CSE 5324: Software Engineering I
(Analysis, Design, Creation)
Process and process improvement
The way you get better is practice, practice, practice. (And learn from your mistakes)

Defining your process:
what steps are done when? what is the result?

Learn from (your) history

What can you measure?

Transactions/sec.
Average module (or method) size (or complexity)
Time/module (or method)
Reuse rate
Time per phase
Fault density
Requirements "creep"
(changes in new/changed function points)
Schedule slips
and much more

Quantitative measurements

What can change? What could be improved?
Variability, quality, etc.

Predictive: what can one predict and plan
knowing history
Many tools are statistical: based on broad sample of industry, many types of projects, many types of programming languages, styles, methods, etc.

COCOMO
FP counting
...

Your history is much more accurate

Set targets

Set baselines

Goals
CSE 5324: Software Engineering I

(Analysis, Design, Creation)
Review: SQA and Testing
   Cleanroom
   Formal Methods
   Real Time
   Maintenance
   Process and process improvement

Exam 3: November 18
   Primarily Text: Chap 8, 9.2, 10, 11.1 - 11.6, 4.5 (Z)
   Real Time, Z, Test, Maintenance,
   Process measurement and improvement

Review
Why Grades are not important

(But what is....)
Real-Time Systems:

What are they?

When (where) are they used?

How are they different? (From non-RT)
Many (most) real-time systems are embedded

Where are they seen:
  where real-world events are time critical:
  navigation (air, space)
  medical
  telecomm, network
  games

Often classified into:
  Soft (RT)
  Hard (RT)

Usually both (and non-RT requirements also)

Degraded operation versus Incorrect results
Often viewed as stimulus - response system

Stimuli:
- Periodic (predictable time intervals)
- Aperiodic (irregular)

Response:
- Actuator

Classification:
- Monitoring systems
- Data Acquisition
- Command and Control
- etc.
Usually hardware and software done in parallel (and tradeoffs are made)

Establish system requirements

<table>
<thead>
<tr>
<th>Partition Requirements</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Reqs</td>
<td>Hardware Reqs</td>
</tr>
<tr>
<td>Software Design</td>
<td>Hardware Design</td>
</tr>
</tbody>
</table>

Steps:

Identify Stimuli (to process) and associated response

For each stimuli and response identify timing constraints

Group stimuli and response into concurrent processes

For each, design algorithms (usually early in design process, may make tradeoffs)

Design scheduling system to ensure processes started on time

Integrate with a real-time executive (OS)
Implementation overhead is critical:

Design tools and methodologies

Programming language:

Is C appropriate?
  (NO)

C++?
  (Even worse)

Ada
  Ada 83 was not (design errors) but Ada 95 is

What is needed:
  HLL synchronization
  OS interface constructs: mutual exclusion, IPC, start process by time, scheduler changes, etc.
  Interrupt handling
OS:

(Not Unix!)

Real time clock
Interrupt handler
Scheduler
Resource manager
Dispatcher

May include:
  Configuration manager
      Static or dynamic
Fault manager
      Hardware and software
Some Issues:

Is OO a good idea for RT?

Timing: how to make estimates:
   Counting instructions (pipelines, etc.)
   Compiler helping

Analysis

Simulate

Use good tools:
CSE 5324: Software Engineering I
(Analysis, Design, Creation)
"Last week NASA announced that the "human" error stemming from space engineers using two sets of measurements -- one utilizing miles and the other kilometers -- caused the loss of the Mars Climate Orbiter spacecraft last week, NASA said Thursday."

The teams, located at the National Aeronautics and Space Administration's Jet Propulsion Laboratory in Pasadena and at Lockheed Martin Astronautics in Colorado, complicated matters further by failing to realize the error, the agency said in a statement.

The $125 million orbiter, intended to serve as the first interplanetary weather satellite, is believed to have broken up when it hit the Martian atmosphere last week after an approach that was too near the surface.
Today's question is:

What kind of error was this?

How could/(should) it have been prevented?
Object Oriented Methodology: Terms, ideas, techniques
OO Concepts, OO Analysis, OO Design
Design Concepts, Design Methods (Structured)
Real Time Analysis and Design
What is designed:

Data
Architecture
Interface
Procedural
Object Oriented Design:

Find objects, factor into good classes, define class hierarchies and interfaces, make it reusable, modifiable.

OO Design architecture stresses object relations rather than flow of control

What is different: (Fichman, Kemerer) (From SA)

Module Hierarchy representation
Data Definition Spec
Procedural Spec
End-to-end processing sequences
Class and Hierarchy definition
Operations assigned to classes
...

Bertrand Meyer suggests:

- Decomposable – design method makes easier subproblems
- Composable – reusable
- Understandable – by module
- Continuity – coupling (local)
- Protection – coupling (isolation)

To achieve:
- Few, small, explicit interfaces. Information hiding.

OOD:
- Booch
- Coad – Yourdon
- Rumbaugh

Jackson – design a part of analysis!

Coad suggest (for any method): Define:
- Problem Domain
- User I/F
- Task Management
- Data Management
Design Patterns:
(This is another of the SE Buzz words!)

Recurring and reusing of classes and objects
(Gamma)

Name of pattern
Problem to which applied
Characteristics
Consequences of applying

Should one use inheritance or composition?
(When both possible)
Use what is general, or make it specific?
Compare SA, SD versus OO:

Hospital system: Patient record system.

Spreadsheet
Is OO Important?

OO Prog languages: JAVA, et al

OO Interfaces:
  Windowing systems

CORBA
  Distributed objects
Cleanroom: (Formal Methods)

Requirements:

Remember: What to do.
   (what, NOT how.)

What is wrong with what is done now?

   Vague
   Ambiguity
   Incomplete
   Contradictions
   Mixed Abstraction and methods

Clients must agree to requirements,
Developers must be able to design from them.

Word descriptions (suffer from above)

Structured Analysis

O-O Analysis
Cleanroom Software Engineering

Build correct software as it is developed.

Use Mathematical Notation

Verify Correctness
   (Statistical Quality Certification)

Formal Design and verification

Code Generation, Inspection, Verification

Statistical Test Planning, Usage Testing

Certification, Integration
Techniques used in Cleanroom:

Incremental life-cycle

Formal Techniques for Specification and Design

Non-Execution Based Module-Testing
  (Code reading, walkthroughs, inspections, proofs)

Module not compiled until AFTER passed inspection
Does Cleanroom work?

US Naval Underwater systems Center:
18 Faults detected when design “functionally verified”
(Informal proofs used mostly, only when participants unsure of correctness were full mathematical proofs used)
19 more faults during walkthrough (about 2KLOC)
NO compilation errors
NO Failures during execution

Does it scale?
Testing Fault Rate
(Faults detected per KLOC)
This is different between Cleanroom and other Development techniques.

Ericsson Telecom OS32
350,000 Lines of code
18 months, team of 70 people
Testing Fault Rate 1.0 Faults per KLOC
(Navy was 0.0)

Linger reports average Testing Fault Rate 2.3 Faults per KLOC based on 17 products of 1 million LOC. (This is very high quality)
What's not so good about Cleanroom:

Problems?:

Henderson:
Too theoretical, mathematical, radical for real-world
No Unit testing for developers
Maturity of software developers, industry
Strategy:

Increment Planning

Requirements Gathering

“Box structure” Specification

Formal Design

Correctness Verification

Code Generation, Inspect, Verify

Statistical Test Planning

Statistical Usage Testing

Certification,

Integration
Strategy:
Increment Planning
Requirements Gathering
“Box structure” Specification
Formal Design
Correctness Verification
Code Generation, Inspect, Verify

Statistical Test Planning
Statistical Usage Testing
Certification,
Integration
How is this different (from other methods)?

Statistical Q. C.

Design Specs mathematically proved correct

Statistical Usage Testing
Functional Specification:

Box Structure:
   Referential transparency (information content stands alone)

Black Box

State Box

Clear Box
Design Refinement: Structured Programming

Stepwise expansion

Verify at each level of refinement
(Linger)

Verification is a finite process

Every line of design and code verified

Very low defect levels

It scales up

Better code than unit testing

Statistical Use Testing
   Probability Distribution

Certification
Requirements:

What to do.
   (what, NOT how.)

Formal Methods:

What is wrong with what is done now?

Vague
Ambiguity
Incomplete
Contradictions
Mixed Abstraction and methods

Clients must agree to requirements,
Developers must be able to design from them.

Word descriptions (suffer from above)

Structured Analysis

O-O Analysis
Use Mathematical Notation

(Must extend some)

Many choices available:
CSP, VDM, Z, etc.

Z ("Zed")
Should I Use Formal Methods?

(Bowen and Hinchley)

Use Appropriate Notation
Formalize, not overformalize
Costs?
Have a FM guru available
Don’t abandon what you have
Document
Still need SQA
No dogma
Test, test, …
Reuse
What’s good about Formal Methods:

- It is Real Engineering
- Precise
- Validation, Verification
- Testable
- Automated tools
- Good SE s can do it
- Don’t need to spel gud or English Grammar
- Reusable
- (Many more)
What’s not so good about Formal Methods:

Problems?:

- Hard to learn
- I can’t read this
- I can’t write this
- Client can’t either
- Is this the future?
- Lack of scoping and structuring
<table>
<thead>
<tr>
<th>example</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema signature</td>
<td></td>
</tr>
<tr>
<td>Schema predicate</td>
<td></td>
</tr>
<tr>
<td>CoffeePot</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>Coffee: ( \mathbf{N} )</td>
<td></td>
</tr>
<tr>
<td>PotCapacity: ( \mathbf{N} )</td>
<td></td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Coffee ( \leq ) PotCapacity</td>
</tr>
<tr>
<td>CoffeeLight</td>
</tr>
<tr>
<td>-------------</td>
</tr>
</tbody>
</table>

Light: \{ off, on \}  
CoffeeLevel: **N**  
PotCapacity: **N**

Light = on ⇔ Coffee > PotCapacity
$\Delta_{\text{CoffeePot}}$

FreshCoffee?: $\text{N}$

FreshCoffee? + CoffeeLevel < PotCapacity
CoffeeLevel' = CoffeeLevel + FreshCoffee?
CoffeeSystem

CoffeeMade ∨ CoffeeBrewing ∨ CoffeeOverflow
Some more notation:

Domain to range:

Coins = \{ \text{cent} \rightarrow 1, \text{nickel} \rightarrow 5 \}

\forall \ l, j: \text{dom } s \bullet (l < j) \Rightarrow s(l) < \tau s(j)

\exists \text{Name } \text{ means that state variables are unchanged}
<table>
<thead>
<tr>
<th>MakeChange</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>CostOfItem: N</td>
</tr>
<tr>
<td>ChangeFrom: N</td>
</tr>
<tr>
<td>ChangeFrom = 1.00</td>
</tr>
</tbody>
</table>
Software Testing

Object Oriented Testing
REMEMBER:

What is the goal of testing?

(Finding errors.)

Assess quality

Uncover errors

Ensure standards are met
Is OO Test different?
(From other methods)

- OO models are different
- Unit testing is based on classes
- Integration is different
- OO assumes re-use

The basic goals of testing remain the same

We want to find errors as early as possible
(in process steps: requirements, design, code)
to avoid propagation
Errors made in analysis may:

- Lead to errors in classes in design
- Make more work in procedures for a bad attribute (update, inherit, etc.)
- Cause extra (or incorrect) subclasses or system behavior

How are analysis and design "models" tested?

Different methodologies (UML, Rumbaugh, Booch, etc) use different notation

We still want to check that "some" correct syntax was used

"Experts" in the problem domain can review class definitions and hierarchy for missing or ambiguous

Do class relationships reflect "the real world"?
Class - Responsibility - Collaboration (CRC) model

classes and relationships may be inconsistent

CRC lists
  class name
  responsibilities (operations, what it does)
  collaborators (other classes needed, sent messages)
  and optionally:
    class type (like: transaction)
    class characteristics (systems related: atomic, secure)
CRC Steps:

Cross check CRC and O-R model

Ensure that a delegated responsibility is actually done by each (that) collaborator

Ensure each collaborator receives requests from correct (reasonable) source

Are other classes needed, is work grouped correctly?

Can (should) widely requested services be grouped (combine responsibilities that always go together)

NOTE: this may be a bad idea - reuse, etc.
OO Unit testing

With encapsulation, smallest testable unit is a class (or object)

How can we isolate and test?

Class testing - unlike unit test based on algorithms and data flow - state behavior of class and encapsulated operations

Integration testing

What is top-down or bottom-up (with classes)?

Integration often described as:

Thread-based - based on response to an input or event.

Use-based - test independent classes that use fewest server classes, then move on to dependant classes
Cluster testing

collaborating classes are clustered
(by O-R and CRC models)
and exercised

Validation testing

At system level, class connections not visible

Black Box works here

Test cases may be done from OOA:
Object Behavior, event flow
Test Case Design

Berard

Test case explicitly connected to class

State test purpose

Testing:
Object states
operations, messages
exceptions tested

May be difficult to know the internal state of an object
It might be necessary to build in operations to report the state

Inheritance requires retesting
Conventional testing (and differences):

Fault based test - high probability of finding an error where are most likely errors (to be found)?

Derived and Inherited classes - redefined class tests

Scenario-based testing
What does the user do? (Fault based - what does software do)

Use Case

Surface structure test - structure visible to user (commands, etc.)

Deep Structure - examine design and code
Class Testing

Random tests

Partition test - similar to equivalence partition

based on:

- attribute
- class states
- category - functions of operations

Interclass test

Multiple class testing
  random testing, partition testing

Behavior model test
  based on state transition diagram
    (dynamic behavior)
  breadth first usually
Reasons not to test:

You didn't know it was policy

Blame someone else's code (design, etc.)

Good choices:
   dead co-workers
   children that can't talk

Bad choices:
   bosses children
   boss

Not on the schedule

We keep finding problems

Too complex

The risks (product liability) not great

Waste of (my) time
REMEMBER:

What is the goal of testing?

(Finding errors.)

Assess quality

Uncover errors

Ensure standards are met
Is OO Test different?
(From other methods)

- OO models are different
- Unit testing is based on classes
- Integration is different
- OO assumes re-use

The basic goals of testing remain the same

We want to find errors as early as possible
(in process steps: requirements, design, code)
to avoid propagation
Errors made in analysis may:

- Lead to errors in classes in design
- Make more work in procedures for a bad attribute (update, inherit, etc.)
- Cause extra (or incorrect) subclasses or system behavior

How are analysis and design "models" tested

Different methodologies (UML, Rumbaugh, Booch, etc) use different notation

We still want to check that "some" correct syntax was used

"Experts" in the problem domain can review class definitions and hierarchy for missing or ambiguous

Do class relationships reflect "the real world"?
Class - Responsibility - Collaboration (CRC) model

classes and relationships may be inconsistent

CRC lists
  class name
  responsibilities (operations, what it does)
  collaborators (other classes needed, sent messages)
  and optionally:
    class type (like: transaction)
    class characteristics (systems related: atomic, secure)
CRC Steps:

Cross check CRC and O-R model

Ensure that a delegated responsibility is actually done by each (that) collaborator

Ensure each collaborator receives requests from correct (reasonable) source

Are other classes needed, is work grouped correctly?

Can (should) widely requested services be grouped (combine responsibilities that always go together)

NOTE: this may be a bad idea - reuse, etc.
OO Unit testing

With encapsulation, smallest testable unit is a class (or object)

How can we isolate and test?

Class testing - unlike unit test based on algorithms and data flow - state behavior of class and encapsulated operations

Integration testing

What is top-down or bottom-up (with classes)?

Integration often described as:

Thread-based - based on response to an input or event.

Use-based - test independent classes that use fewest server classes, then move on to dependant classes
Cluster testing

  collaborating classes are clustered
  (by O-R and CRC models)
  and exercised

Validation testing

  At system level, class connections not
  visible

  Black Box works here

  Test cases may be done from OOA:
    Object Behavior, event flow
Test Case Design

Berard

Test case explicitly connected to class

State test purpose

Testing:
  Object states
  operations, messages
  exceptions tested

May be difficult to know the internal state of an object
It might be necessary to build in operations to report the state

Inheritance requires retesting
Conventional testing (and differences):

- Fault based test - high probability of finding an error where are most likely errors (to be found)?

- Derived and Inherited classes - redefined class tests

Scenario-based testing
  What does the user do? (Fault based - what does software do)

  Use Case

- Surface structure test - structure visible to user (commands, etc.)

- Deep Structure - examine design and code
Class Testing

Random tests

Partition test - similar to equivalence partition

based on:

attribute
class states
category - functions of operations

Interclass test

Multiple class testing
random testing, partition testing

Behavior model test
based on state transition diagram
(dynamic behavior)
breadth first usually
Reasons not to test:

You didn't know it was policy

Blame someone else's code (design, etc.)

Good choices:
  dead co-workers
  children that can't talk

Bad choices:
  bosses children
  boss

Not on the schedule

We keep finding problems

Too complex

The risks (product liability) not great

Waste of (my) time
Re-engineering

(This is a good "YAB" - Yet Another Buzzword)

As the rules of business change the software supporting those rules must change.

A Business Process ("BP") - like other processes is a set of related tasks to achieve a defined outcome

BPR (Reengineering) is implementing change for "breakthroughs"

BP has a "customer" - who receives the outcome (product)
Hammer suggests:

Organize around outcomes (not tasks) (easier to "track" work status)

The process customer performs (controls) the process.

Put information processing into place where work is done.

Get the data once, where it is produced.

View distributed (geographically) resources as if they were centralized.

Make decisions where work is done (not higher up).
BPR is a cycle, including:

Define Business Goals

Identify process, prioritize

Evaluate processes, measure

Specify and design process

Prototype (try it out)

Refine and use
Software Re-engineering:

Maintenance is:
  Corrective
  Adaptive
  Perfective (enhance)
  Preventive (re-engineer)

Process Model:
  Inventory Analysis - details of applications
  Document restructuring
  Reverse Engineer - create an abstract representation
  Restructure Code - most common
  Restructure Data - may actually be more valuable
  Forward Engineering - apply SE principles to recreate
  existing application (with new features)
Some details:

Reverse Engineer to understand:
  Data Structures
  Database Model
  User IF

Restructure to:
  Improve quality (how to measure?)
  Improve programmer productivity
  Maintainable, testable

Forward Engineer to:
  Use new technology:
    Distributed, new GUI, OO methods, CASE Tools

Is it worth it?
  Cost benefit is Maintenance costs - Re-engineering costs