worth 20% of grade

3 pages of notes, typed, are permitted.

Then: Chapter 21-22 (project management)

Comprehensive take-home final, 25% of grade; variation on the “what is software engineering?” paper, summarizing the content of the entire course. 20 pages. Due Monday, June 4th.

May 2, 2007

Chapter 9

Transform between analysis model and design model

Data/Class – turn class-based analysis into classes and data structures

Architectural design – derived from class-based and flow-oriented elements

Interface design – derived from uses-cases and other scenario-based models, flow-oriented elements and behavioral elements

Component-level design – derived from architectural design; structural elements transformed into procedural descriptions, uses flow, class-based, and behavioral elements.

From top to bottom:

Component-level design

Interface design

Architectural design

Data/Class design (public interfaces)

FURPS Quality Factors (H-P)

Functionality

Usability

Reliability

Performance

Supportability – includes extensibility, adaptability, and serviceability; all of which represent maintainability.

May 7, 2007

Testing – exercising a program to find bugs, hopefully before the customer does

* Tests contain inputs (steps) and outputs (verifications)
* Performed by someone other than the programmer
* Planned, sometimes using test specs
* Happens in a spiral, from unit to integration to validation to system, corresponding to implementation, design, and requirements in the SDLC

Types:

* Unit
  + Exercise control paths through public interfaces
  + OO: Exercise classes
* Integration/Scenario
  + Functional: issues with integrated components
  + OO: issues with integrated classes (collaboration and communication errors)
  + “sandwich integration”, top-down or bottom-up
  + Regression
  + Smoke
* Validation/Acceptance
  + Alpha
  + Beta
* System
  + Recovery (?)
  + Security
  + Stress
  + Performance
  + UI and usability
* Debugging is tactical (ad-hoc or on-demand), testing is strategic (planned)
* Verification (does the feature work?) vs. Validation (are we building the right feature?)
  + Testing is one element of SQA (software quality assurance)
  + Quality is built-in; cannot be grafted on later
* Organizing
  + Separate test org for objectivity
  + Devs must do some testing
  + Test team is not responsible for all testing (“over the wall”)
  + Test team should involved from the beginning
  + Dev and test must work together
* Test strategy
* Testing small to testing large
* Unit tests!!! They do exist!!
* I know, amazing.

Unit tests

* Module interfaces tested for input and flow
* Boundary, error-handling, functionality
* OO: methods in classes
* Stubs and drivers may be required to test just the component.
* Drawback: A positive verification may be less indicative of overall product quality, since actual system components are substituted with stubs.
* High cohesion and low coupling make this easier

Integration (scenario) testing

* Testing modules in the system as a whole
* Big bang or incremental
* Top-down integration: Stubbing systems, width-first or depth-first
* Bottom-up integration: cluster low-level components and drive it from above
* Thread tests
* User scenarios
* Cluster (?) testing; a set of classes that demonstrates one collaboration. Hmm.

Smoke (incremental/iteration) testing

* Showstopper bugs
* Validate that further testing can continue