

CSE1301
Computer Programming
Lecture 25:
Software Engineering

1

Topics

- Software Engineering
- Structure charts
- Structured design
 - Modularity
 - Coupling
 - Cohesion

2

Software Engineering

- Structured development of software systems
 - *Examples:* software for banking, stock exchange, space probe control, toll charge, operating system, games
- Tools and Techniques
 - Structured process rather than trial-and-error
 - *Important goal:* eliminate errors
- “Engineering”
 - Zero tolerance to error
 - Pre-fabricated components
 - Re-usable components

3

Software Engineering (cont)

- Tasks
 - project planning
 - project management
 - documentation
 - prototyping and simulation
 - interface design
 - programming
 - testing
- **Computer-Aided Software Engineering (CASE) Tools**

4

Recall: **Components of the Software Development Process**

- Define the problem clearly
- Analyze the problem
- Design an algorithm
 - top-down / bottom-up design
- Code (Implement) the algorithm
- Test the code
- Document the system

5

Development Approach: Water-Fall Model (Old)

```
graph LR; A[Analysis] --> B[Design]; B --> C[Implement]; C --> D[Test];
```

- Requires that each stage be completed before beginning next stage

6

Development Approach: Incremental Model

```

    graph LR
      Analysis[Analysis] --> Design[Design]
      Design --> Implement[Implement]
      Implement --> Test[Test]
      Test --> Analysis
      Test --> Design
      Design --> Analysis
      Implement --> Design
      Test --> Implement
  
```

- Increments from **simplified version** with limited functionality to **complete system**
- At each stage: **prototype**

7

Prototyping

- Construction of simplified version of parts of the proposed system that can be analyzed before further development
- Types of items that can be prototyped:
 - screen and report format, etc
- Note: *different* concept to 'function prototype' in C

8

Modularity

- Top-down analysis and design
- Break problem down into manageable sub-problems

Example:

```

    graph TD
      Invite[Invite to a party] --> RingUp[Ring up]
      Invite --> AskParty[Ask to party]
      Invite --> SayGoodbye[Say goodbye]
      RingUp --> FindPhone[Find phone number]
      RingUp --> DialNumber[Dial number]
      RingUp --> IntroduceSelf[Introduce self]
      AskParty --> Invite[Invite]
      AskParty --> SayWhen[Say when]
      AskParty --> SayWhere[Say where]
      SayGoodbye --> SayGoodbyeTask[Say goodbye]
      SayGoodbye --> HangUp[Hang up phone]
  
```

9

Modularity

- Top-down analysis and design
- Break problem down into manageable sub-problems

Module Hierarchy:

```

    graph TD
      Invite[invite to party] --> RingUp[ring up]
      Invite --> AskParty[ask to party]
      Invite --> SayGoodbye[say goodbye]
      RingUp --> SearchAddress[search address book]
      RingUp --> Greetings[greetings]
  
```

10

Golden Rule #1

Design Top-Down,
but *always* build Bottom-Up

- Code and test the **simplest components** first
- **Test each component** before using it to build more complex components

11

Structure Chart

```

    graph TD
      Invite[invite to party] --> RingUp[ring up]
      Invite --> AskParty[ask to party]
      Invite --> SayGoodbye[say goodbye]
      RingUp --> SearchAddress[search address book]
      RingUp --> Greetings[greetings]
  
```

- Pictorial representation of a modular structure
 - each function/module represented by a rectangle
 - arrows represent **dependencies** between them

12

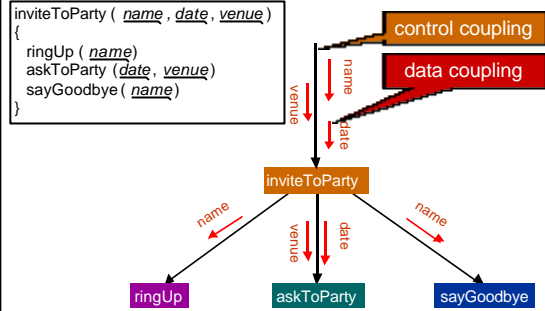
Coupling

- Links and dependencies between functions
- **Control coupling:**
 - when one function passes control to another
 - *Examples:* function call, return
- **Data coupling:**
 - sharing of data between functions
 - *Examples:* function parameters, return values

13

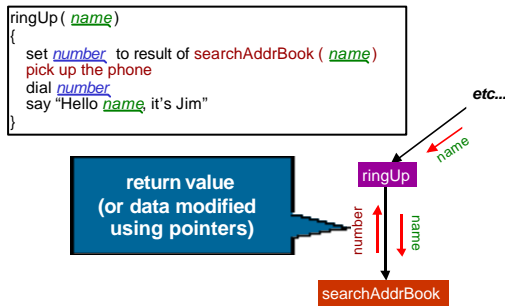
Example 1

Structure chart with labels showing data coupling



Example 2

Structure chart with labels showing data coupling



15

Notes on Coupling

- **Aim:** maximize module independence = minimize coupling
- Use of global variables (not constants) is a form of **implicit data coupling: dangerous!**
 - It is NOT recommended to have global variables in your program

16

Notes on Coupling (cont)

- How is control coupling represented in code?
 - **Function call:** When a function is called, control is passed to that function
 - **Function return:** When the code in a function has been executed, control is returned to the code that called the function
- How is data coupling represented in code?
 - Data sent to a function via **parameters**
 - Data **returned** from function using "return"
 - Data modified by pointers

17

Cohesion

- Cohesion: refers to *how closely related* the function's *internal parts* are
- **Logical cohesion (weak)**
 - Internal elements perform activities that are logically similar, e.g., I/O function
- **Functional cohesion (strong)**
 - all parts are geared towards a single activity
- **Aim:** high intra-module cohesion

18

Example 1

```
contact ( company, message, mode )  
{  
  if mode is by fax  
  {  
    sendFax ( company, message )  
  }  
  else if mode is by email  
  {  
    sendEmail ( company, message )  
  }  
}
```

Cohesive

19

Example 2

```
contact ( company, message, mode )  
{  
  if mode is by fax  
  {  
    sendFax ( company, message )  
  }  
  else if mode is by email  
  {  
    sendEmail ( company, message )  
  }  
  printAddressBook ( )  
}
```

Not cohesive

20

Golden Rule #2



Modules in a well-structured program are highly cohesive and loosely coupled

- The **size** of the data coupling corresponds roughly to the level of the module in the hierarchy

21

Summary

- **Modularity** is crucial for developing large software projects
- **Structure charts** are a useful design tool
- Design **top-down**, but build **bottom-up**
- Build incrementally with **prototypes**
- Well-structured programs are **highly cohesive**, and **loosely coupled**

22

Reading

- Brooks: Chapter 6
- King: Section 19.1

23