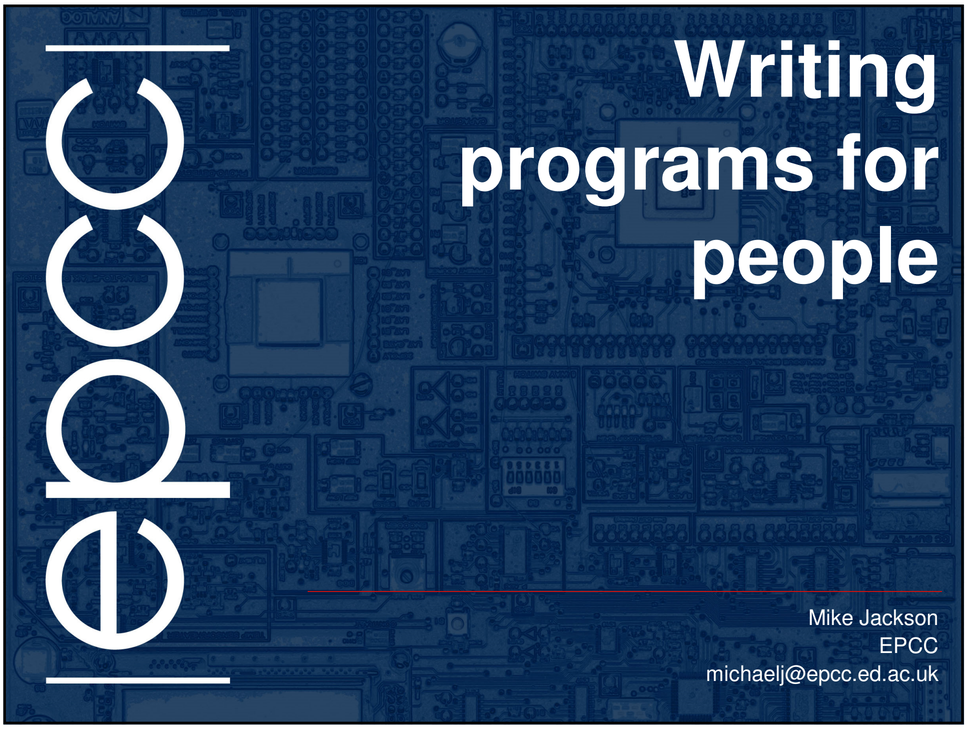
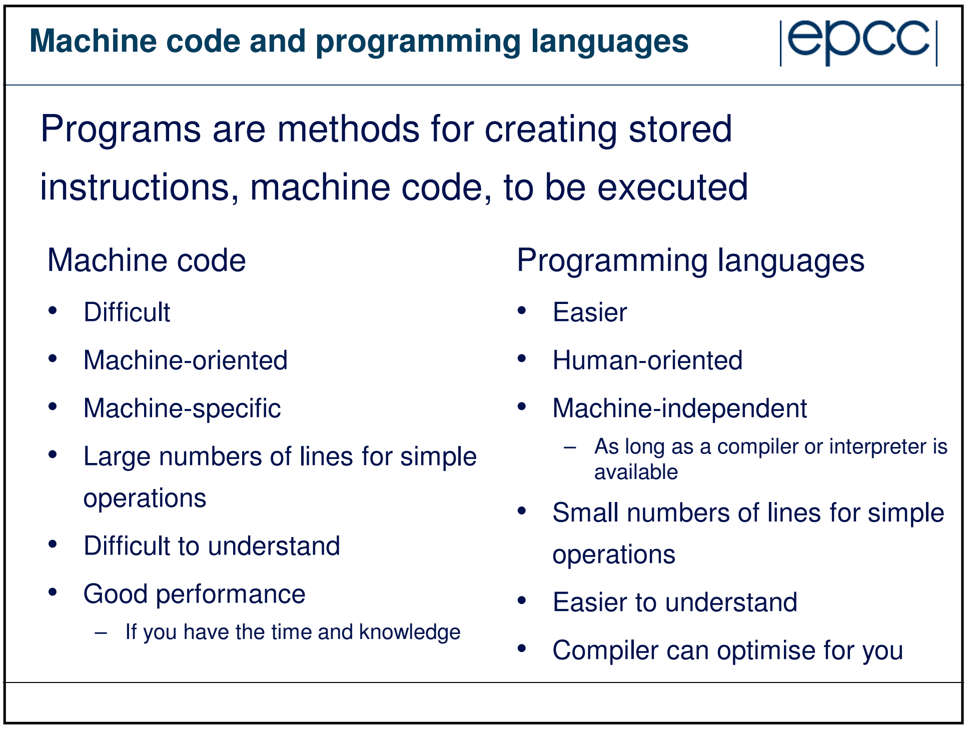


Writing programs for people

Mike Jackson
EPCC
michaelj@epcc.ed.ac.uk



Machine code and programming languages

Programs are methods for creating stored instructions, machine code, to be executed

Machine code

- Difficult
- Machine-oriented
- Machine-specific
- Large numbers of lines for simple operations
- Difficult to understand
- Good performance
  - If you have the time and knowledge

Programming languages

- Easier
- Human-oriented
- Machine-independent
  - As long as a compiler or interpreter is available
- Small numbers of lines for simple operations
- Easier to understand
- Compiler can optimise for you

The human factor

- “Regardless of whether one is dealing with assembly language or compiler language, the number of debugged lines of source code per day is about the same”
  - Corbató, F. J. “PL/I as a Tool for System Programming". Datamation”, 15(5), pp 68–76, May 1969.
- “The number of lines of code a programmer can write in a fixed period of time is the same independent of the language used”
  - Corbato’s Law
- “performance variability that derives from differences among programmers of the same language … is on average as large or larger than the variability found among the different languages.”
  - Prechelt, L. An Empirical Comparison of Seven Programming Languages, IEEE Computer, 33(10), pp23-29, October 2000.

Where does time go?

- What %age of software costs is spent on maintenance? 60
- Of that, what %age of maintenance is spent on?
  - Bug fixing 17
  - Adaptation to new platforms, dependencies, environments 23
  - Enhancements / new requirements 60
- Glass, R. “Facts and Fallacies of Software Engineering”, Addison-Wesley, 2002. Fact 41.

Where is the challenge?

- What is the most challenging aspect of maintenance?

	Development	Maintenance
– Defining and understanding	15	20
– Reviewing and tracing	30	20
– Implementing	20	20
– Testing and debugging	30	40
– Updating the documentation	5	

- Maintenance can be more difficult than development
- Glass, R. “Facts and Fallacies of Software Engineering”, Addison-Wesley, 2002. Fact 44.

Which code would you like to maintain?

```
int a, f; a = b & c ? d : e;
n[i] *= *m++ - k*l & i++ + ++p->j++;
```

```
    for (int i=0; i < member.length; i++)
    {
        if (member[i].isRetired())
        {
            sendInvitation(member[i]);
        }
    }
```

Why write code for humans?

- Code is compiled/interpreted and run by computer
- It is maintained by us
- Our time is (far) more valuable
- Readable code is easier to:
  - Maintain
  - Understand
  - Validate and trust
  - Trust
  - Reuse
- …both now and in the future
  - 6 months later when you spot an error in one of your thesis graphs
- Increase a project's "bus factor"

Readable code (before)

```
// Sum values in a file
String f = "data.txt";
int ac = 0;
int ec = 0;
BufferedReader br = new BufferedReader(new FileReader(f));
String l;
while ((l = br.readLine()) != null) {
if (l.startsWith("#-"))
// Split string on #- and parse part following #- into an integer
ec = Integer.parseInt(l.split("#-")[1]);
// Check if l starts with #
if (l.startsWith("#"));
// Check if l starts with D
else if (l.startsWith("D"));
// Increment count
else ac += 1;
}
br.close();
```

Readable code (after)

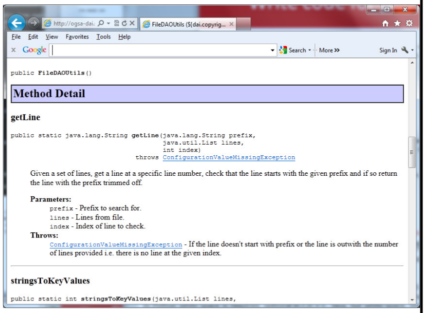
```
String filename = "data.txt";
int actual_line_count = 0;
int expected_line_count = 0;
BufferedReader br = new BufferedReader(new FileReader(filename));
String line;
// Count number of data records.
while ((line = br.readLine()) != null) {
    if (line.startsWith("#-"))
    {
        // Read number of records as recorded in file.
        expected_line_count = Integer.parseInt(line.split("#-")[1]);
    }
    if (line.startsWith("#"))
    {
        // Skip comments.
    }
    else if (line.startsWith("D"))
    {
        // Skip file description.
    }
    else
    {
        actual_line_count += 1;
    }
}
br.close();
```

- Names are self-documenting
- Indentation indicates structure
- Comments describe anything not clear from the code, why the code is as it is
- Coding standards/guidelines promote readable, and consistent, code

Application program interfaces (APIs)

- Document purpose, inputs, outputs, exceptions, error codes of packages, modules, classes, methods, functions
- How the component can be used by another bit of code

```
/**
 * Given a set of lines, get a line at a specific
 * line number, check that the line starts with
 * the given prefix and if so return the line with
 * the prefix trimmed off.
 *
 * @param prefix
 *     Prefix to search for.
 * @param lines
 *     Lines from file.
 * @param index
 *     Index of line to check.
 * @throws ConfigurationValueMissingException
 *     If the line doesn't start with prefix or the
 *     line is outwith the number of lines provided
 *     i.e. there is no line at the given index.
 */
public static String getLine(String prefix, List lines, int index)
    throws ConfigurationValueMissingException
    . . .
```


http://ogsa-dai.
FileDAOUtils (${dai.copyrig...
File Edit View Favorites Tools Help
Google
Search
More »
Sign In

```
public FileDAOUtils()
```

Method Detail

getLine

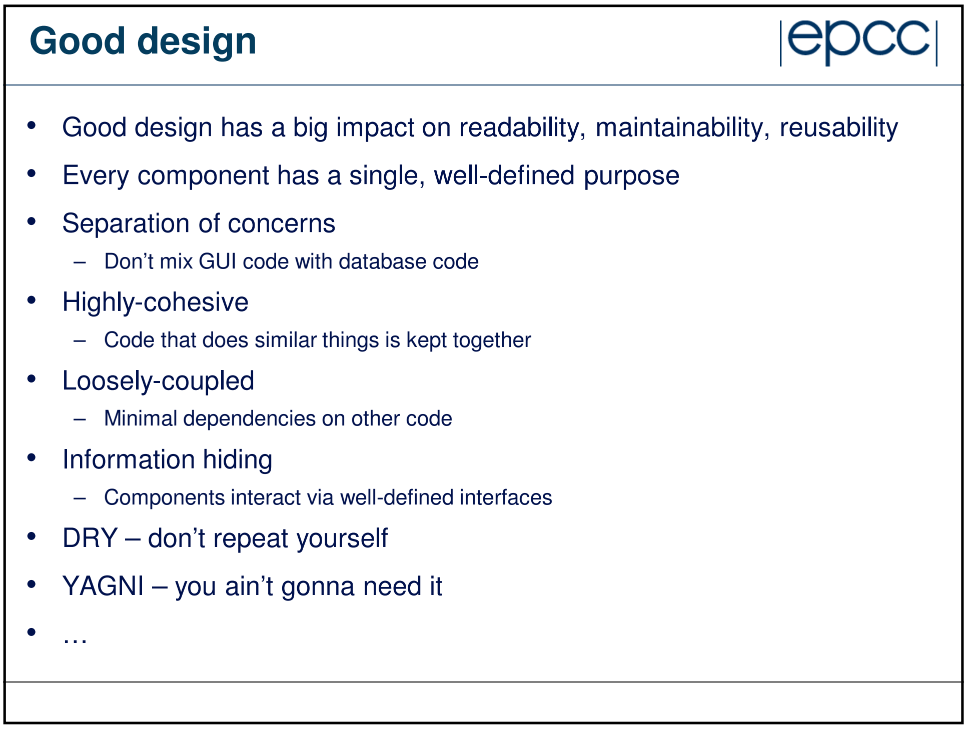
```
public static java.lang.String getLine(java.lang.String prefix,
                                       java.util.List lines,
                                       int index)
                                throws ConfigurationValueMissingException
```

Given a set of lines, get a line at a specific line number, check that the line starts with the given prefix and if so return the line with the prefix trimmed off.

Parameters:
`prefix` - Prefix to search for.
`lines` - Lines from file.
`index` - Index of line to check.
Throws:
`ConfigurationValueMissingException` - If the line doesn't start with prefix or the line is outwith the number of lines provided i.e. there is no line at the given index.

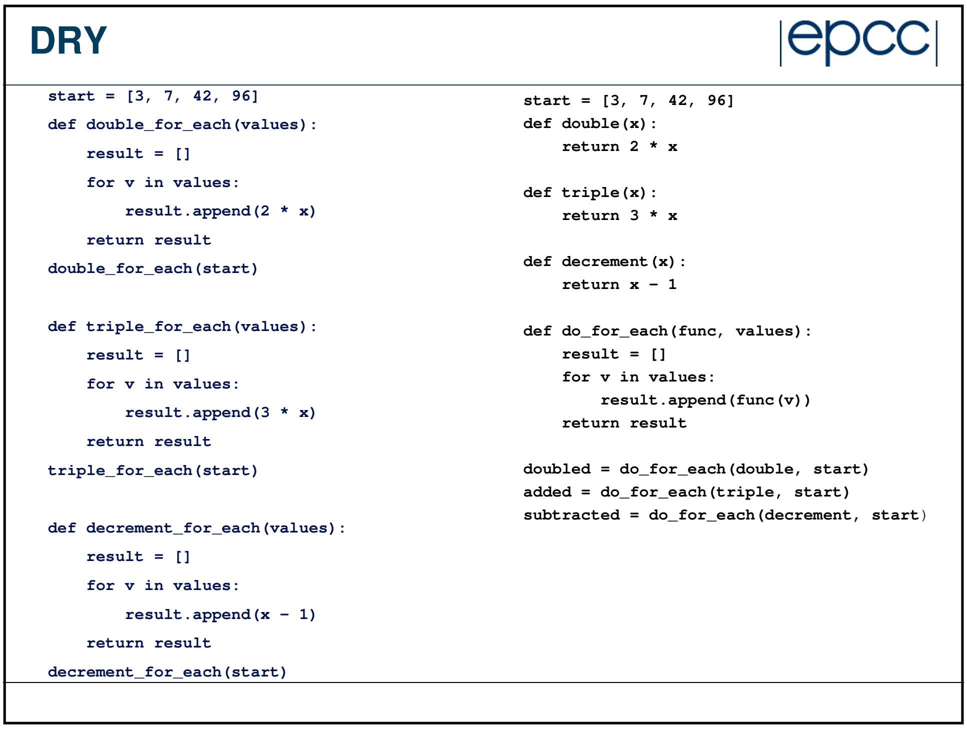
stringsToKeyValues

```
public static int stringsToKeyValues(java.util.List lines,
```



Good design

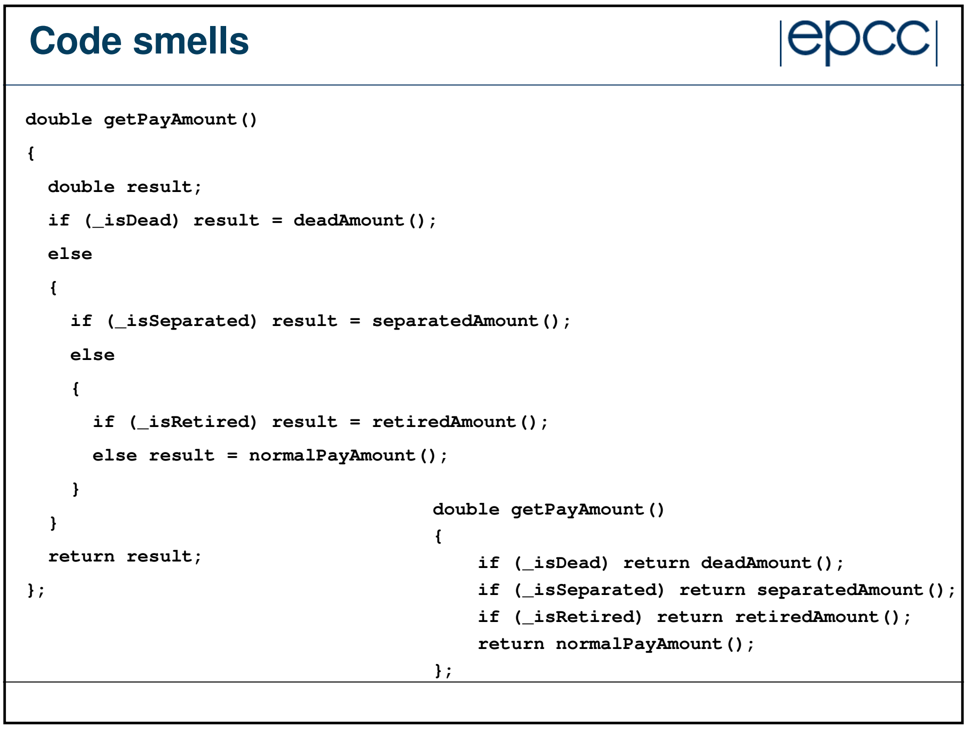
- Good design has a big impact on readability, maintainability, reusability
- Every component has a single, well-defined purpose
- Separation of concerns
  - Don't mix GUI code with database code
- Highly-cohesive
  - Code that does similar things is kept together
- Loosely-coupled
  - Minimal dependencies on other code
- Information hiding
  - Components interact via well-defined interfaces
- DRY – don't repeat yourself
- YAGNI – you ain't gonna need it
- …



DRY

```
start = [3, 7, 42, 96]
def double_for_each(values):
    result = []
    for v in values:
        result.append(2 * x)
    return result
double_for_each(start)

def triple_for_each(values):
    result = []
    for v in values:
        result.append(3 * x)
    return result
triple_for_each(start)

def decrement_for_each(values):
    result = []
    for v in values:
        result.append(x - 1)
    return result
decrement_for_each(start)
```

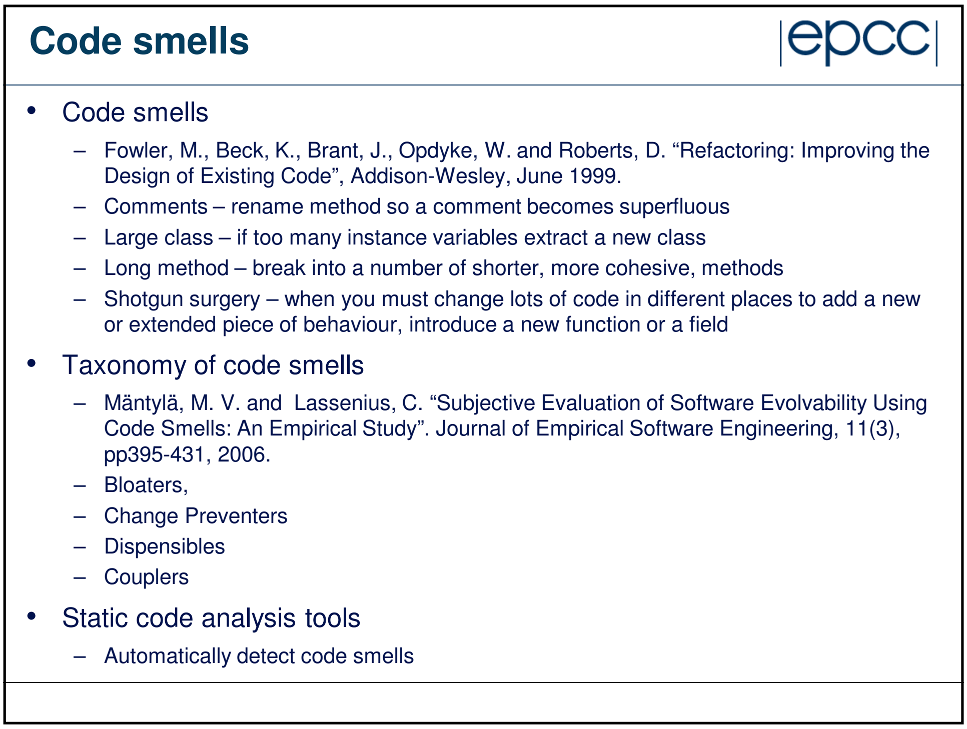
```
start = [3, 7, 42, 96]
def double(x):
    return 2 * x

def triple(x):
    return 3 * x

def decrement(x):
    return x - 1

def do_for_each(func, values):
    result = []
    for v in values:
        result.append(func(v))
    return result

doubled = do_for_each(double, start)
added = do_for_each(triple, start)
subtracted = do_for_each(decrement, start)
```



Code smells

```
double getPayAmount()
{
  double result;
  if (_isDead) result = deadAmount();
  else
  {
    if (_isSeparated) result = separatedAmount();
    else
    {
      if (_isRetired) result = retiredAmount();
      else result = normalPayAmount();
    }
  }
  return result;
};
```

```
double getPayAmount()
{
    if (_isDead) return deadAmount();
    if (_isSeparated) return separatedAmount();
    if (_isRetired) return retiredAmount();
    return normalPayAmount();
};
```



Code smells

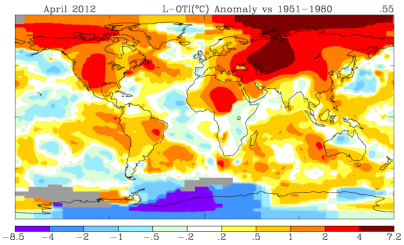
- Code smells
  - Fowler, M., Beck, K., Brant, J., Opdyke, W. and Roberts, D. “Refactoring: Improving the Design of Existing Code”, Addison-Wesley, June 1999.
  - Comments – rename method so a comment becomes superfluous
  - Large class – if too many instance variables extract a new class
  - Long method – break into a number of shorter, more cohesive, methods
  - Shotgun surgery – when you must change lots of code in different places to add a new or extended piece of behaviour, introduce a new function or a field
- Taxonomy of code smells
  - Mäntylä, M. V. and Lassenius, C. “Subjective Evaluation of Software Evolvability Using Code Smells: An Empirical Study”. Journal of Empirical Software Engineering, 11(3), pp395-431, 2006.
  - Bloaters,
  - Change Preventers
  - Dispensibles
  - Couplers
- Static code analysis tools
  - Automatically detect code smells

A little bit of documentation goes a long way

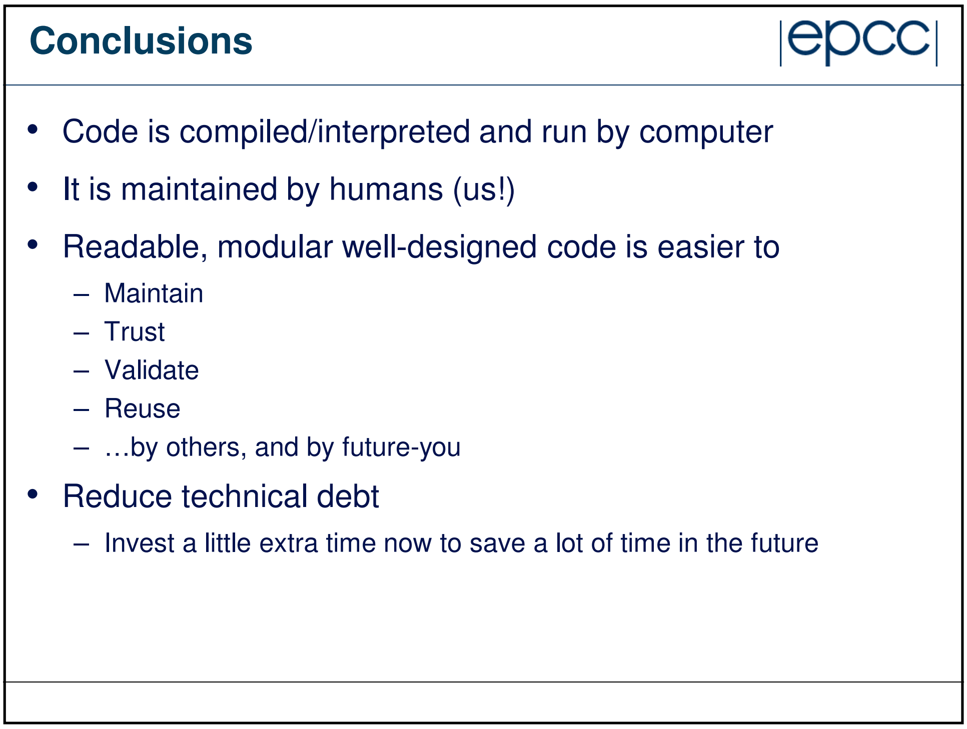
- Types of documentation
  - What the code does
  - How the code does it
  - **How to use (build and run) it**
- 10 minute quick start guide
  - How someone can use your software (on ~~your~~ their data)
- How to set up a development environment
  - What packages, libraries and tools are needed
  - In-house and open source projects
- Precision in all things
  - Does “Python” mean Python 2 or Python 3?
  - Does “Linux” mean Scientific Linux 7 or Ubuntu?

Why write code for humans?

- GISS Surface Temperature Analysis
  - http://data.giss.nasa.gov/gistemp/
- Climate sceptics ask “Where’s the source code?”
- Release the source code
  - “Obvious bugs”
  - “Incomplete”
  - “This can’t be the actual code!”



- Rewrite
  - http://code.google.com/p/ccc-gistemp/
  - “increase public confidence in climate science results”
  - Barnes, N. and Jones, D. Clear Climate Code: Rewriting Legacy Science Software for Clarity. IEEE Software 28(6), pp36-42, November 2011.



Conclusions

- Code is compiled/interpreted and run by computer
- It is maintained by humans (us!)
- Readable, modular well-designed code is easier to
  - Maintain
  - Trust
  - Validate
  - Reuse
  - …by others, and by future-you
- Reduce technical debt
  - Invest a little extra time now to save a lot of time in the future