# Software Design

Michael Hilton

institute for SOFTWARE RESEARCH | Carnegie Mellon University School of Computer Science

# About Me

Assistant Teaching Professor, ISR

Teach classes such as

- 17-313 Foundations of Software Engineering
- 17-356 Software Engineering for Startups

Director of SE minor, SE concentration

Worked for >10 years as a professional developer,

- ...but I am not a C wizard

# What is (good) Software Design?

# Definition of software design

A good software design goes beyond functional correctness, and considers such quality attributes as:

– Performance
– Availability
– Modifiability, portability
– Scalability
– Security
– Testability
– Usability
– Cost to build, cost to operate

## Controversial Statement:

# A good design is a **testable** design

# Good Design is about:
# Complexity Management
# &
# Communication

# Complexity Management

There are well known limits to how much complexity a human can manage easily.

VOL. 63, NO. 2 MARCH, 1956

THE PSYCHOLOGICAL REVIEW

THE MAGICAL NUMBER SEVEN, PLUS OR MINUS TWO:
SOME LIMITS ON OUR CAPACITY FOR
PROCESSING INFORMATION [1]

GEORGE A. MILLER
*Harvard University*

# Complexity Management

However, patterns can be very helpful…

COGNITIVE PSYCHOLOGY 4, 55–81 (1973)

## Perception in Chess[1]

WILLIAM G. CHASE AND HERBERT A. SIMON
*Carnegie–Mellon University*

This paper develops a technique for isolating and studying the perceptual structures that chess players perceive. Three chess players of varying strength — from master to novice — were confronted with two tasks: (1) A perception task, where the player reproduces a chess position in plain view, and (2) de Groot's (1965) short-term recall task, where the player reproduces a chess position after viewing it for 5 sec. The successive glances at the position in the perceptual task and long pauses in the memory task were used to segment the structures in the reconstruction protocol. The size and nature of these structures were then analyzed as a function of chess skill.

# Complexity Management

Many techniques have been developed to help manage complexity:

- Separation of concerns
- Modularity
- Reusability
- Extensibility
- DRY
- Abstraction
- Information Hiding
- …

# Communication

When writing code, the author is communicating with:

- The machine
- Other developers of the system
- Code reviewers
- Their future self

# Communication

There are many techniques that have been developed around code communication:

- Comments
- Naming
- Tests
- Commit Messages
- Code Review
- Design Patterns
- ...

# Why is a testability the measure of a good design?

"The act of writing a unit test is more an act of **design** than of verification.

It is also more an act of **documentation** than of verification.

The act of writing a unit test closes a remarkable number of feedback loops, the least of which is the one pertaining to verification of function".

-Agile Software Development, Principles, Patterns, and Practices

# Why are CONTRACTS the measure of a good design?

"The act of writing a **contract** is more an act of **design** than of verification.

It is also more an act of **documentation** than of verification.

The act of writing a **contract** closes a remarkable number of feedback loops, the least of which is the one pertaining to verification of function".

-Agile Software Development, Principles, Patterns, and Practices

# Testing vs Contracts

Testing is more commonly used in industry.

The most important thing is to think about your code, document intentions, design it well.

I will talk more about testing because that is my area of expertise, but that contracts are very valuable, and you should be using them also.

Testing and Contracts should not be seen as competitors, but as collaborators towards better code.

# Running Example: Battleship

```
Enter Board Size:10
Ready to play
 ,1,2,3,4,5,6,7,8,9,10
a _ _ _ _ _ _ _ _ _ _
b _ _ _ _ _ _ _ _ _ _
c _ _ _ _ _ _ _ _ _ _
d _ _ _ _ _ _ _ _ _ _
e _ _ _ _ _ _ _ _ _ _
f _ _ _ _ _ _ _ _ _ _
g _ _ _ _ _ _ _ _ _ _
h _ _ _ _ _ _ _ _ _ _
i _ _ _ _ _ _ _ _ _ _
j _ _ _ _ _ _ _ _ _ _
enter coordinates:
```

# Quick Demo

# Concerns about design?

## Is the complexity manageable?

## Is it clear what the author wanted to do?

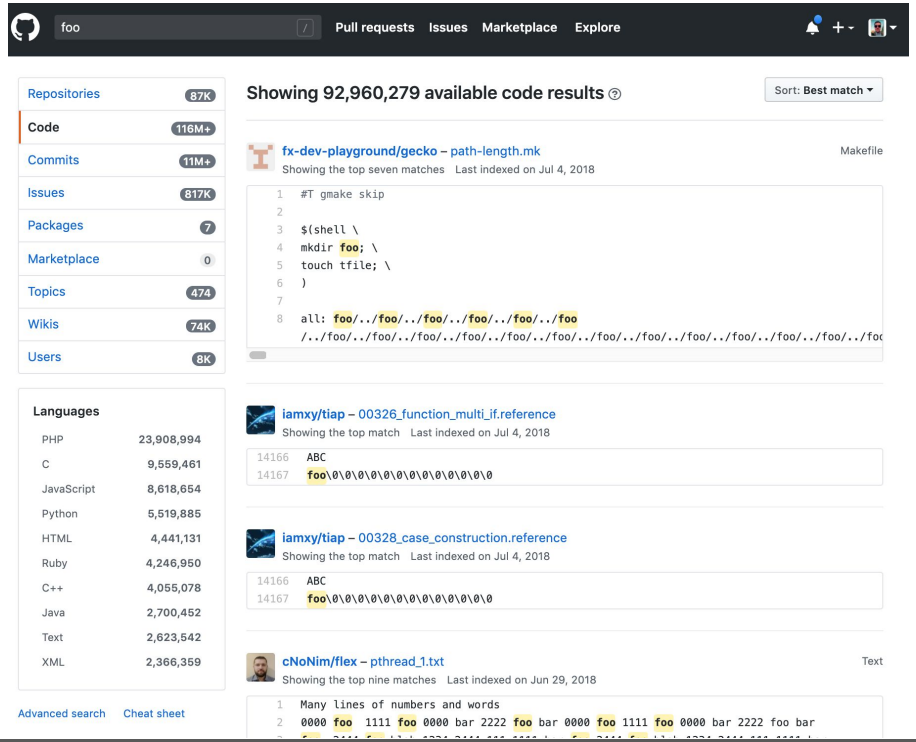
# Activity

Observe the changes to the design as we attempt to test.

# Naming

```
static void print(Brd *b)
{
   printf(" ");
   int i;
   for (i = 1; i <= b->len; i ++)
   {
       printf(",%d", i);
   }
   printf("\n");
   int k;
   for (k = 0; k < b->len; k ++)
   {
       printf("%c", 'a' + k);
       int j;
       for (j = 0; j < b->len; j ++)
       {
           printf(" %c", b->data[k][j]->s);
       }
       printf("\n");
   }
}
```

# Avoid deliberately meaningless names:



# Naming is understanding

"If you don't know what a thing should be called, you cannot know what it is.

If you don't know what it is, you cannot sit down and write the code."

-Sam Gardiner

# Better naming practices

1. Start with *meaning* and *intention*
2. Use words with precise meanings (avoid “data”, “info”, “perform”)
3. Prefer fewer words in names
4. Avoid abbreviations in names
5. Use code review to improve names
6. Read the code out loud to check that it sounds okay
7. Actually rename things

# Naming guidelines - Use dictionary words

Only use dictionary words and abbreviations that appear in a dictionary.

For example: FileCpy -> FileCopy

Avoid vague abbreviations such as acc, mod, auth, etc..

# Avoid using single-letter names

Single letters are unsearchable

Give you no hints as to its use.

Exceptions are loop counters

# Limit Name character length

“constraints breed creativity”

“Good naming limits individual name length, and reduces the need for specialized vocabulary” - Philip Relf

# Limit name word count

Keep names to a four word maximum

Limit names to the number of words that people can read at a glance.

# Describe Meaning

Use descriptive names.

Avoid names with no meaning: a, foo, blah, tmp, etc

# Use a large vocabulary

Be more specific when possible:

Person -> Employee

# Use problem domain terms

Use the correct term in the problem domain's language.

(HINT, as a student, consider the terms in the assignment)

# Use opposites precisely

Consistently use opposites in standard pairs.

first/end -> first/last

# Comments

1) Don’t say what the code does (because the code already says that)
2) Don’t explain awkward logic (improve the code to make it clear)
3) Don’t add too many comments (it’s messy, and they get out of date)

# Explain why code exists

When should I use this code?

When shouldn't I use it?

What are the alternatives to this code?

```
static void print(Brd *b)
{
   printf(" ");
   int i;
   for (i = 1; i <= b->len; i ++)
   {
       printf(",%d", i);
   }
   printf("\n");
   int k;
   for (k = 0; k < b->len; k ++)
   {
       printf("%c", 'a' + k);
       int j;
       for (j = 0; j < b->len; j ++)
       {
           printf(" %c", b->data[k][j]->s);
       }
       printf("\n");
   }
}
```

**Quiz:**
**https://www.cs.cmu.edu/~213/variableNaming**

# How to write good comments

1. Try to write good code first
2. Try to write a one-sentence comment
3. Refactor the code until the comment is easy to write
4. Now write a good comment
5. Maintain your code AND your comments