

CSE 5324: Software Engineering I

(Analysis, Design, Creation)

We do not act rightly because we have virtue or excellence, but we rather have those because we have acted rightly.

We are what we repeatedly do. Excellence, then, is not an act but a habit.

Aristotle

Review

Preview

Brooks Book Chapter

New stuff

What is important

What is next...

Last class:

Overview of the class:

What the class is about

Class "flow"

**Details: Eating and drinking, late to class
talking, questions, etc.**

Grading

Internet availability (of lectures)

Software Engineering is...

Grading:

Course grades:

"A" is "superior"

"B" is "average, acceptable"

"C" is "passing, below average"

Minimal Threshold:

90 - 100 A

80 - 89 B

70 - 79 C

These ranges **may** lower, but **will not** be higher.

Exams:

There will be 3, in class, exams.

Each Exam will be 80 minutes long (full class).

Each exam will be equally weighted.

If these exams grades are satisfactory, the average will count as as the final exam grade.

Fall 1999

Topic Schedule:

Introduction:

What is Software Engineering? (Chap 1)

Process and Life Cycle (Chap 2, and Pages 493-502)

Requirements:

Concepts, methods and specifying requirements (Chap 4)

Object Oriented Analysis

Followed by: Design (Chap 5), Object Oriented Design, and more...

Preview:

Why learn Software Engineering (SE)?

Benefits and perils

What is SE?, terms, concepts, questions

History and future trends

Success and failure in software projects

Fred Brooks

Software Engineering

People Months and Reality

The Book (that EVERYONE has, and some read)

Introduction:

What we will talk about: (Questions and concepts)

What is Software Engineering?

Why Study Software Engineering?

How software has evolved

Software Characteristics

Software Components

Software Applications and Systems

Software Myths

What do Software Engineers do?

Program

(Actually this is a small part of the job)

Analyze - what needs to be done?

Ask lots of questions, eliminate contradictions, fill in what's missing

Specify - write down the details of what needs to be done

Design - how to do it.

Module hierarchy, pseudo-code, etc.

Program - write code

Integrate - units, modules, systems

Test - units, subsystems, systems, alpha/beta

Maintain - by far the largest effort

Maintenance

Successful products spend most of their lifetime:
time, money, effort, etc. here

Corrective - fix bugs (defects, errors, faults, failures)

Adaptive - new network, operating system, etc.

Enhance - new features, functions

(also "perfective" - other than original functional requirements)

Preventive - re-engineering

How does one develop software?

The development "process"

Ad-hoc, chaos

Prototype

"Waterfall"

...

"Maturity" of development process:
CMU's SEI model: CMM

ISO 9000, etc.

Why is software important?

What's wrong with how it is done now?

What are common problems?

(Terminology - is important)

Quality
What is it?

Metrics and measurement

A few quick terms and concepts:

"The Process"

(What is a Process?)

Methods

Tools

Paradigm

Heuristics - (rules of thumb)

Definition (what)

Development (how)

Maintenance

correct, adapt, enhance, prevent

Client, Developer, User

Software gone bad:

The bill for \$0.00

Strategic Air Command WWMCCS
(misread a simulated attack)

Therac-25 medical linear accelerator
(2 dead due to radiation overdose)

Patriot missile failed to deploy: timer overflow (reset)
Problem detected but not fixed.
(28 dead, 98 wounded)

Ariane-5 rocket launching 4 satellites blew up;
Loss of \$500 million; really much more - in 1996
CNES had over half the world launch contracts;
replaced by Russia, China, US, etc.

Year 2000 "Problem"

What is Software Engineering?

Apply Engineering Principles to software.
(My definition, not original)

Pressman: (Text Book): Technology encompassing a process,
a set of methods, an array of tools.

Sommerville: The specification, development,
management, and evolution of software
systems.

And there are many more definitions.

What is the "core" idea here?

Techniques to apply to software projects.

Q:

Why can't we (or I) keep developing software as I have been. It has worked so far...

A:

You have been building bird houses. One room. Simple. Very small. You don't need to plan, if the walls are not straight - so what?

The birds don't complain (Customer satisfaction)

Maybe you have built dog houses - bigger, some planning and design is needed, and the dog might complain.

Have you built a multi-room family house?

Can you decide to interchange two rooms- is it as easy as moving program code with an editor? When can you re-arrange rooms: during design or when building?

How do you build a big multi floor office building?
(In Los Angeles - an earthquake zone?)

Consider:

Computing is about 50 years old.

Moore's Law says capabilities double every year and a half:

Processors are twice as fast, memory and disk are twice as big, etc.

New technologies arise: CD-Roms, DVD, Higher Speed LANS, telecommunication - datacom, Internet, WWW, video, and audio, even movies and TV are merging (Movie companies start software firms, software firms start TV and movie companies)

For the first 10 years of computers, systems typically executed a few thousand "instructions" per second.

For the middle 10 years the execution was 10's of thousands to a very few million per second.

Now a low end PC is over a 100 million per second, with embedded systems - microwave ovens or PDA organizers - in the millions per second.

Moore - and most technologists - believe this progress will not slow down - and many believe it will accelerate.

BUT

Has software kept up?

Hardware is faster/better/cheaper

Customer expectation has increased:

Software is easier to use: GUI-WIMP

(Windows, Icon, Mouse Pointer)

Customers can do things in application domains that used to require programmers: Spreadsheets, Data Bases, Publishing.

But we (software engineers) are expected to create

Easy to Use

Reliable

Efficient

Correct (Bug free)

Software

And managers want:

Software

On time

Easy to fix

Easy to port to other OS, Networks, Data bases

New, enhanced versions quickly

Usable by the customer

And some systems require

A financial software shouldn't loose money.

It must comply with accounting practices.

A pacemaker software must be very reliable

(peoples lives are at risk) and even "self correcting"

Are we up to it?

What do customers think of the process?
(Of developing software)

The programmers don't listen.
They develop what they want not what I asked for.
And then it's buggy anyway.
Why do I have to keep punching in that account
number on every screen, can't this program remember
one stupid number!?

What does the boss (the SE manager) think?

These programmers couldn't keep to a schedule if their
life depended on it. Always late.
Doesn't anyone test anything - why are these obvious bugs
here?
I never get a straight answer out of these developers.
Why does the customer change their mind every 5 minutes?

What do we (practitioners) think?

I can write the program - the difficult part. Let some English major write the documentation. Anyone can test.

My "deliverable" is the program.

The customer must wait for the product until I'm done.

Quality is me writing code really careful. Then testing.

Some scary facts:

There are BILLIONS of lines of working, currently being used code.

Much (most) is undocumented and is in:
FORTRAN, COBOL, or (some kind of) assembly

Does it "wear out"?

Is there really a benefit to SE?

(December 1998, Scientific American,
Capers Jones on Sizing Up Software)

One can measure capability in terms of Function Points
(Number of inputs, outputs, files, queries, etc)

Word processors: 5000 FP, Business App: 40 K FP,
OS: 100 K FP, Defense System 250 K FP.

Some statistics based on 100,000 projects:

$FP^{1.25}$ number of errors

$FP/150$ number of programmers

$FP^{0.4}$ number of months (Calendar time) to complete

Number of lines of code: (in C 150 lines/FP)

Cost per FP: (\$2000/FP)

What I need to know:

- Terms, definitions (but don't memorize)

- Some names: (like Wasserman points out...)

- Value of SE

- Good (quality) and bad software

- Good, concrete examples and explanations (when your boss asks)

What's next:

- Processes and Life cycles

- (What came first? The chicken or chickenpox?)

Last class:

Internet availability (of lectures)

Why study/use SE?

Some Terms

Process and Life Cycle

None (probably "Code and Fix")

Waterfall

Prototype

Incremental

Spiral

Some terms:

Real Time

Formal

Scenarios

Today:

Finish up Introduction

Finish Life Cycles

Team Projects

Requirements

Brooks Book Chapter

Introduction and some review:

Analysis - break into understandable pieces

Synthesis - put small pieces (blocks) together to create large system

Method (or technique, step) - formal way (step) to produce a result
(a step to cook...)

Tool - automated system (instrument) to do something better

Procedure - recipe to combine methods and tools

Paradigm - a cooking style or philosophy

More terms:

Bugs, faults, errors, failures

Testing, customer finds bugs, peer reviews

Safety-critical - for example reliability of 10^{-9} (hours) means
1 failure in 114,000 years - how do you test?

(recent "New Yorker" magazine article about medical
safety-critical and review "M & M" (Mortality and morbidity)
in 35 million general anesthesia what is "acceptable" failure?)
(1 in a million)

Quality: ISO 9001, CMM, etc.

Yet some more:

ROI (return on investment) - training?, quality?, process improvement?

Customers; developers, users

COTS - commercial off the shelf

Subcontracts, maintenance, configuration management

Wasserman notes: time to market, OOT, networks, UI, PC's,
Waterfall problems forced SE changes

Abstraction - generalization

Transformation - translate, often real to math

Design and notation

Measurement

Software development includes:

Analysis and definition of requirements

System and program design

Coding

Testing: Unit, integration, system

Maintenance

A process model puts these into some order

None (Code + fix)

Waterfall

Prototype

Verify Waterfall - "V" model

Operational specification - execute requirements to demo behavior

Transformational model - formal definition transformed to implementation

Incremental model

Spiral

Process modeling can be textual (words) or graphic (or both)

Models be static - inputs transformed to outputs or

dynamic - see intermediate products and how it works over time

CMM

SEI (CMU) developed CMM

	Level	Focus	Key Practices (KPA)
1	Initial		
2	Repeatable	Mgmt responsibility	Training, staffing
3	Defined	Compet.	Career develop, planning,
4	Managed	Measured	Organizational, teams
5	Optimizing	Continuous	Improvement

Requirements

Introduction and some review:

Team projects:

This is what SE is about

Projects:

Build a SE "tool" (An OO CRC web based tool)

A virtual map of UTA where current classrooms are determined and displayed (VRML or similar)

Simulation: Computer architecture, OS, Network (web)

External customer (the library)

(May allow others)

Models of software development (Life cycles)

Waterfall (linear sequential)

Prototype

Incremental

Spiral

Cleanroom

What is:

Real Time

Component, re-use

"Formal"

Scenarios

...

What I need to know:

- Teams and projects

- Terms, definitions (but don't memorize)

- Process models

What's next:

- Groups

- More processes and life cycles

- Requirements

