Fundamentals of Design

PSU CS 300 Lecture 6-1

Bart Massey
Assoc Prof Computer Science
Portland State University
<bart@cs.pdx.edu>
You can't teach design

- Everyone agrees on teaching design
- In this lecture, I will
  - explain SE improvement path
  - show skills
Design bridges reqs and code

- **Hamlet**: process of
  - refining reqs into code (↓)
  - defining libraries (↑)
- **The hard part is architecture**
- “Meet in middle” vs “Middle out”
Traceability

- (Virtual) matrix vs reqs, code

<table>
<thead>
<tr>
<th></th>
<th>D1.1</th>
<th>D1.2</th>
<th>D2.1</th>
<th>D2.2</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1.2</td>
<td>•</td>
<td></td>
<td>•</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1.3.1</td>
<td></td>
<td>•</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1.3.2</td>
<td>•</td>
<td></td>
<td></td>
<td>•</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>R2.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2.2.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2.2.2</td>
<td></td>
<td></td>
<td></td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>R2.2.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>•</td>
</tr>
</tbody>
</table>
Top-down: natural boundaries

- Need to find partition of
  - data
  - algorithms
- Module has
  - one simple interface
  - relative internal simplicity
Bottom-up: natural support

- Need to find modules with
  - one simple interface
  - relative internal simplicity
  - provides abstraction
  - will be re-used
- E.g. **libs**, syscall
- **Danger:** over-general
Composition / decomposition

- Iterative process
- First step is the hardest
- Usually max 3-4 levels
- Look for
  - noun, verb phrases
  - model correspondences
  - limits on generality
Objects

- Group verbs w/ nouns
- Info hiding
- vs “classes”
- Enable reuse
- Top-down
ADTs

- Algebraic / Abstract Data Types
- Parnas: *Info hiding*
- Allow higher-level reasoning
- Bottom-up
Relational database

- **DB is strong tool**
  - ACID: atomicity, consistency, isolation, durability
  - Types; regularity

- **DB is not magic bullet**
  - Simple structure
  - Clunky
Art

- *Form follows function:* see what the problem tells you
- *Late commitment:* never make an unforced choice
- *Minimality:* don't do extra—small is simple
- *Clarity:* obvious is good
Data (+ Algorithms)

- Where is data in system?
- What data structures can represent data?
- What functions touch the data?
- How does data move?
Role of knowledge

- Don't sell your textbooks!
  - data structures
  - algorithms
- Limit number, scope of unsolved problems
- Research vs development
Design prototypes

- Don't go to code too soon!
- Build prototypes to answer specific questions
  - design
  - code
  - learn
  - discard!
Experience

- The above takes 5+ years of continuous experience
- Constantly pick challenges “just a bit too hard”
- Get expert feedback
  - co-design
  - mentor
Detailed Design

- Last level is “special”
- Pseudocode
  - bridges design and code
  - bridges English and code
- **Refine** other levels
  - interfaces, error handling, docs
Test plan

- Should construct test cases during design
- Unit / integration test
- Helps to guide design
Is it a design yet?

• Can you (your team) build it “mechanically”?
• Are you confident that it is valid / verified?
  – inspection
  – formal methods
Case study: AutoDealy

- **SW for poker-dealing machine**
  - robot HW
  - **reqs**
    - U1.1: 2-5 hands
    - U1.2: 5-7 cards per hand
    - U2: Hands dealt must be truly random
    - U3: Hand in 30 seconds
You can't teach design

• Everyone agrees on teaching design
• In this lecture, I will
  – explain SE improvement path
  – show skills
Design bridges
reqs and code

• **Hamlet: process of**
  - refining reqs into code (↓)
  - defining libraries (↑)

• The hard part is architecture

• “Meet in middle” vs “Middle out”
Traceability

- (Virtual) matrix
  - vs reqs, code

<table>
<thead>
<tr>
<th></th>
<th>D1.1</th>
<th>D1.2</th>
<th>D2.1</th>
<th>D2.2</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1.2</td>
<td></td>
<td>•</td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>R1.3.1</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1.3.2</td>
<td>•</td>
<td></td>
<td>•</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2.2.1</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>R2.2.2</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2.2.3</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Top-down: natural boundaries

• Need to find partition of
  – data
  – algorithms
• Module has
  – one simple interface
  – relative internal simplicity
Bottom-up: natural support

- Need to find modules with
  - one simple interface
  - relative internal simplicity
  - provides abstraction
  - will be re-used
- E.g. **libs**, **syscalls**
- **Danger:** over-general
Composition / decomposition

- Iterative process
- **First step is the hardest**
- Usually max 3-4 levels
- Look for
  - noun, verb phrases
  - model correspondences
  - limits on generality
Objects

- Group verbs w/ nouns
- Info hiding
- vs “classes”
- Enable reuse
- Top-down
ADTs

- Algebraic / Abstract Data Types
- Parnas: Info hiding
- Allow higher-level reasoning
- Bottom-up
Relational database

- **DB is strong tool**
  - ACID: atomicity, consistency, isolation, durability
  - Types; regularity
- **DB is not magic bullet**
  - Simple structure
  - Clunky
**Art**

- *Form follows function:* see what the problem tells you
- *Late commitment:* never make an unforced choice
- *Minimality:* don't do extra—small is simple
- *Clarity:* obvious is good
Data (+ Algorithms)

- Where is data in system?
- What data structures can represent data?
- What functions touch the data?
- How does data move?
Role of knowledge

- Don't sell your textbooks!
  - data structures
  - algorithms
- Limit number, scope of unsolved problems
- Research vs development
Design prototypes

- Don't go to code too soon!
- Build prototypes to answer specific questions
  - design
  - code
  - learn
  - discard!
Experience

- The above takes 5+ years of continuous experience
- Constantly pick challenges “just a bit too hard”
- Get expert feedback
  - co-design
  - mentor
Detailed Design

- Last level is “special”
- Pseudocode
  - bridges design and code
  - bridges English and code
- Refine other levels
  - interfaces, error handling, docs
Test plan

- Should construct test cases during design
- Unit / integration test
- Helps to guide design
Is it a design yet?

- Can you (your team) build it “mechanically”?
- Are you confident that it is valid / verified?
  - inspection
  - formal methods
Case study: AutoDealy

- **SW for poker-dealing machine**
  - robot HW
  - reqs
    - U1.1: 2-5 hands
    - U1.2: 5-7 cards per hand
    - U2: Hands dealt must be truly random
    - U3: Hand in 30 seconds