SOFTWARE DEVELOPMENT / CONSTRUCTION

The Programmer

- *Mostly, when you see programmers, they aren't doing anything. One of the attractive things about programmers is that you cannot tell whether or not they are working simply by looking at them. Very often they're sitting there seemingly drinking coffee and gossiping, or just staring into space. What the programmer is trying to do is get a handle on all the individual and unrelated ideas that are scampering around in his head.*

—Charles M. Strauss

The Fact on Construction

Code Complete of McConnell

- Construction typically makes up about 80 percent of the effort on small projects and 50 percent on medium projects.
- Construction accounts for about 75 percent of the errors on small projects and 50 to 75 percent on medium and large projects.
- Any activity that accounts for 50 to 75 percent of the errors presents a clear opportunity for improvement.

The Fact on Construction

Code Complete of McConnell

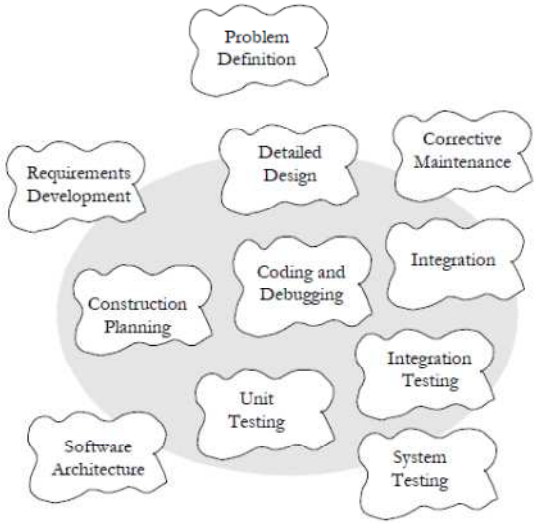
- Small-scale coding errors might be less expensive to fix than errors in requirements or architecture, but an inexpensive cost to fix obviously does not imply that fixing them should be a low priority.
- The claim that construction errors cost less to fix is true but misleading because the cost of not fixing them can be incredibly high.

The Irony of Construction

Code Complete of McConnell

- Requirements can be assumed rather than developed; architecture can be shortchanged rather than designed; and testing can be abbreviated or skipped rather than fully planned and executed.
- But, if there's going to be a program, there has to be construction and that makes it a uniquely fruitful area in which to improve development practices.

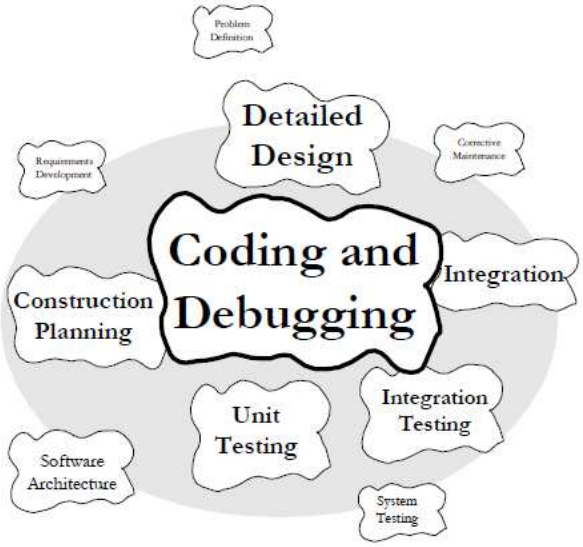
What is "Construction"?



Coding=Construction?

- Construction is also sometimes known as "coding" or "programming."
- "Coding" isn't really the best word because it implies the mechanical translation of a preexisting design into a computer language;
- Construction is not at all mechanical and involves substantial creativity and judgment.
- It is better to use "programming" interchangeably with "construction."

The Proportions



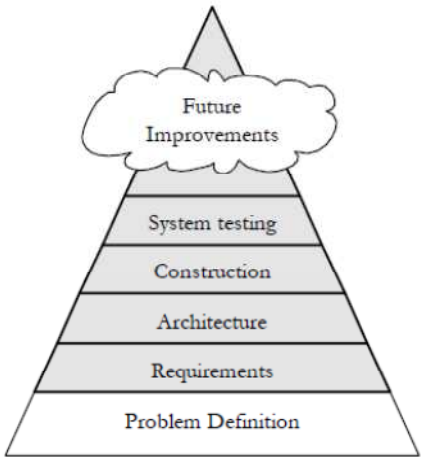
Tasks of Construction

- Verifying that the groundwork has been laid so that construction can proceed successfully
- Determining how your code will be tested
- Designing and writing classes and routines
- Creating and naming variables and named constants
- Selecting control structures and organizing blocks of statements
- Unit testing, integration testing, and debugging your own code
- Reviewing other team members' low-level designs and code and having them review yours
- Polishing code by carefully formatting and commenting it
- Integrating software components that were created separately
- Tuning code to make it smaller and faster

Why is Construction Important?

- *Construction is a large part of software development*
- *Construction is the central activity in software development*
- *With a focus on construction, the individual programmer's productivity can improve enormously*
- *Construction's product, the source code, is often the only accurate description of the software*
- *Construction is the only activity that's guaranteed to be done*

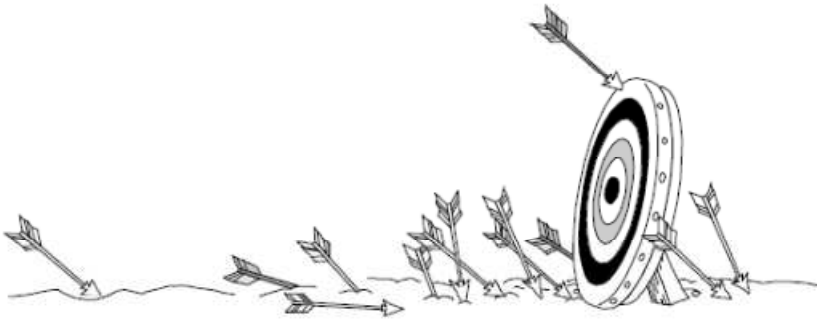
The Foundation



Bad Problem Definition

- *Without a good problem definition, you might put effort into solving the wrong problem. Be sure you know what you're aiming at before you shoot*

Bad Requirement



- *Without good requirements, you can have the right general problem but miss the mark on specific aspects of the problem*

Bad Architecture

- *Without good software architecture, you may have the right problem but the wrong solution. It may be impossible to have successful construction.*

Code's Audiences

- Your code has two audiences:
  - The machine that's the target of the compiled version of the code, what will actually get executed.
  - The people, including yourself, who will *read it in order to understand it and* modify it.
- Your code needs to fulfill the requirements, implement the design, and also be readable and easy to understand

What's Wrong with the Code?

```
void HandleStuff(CORP_DATA inputRec, int crntQtr, EMP_DATA empRec, Double estimRevenue,
   double ytdRevenue, int screenx, int screeny, Color newColor, Color prevColor, StatusType
   status, int expenseType) {
int i;
for ( i = 0; i < 100; i++ )
        {
        inputRec.revenue[i] = 0;
        inputRec.expense[i] = corpExpense[crntQtr][i];
        }
UpdateCorpDatabase( empRec );
estimRevenue = ytdRevenue * 4.0 / (double) crntQtr;
newColor = prevColor;
status = SUCCESS;
if ( expenseType == 1 ) {
        for ( i = 0; i < 12; i++ )
                profit[i] = revenue[i] - expense.type1[i];
        }
else if ( expenseType == 2 ) {
                profit[i] = revenue[i] - expense.type2[i];
        }
else if ( expenseType == 3 )
                profit[i] = revenue[i] - expense.type3[i];
                }
```

What's Wrong with the Code?

- You are not writing for the compiler here, you are writing for the human. Visibility modifiers make things explicit for the human reader.
- The method name is terrible. HandleStuff doesn't tell you anything about what the method does.
- The method does too many things. It seems to compute something called profit based on an expenseType. But it also seems to change a color and indicate a success. Methods should be small. They should do just one thing.

What's Wrong with the Code?

- Where are the comments? There is no indication of what the parameters are or what the method is supposed to do. All methods should tell you at least that.
- The layout is just awful. And it's not consistent. The indentation is wrong.
- The method doesn't protect itself from bad data. If the crntQtr variable is zero, then the division in line 8 will return a divide-by-zero exception.

What's Wrong with the Code?

- The method uses magic numbers including 100, 4.0, 12, 2, and 3. Where do they come from? What do they mean? Magic numbers are bad.
- The method has way too many input parameters. If we knew what the method was supposed to do maybe we could change this.
- There are also at least two input parameters – screenx and screeny – that aren't used at all. This is an indication of poor design; this method's interface may be used for more than one purpose and so it is "fat," meaning it has to accommodate all the possible uses.

What's Wrong with the Code?

- The variables corpExpense and profit are not declared inside the method, so they are either instance variables or class variables. This can be dangerous. Because instance and class variables are visible inside every method in the class, we can also change their values inside any method, generating a side effect. Side effects are bad.
- Finally, the method doesn't consistently adhere to the Java naming conventions.

Function, Method, & Size

- Your classes, functions, and methods should all *do just one thing.* This is the fundamental idea behind encapsulation.
  - Having your methods do just one thing isolates errors and makes them easier to find.
  - It encourages re-use because small, single feature methods are easier to use in different classes.
  - Single feature (and single layer of abstraction) classes are also easier to re-use.

Function, Method, & Size

- Single feature implies small. Your methods/ functions should be small. And the small mean – 20 lines of executable code is a good upper bound for a function.
  - Smaller function is easy to maintain.
  - Smaller function is easy to test, because it requires fewer unit test.

"Small is beautiful."

Formatting, Layout, and Style

- Formatting, layout, and style are all related to how your code looks on the page. It turns out that how your code looks on the page is also related to its correctness.
- Good visual layout shows the logical structure of a program (McConnell)
- Good visual layout not only makes the program more readable, it helps reduce the number of errors because it shows how the program is structured

Formatting, Layout, and Style

- The converse is also true; a good logical structure is easier to read.
- The objectives of good layout and formatting should be:
  - to accurately represent the logical structure of your program;
  - to be consistent so there are few exceptions to whatever style of layout you've chosen;
  - to improve readability for humans; and
  - to be open to modifications. (You do know you're code is going to be modified, right?)

White Space

- White space is your friend. You wouldn't write a book with no spaces between words, or line breaks between paragraphs, or no chapter divisions.
- Some suggestions:
  - Use blank lines to separate groups (just like paragraphs).
  - Within a block align all the statements to the same tab stop (the default tab width is normally four spaces).
  - Use indentation to show the logical structure of each control structure and block.
  - Use spaces around operators.
  - In fact, use spaces around array references and function / method arguments as well.
  - Do not use double indentation with begin-end block boundaries.

Defensive Programming

- By defensive programming we mean that your code should protect itself from bad data.
- The bad data can come from user input via the command line, a graphical text box or form, or a file.
- Bad data can also come from other routines in your program via input parameters.
- How do you protect your program from bad data? **Validate! Validate! Validate!**

Defensive Programming

- Check the following:
  - Check the number and type of command line arguments.
  - Check file operations.
    - Did the file open?
    - Did the read operation return anything?
    - Did the write operation write anything?
    - Did we reach EOF yet?
  - Check all values in function/method parameter lists. Are they all the correct type and size?
  - You should always initialize variables and not depend on the system to do the initialization for you

Defensive Programming

- Also check the following:
  - Null pointers (references in Java, C++)
  - Zeros in denominators
  - Wrong type
  - Out of range values
- Use *assertion*. Defensive programming means that using assertions is a great idea if your language supports them. It is good for testing and debugging.

Defensive Programming

```
int total = countNumberOfUsers();
if (total % 2 == 0) {
        // total is even
} else {
        // total is odd and non-negative
        assert(total % 2 == 1);
}
```


Error Handling

- The main purpose of error handling is to have your program survive and run correctly for as long as possible.
- When it gets to a point where your program cannot continue, it needs to report what is wrong as best as it can and then exit gracefully.
- Exiting is the last resort for error handling.
- Some programming languages have built-in error reporting systems that will tell you when an error occurs, and leave it up to you to handle it one way or another. These errors that would normally cause your program to die a horrible death are called *exceptions.*

Programmer's Happiness

I am rarely happier than when spending an entire day programming my computer to perform automatically a task that it would otherwise take me a good ten seconds to do by hand.

—Douglas Adams

Debugging

- *It is a painful thing to look at your own trouble and know that you yourself and no one else has made it.*

—Sophocles

Debugging

- Getting your program to work is a process with three parts:
  - Debugging
  - Reviewing/inspecting
  - Testing

Debugging

- *Debugging* is the process of finding the root cause of an error and fixing it.
- This doesn't mean treating the symptoms of an error by coding around it to make it go away; it means to find the real reason for the error and fixing that piece of code so the error is removed.
- Debugging is normally done once you finish writing the code and before you do a code review or unit testing.

Reviewing

- Reviewing (or inspecting) is the process of reading the code as it sits on the page and looking for errors.
- The errors can include errors in how you've implemented the design, other kinds of logic errors wrong comments, etc.
- Reviewing code is an inherently static process because the program isn't running on a computer – you're reading it off a screen or a piece of paper. So although reviewing is very good for finding static errors, it can't find dynamic or interaction errors in your code.

Testing

- Testing, is the process of finding errors in the code, as opposed to fixing them, which is what debugging is all about.
- Testing occurs, at minimum, at the following three different levels:
  - *Unit testing: Where you test small pieces of your code, notably at the function or* method level.
  - *Integration testing: Where you put together several modules or classes that relate* to each other and test them together.
  - *System testing: Where you test the entire program from the user's perspective; this* is also called *black-box testing, because the tester doesn't know how the code was* implemented, all they know is what the requirements are and so they're testing to see if the code as written implements all the requirements correctly.

Aspek yang Perlu Diperhatikan

- Pemilihan Teknologi
  - Desktop vs Web-based programming
  - Server-side vs Client side
  - Windows vs Linux
  - Open Source vs Proprietary Source
  - Graphical UI vs Textual-based UI
  - Mobile App vs Immobile App
  - Oracle | SQL Server vs MySQL | PostgreSQL
  - Modular (OOP) vs Traditional
  - MVC framework vs Traditional