Modifying Code

Idaho State University | Computer Science

Isaac Griffith

CS 4422 and CS 5599
Department of Computer Science
Idaho State University

ROAR

Idaho State University
Computer Science

Outcomes

At the end of Today's Lecture you will be able to:

- Program in a maintainable way.
- Describe the maintenance activities.

ROAR

Idaho State University
Computer Science

Inspiration

“Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.” – John F. Woods

ROAR

Programming for Maintainability

1. **Understanding the Program**
2. Programming for Change
3. Coding Style

Idaho State University
Computer Science

Major Maintenance Activities

We Must understand an existing system before changing it

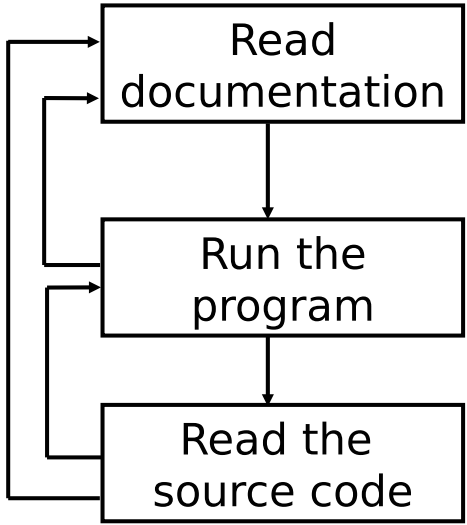
- How to accommodate the change?
- What are the potential ripple effects?
- What skills and knowledge are required?

1. **Identify** the change
   - What to change, why to change
2. **Manage** the process … what resources are needed?
3. **Understand** the program
   - How to make the change, determining the ripple effect
4. **Make** the change
5. **Test** the change
6. **Document** and record the change

ROAR

Idaho State University
Computer Science

Comprehension Process



Dynamic analysis

Static analysis

ROAR

What Influences Understanding?

- **Expertise**: Domain knowledge, programming skills
- **Program Structure**: Modularity, level of nesting
- **Documentation**: Readability, accuracy, up-to-date
- **Coding conventions**: Naming style, small design patterns
- **Comments**: Accuracy, clarity, and usefulness
- **Program presentation**: Good use of indentation and spacing

Programming for Maintainability

1. Understanding the Program
2. **Programming for Change**
3. Coding Style

Avoid Unnecessary Fancy Tricks

- Write for **humans**, not compilers
  - Fully **parenthesize** expressions
  - Pointer arithmetic is anti-engineering
  - Clever programming techniques are for children, not engineers
- In **1980**, computers were slow and memory expensive
  - **Control flow** dominated the running time
  - Hence the undergraduate CS emphasis on **analysis of algorithms**
- **Today**: Make it **easier** to **change** the program
  - Readable code is easier to **debug**, more **reliable**, and more **secure**
  - **Optimizing** compilers are far better than humans
  - **Overall architecture** usually dominates running time

Document Clearly

- Include **header blocks** for each method (**author** & **version**)
- Add a **comment** every time you stop to **think**
  - **Why** a method does something is more important than **what**
  - **What** is more important than **how**
- Document:
  - **Assumptions**
  - **Variables** that can be overridden by child methods
  - Reliance on default and superclass **constructors**
- Write **pseudocode** as comments, then write the method
  - **Faster** and more **reliable**
- Use a **version control** system with an edit history
  - **Explain why** each change was made clearly

Idaho State University
Computer Science

Use White Space Effectively

- A 1960s study asked "**how far should we indent**"
  - **2–4** characters is ideal
  - Fewer is **hard to see**
  - More makes programs **too wide**
- **Never use tabs** - they look different in every editor and printer
  - **Mixing** tabs and spaces is even worse
- Use plenty of **spaces**
  - `newList(x+y)=fName+space+lName+space+title;`
  - `newList(x + y) = fName + space + lName + space + title;`
- Don't put more than one statement per line

ROAR

Write Maintainable Java

- Be **tidy**
  - **Sloppy style** looks like **sloppy thinking**
  - Sloppy style creates **maintenance debt**
- Use clear **names**
  - Long names are simpler than short names
  - Don't make them so long they're hard to read
- Don't test for **error conditions** you can't handle
  - Let them **propagate** to someone who does

If you can't develop these habits, find a non-developer job

Java Specific Tips

- Implement **both or neither** `equals()` and `hashCode()`
  - Implementing just one can cause some very subtle faults
- Always **override** `toString()` to produce a **human-readable** description of the object
- If `equals()` is called on the wrong type, **return false**, not an exception
- If your class is **cloneable**, use `super.clone()`, not `new()`
  - `new()` will break if another programmer inherits from your class
- **Threads** are hard to get right and harder to modify
- Don't add **error checking** the VM already does
  - Array bounds, null pointers, etc.

Keep It Simple, (and) Stupid

- Long **methods** are not simple
  - Good programmers write **less code** not more
- **Bad designs** lead to more and **longer methods**
- Don't **generalize** unless it's necessary
- **Ten** programmers …
  - deliver **twice** as much code
  - **four times** as many faults, and
  - **half** the functionality as

… **five** programmers

Classes and Objects

The point of OO design is to look at nouns (data) first, then verbs (algorithms and methods)

- Think about **what** it **is**, not **what** it **does**
  - Class names should not be **verbs**
- Objects are defined by **state**—the class defines **behaviors**
- Lots of **switch statements** may mean the class is trying to do too many things
  - Use **Inheritance** or **Type Parameterization**
- Make methods that don't use class instance variables **static**
- Don't confuse **inheritance** with **aggregation**
  - **Inheritance** implements "**is-a**"
  - **Aggregation** implements "**has-a**"

Programming for Change

- The cost of writing a program is a small fraction of the cost of fixing and maintaining it…
- Don’t be lazy or selfish…
- Be an engineer!
- Remember that complexity is the number one enemy of maintainability

Programming for Maintainability

1. Understanding the Program
2. Programming for Change
3. **Coding Style**

Using Style Conventions

- Select a set of **style** conventions
  - Follow them **strictly**!
- Follow the **existing style** when making changes
  - **Even** if you do not like it
- **Lots** of style conventions are available
  - It is more important to be **consistent** than to have perfect style
- Programmers need to be told to follow the team's style

What Style Guides Tell Us

- Case for names
  - Variables, methods, classes, ...
- Guidelines for choosing names
- Width, special characters, and splitting lines
- Location of statements
- Organization of methods and use of types
- Use of variables
- Control structures
- Proper spacing and white space
- Comments

Summary

- Programming habits have a major impact on **readability**
- **Readability** has a major impact on **maintainability**
- **Maintainability** determines **long-term costs**

The minor decisions that engineers make determine how much money the company makes

That is what engineering means!

Are there any questions?