Detailed Design & Programming Considerations

Outline

Walkthroughs

Detailed Design

Programming Considerations

Internal Documentation

Project Configuration Management
Project Schedule – Spring

• Design Walkthroughs
  Jan. 20  Physics
  Jan. 22  Headstart
  Jan. 23 Optometry  Woodburn 007
  Jan. 27 RS - Player Board
  Jan. 29 BDU
  Jan. 30 RS - Shift Substitution  Woodburn 007

• Detailed Design
  February 5 (R)

• Implementation: complete “first-draft”
  March 3 (T)

• Testing complete
  March 315 (T)

• Installation complete, documentation draft done
  April 14 (T)

• Documentation complete
  April 21 (T)

• Post Partem presentations
  April 21-30
Design Walkthroughs

Explain both Architecture and Details

Tables

- review ER only if concerns
- variations from “standard” mapping
- problematic representations
  - do not describe all fields

Forms:

- general characteristics/form standards
- navigation events
- database/system interaction events
  - discuss issues, not details of all fields

Modules:

- overall structure
  - interactions/how activated
  - changes from Preliminary Design
- interfaces - parameters, globals, ...
- essential internals
  - exceptions

Installation
Walkthrough Reviewers

Reviewers:

another team specifically responsible for critiquing

Deficiencies:

Identify aspects that are
★ unclear
★ ambiguous
★ lacking essential details

Improvements:

Suggest ways to improve
★ modularity for consistency, robustness, ...
★ representation of data
★ code for efficiency, consistency, ...
★ usability via layout, flow, ...

Strengths:

Identify aspects of the design that enhance
★ consistency
★ maintainability
★ usability
Detailed Design Process

What it is?

- Iteration (not *repetition*) of the design process.
- Incorporates details of implementation environment.
- Final chance to “get it right.”

How is it constructed?

1. Start with the Preliminary Design
2. Quickly review
   a. implementation feasibility on platform
   b. places where details needed
3. Correct design architecture
4. Provide *necessary* details
5. Walkthrough
6. Final revisions
Detailed Design Document

Possible contents:

Database issues:
data model (ER) †
data dictionary †
logical database design (relational schema) †
physical database design

Program (product) issues:
module decomposition and definition †
algorithms & module internals
programming standards
specific programming techniques
user interface details (prototype †)
test points

Process issues:
task allocation & scheduling
software test plan †
programming standards
test scripts

† carried forward & refined

Notation for expressing module internals

Several possible ways:
structure charts
Nassi-Shneiderman diagrams
structured English
pseudo code
UML

The important factors are expressiveness, simplicity, and consistency.
Level of Detail

• As required for Coding

• Detailed design should translate directly into coding.

   “Direct translation” ⇒ module could be coded by looking only at the design for that module

• design vs. coding ≈ semantics vs. syntax

• design is what you want to say, coding is how you say it
Robust Coding

It is the difference between professionals and “hackers”.

Robustness

• during operation
• over life-cycle

“Anticipate the unanticipatable.”

• inputs
• applications
• changes

Robust coding begins with good design.

• modules
• exception conditions
Impact of Tools

“If all you have is a hammer, every problem looks like a nail.”

Other variations of the metaphor:
- have you ever tried to cut wood with a hammer?
- a Swiss army knife is neat, but who needs a fish scaler?
- working on an old American car, English wrenches are required but metric screwdrivers are OK

Implications:
1. sometimes design constrained by tools
2. sometimes design must overcome tools
3. project might include tool building

Application generators present a special problem:
- “workarounds” must be re-done with every form change
Programming Style Goals

- Accurately represent design
- Aid team coding
- Aid testing/verification
- Facilitate maintenance correction, enhancement, and adaptation
- Induce efficiencies
Programming Style Factors

- Modularity
  good design is most important!
- Storage structures & algorithms
  factor from detailed designs
- Structure
- Comments
- Form
  naming conventions
  statement layout
- Uniformity

Comments and form often aggregated as *internal documentation*
Code “Prologue” Comments

- **Identification**
- **Purpose of Module/Procedure**
- **Interface description**
  - arguments in & out
  - pre- and post-conditions
  - sample call
  - external dependencies
  - external actions/side effects
  - called procedures
- **Important internal objects**
  - types & variables (syntax)
  - structures & representations (semantics)
- **Exceptions**
  - cause & effect
- **Development history**
  - events to record
    - code
    - review
    - test
    - change
  - information to record
    - why
    - when
    - what
    - by whom
Other Internal Documentation

Comments and form intertwine

• Page Layout
  ◊ White space
  ◊ Pagination
  ◊ Delimiters

• Well-chosen identifier names

• Layout to reflect scoping & blocking
  lots of rules for ifs, blocks, etc.: where does the “then” go? consistency most important
  applies to expressions, lists, etc. too
  \[ A + (B \ast C) \]
  preferable to
  \[ A + (B \ast C) \]

• Limited comments
  indicate subsections
  clarify obscure points
  link to purpose

✧ Pretty printers
Layout for Semantics, not Syntax

Which is clearer?

• Layout by syntax

```plaintext
if ( x < y )
  # case 1 code
else
  if ( y < 0.5 )
    # case 2 code
  else
    # case 3 code
```

• Layout by cases

```plaintext
if ( x < y )
  # case 1 code
elsif ( y < 0.5 )
  # case 2 code
else
  # case 3 code
```
Moral for Internal Documentation

- Keep internal documentation consistent with executable code

- If you can’t explain it, you shouldn’t code it
Configuration Management Motivation

Goal:

Know when code is stable.

$C^3M$

Change Control and Configuration Management

- connects to testing
  regression testing
- connect to QA
  record keeping
- most importantly, provides programmability

Model:

- a code module may be in one of two states: in-process or accepted
- a module is accepted when it has passed all tests as required in Test Plan
- after a module is accepted, it is
  - handed-over to Program Librarian
  - read-only except following $C^3M$
Configuration Management Implementation

Process for C³M

• changes to a module must record
  ◊ Why
  ◊ Who
  ◊ When
  ◊ What

Program librarian *essential*!

☆ especially when development is distributed
☆ central repository of “accepted” code
OR carefully choreographed software distribution