Good Programming Practices

Thomas Dauser, Maximilian Lorenz, Jakob Stierhof, Philipp Weber

@ Remeis Zoom Meeting, 11.12.2020

Agenda

- Why does “Good Code” matter?
- Basic Concepts towards writing “Good Code”
- Basic Concepts of Software/Script Design
- Literature and further reading

- Real-life Examples
  - code snippets
  - the isisscripts
  - gitlab & documentation
- Discussion

Purpose of the workshop

- Convince you that Good Code
  - … is important, also as Astrophysicists
  - … is (very likely) not what you are currently producing
  - … is hard to write and needs practice
  - … can be written and maintained by a simple set of standards and rules
- Give you a basic understanding on how to write Better Code
- Motivate you that it is worth spending time on producing it Good Code
- Discuss the applicability of the concepts to our problems/projects

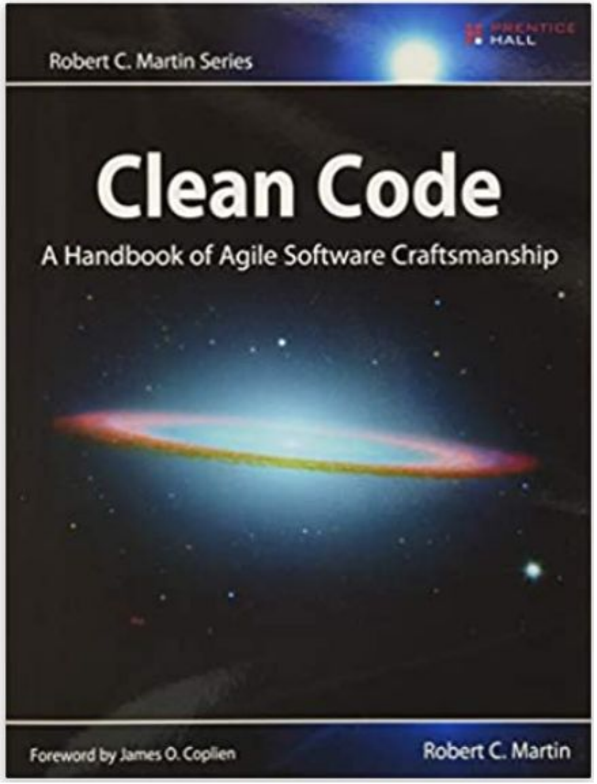
We will NOT discuss implementation / programming language specific solutions

Bad Code

Robert C. Martin:

“*Have you ever been significantly impeded by bad code? If you are a programmer of any experience then you’ve felt this impediment many times. Indeed, we have a name for it. We call it wading. We wade through bad code. We slog through a morass of tangled brambles and hidden pitfalls. We struggle to find our way, hoping for some hint, some clue, of what is going on; but all we see is more and more senseless code.*

Of course you have been impeded by bad code. So then—why did you write it?”


Robert C. Martin Series
PRENTICE HALL
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin

Bad Code: The Total Cost of Owning a Mess

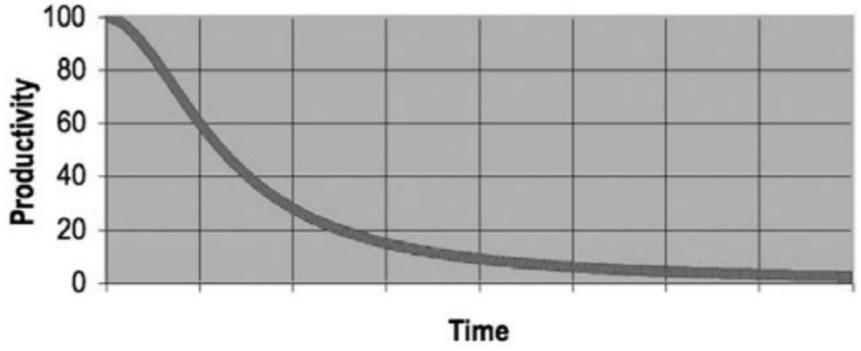


Figure 1-1
Productivity vs. time

Robert C. Martin:

"*Have you ever waded through a mess so grave that it took weeks to do what should have taken hours? Have you seen what should have been a one-line change, made instead in hundreds of different modules? [...] Why does good code rot so quickly into bad code?*

But the fault [...] is not in our stars, but in ourselves. We are unprofessional."

Why does “Good Code” matter?

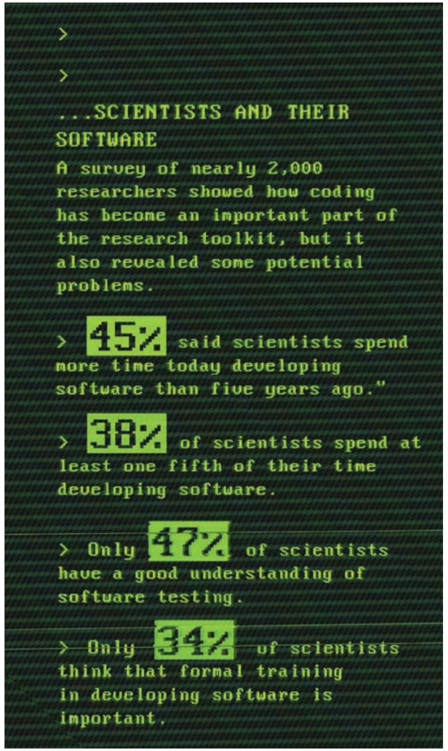
We are Astrophysicists!

So, as we (most of us) do not write production code, does it actually matter to produce clean code?

Questions:

- What do you think?
- What are the problems you solve with scripts / programs?
- Have you been stuck in an old code, not understanding what it does any more?

Programming in Science



>

>

...SCIENTISTS AND THEIR SOFTWARE

A survey of nearly 2,000 researchers showed how coding has become an important part of the research toolkit, but it also revealed some potential problems.

> 45% said scientists spend more time today developing software than five years ago."

> 38% of scientists spend at least one fifth of their time developing software.

> Only 47% of scientists have a good understanding of software testing.

> Only 34% of scientists think that formal training in developing software is important.

Merali, Nature, 2010

- programming has become a **fundamental tool** for science
- it is a **craftsmanship that needs to be studied and requires practice**: simply knowing the syntax is not enough

if you are already spending time on it, you might as well do it well (and save yourself and others a lot of trouble)

big promise: it will be more fun!

Importance of Clean Code

- The ONLY way to keep your code maintainable!
- Problems become easier to solve: trying to write clean code is a first step to solve the actual problem
- Speed up your research
  - make your own code re-usable (already applies to any plotting)
  - will save countless hours of debugging and frustration
- making code accessible and sharing it publicly will encourage others to use it and not waste their time reinventing the wheel (isisscripts)

Importance of Clean Code for Scripting

… now you might ask: “does this apply to me, if I’m only writing data analysis script?”

- applies to everything larger than a few lines of code
- applies to any data analysis script you would like to understand weeks, months, years (?) later
- applies to any script someone else might eventually read

➔ a data analysis script is also a record/journal of how you came up with the results

➔ clean code does not require a complex usage of classes, hierarchies

What are the main aspects of writing Clean Code?

A word of Caution...

the following set of guidelines

- are not meant to be followed religiously
- might only partly apply to your specific problem, but are a good starting point to improve the quality of any script/code
- requires practice and critical reflection/discussions to improve

A word of advice: have a look at the literature and read it. Then write code, read some more and improve your code. Talk to your colleges. Iterate …

Clean Code: Simplified Summary

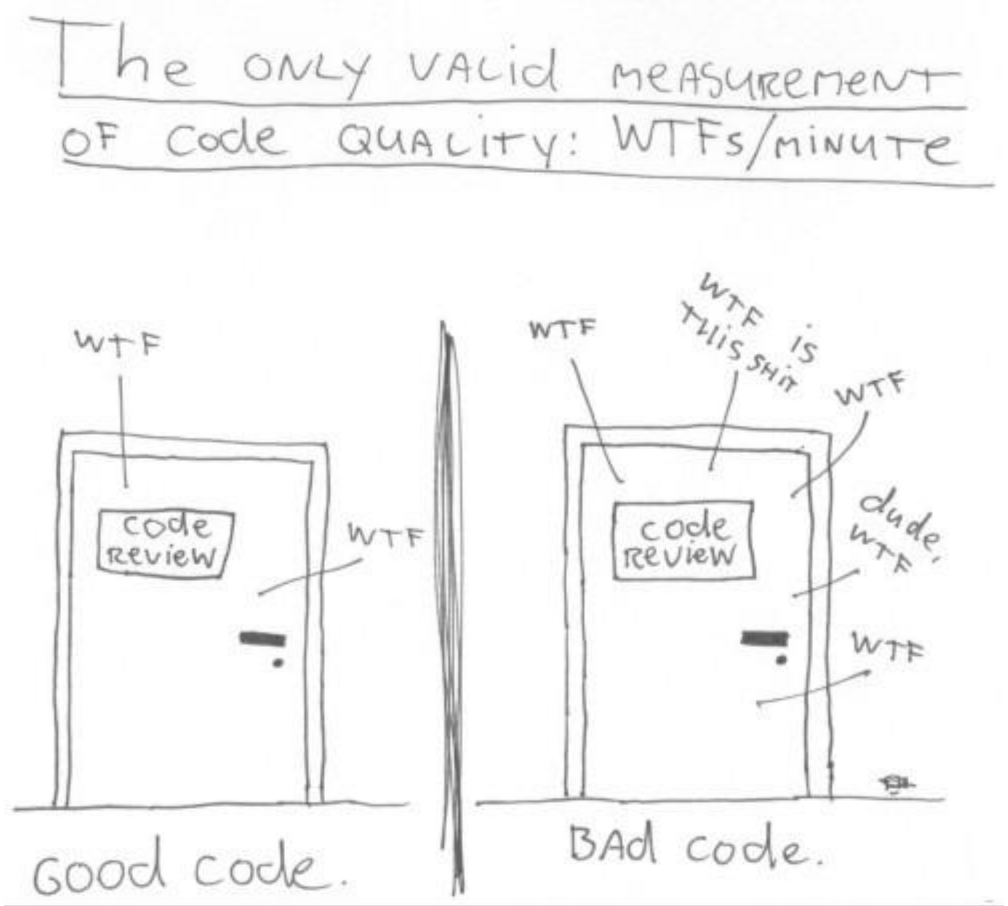
1. contains **no duplication**
2. **minimizes** the **complexity** (length) and number of functions/classes
3. **reveals** its full **intention** when being read

We are Authors!

- Most of the time programming is spend reading (and thinking)
- Only 1/10 is spent actually writing

So, wouldn’t making it easier to read actually make it easier to write?

Consider programming like writing literature. Simply putting words on a sheet of paper is simple, but writing a story people enjoy reading takes effort.


The only valid measurement of code quality: WTFs/minute
WTF
code review
WTF
WTF
WTF is this shit
WTF
code review
dude, WTF
WTF
Good code.
BAd code.

Naming

```
variable d; // elapsed time in days

// ... some code .......... //

if (d > 284) {
     do_something_cool();
}
```

```
variable daysSinceLaunch;

// ... some code .......... //
variable day_erass1_complete = 284;
if (daysSinceLaunch > day_erass1_complete) {
       do_something_cool();
}
```

Naming

- Use Intention-Revealing Names

```
variable daysSinceLaunch;
```

instead of

```
variable d; % days since launch
```

- Avoid disinformation: the name suggests one thing, but the function does something else

Naming

- Make Meaningful Distinctions

```
getLaunchDate();

getLaunchDays();

retrieveLaunchDay();   %%% ???????
```

- Add meaningful context

```
getLaunchDate();

setLaunchDate();
```

Naming

Use Pronounceable and Searchable Names

- do not just use short names

```
variable a, a1, a_new, …

define b(){ … };
```

- prefer longer, understandable names

`variable generation_date;` instead of `variable genymdhms;`

Naming: Summary

- ➔ Proper naming takes time and effort, but pays off
- ➔ Finding a fitting name helps thinking about what you want to do
- ➔ Requires constant changes to the code (refactoring)

need to be able to constantly update your code → use Search-and-Replace or a proper IDE to easily rename variables

Comments

What is the purpose of a comment?

- ➔ To explain the purpose of code if the code cannot explain its own purpose.
- ➔ Every use of a comment represents a failure.
- ➔ Write code that explains itself.
- ➔ Comment as last resort.

Comments

- **Comments lie**
- Not always, not intentionally.
- As code changes and evolves, comments silently rot and migrate.
- Inaccurate comments are worse than no comments at all.

Typical situation with confusing, messy code you hardly understand yourself:
“Ooh, I’d better comment that” → **“No! You better clean it!”**

Comments

- **Explain yourself in code**

```
// Check to see if the employee is eligible for full benefits
if ((employee.flags & HOURLY_FLAG) && (employee.age > 65))
```

```
if (employee.isEligibleForFullBenefits())
```

Good Comments

- **Legal comments** (copyright, authorship statements, etc.)
- **Informative comments**

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

- **Explanation of intent** (*why*, not *what*)

```
// Loop backwards through all elements (they should be processed
// chronologically).

// Need a stable sort (the performance does not matter).
```

- **TODO Comments** (OK during development, delete before check-in)
- **Documentation comments** (doxygen, Javadoc, etc. - but don’t overdo it)

Bad Comments

- **Redundant comments** and **noise comments**

```
// Initialize the days since the launch
int daysSinceLaunch = 0;

// Increment daysSinceLaunch
daysSinceLaunch++;
```

- **Journal comments** (use version control systems)
- **Closing Brace Comments**

```
    } // END of loop over all neighbors
} // END of loop over all elements
```

- **Commented-out code** (should never be checked-in!)
- **Nonlocal information** (comment should only describe code it appears near)

Comments

“Good code is self-documenting.”

“Don’t comment the code, clean the code.”

- Should be your goal.
- Of course not always possible in real world.
- Only add a comment if you can’t make the code simpler.
- Easy action: Replace a commented code piece with a function.

Don’t repeat yourself (DRY)

- **try not do have any code repetition**
- goal: if you have to change one thing, you only have to change it in one place
- should be applied at all scales: variables, functions, classes, scripts, … (did you ever manage to end up with two similar but not identical scripts?)

strategy: if possible, publish your code and re-use existing code that is known to work (isisscripts)

Functions (and Classes)

- keep them short
- keep them even shorter!

Naming:

- Classes / Objects: use nouns

  `Satellite, Detector, Dataset, Plot`
- Functions: use verbs (they do something)

  `getLaunchDate();` …

Functions (and Classes)

1. Should do **one thing only** (and do it well)
2. Should have **no side effects** (already conflicts with rule 1)
3. Should be short and **even shorter**

Do NOT create a God-Function

or a God-SLang/ISIS script

Scopes: Local and Global Variables

Variables are visible in the scope (e.g., function) where they are defined

- define variables as close as possible to where you need them
- use local variables (only defined inside the function)
- avoid using global variables!

SLang/isis: use namespaces

Example: variables/simple_function_bare.sl

Data Structures and Classes

Classes/Objects

hide data behind abstractions and expose functions that operate on this data

```
variable Plot = xfig_plot_new();
```

Data Structures

expose their data and have no meaningful functions

```
variable data_struct =
              get_data_counts(1);
```

understand the language you are using

Basic Software Development:

The (slightly) Bigger Structure

Basic Software Engineering Principles

1. Modularity
2. Documentation
3. Testing
4. Version-Control

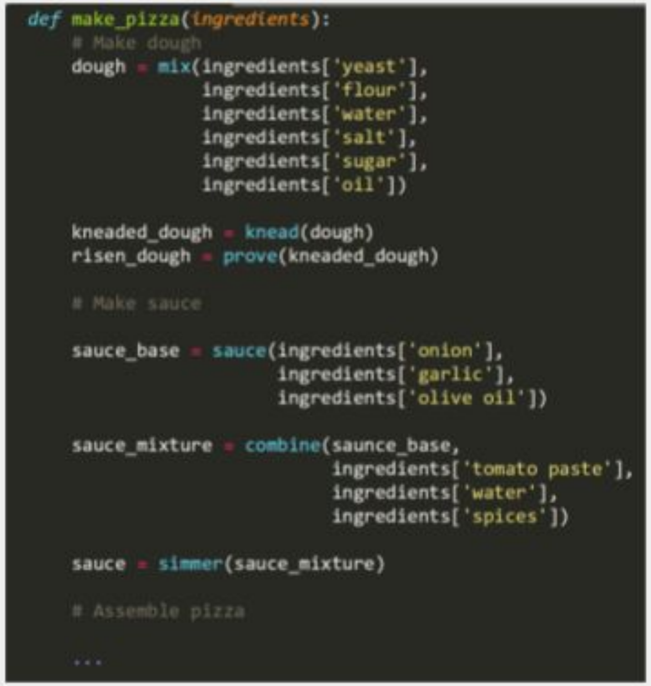
https://astrobites.org/2020/10/23/towards-better-research-code-and-software/

Modularity

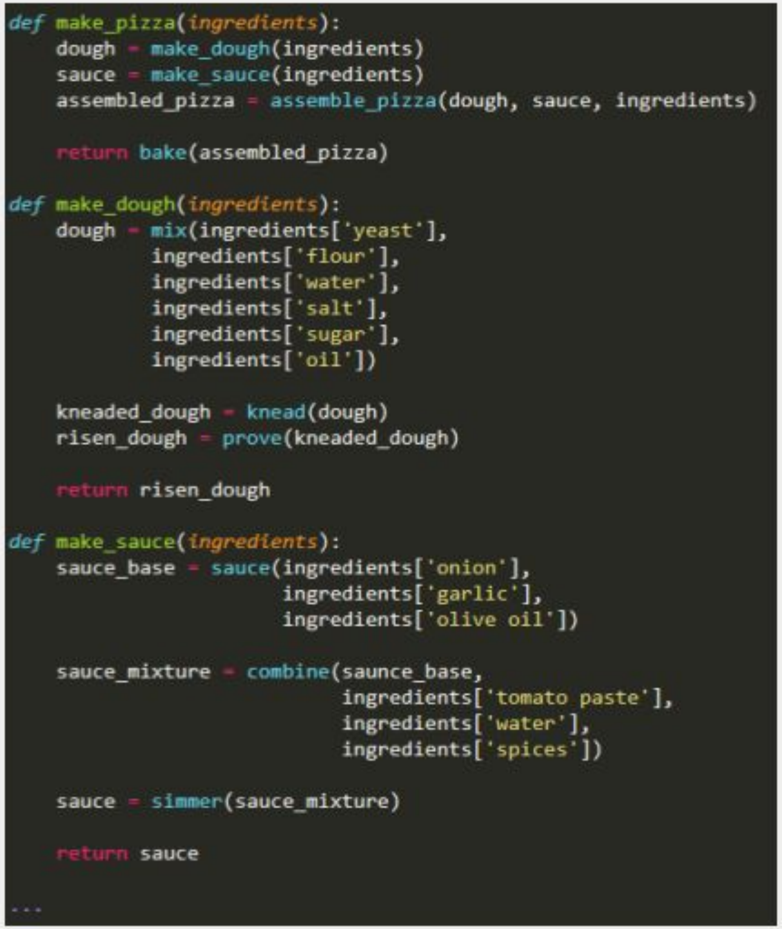
Making a code modular means dividing it into small functional units, encapsulating complexity and removing duplication:

1. It’s much more human readable
2. The code can be fixed easily when it breaks (otherwise we will have to make the corrections to every copy of the code)
3. The code can be easily used in another project
4.

https://astrobites.org/2020/10/23/towards-better-research-code-and-software/



```python
def make_pizza(ingredients):
    # Make dough
    dough = mix(ingredients['yeast'],
                ingredients['flour'],
                ingredients['water'],
                ingredients['salt'],
                ingredients['sugar'],
                ingredients['oil'])

    kneaded_dough = knead(dough)
    risen_dough = prove(kneaded_dough)

    # Make sauce

    sauce_base = sauce(ingredients['onion'],
                       ingredients['garlic'],
                       ingredients['olive oil'])

    sauce_mixture = combine(saunce_base,
                            ingredients['tomato paste'],
                            ingredients['water'],
                            ingredients['spices'])

    sauce = simmer(sauce_mixture)

    # Assemble pizza

    ...
```



```python
def make_pizza(ingredients):
    dough = make_dough(ingredients)
    sauce = make_sauce(ingredients)
    assembled_pizza = assemble_pizza(dough, sauce, ingredients)

    return bake(assembled_pizza)

def make_dough(ingredients):
    dough = mix(ingredients['yeast'],
                ingredients['flour'],
                ingredients['water'],
                ingredients['salt'],
                ingredients['sugar'],
                ingredients['oil'])

    kneaded_dough = knead(dough)
    risen_dough = prove(kneaded_dough)

    return risen_dough

def make_sauce(ingredients):
    sauce_base = sauce(ingredients['onion'],
                       ingredients['garlic'],
                       ingredients['olive oil'])

    sauce_mixture = combine(saunce_base,
                            ingredients['tomato paste'],
                            ingredients['water'],
                            ingredients['spices'])

    sauce = simmer(sauce_mixture)

    return sauce

...
```

https://astrobites.org/2020/10/23/towards-better-research-code-and-software/

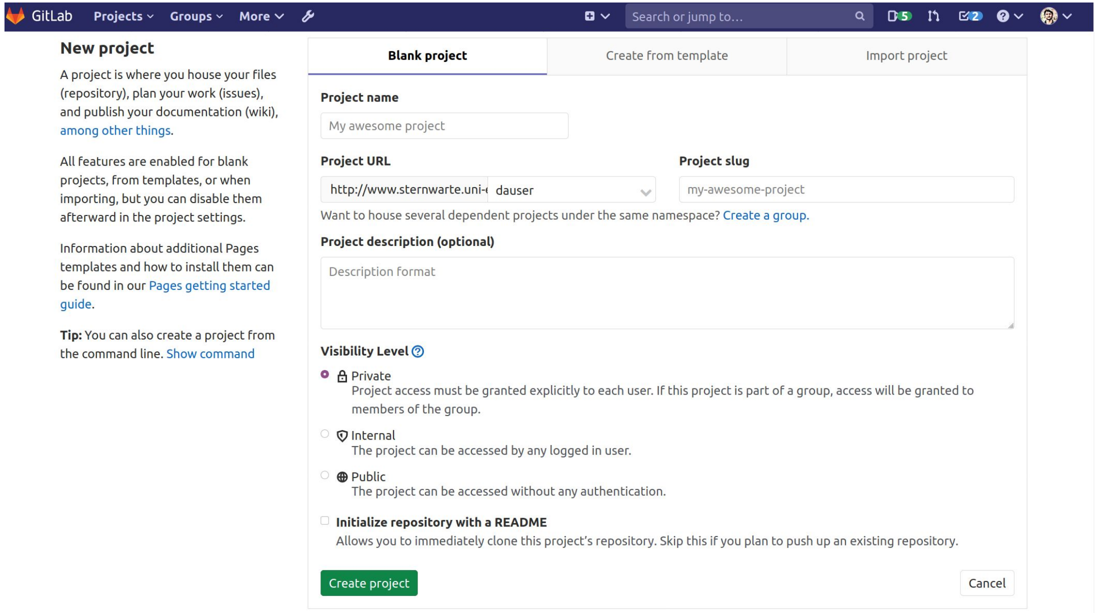
Documentation

1. enables other people to use and change (and potentially improve) your code
2. ensures you write code you understand yourself

see ***isisscripts*** *example*

Version Control

- we have git and gitlab installed
- use it, there is no excuse ...



New project

A project is where you house your files (repository), plan your work (issues), and publish your documentation (wiki), among other things.

All features are enabled for blank projects, from templates, or when importing, but you can disable them afterward in the project settings.

Information about additional Pages templates and how to install them can be found in our Pages getting started guide.

Tip: You can also create a project from the command line. Show command

Blank project	Create from template	Import project

Project name

My awesome project

Project URL

http://www.sternwarte.uni- dauser

Project slug

my-awesome-project

Want to house several dependent projects under the same namespace? Create a group.

Project description (optional)

Description format

Visibility Level

- [✓ Private] Project access must be granted explicitly to each user. If this project is part of a group, access will be granted to members of the group.
- Internal The project can be accessed by any logged in user.
- Public The project can be accessed without any authentication.

☐ **Initialize repository with a README**
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project | Cancel

Re-visiting code: the “Boy Scout Rule”

- “leave the camp cleaner than you found it”
- improve the code every time you look at it, even if it is just a small change (e.g., re-name a variable, add some documentation; isisscripts!)

code would get better over time and not rot, how good is this??

Testing

could spend hours talking about it...

idea: usually when scripting/programming you have to write some sort of **tests** to verify the functionality.

➔ **Why not write it explicitly and run it (automatically)?**

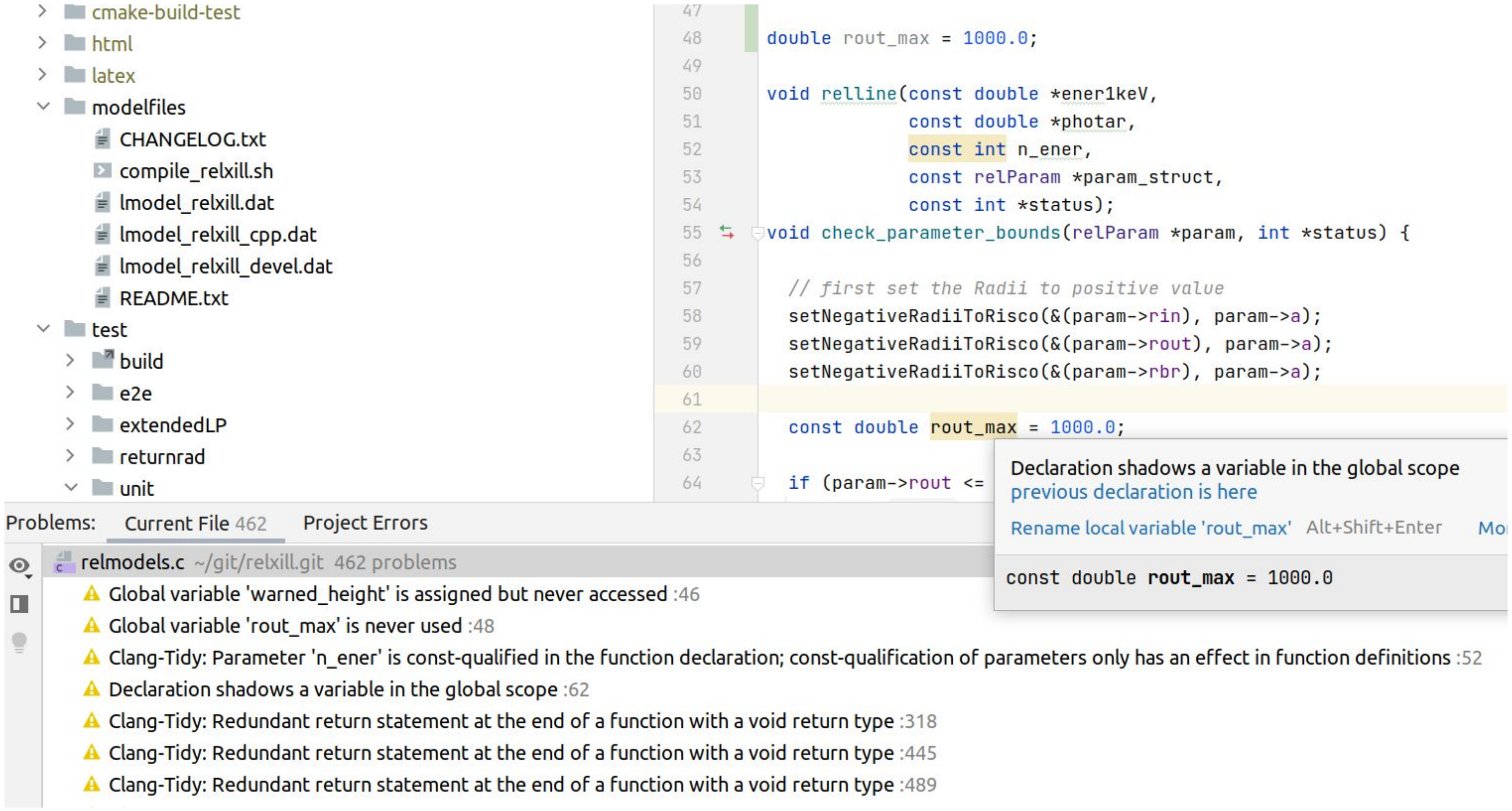
Linting and IDEs

Static Code Analysis: Tools will discover problems with your code, partly even while you type (unfortunately not available for SLang)

IDEs containing these tools (and much more) automatically:

- python: pycharm
- C/C++: CLion
- ...

Linting and IDEs



cmake-build-test
html
latex
modelfiles
CHANGELOG.txt
compile_relxill.sh
lmodel_relxill.dat
lmodel_relxill_cpp.dat
lmodel_relxill_devel.dat
README.txt
test
build
e2e
extendedLP
returnrad
unit

```
double rout_max = 1000.0;

void relline(const double *ener1keV,
             const double *photar,
             const int n_ener,
             const relParam *param_struct,
             const int *status);
void check_parameter_bounds(relParam *param, int *status) {

  // first set the Radii to positive value
  setNegativeRadiiToRisco(&(param->rin), param->a);
  setNegativeRadiiToRisco(&(param->rout), param->a);
  setNegativeRadiiToRisco(&(param->rbr), param->a);

  const double rout_max = 1000.0;

  if (param->rout <=
```

Declaration shadows a variable in the global scope
previous declaration is here
Rename local variable 'rout_max' Alt+Shift+Enter Mo

```
const double rout_max = 1000.0
```

Problems: Current File 462 Project Errors

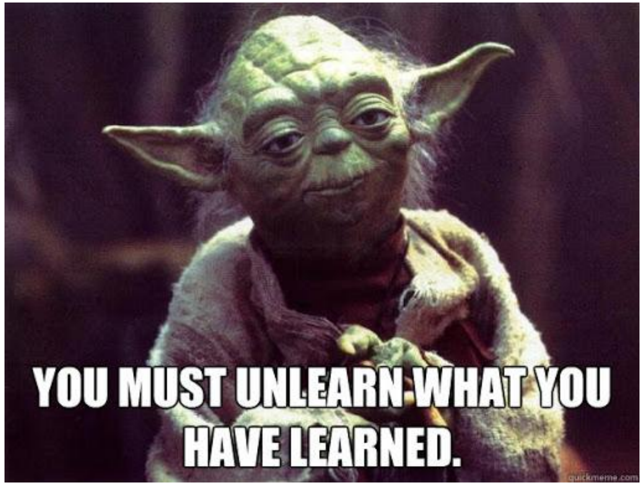
relmodels.c ~/git/relxill.git 462 problems
- Global variable 'warned_height' is assigned but never accessed :46
- Global variable 'rout_max' is never used :48
- Clang-Tidy: Parameter 'n_ener' is const-qualified in the function declaration; const-qualification of parameters only has an effect in function definitions :52
- Declaration shadows a variable in the global scope :62
- Clang-Tidy: Redundant return statement at the end of a function with a void return type :318
- Clang-Tidy: Redundant return statement at the end of a function with a void return type :445
- Clang-Tidy: Redundant return statement at the end of a function with a void return type :489

Last words of advice: Use the right tools for the right problems

- before starting any new (even little) task, take some time to think about the approach to a problem and the available toolset you have
- consider looking in the isisscripts (or on the web for other languages), part of your problem might have already been solved
  - When doing so, improve the code and add better help, while you're at it
- larger projects: choose the programming language(s) wisely!
  - most languages might solve your problem, but there is likely one which is best suited

Last words of advice: Learning

- get advice: ask your colleagues (discuss tools, usage of programming language, good practices similar to)
- look at the code of masters
  - how is code in my language X written by knowledgeable coders / my experienced colleague
- Retire old ideas and learn new ones
  - **Languages change, concepts change**
  - Things you learnt some years ago might be considered bad practice today
  - Try to adjust yourself to new standards (linters can help)



Summary: Clean Code

1. contains **no duplication**
2. **minimizes** the **complexity** (length) and number of functions/classes
3. **reveals** its full **intention** when being read

- Naming: Intention Revealing
- Functions: Single Responsibility → should do one thing and be short
- Comments: used sparsely, a necessary evil

Read: *“Clean Code” by Robert C. Martin*

Summary: The Bigger Picture

1. be prepared for change: invest time to adapt your code/script to changes (otherwise it will rot and be unusable)
2. documentation (for others and yourself)
3. collaborate: share your code, use existing functions, and discuss
4. programming develops: have an open mind for changes

Programming has become THE most important tool for us

➔ Spend time to study and improve your skills

➔ Well maintained scripts/code increase the science output

Quotes

Martin Fowler: *“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”*

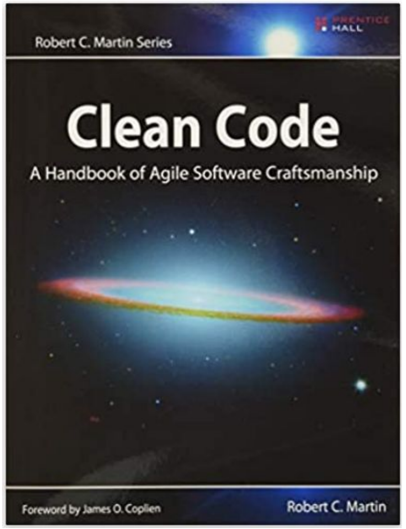
Tim Peters, The Zen of Python: *“Beautiful is better than ugly. Explicit is better than implicit. Simple is better than complex. Complex is better than complicated. Flat is better than nested. Sparse is better than dense. Readability counts.“*

Brian W. Kernighan: *“Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?”*

The Perl programming motto (modern version): *“There is more than one way to do it -- but sometimes consistency is not a bad thing either.”*

Literature

- Short, nice write-up of basic concepts: https://www.makeuseof.com/tag/10-tips-writing-cleaner-better-code/
- Clean Code: A Handbook of Agile Software Craftsmanship (Robert C. Martin)
- Youtube:
    - Breaking dependencies with SOLID https://www.youtube.com/watch?v=Ntraj80qN2k
    - …
    - Basically every talk by Kevlin Henney
- Books on the language you use
- NOT stackoverflow


Robert C. Martin Series
PRENTICE HALL
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin

Discussion: Tribal Knowledge

- Undocumented things which are just ‘common knowledge’
  - Sign for incredibly bad documentation
  - See isisscripts: You don’t know about a lot of undocumented functions, unless somebody told you about them
  - ⇒ slxfig

Example: The Isisscripts

- General use functions
- Modularity
- Documentation
- (Tests)

See presentation by Jakob

Example: Some Code Snippets

- script/plot_paper_spectra.sl