CSE 5324: Software Engineering I
(Analysis, Design, Creation)

Instructor: David Levine
Office: 308 Nedderman Hall
August 24, 1999

or

August 24, Y2K - 1
Who is afraid of this class?

Who is afraid of me?
Flow:

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
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?)
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?)
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

<table>
<thead>
<tr>
<th>Level</th>
<th>Focus</th>
<th>Key Practices (KPA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initial</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Repeatable</td>
<td>Mgmt responsibility</td>
</tr>
<tr>
<td>3</td>
<td>Defined</td>
<td>Compet.</td>
</tr>
<tr>
<td>4</td>
<td>Managed</td>
<td>Measured</td>
</tr>
<tr>
<td>5</td>
<td>Optimizing</td>
<td>Continuous</td>
</tr>
</tbody>
</table>

Training, staffing
Career develop, planning,
Organizational, teams
Improvement
Requirements
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
Last class:

Rational Terms
Motivation
Life of a Software Project
Preview:

Software development
  Life Cycles
  Paradigms
  What does a manager do?

Requirements

Success and failure in software projects
Developing Software Projects

Software development life cycles

How does one (team) develop software?
Process and Life Cycle
  None (probably "Code and Fix")
  Waterfall
  Prototype
  Incremental
  Spiral
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
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 (Capability Maturity Model)

SEI (CMU) developed CMM

<table>
<thead>
<tr>
<th>Level</th>
<th>Focus</th>
<th>Key Practices (KPA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initial</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Repeatable</td>
<td>Mgmt responsibility</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Training, staffing, SQA, planning</td>
</tr>
<tr>
<td>3</td>
<td>Defined</td>
<td>Organization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Career develop, training, process def.</td>
</tr>
<tr>
<td>4</td>
<td>Managed</td>
<td>Measured</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Organizational, teams, Quantitative</td>
</tr>
<tr>
<td>5</td>
<td>Optimizing</td>
<td>Continuous</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Improvement, defect prevention</td>
</tr>
</tbody>
</table>
CMM (Capability Maturity Model)

(Model for a 200,000 line product)

<table>
<thead>
<tr>
<th>Level</th>
<th>Duration (Cal months)</th>
<th>Effort (Person m)</th>
<th>Faults (Develop)</th>
<th>Cost of Development</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>29.8</td>
<td>593.5</td>
<td>1,348</td>
<td>$5,440,000</td>
</tr>
<tr>
<td>2</td>
<td>18.5</td>
<td>143.0</td>
<td>328</td>
<td>$1,311,000</td>
</tr>
<tr>
<td>3</td>
<td>15.2</td>
<td>79.5</td>
<td>182</td>
<td>$728,000</td>
</tr>
<tr>
<td>4</td>
<td>12.5</td>
<td>42.8</td>
<td>97</td>
<td>$392,000</td>
</tr>
<tr>
<td>5</td>
<td>9.0</td>
<td>16.0</td>
<td>37</td>
<td>$146,000</td>
</tr>
</tbody>
</table>
Requirements
Models of software development
(Life cycles)

Waterfall (linear sequential)

Prototype

Incremental

Spiral

Cleanroom
CSE 5324: Software Engineering I
(Analysis, Design, Creation)
Today: 9-9-99

Many software systems failed today.

Why?

Sentinel values used for end of data set (end of file) like dates: 4-1-99, 1-1-99, 9-9-99

(In band signaling: 9-9-99; out of band: -35-800--1 for dates)

What type of error was this?
Requirements, design, code, test, other?
Review

Preview

Brooks Book Chapter

New stuff

What is important

What is next...
Last class(es):

Software Engineering is...
Introduction, Terms, concepts, etc.
Process: what is, life cycles
Requirements
Requirements

What are good requirements?

How do you do requirements and specification?
Requirements:

Architecture templates (ACD, AFD)

<table>
<thead>
<tr>
<th>User Interface</th>
<th>Input</th>
<th>Process</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintenance, Self-test</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Requirement - feature of the system

Elicitation - capture the users needs, categorize: must be met, desirable, (etc.)

Definition vs. Specification

Functional vs. non-functional requirements
Structured Analysis:

Based on ideas of structured programming  
(when programming was most important)

Source to Sink: 
  Input to output 
  Flows 
  Transform

Data Flow Diagrams

Data Dictionaries

(Other ways might be OO, for example)
Class Song
(or school song)
CSE 5324: Software Engineering I
(Analysis, Design, Creation)
Review

Preview

Brooks Book Chapter

New stuff

What is important

What is next...
Last class(es):

- Software Engineering is...
- Introduction, Terms, concepts, etc.
- Process: what is, life cycles
- Requirements
- Structured Analysis
Requirements

What are good requirements?

How do you do requirements and specification?
Team projects

Group "Job" application

Introduction and some review:

Team projects:

This is what SE is about

Projects:

Build a SE "tool" (An OO 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 (volunteer service groups, CSE, the library)

(May allow others)
Objected Oriented Software Engineering:

Is this (OO) really so different?

Review:

Requirements (can be):
- Text
- Structured (SA)
- Formal Methods
- Object Oriented
- Many more

What is good, what’s bad (about each)?
Objects:

(Should) model the “real” world

Assumes an evolutionary process model
Tend to evolve; allow re-use

O-O is analysis, design, and programming

An object may represent real world entities

A Class is an abstraction of objects

You do this now: Pascal and C:
Types, records, structures.
May build upon other types, records and structs.
Objected Oriented Software Engineering:

OOA – Analysis

What is analysis: specify and model a problem.

OOA:

  What are the objects?
  How do they interact?
  How do objects act (behave) in the system?
  How to specify or model a problem with objects to Create a design?
Objects are closer to the way we really think about problems. We categorize, classify, make relationships, actions are on objects.

Brooks: manipulate the essence, rather than the mapping into an implementation accident.

The benefits are “up-front”. Conceptual issues rather than implementation have benefit for later phases. Don’t need to use OO programming to get benefit of OOA (or OOD).

All OO includes:

“Identity” (Objects)
Classification (Objects with same attributes and Operations are grouped into a class)
Polymorphism (same operation behaves different on different classes)
Inheritance (sharing of attributes and operations Based on hierarchical relationship)
Object Modeling Technique (OMT)
(Rumbaugh, etc)

1. Analysis: (what)
2. System Design (overall architecture – subsystems)
3. Object Design (Implementation details of objects)
4. Implement (minor and mechanical)

Three “models” to describe a system:

1. Object Model (static structure)
2. Dynamic Model (Control – how system changes over time, state diagrams, transitions, events)
3. Functional Model (DFD’s)

This is different from function oriented methodology:
FO specifies and decomposes system functions.
OO identifies application domain objects, fits Procedures around them.
Some themes:

Abstraction (essential aspects, not accidental)

Encapsulation (information hiding – separate external accessible aspects from internal implementation.)

Combine data and behavior (data hierarchy and Procedure hierarchy are combined)

Sharing (inheritance)

Emphasis on Objects not procedures (what Object is, not how used)
OOA Methods:

Booch:
  Micro and macro development
  Micro is re-applied to each macro step.

Coad and Yourdon:
  Simple. Like SA and other Yourdon Methodology. “What to look for”
  Then top-down. General to specific,
  Whole to part.

Rumbaugh
  OMT (above)

Unified Method (UML)
  Booch and Rumbaugh

Wirfs-Brock
  Analysys and design combined.
  Tools to extract classes from specification.
  Identify super classes. More bottom-up.
UML:

"Unified" Modeling Language

Model to simplify reality

  Visualize a system
  Specify structure and behavior
  Template to help construct system
  Helps document system

The choice of a model has profound influence on how system is analyzed and solution built

May specify at increasing levels of "precision" (detail)

Best models are connected to reality

No single model is sufficient
<table>
<thead>
<tr>
<th>Classes</th>
<th>Class Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>Origin</td>
</tr>
<tr>
<td></td>
<td>Size</td>
</tr>
<tr>
<td></td>
<td>open( )</td>
</tr>
<tr>
<td></td>
<td>close( )</td>
</tr>
<tr>
<td></td>
<td>move( )</td>
</tr>
<tr>
<td></td>
<td>show( )</td>
</tr>
</tbody>
</table>

**Attributes**

**Operations (methods)**
UML:

Generalization (Single inheritance)

Shape

Position

move()
display()

Box

Corner

Circle

Radius
Structural Relationships
aggregation

University has 1 or more students
Each student may attend many courses;
    each course may have many students
University has one or more departments
    (University is a "whole" student and departments are its parts)
UML:

Structural "things":
- Classes (and class interfaces, collaborators - what classes are needed,
  use case - sequence of actions yielding observable result;
  use case from "actors")

Also has:
- Behavioral
  - Interactions - messages between objects
  - State machines - sequence of states in response to events
    (And some others - like "groupings" and runtimes, etc.)
Reuse and domain analysis

Common requirements for specific application

Domain

Use class libraries:
  Faster, less cost, fewer defects
Examples:
  Microsoft, Graphics, Database

Berard:

What is the domain?

Categorize Items

Collect representative sample of applications

Analyze each application

Develop analysis model for objects
Use Case:

Scenario of how system will be used.

Actors – people (or machines, or other software) that Represent roles (not a user – who is typically different actors at different times.)

Jacobson:
What preformed by actor?
What will actor acquire, produce, modify?
What does actor want from system?

Firesmith
Taxonomy of class types:
Device classes
Interaction classes
Tangible? (real or abstract)
Inclusive?
Sequential (or concurrent control - access)
Persistent (transient, permanent)
Wirfs
   Evenly distributed intelligence
   Generalized responsibility
   Encapsulate
   Localize information in a class

Object Relationships

   Verbs – location, placement (part of, next to)
   Ownership – made up of
   Manages, controls, etc.
What is good:
  Reality
  Success on many projects
  Reuse
  Tools

What is bad?
  Difficult to get used to
  Can user understand
  Can you?
Exams

1  September  21
2  October  21
3  November  18
Final  December  7

Exam 1:
Chap 1, 2, 4, and CMM and SA
(Brooks for extra credit, Chap 1 - 8)

Closed book, closed notes

However you may bring one single page (front and back) of notes (I suggest handwritten but you may use a printer) maximum 8.5" x 11" (or A4) – you may use both sides

Note:
If you use a reminder sheet you must turn it in with test.
Exam 1 will primarily test student knowledge and understanding of the material presented so far.

Source of Material:

Lectures

Reading assignments
(regardless of whether material was discussed)

(Bonus may be from Brook's book)

100 points total (not including a possible bonus)

In class, you will have entire class time

Closed book, closed notes.

Bring dark ink pen or dark lead pencil with eraser (2 or more)

Bring your own calculator (if you need one)
ANSWER QUESTIONS!!! Read the entire exam before starting. Types of questions to expect: definitions, problems, essay, etc (NOT: true/false, multiple choice, etc.)

TEST understanding of terminology used in course
TEST understanding of concepts
TEST ability to apply methods to particular problems

Explain means EXPLAIN not DEFINE - do not just give definition

Specifically answer the question, not "generically", getting the buzzwords in the answer won't count: "many", "fast", etc. rarely (if ever) answer any question (or they answer ALL questions).

Relax, don't Panic! You can afford to miss a few points.
MATERIAL COVERED

INTRODUCTION
The roll of software.
The state of software (today), competitiveness
Characteristics of software.
Software domains, applications.
The software “crisis”.
Software myths.
General introduction to software engineering
What is Software Engineering? Gave several definitions.
Differences between Programming and Software Engineering
Why is software engineering important?
Software Engineering Goals
PROCESS
  Process, methods, tools, (KPA)
  What is software engineering?
  Maintenance
  Process
  CMM SEI
  Process models
    Code and Fix,
    Waterfall,
    prototyping, RAD,
    incremental,
    spiral,
    component assembly,
    formal methods

Software Life Cycle
  Concepts of Software life cycle versus Project Life Cycle
PRINCIPLES OF ANALYSIS AND ANALYSIS MODELING
Requirements
models, prototype,
specification and review

REQUIREMENTS
Structured, Formal Methods, Cleanroom

Requirements Analysis - General
Focus and Objectives
Determine WHAT is needed, not HOW it will work
specify software functionality
performance criteria
software interfaces with other systems
design constraints
Benefits of requirements engineering
Requirements Engineering Process
  Requirements Analysis
    Definition: Requirements Elicitation
    Domain Understanding
    Requirements Collection
    Classification
    Conflict Resolution
  Prioritization
  Requirements Validation
  Requirements Definition
  Requirements Specification

What are good requirements?
  Specification Principles
  Characteristics of good requirements
Structured Analysis Method

Dataflow Diagrams (DFDs)
Notation
  DEF : CONTEXT DIAGRAM
  DEF : DATA FLOW DIAGRAM (DFD)
Hierarchy, concept of leveling and balancing
Guidelines for creating data flow diagrams

Data Dictionary (DD)
Information for primitive and group DD entries
Definition notation for a group

Process Specifications (Pspects)
Definition: primitive process
Definition: PSPEC (Process Specification)
Pspec information:
  PSPEC ID, Process Name, Input and Output flows, Specification, Comments
Styles of Specification
  Narrative English
  Structured English
Today:

Review? (Exam 1)

Requirements

Team Projects
Brooks Book Chapter
Requirements

What are good requirements?

How do you do requirements and specification?
Structured Analysis (requirements and specification)

Object Oriented
What I need to know:
   Teams and projects
Requirements:
   What are good?
   How to do?
   SA
   OOA

What's next:
   Exam
   then
   More Requirements
   Other ways to do...
Short answers:

Several software process models have been discussed, 3 are: "linear sequential", "prototyping" and "incremental" models. Give one similarity to all models. Give one difference between each pair of models.

Similar:

Differences:
You go to work for a company that is CMM level 2 organization. 
(a) List 3 KPA's that you should see. 
The company is discussing trying to evolve to a level 4 
organization, but has estimated that it will cost 2 million dollars 
to do so, plus an additional 1 million dollars per year. 
Currently there are 200 software engineers costing 
(on average) $100 thousand each.

(b) Present an argument (for or against) that it is economically 
worthwhile, and when (how soon) is the pay back.

(c) Why would a level 4 organization need to spend  million 
extra per year over a level 2?
A bicycle "computer" is a device that allows a bicycle rider to calculate a few interesting parameters during a trip. The bicycle computer (called "BiC") has a simple 6 digit display, and additionally there are display indicators that display the "mode" of the BiC. These mode indicators show what the BiC is currently displaying (distance traveled or average speed).

To reset the time and distance counts there are buttons to: clear (reset counts), set distance measuring mode, set average speed mode, and turn BiC off.

A small computer provides control functions and has as a time base a small clock that it can read (hours:minutes:seconds).

The revolution of the wheel may interrupt or be read by the BiC, signifying a 1/3 of a meter traveled. (The wheel turns 3 times per meter)
(If you think that additional hardware is necessary who must explain why and then describe in detail.)

a.) Show an ACD for Bic.
b.) What is the ACD used for, what is its purpose?
A cell telephone needs to store telephone numbers for "rapid dial" (ie the user hits a rapid dial key, then N to dial the N'th phone number)
Only legal telephone numbers may be stored: local 7 digit phone numbers (that do not begin with 1 or 0), special numbers (911, 411, and 0),
US long distance ("1" followed by a three digit area code - not starting with 0 - followed by a 7 digit phone number, described above), or international numbers (011 followed by a country code of up to 3 digits, followed by a city code of up to 4 digits, followed by a phone number of up to 9 digits; where the country, city, and phone numbers may not begin with 0)

(a) Please show the data dictionary for legal phone numbers.
(b) Why is this data dictionary needed, where would it be used?
(c) In your software development group an argument starts, during analysis, about whether to store the telephone numbers in long binary format (fixed length) or as 4 bit digits (variable length, to save space). You are the team leader, settle the argument. Explain.
6. [20 pts]
   Your organization is given the job of developing a portable, electronic, downloadable book. The book is a small display (40 lines of about 50 characters each) in a plastic case with buttons at the bottom for: scrolling (up, down, left, right), and a menu selector for allowing the user to "command" the book: download a new book, go to a page number, etc. At the top is a small infrared "port" that allows communication to a special book provider who sends books to the device.

a.) If (or where) there are ambiguities or omissions, please describe them and describe how you will deal with them.

b.) Show a context diagram (DFD) and as many levels of decomposition as needed, to a maximum level of 2, for the book (Follow the standard for DFD). You don't need to write PSpec or DD's.

c.) Please write a process specification (PSpec) that is invoked to handle the scroll down. Use structured English for the specification, follow standards for process specifications.
Bonus:

What does Brooks say about "the second system" effect? Explain. Why does it cause problems?
Bonus:

You are shipwrecked on a deserted tropical island. You may choose the one person with you. That person is:

a.) A medical doctor with survival training

b.) An expert boat builder

c.) The instructor of this class

d.) Someone who looks good in a swimming suit