SENG 265

Amanda Dash, CET, BSeng, PhD Candidate
University of Victoria

Topics for the Week

- The Search for Bugs
  - Whom To Trust
  - Evaluation
  - Debugging Tips
- Development for the Lazy
  - The Three Virtues
  - Time Management
  - Task Estimation
    - The Dreaded Last 10%
  - Code Quality
- Case Study: Battlesnake

The Search for Bugs

“There are two ways to write error-free programs; only the third one works.”

-- Alan Perlis

The Search for Bugs: Bugs

- From Lecture 8 slides:
  - Validate your inputs
  - Assertions to guarantee “impossible” scenarios don’t occur
  - Reuse reliable code
  - Check Results
  - KISS (Keep It Simple Stupid)
  - Watch out for gotchas that produce no errors

The Search for Bugs: Bugs

- From Lecture 8 slides:
  - Validate your inputs
  - Assertions to guarantee “impossible” scenarios don’t occur
  - Reuse reliable code
  - Check Results
  - KISS (Keep It Simple Stupid)
  - Watch out for gotchas that produce no errors
- If bugs can’t be avoided, then how to find them?
  - Debugger (i.e. gdb)
  - Rubber Ducky Debugging
  - Print statements

The Search for Bugs: Whom to Trust?

- When we program, we are inherently trusting:
  - The computer hardware and operating system
  - The programming language and interpreter/compiler
  - The programming framework
  - User input or data
  - Teammates, Coworkers and Stack-Overflow
  - Yourself

The Search for Bugs: Whom to Trust?

- When we program, we are inherently trusting:
  - The computer hardware and operating system
  - The programming language and interpreter/compiler
  - The programming framework
  - User input or data
  - Teammates, Coworkers and Stack-Overflow
  - Yourself
- But **should** we place our trust?

The Search for Bugs: Whom to Trust?

- Pragmatic Paranoia or Defensive Programming
  - The Pragmatic Programmer: From Journeyman to Master by Andrew Hunt and David Thomas
  - Anticipate problem that *may* occur instead of being surprised when they do
- Key Elements:
  - Ask yourself: What if?
    - If X happens, does you code handle it?
    - What could go wrong (WCGW)?
  - Consider and think about boundary decisions carefully
  - Test driven development only works if
    - a) the tests are good
    - b) you know what your testing for
  - And most importantly, **trust no one's code.**

WCGW: The Hardware and OS

More Likely

Less Likely

- Embedded Systems
  - Cars, Airplanes, Satellites
  - Internet-of-Things (Smart Watches, Amazon Echo)
- Mobile Computers
  - Custom Operating Systems
  - Limited Resources
- Desktop Computers
  - Hard Drive and RAM
  - Operating System updates
- Virtual Services
  - Amazon AWS, Google Cloud, Hyper-V
  - VirtualBox

Image source:
https://embedded.fm/blog/2016/12/6/an-introduction-to-the-tricky-parts-of-embedded-systems-1

WCGW: Languages

- Some programming languages have been around for a long time:
  - C (1972-1973), Python (1991)
- Some are much newer!
  - Rust 1.0 (2015), Swift (2014), Golang (2009)
- Each language has its own pros, cons, and quirks
- Programming language behaviour change over time, be careful on which version you use!

Image source: https://datafloq.com/read/7-new-programming-languages-that-you-should-know/3385

WCGW: Python Example

```
def foo(bar=[]):      # bar is optional, defaults to []
    bar.append("baz")
    return bar
```

```
>>> foo()
["baz"]
>>> foo()
```

What will be the output?

source: https://www.toptal.com/python/top-10-mistakes-that-python-programmers-make

WCGW: Python Example

```
def foo(bar=[]):      # bar is optional, defaults to []
    bar.append("baz")
    return bar
```

Why does this occur?

```
>>> foo()
["baz"]
>>> foo()
["baz", "baz"]
```

source: https://www.toptal.com/python/top-10-mistakes-that-python-programmers-make

WCGW: Python Example

```
def foo(bar=[]):      # bar is optional and defaults to []
    bar.append("baz")
    return bar
```

The default value for a function argument is only evaluated once, at the time that the function is defined.

Solution?

```
>>> foo()
["baz"]
>>> foo()
["baz", "baz"]
```

source: https://www.toptal.com/python/top-10-mistakes-that-python-programmers-make

WCGW: Python Example

```
def foo(bar=None):
    If bar is None: bar = []
    bar.append("baz")
    return bar
```

Knowing why the error occurs, we can easily apply a fix.

```
>>> foo()
["baz"]
>>> foo()
["baz"]
```

source: https://www.toptal.com/python/top-10-mistakes-that-python-programmers-make

WCGW: C Example

```
int main() {
     char *s = "foobar";
     s[0] = 'g';
     printf("%s\n", s);
     return 0;
}
```

```
>>> ./main
```

What will be the output?

source: https://www.toptal.com/python/top-10-mistakes-that-python-programmers-make

WCGW: C Example

```
int main() {
     char *s = "foobar";
     s[0] = 'g';
     printf("%s\n", s);
     return 0;
}
```

```
>>> ./main
```

Undefined behaviour (likely it will SEGFAULT)

Why?

source: https://www.toptal.com/python/top-10-mistakes-that-python-programmers-make

WCGW: C Example

```
int main() {
     char *s = "foobar";
     s[0] = 'g';
     printf("%s\n", s);
     return 0;
}
```

```
>>> ./main
```

The string literal may be stored in read-only (unchangeable) or read-write memory.

source: https://www.toptal.com/python/top-10-mistakes-that-python-programmers-make

WCGW: C Example

```
int main() {
     char s[] = "foobar";
     s[0] = 'g';
     printf("%s\n", s);
     return 0;
}
```

```
>>> ./main

goobar
```

The string literal is copied over to the array (into read-write memory).

source: https://www.toptal.com/python/top-10-mistakes-that-python-programmers-make

WCGW: Programming Framework

- Frameworks are “wrappers” for a programming language where common tasks are already done for you
  - Very similar to library functions (C) and modules (Python)
  - Web Development
    - NodeJS, React, .NET
  - Deep Learning
    - PyTorch, TensorFlow
  - Mobile App Development
    - Xamarin, Cordova
- **Review known/bugs issues:**
  - **And not all bugs produce errors**
  - **And not all errors have solutions**

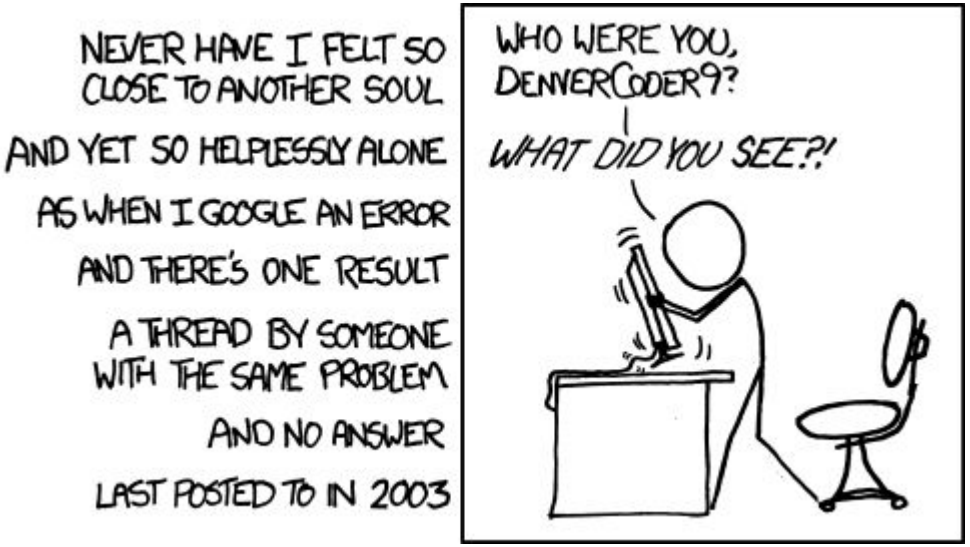


Image source: xkcd.com

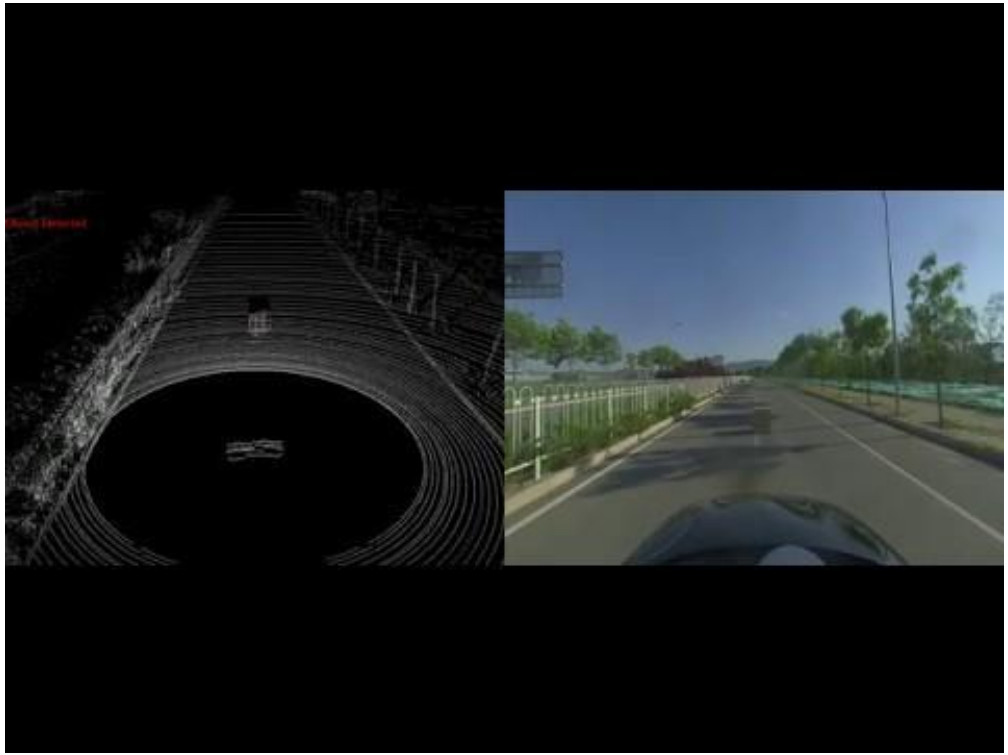
“Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.”

— Rick Cook, The Wizardry Compiled

WCGW: Users and Data

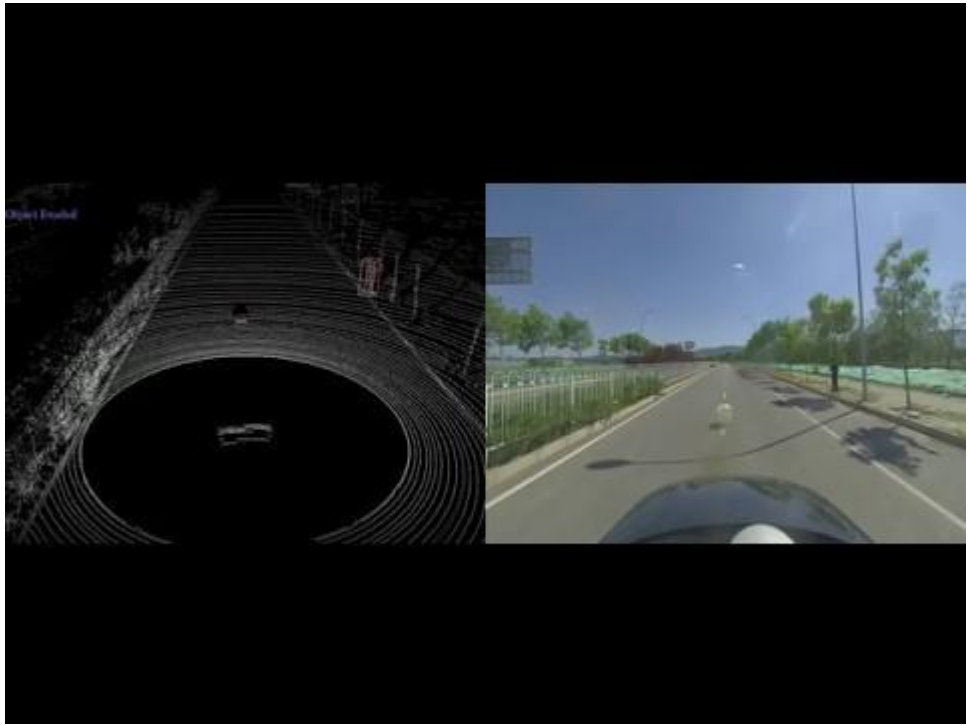
- User input and data can have many problems:
  - Garbage in Garbage out
  - Provided data not in expected format
    - Data corruption
    - Data with errors
  - User not using program as expected
    - Incorrect use
    - Missing data
    - Malicious intent
      - Adversarial attacks

Image source: https://xkcd.com/1838/



Autonomous Car Detecting a Box

Video Source: https://sites.google.com/view/lidar-adv



Autonomous Car Detecting an Adversarial Object

Video Source: https://sites.google.com/view/lidar-adv

WCGW: Yourself

- Failure of Imagination
  - Have you considered all possible scenarios?
  - Are you checking/asserting/validating?
  - Do you understand the problem correctly?
- Do you have a reason for what you're doing
  - Don't use code you don't understand
  - Don't program what you don't understand

WCGW: Yourself

- Instincts and Assumptions
  - **Don’t assume it -- Prove it**
- Who is evaluating the performance?
  - If the answer is only you, you have problems
  - Always follow the production/evaluation process

Ask for Help!

Evaluating your Program

- How do you evaluate how **good** your program is?
- “Good” could be defined as:
  - Fulfils the requirements of the user/customer without bugs
    - Are you solving the problem required to be solved?
  - Runs in X time or uses X resources
    - i.e. uses little memory or runs in less than a second
  - Better than human ability
  - Better than State-of-the-Art (SOTA) or your competitors
- Sometimes you define this, but usually it’s by:
  - Your boss or team leader or Professors
  - The customer
    - Usually done through requirements gathering

Evaluation

- Using Test Cases
  - These test cases are either provided to you or you write your own
    - Writing your own is very good practice as it require you to completely understand the problem you are trying to solve
  - These are inputs or scenarios that should test:
    - Where could the program break
    - What are the most important scenarios required to pass
    - Scalability (can your program handle large number of users)
- Reduce the performance of your program to a single value:
  - Easy to interpret and visualize (i.e. graphs)
  - Fixed value range (i.e. 0-1)
  - Have inherent meaning (i.e. probability, accuracy)

How often am I right

- Pass the given N tests, and you get mark X
  - This is basically the **accuracy** of your program
  - To get 100% on your assignments, you need to pass 100% of the tests
- In industry, all test cases must pass.
- **Why isn't 99% accuracy enough?**
  - Why is accuracy not always a good way to evaluate a program?

How often am I right

- Pass the given N tests, and you get mark X
  - This is basically the **accuracy** of your program
  - To get 100% on your assignments, you need to pass 100% of the tests
- In industry, all test cases must pass.
- **Why isn't 99% accuracy enough?**
  - Why is accuracy not always a good way to evaluate a program?

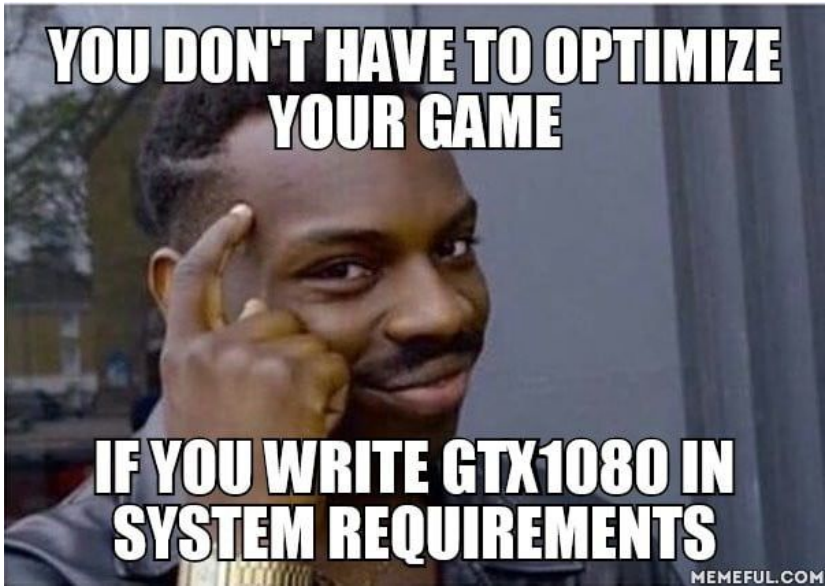
What if the 1% of tests that fail cause a system crash or security breach?

What if I am wrong?

- For example, for medical programs is it more important that you are:
  - Usually right, or
  - Rarely/never wrong
- Evaluate when you are wrong
  - Pay a cost of Y if your N is wrong
  - For example:
    - performance = 0.20 * (# of times I'm right)
      + 0.80 * (# of times I'm wrong)
- Minimum Viable Product
  - What is the most important aspect of your project? What **must** work
  - What do I need to work to get the grade I want?

Optimization

- Optimization is making some aspect of a program or algorithm work more efficiently or use fewer resources
  - Use less memory
  - Draw less power
  - Execute faster
  - ...
- Make it work and then make it pretty
  - Optimize after it works so you know it still works after you optimize



“I'M NOT A GREAT PROGRAMMER;
I'M JUST A GOOD PROGRAMMER
WITH GREAT HABITS.”

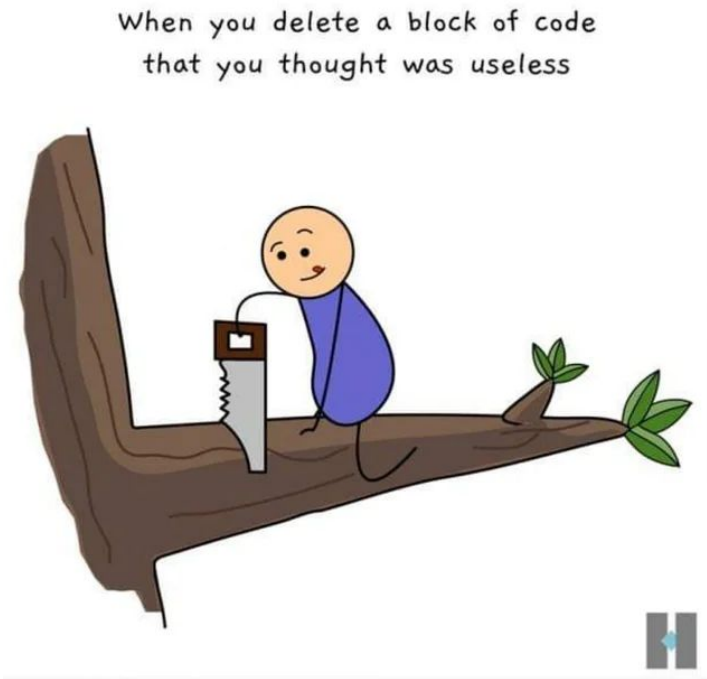
- Kent Beck

Debugging Tips

- If you don’t know what or where the bug occured, you can’t fix it
  - If test X failed, what did test X check?
  - Track down **exactly** what your program is doing
- Design your code to be testable
- Think and Ask!
  - Programming is **putting thoughts to code**
  - If your code is wrong, then your thinking was likely wrong

Debugging Tips

- Perform one “crime” at a time
  - When fixing a bug, only fix/change **one thing at a time and test**
- Always use source control
  - Tag any test successes and failures with your commit hash so you know which version of the code worked or failed
  - **Never delete code**
    - Comment it out until you’re sure it works



Debugging Tips

- Refractor early and often
  - Well-written and understandable code is much easier to debug
  - Code “debt” accumulates over time
- Test early. Test often. Test Well.
  - Badly written tests will slow or halt development
  - Automation removes human error
    - Error compound on each other
    - How many times have you been testing only to realize you forgot to recompile?
  - Don’t “eyeball” your solutions to see if you they’re correct
    - Why? Let’s look at an example

I’m writing a program and A and B are supposed to be the same color and eyeballing the output I see a bug.

But I can’t find the bug in the code!

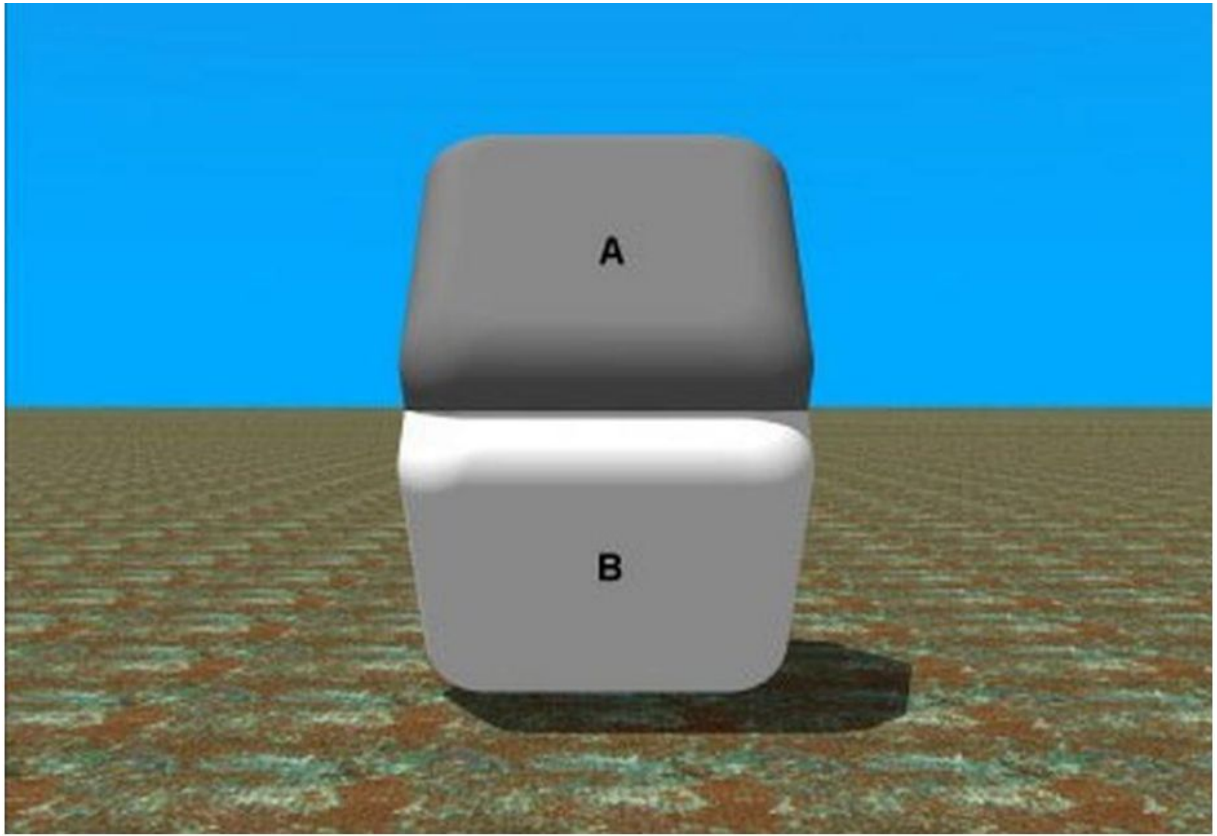


Image source: https://nerdist.com/5-optical-illusions-that-show-you-why-your-brain-messes-with-the-dress/

I’m writing a program and A and B are supposed to be the same color.

There was never a bug, my “eyeballing” error checking failed.

Usually, a bug exists that you don’t see (like extra white spaces).

Remove human error from testing as much as you can.

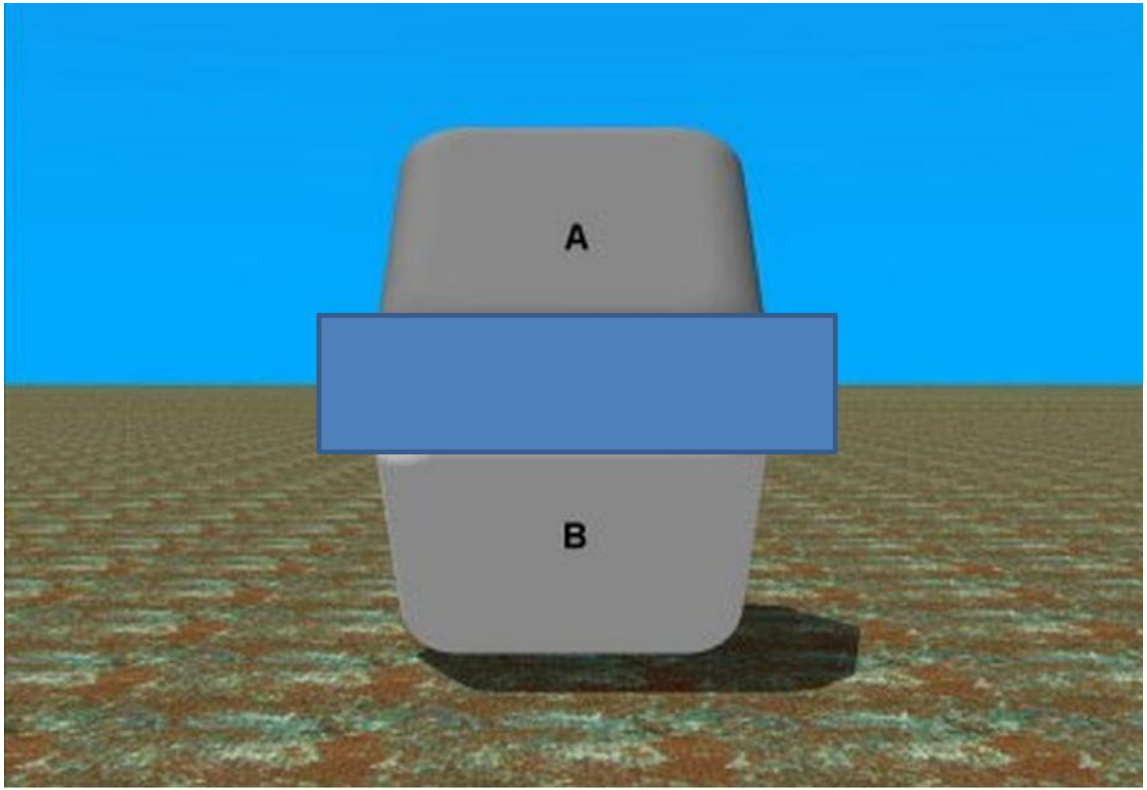


Image source: https://nerdist.com/5-optical-illusions-that-show-you-why-your-brain-messes-with-the-dress/

Development for the Lazy

"I realized that, paradoxically enough, good programmers need to be both *lazy* and *dumb*."

-- Philip Lenssen

The Three Virtues

1. **Laziness**: The quality that makes you go to great effort to reduce overall energy expenditure. It makes you write labor-saving programs that other people will find useful and document what you wrote so you don't have to answer so many questions about it.
2. **Impatience**: The anger you feel when the computer is being lazy. This makes you write programs that don't just react to your needs, but actually anticipate them. Or at least pretend to.
3. **Hubris**: The quality that makes you write (and maintain) programs that other people won't want to say bad things about.

-- Larry Wall, Programming Perl

The Three Virtues

1. **Laziness**: The quality that makes you go to great effort to reduce overall energy expenditure. It makes you write labor-saving programs that other people will find useful and document what you wrote so you don't have to answer so many questions about it.
2. **Impatience**: The anger you feel when the computer is being lazy. This makes you write programs that don't just react to your needs, but actually anticipate them. Or at least pretend to.
3. **Hubris**: The quality that makes you write (and maintain) programs that other people won't want to say bad things about.

-- Larry Wall, Programming Perl

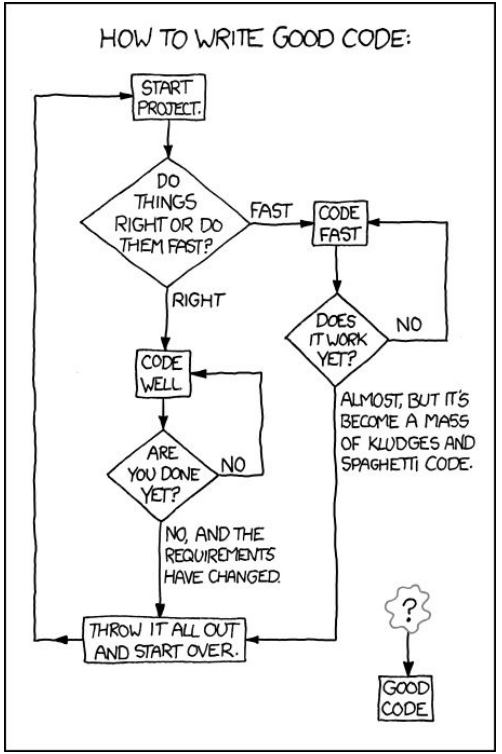
The Three Virtues

1. **Laziness**: The quality that makes you go to great effort to reduce overall energy expenditure. It makes you write labor-saving programs that other people will find useful and document what you wrote so you don't have to answer so many questions about it.
2. **Impatience**: The anger you feel when the computer is being lazy. This makes you write programs that don't just react to your needs, but actually anticipate them. Or at least pretend to.
3. **Hubris**: The quality that makes you write (and maintain) programs that other people won't want to say bad things about.

-- Larry Wall, Programming Perl

Laziness: Time Management

- How much time you *think* you have is usually very, very wrong
- Assignment #3 was posted on July 9th and was due July 23rd (2 weeks)
- How much time did you *really* have to work on it?



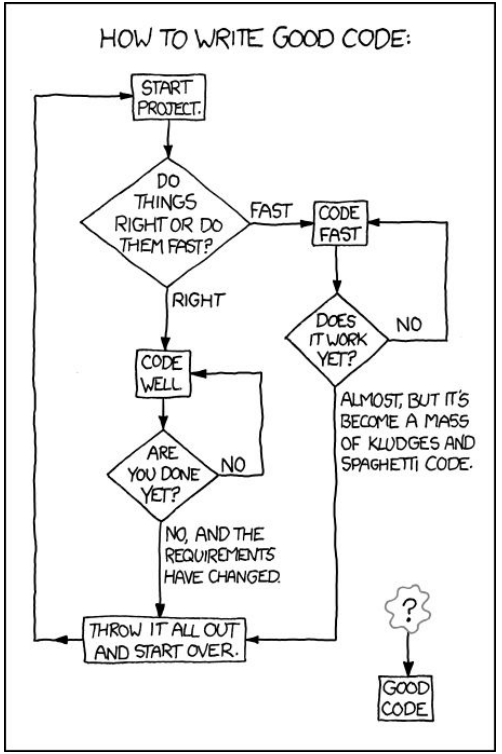
Laziness: Time Management

Total hours: 336 hours (14 * 24)

Non-productive hours:

- Sleeping: 14 nights * 7 hours = 98 hours
- Classes: 3 hrs/class * 5 classes * 2 weeks = 30 hours
- Travel: 1 hr/day * 14 days = 14 hours
- Eating,Housework,etc: 2hrs/day * 14 days = 28 hours

= 170 hours What am I missing?



Laziness: Time Management

Total hours: 336 - 170 = 166 hours

- Part-time or Full-time job
- Sickness
- Concentration limits and breaks
- Labs, Homework, Exams
- Clubs, Hobbies and Having a life

Roughly 20-25 hours available in a 2 week period to work on an assignment.

Laziness: Time Management

Total hours of help available: ??

- Professor office hours
- Assistance Centre hours
- Labs

The time when help is available to you is much smaller and restricted than your total hours to work on the assignment.

Impatience: Task Estimation

- Trickiest part: How long do you think a task will take?
  - Is this best case? Worst Case?
    - Your guess should be in the middle
  - Skill that takes practise
    - Marketable skill in the job market
- In what order do you need to do the tasks?
  - Industry: Task management software like Jira
  - Academia: Which assignments are due first? Which will take longer?

Impatience: Task Estimation

- Ask yourself
  - Do you already know how to solve the problem?
  - Do you know where to look for the solution?
  - Are you very familiar with the programming language?
  - Have you double checked that your understanding of the requirements is correct?
    - Really? Or are you just assuming?
    - Code that no one wants is useless code
- The Dreaded Final 10% of a project

The Ninety-Ninety Rule

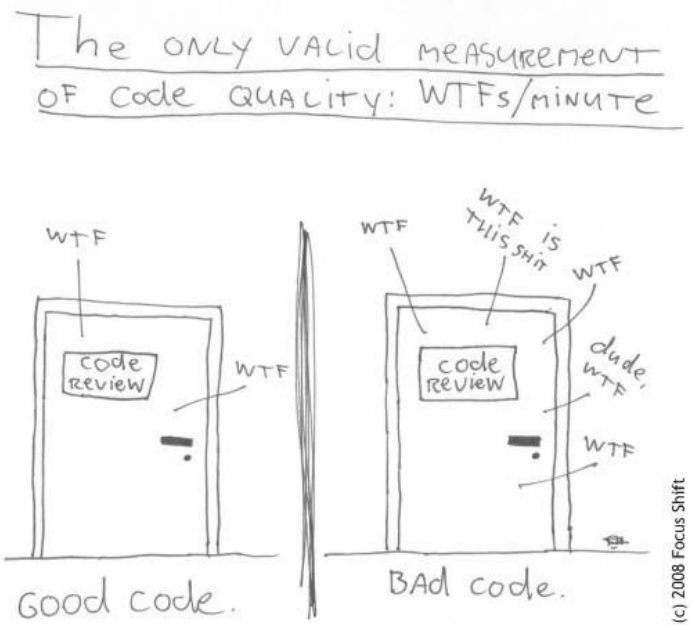
"The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time."

— Tom Cargill, Bell Labs

- Results from failure to anticipate difficult, often unpredictable, complexities
  - Inadequate test coverage
- Often involves lots of code that is "relatively done"

Hubris: Code Quality and Comments

- Assigning names to all variables, functions, and methods to make the code easier to read, understand, and maintain
- The shorter and clearer your names or commands are, the better off you will be
- The names of the functions should be meaningful and descriptive.
- The function should begin with an action word for what it does
- Write skeleton code
    - By designing the your program architecture first, that forces you to consider the best way to approach the problem



Hubris: Code Quality and Comments

- Was the answer immediately obvious to you before you wrote the code?
  - No? Then comment your solution
  - Yes? Then no comment is required
- Don’t write code when you are tired or in a bad mood
  - If you have poor time management, this becomes all your code
- Don’t write all at once—make developing iterative.
  - Taking breaks lets your mind work out problems subconsciously

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

-- Martin Golding

Case Study: Battlesnake

- Battlesnake is a hackathon that occurs in Victoria every year
  - Last event was March 2, 2019
  - Participants compete against each other by writing their own Snake AIs
  - Game Engine is publically available on GitHub

Rules

- 2019 Game Rules
- TL;DR
  - A square board with 10 food and starting health 100
    - Beginner: 9x9 board
  - Each turn reduces health by 1 and snake dies if health reaches 0
  - Eating food increases health to 100 and increases snake length (at the tail)
  - Death caused by: hitting the wall, hitting another snake*
    - NOTE: There are more rules regarding this, but we'll ignore them for now

Getting started...

- You interact with the game engine (a http server) with HTTP requests
- To start, fork a copy of a starter snake
- I've forked a copy and will update the contents with the in-class example code.
  - Git Repo
  - Code will be posted sometime during the weekend.

Your snake receives JSON game data


```
{
  "game": {
    "id": "game-id-string"
  },
  "turn": 1,
  "board": {
    "height": 11,
    "width": 11,
    "food": [{
      "x": 1,
      "y": 3
    }],
    "snakes": [{
      "id": "snake-id-string",
      "name": "Sneky Snek",
      "health": 100,
      "body": [{
        "x": 1,
        "y": 3
      }]
    }]
  },
  "you": {
    "id": "snake-id-string",
    "name": "Sneky Snek",
    "health": 100,
    "body": [{
      "x": 1,
      "y": 3
    }]
  }
}
```