COMP314:

Design

Readings:

Chapter 4+5 Code Complete
Where have we got to?

Problem definition

Requirements

Use cases

Architecture Design

Modeling Analysis

Code

Unit tests

Integration

System tests

Demonstration
Where is the feedback?
Choice of Programming Language

☐ Sapir-Whorf hypothesis: “can only think thoughts expressible in words” [e.g. division using roman numerals]

☐ Programming language choice:
  - productivity (C-1, C++/Java 2.5, Python 6)
  - code quality
  - language expert 3x more productive
Major decisions

- Coding
  - Amount of upfront design
  - Coding conventions
- Pair programming
- Quality:
  - Units tests, integration tests before checkin
  - Review/inspect each other’s code?
- Tools:
  - Revision control tool (svn, cvs, ...)
  - Language / Compiler / OS version
  - Other: IDE (Eclipse), other libraries
Art and Science of Design

Design is a “wicked” problem - can only be defined by solving it
Art and Science of Design

- Tradeoffs and priorities
- Restrictions
- Non-deterministic
- Heuristic Process
- Emergent
Art and Science of Design

Art of creating a design

The science of what a good design should be
Complexity - the great enemy

- Many other domains can hide essential complexity behind a wall of accidental complexity

- In computer science (often) the accidental complexity is removed exposing naked essential complexity
Complexity - the great enemy

“Everything should be as simple as possible and no simpler” - Einstein (maybe).
Complexity - how things go wrong

- complex solution to a simple problem
- simple, but wrong solution for a complex problem
- a complex, but wrong solution for a complex problem
Complexity - how to discover it isn’t there and to manage it

- Divide into parts
- Hide information
- Iterate
- Testability
- Identify potential changes
- Use appropriate language to describe problem
- Avoid accidental complexity
Decomposition into parts

- Loose coupling
- High fan-in
- Low-to-medium fan-out
- Stratification
- Testability
- What does each part have to know about?
- Hide things
Decomposition into parts

<table>
<thead>
<tr>
<th>Data</th>
<th>Model</th>
<th>User interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operations on data</td>
<td>Major Operations</td>
<td>Mouse clicks</td>
</tr>
<tr>
<td>- update loans db</td>
<td>- borrow a book</td>
<td>Input values</td>
</tr>
<tr>
<td>- write to log</td>
<td>- return a book</td>
<td>Colors</td>
</tr>
<tr>
<td></td>
<td>- Transactions</td>
<td>Appearance</td>
</tr>
<tr>
<td></td>
<td>- Multiple threads</td>
<td>Layout</td>
</tr>
</tbody>
</table>

Operations
- update loans db
- write to log

Major Operations
- borrow a book
- return a book
- Transactions
- Multiple threads

User interface
- Mouse clicks
- Input values
- Colors
- Appearance
- Layout
Decomposition into parts - testing

- Data
  - Unit tests
- Model
  - Unit tests
  - Stress tests
  - Multiple threads
- User interface
  - Walkthrough
  - Manual scripts
Example from ReelTwo - handling millions of documents

Rest of system

Document Interface

- Text file
- XML
- PDF
- Word
- HTML

Test documents

String of text
Unique ids

Document Cache

Available memory
Recent usage
Disk storage

Details of format layout
Character encoding
Where have we got to?

- Problem definition
- Requirements
- Use cases
- Architecture Design
- Modeling Analysis
- Code
- Unit tests
- Integration
- System tests
- Demonstration
Why hide information?

- Size of human brain
- Reason (informally) about correctness
- Ease of testing
- Ease of maintenance
- Proven in practice
Barriers to hiding

- Global data
  - everyone can see it
  - everyone can modify it
  - hard to know when it will be changed
  - hard to test

- Constants
  - 6 ?? MAX_LOANS
Barriers to hiding

- Excessive worry about performance
- when you can measure it and it is too slow

then do a more complicated and faster thing
Ways to document design

- Java Interfaces
- UML diagrams
- English
- Wiki
- Comments
Checklist: design practices - p122 Code Complete

- Iteration - best of several possibilities
- Split system in different ways
- Top down and bottom-up
- Prototype
- Design review
- Implementation obvious?
- Captured design?
Checklist: design goals - p122
Code Complete

☐ Reflect architecture and requirements
☐ Stratified
☐ Good decomposition
   subsystems, packages, classes, routines
☐ Minimal interaction
☐ Is everything necessary?
☐ Standard techniques
☐ Minimize complexity
Design

- Much room for art and creativity have been given ideas for how to do it

- Strict guidelines for what constitutes a good design - keep it simple.