Essentials in CS - Part II

15th October 2015
@ 1630-1830
Simone I. Conte (sic2)
http://sic2.host.cs.st-andrews.ac.uk/

Thanks to:
Shyam Reyal, Graham Kirby, Steve Linton, Adrian O'Lenskie, Neil Moore, Peter Brown, Mark-Jan Nederhof, Alex Voss, Martin McCaffery, Shyam Reyal, Ian Paterson, Nick Goodall, Richie McMahon, Ishbel Duncan, Jan de Muijnck-Hughes, Lee Huang, Andrew Matheson, Daniel Patterson, Chris Jefferson

all other people I have met over the years

and Stackoverflow

Credits

Clean Code - Available in library (x4)

@@@

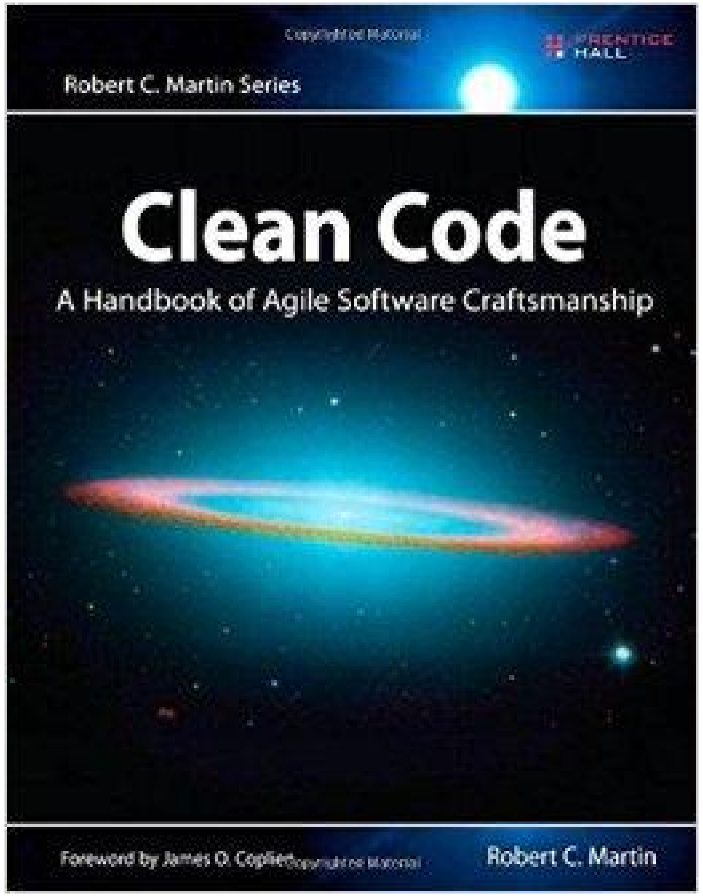
CS5030/1 by Alex Voss

https://studres.cs.st-andrews.ac.uk/CS5031/

@@@

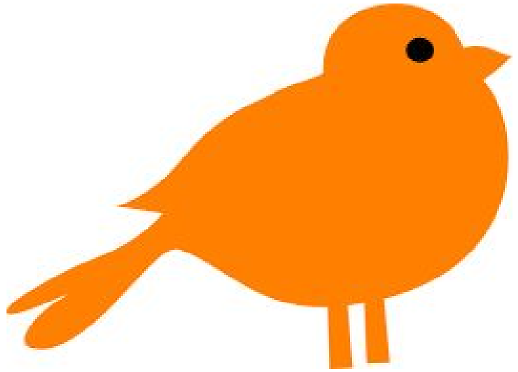
Experts' opinions

https://sic2.host.cs.st-andrews.ac.uk/aiq.html



Outline

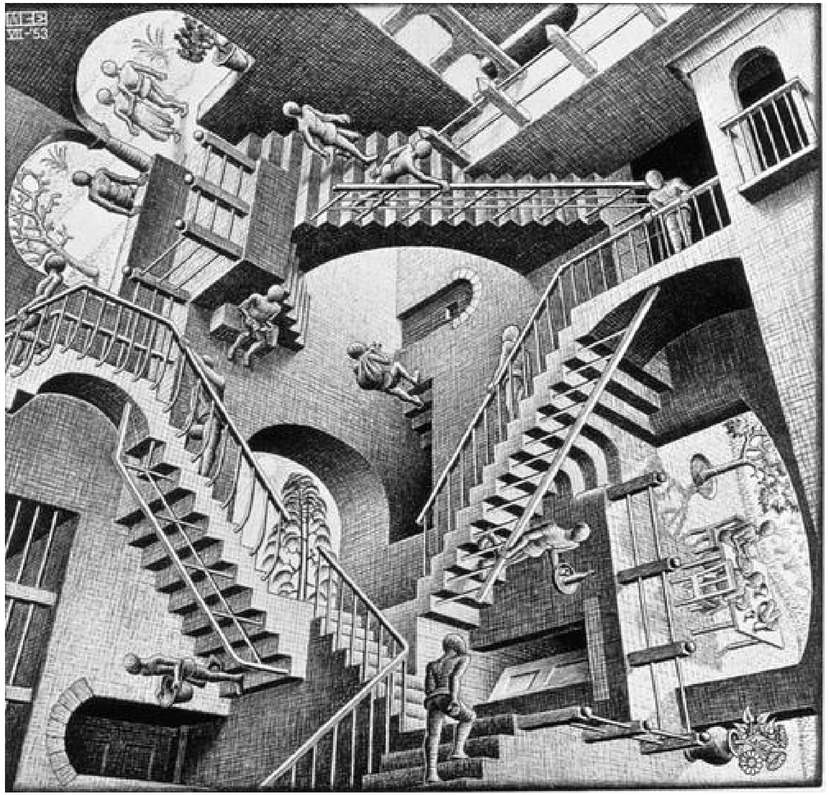
- Aims
- Marking scale
- Becoming a better
  - Programmer
  - Computer Scientist
- Good and Bad code
- Reproducibility and Testing



The 20 marking scale

The 20 marking scale

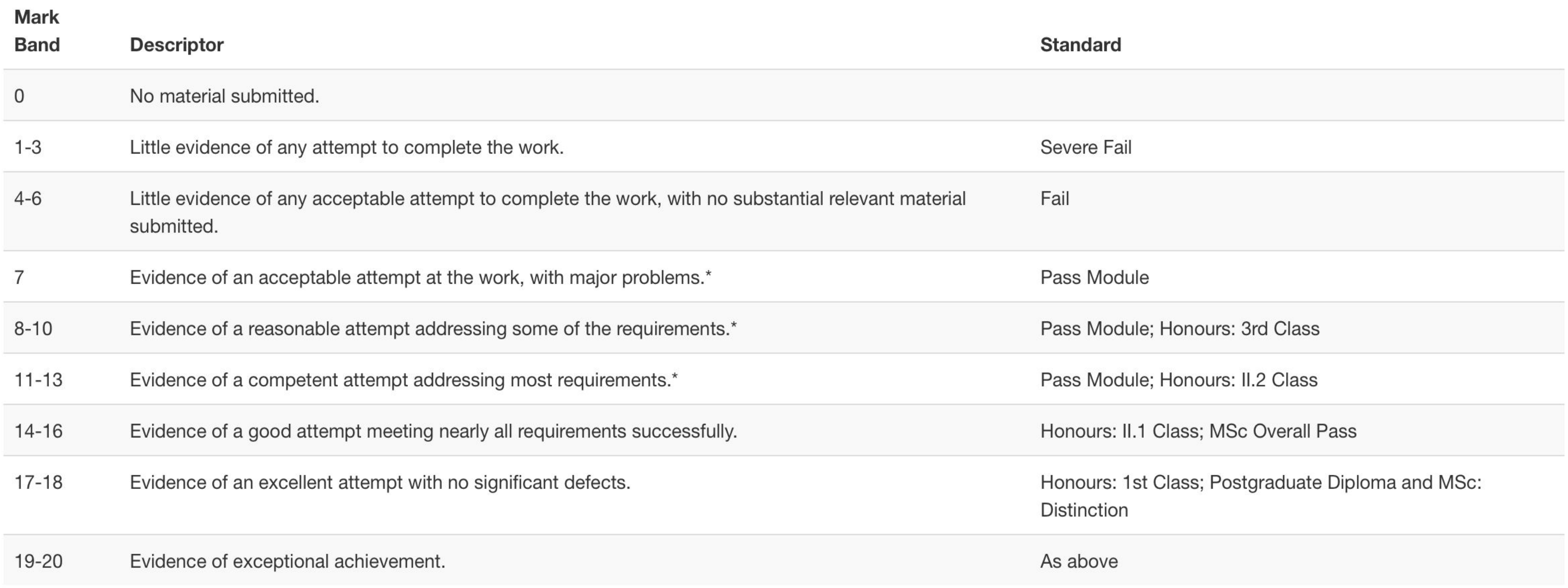
The 20 marking scale



***Relativity* (M. C. Escher)**

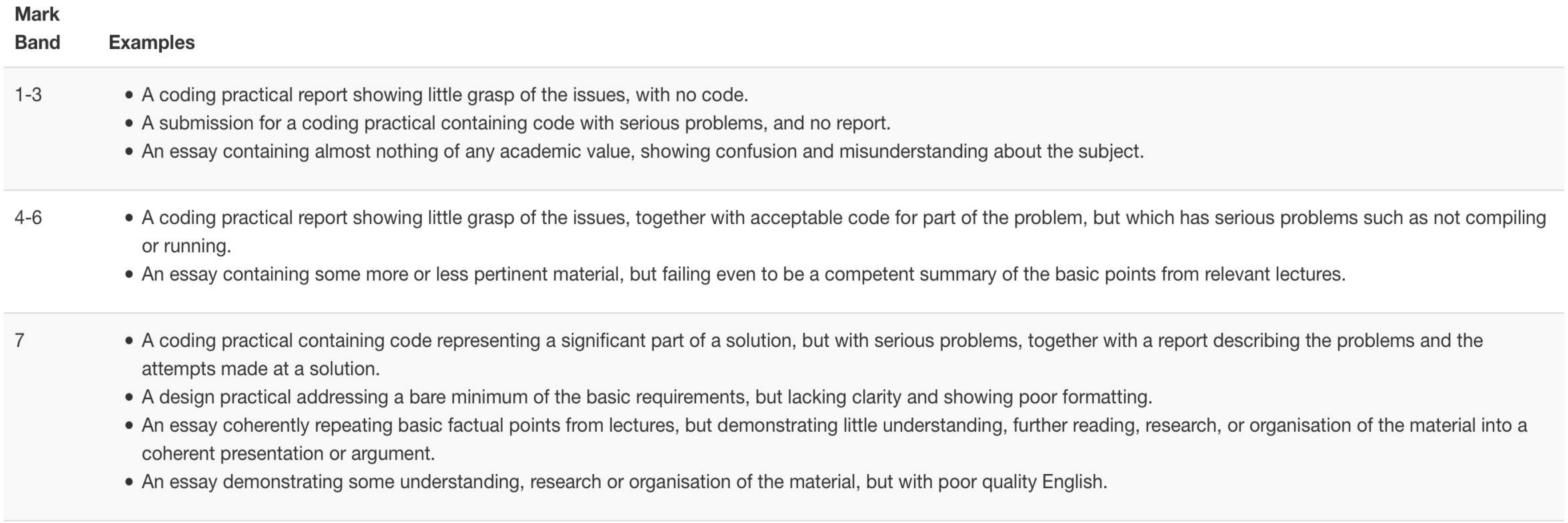
1953

Mark Descriptors (General)



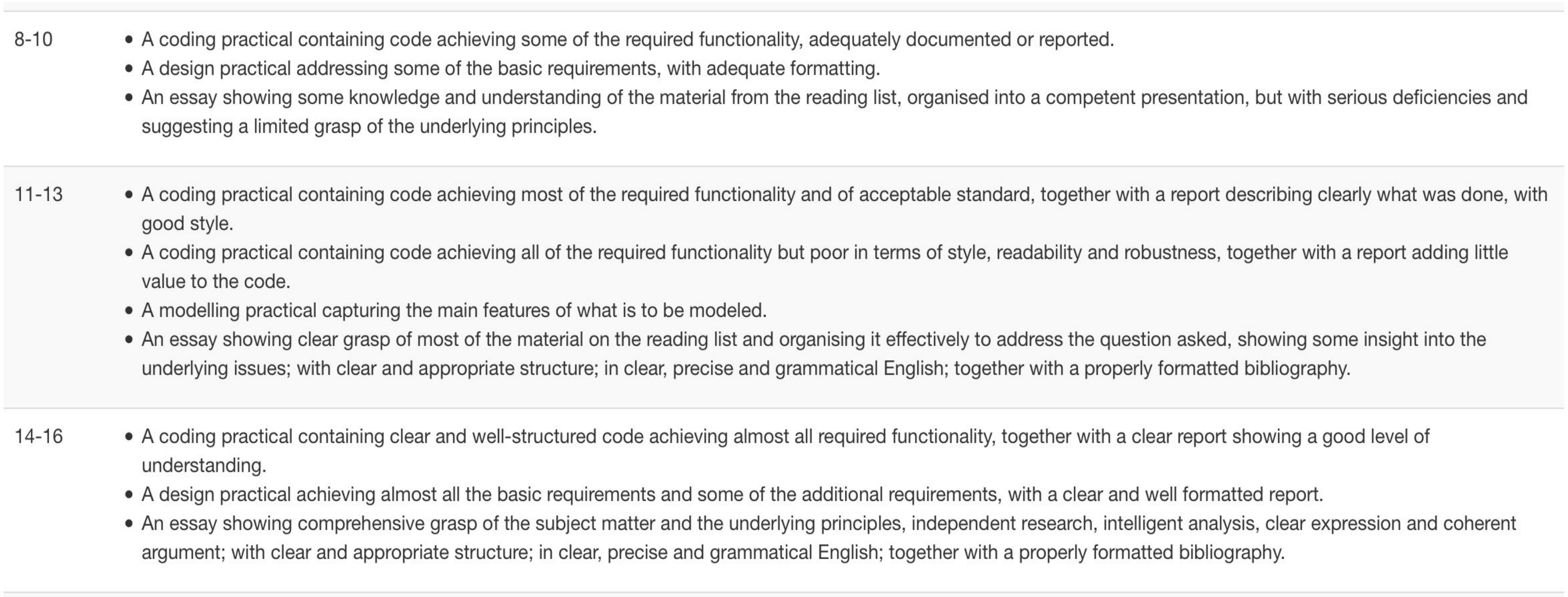
Mark Band	Descriptor	Standard
0	No material submitted.	
1-3	Little evidence of any attempt to complete the work.	Severe Fail
4-6	Little evidence of any acceptable attempt to complete the work, with no substantial relevant material submitted.	Fail
7	Evidence of an acceptable attempt at the work, with major problems.*	Pass Module
8-10	Evidence of a reasonable attempt addressing some of the requirements.*	Pass Module; Honours: 3rd Class
11-13	Evidence of a competent attempt addressing most requirements.*	Pass Module; Honours: II.2 Class
14-16	Evidence of a good attempt meeting nearly all requirements successfully.	Honours: II.1 Class; MSc Overall Pass
17-18	Evidence of an excellent attempt with no significant defects.	Honours: 1st Class; Postgraduate Diploma and MSc: Distinction
19-20	Evidence of exceptional achievement.	As above

Mark Descriptors (CW I)



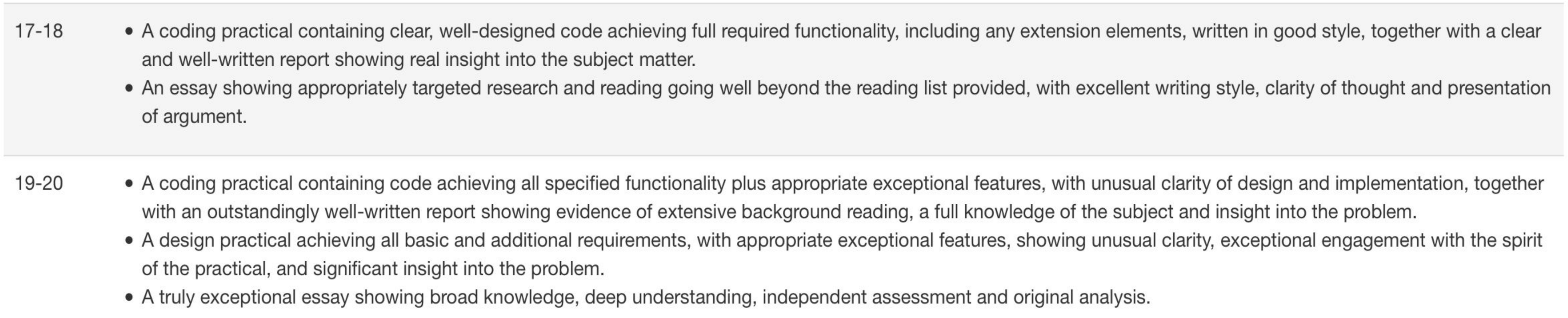
Mark Band	Examples
1-3	• A coding practical report showing little grasp of the issues, with no code. • A submission for a coding practical containing code with serious problems, and no report. • An essay containing almost nothing of any academic value, showing confusion and misunderstanding about the subject.
4-6	• A coding practical report showing little grasp of the issues, together with acceptable code for part of the problem, but which has serious problems such as not compiling or running. • An essay containing some more or less pertinent material, but failing even to be a competent summary of the basic points from relevant lectures.
7	• A coding practical containing code representing a significant part of a solution, but with serious problems, together with a report describing the problems and the attempts made at a solution. • A design practical addressing a bare minimum of the basic requirements, but lacking clarity and showing poor formatting. • An essay coherently repeating basic factual points from lectures, but demonstrating little understanding, further reading, research, or organisation of the material into a coherent presentation or argument. • An essay demonstrating some understanding, research or organisation of the material, but with poor quality English.

Mark Descriptors (CW II)



8-10	• A coding practical containing code achieving some of the required functionality, adequately documented or reported. • A design practical addressing some of the basic requirements, with adequate formatting. • An essay showing some knowledge and understanding of the material from the reading list, organised into a competent presentation, but with serious deficiencies and suggesting a limited grasp of the underlying principles.
11-13	• A coding practical containing code achieving most of the required functionality and of acceptable standard, together with a report describing clearly what was done, with good style. • A coding practical containing code achieving all of the required functionality but poor in terms of style, readability and robustness, together with a report adding little value to the code. • A modelling practical capturing the main features of what is to be modeled. • An essay showing clear grasp of most of the material on the reading list and organising it effectively to address the question asked, showing some insight into the underlying issues; with clear and appropriate structure; in clear, precise and grammatical English; together with a properly formatted bibliography.
14-16	• A coding practical containing clear and well-structured code achieving almost all required functionality, together with a clear report showing a good level of understanding. • A design practical achieving almost all the basic requirements and some of the additional requirements, with a clear and well formatted report. • An essay showing comprehensive grasp of the subject matter and the underlying principles, independent research, intelligent analysis, clear expression and coherent argument; with clear and appropriate structure; in clear, precise and grammatical English; together with a properly formatted bibliography.

Mark Descriptors (CW III)



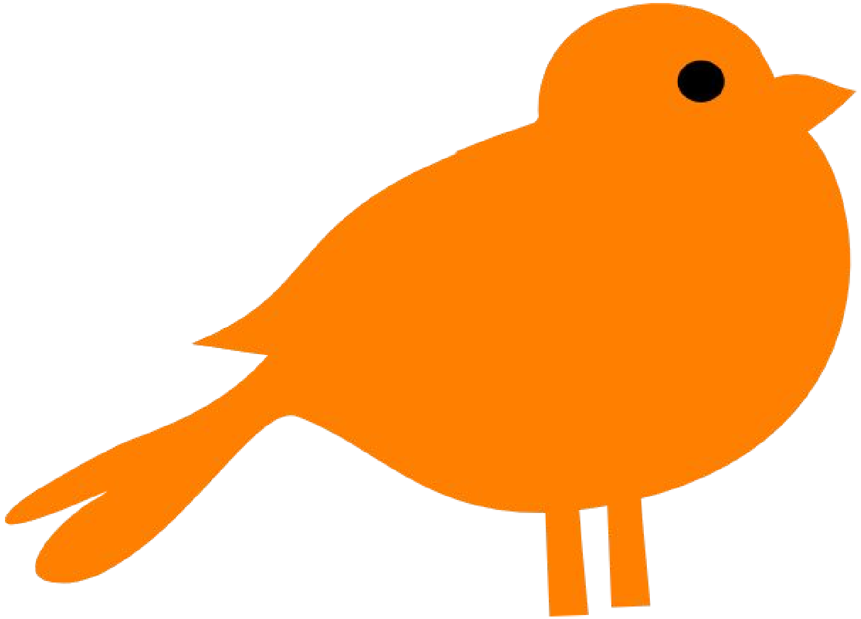
17-18	• A coding practical containing clear, well-designed code achieving full required functionality, including any extension elements, written in good style, together with a clear and well-written report showing real insight into the subject matter. • An essay showing appropriately targeted research and reading going well beyond the reading list provided, with excellent writing style, clarity of thought and presentation of argument.
19-20	• A coding practical containing code achieving all specified functionality plus appropriate exceptional features, with unusual clarity of design and implementation, together with an outstandingly well-written report showing evidence of extensive background reading, a full knowledge of the subject and insight into the problem. • A design practical achieving all basic and additional requirements, with appropriate exceptional features, showing unusual clarity, exceptional engagement with the spirit of the practical, and significant insight into the problem. • A truly exceptional essay showing broad knowledge, deep understanding, independent assessment and original analysis.

More here → https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html

The 20 grading scale

- We do not mark down
- We do not mark up
- We use mark descriptors and our own judgment

- This workshop is about *What we think good code looks like* and there might not be a 1:1 correspondence with the mark you will get
- Work hard



Who are you?

- Students
- Interns
- Future engineers
- Future scientists
- Computer scientists
- Future St Andrews Graduates
- Innovators
- Lots more

Improve your coursework

Aims of workshop

- Not necessarily better grades, that is up to the markers
- Cannot cover everything in 2 hours (150+ slides)
  - Will be superficial over some bits.
- Understand the basic ideas behind good code
- Improve yourself, know what to expect when working for industry

Some feedback - Steve Linton

“Overall, you're good at getting the program to do what you want it to, but that isn't the point (at least in the upper grade ranges where you should be playing). **You should produce simple, clean code, relevant comments, matching design and you should, always, always, stick to the documented behaviour of interfaces.**”

Some feedback - Mark-Jan Nederhof

* Foremost, carefully read the specification of the assignment.
Creating software **to match specs is an important skill** in computer science.

* Complexity is not something to strive for in programming.
If you can **solve a task in a simple way**, it is almost always better than solving it in a complex way.

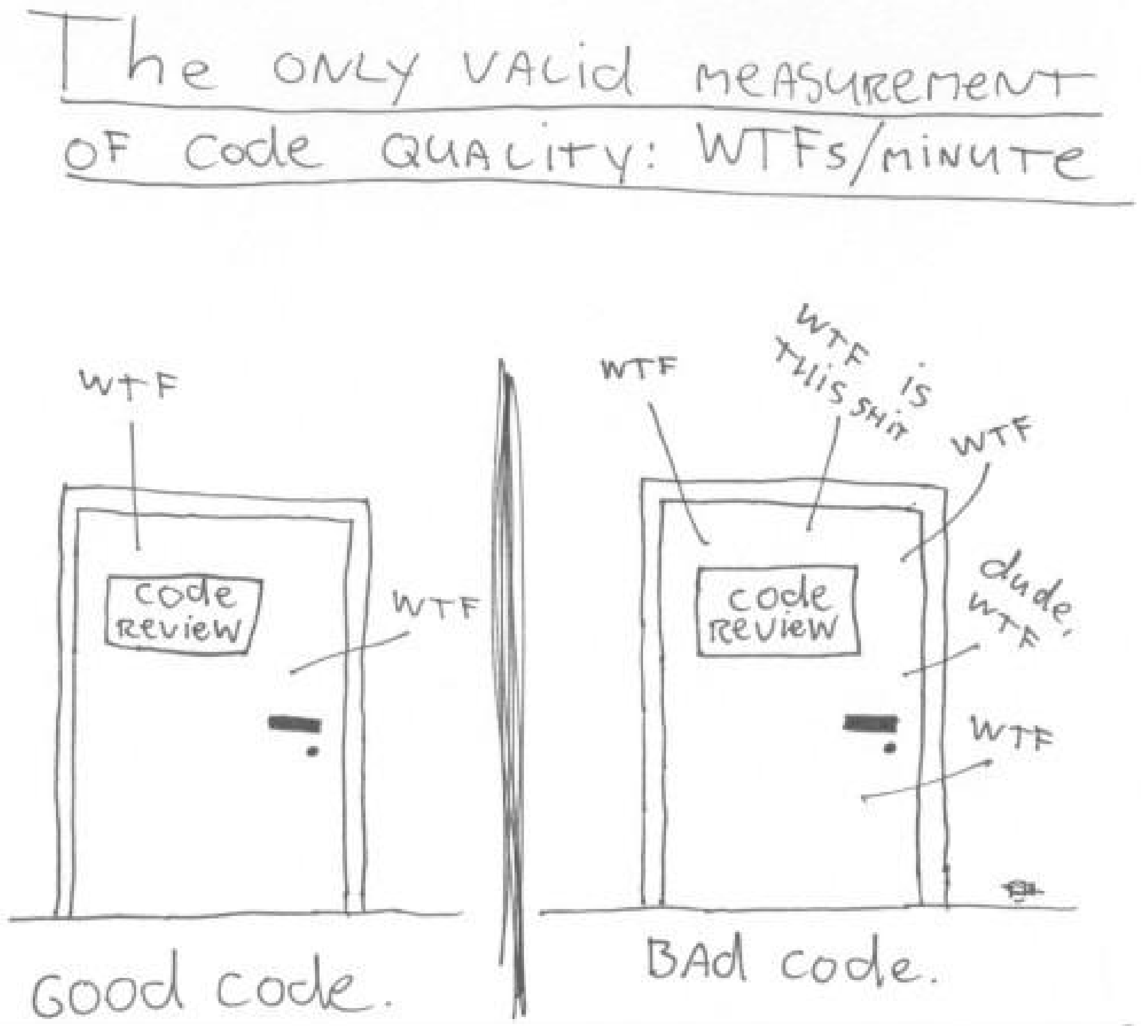
Why good code matters

- Your marker
- Yourself
- You are not alone
- Mantaining code
    - agility - ability to adapt quickly to needs
    - billions of lines of code (see Google, FB, etc)

Why good code matters

4 - **Code is like a poem;** it's not just something we write to reach some practical result. Sometimes people that are far from the Redis philosophy suggest using other code written by other authors (frequently in other languages) in order to implement something Redis currently lacks. But to us this is like if Shakespeare decided to end Enrico IV using the Paradiso from the Divina Commedia. **Is using any external code a bad idea? Not at all.** Like in "One Thousand and One Nights" smaller self contained stories are embedded in a bigger story, we'll be happy to use beautiful self contained libraries when needed. At the same time, when writing the Redis story we're trying to write smaller stories that will fit in to other code.

[Antirez, Redis Manifesto]



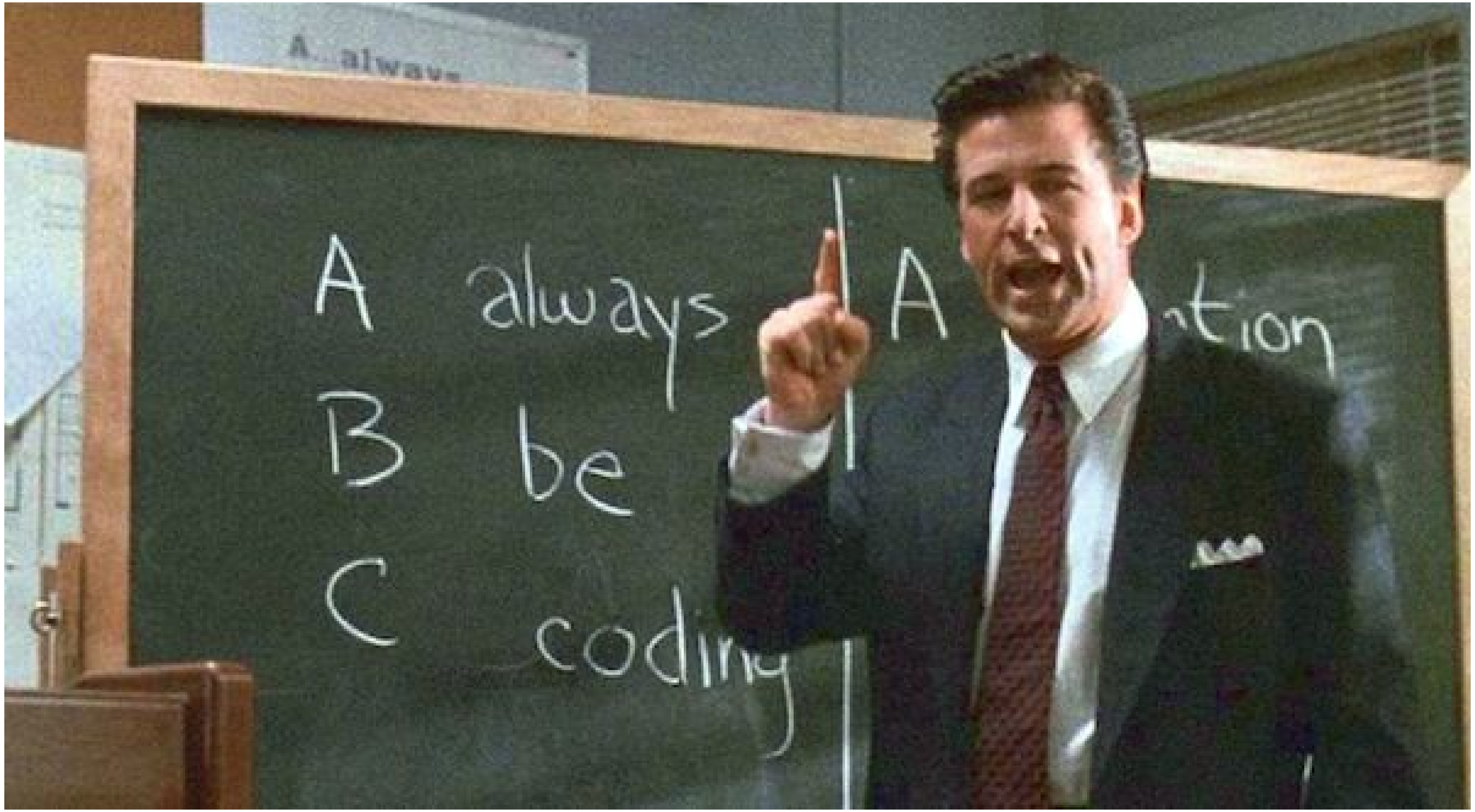
http://www.osnews.com/story/19266/WTFs_m

What is Good Software?

- The software should deliver the required functionality and should be:
  - Delivered on time and on budget
  - Maintainable
  - Dependable
  - Secure
  - Efficient
  - Usable
  - Accepted by its users

- Software engineering is concerned with producing the best possible software but this involves trade-offs.

ABC

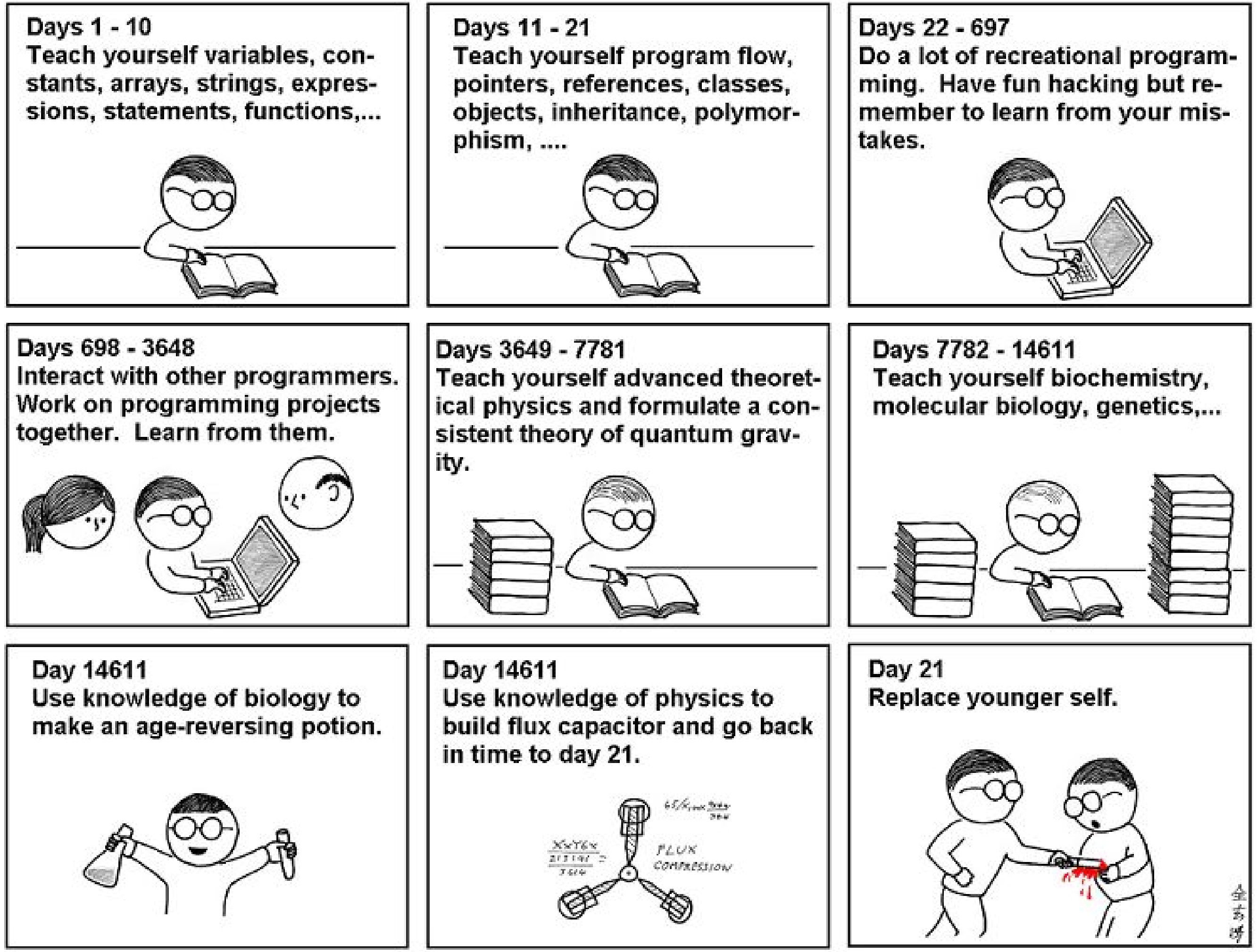


Credits to https://medium.com/@davidbyttow/abc-always-be-coding-d5f8051afce2

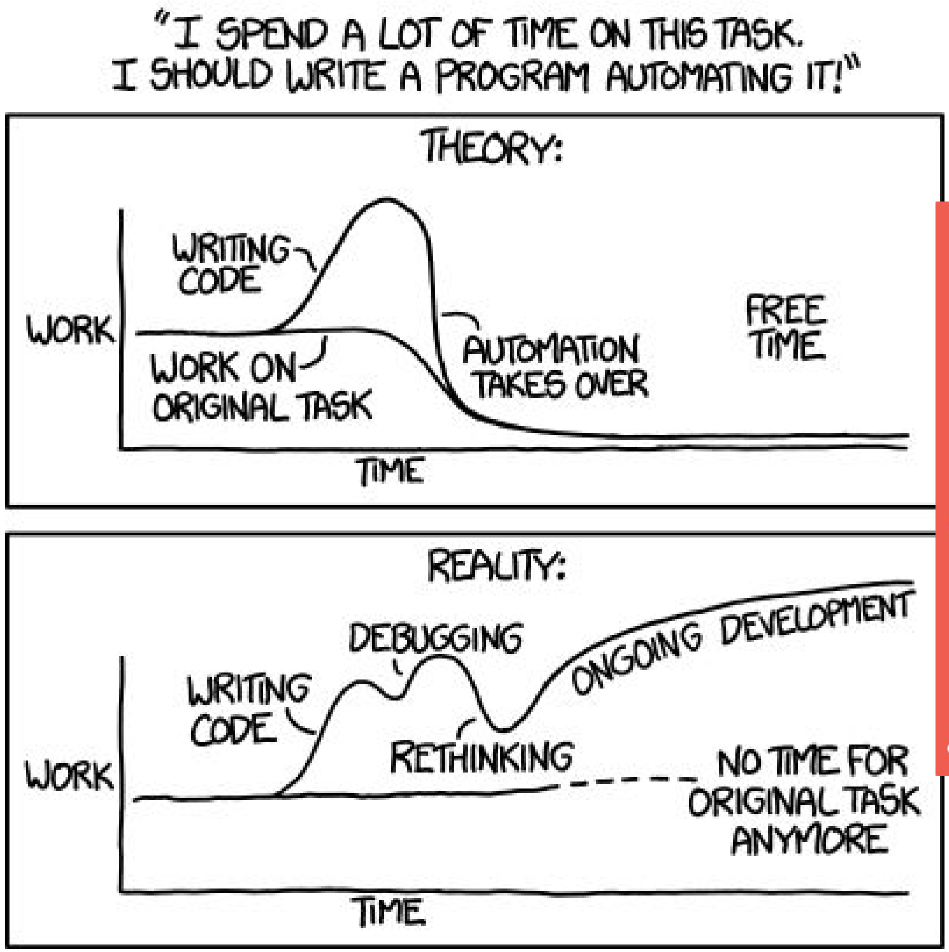
I want to be a better Computer Scientist

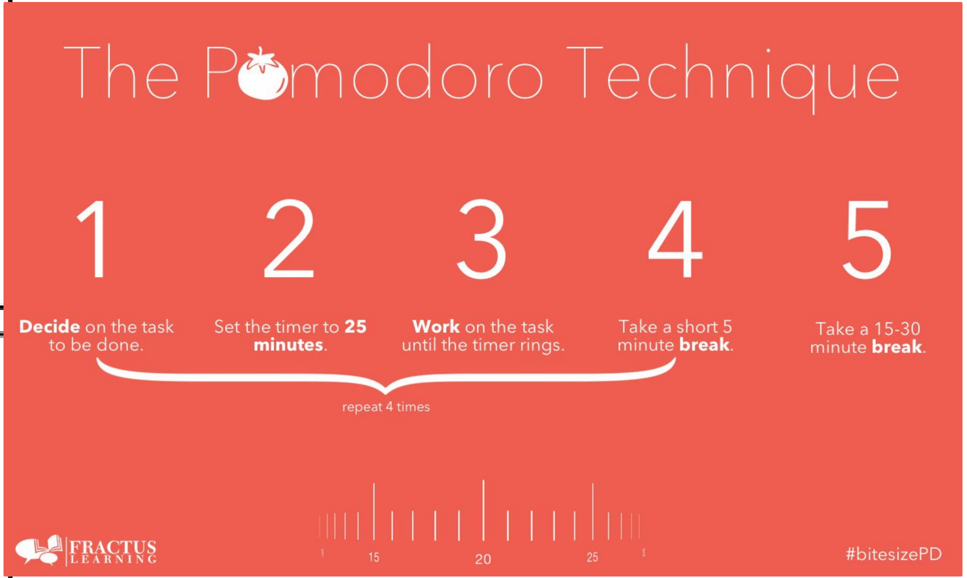
- Be passionate, stubborn, humble, and read-read-read!
- Also, think out of the box :)
- 10k-hrs rule - http://norvig.com/21-days.html

GALILEO
GALILEI


Days 1 - 10
Teach yourself variables, constants, arrays, strings, expressions, statements, functions,...
Days 11 - 21
Teach yourself program flow, pointers, references, classes, objects, inheritance, polymorphism, ...
Days 22 - 697
Do a lot of recreational programming. Have fun hacking but remember to learn from your mistakes.
Days 698 - 3648
Interact with other programmers. Work on programming projects together. Learn from them.
Days 3649 - 7781
Teach yourself advanced theoretical physics and formulate a consistent theory of quantum gravity.
Days 7782 - 14611
Teach yourself biochemistry, molecular biology, genetics,...
Day 14611
Use knowledge of biology to make an age-reversing potion.
Day 14611
Use knowledge of physics to build flux capacitor and go back in time to day 21.
FLUX COMPRESSION
Day 21
Replace younger self.

http://mishadoff.com/blog/programming-digest-2/




The Pomodoro Technique
1
Decide on the task to be done.
2
Set the timer to 25 minutes.
3
Work on the task until the timer rings.
4
Take a short 5 minute break.
5
Take a 15-30 minute break.
repeat 4 times
15
20
25
FRACTUS LEARNING
#bitesizePD

https://xkcd.com/1319/

http://pomodorotechnique.com/

Time management - Summary

Different philosophies, different approaches, different techniques.

All have:

- Have breaks while working
- Have a life too
- Do not be afraid of getting it wrong

Later Equals Never
http://robertgreiner.com/2011/07/later-equals-never/

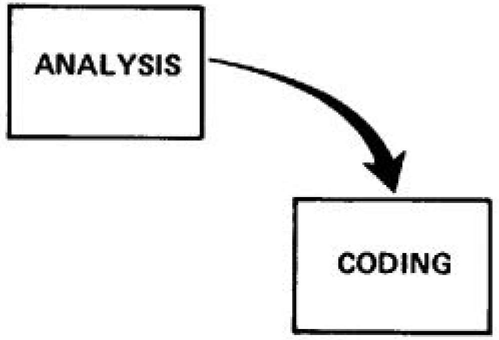
Process Models

5 Minutes in Software Engineering techniques to go from requirements to code to product.

- Minimal Model (**X**)
- Waterfall (w/wo feedback) (**X**)
- Scrum (3rd year project) (**V**)
- Pair Programming (1st, 2nd year group projects) (**?**)
- Etc.

Not all of these are good

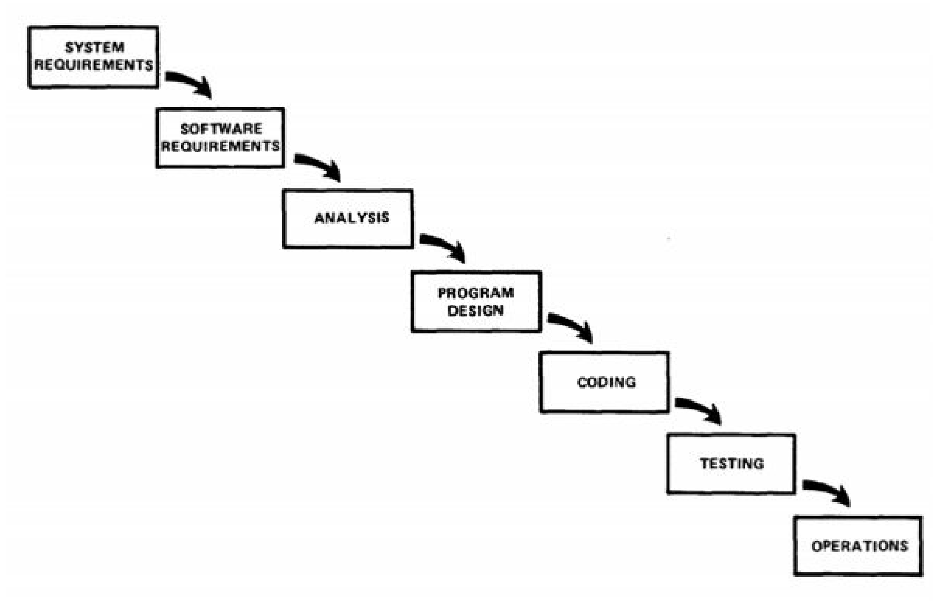
Minimal Software Process Model

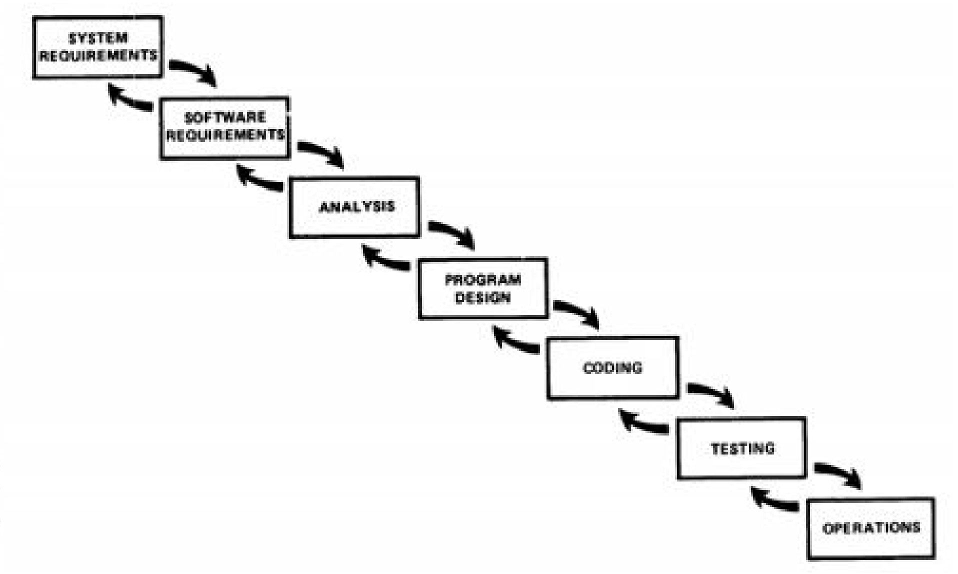


Winston W. Royce,

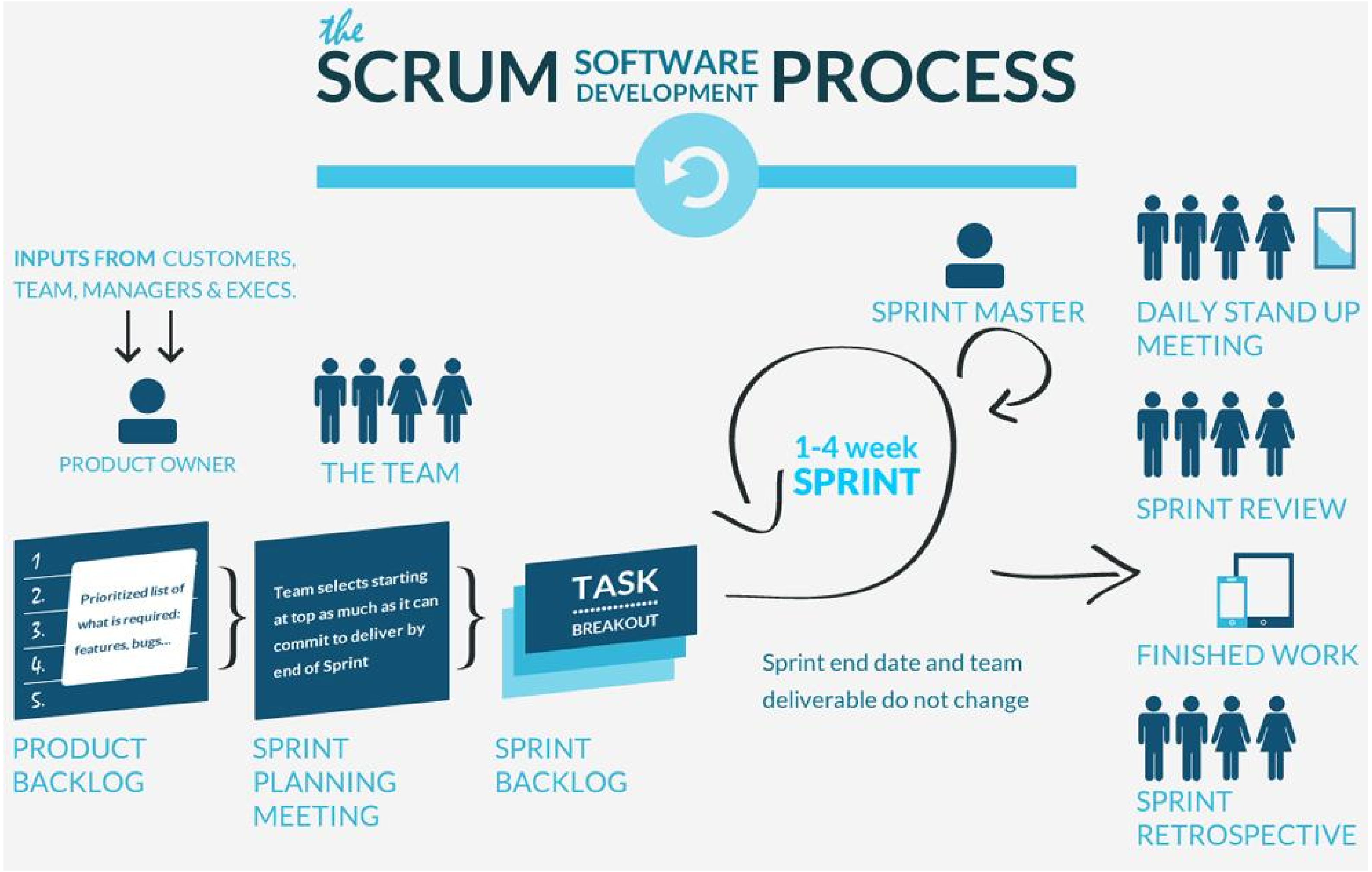
Managing the Development of Large Software Systems

The “Waterfall” Model - w/wo feedback



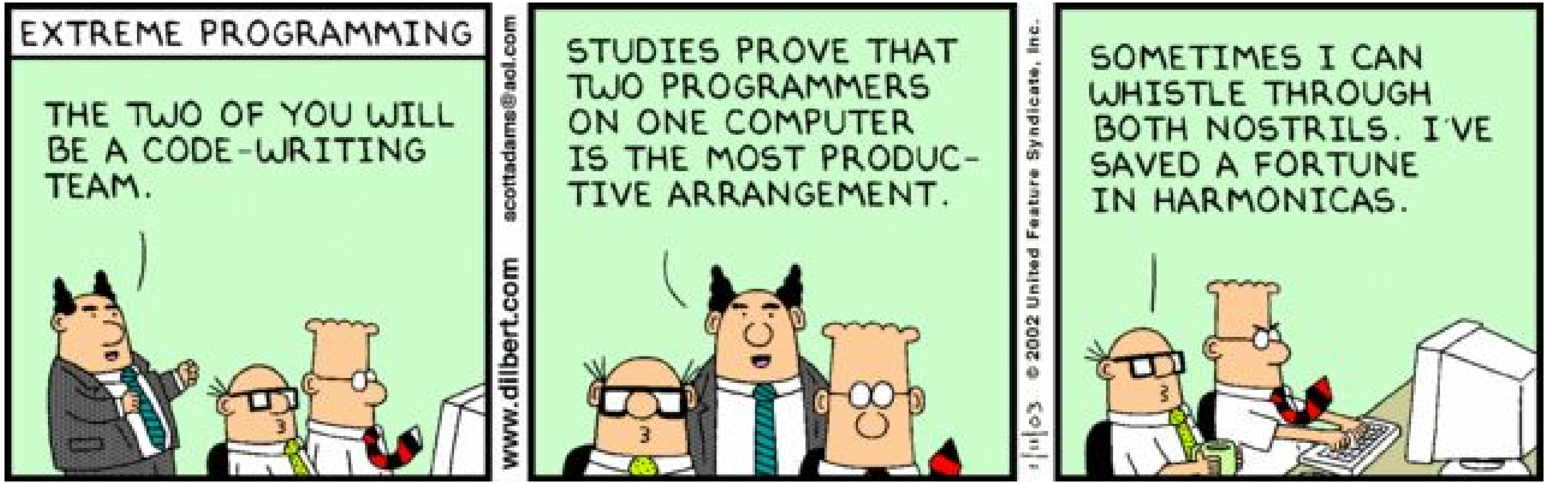


Paper http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf


the SCRUM SOFTWARE DEVELOPMENT PROCESS
INPUTS FROM CUSTOMERS, TEAM, MANAGERS & EXECS.
PRODUCT OWNER
THE TEAM
SPRINT MASTER
DAILY STAND UP MEETING
1-4 week SPRINT
SPRINT REVIEW
1
2.
3.
4.
5.
Prioritized list of what is required: features, bugs...
Team selects starting at top as much as it can commit to deliver by end of Sprint
TASK BREAKOUT
Sprint end date and team deliverable do not change
FINISHED WORK
PRODUCT BACKLOG
SPRINT PLANNING MEETING
SPRINT BACKLOG
SPRINT RETROSPECTIVE

Source https://www.ics.ie/news/view/1653

Pair Programming

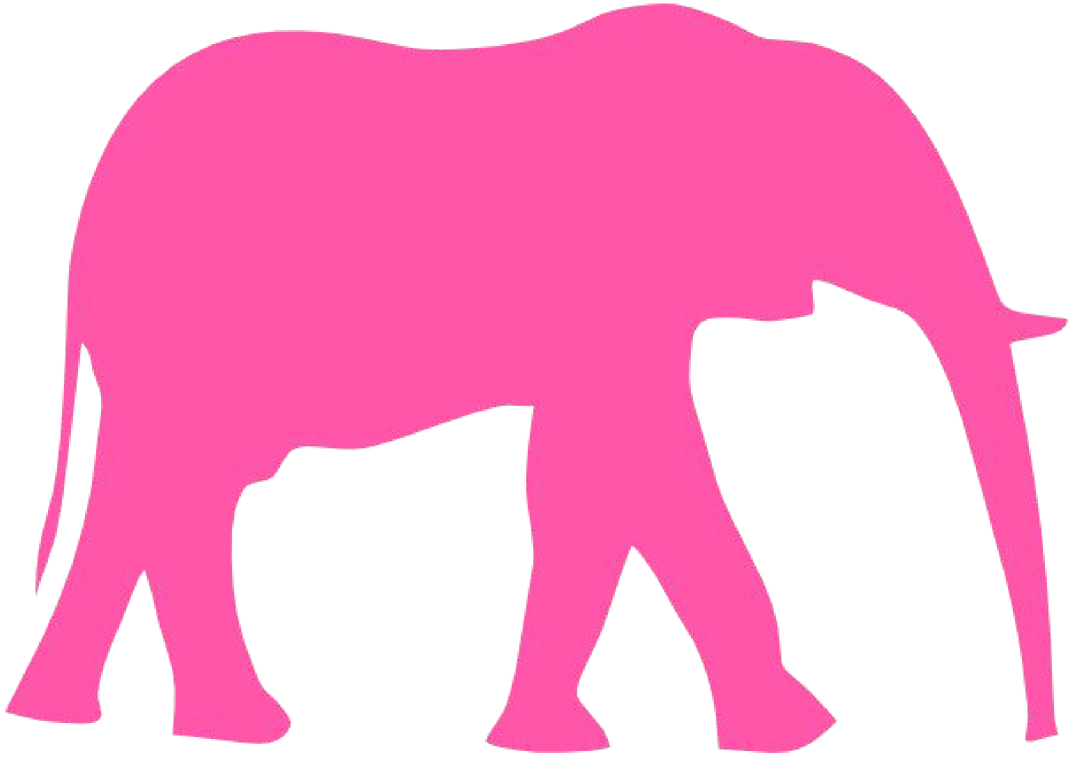

EXTREME PROGRAMMING
THE TWO OF YOU WILL BE A CODE-WRITING TEAM.
STUDIES PROVE THAT TWO PROGRAMMERS ON ONE COMPUTER IS THE MOST PRODUCTIVE ARRANGEMENT.
SOMETIMES I CAN WHISTLE THROUGH BOTH NOSTRILS. I'VE SAVED A FORTUNE IN HARMONICAS.
www.dilbert.com scottadams@aol.com
© 2002 United Feature Syndicate, Inc.

Source: http://techiejs.com/Blog/Post/Working-Agile-like-with-SCRUM-and-Extreme-Programming

Summary about SE Processes

- Do you need to know all this stuff to be successful in my coursework?
  - **NO**

- Be agile
- Listen to your client
- Use/Test your code
- Do not be afraid of changing your design/code
- Tradeoff between perfection and productivity



5-10 minutes of Common Mistakes in Java/C

Common Mistakes (Java) I

```
public class StaticDemo
{
       public String my_member_variable = "somedata";

       public static void main (String args[])
       {
       // Access a non-static member from static method
       System.out.println ("This generates a compiler error" +
           my_member_variable );
       }
}
```

Common Mistakes (Java) II

```
public class MyWindowListener extends WindowAdapter {
    // This should be WindowClosed
    public void WindowClose(WindowEvent e) {
        // Exit when user closes window
        System.exit(0);
    }
});
```

Common Mistakes (Java) IV

```
{
Connection dbConnection = initConnect();
PreparedStatement query = c.preparedStatement(...);
…
ResultSet resultSet = query.execute();
…
???

}
```

Common Mistakes (Java) IV

```
{
Connection dbConnection = initConnect();
PreparedStatement query = c.preparedStatement(...);
…
ResultSet resultSet = query.execute();
…
resultSet.close();
query.close();
dbConnection.close();

}
```

Common Mistakes (Java 7+) IV - AutoClosable

```
try (
    Connection dbConnection = initConnect();
    PreparedStatement query = c.preparedStatement(...);
) {
    // code
    resultSet = query.execute();
    // code
} catch (...) {}
finally {
    if (resultSet != null) resultSet.close();

}
```

Common Mistakes (Java) V

= instead of ==

== instead of equals

by value (primitive) vs by reference (objects)

Common Mistakes (C/Java) I

```
int x = 2;
switch(x) {
case 2:
  printf("Two\n");
case 3:
  printf("Three\n");
}
```

Common Mistakes (C) II

```
int a[10]; // from 0 to 9, not to 10 (same in Java)

char ch = 'A';      /* correct */
char ch = "A";      /* error   */

const char * st = "A";     /* correct */
const char * st = 'A';     /* error   */
```

Common Mistakes (C) III

```
char st1[] = "abc";
char st2[] = "abc";
if ( st1 == st2 )
  printf("Yes");
else
  printf("No");
```

Common Mistakes (C) III

```
if ( strcmp(st1,st2) == 0 )
  printf("Yes");
else
  printf("No");
```

What the experts say

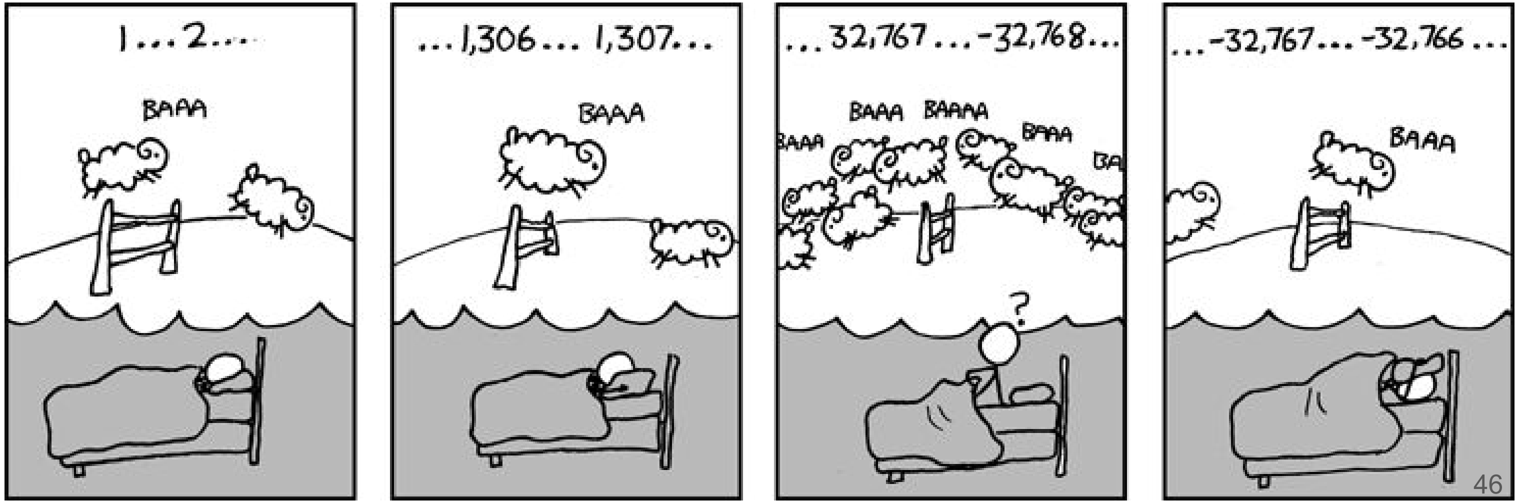
“**Don't judge people based on code.** You don't know under what conditions the code was written, there was likely an idiot manager forcing the developer to compromise.”
(A. O’Lenskie - Adobe)

“I think choosing technology/platforms wisely is quite a big one. Also timing is important.”
(N. Moore - Adobe)

“I think for early students, **the most important thing is to try to make their program compile and run** (so they can test it) as early as possible, **then keep iterating by adding small features and retesting.**”
(C. Jefferson - St Andrews)

Know the basics

- Primitives
- Strings
- OOP
- Streams
- Data Structures
- Exceptions
- Etc



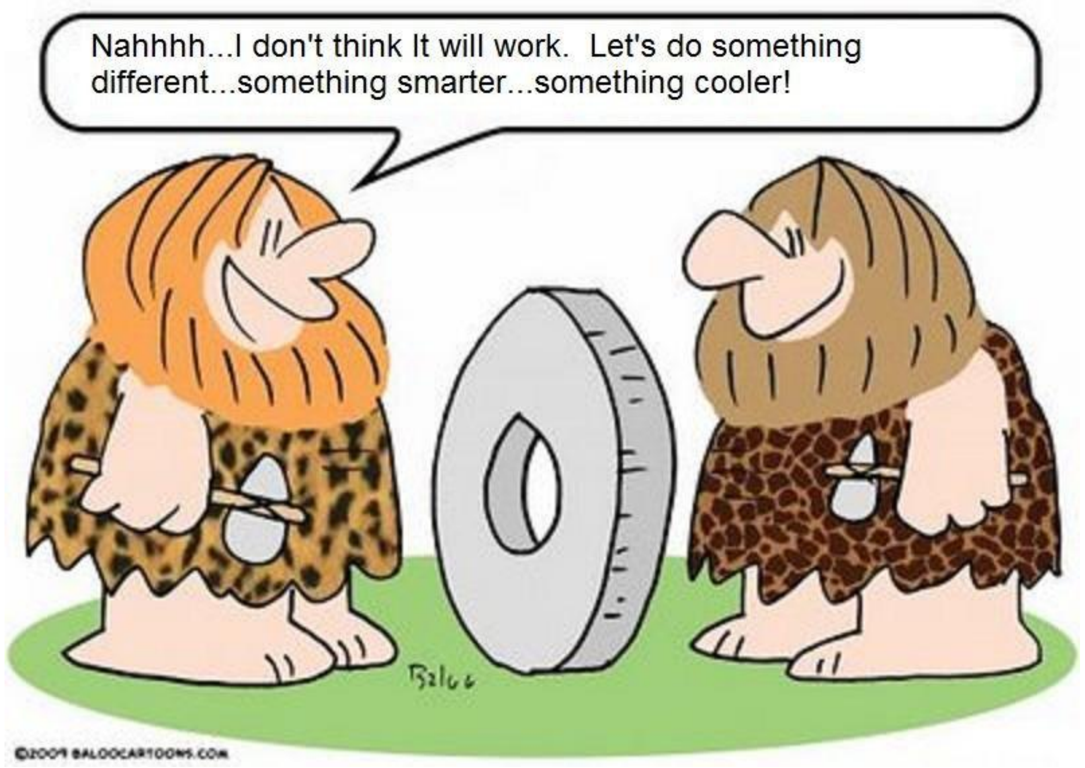
https://xkcd.com/571/

The wheel problem

"I reinvented the wheel last week. I sat down and deliberately coded something that I knew already existed, and had probably also been done by many many other people. In conventional programming terms, ***I wasted my time. But it was worthwhile****, and what's more I would recommend almost any serious programmer do precisely the same thing."*

James Hart

Who's James Hart?
Just another programmer.


Nahhhh...I don't think It will work. Let's do something different...something smarter...something cooler!
Baloo
©2009 BALOOCARTOONS.COM

https://www.linkedin.com/pulse/stop-reinventing-wheel-philip-holt

Verification & Validation

- Verification:
  - are we building the system right (according to its specification)?

- Validation:
  - are we building the right system (the one the users require)?

Break your own code!

Be active and become the hacker of your own product.

There is not a guide book or some other type of Holy Grail.
The more you program, the more you challenge yourself, the better you get to break code.

Note: breaking others’ code is illegal

TopCoder challenge’s steps:

- Solve
- Break
- Test

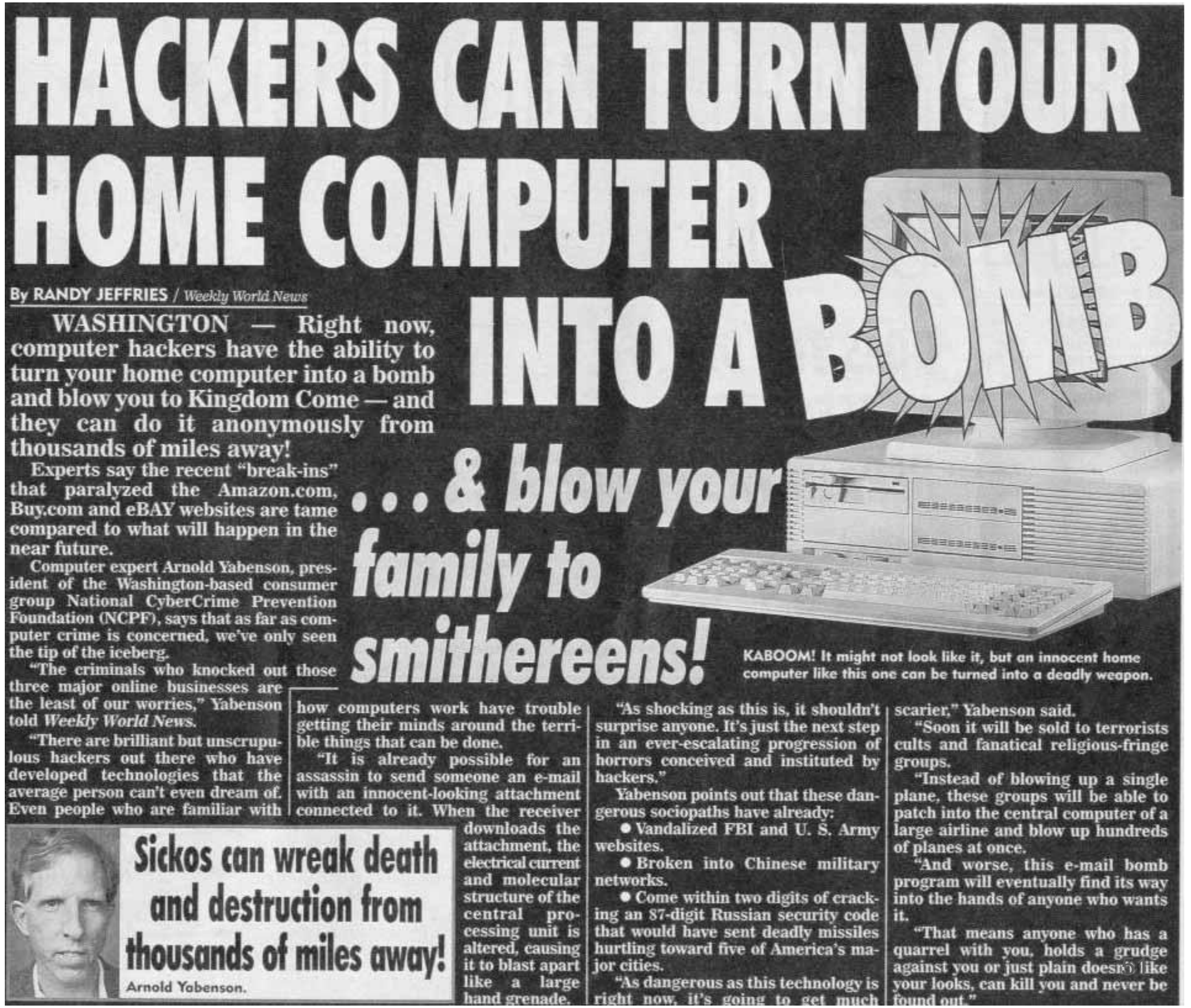
https://www.topcoder.com/

I AM A BOMB!
Michael Jackson
System Devel
STRATUS
Introduction to Java Programming
Software Engineer

Know how to use internet!

This is what I got when looking for: “Bomb in a shelf security example”

https://www.reddit.com/r/pics/comments/klr85/hackers_can_turn_your_home_computer_into_a_bomb/



HACKERS CAN TURN YOUR HOME COMPUTER INTO A BOMB

. . . & blow your family to smithereens!

By RANDY JEFFRIES / *Weekly World News*

WASHINGTON — Right now, computer hackers have the ability to turn your home computer into a bomb and blow you to Kingdom Come — and they can do it anonymously from thousands of miles away!

Experts say the recent “break-ins” that paralyzed the Amazon.com, Buy.com and eBAY websites are tame compared to what will happen in the near future.

Computer expert Arnold Yabenson, president of the Washington-based consumer group National CyberCrime Prevention Foundation (NCPF), says that as far as computer crime is concerned, we've only seen the tip of the iceberg.

“The criminals who knocked out those three major online businesses are the least of our worries,” Yabenson told *Weekly World News*.

“There are brilliant but unscrupulous hackers out there who have developed technologies that the average person can't even dream of. Even people who are familiar with how computers work have trouble getting their minds around the terrible things that can be done.

“It is already possible for an assassin to send someone an e-mail with an innocent-looking attachment connected to it. When the receiver downloads the attachment, the electrical current and molecular structure of the central processing unit is altered, causing it to blast apart like a large hand grenade.

“As shocking as this is, it shouldn't surprise anyone. It's just the next step in an ever-escalating progression of horrors conceived and instituted by hackers.”

Yabenson points out that these dangerous sociopaths have already:

- Vandalized FBI and U. S. Army websites.
- Broken into Chinese military networks.
- Come within two digits of cracking an 87-digit Russian security code that would have sent deadly missiles hurtling toward five of America's major cities.

“As dangerous as this technology is right now, it's going to get much scarier,” Yabenson said.

“Soon it will be sold to terrorists cults and fanatical religious-fringe groups.

“Instead of blowing up a single plane, these groups will be able to patch into the central computer of a large airline and blow up hundreds of planes at once.

“And worse, this e-mail bomb program will eventually find its way into the hands of anyone who wants it.

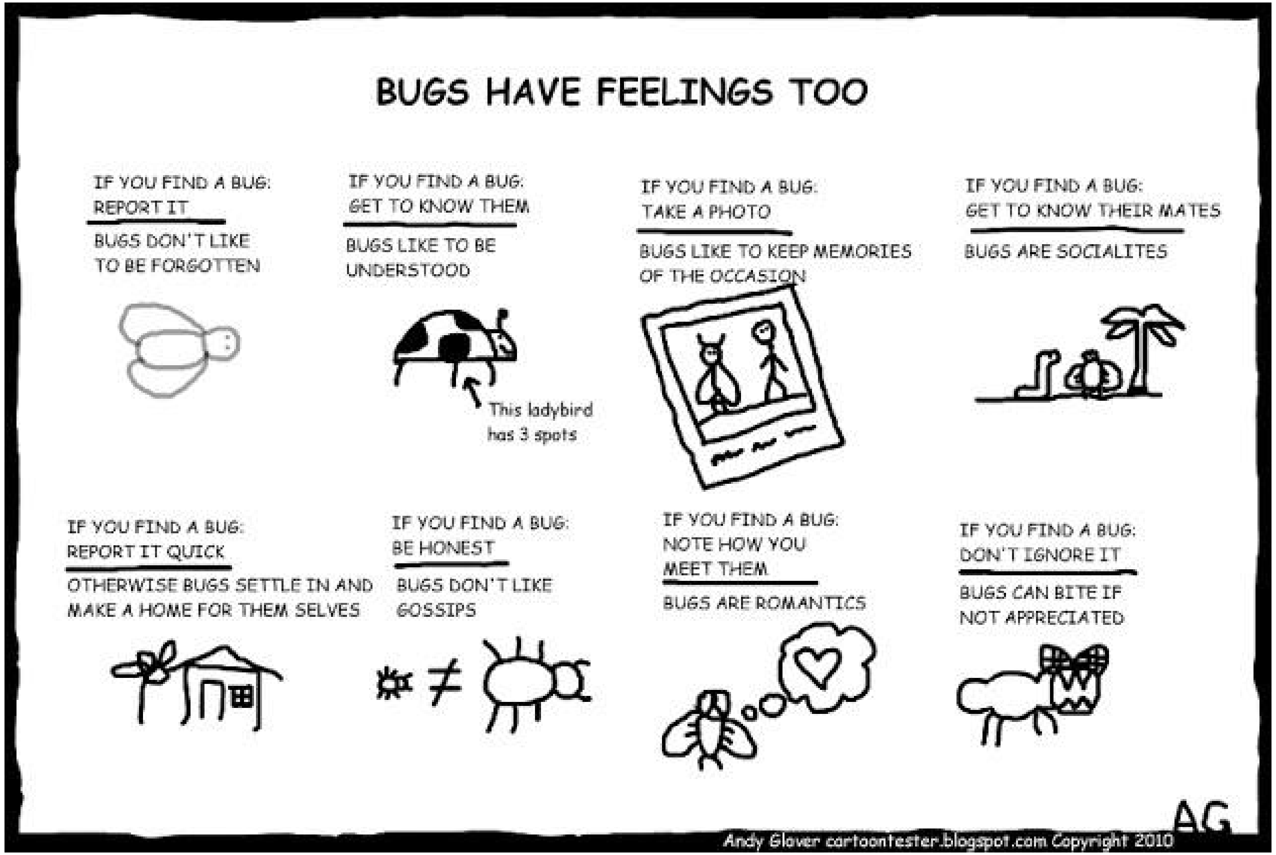
“That means anyone who has a quarrel with you, holds a grudge against you or just plain doesn't like your looks, can kill you and never be found out.”

KABOOM! It might not look like it, but an innocent home computer like this one can be turned into a deadly weapon.

Sickos can wreak death and destruction from thousands of miles away!

Arnold Yabenson.

Software Testing


BUGS HAVE FEELINGS TOO
IF YOU FIND A BUG: REPORT IT
BUGS DON'T LIKE TO BE FORGOTTEN
IF YOU FIND A BUG: GET TO KNOW THEM
BUGS LIKE TO BE UNDERSTOOD
This ladybird has 3 spots
IF YOU FIND A BUG: TAKE A PHOTO
BUGS LIKE TO KEEP MEMORIES OF THE OCCASION
IF YOU FIND A BUG: GET TO KNOW THEIR MATES
BUGS ARE SOCIALITES
IF YOU FIND A BUG: REPORT IT QUICK
OTHERWISE BUGS SETTLE IN AND MAKE A HOME FOR THEM SELVES
IF YOU FIND A BUG: BE HONEST
BUGS DON'T LIKE GOSSIPS
IF YOU FIND A BUG: NOTE HOW YOU MEET THEM
BUGS ARE ROMANTICS
IF YOU FIND A BUG: DON'T IGNORE IT
BUGS CAN BITE IF NOT APPRECIATED
AG
Andy Glover cartoontester.blogspot.com Copyright 2010

http://cartoontester.blogspot.co.uk/

What to test?

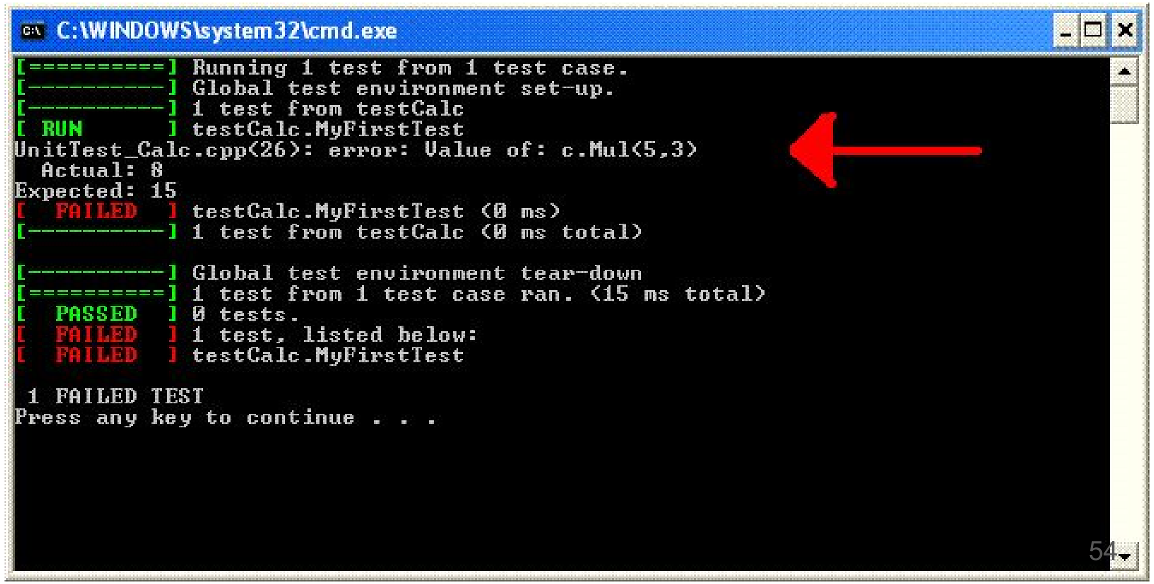
- Inputs
- Edge-cases
- Conditionals
- Manual vs Automated

Make it so simple that there are obviously no bugs,
or so complex that there are no obvious bugs.
[Tony Hoare - The Emperor’s Old Clothes (1981)]

What to test?

- Inputs
- Edge-cases
- Conditionals
- Manual vs Automated

- JUnit, TestNG, mockito
- gtest, gmock
- Karma, Protractor
- etc..



```
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from testCalc
[ RUN      ] testCalc.MyFirstTest
UnitTest_Calc.cpp(26): error: Value of: c.Mul(5,3)
  Actual: 8
Expected: 15
[  FAILED  ] testCalc.MyFirstTest (0 ms)
[----------] 1 test from testCalc (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (15 ms total)
[  PASSED  ] 0 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] testCalc.MyFirstTest

 1 FAILED TEST
Press any key to continue . . .
```

F.I.R.S.T.

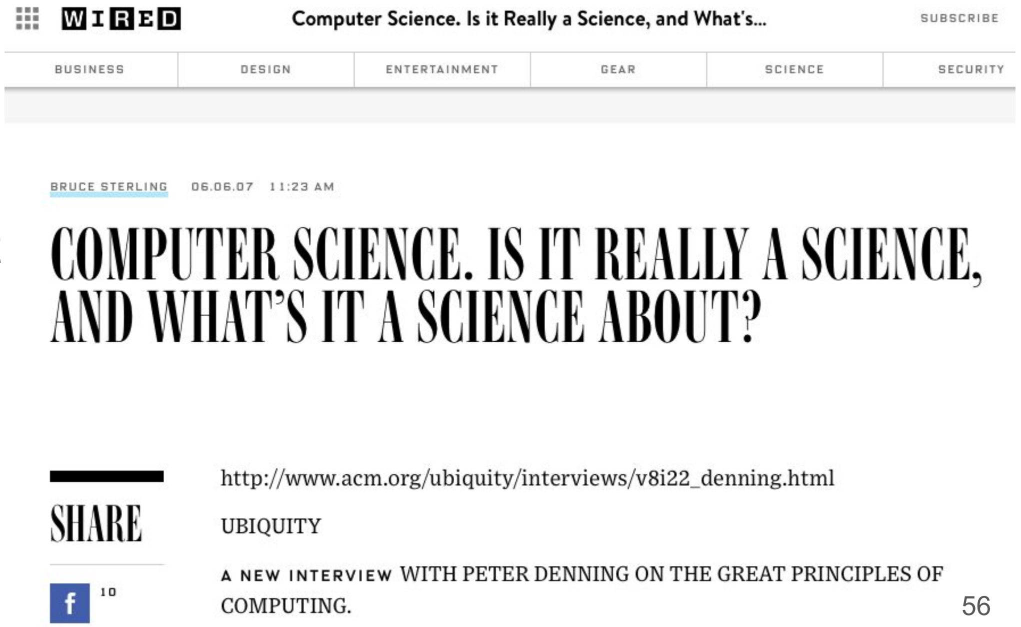
- Fast
  - so you can run them often
- Independent
  - from each other
- Repeatable
  - in any envirnment
- Self-validating
  - either pass or fail
- Timely
  - write them beforehand

Source: http://agileinaflash.blogspot.de/2009/02/first.html

Computer **Science**

Make sure your code is

- Clear
- Can take different inputs
- Repeatable
- Produces clear outputs
- Runs in the lab machines at least
- **Scientific!**



Computer Science. Is it Really a Science, and What's...

BRUCE STERLING 06.06.07 11:23 AM

COMPUTER SCIENCE. IS IT REALLY A SCIENCE, AND WHAT'S IT A SCIENCE ABOUT?

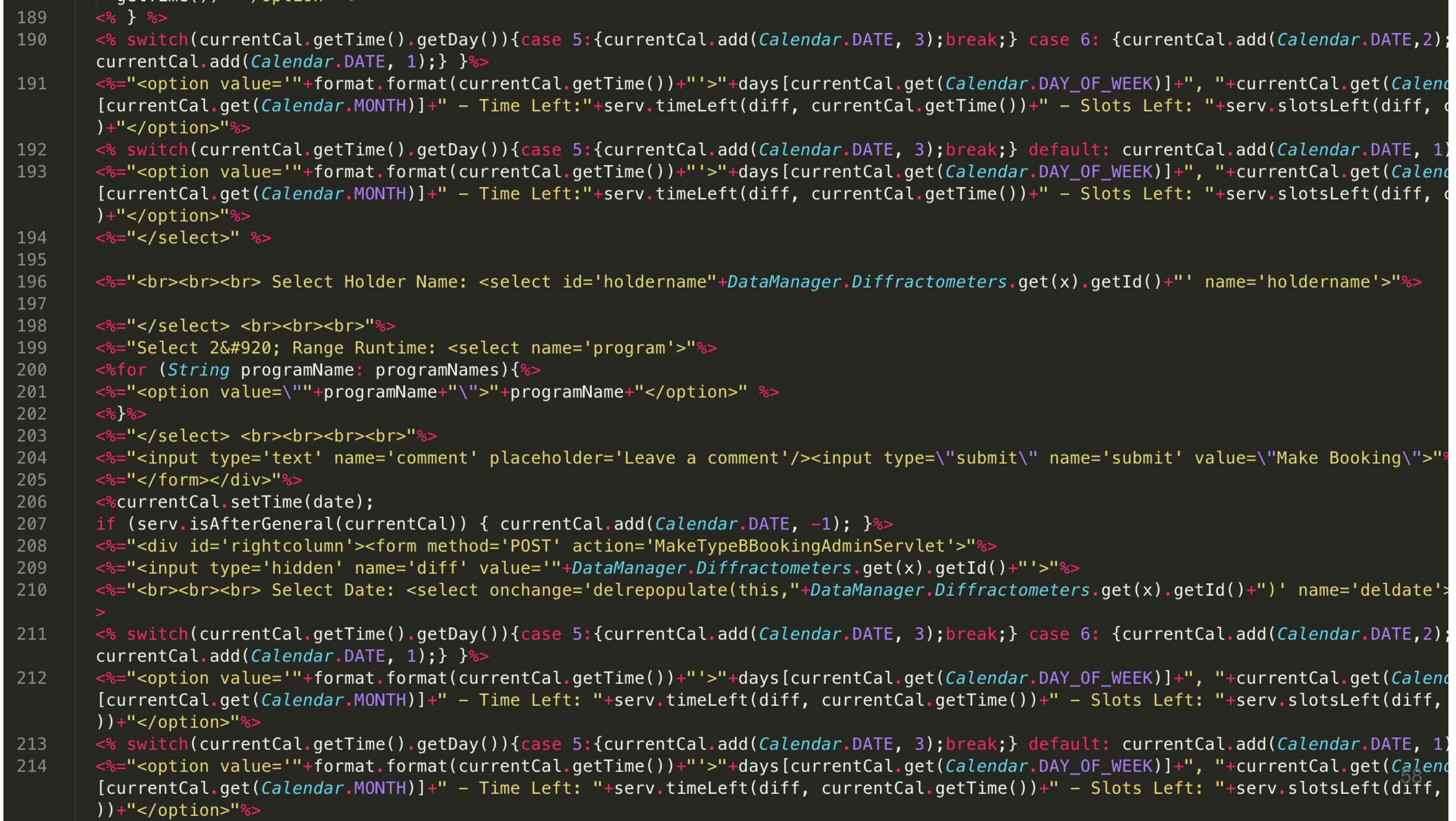
http://www.acm.org/ubiquity/interviews/v8i22_denning.html

UBIQUITY

A NEW INTERVIEW WITH PETER DENNING ON THE GREAT PRINCIPLES OF COMPUTING.

http://www.wired.com/2007/06/computer_scienc/

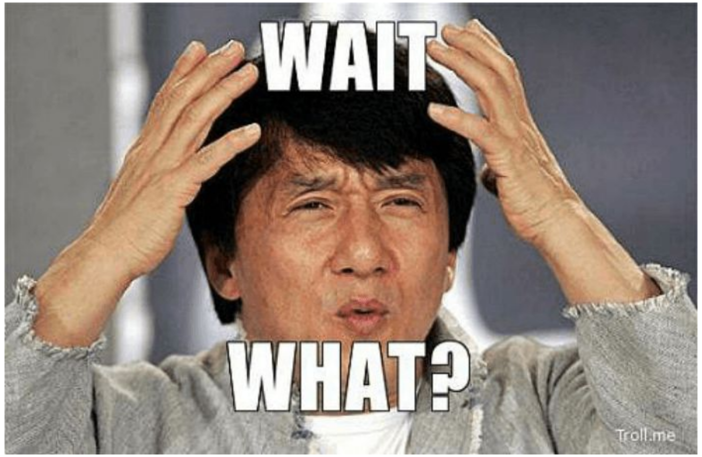
Style Matters



```
<% } %>
<% switch(currentCal.getTime().getDay()){case 5:{currentCal.add(Calendar.DATE, 3);break;} case 6: {currentCal.add(Calendar.DATE,2);
currentCal.add(Calendar.DATE, 1);} }%>
<%="<option value='"+format.format(currentCal.getTime())+"'>"+days[currentCal.get(Calendar.DAY_OF_WEEK)]+", "+currentCal.get(Calend
[currentCal.get(Calendar.MONTH)]+" - Time Left:"+serv.timeLeft(diff, currentCal.getTime())+" - Slots Left: "+serv.slotsLeft(diff, c
)+"</option>"%>
<% switch(currentCal.getTime().getDay()){case 5:{currentCal.add(Calendar.DATE, 3);break;} default: currentCal.add(Calendar.DATE, 1)
<%="<option value='"+format.format(currentCal.getTime())+"'>"+days[currentCal.get(Calendar.DAY_OF_WEEK)]+", "+currentCal.get(Calend
[currentCal.get(Calendar.MONTH)]+" - Time Left:"+serv.timeLeft(diff, currentCal.getTime())+" - Slots Left: "+serv.slotsLeft(diff, c
)+"</option>"%>
<%="</select>" %>

<%="<br><br><br> Select Holder Name: <select id='holdername"+DataManager.Diffractometers.get(x).getId()+"' name='holdername'>"%>

<%="</select> <br><br><br>"%>
<%="Select 2&#920; Range Runtime: <select name='program'>"%>
<%for (String programName: programNames){%>
<%="<option value=\""+programName+"\">"+programName+"</option>" %>
<%}%>
<%="</select> <br><br><br><br>"%>
<%="<input type='text' name='comment' placeholder='Leave a comment'/><input type=\"submit\" name='submit' value=\"Make Booking\">"%
<%="</form></div>"%>
<%currentCal.setTime(date);
if (serv.isAfterGeneral(currentCal)) { currentCal.add(Calendar.DATE, -1); }%>
<%="<div id='rightcolumn'><form method='POST' action='MakeTypeBBookingAdminServlet'>"%>
<%="<input type='hidden' name='diff' value='"+DataManager.Diffractometers.get(x).getId()+"'>"%>
<%="<br><br><br> Select Date: <select onchange='delrepopulate(this,"+DataManager.Diffractometers.get(x).getId()+")' name='deldate'>
>
<% switch(currentCal.getTime().getDay()){case 5:{currentCal.add(Calendar.DATE, 3);break;} case 6: {currentCal.add(Calendar.DATE,2);
currentCal.add(Calendar.DATE, 1);} }%>
<%="<option value='"+format.format(currentCal.getTime())+"'>"+days[currentCal.get(Calendar.DAY_OF_WEEK)]+", "+currentCal.get(Calend
[currentCal.get(Calendar.MONTH)]+" - Time Left: "+serv.timeLeft(diff, currentCal.getTime())+" - Slots Left: "+serv.slotsLeft(diff,
))+"</option>"%>
<% switch(currentCal.getTime().getDay()){case 5:{currentCal.add(Calendar.DATE, 3);break;} default: currentCal.add(Calendar.DATE, 1)
<%="<option value='"+format.format(currentCal.getTime())+"'>"+days[currentCal.get(Calendar.DAY_OF_WEEK)]+", "+currentCal.get(Calend
[currentCal.get(Calendar.MONTH)]+" - Time Left: "+serv.timeLeft(diff, currentCal.getTime())+" - Slots Left: "+serv.slotsLeft(diff,
))+"</option>"%>
```

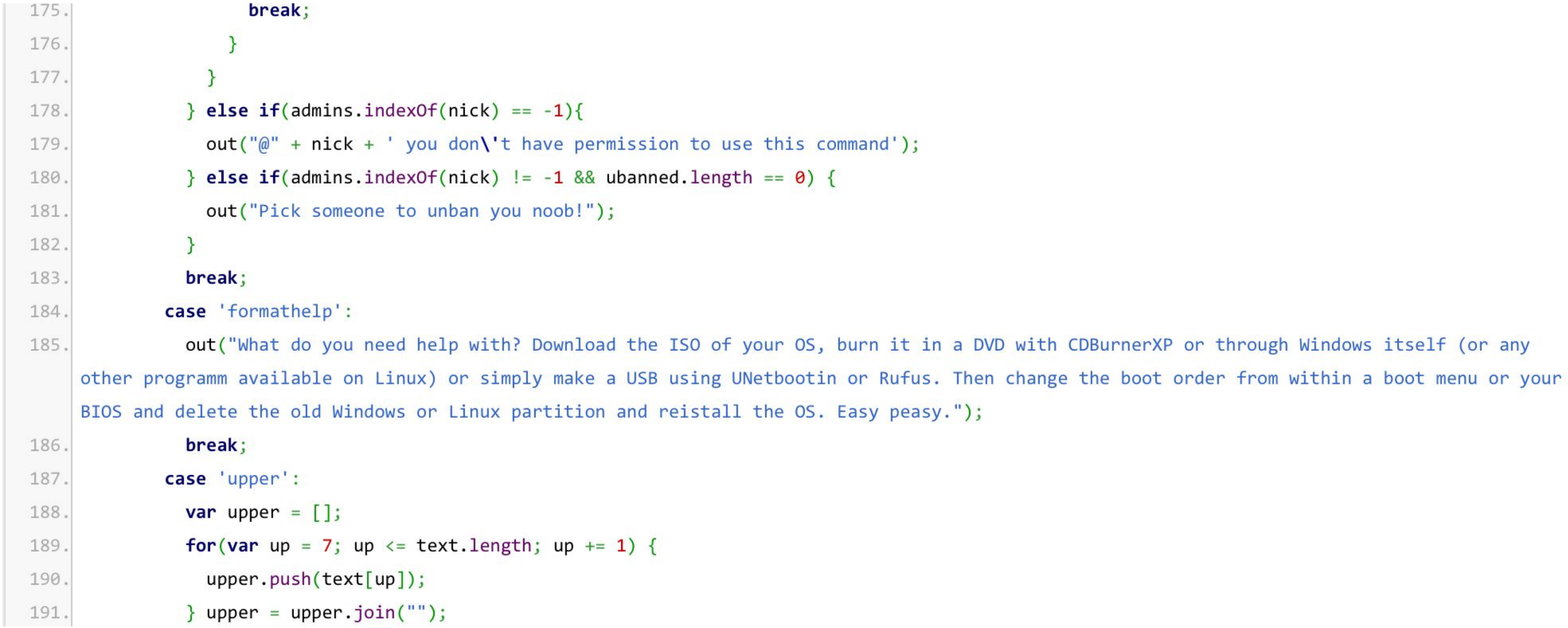
Wait… what?

```
private int parentClassType() {
  if (1 == 2)
  {
    return PARENT_MY_SCHEDULE_ACTIVITY;
  } else
  {
    return PARENT_MY_SCHEDULE_ACTIVITY;
  }
}
```



Other examples here: https://www.reddit.com/r/badcode

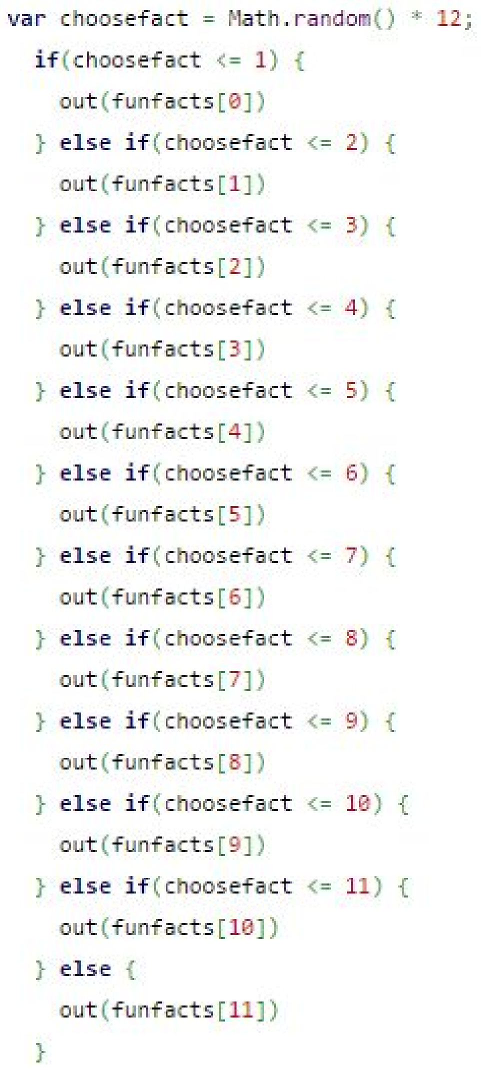
This was for a chatbot



```
                break;
              }
            }
          } else if(admins.indexOf(nick) == -1){
            out("@" + nick + ' you don\'t have permission to use this command');
          } else if(admins.indexOf(nick) != -1 && ubanned.length == 0) {
            out("Pick someone to unban you noob!");
          }
          break;
        case 'formathelp':
          out("What do you need help with? Download the ISO of your OS, burn it in a DVD with CDBurnerXP or through Windows itself (or any other programm available on Linux) or simply make a USB using UNetbootin or Rufus. Then change the boot order from within a boot menu or your BIOS and delete the old Windows or Linux partition and reistall the OS. Easy peasy.");
          break;
        case 'upper':
          var upper = [];
          for(var up = 7; up <= text.length; up += 1) {
            upper.push(text[up]);
          } upper = upper.join("");
```

https://www.reddit.com/r/badcode/comments/3ko55o/someone_tried_to_make_a_chatbot_for_a_programming/

Rounding numbers



```
var choosefact = Math.random() * 12;
  if(choosefact <= 1) {
    out(funfacts[0])
  } else if(choosefact <= 2) {
    out(funfacts[1])
  } else if(choosefact <= 3) {
    out(funfacts[2])
  } else if(choosefact <= 4) {
    out(funfacts[3])
  } else if(choosefact <= 5) {
    out(funfacts[4])
  } else if(choosefact <= 6) {
    out(funfacts[5])
  } else if(choosefact <= 7) {
    out(funfacts[6])
  } else if(choosefact <= 8) {
    out(funfacts[7])
  } else if(choosefact <= 9) {
    out(funfacts[8])
  } else if(choosefact <= 10) {
    out(funfacts[9])
  } else if(choosefact <= 11) {
    out(funfacts[10])
  } else {
    out(funfacts[11])
  }
```

https://www.reddit.com/r/badcode/comments/3j37iy/if_you_can_not_round_numbers/

Obfuscated Code

```
float s=1944,x[5],y[5],z[5],r[5],j,h,a,b,d,e;int i=33,c,l,f=1;int g(){return f=
(f*6478+1)%65346;}m(){x[i]=g()-l;y[i]=(g()-l)/4;r[i]=g()>>4;}main(){char t[1948
]=" `MYmtw%FFlj%Jqig~%`jqig~Etsqnsj3stb",*p=t+3,*k="3tjlq9TX";l=s*20;while(i<s)
p[i++]='\n'+5;for(i=0;i<5;i++)z[i]=(i?z[i-1]:0)+l/3+!m();while(1){for(c=33;c<s;
c++){c+=!((c+1)%81);j=c/s-.5;h=c%81/40.0-1;p[c]=37;for(i=4;i+1;i--)if((b=(a=h*x
[i]+j*y[i]+z[i])*a-(d=1+j*j+h*h)*(-r[i]*r[i]+x[i]*x[i]+y[i]*y[i]+z[i]*z[i]))>0)
{for(e=b;e*e>b*1.01||e*e<b*.99;e-=.5*(e*e-b)/e);p[c]=k[(int)(8*e/d/r[i])];}}for
(i=4;i+1;z[i]-=s/2,i--)z[i]=z[i]<0?l*2+!m():z[i];while(i<s)putchar(t[i++]-5);}}
```

Source http://www.ioccc.org/years.html#1996_eldby

Good code example

```
public static void main(String[] args) {
    System.out.println("Hello, World");
}
```

Checking Style Tools

There are many of them.

- Java - **PMD, FindBugs**, JTest, **Checkstyle**
- C - QA-C, CodeSonar
- C++ - PC-Lint, C++Test, Coverity
- C# - FxCop, StyleCop, TICS
- Python - pylint, pep-8, pychecker
- etc ...

Many False-Positives!

Many options. There is not a right one → best check style tool is you!

Watch-Out for warnings

- C/C++
  - -Werror (make all warnings into errors); -Wall; -Wextra
  - Windows - /Wn; /WX; /Wall; etc
  - Look after your compiler
- Java
  - -Xlint (enables all recommended warnings)
  - http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/javac.html
- Know your language

Indentation

Many styles → https://en.wikipedia.org/wiki/Indent_style

There is not a right one

Be reasonable and consistent

Formatting

- Vertical formatting
  - keep things to a length that people will want to read.
  - separate thoughts by empty lines, keep related thoughts close
- Place variables close to where they are used.
- Instance variables should be at the top of a class.
- Prefer short lines over long ones, avoid lines of more than 80 characters, never use more than, say, 100.
- Use IDE tools to keep things tidy and aligned with a consistent code style (week 7 - 1st year)

Formatting

- Space vs Tabs

Formatting - Example

```
private int x; // this is fine
private Color color; // this too

private int  x;       // permitted, but future edits
private int  y;


private Color color;  // may leave it unaligned
private int  y;
```

Google Java Style guide: https://google.github.io/styleguide/javaguide.html#s4.6.3-horizontal-alignment

Comments (I)

```
/**
* For the brave souls who get this far: You are the chosen ones,
* the valiant knights of programming who toil away, without rest,
* fixing our most awful code. To you, true saviors, kings of men,
* I say this: never gonna give you up, never gonna let you down,
* never gonna run around and desert you. Never gonna make you cry,
* never gonna say goodbye. Never gonna tell a lie and hurt you.
*/
```

Credits to the SO Community - http://stackoverflow.com/questions/184618/what-is-the-best-comment-in-source-code-you-have-ever-encountered

Comments (II)

```
//
// Dear maintainer:
//
// Once you are done trying to 'optimize' this routine,
// and have realized what a terrible mistake that was,
// please increment the following counter as a warning
// to the next guy:
//
// total_hours_wasted_here = 42
//
```

Credits to the SO Community - http://stackoverflow.com/questions/184618/what-is-the-best-comment-in-source-code-you-have-ever-encountered

Comments (III)

```
//When I wrote this, only God and I understood what I was doing
//Now, God only knows
```

Credits to the SO Community - http://stackoverflow.com/questions/184618/what-is-the-best-comment-in-source-code-you-have-ever-encountered

Comments (IV)

```
return 1; # returns 1
```

Credits to the SO Community - http://stackoverflow.com/questions/184618/what-is-the-best-comment-in-source-code-you-have-ever-encountered

Comments (V)

```
/**
 * Always returns true.
 */
public boolean isAvailable() {
    return false;
}
```

Credits to the SO Community - http://stackoverflow.com/questions/184618/what-is-the-best-comment-in-source-code-you-have-ever-encountered

Comments (VI)

```
long long ago; /* in a galaxy far far away */
```

Credits to the SO Community - http://stackoverflow.com/questions/184618/what-is-the-best-comment-in-source-code-you-have-ever-encountered

Comments (VII)

```
// I can't divide with zero, so I have to divide with something
very similar
result = number / 0.00000000000001;
```

Credits to the SO Community - http://stackoverflow.com/questions/184618/what-is-the-best-comment-in-source-code-you-have-ever-encountered

Lessons learned

Funny comments waste time - wasted time to write, wasted time to read, wasted time to show to your colleagues the funny remark that is (almost always) merely puzzling and so on.

If funny comments were actually funny it would change my mind. But once you encourage jokes, do you encourage swearing or insults or maliciousness?

Credits to user **amelvin** - http://programmers.stackexchange.com/questions/60699/is-funny-commenting-a-bad-practice-or-not

Other bad examples

```
fred.penColor(Color.blue);
     // the colour of the pen has been set to blue


tom.bodyColor(Color.red);
     // the turtle's body colour has been changed to blue



//imports Turtle Donald into Drawing Window Object
DrawingWindow.add(Martin);
```

Good Comments

- Legal comments (author, copyright, license etc.);
  - IDEs might collapse lengthy legal comments
- Informative Comments
- JavaDoc comments in public APIs
- Expressing intention
- Clarifications
- Explaining code that cannot be changed and is therefore unclear
  - e.g., when using a standard library
- Warning of consequences
- TODO/FIXME comments
- Highlighting something as important that is not obvious

Good Comments - Example I

```
// format matcher kk:mm:ss EEE, MMM dd, yyyy
Pattern timeMatcher =
       Pattern.compile(“\\d*:\\d*:\d* \\w*, \\w* \\d*, \\d*”);
```

Credits to CleanCode

Good Comments - Example II

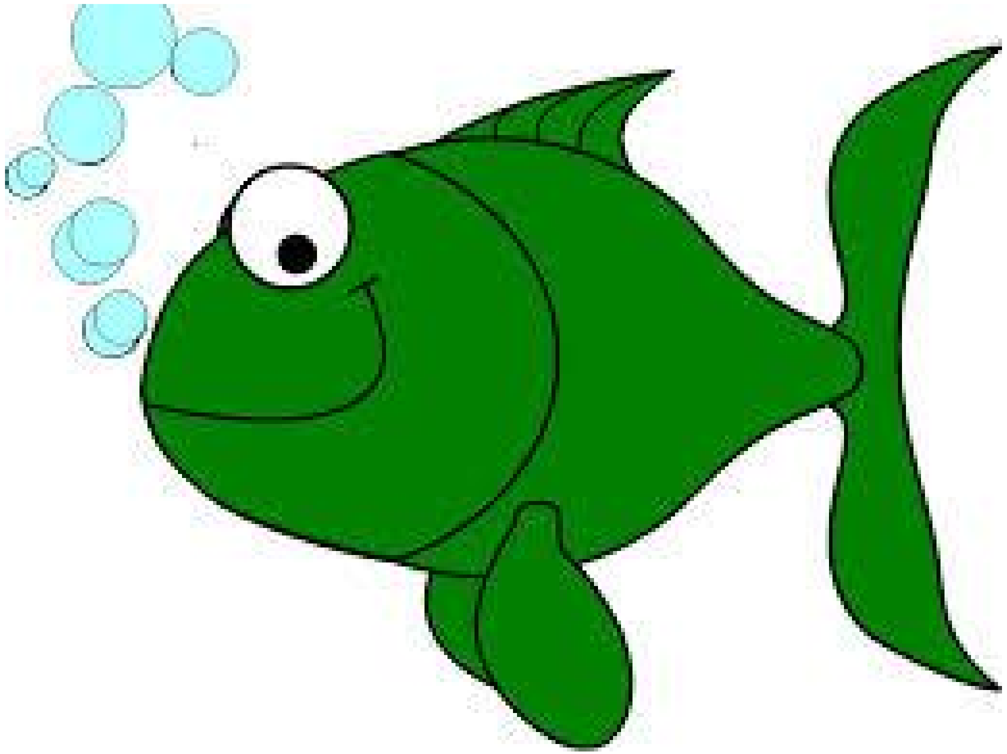
```
// Every multiple in the array has a prime factor that
// is less than or equal to the root of the array size,
// so we don’t have to cross out multiples of numbers
// larger than that root
double iterationLimit = Math.sqrt(crossedOut.length);
return (int) iterationLimit;
```

Credits to CleanCode

Summary

Don’t comment bad code, rewrite it
(Kernighan, Plaugher)

- Be informative
- Be clear
- Be consistent
- Avoid comments if you can communicate intention through code
- Do not leave commented code around if not needed



Naming Practices

Names are everywhere in software. And software must be readable by us and others.

- Intention-revealing names
- Avoid disinformation
- Pronouncable and searchable names

Use Intention-Revealing names

- the names of variables, methods, classes etc. should tell what these things are about.

Bad

- `d, h, t, es, p`
- `list, array, space`
- `getThem(), get()`
- `users, active`

Good

- `distance, height, time`
- `board, cell`
- `getWeight(), getMachine()`
- `numberOfUsers, numberOfActiveUsers`

Avoid Disinformation

- do not refer to something as a List unless it actually is a List.

Bad

- `hp, aix, sco (UNIX)`
- `accountList (for array)`
- `XYZAccountControllerList vs XYZAccountHandlerList`
- `l (vs 1)`
- `O (vs 0)`
- `O1, Ol, 0l, 01`

Good

- `hypotenuse, axes, etc`
- `accountGroups, accounts`
- `accountController vs accountHandler`

Pronouncable and Searchable names

Humans are good at words. And words are pronouncable.

Bad	Good
`DtaRcrd102`	`Customer`
`genymdhms`	`generationTimestamp`
`modymdhms`	`modificationTimestamp`
`srchbl`	`searchable`
`7, 15`	`MAX_NUMBER_MACHINES, TASKS_THRESHOLD`
`t`	`tasks`

Others

- Classes should have noun or noun-phrase names rather than verb names
    - avoid empty words such as Info, Data, Manager, Processor in a class name
    - Ex: Customer, WikiPage, Account, HashIntegrationTest, ImmutableList, etc
- **Methods should have a verb name**
    - Accessors, mutators, and predicates should be named for their value and prefixed with get, set, and is
    - Ex: sendMessage(), stop(), receiveMessage(), sendAck(), getUsername(), etc
- **"Don't be cute"**
    - Ex: kill(), abort(), deleteItems() instead of whack(), eatMyShorts(), HolyHandGranade()
- **Avoid single-letter variable names** and similar things that have nothing to do with the problem domain.

Check https://studres.cs.st-andrews.ac.uk/CS5031/Lectures/week04/cleancode.html (based on Clean Code book)

Others

- Classes → use singular names
- Fine to have one-single-letter variables for counter variables
  - for(int i = 0; i < CONSTANT; i++)
- Some counter variable may be better “extended”
  - for(int cell=0; cell < board[0].length; cell+)

Check https://studres.cs.st-andrews.ac.uk/CS5031/Lectures/week04/cleancode.html (based on Clean Code book)

Summary

- Be consistent
- Be reasonable
- Use meaningful names
- Use readable names
- Use searchable names
- Use unambiguous names
- Avoid colloquialism

Daniel Patterson, Sumdog

Think hard about how to name things. Names for objects and variables should describe what it is so that you don't need to refer back to its definition. It should describe what it is now, not what you want it to be eventually.

Single Responsibility Principle - objects and functions should do one thing. This means that you should be able to describe them without using 'and' / 'or'. Typically, it means **functions shouldn't be more than 5 or 6 lines of code** and shouldn't be indented more than once - so only one level of loops or conditional branching. This makes code much easier to read and reuse.

Methods/Functions

- Make them small!
- **Do one thing, do it well, do it only!**
- Keep them about 15-20 lines long MAX!
    - Others suggest to have about 5-10 lines AND max 1 conditional statement

- Do not mix levels of abstraction
- Have no side effects
- Either change state or return a value/object
- Do not repeat yourself!

Methods/Functions - Example I

Bad

```
public boolean checkPwd(String[] usr, String[] pwd) {
    this.pwd = pwd;
    int[] valids = new int[10];
    for(int i = 0; i < 10; i++)
        if (usr[i].pwd == pwd[i] && pwd[i].length() > 10)
            valids[i] = 1;
    int c = 0;
    for(int i = 0; i < 10; i++) if (valids[i]) c++;
    return c > 5 ? true : false;
}
```

Methods/Functions - Example I

Better

```
public boolean checkPassword(String user, String password) {
    return (user.password.equals(password) &&
        password.length() > MIN_PASSWORD_SIZE);
}
```

Methods/Functions - Example I

Better ???

Ideally we should check the password validity before having set it

```
public boolean checkPassword(String user, String password) {
    return (user.password.equals(password) &&
        password.length() > MIN_PASSWORD_SIZE);
}
```

Methods/Functions - Example II

BAD

```
boolean val = true;
boolean stop = false;
while (i < n && !stop) {
    if (f(i)) {
        val = false;
        stop = true;
    }
    i++;
}
return val;
```

Methods/Functions - Example II

BETTER

```
while (i < n) {
    if (f(i))
        return false;
    i++;
}
return true;
```

Methods/Functions - Example III

BAD

```
if (f1(i))
    return true;
else if (f2(i))
    return true;
else
    return false;
```

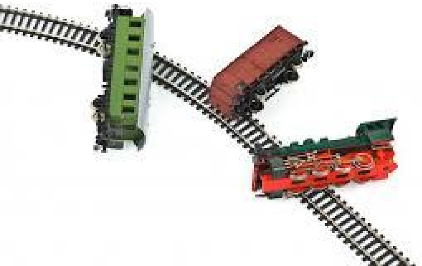
Methods/Functions - Example III

BETTER

```
return f1(i) || f2(i);
```

Trainwrecks

```
client.getAccounts().getAccount(1).applyPayment(300.00);
```

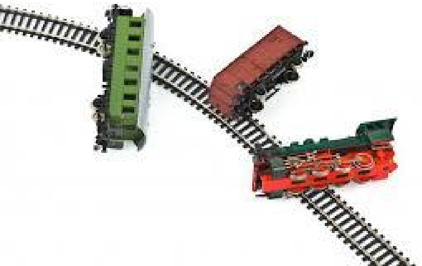


Trainwrecks

client.getAccounts().getAccount(1).applyPayment(300.00)

Better

```
AccountsGroup accounts = client.getAccounts();

Account accountToPay = accounts.getAccount(1);

accountToPay.applyPayment(300.0);
```



Magic Numbers

```
if (boom==3)
    explode();
else
    later(5000);
```

Magic Numbers

Better

```
private final static int TIMER_IS_OFF = 3;
private final static int BOMB_WAITING_WINDOW = 5000; // in ms

if (boom==TIMER_IS_OFF)
    explode();
else
    later(BOMB_WAITING_WINDOW);
```

Static methods

Ask yourself:

“does it make sense to call this method, even if no Object has been constructed yet?”

- convertMilesToKilometers()
- setSpeed()
- performFuelCleanup()
- removeWheel()
- makeString(int[])

Static methods - Tips

1. If you are writing **utility classes** and they are not supposed to be changed.
2. If the method is not using any instance variable.
3. If any operation is not dependent on instance creation.
4. If there is **some code that can easily be shared by all the instance methods**, extract that code into a static method.
5. If you are sure that **the definition of the method will never be changed or overridden.** As static methods can not be overridden.

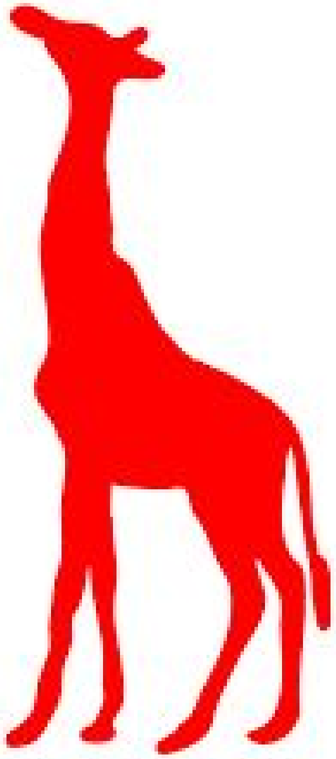
Some advice from great experts

Lots of indentation (> 2 levels) is a bad smell - I would tend to extract the indented code to another private method.
[R. McMahon - Adobe]

Some advice from great experts

I think for early students, **the most important thing is to try to make their program compile and run** (so they can test it) as early as possible, then **keep iterating** by adding small features and retesting. [...] Automating these tests will make things easier, but **the important thing is to try not to write a few hundred lines of code, and only then try to make their code compile / run**.

[C. Jefferson - St Andrews]



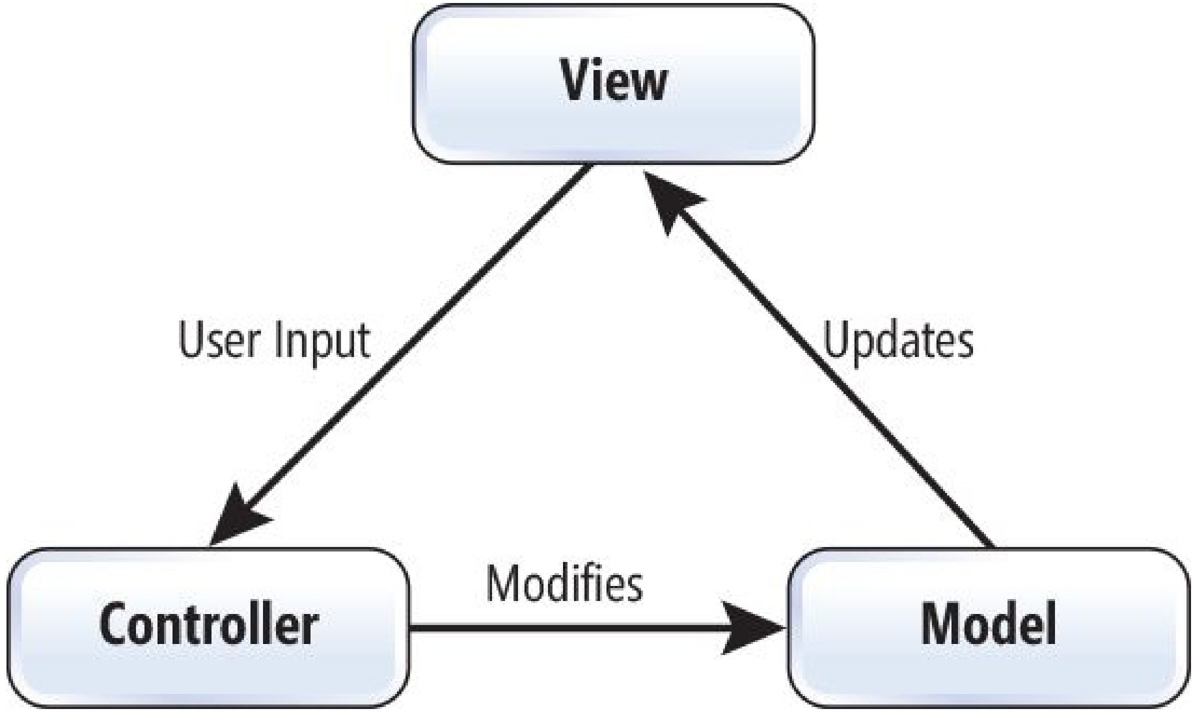
The Simpler The Better

- Patterns
- OOP
- Reusability
- Refactoring
- APIs

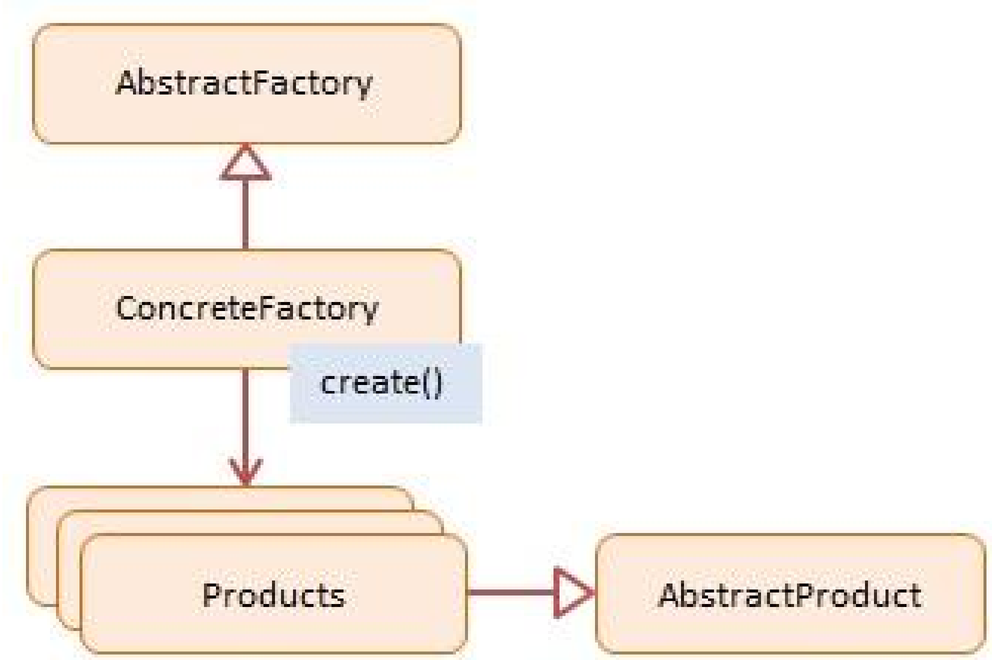
Software Design Patterns

- Abstract Factory
- MVC
- Singleton
- Observer
- RAII - Resource Acquisition Is Initialisaiton
- Proxy
- Etc...

MVC - Model-View-Controller



Abstract Factory



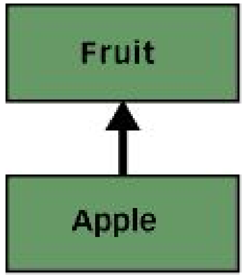
OOP

- Inheritance
- Composition
- Generics
- Interfaces
- Abstraction
- Polymorphism
- Etc.

Example: Composition over Inheritance

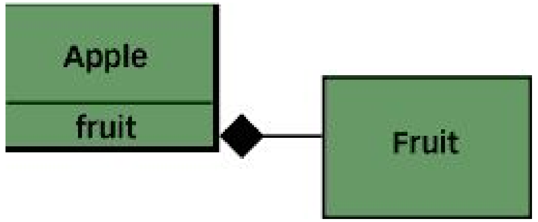
B:

- higher flexibility
- better maintanability in the long term
- changes do not propagate over the hierarchy
- Go uses composition exclusively



C:

- multiple methods’ implementations



What the experts suggest

- Lean heavily on your **tools**.
- Design systems in layers, so that they can be composed, layers can be swapped out, etc.
- Try to keep your layers **simple** - the less they do the less likely you are to want/need to replace them at a later date.

[Louis Morgan - Adobe]

Reusability & Refactoring

Write code once if possible - if you find yourself writing the same code twice (or cutting and pasting some code) use a function instead.

[Peter Brown - Adobe]

Do not be afraid of changing and refactoring your code/design.

Interfaces/API

- Be clear and consistent
- Offer a variety of data formats (JSON, XML, etc) (API mostly)
- Make the syntax intuitive and easy to understand
- Thorough documentation
- Use proper response codes

Don’t put instance data in interfaces (interfaces are for function declarations).
[Peter Brown - Adobe]

Resources:
http://stackoverflow.com/questions/6500468/recommendations-for-writing-an-api
http://lcsd05.cs.tamu.edu/slides/keynote.pdf

Reproducibility

- CS → Computer **Science**
- Do not hardcode the state of your program
- Let the user input parameters
- Let the user use different configurations
- Avoid randomness wherever possible

Simplistic vs Engineered

Hot-topic

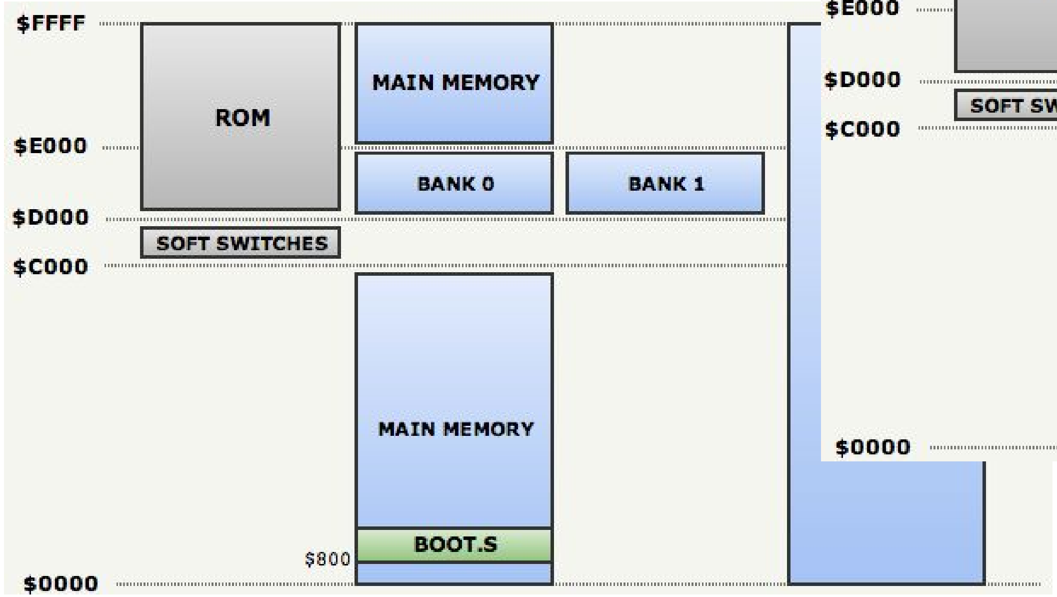
- Too simplistic
- Overly engineered
- Find a tradeoff

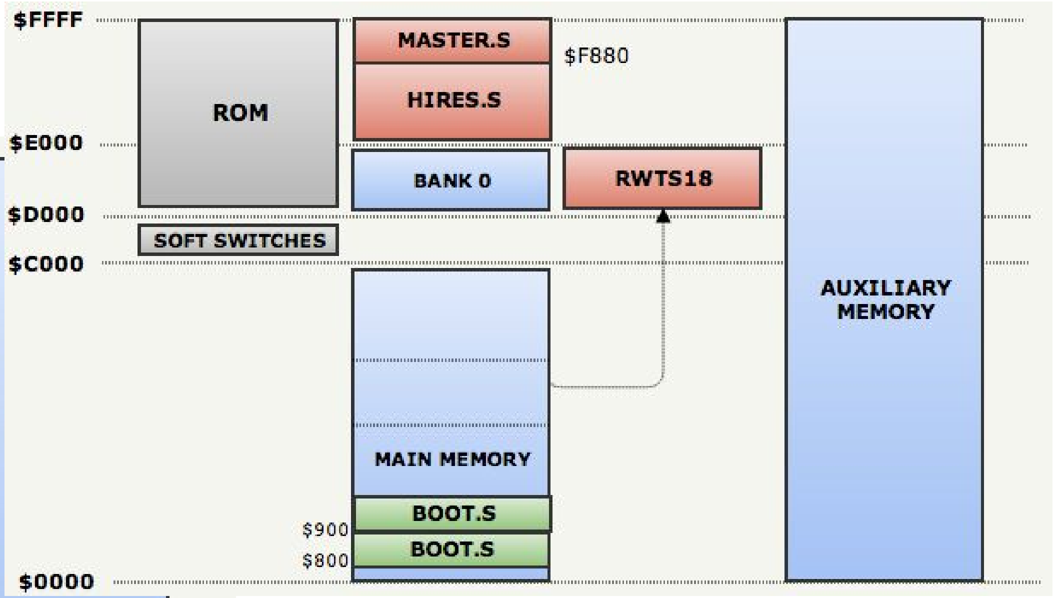
HTTP Status Code example:
https://blogs.dropbox.com/developers/2015/04/how-many-http-status-codes-should-your-api-use/

Prince of Persia

http://www.fabiensanglard.net/prince_of_persia/pop_boot2.php

Prince of Persia





http://www.fabiensanglard.net/prince_of_persia/pop_boot2.php

Today your life is easier...or not?

Today your life is easier...or not?

DO NOT DO THIS!

1. Lie in the comments. You don't have to actively lie, just fail to keep comments as up to date with the code.
2. Make sure that every method does a little bit more (or less) than its name suggests.
3. In the name of efficiency, use cut/paste/clone/modify. This works much faster than using many small reusable modules.
4. Try to pack as much as possible into a single line
5. Use very long variable names that differ from each other by only one character, or only in upper/lower case.
6. In naming functions, make heavy use of abstract words like *it*, *everything*, *data*, *handle*, *stuff*,..
7. Make as many of your variables as possible static.
8. Declare every method and variable public. After all, somebody, sometime might want to use it.
9. You get the pattern...

https://www.doc.ic.ac.uk/~susan/475/unmain.html

Good Academic Practice

TGAP - https://info.cs.st-andrews.ac.uk/student-handbook/academic/gap.html

CS Handbook - https://info.cs.st-andrews.ac.uk/student-handbook/academic/gap.html

Example - README

# Third party resources

The makefile is a modified version of the one provided by [Michael Safyan](https://sites.google.
com/site/michaelsafyan/software-engineering/how-to-write-a-makefile)

Wavefront .obj loader (slightly modified version): [objLoader](http://www.kixor.net/dev/objloader/)

Image library (C/C++): [CImg](http://cimg.sourceforge.net/)

Images were all downloaded from internet. See appropriate text file in Data/ for specific links.

Example - In your code

```
/*
* Profiling requires Google Perf Tools to be installed.
* The tools are installed in my Mac, but not in the Lab Linux Machine I use. Therefore, profiling is always disabled for the
latter machine.
* http://goog-perftools.sourceforge.net/
*/
#ifdef __APPLE__
#include <google/profiler.h>
void Helper::START_PROFILING(std::string fileName)
{
   if(PROFILER)
             ProfilerStart(fileName.c_str());
}
void Helper::STOP_PROFILING()
{
   if(PROFILER)
             ProfilerStop();
}
#else
void Helper::START_PROFILING(std::string fileName) {}
void Helper::STOP_PROFILING() {}
#endif
```

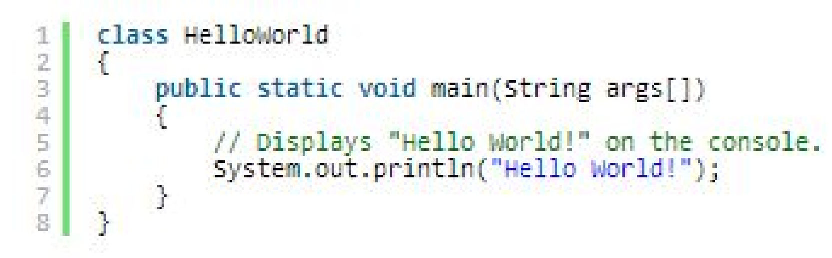
Example - In your code

```
// @see http://stackoverflow.com/questions/3585846/color-text-in-terminal-aplications-in-unix
// e.g. printf("%sred\n", KRED);

#define KGRN  "\x1B[32m"
#define KNRM  "\x1B[0m"
#define KRED  "\x1B[31m"
#define KYEL  "\x1B[33m"
#define KBLU  "\x1B[34m"
#define KMAG  "\x1B[35m"
#define KCYN  "\x1B[36m"
#define KWHT  "\x1B[37m"
```

Give credits to others

- There is not a specific way of doing it as of today (15/10/2015)
- Ask yourself “am I telling others that this code is from XYZ?”
  - If not, probably you are doing something wrong

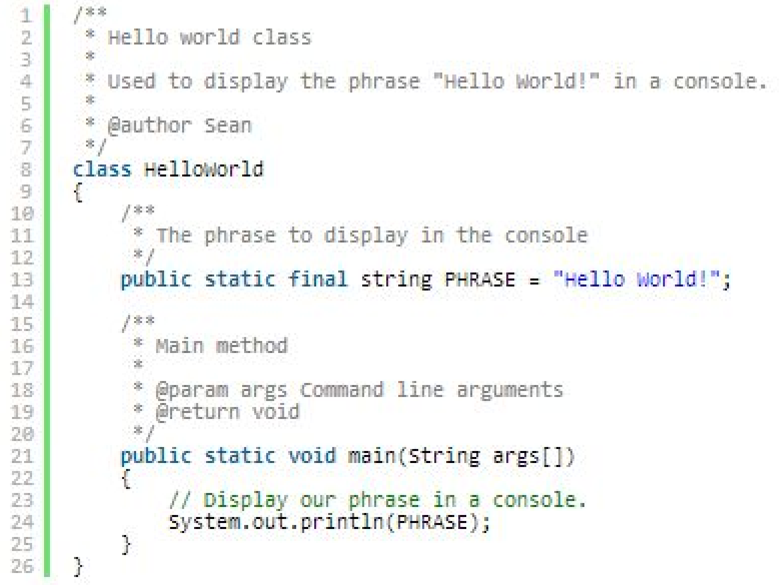
The evolution of a SE (I yr)



```
class HelloWorld
{
    public static void main(String args[])
    {
        // Displays "Hello World!" on the console.
        System.out.println("Hello World!");
    }
}
```

Credits to Sean Hickey - https://medium.com/@webseanhickey/the-evolution-of-a-software-engineer-db854689243

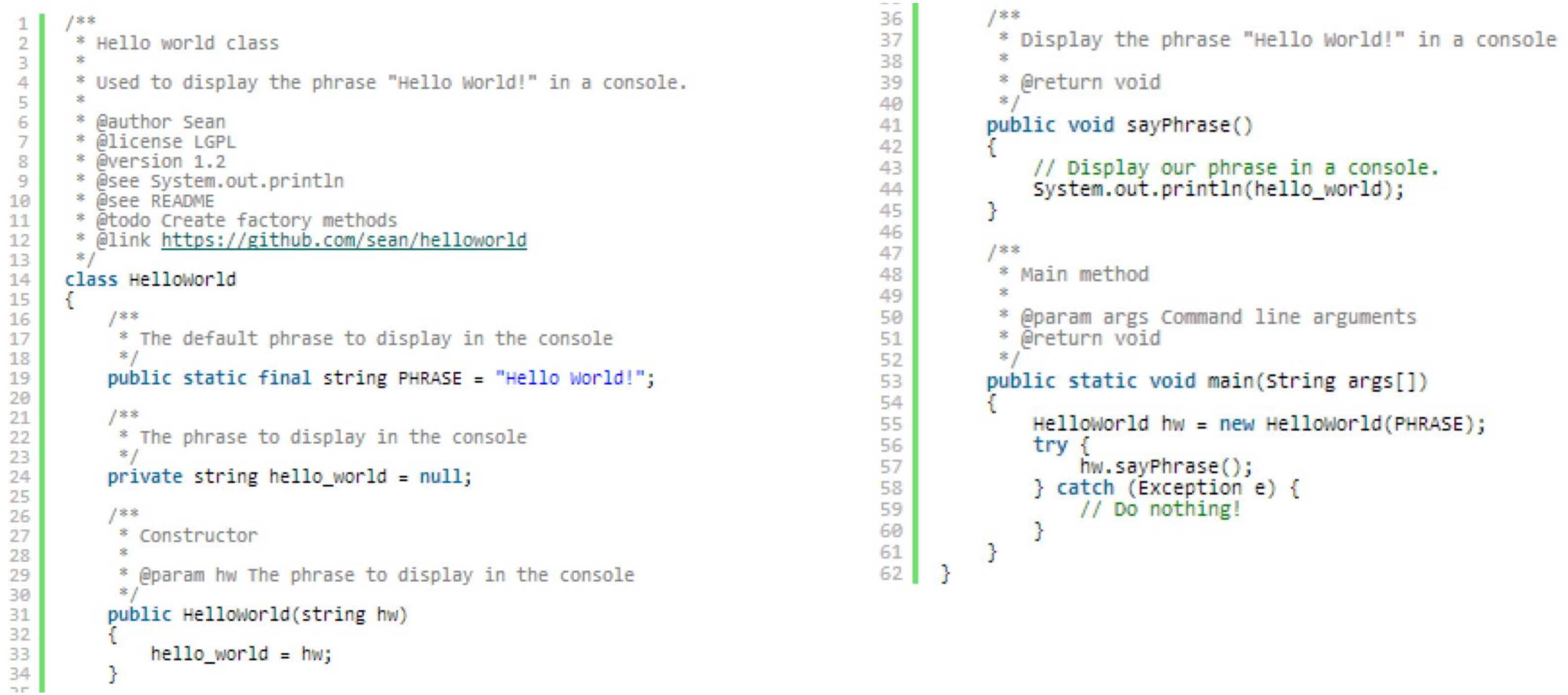
The evolution of a SE (II yr)



```
/**
 * Hello world class
 *
 * Used to display the phrase "Hello World!" in a console.
 *
 * @author Sean
 */
class HelloWorld
{
    /**
     * The phrase to display in the console
     */
    public static final string PHRASE = "Hello World!";

    /**
     * Main method
     *
     * @param args Command line arguments
     * @return void
     */
    public static void main(String args[])
    {
        // Display our phrase in a console.
        System.out.println(PHRASE);
    }
}
```

Credits to Sean Hickey - https://medium.com/@webseanhickey/the-evolution-of-a-software-engineer-db854689243

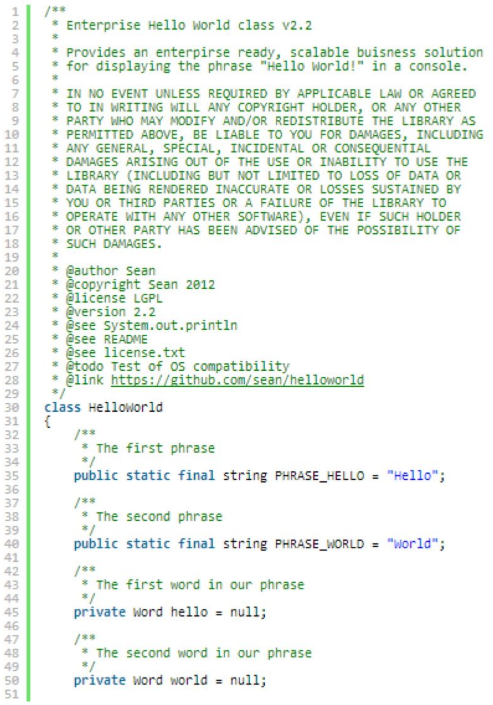
The evolution of a SE (III yr)



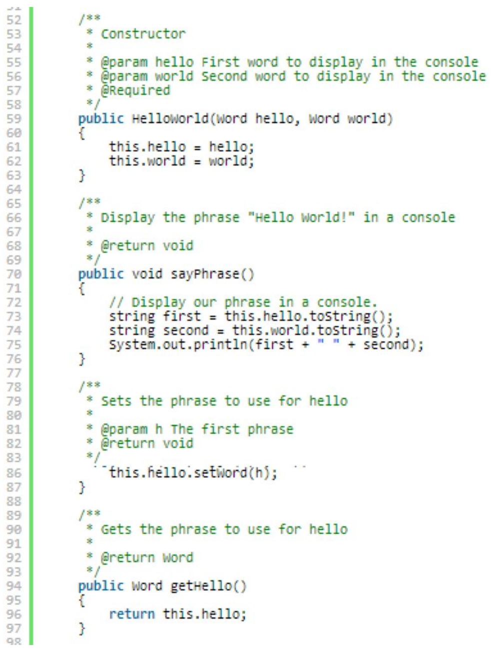
```
/**
 * Hello world class
 *
 * Used to display the phrase "Hello World!" in a console.
 *
 * @author Sean
 * @license LGPL
 * @version 1.2
 * @see System.out.println
 * @see README
 * @todo Create factory methods
 * @link https://github.com/sean/helloworld
 */
class HelloWorld
{
    /**
     * The default phrase to display in the console
     */
    public static final string PHRASE = "Hello World!";

    /**
     * The phrase to display in the console
     */
    private string hello_world = null;

    /**
     * Constructor
     *
     * @param hw The phrase to display in the console
     */
    public HelloWorld(string hw)
    {
        hello_world = hw;
    }

    /**
     * Display the phrase "Hello World!" in a console
     *
     * @return void
     */
    public void sayPhrase()
    {
        // Display our phrase in a console.
        System.out.println(hello_world);
    }

    /**
     * Main method
     *
     * @param args Command line arguments
     * @return void
     */
    public static void main(String args[])
    {
        HelloWorld hw = new HelloWorld(PHRASE);
        try {
            hw.sayPhrase();
        } catch (Exception e) {
            // Do nothing!
        }
    }
}
```

Credits to Sean Hickey - https://medium.com/@webseanhickey/the-evolution-of-a-software-engineer-db854689243

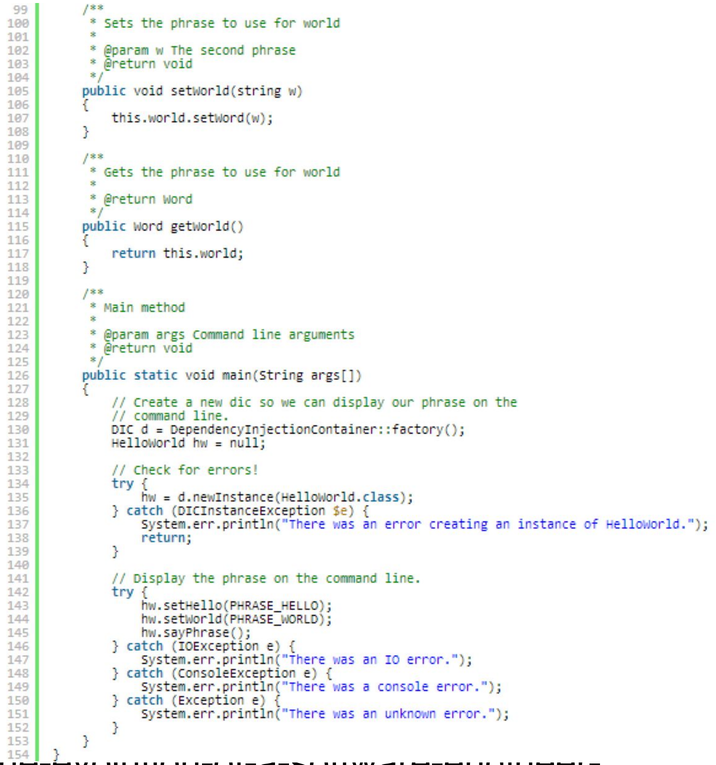
The evolution of a SE (V yr)



```
/**
 * Enterprise Hello World class v2.2
 *
 * Provides an enterpirse ready, scalable buisness solution
 * for displaying the phrase "Hello World!" in a console.
 *
 * IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED
 * TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER
 * PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS
 * PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING
 * ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL
 * DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE
 * LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR
 * DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
 * YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO
 * OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER
 * OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGES.
 *
 * @author Sean
 * @copyright Sean 2012
 * @license LGPL
 * @version 2.2
 * @see System.out.println
 * @see README
 * @see license.txt
 * @todo Test of OS compatibility
 * @link https://github.com/sean/helloworld
 */
class HelloWorld
{
    /**
     * The first phrase
     */
    public static final string PHRASE_HELLO = "Hello";

    /**
     * The second phrase
     */
    public static final string PHRASE_WORLD = "World";

    /**
     * The first word in our phrase
     */
    private Word hello = null;

    /**
     * The second word in our phrase
     */
    private Word world = null;

```



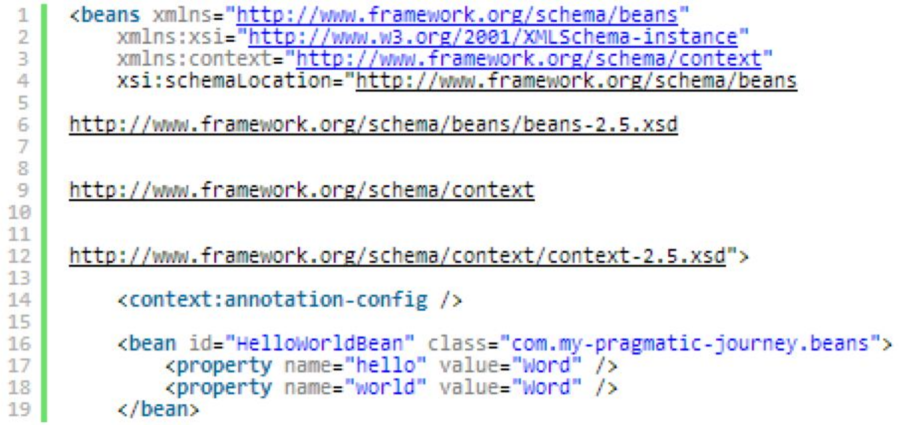
```
/**
 * Constructor
 *
 * @param hello First word to display in the console
 * @param world Second word to display in the console
 * @Required
 */
public HelloWorld(Word hello, Word world)
{
    this.hello = hello;
    this.world = world;
}

/**
 * Display the phrase "Hello World!" in a console
 *
 * @return void
 */
public void sayPhrase()
{
    // Display our phrase in a console.
    string first = this.hello.toString();
    string second = this.world.toString();
    System.out.println(first + " " + second);
}

/**
 * Sets the phrase to use for hello
 *
 * @param h The first phrase
 * @return void
 */
    this.hello.setWord(h);
}

/**
 * Gets the phrase to use for hello
 *
 * @return Word
 */
public Word getHello()
{
    return this.hello;
}
```



```
/**
 * Sets the phrase to use for world
 *
 * @param w The second phrase
 * @return void
 */
public void setWorld(string w)
{
    this.world.setWord(w);
}

/**
 * Gets the phrase to use for world
 *
 * @return Word
 */
public Word getWorld()
{
    return this.world;
}

/**
 * Main method
 *
 * @param args Command line arguments
 * @return void
 */
public static void main(String args[])
{
    // Create a new dic so we can display our phrase on the
    // command line.
    DIC d = DependencyInjectionContainer::factory();
    HelloWorld hw = null;

    // Check for errors!
    try {
        hw = d.newInstance(HelloWorld.class);
    } catch (DICInstanceException $e) {
        System.err.println("There was an error creating an instance of HelloWorld.");
        return;
    }

    // Display the phrase on the command line.
    try {
        hw.setHello(PHRASE_HELLO);
        hw.setWorld(PHRASE_WORLD);
        hw.sayPhrase();
    } catch (IOException e) {
        System.err.println("There was an IO error.");
    } catch (ConsoleException e) {
        System.err.println("There was a console error.");
    } catch (Exception e) {
        System.err.println("There was an unknown error.");
    }
}
}
```

Credits to Sean Hickey - https://medium.com/@webseanhickey/the-evolution-of-a-software-engineer-db854689243

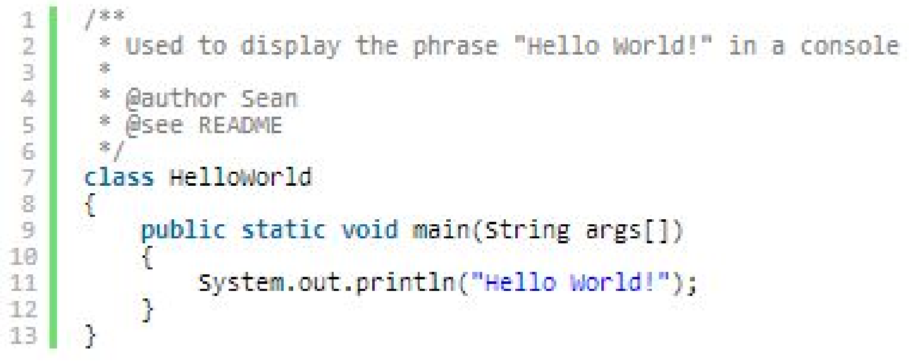
The evolution of a SE (V yr) cont



```xml
<beans xmlns="http://www.framework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.framework.org/schema/context"
    xsi:schemaLocation="http://www.framework.org/schema/beans

http://www.framework.org/schema/beans/beans-2.5.xsd


http://www.framework.org/schema/context


http://www.framework.org/schema/context/context-2.5.xsd">

    <context:annotation-config />

    <bean id="HelloWorldBean" class="com.my-pragmatic-journey.beans">
        <property name="hello" value="Word" />
        <property name="world" value="Word" />
    </bean>
```

Credits to Sean Hickey - https://medium.com/@webseanhickey/the-evolution-of-a-software-engineer-db854689243

The evolution of a SE (X yr)

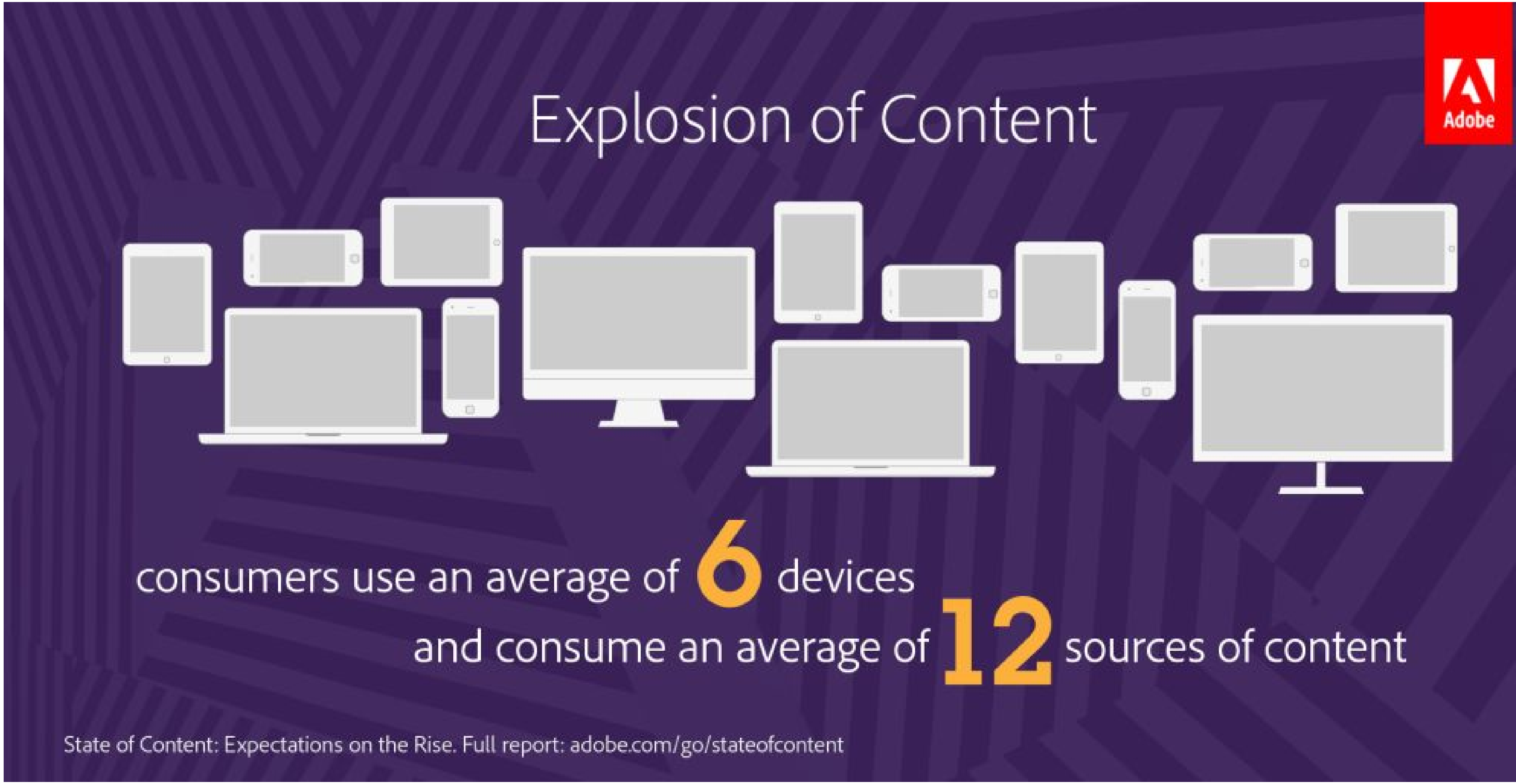


```
/**
 * Used to display the phrase "Hello World!" in a console
 *
 * @author Sean
 * @see README
 */
class HelloWorld
{
    public static void main(String args[])
    {
        System.out.println("Hello World!");
    }
}
```

Credits to Sean Hickey - https://medium.com/@webseanhickey/the-evolution-of-a-software-engineer-db854689243

So what?

- Programming is hard

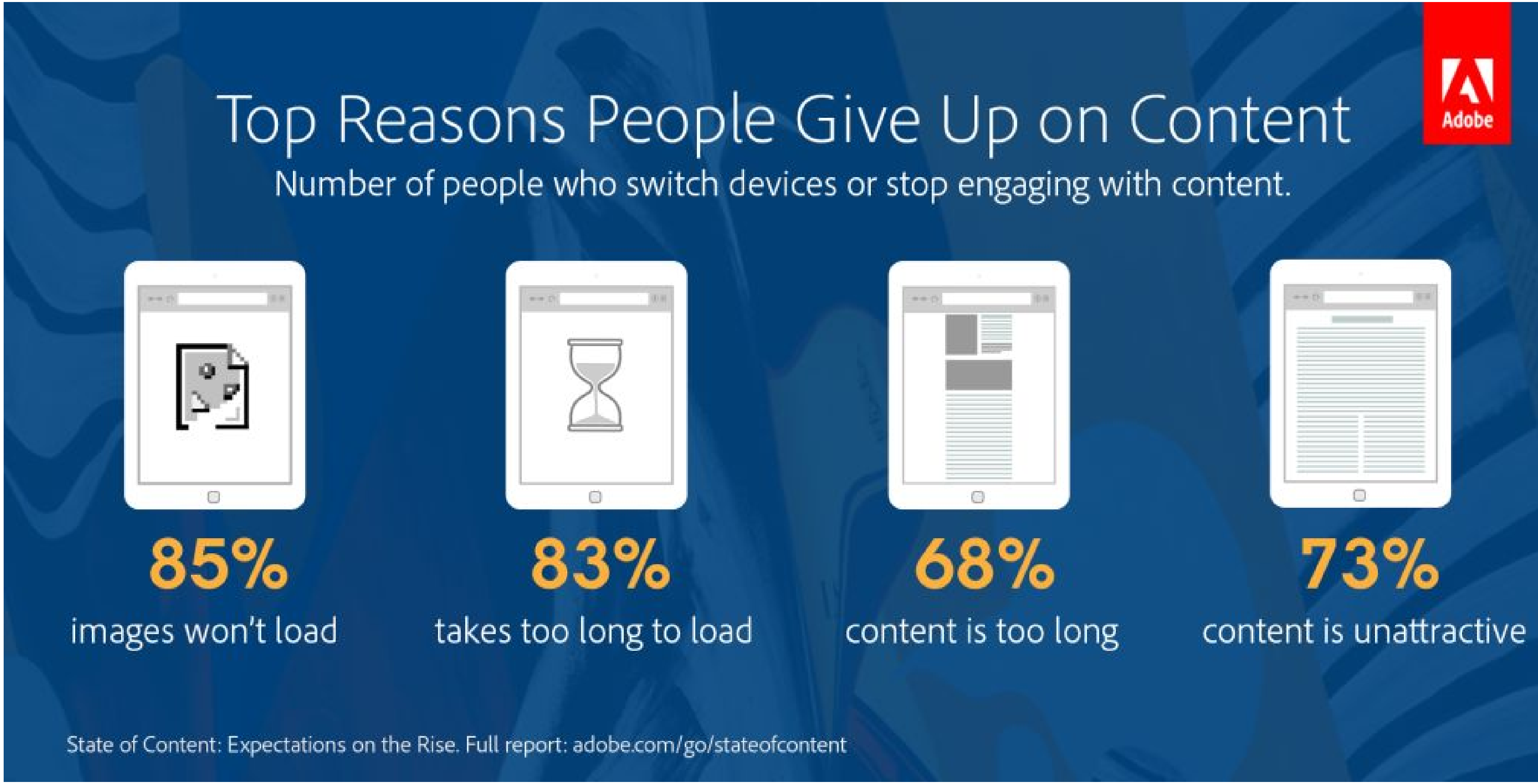


Explosion of Content

consumers use an average of **6** devices and consume an average of **12** sources of content

State of Content: Expectations on the Rise. Full report: adobe.com/go/stateofcontent

http://blogs.adobe.com/conversations/2015/10/at-max-content-velocity-meets-its-match.html?scid=social53499006&adbid=967586483284685&adbpl=fb&adbpr=341657335877606



Top Reasons People Give Up on Content

Number of people who switch devices or stop engaging with content.

85% images won't load

83% takes too long to load

68% content is too long

73% content is unattractive

State of Content: Expectations on the Rise. Full report: adobe.com/go/stateofcontent

So what?

- Programming is hard
- Takes time
- Talk about your design
- Be stubborn
  - do not be afraid of changing/refactoring your code
- Have some (a lot of) self-criticism
- The simpler the better
- Do not over-engineer if not needed
- Check how others write code (i.e. GitHub)
  - remember that most people write bad code!

Thanks! Questions?

Experts' Opinions - keeps updating - https://sic2.host.cs.st-andrews.ac.uk/aiq.html

Additional Slides - just for reference

School Handbook - https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/programming-style.html

Other bad examples

```
public void addChange(int change) {
    this.takings = takings + change;
}
```

could be

```
public void addChange(int change) {
    takings += change;
}
```

Other bad examples

```
 if (condition && condition2 && condition3 && condition4 && condition5 && condition6) {
}
```

could be

```
  if (condition && condition2 &&
      condition3 && condition4 &&
      condition5 && condition6) {
  }
```

Comments on static constants

Where you use an attribute as an absolute constant like MAX_ROWS it is usual to declare it **static** (so that only one copy is kept) and **final** (so that the compile knows it can't be overridden or changed. This allows the compiler to optimize better.”

Done Manifesto

1. There are three states of being. Not knowing, action and completion.
2. Accept that everything is a draft. It helps to get it done.
3. There is no editing stage.
4. Pretending you know what you're doing is almost the same as knowing what you are doing, so just accept that you know what you're doing even if you don't and do it.
5. Banish procrastination. If you wait more than a week to get an idea done, abandon it.
6. The point of being done is not to finish but to get other things done.
7. Once you're done you can throw it away.
8. **Laugh at perfection.** It's boring and keeps you from being done.
9. People without dirty hands are wrong. Doing something makes you right.
10. **Failure counts as done. So do mistakes**.
11. **Destruction is a variant of done.**
12. If you have an idea and publish it on the internet, that counts as a ghost of done.
13. Done is the engine of more.

Credits: http://www.brepettis.com/blog/2009/3/3/the-cult-of-done-manifesto.html

Common erros in Java - Refs

http://www.open.ac.uk/StudentWeb/m874/!synterr.htm

https://docs.oracle.com/javase/tutorial/getStarted/problems/