Introduction to Computers and Programming

Prof. I. K. Lundqvist

Outline

• Bhorbugs and Heisenbugs

• Designing Large Programs
  – Software design quality
  – Modularity
  – Design by Contract
Real Bugs and Software Bugs

- Bugs **adjust** to the level of experience of the programmer
- Bugs **invade** the test environment
- Bugs **replace** previously caught bugs

## Taxonomy of Bugs

- Reproducible bugs / Bohrbugs
- Unreproducible / Heisenbugs
- Tasking / Timing bugs
Reproducible Bugs/ Bhorbugs
Always cause a failure and can be reproduced

- Try **explaining** what should be happening
- **Verbalization** often clarifies muddled thoughts
- Have a **friend** do a quick sanity check
- **Don’t randomly** change things, your actions should have a purpose

Heisenbugs
A bug that **disappears** or **changes behavior** when you are trying to track it down

- Try to make the bug reproducible by switching platforms
- Insert checks for invariants and have the program stop everything when one is violated
- Verify each layer with small, simple tests
- Find the smallest system which demonstrates the bug
Tasking / Timing Bugs

- Synchronization properties are not specified
- Unconditional waits
- Deadlocks and races

Software Design Quality

- What is quality?
  - Construction quality
  - Aesthetic quality
  - Fit for purpose?
- How can we measure quality?
  
- Design quality: Fitness to purpose
- Quality is a measure of Software together with its application domain
  - Requirements analysis
  - Quality predictors
Quality Predictors

• **Simplicity**
  – Meets its objectives, without any extra decorations
  – Look for complexity
    • Control flow complexity
    • Information flow complexity
    • Name space complexity

Quality Predictors

• **Modularity** is a logical partitioning of the software design that allows complex software to be manageable for purposes of implementation and maintenance

  – **Coupling**
    • Property of a collection of modules

  – **Cohesion**
    • Property or characteristic of an individual module
Coupling

- Coupling indicates:
  - how closely two modules interact or how interdependent they are
  - the degree of coupling between two modules depends on their interface complexity

Classes of Coupling

- data: low / best
- stamp
- control
- common
- content: high / worst


**Coupling**

- **Data** coupling: Two modules are data coupled if they communicate via a parameter 

  

  (+++)

- **Stamp** coupling: Two modules are stamp coupled if they communicate through a composite data structure 

  (+)

- **Control** coupling: Data from one module is used to control the direction of the execution in the other module 

  (0)

**Coupling**

- **Common** Coupling: Two modules are said to be common coupled when both reference the same shared/global data 

  (-)

- **Content** Coupling: Two modules are said to be content coupled when they share code 

  (---)
Concept Question

Test_stack.adb and my_stack package are:
1. Not Coupled
2. Are Content Coupled
3. Stamp Coupled
4. I still don’t understand coupling

Classes of Cohesion

<table>
<thead>
<tr>
<th>Classes of Cohesion</th>
</tr>
</thead>
<tbody>
<tr>
<td>functional</td>
</tr>
<tr>
<td>sequential</td>
</tr>
<tr>
<td>communicational</td>
</tr>
<tr>
<td>procedural</td>
</tr>
<tr>
<td>temporal</td>
</tr>
<tr>
<td>logical</td>
</tr>
<tr>
<td>coincidental</td>
</tr>
</tbody>
</table>

Degree of cohesion

- Best / high
- Worst / low
Cohesion

• **Coincidental** cohesion exists when subprograms in the module relate to each other very loosely, if at all (---)

• **Logical** cohesion exists when all elements in the module perform similar operations (---)

Cohesion

• **Temporal** cohesion exists when a module contains tasks that must be executed within the same time span (+)

• **Procedural** cohesion exists when the subprograms in the module are part of the same algorithm (+)
Cohesion

• **Communication** cohesion exists when all subprograms in the module reference or update the same data structure (+)

• **Sequential** cohesion exists when elements of a module form different parts of a sequence, i.e., output from one element of the sequence is input to the next (+++)

Effects: initialize the data structures and initialize the screen display and initialize the history stack and initialize the layout defaults and display an introductory text

**Describe the functions in a single sentence**

**Effects**: if \( x = 0 \) then returns size(\( a[] \)) else if \( x = 1 \) then returns sum(\( a[] \)) else if \( x = 2 \) then returns mean(\( a[] \)) else if \( x = 3 \) then returns median(\( a[] \))

Cohesion

• **Functional** cohesion exists when all subprograms in the module cooperate to achieve a single function (++++)

Effects: initialize the data structures and initialize the screen display and initialize the history stack and initialize the layout defaults and display an introductory text
Concept Question

my_stack package has:

1. Logical cohesion

2. Functional cohesion

3. No Cohesion

4. I still don’t understand cohesion