



Programming

for Today's Physicists and Engineers

ISOTDAQ 2020

University of Valencia, Valencia

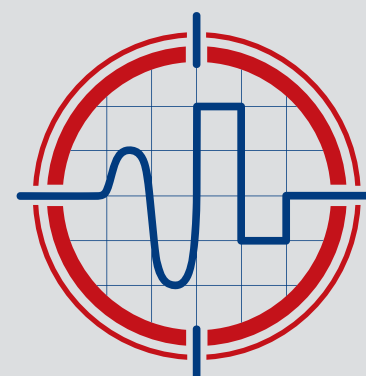
January 13, 2020

Alessandro Thea

Rutherford Appleton Laboratory - PPD



Science and
Technology
Facilities Council



ISOTDAQ

International School of Trigger
and Data Acquisition



How to survive

Programming

for Today's Physicists and Engineers

ISOTDAQ 2020

University of Valencia, Valencia

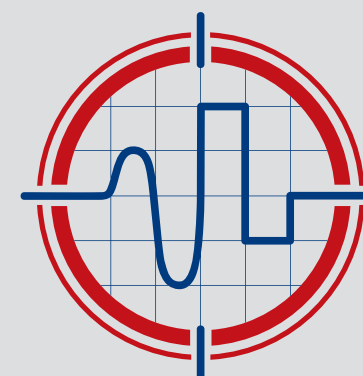
January 13, 2020

Alessandro Thea

Rutherford Appleton Laboratory - PPD



Science and
Technology
Facilities Council



ISOTDAQ

International School of Trigger
and Data Acquisition

Opening words

Disclaimer: This is more a collection of pointers* than a tutorial, it's a starting point...

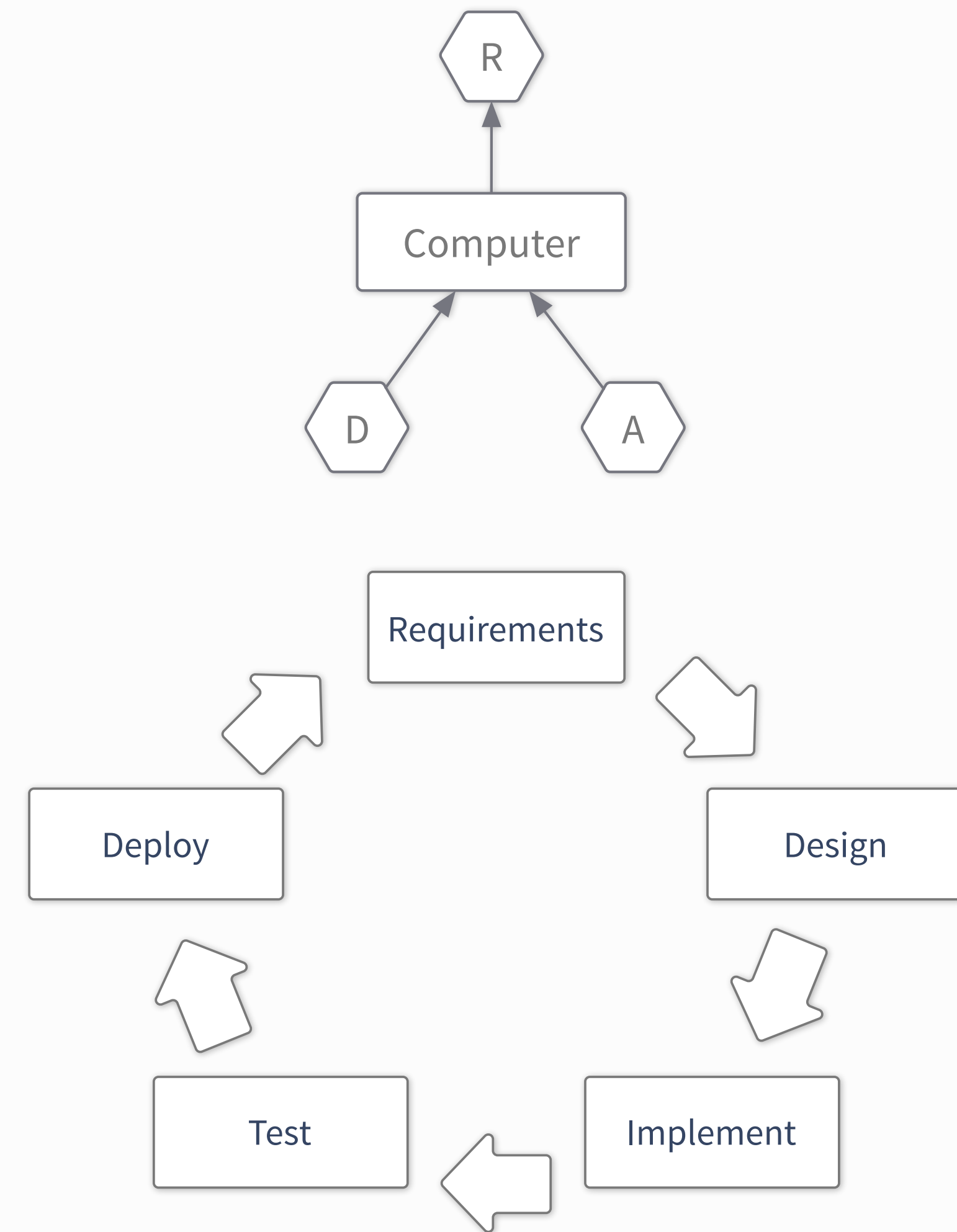
(Almost) no code but a bias towards C++ and Python

Note: While the lecture focus is software, most of the content equally applies to firmware programming.

Acknowledgment: Slides are based on previous lectures by Joschka Poettgen (Lingemann) and Erkcan Ozcan

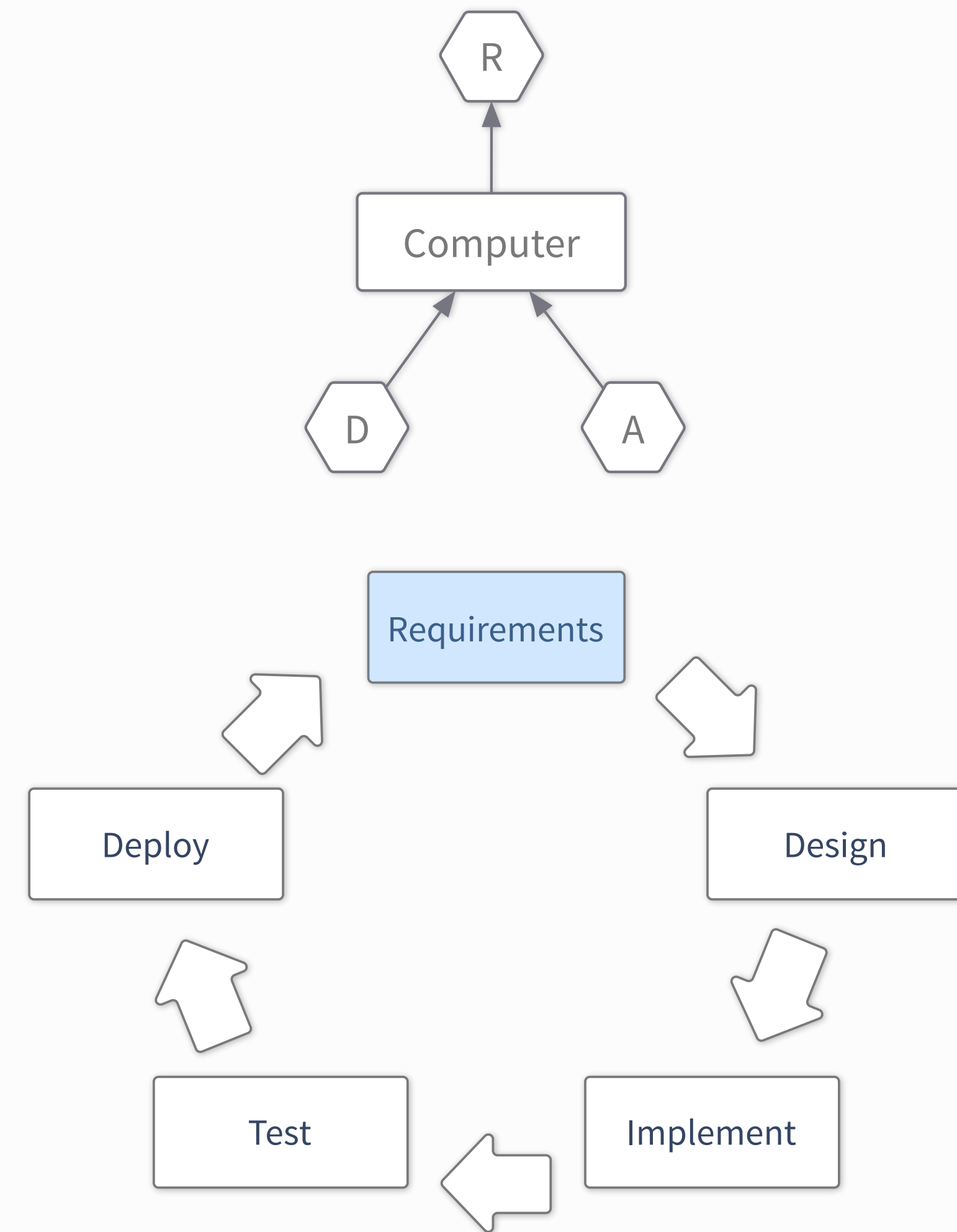
*further reading and tips in these boxes

What is programming?



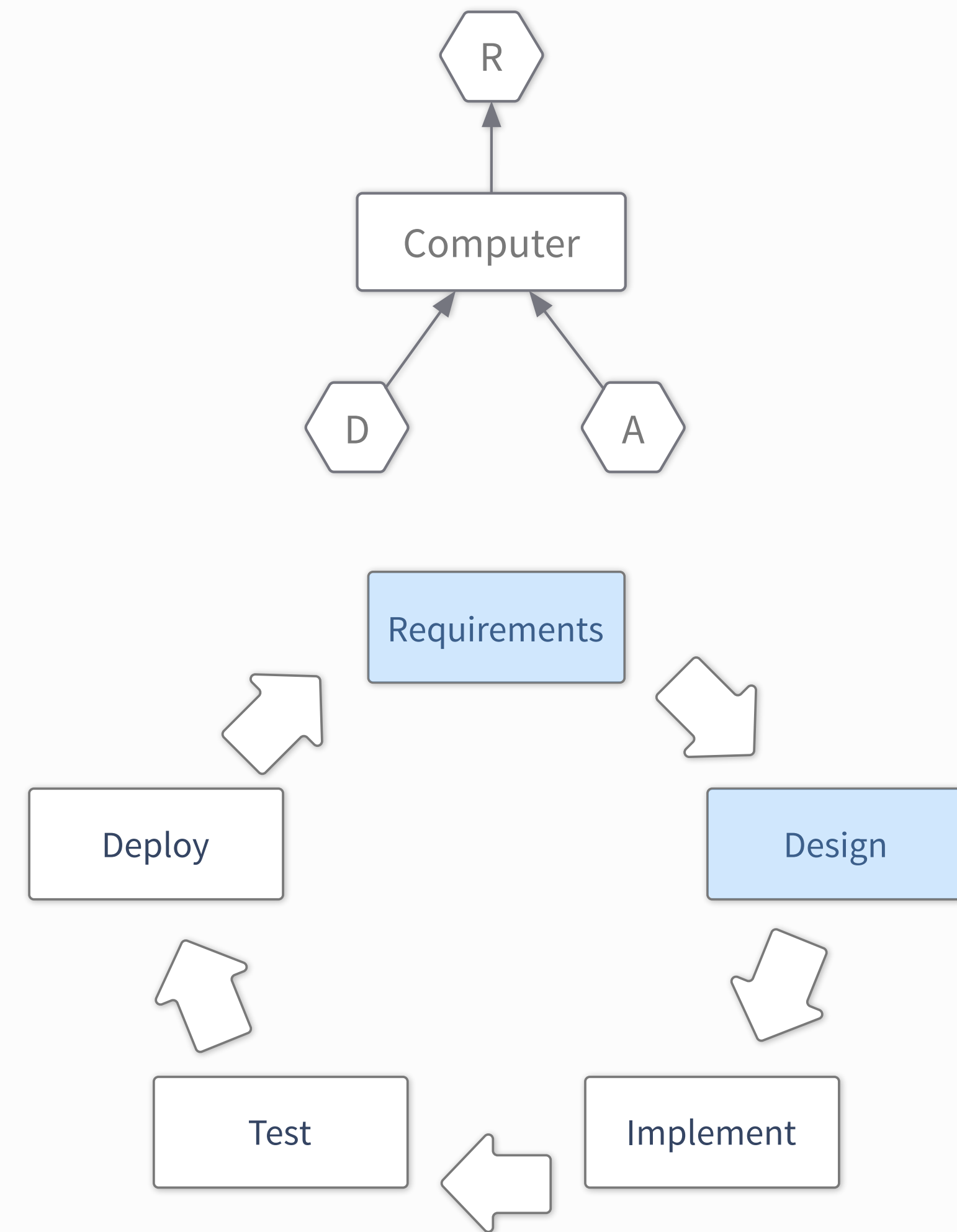
What is programming?

- **Understand** & define the problem to solve
 - ▶ Define the **requirements** for your software



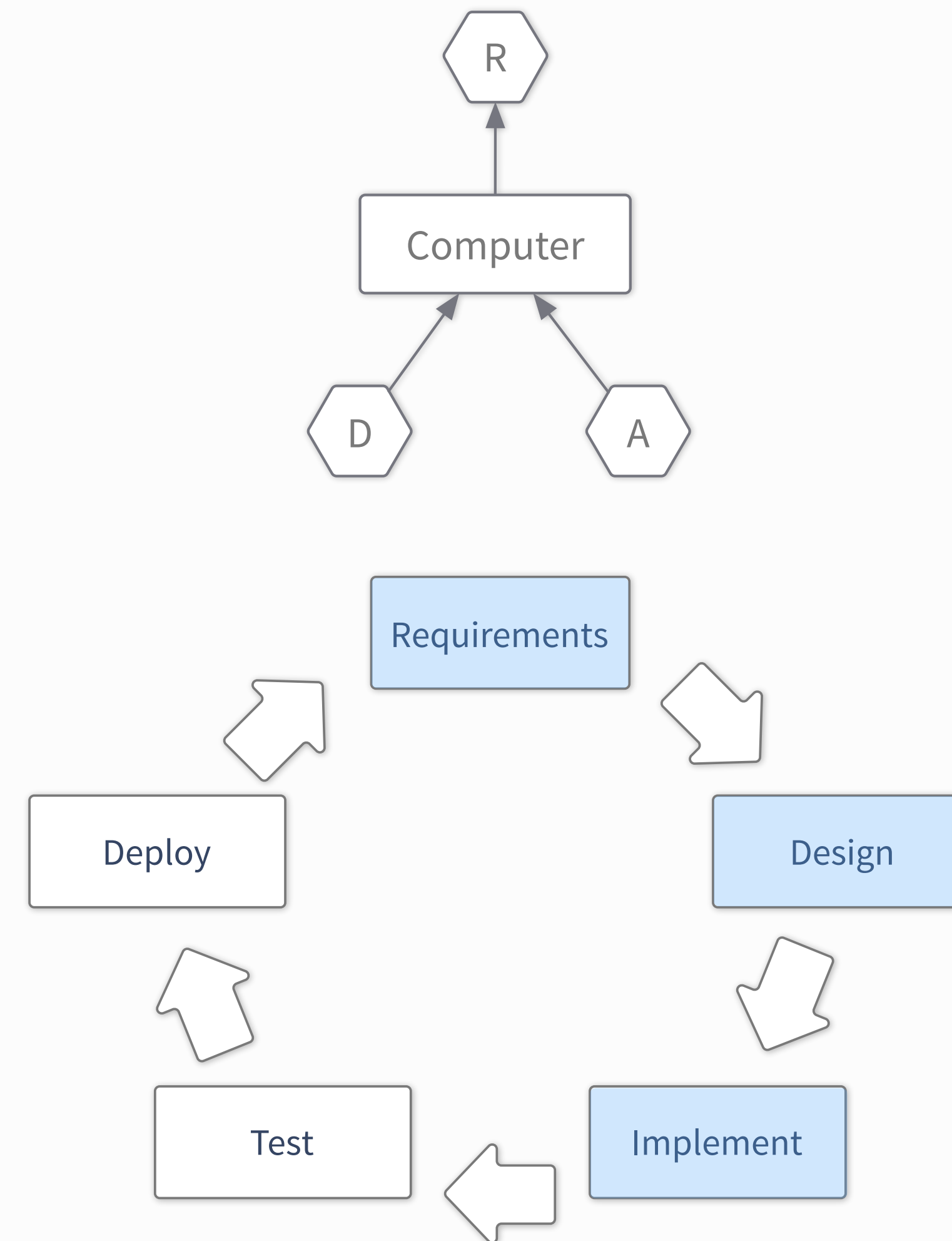
What is programming?

- **Understand** & define the problem to solve
 - ▶ Define the **requirements** for your software
- Formulate a **possible solution** (design)



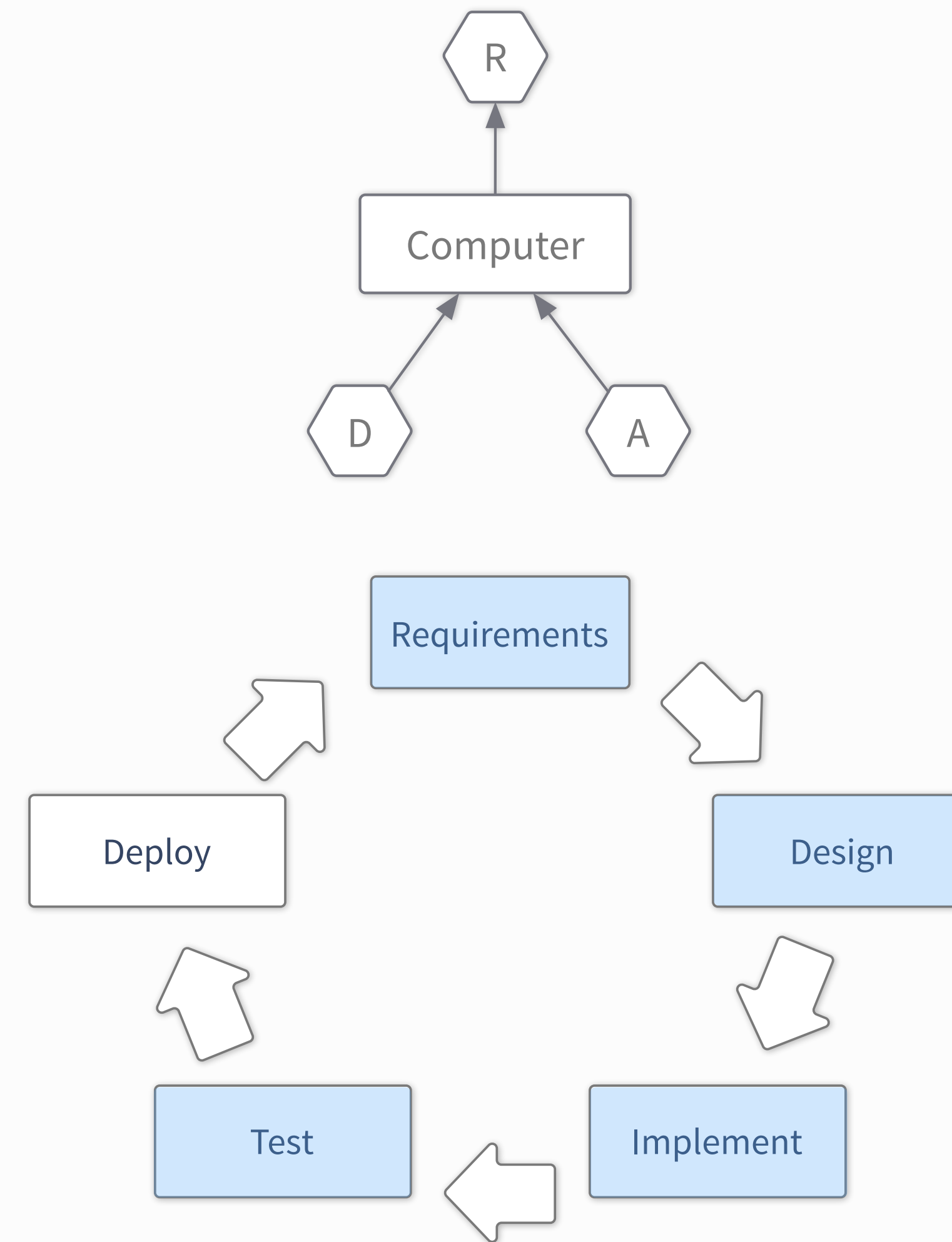
What is programming?

- **Understand** & define the problem to solve
 - ▶ Define the **requirements** for your software
- Formulate a **possible solution** (design)
 - ▶ Identify key functionalities and features
- **Implement** the design
 - ▶ Choose the language
 - ▶ Write code, debug it
 - ▶ Prepare documentation



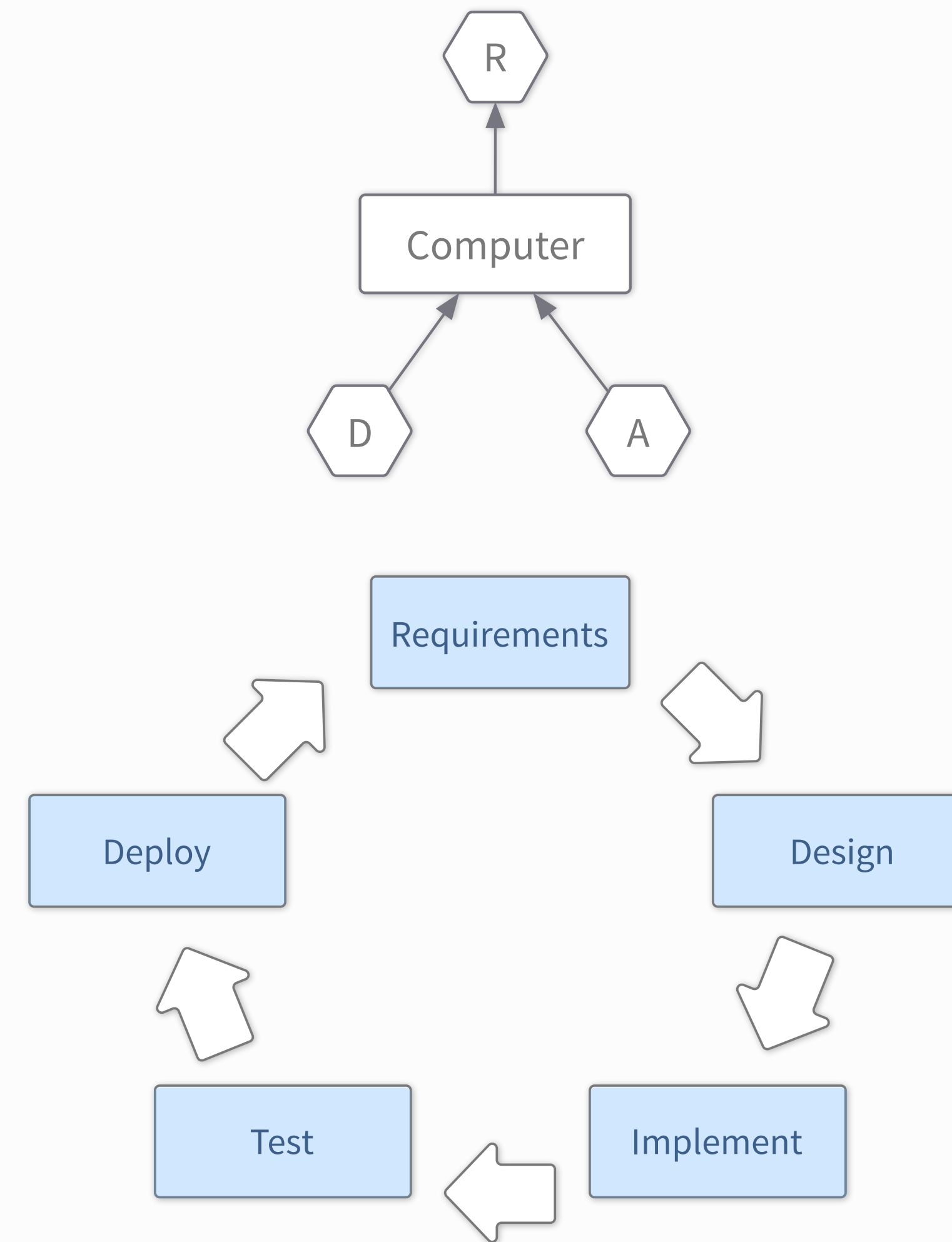
What is programming?

- **Understand** & define the problem to solve
 - ▶ Define the **requirements** for your software
- Formulate a **possible solution** (design)
 - ▶ Identify key functionalities and features
- **Implement** the design
 - ▶ Choose the language
 - ▶ Write code, debug it
 - ▶ Prepare documentation
- **Validate** the code
 - ▶ Perform thorough verification

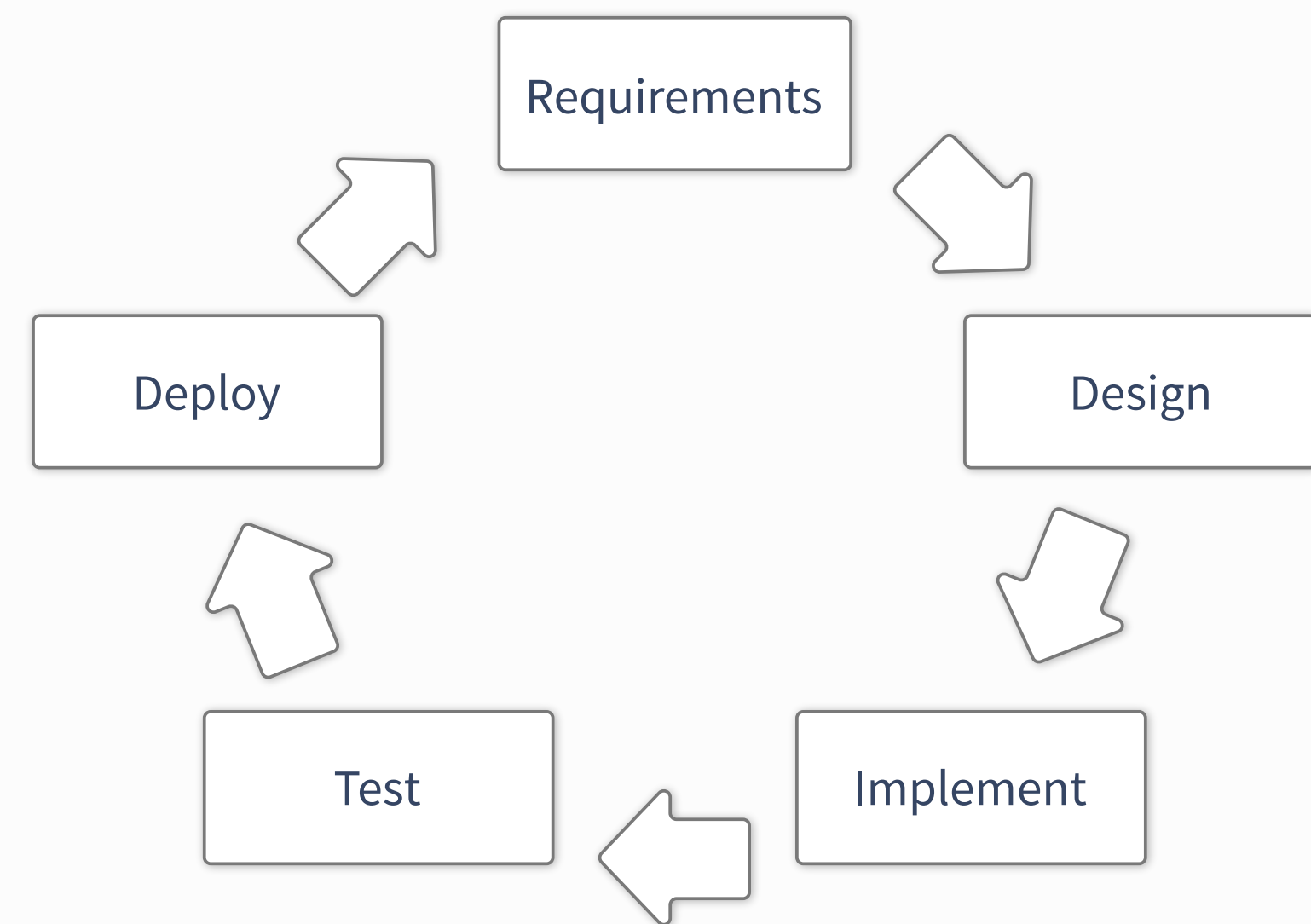


What is programming?

- **Understand** & define the problem to solve
 - ▶ Define the **requirements** for your software
- Formulate a **possible solution** (design)
 - ▶ Identify key functionalities and features
- **Implement** the design
 - ▶ Choose the language
 - ▶ Write code, debug it
 - ▶ Prepare documentation
- **Validate** the code
 - ▶ Perform thorough verification
 - ▶ Execute unit and system tests
- **Deliver** the code
 - ▶ Collect feedback
 - ▶ Ensure portability to different platforms?
- Go back to square 1

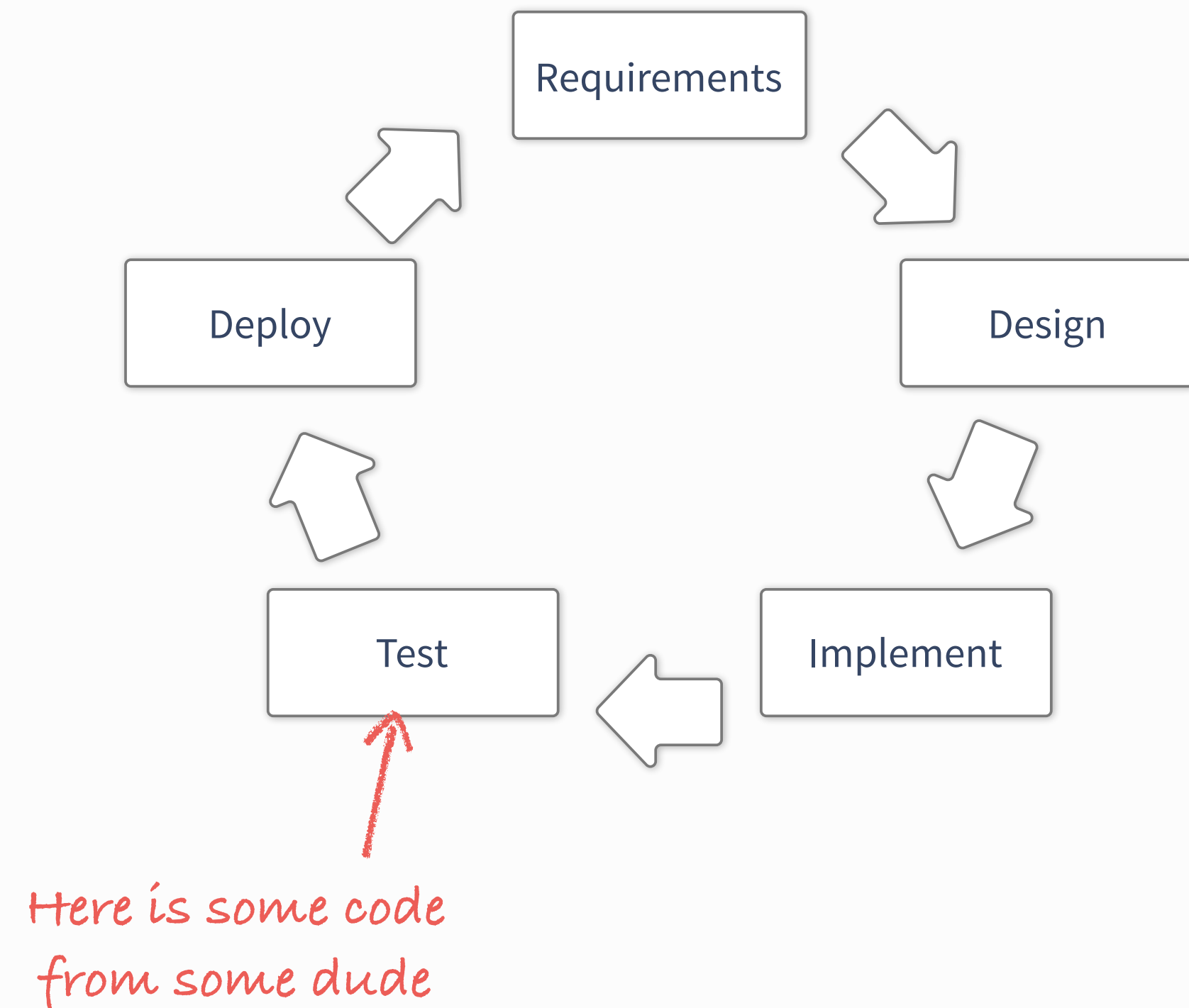


What programming is really like:



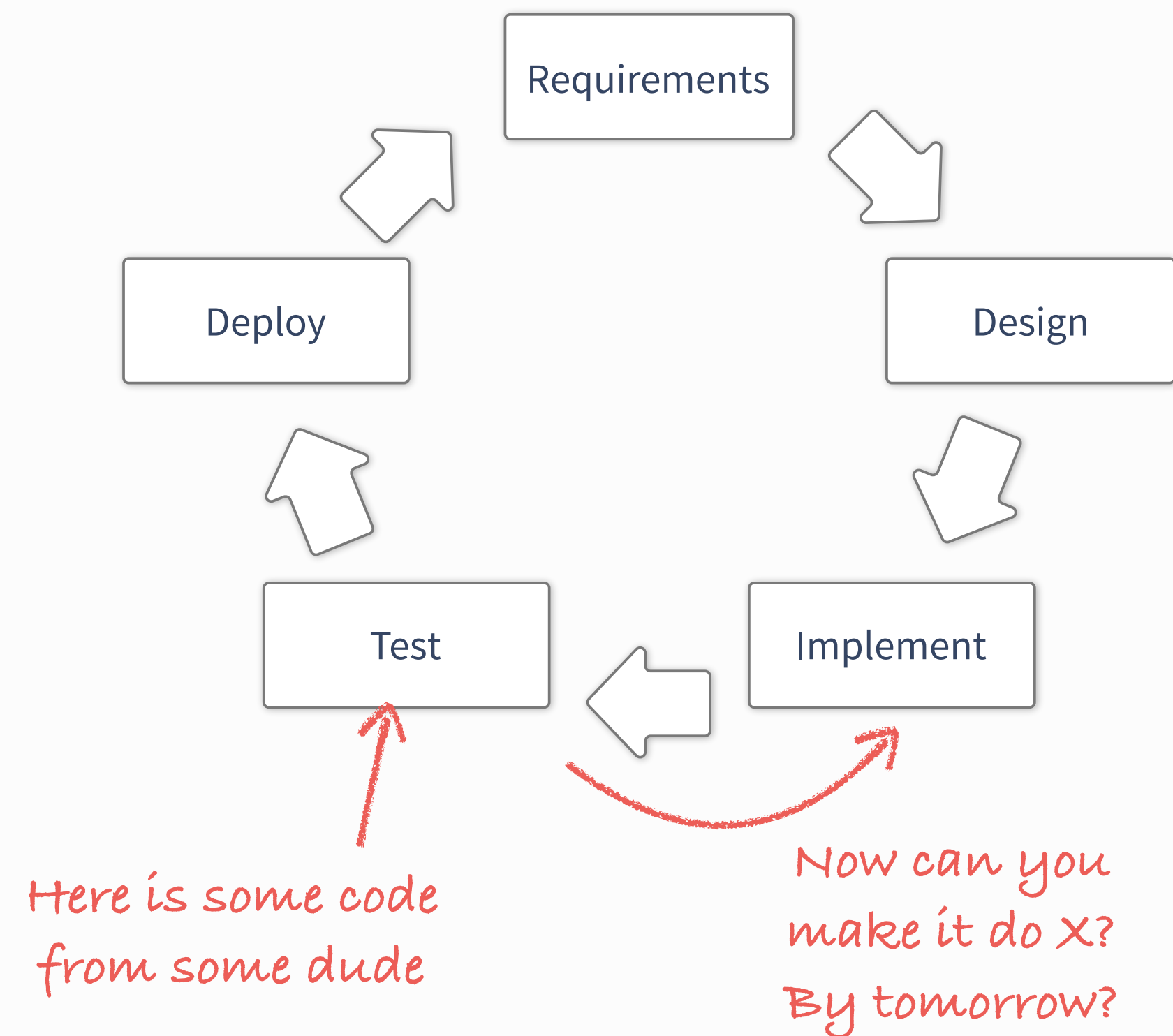
What programming is really like:

- Inherit **some code**
 - give it a try to get the hang of it



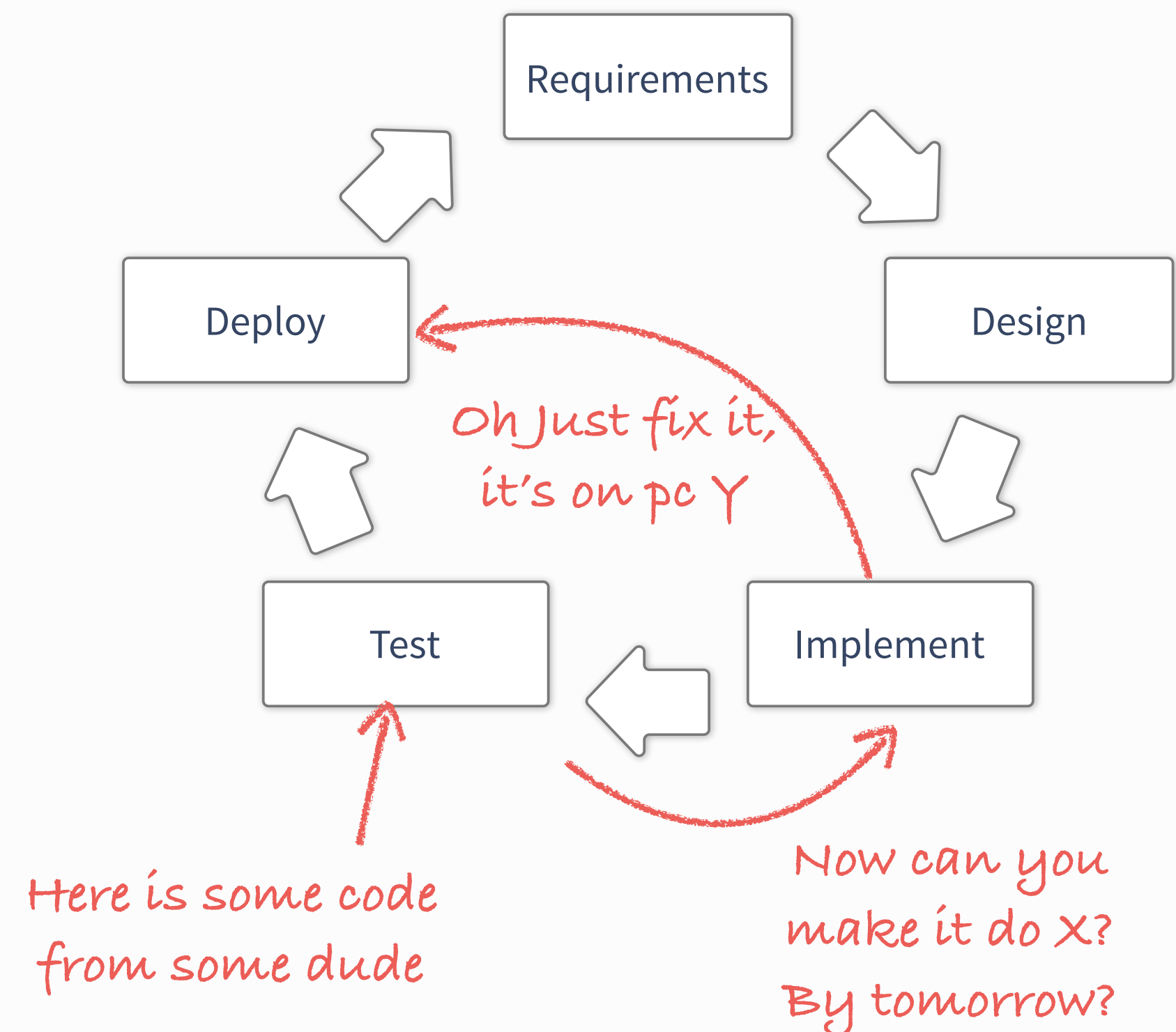
What programming is really like:

- Inherit **some code**
 - ▶ give it a try to get the hang of it
- Add **some features**
 - ▶ the purpose of which is not completely clear
 - ▶ by ~~hack~~... patching some files



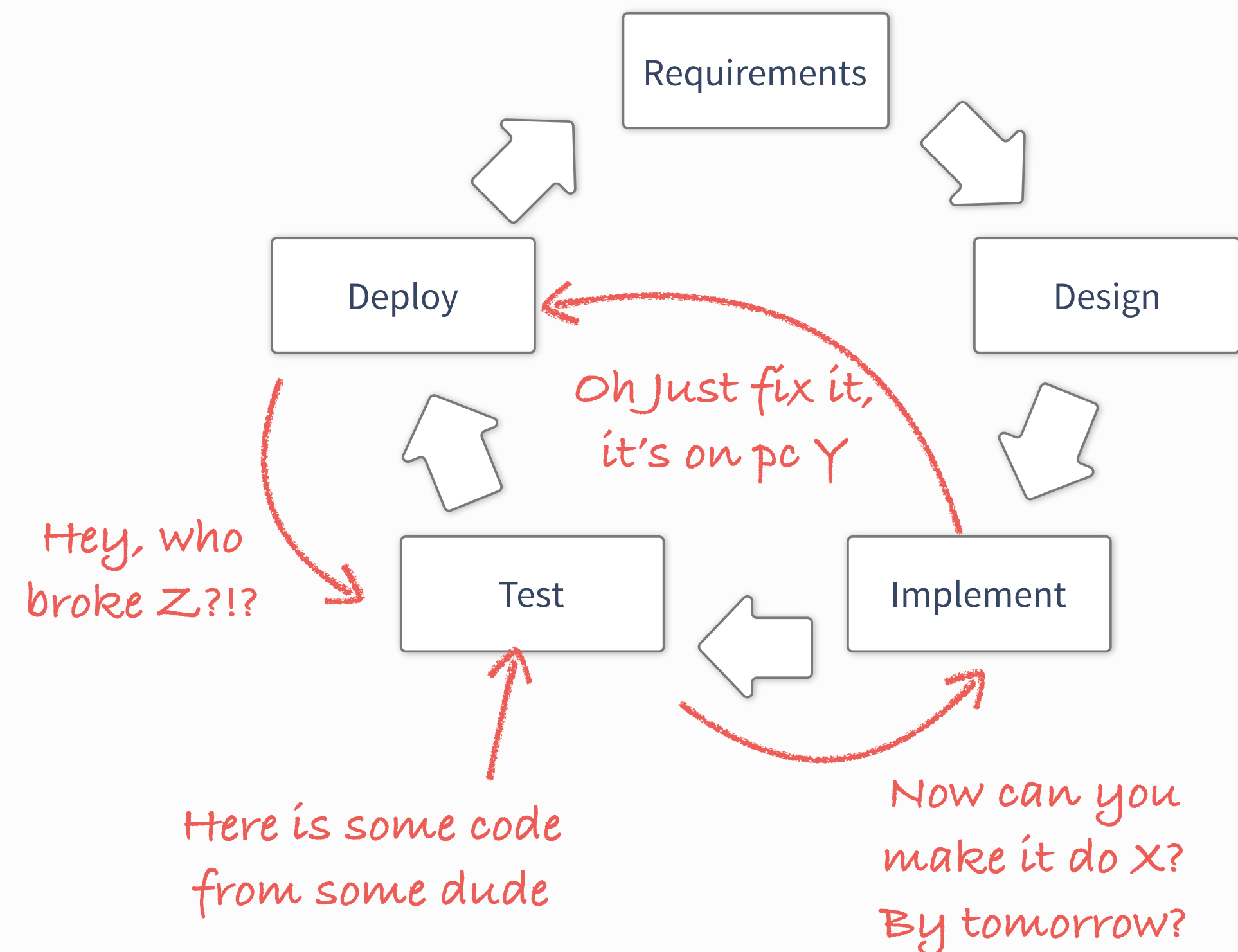
What programming is really like:

- Inherit **some code**
 - ▶ give it a try to get the hang of it
- Add **some features**
 - ▶ the purpose of which is not completely clear
 - ▶ by ~~hack~~... patching some files
- On the **only existing working system**
 - ▶ well, it's the only place where the code runs, isn't it?



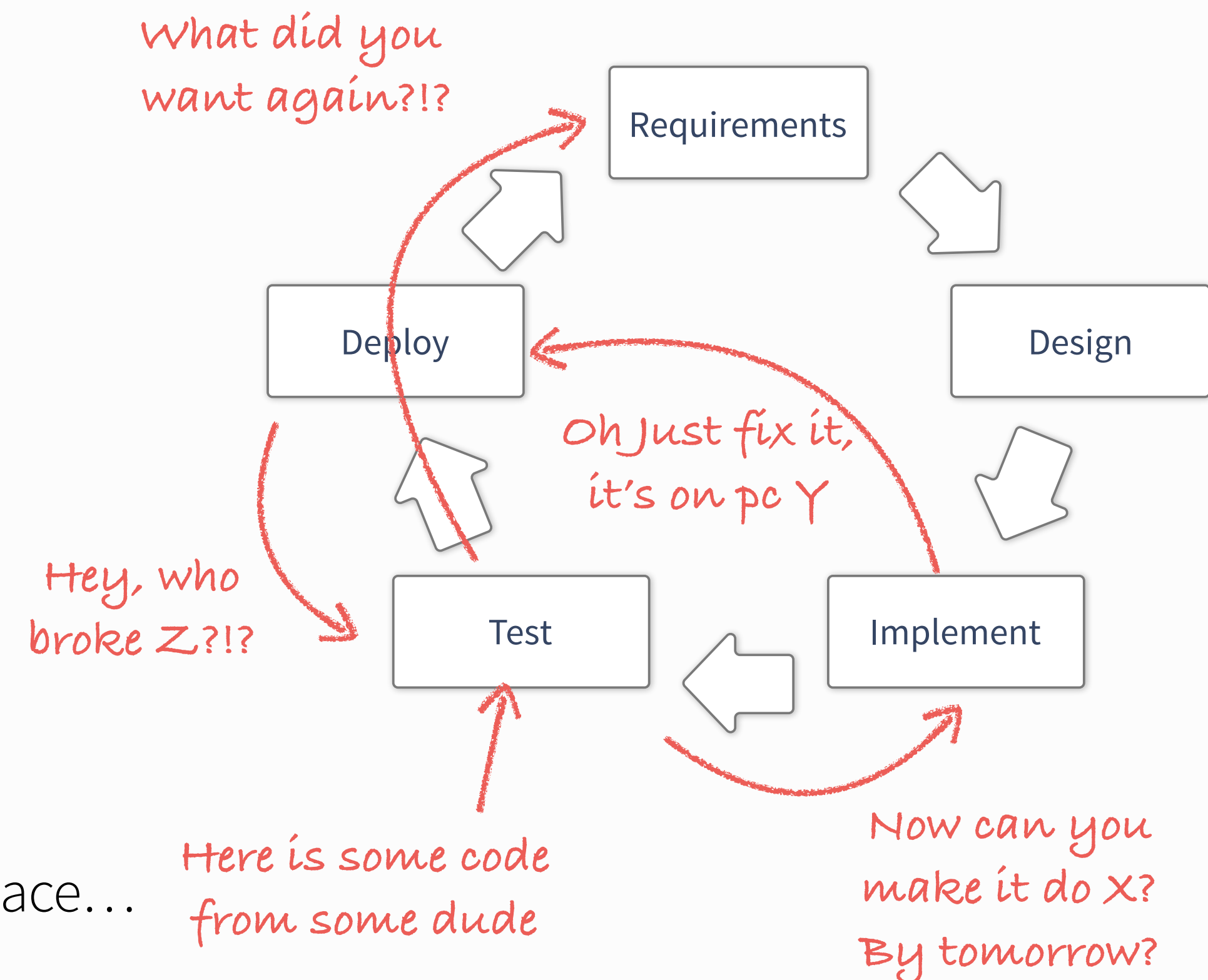
What programming is really like:

- Inherit **some code**
 - ▶ give it a try to get the hang of it
- Add **some features**
 - ▶ the purpose of which is not completely clear
 - ▶ by ~~hack~~... patching some files
- On the **only existing working system**
 - ▶ well, it's the only place where the code runs, isn't it?
- **Break some other code** by accident
 - ▶ Desperately try to figure out why.



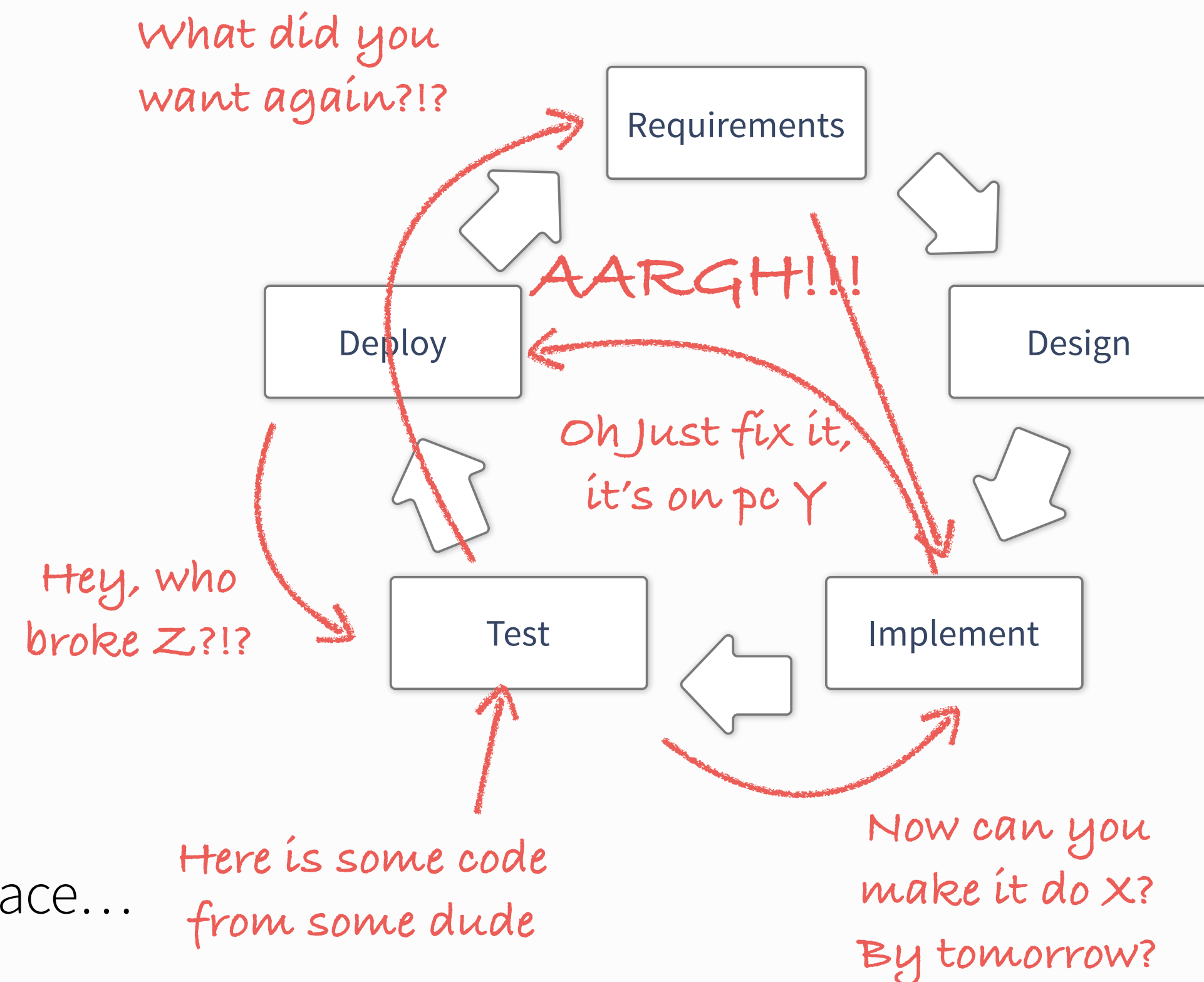
What programming is really like:

- Inherit **some code**
 - ▶ give it a try to get the hang of it
- Add **some features**
 - ▶ the purpose of which is not completely clear
 - ▶ by ~~hack~~... patching some files
- On the **only existing working system**
 - ▶ well, it's the only place where the code runs, isn't it?
- **Break some other code** by accident
 - ▶ Desperately try to figure out why.
- Just to finally realise you **got it wrong** in the first place...

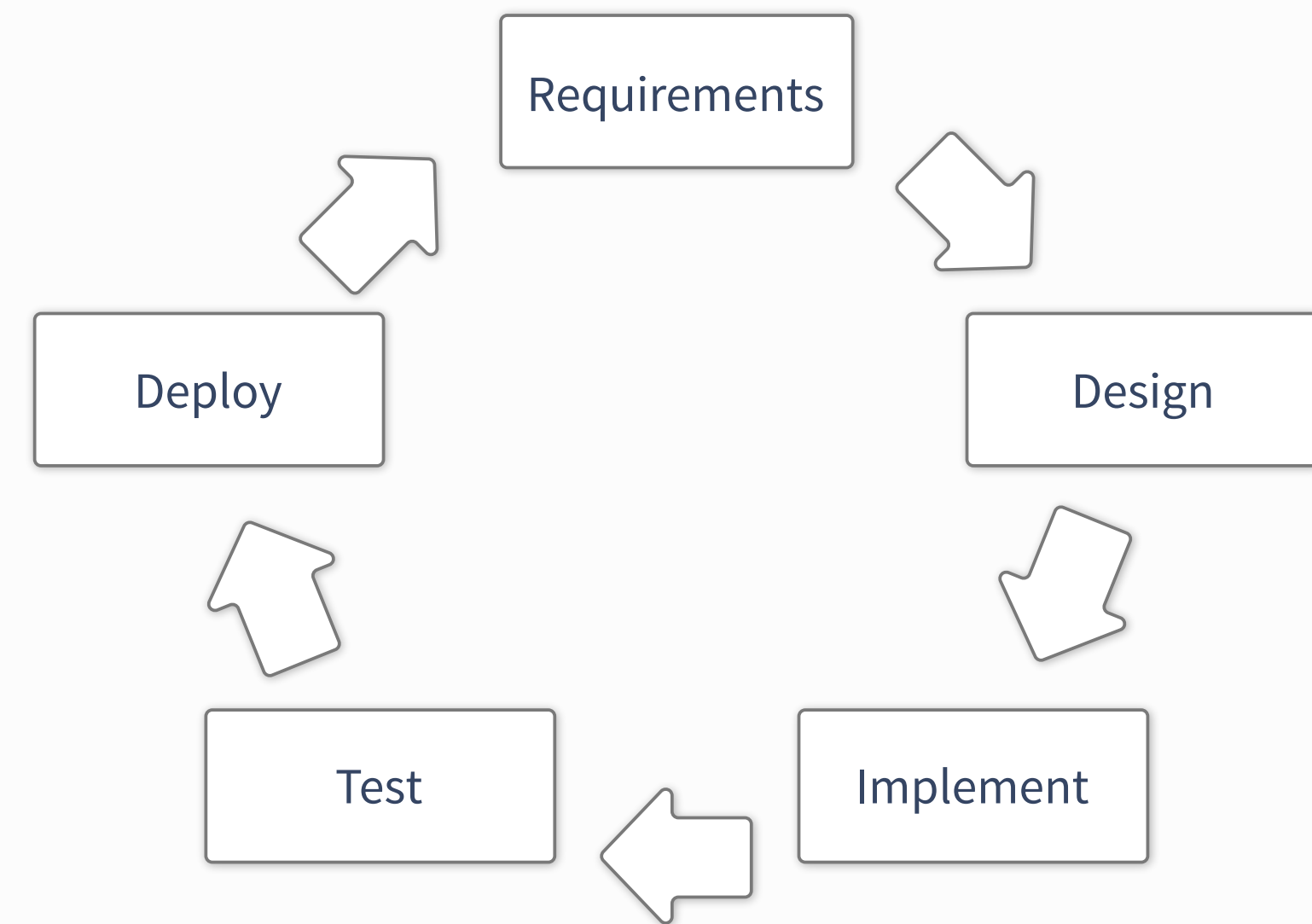


What programming is really like:

- Inherit **some code**
 - ▶ give it a try to get the hang of it
- Add **some features**
 - ▶ the purpose of which is not completely clear
 - ▶ by ~~hack~~... patching some files
- On the **only existing working system**
 - ▶ well, it's the only place where the code runs, isn't it?
- **Break some other code** by accident
 - ▶ Desperately try to figure out why.
- Just to finally realise you **got it wrong** in the first place...
 - ▶ and so on and so on...



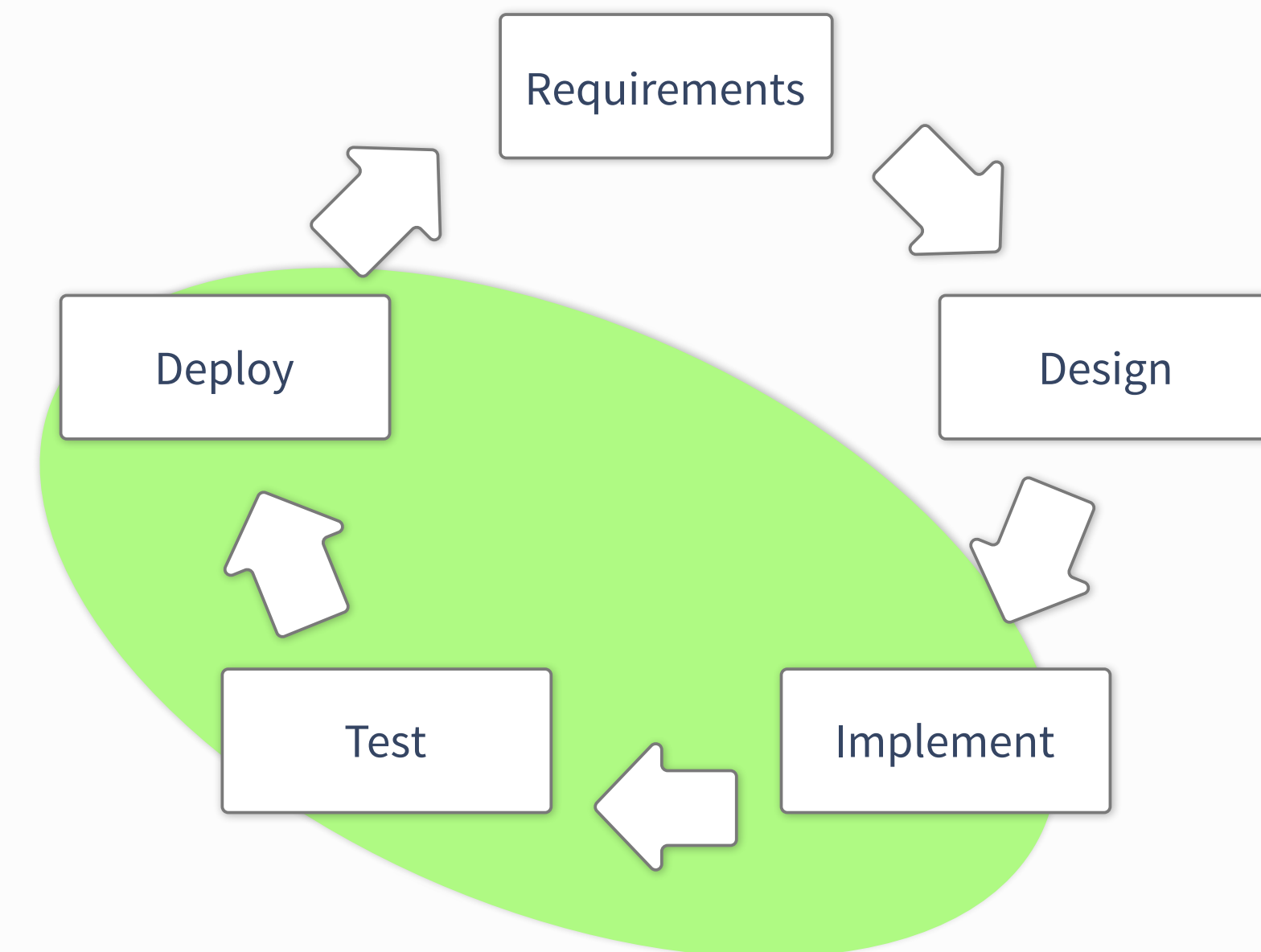
(some) Typical programming contexts



(some) Typical programming contexts

Small projects

- Shortened dev-cycle: Implement, Test, Deploy
 - Requirements and design pre-defined
- Mostly self-contained
 - no/few external interfaces and dependencies
- Few developers (typically **you**)



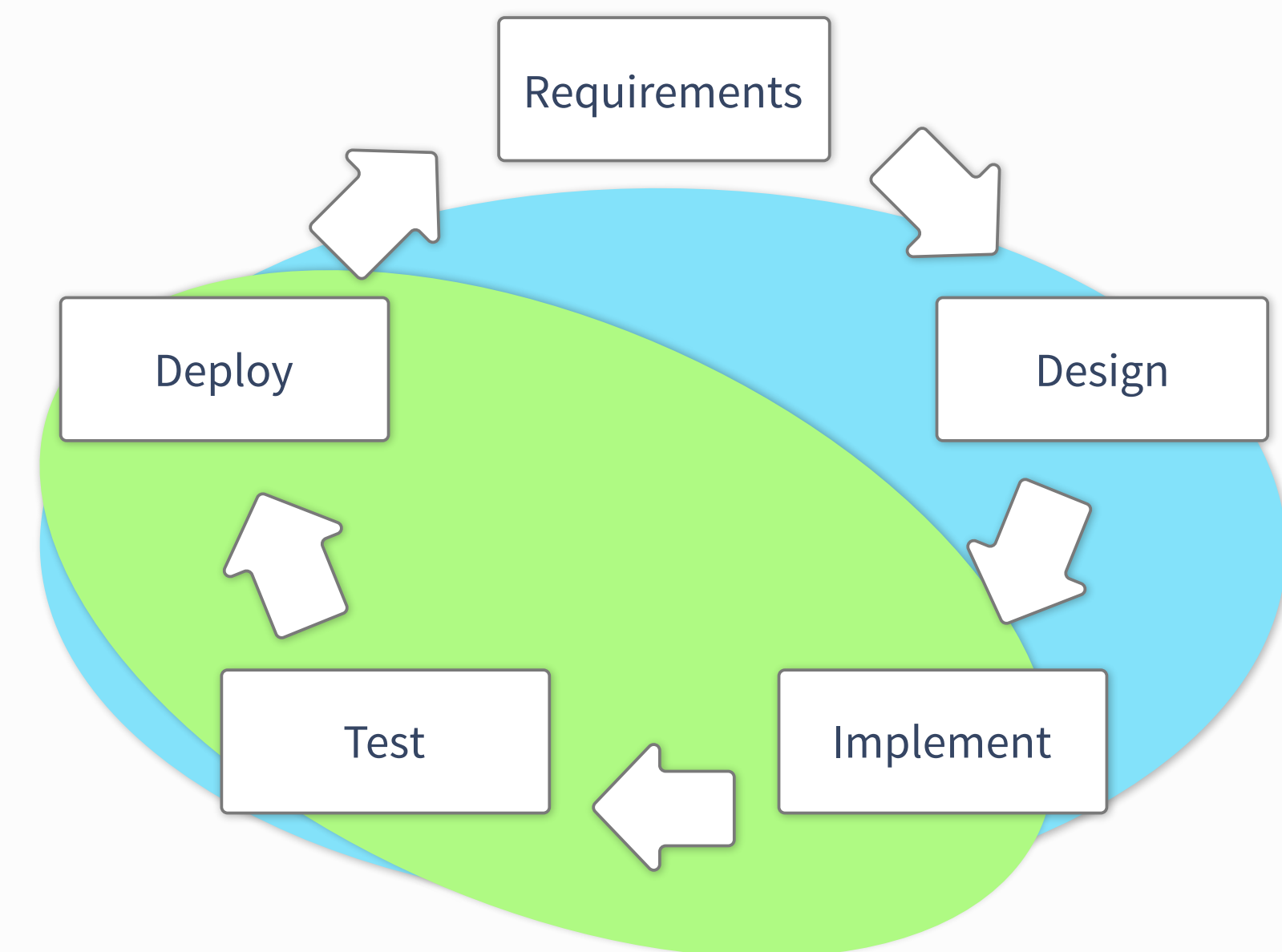
(some) Typical programming contexts

Small projects

- Shortened dev-cycle: Implement, Test, Deploy
 - Requirements and design pre-defined
- Mostly self-contained
 - no/few external interfaces and dependencies
- Few developers (typically **you**)

Medium projects

- The design effort becomes unavoidable
- Well defined interfaces and dependencies
 - e.g. external frameworks
- Multiple developers
 - human interaction becomes non-trivial



(some) Typical programming contexts

Small projects

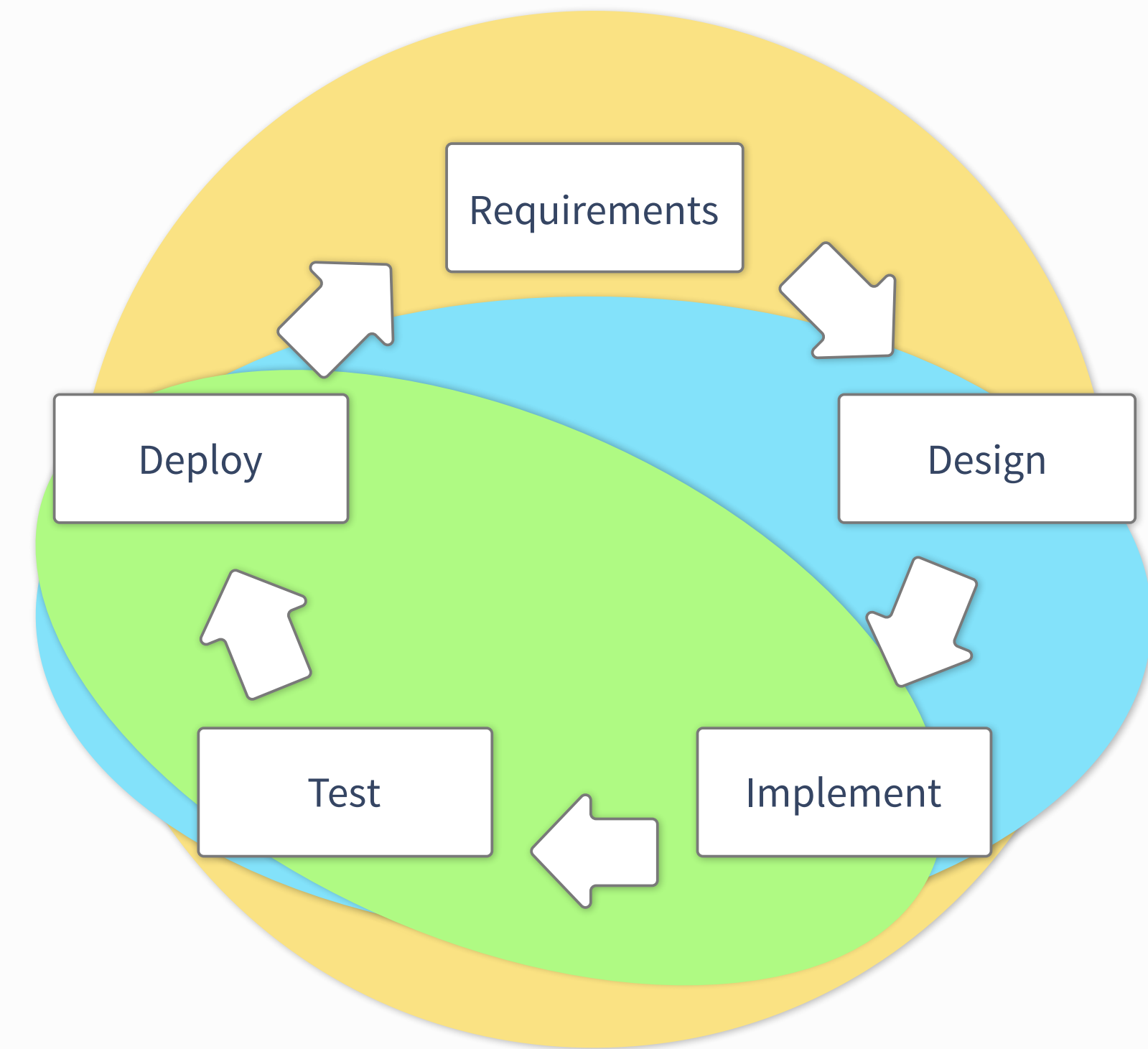
- Shortened dev-cycle: Implement, Test, Deploy
 - Requirements and design pre-defined
- Mostly self-contained
 - no/few external interfaces and dependencies
- Few developers (typically **you**)

Medium projects

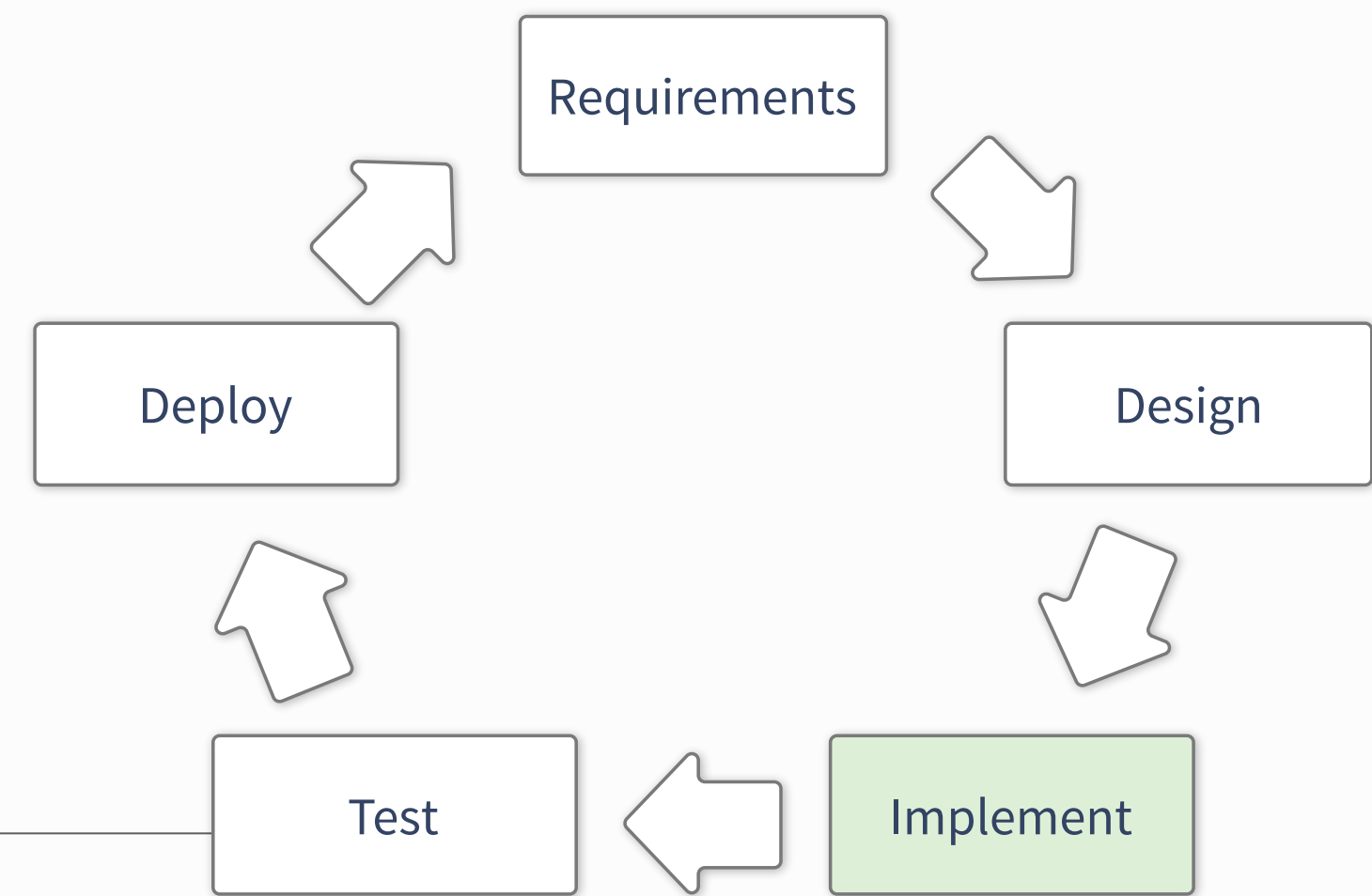
- The design effort becomes unavoidable
- Well defined interfaces and dependencies
 - e.g. external frameworks
- Multiple developers
 - human interaction becomes non-trivial
- **Maintenance issues** make their appearance

Large projects (TDAQ)

- Requirements and specifications become crucial
- Many interfaces, complex dependencies
- Sizeable userbase
 - Support becomes your worst nightmare



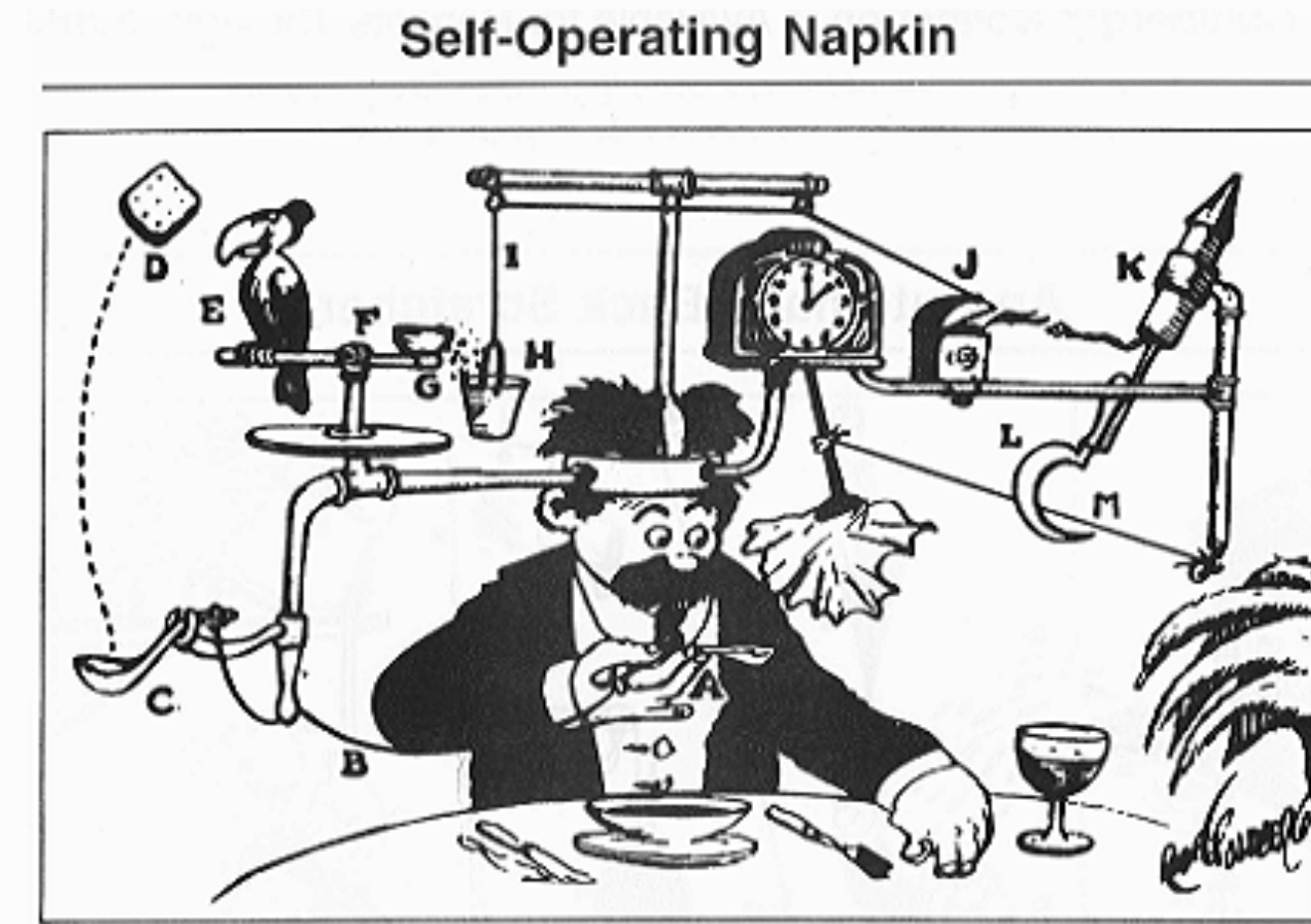
Implementation



Do not reinvent the wheel

Look around for existing solutions

- Many problems have already been solved
- (Sometimes necessary — avoid dependencies)
 - ▶ Do not reject a library because of too many features
- Look for libraries where:
 - ▶ Active community? Well maintained? Tested?
 - ▶ Rule of thumb: Last commit a few days ago, at most over a year old



“Prof. Lucifer Butts and his Self-Operating Napkin”,
by Rube Goldberg

Getting to know new frameworks:

- Try the simple tools and then ask for advice
 - ▶ Read the docs (RTFM)
 - Investing time in the beginning will pay off
 - ▶ Are there wikis? Has it been asked on StackOverflow?
 - ▶ python packages: try the ipython “help”
- Start with a simple test
(existing examples -> what you want to do)
 - ▶ Does the code do what you expect?

before looking at external libraries:
Look at the STL / python standard library

When coding - Avoid feature bloating

If you do squeeze every-possible-conceivable-feature in one place:

- You'll probably end up doing nothing right
- **Write specialised toolkits / libraries**

Define features by writing a test that needs to be passed

- Only implement what is strictly needed to pass that test.

Be pragmatic

- Generalising a problem before solving it:
 - ▶ Probably not a good idea
 - ▶ Only do it when you have a use case
- Keep everything as concise as possible (increased readability)
 - ▶ Introduce abstraction only when likely to be actually used
- **Keep it simple!**



**Don't reinvent
the wheel**

Tools of the Trade: Editor and Terminal

Whatever you do, you'll end up using:

- Editor
 - ▶ Know* at least one “omnipresent” editor: **nano**, **vi (m)**, **emacs**, etc.
 - ▶ More modern solutions: have a number of clear benefits for development
 - ▶ Depending on the language / platform (e.g. Java): IDEs are the best choice **Eclipse**, **Netbeans**
- Terminal
 - ▶ Learn about shortcuts (minimal set: **tab**, **ctrl+r**, **ctrl+e**, **ctrl+a** ... have a look)
 - ▶ Knowing about some basic command line-tools will come in handy

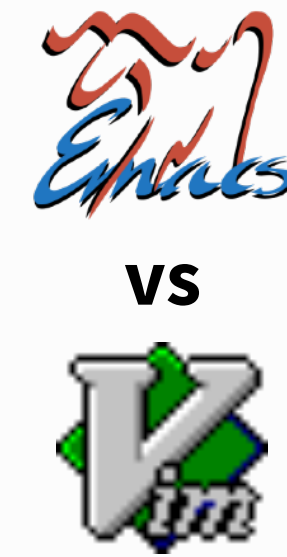
* at least know how to save and exit :)
for the more daring: try **ed**

A few words on editors: Choose what suits you and be effective

The choice of editor is yours to make...¹

- Do you want “a great operating system, lacking only a decent editor”
- Or one with two modes: “beep constantly” and “break everything”²

Both are versatile and learning them is worthwhile



However: modern alternatives have a less-steep learning-curve

- Some are commercial ([Sublime Text](#), TextMate,...)
- Some are open: github’s [Atom](#) & Microsoft’s [VSCode](#)
 - Plugins, git integration, active communities, more plugins...



Once you decided which one is best for you:

- Spend some time learning about features and keybindings
- Many things that might require dozens of keystrokes can be done with 2 (5 in emacs ;))
- Learn about: Linters, extensibility — look at existing plugins



**Use what you find most comfortable
and learn to be efficient with it**

1. an insightful guide: [Text Editors in the Lord of the Rings](#)
2. from the [Editor war](#)

The Terminal - Get used to it

At the beginning: clicking is faster than typing, no need for the terminal

- After learning about some command line tools... probably not
- What if you don't have a GUI?

Searching files: grep, find — example:

```
$ find . -name "*.cc" -exec grep -A 3 "foo" {} +
```

- Displays all matches of “foo” (+3 lines below) in all .cc files from the current work dir

Once you learn some tools it becomes very versatile:

- **sed, head, tail, sort... awk** (a turing-complete interpreted language)
- At the beginning: note down often used commands...
- After a tutorial dump your history* (increase cache size for max usage)

Shell-scripting:

- Anything you do with the shell can just be dumped in a script
- Alternative: Can solve most things more conveniently with an interpreted language
 - Con: interpreters / bindings might not always be available

* dump the last 100 steps:

```
$ history | tail -n 100 > steps.txt
```

log the terminal “responses”:

```
$ script # press ctrl+d to stop
```

tune your bashrc / bash-profile
see additional material

Eventually: terminal is so versatile that typing beats clicking 9 times out to 10

Interlude: Working on the road — SSH

SSH — very, very versatile, more than you think:

- Tunneling
 - ▶ Secure connections to other machines
 - ▶ Use with VNC to avoid man-in-the-middle vulnerability
- Generate keys for authentication
- Working around bad latency / shaky connection
 - ▶ Always use **tmux/screen** or a similar terminal multiplexer
 - ▶ Alternative: **mosh** (mosh.mit.edu)
 - mitigates intermittent connectivity, roaming or just moving to the next meetings...

SSHFS

- Work locally but have files on remote host

SSH tunnel for VNC connection:

```
ssh -L 5902:<VNCServerIP>5902 <user>@<remote>\
  vncserver :<session> -geometry\
  <width>x<height> -localhost -nolisten tcp
```

SSH authentication via kerberos token. In ~/.ssh/config:

```
GSSAPIAuthentication yes
GSSAPIDelegateCredentials yes
HOST lxxplus*
  GSSAPITrustDns yes
```

Lots of things possible with the ssh-config:

```
HOST <host>
  USER <remote-user>
  ProxyCommand ssh <tunnel> nc <host> <port>
```

more on (auto-)tunnelling:

https://security.web.cern.ch/security/recommendations/en/ssh_tunneling.shtml

tmux guides and courses:

<https://robots.thoughtbot.com/a-tmux-crash-course>
<http://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/>

The right tool for many jobs - interpreted languages

Keep your code as short as possible

while maintaining readability

- Sometimes means to use the right language
- Often quicker / nicer: interpreted languages
 - ▶ python, perl, ruby, tcl, lua
- Used as binding languages:
 - ▶ **Performance critical** code in C/C++
 - ▶ Instantiated within python
(e.g. in CMS, ATLAS & LHCb offline Software)
 - ▶ Best of both worlds
- Python: large standard library & very expressive!

```
from __future__ import print_function
from argparse import ArgumentParser

parser = ArgumentParser(description="Get number of days")
parser.add_argument("month", type=str, nargs='+', help="Name of month")
args = parser.parse_args()

months = {"january": 31, "february": 28, "march": 31,
          "april": 30, "may": 31, "june":30,
          "july": 31, "august": 31, "september": 30,
          "october": 31, "november": 30, "december": 31}

for usermonth in args.month:
    if usermonth in months:
        print("{0} has {1} days.".format(usermonth, months[usermonth]))
    else:
        print("sorry. month '{0}' not known.".format(usermonth))
```

Keep it
easy to read

Easier to maintain; Easy to re-use

Interlude: iPython

```
ArrayType = class array(__builtin__.object)
| array(typecode [, initializer]) -> array
|
| Return a new array whose items are restricted by typecode, and
| initialized from the optional initializer value, which must be a list,
| string or iterable over elements of the appropriate type.
|
| Arrays represent basic values and behave very much like lists, except
| the type of objects stored in them is constrained.
|
| Methods:
|
| append() -- append a new item to the end of the array
| buffer_info() -- return information giving the current memory info
| byteswap() -- byteswap all the items of the array
| count() -- return number of occurrences of an object
| extend() -- extend array by appending multiple elements from an iterable
| fromfile() -- read items from a file object
| fromlist() -- append items from the list
```


Interlude: iPython

> ipython

```
In [1]: import array
```

```
In [2]: help (array)
```

```
In [3]: import ROOT
```

```
In [4]: help (ROOT.TH1D)
```

Interlude: iPython

```
class TH1D(TH1, TArrayD)
|   Method resolution order:
|   TH1D
|   TH1
|   TNamed
|   TObject
|   TAttLine
|   TAttFill
|   TAttMarker
|   TArrayD
|   TArray
|   ObjectProxy
|   __builtin__.object
|
|   Methods defined here:
|
|   AddBinContent(self, *args)
|       void TH1D::AddBinContent(int bin)
|       void TH1D::AddBinContent(int bin, double w)
```

Interlude: iPython

> ipython

```
In [1]: import array
```

```
In [2]: help (array)
```

```
In [3]: import ROOT
```

```
In [4]: help (ROOT.TH1D)
```

```
In [4]: run myscript.py
```

Documentation: Do it while it's fresh

Two sides of the same coin: embedded and standalone documentation

- Both necessary to make your programs easy to use
- They have different purpose!

Embedded documentation:

- Explain interfaces, i.e. function signatures
- Make note of possible future problems (better: prevent them)
- Sometimes might be good to document your reasoning
- Do not “over-comment”
- Clean code: **You write it once and you read it many times**

Standalone documentation:

- Again: Explain your interfaces (can be derived from internal, e.g. doxygen.org)
- For large projects: **Explain the big picture**
 - ▶ Wiki pages with use-cases and examples
 - ▶ Consider using UML (unified modelling language)

```
class TheClass(object):
    """ Documentation of this class. """
    def __init__(self, var):
        self.var_ = var
        ## @var var_
        # my member variable

        ## Documentation of this function.
        # More on what this function does.
        ## @param arg1 an integer argument
        ## @param arg2 a string argument
        ## @returns a list of ...
    def some_function(self, arg1, arg2):
        pass
```

```
if a > b: # when a is greater than b, do...
```

Documentation: Do it while it's fresh

Two sides of the same coin: embedded and standalone documentation

- Both necessary to make your programs easy to use
- They have different purpose!

Embedded documentation:

- Explain interfaces, i.e. function signatures
- Make note of possible future problems (better: prevent them)
- Sometimes might be good to document your reasoning
- Do not “over-comment”
- Clean code: **You write it once and you read it many times**

Standalone documentation:

- Again: Explain your interfaces (can be derived from internal, e.g. doxygen.org)
- For large projects: **Explain the big picture**
 - ▶ Wiki pages with use-cases and examples
 - ▶ Consider using UML (unified modelling language)

```
class TheClass(object):
    """ Documentation of this class. """
    def __init__(self, var):
        self.var_ = var
    ## @var var_
    # my member variable

    ## Documentation of this function.
    # More on what this function does.
    ## @param arg1 an integer argument
    ## @param arg2 a string argument
    ## @returns a list of ...
    def some_function(self, arg1, arg2):
        pass
```

```
if a > b: # when a is greater than b, do...
```

Document while coding

You write it once, read it many times

Write build scripts to ease your life

Makefiles — makes compilation **easier and faster**

- Makefiles might look complex
- More than one source file: Useful!
 - ▶ Again: Think about yourself in 2 years
- Write your own for a small project
- Automatically allows parallel compilation (option -j)

```
CC=clang++
CCFLAGS=-Wall -pedantic -std=c++14
SOURCES=src/howmanydays.cc
OBJECTS=$(SOURCES:.cc=.o)
EXE=howmanydays

all: $(SOURCES) $(EXE)

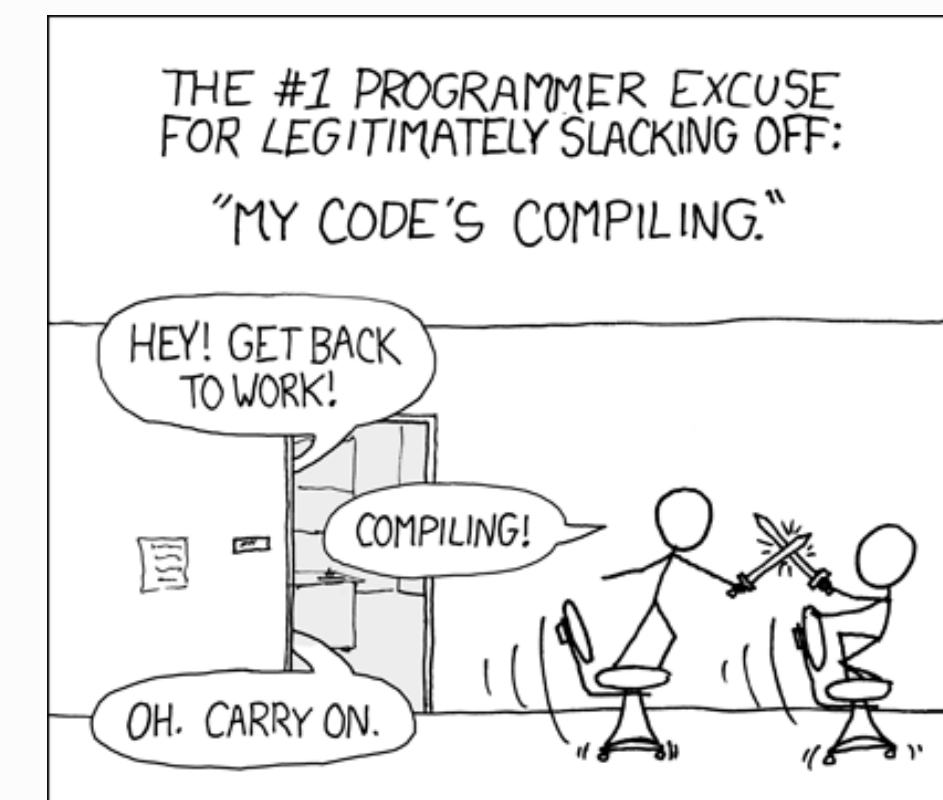
$(EXE): $(OBJECTS)
    $(CC) $(CCFLAGS) $(OBJECTS) -o bin/$@

%.o: %.cc
    $(CC) $(CCFLAGS) -c -o $@ $<

.PHONY: clean all
clean:
    rm -f $(OBJECTS) bin/$(EXE)
```

Abstraction layer on top: CMake and others

- Might look like overkill; Makes things easier in the long run
 - ▶ CMake is easier to read and better documented
 - ▶ Improved **portability**
 - ▶ Support different build-systems: ninja, GNU make, ...
- At least you should learn how to compile with it



“Compiling” by Randall Munroe
xkcd.com

Use appropriate tools for debugging

While running your code:

- printing to console: only suitable for (very) small code base
- Sooner or later have to use a debugger: **gdb** (GNU debugger) — better sooner than later
 - ▶ basic commands: **run**, **bt**, **info <*>**, **help**
 - ▶ very useful trick - attach to a running program: **gdb <executive> <pid>**
- Python debugger (**pdb** or rather **ipdb***):

General hints for debugging

- Segmentation violations due to memory management
 - ▶ Life-time vs. scope
 - ▶ Only use raw pointers when you have to!
(I.e. when performance becomes crucial and *you know what you're doing*)
 - ▶ Look at smart pointers (part of C++11/14 standards, alternative: boost)
- [Even if you don't have crashes: Memory Leaks. Try **valgrind** \(valgrind.org\)](#)

```
*import ipdb; ipdb.set_trace() # set a breakpoint
```


Static Code Checking

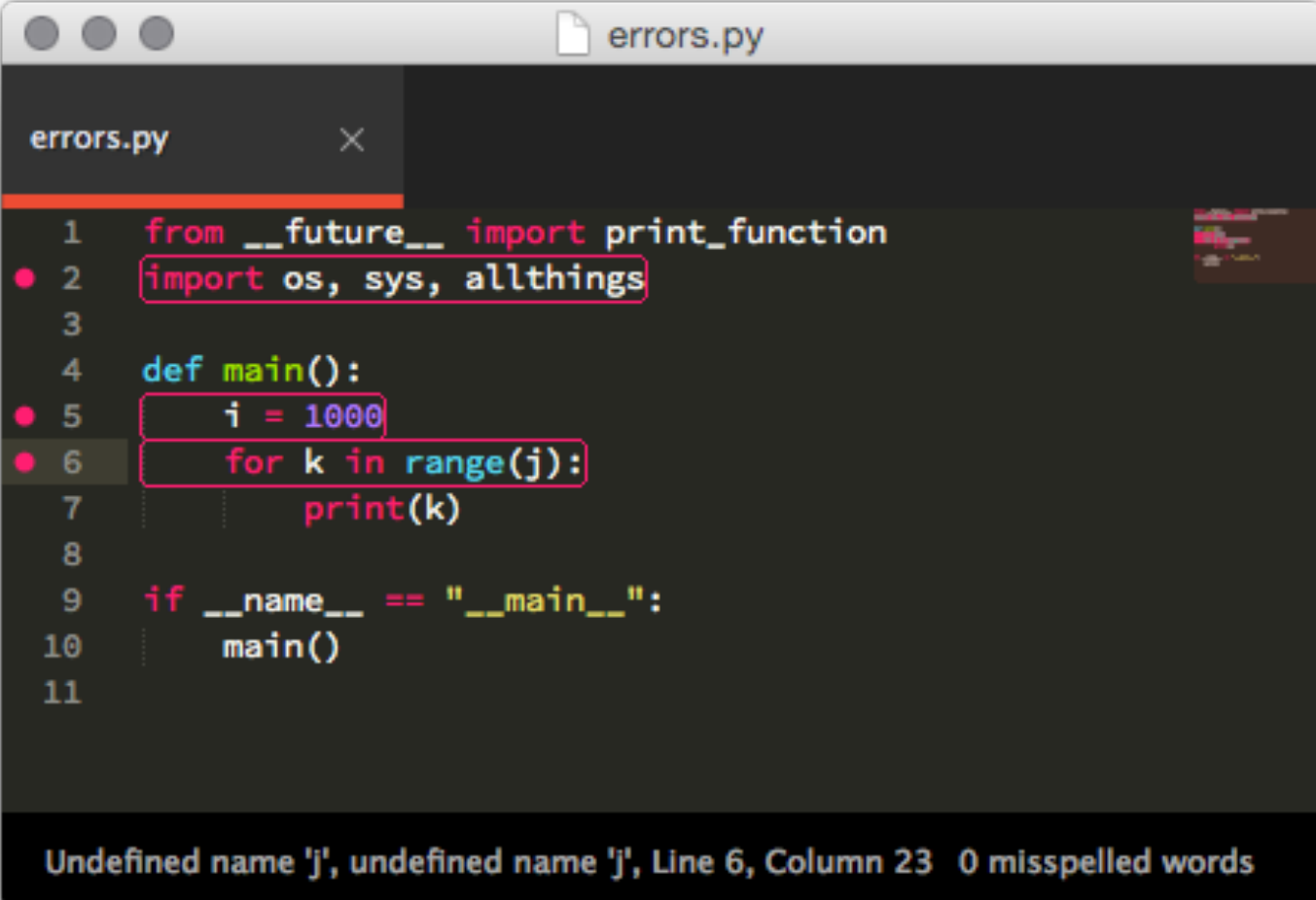
While writing your code:

- There are static code analysis tools that can help you
- Try out a linter for your preferred editor (e.g. atom: <https://atom.io/packages/linter>)
 - ▶ Highlights potentially problematic code
 - ▶ Your code will be more reliable

Static checking at compile time:

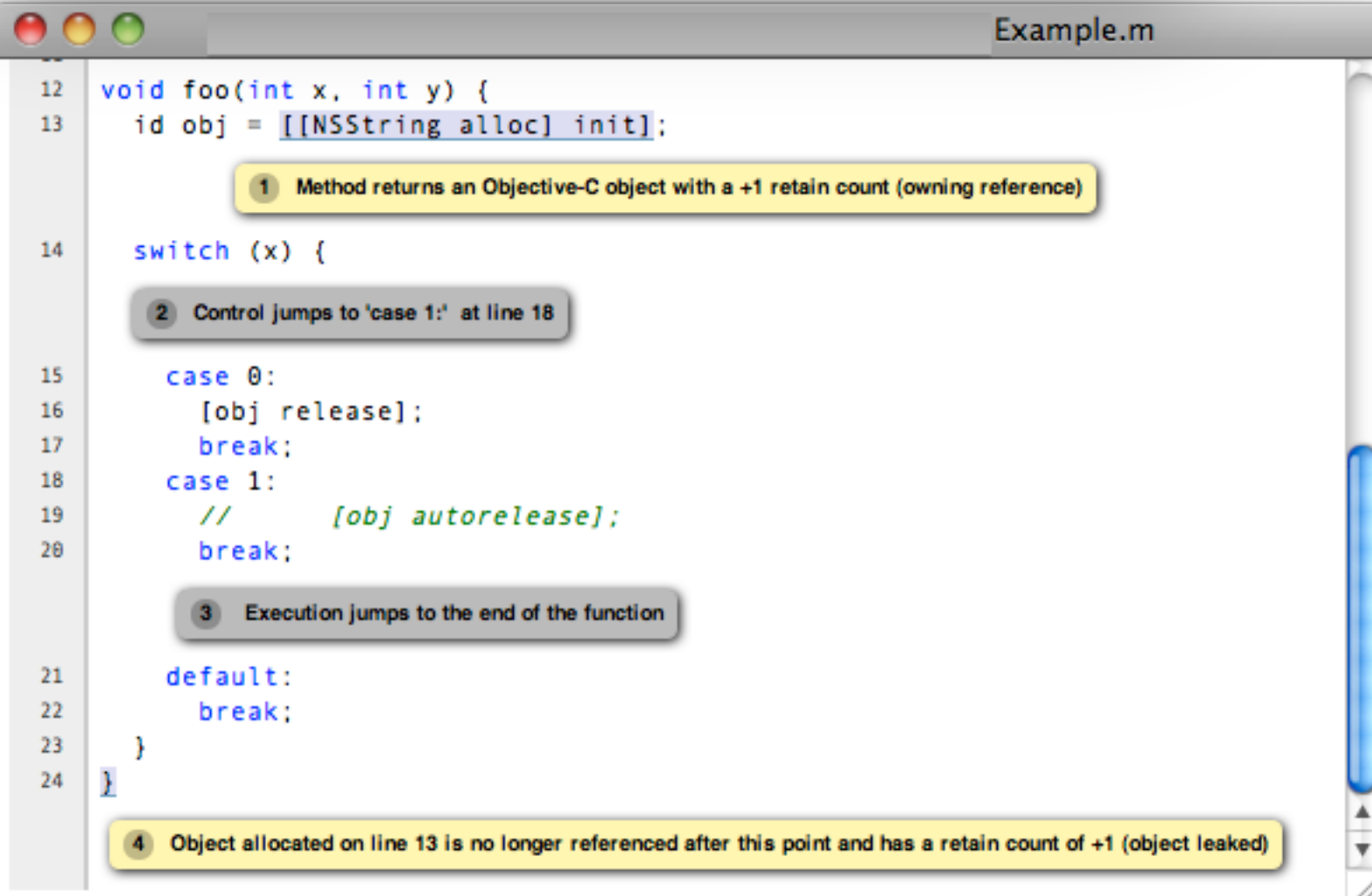
- Clang has a nice suite of static checks implemented <http://clang-analyzer.llvm.org>
 - ▶ Can also enforce coding styles
- Takes longer than compiling; HTML reports with possible bugs
- Might flag some false-positives

Static code checking helps you avoid problems!



```
errors.py
1 from __future__ import print_function
2 import os, sys, allthings
3
4 def main():
5     i = 1000
6     for k in range(j):
7         print(k)
8
9 if __name__ == "__main__":
10     main()
11
```

Undefined name 'j', undefined name 'j', Line 6, Column 23 0 misspelled words



```
Example.m
12 void foo(int x, int y) {
13     id obj = [[NSString alloc] init];
14
15     switch (x) {
16     case 0:
17         [obj release];
18         break;
19     case 1:
20         // [obj autorelease];
21         break;
22     default:
23         break;
24     }
25 }
```

1 Method returns an Objective-C object with a +1 retain count (owning reference)

2 Control jumps to 'case 1' at line 18

3 Execution jumps to the end of the function

4 Object allocated on line 13 is no longer referenced after this point and has a retain count of +1 (object leaked)

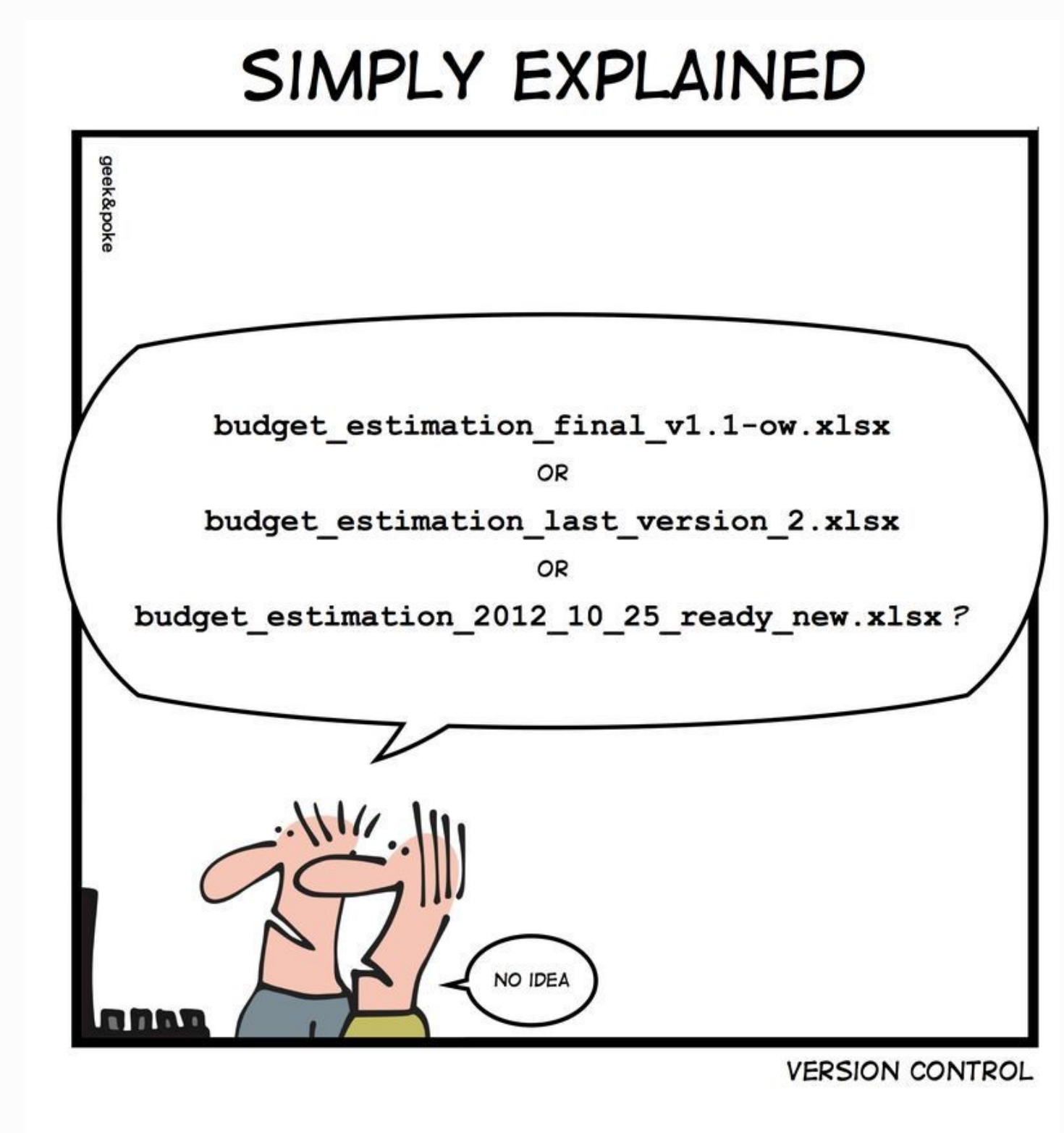
Always track code changes - Revision Control

Don't underestimate the challenge of tracking your code

- Deceitfully simple at the beginning...
 - ▶ e.g.: zip/tar-based backups, versioning and distribution
- but the illusion shatters soon enough
 - ▶ upgrading tools or library, refactoring, rushing-in a patch
 - ▶ “long-range” bugs are a thing, not always immediate to catch

Get familiar with Revision Control early

- **Learn to track (and comment) every code change**
- **RCS Essential (and unavoidable) for collaboration**



Revision Control Software

Once upon a time: CVS and Subversion [“CVS done right”]*

Nowadays: Distributed revision control - Great for personal use

- Easy to work on the go
- Your local copy has everything (including history)

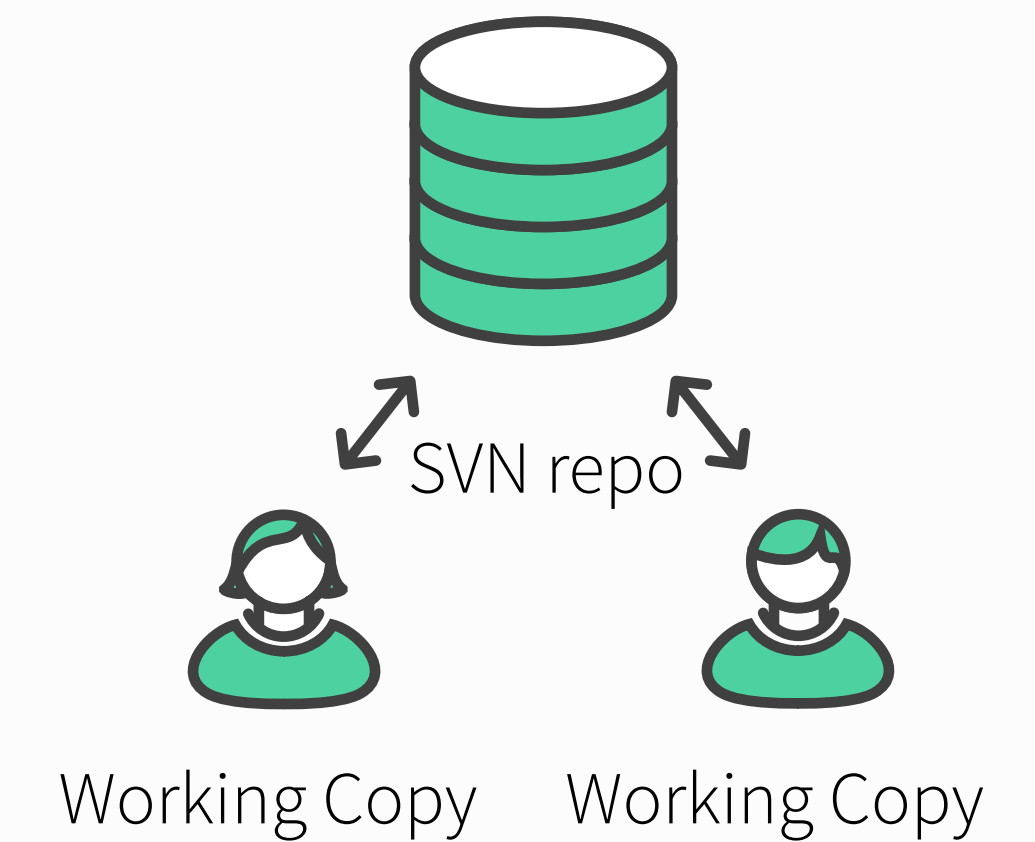
The most popular nowadays **git:** git-scm.com

[“there is no way to do CVS right”]*

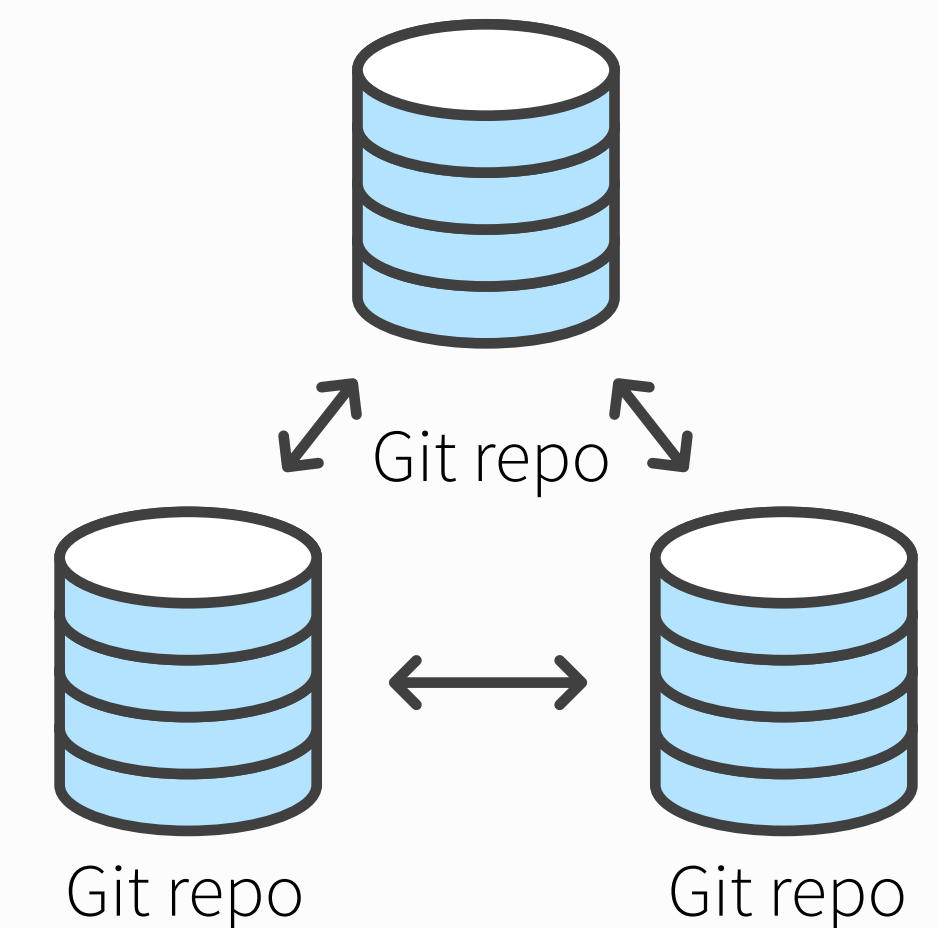
- Other distributed solutions are: Mercurial, bazaar...
- Easy to get started...

* paraphrasing Linus Torvalds

Central-To-Working-Copy Collaboration



Repo-To-Repo Collaboration



Interlude: git basics

```
> git init
Initialized empty Git repository in /TestDirectory/.git/
> vim README.md
skipping this part.
```

Interlude: git basics

```
> git init
Initialized empty Git repository in /TestDirectory/.git/
> vim README.md
skipping this part.
> git add README.md
```

Interlude: git basics

```
> git init
Initialized empty Git repository in /TestDirectory/.git/
> vim README.md
skipping this part.
> git add README.md
> git commit -m "Initial commit of readme."
```

Random github commit messages:
<http://whatthecommit.com/>

Git - in a nutshell

Learn basic concepts and commands

- Create repository, add file, commit new versions and retrieve: **git init, add, commit, checkout**

Familiarise with parallel development concepts

- Branching, merging, rebasing: **git branch, merge, rebase**

Learn how to interact with remote repositories and users

- Retrieve and share code: **git clone, pull, push, fetch**

Not always intuitive

- That's because code management is a **hard problem to solve**
- Worth investing a bit of time reading about it*

Git tutorials:

<http://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>

<http://pcottle.github.io/learnGitBranching/>

*Ultimate git guide: <https://jwiegley.github.io/git-from-the-bottom-up/>



Git - there's more than meets the eye

Create your own `.gitconfig` to get the most out of git

- Colors, aliases, etc...
- Special mention: **graphical history**
 - ▶ a lifesaver when working with many branches/ developers

Git configuration: <https://git-scm.com/docs/git-config>

Integrate git with your shell

- Tab-completion and git information shell prompt
 - ▶ Reduce the risk

Git bash shell integration:

<https://git-scm.com/book/en/v2/Appendix-A%3A-Git-in-Other-Environments-Git-in-Bash>

Magic alias (one of many):

```
[alias]
  lg = log --color --graph --all --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit
```

```
[user]
  name = Alessandro Thea
  email = alessandro.thea@cern.ch
[alias]
  cl = clone
  ci = commit
  co = checkout
  rb = rebase
  st = status -s
  br = branch
[color]
  ui = true
[core]
  editor = vim
```

```
* f62e7c1 (HEAD, master)
| [26 hours ago] user: updated file3
* e8b89cb
| [26 hours ago] user: foo merged
* 8bdfcaf (foo)
| [26 hours ago] user: updated file2
* f0fbbcc
| [26 hours ago] user: updated file1
* 8ed6e3e
| [26 hours ago] user: updated file2
*/
* 44a2925
| [26 hours ago] user: initial
```

```
zephyrus ~/Development/ipbus-up/ipbb > 21f32c7|dev/dune-felix ⚡
10026 ± :
```

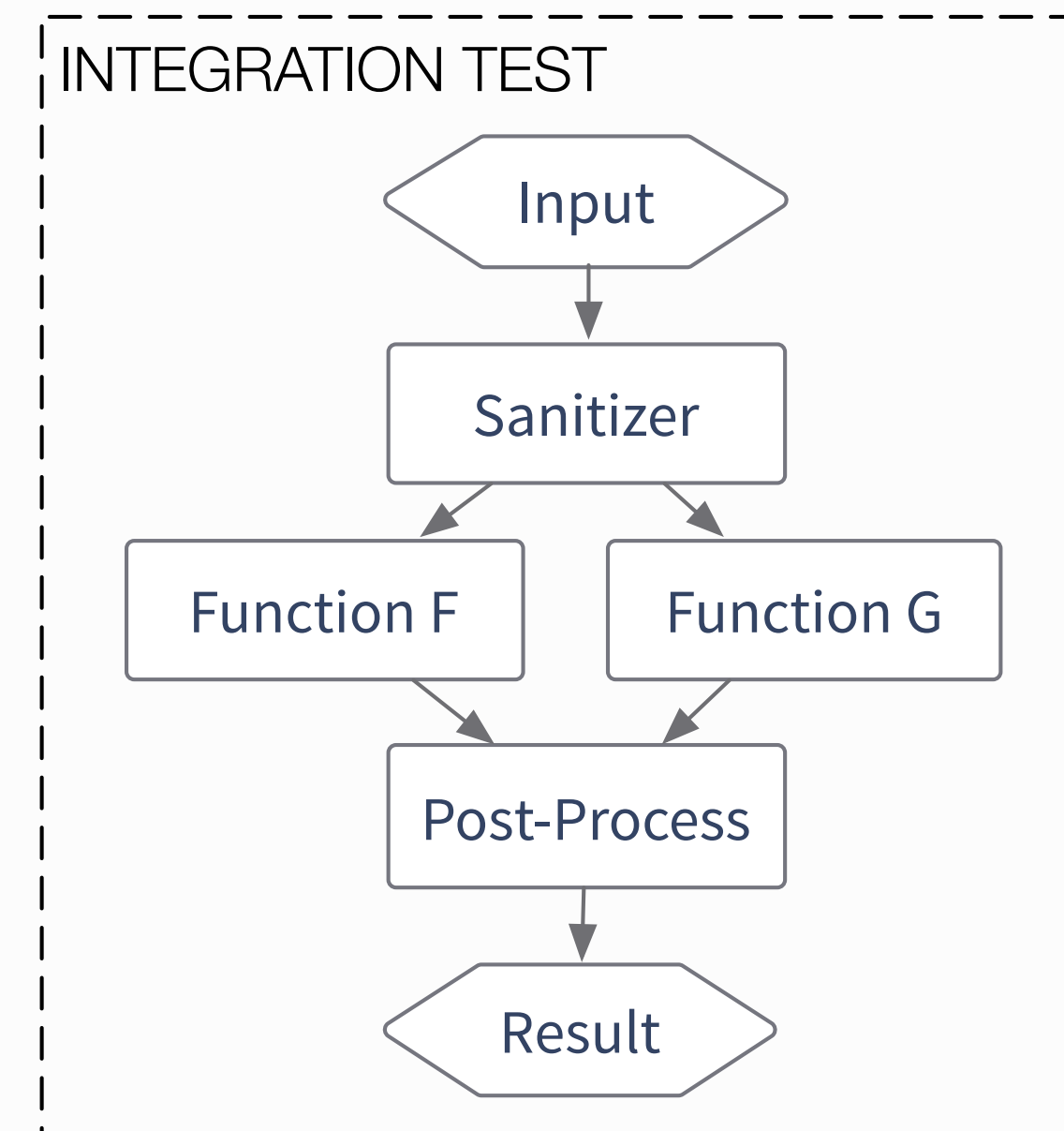
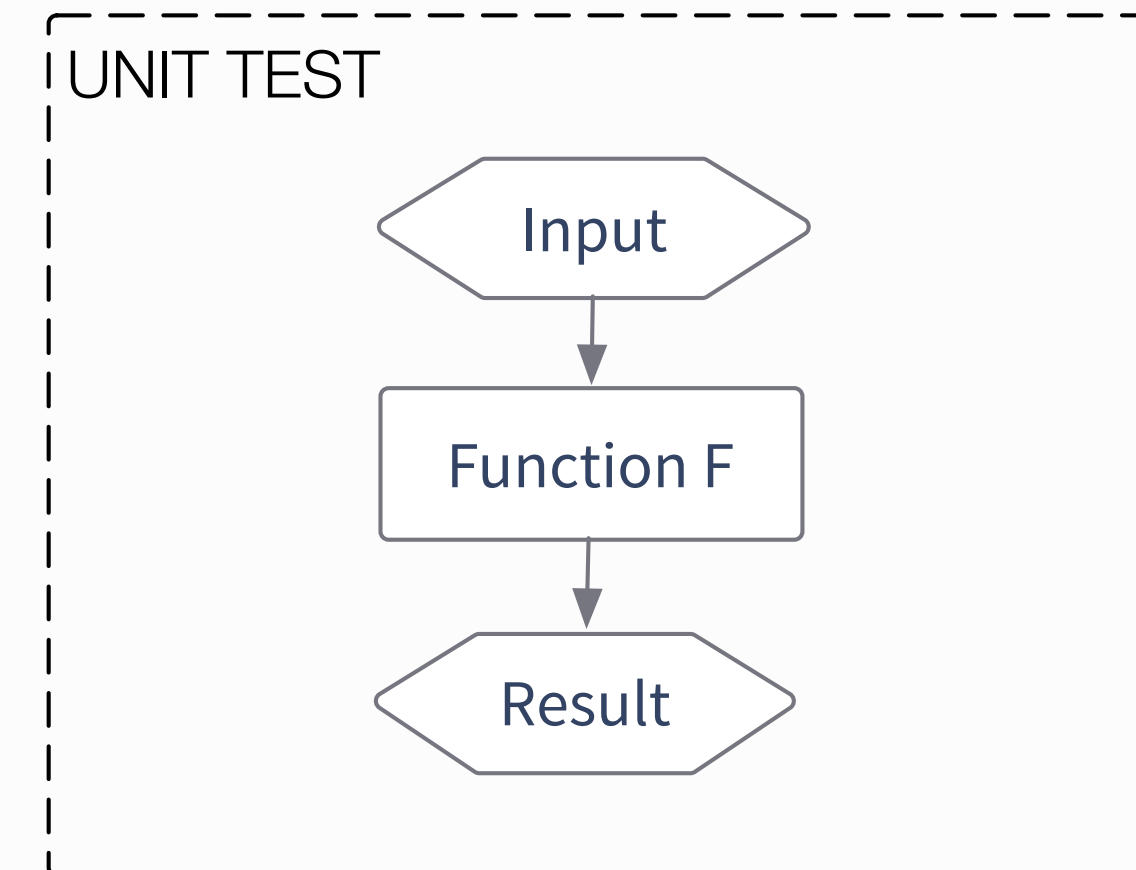

What do we mean with tests?

Different tests, different purposes:

- **Unit test**
 - ▶ Testing “units of code”, e.g. a function or class
 - ▶ Given a defined input => expected output?
- **Integration test**
 - ▶ Testing a larger part of your software
 - ▶ For example running an example and checking output

Do not mix it up with verification

- ▶ Checking if specifications are met



Writing good tests is hard

How to come up with tests?

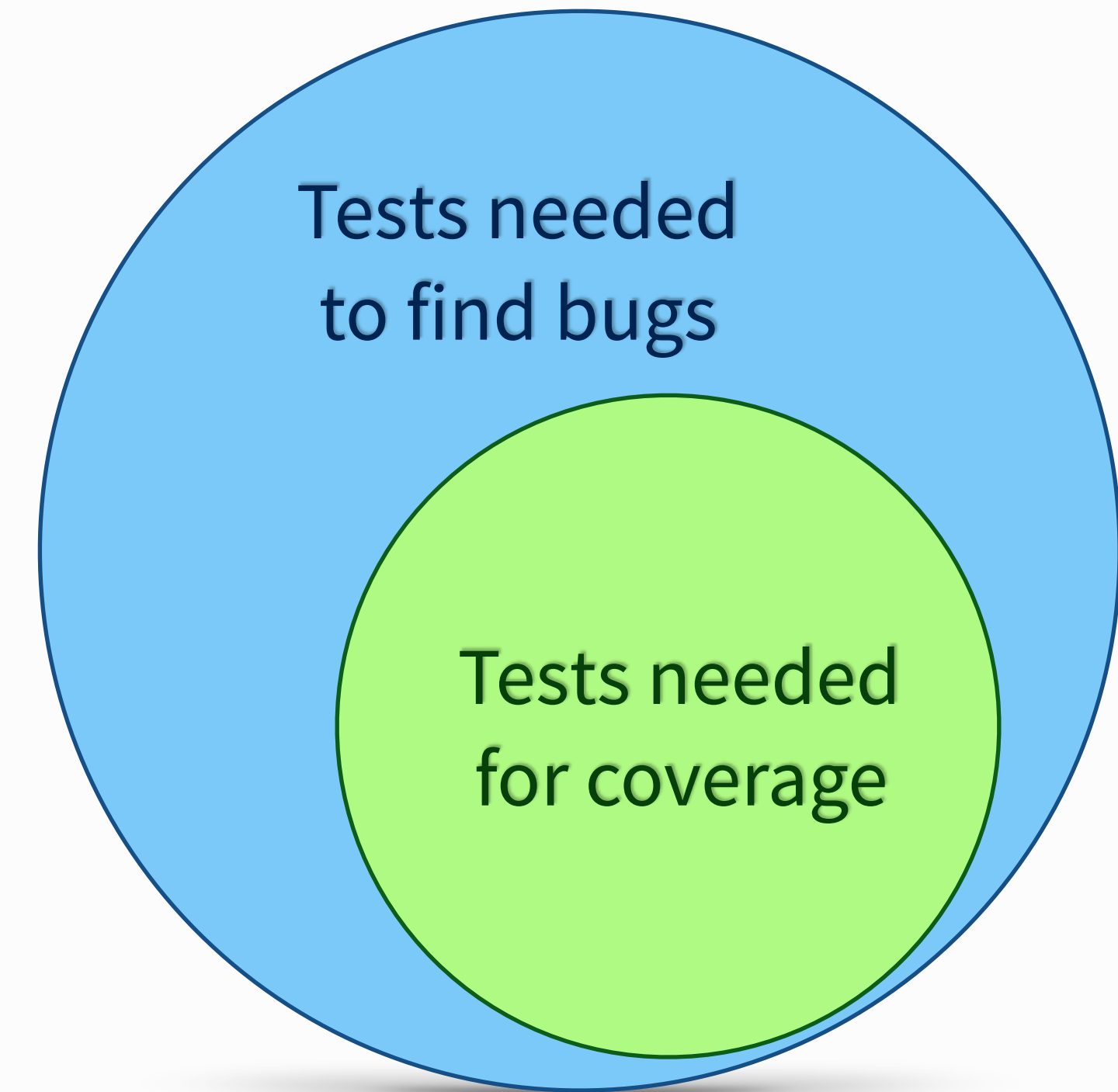
- What should the algorithm do?
 - ▶ Check if well defined input produces correct result
- How should the algorithm fail?
 - ▶ Check if wrong input fails in the way you want

You'll ~~probably~~ miss corner cases

- Once you discover them, implement a test!
 - ▶ **Only let a bug hit you once**
- Have beta-testers / users help you
 - ▶ Use issue tracker
 - ▶ Be responsive!

Look at existing solutions to implement tests

- Python: [doctest](#) and [unittest](#) packages
- C++: [CTest](#) (integrated with cmake) & [Catch](#)



Interlude: doctest

> python testfib.py

```
def fib(n):
    """ Returns the fibonacci series at n
    >>> [fib(n) for n in range(6)]
    [0, 1, 1, 2, 3, 5]
    >>> fib(-1)
    Traceback (most recent call last):
        ...
    ValueError: n should be >= 0
    """
    if n < 0: raise ValueError("n should be >= 0")
    if n == 0: return 0
    a, b = 1, 1
    for i in range(n-1):
        a, b = b, a+b
    return a

import doctest
doctest.testmod()
```

Interlude: doctest

```
> python testfib.py  
>
```

```
def fib(n):  
    """ Returns the fibonacci series at n  
    >>> [fib(n) for n in range(6)]  
    [0, 1, 1, 2, 3, 5]  
    >>> fib(-1)  
    Traceback (most recent call last):  
        ...  
    ValueError: n should be >= 0  
    """  
    if n < 0: raise ValueError("n should be >= 0")  
    if n == 0: return 0  
    a, b = 1, 1  
    for i in range(n-1):  
        a, b = b, a+b  
    return a  
  
import doctest  
doctest.testmod()
```

Interlude: doctest

> python testfib.py -v

```
def fib(n):
    """ Returns the fibonacci series at n
    >>> [fib(n) for n in range(6)]
    [0, 1, 1, 2, 3, 5]
    >>> fib(-1)
    Traceback (most recent call last):
        ...
    ValueError: n should be >= 0
    """
    if n < 0: raise ValueError("n should be >= 0")
    if n == 0: return 0
    a, b = 1, 1
    for i in range(n-1):
        a, b = b, a+b
    return a

import doctest
doctest.testmod()
```

Interlude: doctest

```
> python testfib.py -v
> Trying:
>     [fib(n) for n in range(6)]
> Expecting:
>     [0, 1, 1, 2, 3, 5]
> ok
> Trying:
>     fib(-1)
> Expecting:
>     Traceback (most recent call last):
>         ...
>     ValueError: n should be >= 0
> ok
```

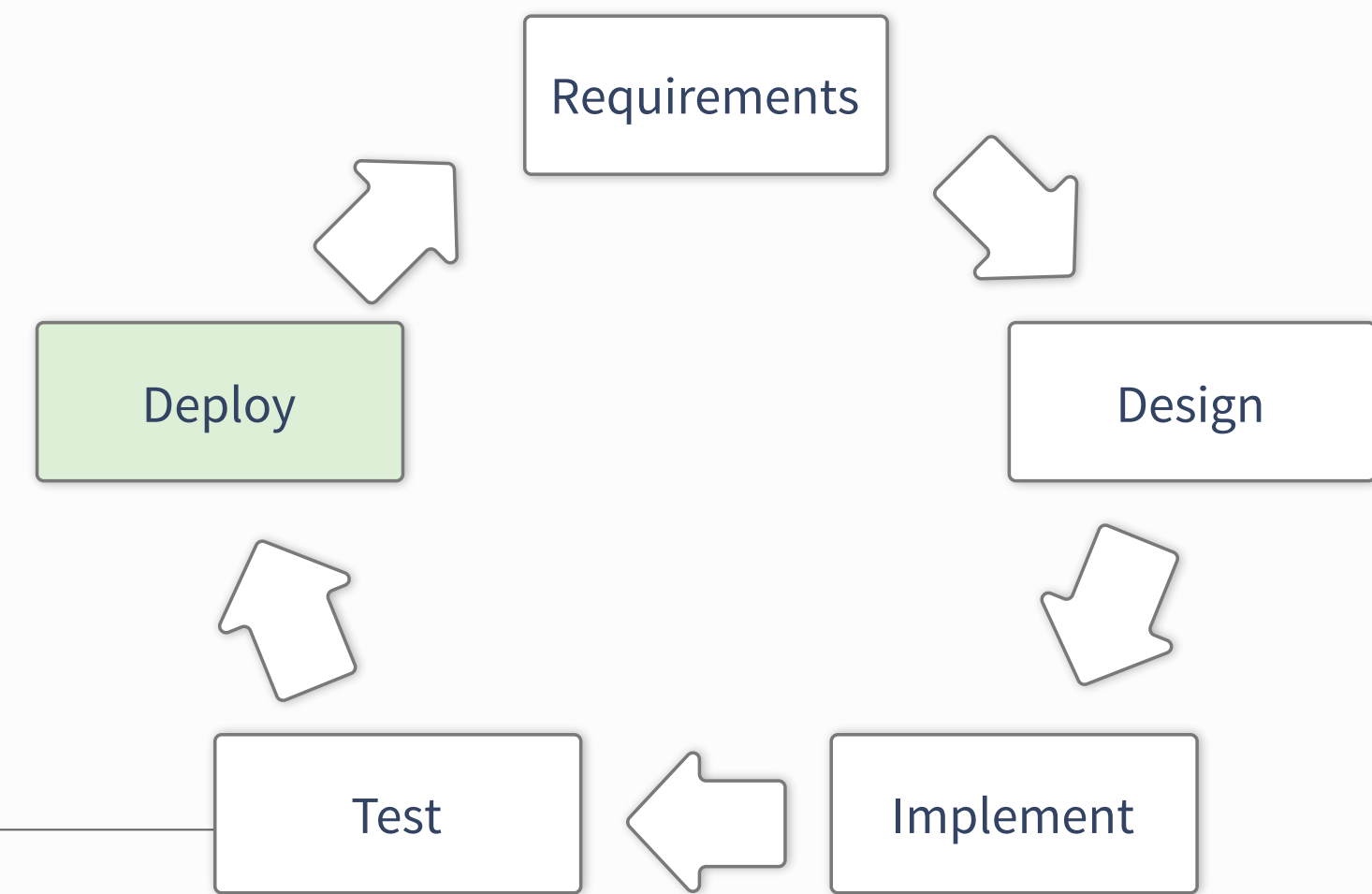
```
def fib(n):
    """ Returns the fibonacci series at n
    >>> [fib(n) for n in range(6)]
    [0, 1, 1, 2, 3, 5]
    >>> fib(-1)
    Traceback (most recent call last):
        ...
    ValueError: n should be >= 0
    """
    if n < 0: raise ValueError("n should be >= 0")
    if n == 0: return 0
    a, b = 1, 1
    for i in range(n-1):
        a, b = b, a+b
    return a

import doctest
doctest.testmod()
```


Test your software

and not just in production!

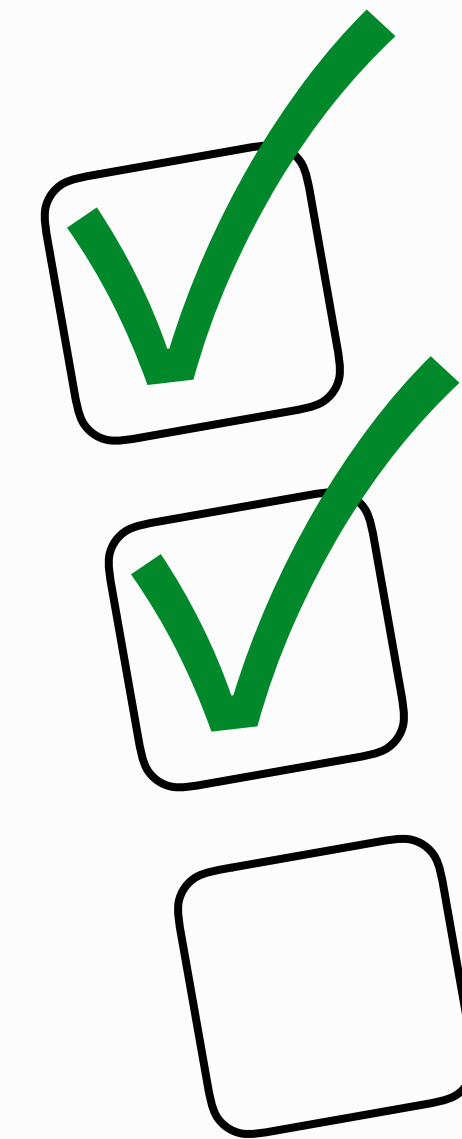
Deploying your software



Releasing the Software

When you release your software:

- Tag the repository
 - ▶ Ensure everyone has the same code
- Test in the target environment
 - ▶ Fresh virtual machine
- Accompanying documentation
 - ▶ Produce Doxygen pages
 - ▶ Update wikis (new version)
 - ▶ Make sure all examples work



Ideal case: All this is done for every commit!

Continuous integration

Working in groups on software can be hard:

- Somebody changes something: Other code breaks
- **Avoid such nuisances by testing regularly!**

New contribution to the code base:

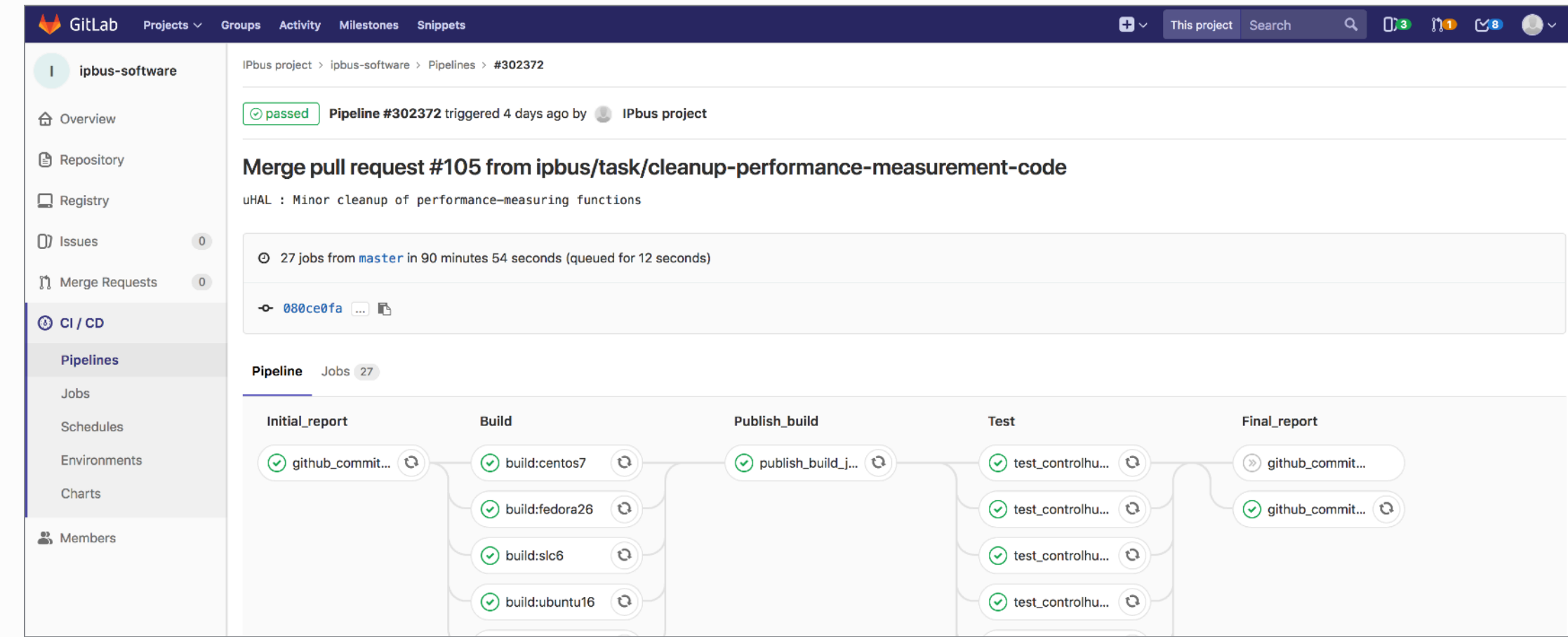
- Check everything works
 - ▶ Can do this by hand... Tedious
 - ▶ Better: Automate it.

Many solutions exist that automatically test things:

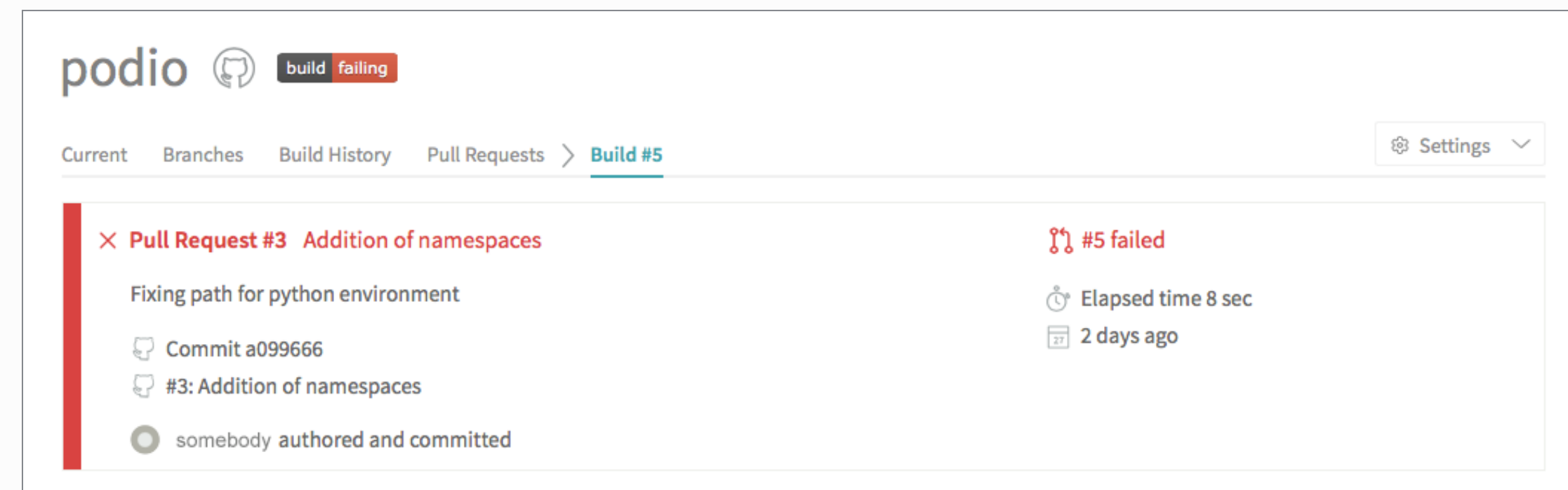
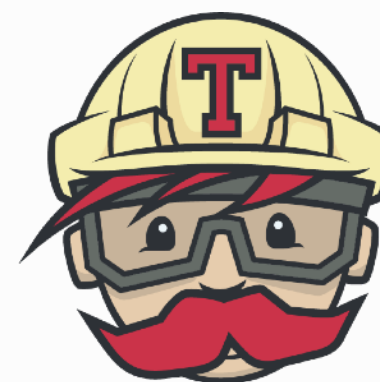
- Check compilation
- Check all defined test cases
- Write nice summaries



CI/CD

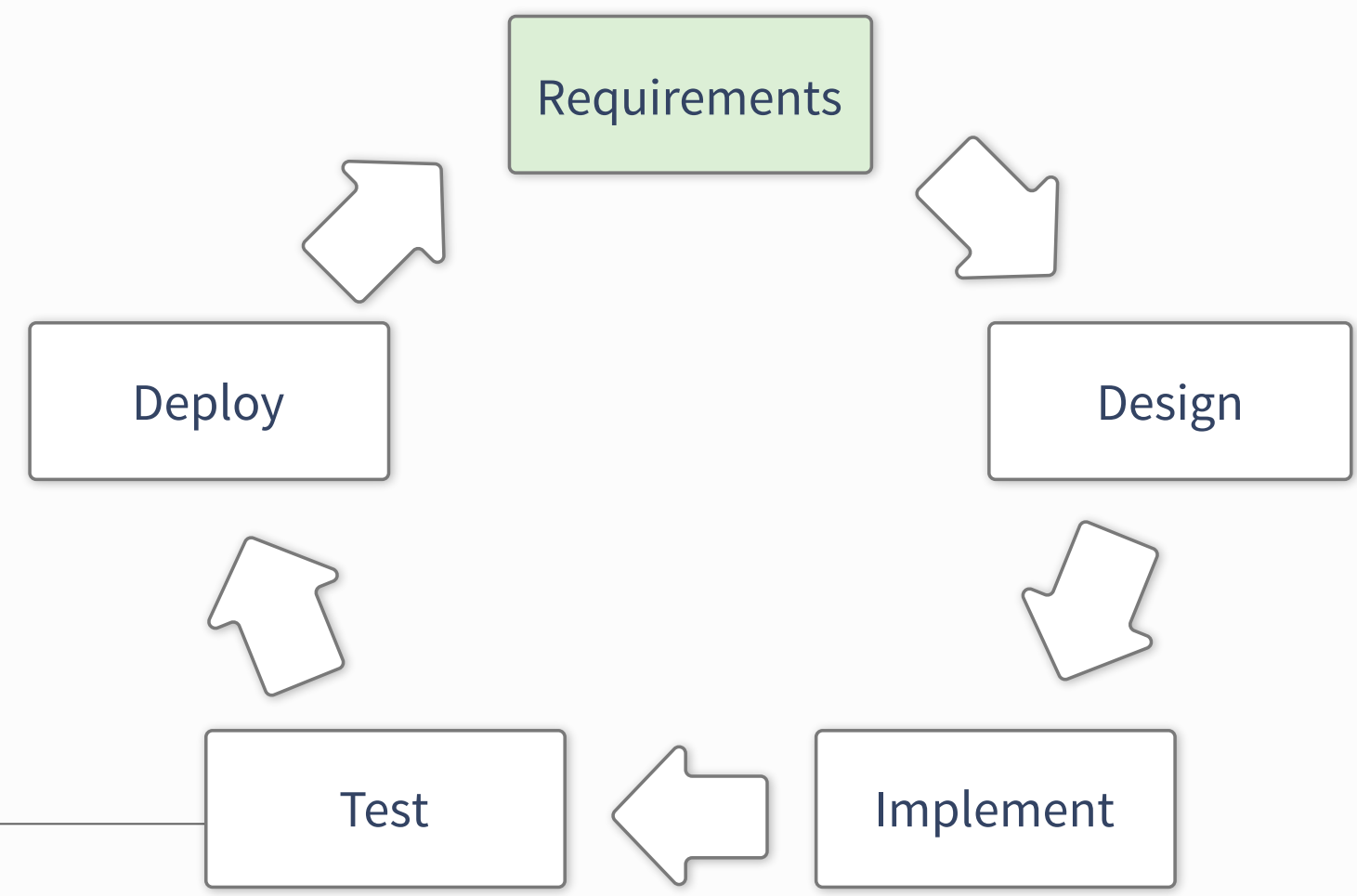


Gitlab CI - <https://about.gitlab.com/features/gitlab-ci-cd/>

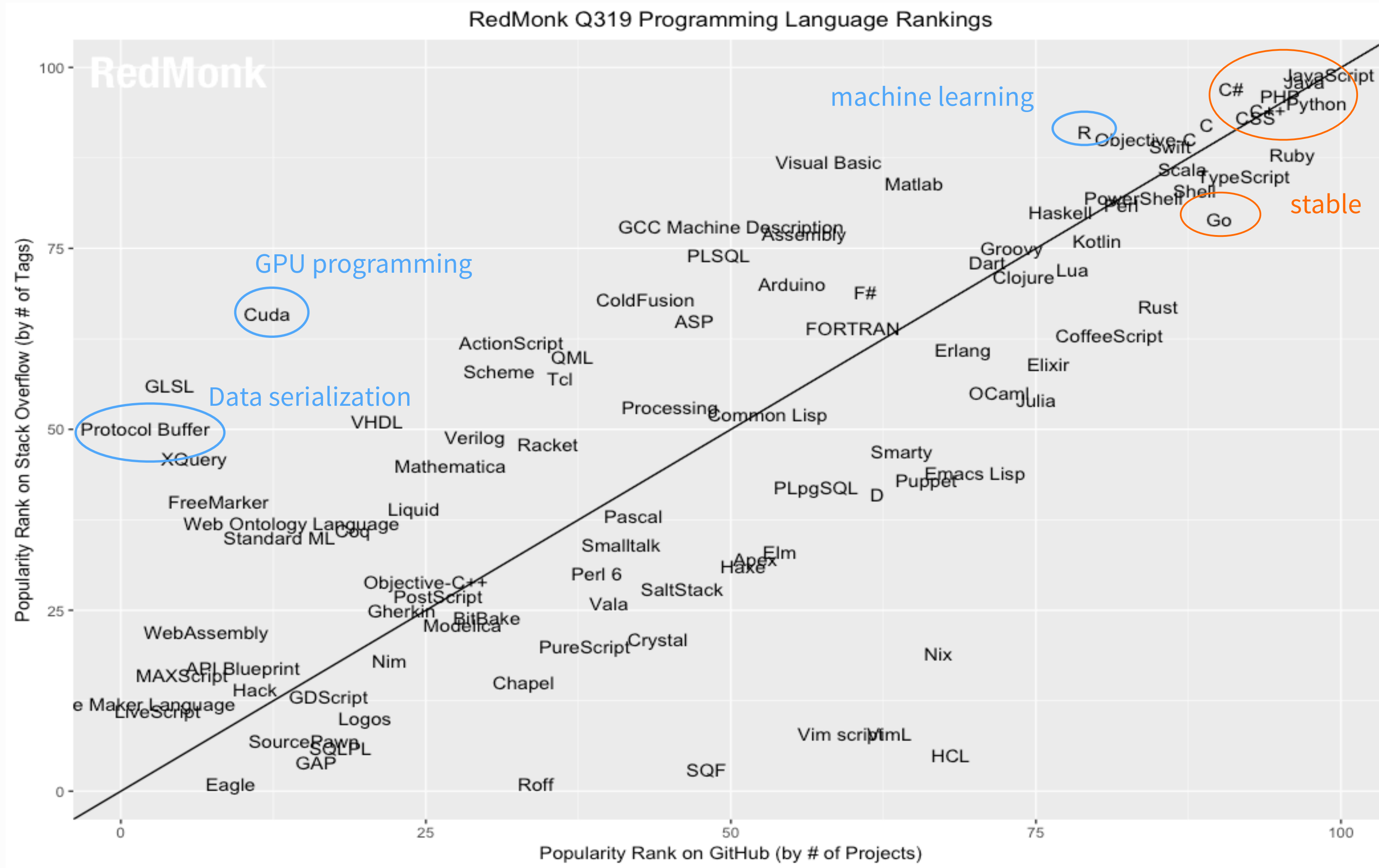


Travis CI - <https://travis-ci.org>

Requirements

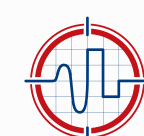


Choosing the programming language

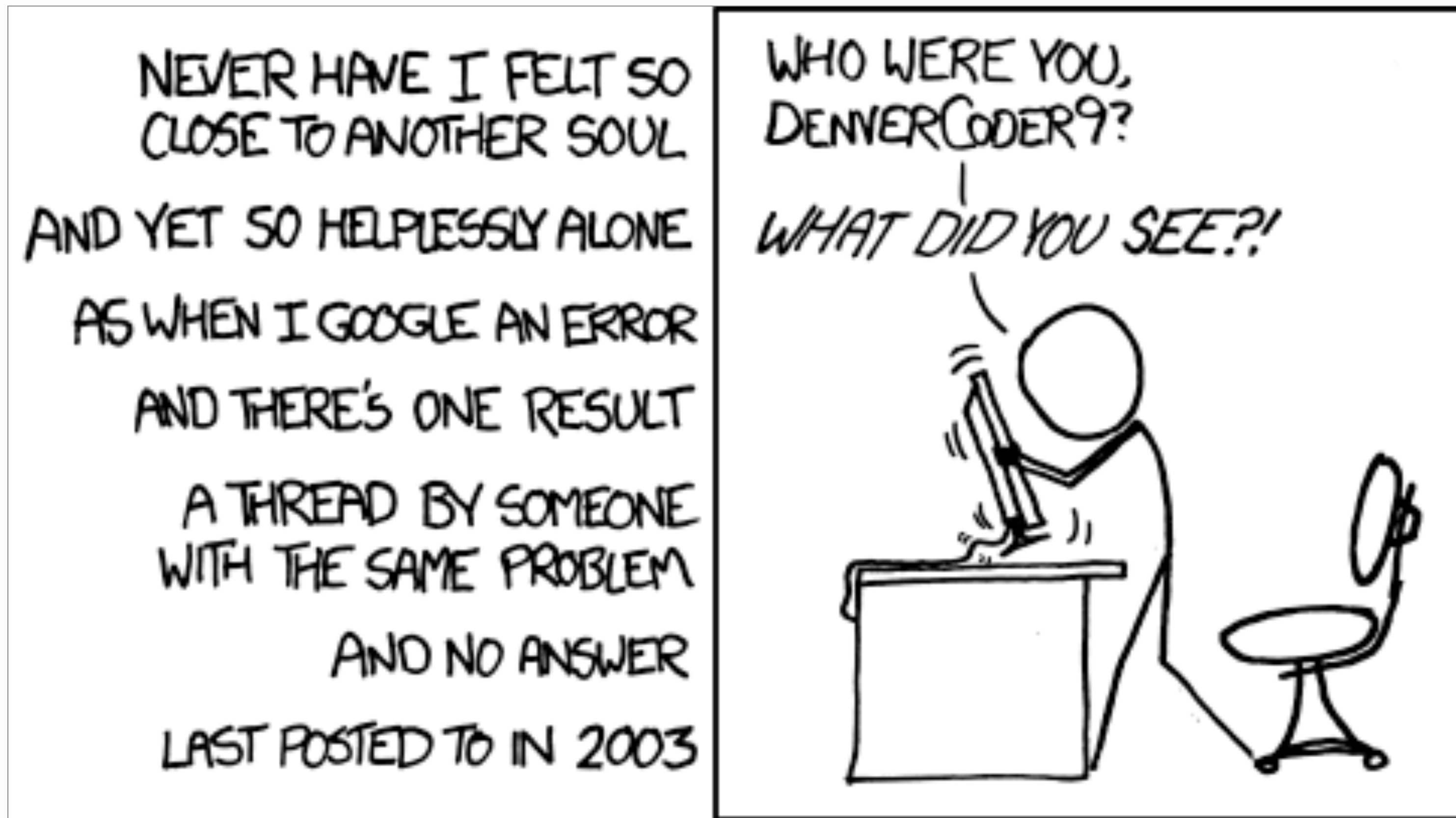


The answer depends:

- Analysis?
- DAQ / Trigger?
- External conditions?
 - Can you choose?



Choosing the programming language



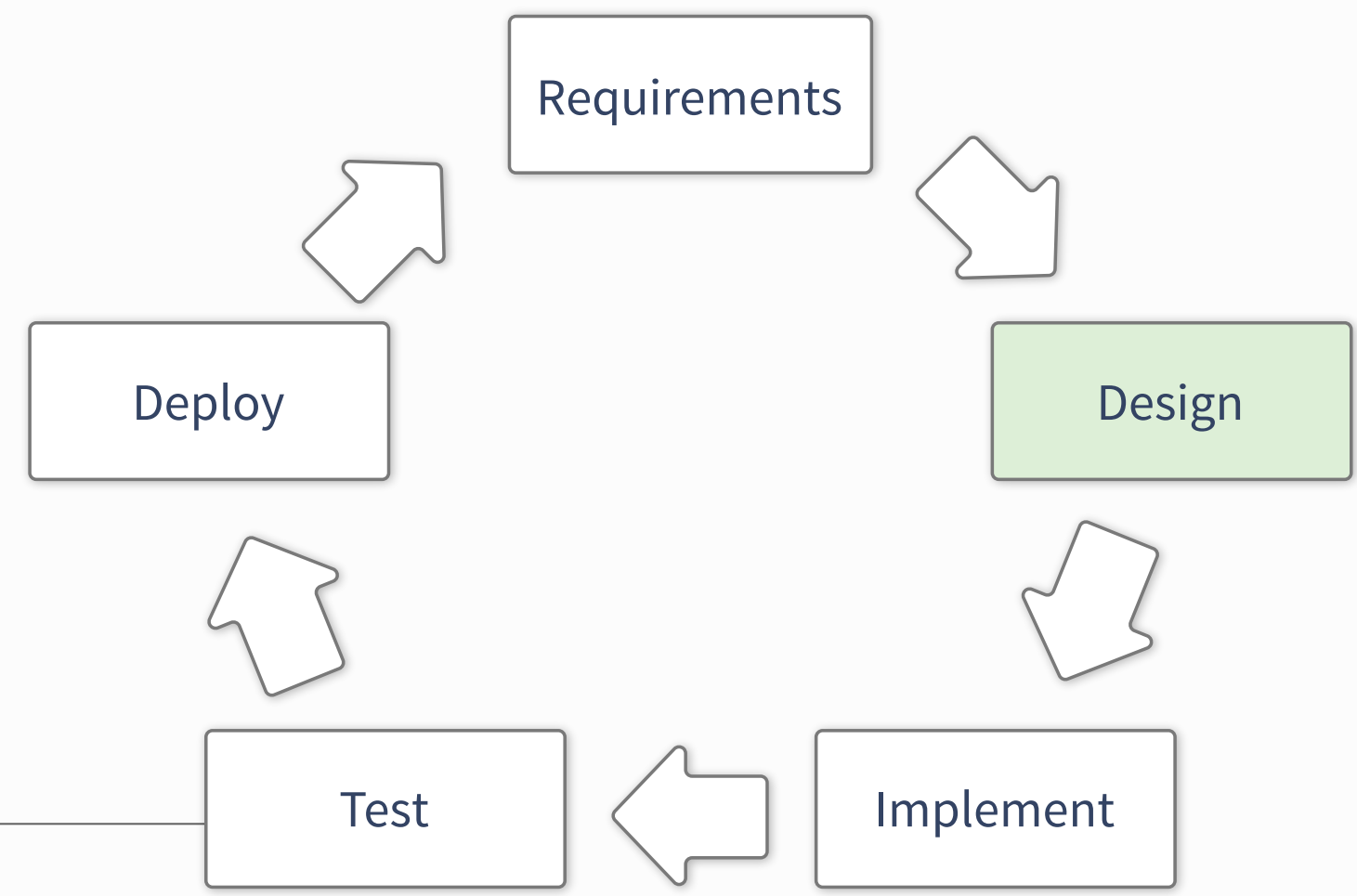
Choose wisely

- Favour documentation and support over features
- Favour large user-bases

Do you really
have to program?

Or has somebody already done it for you?

Design



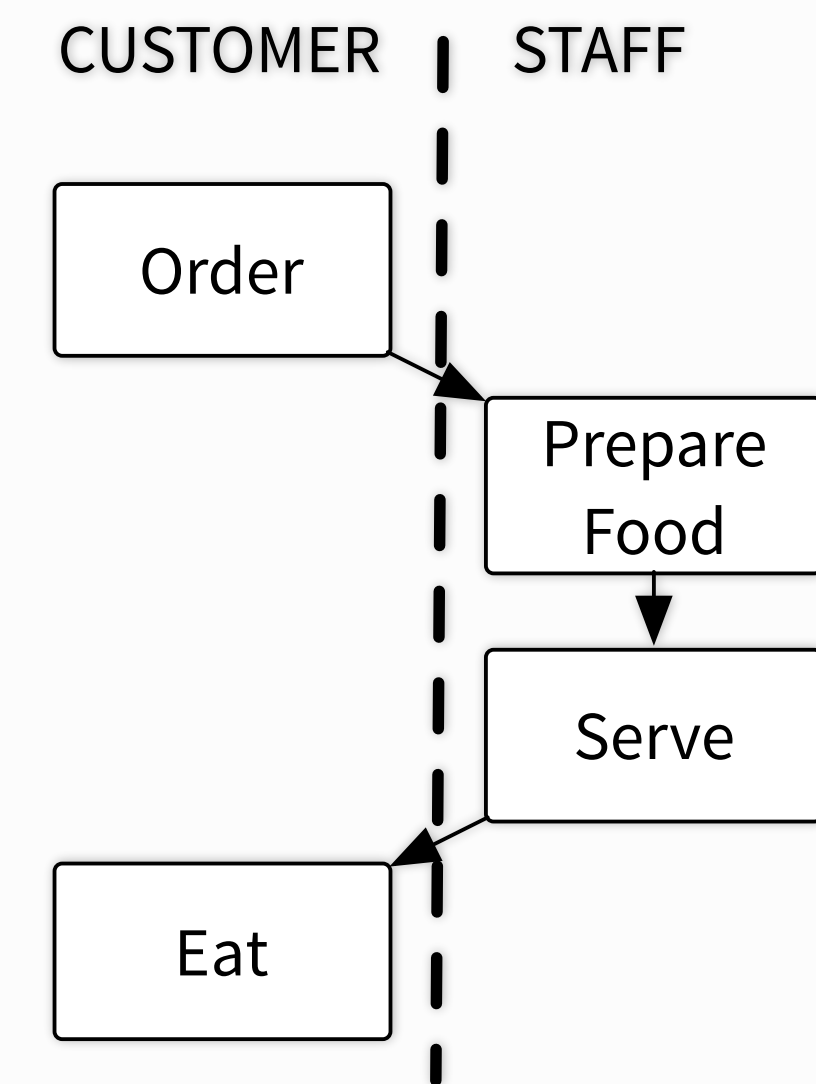
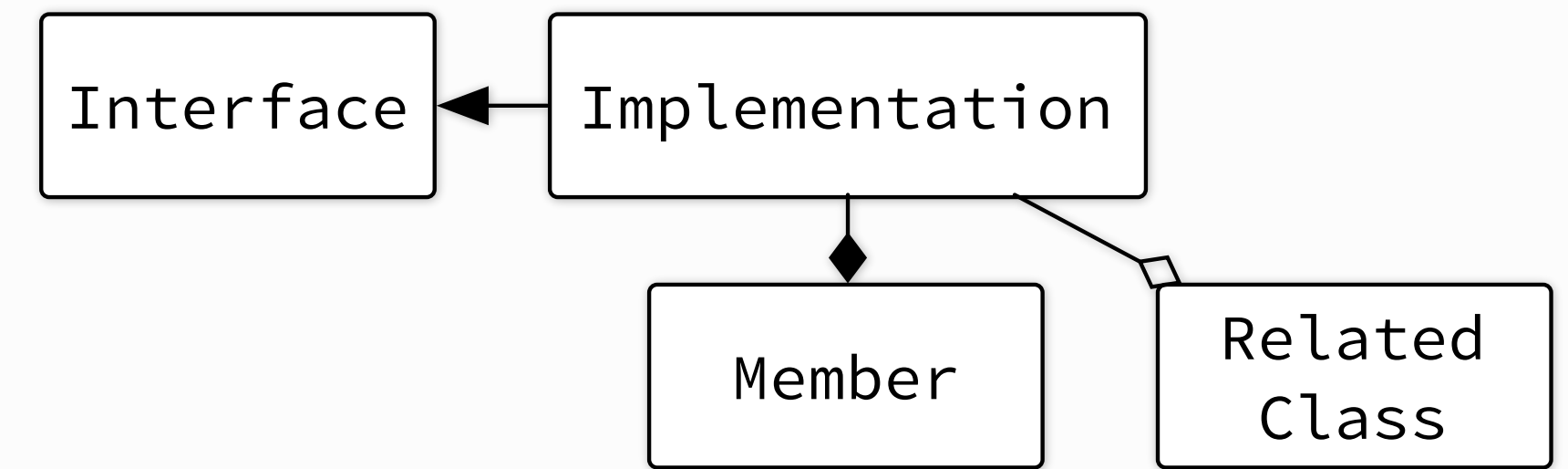
UML Diagrams

Unified Modelling Language: sketch a design

- Probably everyone has seen structure diagrams
 - ▶ Relationships of classes (or larger components)
- Behaviour diagrams
 - ▶ What does the user do and what should be the result?
- Interaction diagrams
 - ▶ How does data and control flow?

Forces you to be concrete!

to make them, look at draw.io or lucidchart.com



Things to keep in mind when designing

Maintainability

- Is it easy to adapt to changed environment?
- Can you cope with (slightly) changed requirements?

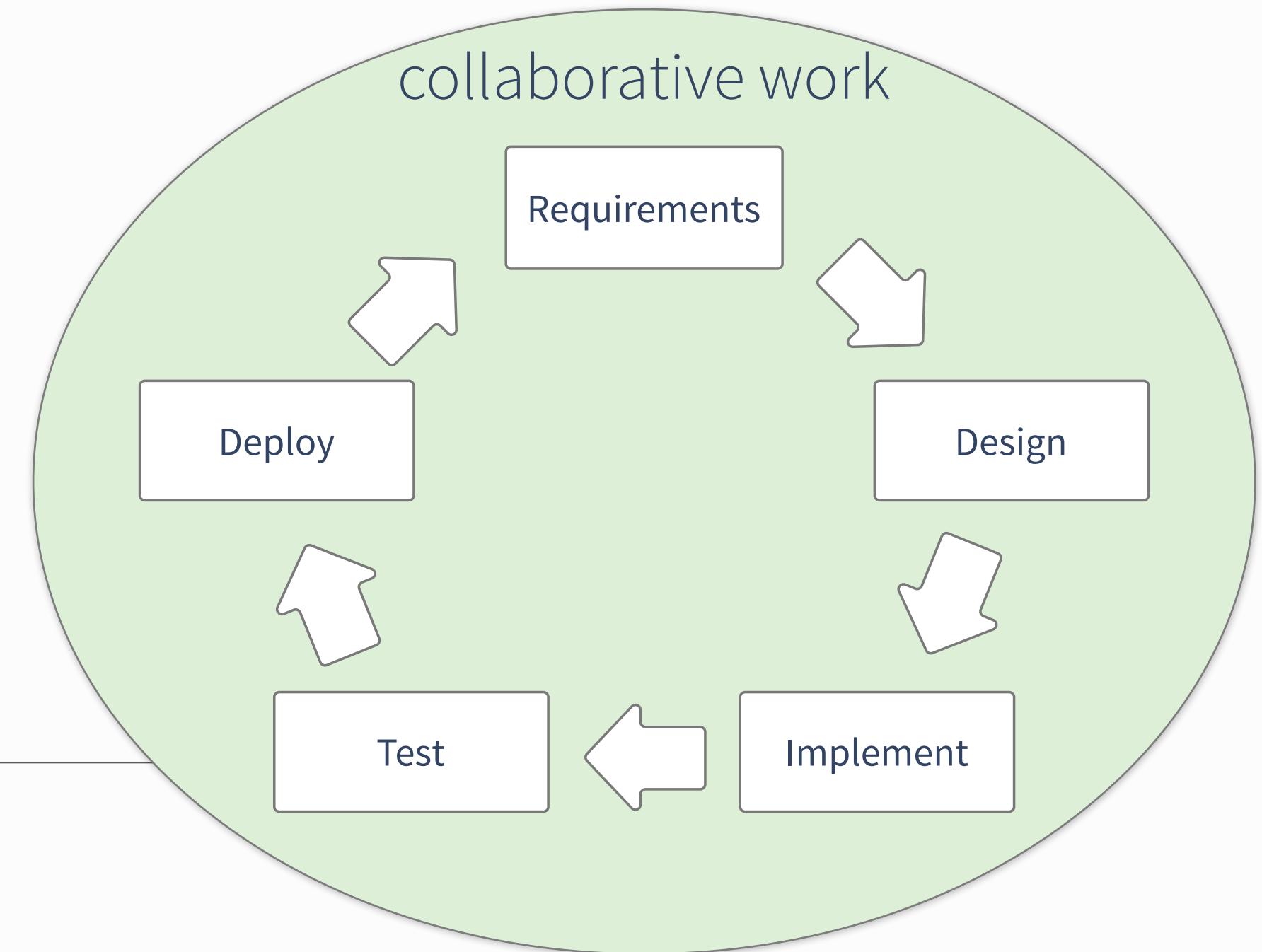
Scalability

- Large data volumes
 - ▶ Think about data-flow and data layout
 - ▶ Try to avoid complicated data structures

Re-usability

- Identify parts of the design that could be used elsewhere
- Could these be extracted in a dedicated library?

Collaborative programming



Development Cycles

Developing software efficiently:

- Avoid duplication of work
- Avoid feature bloating
- Ensure code quality
- Deliver code timely

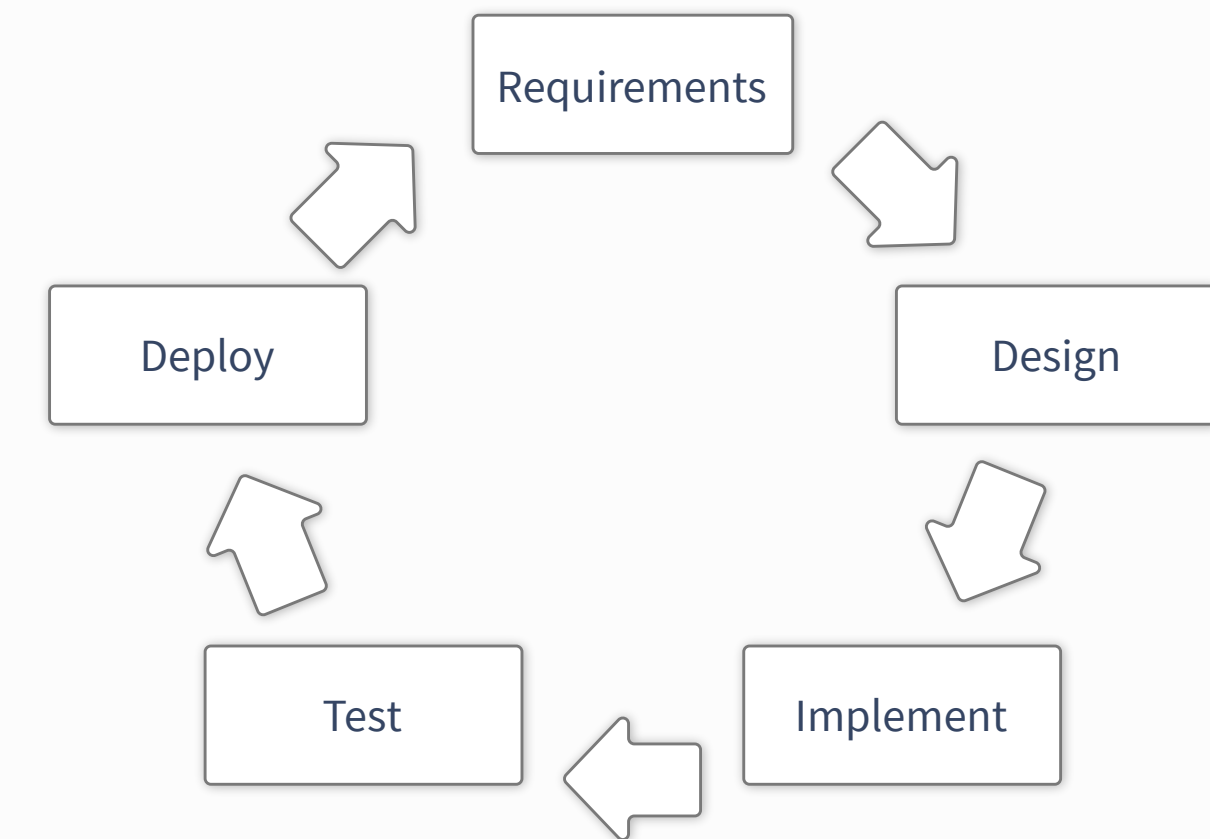
Many approaches to accomplish this:

- Examples: Iterative and Test-Driven Development

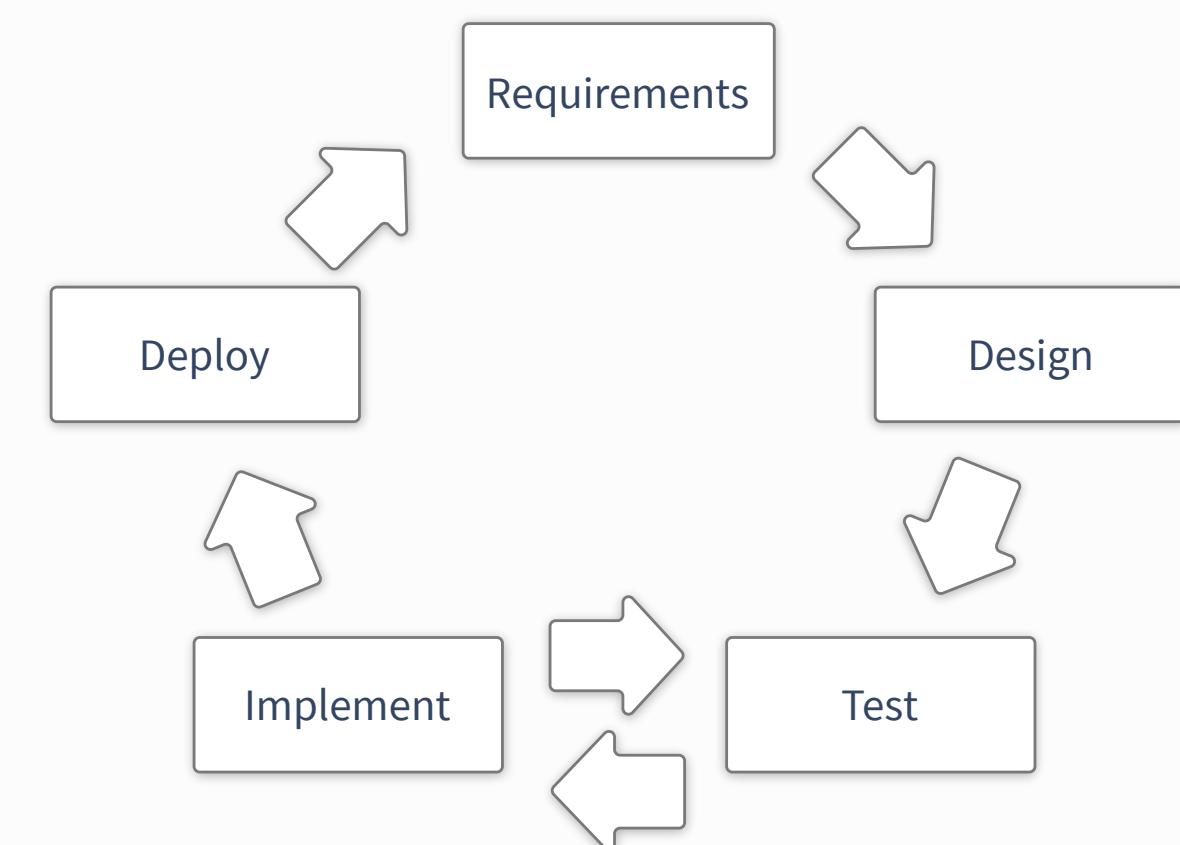
Similar principles, different focus

- on team management (agile development)
- on actual programming style (lean development / TDD)
- broad guidelines to deliver (iterative development)

Iterative Development



Test-Driven Development



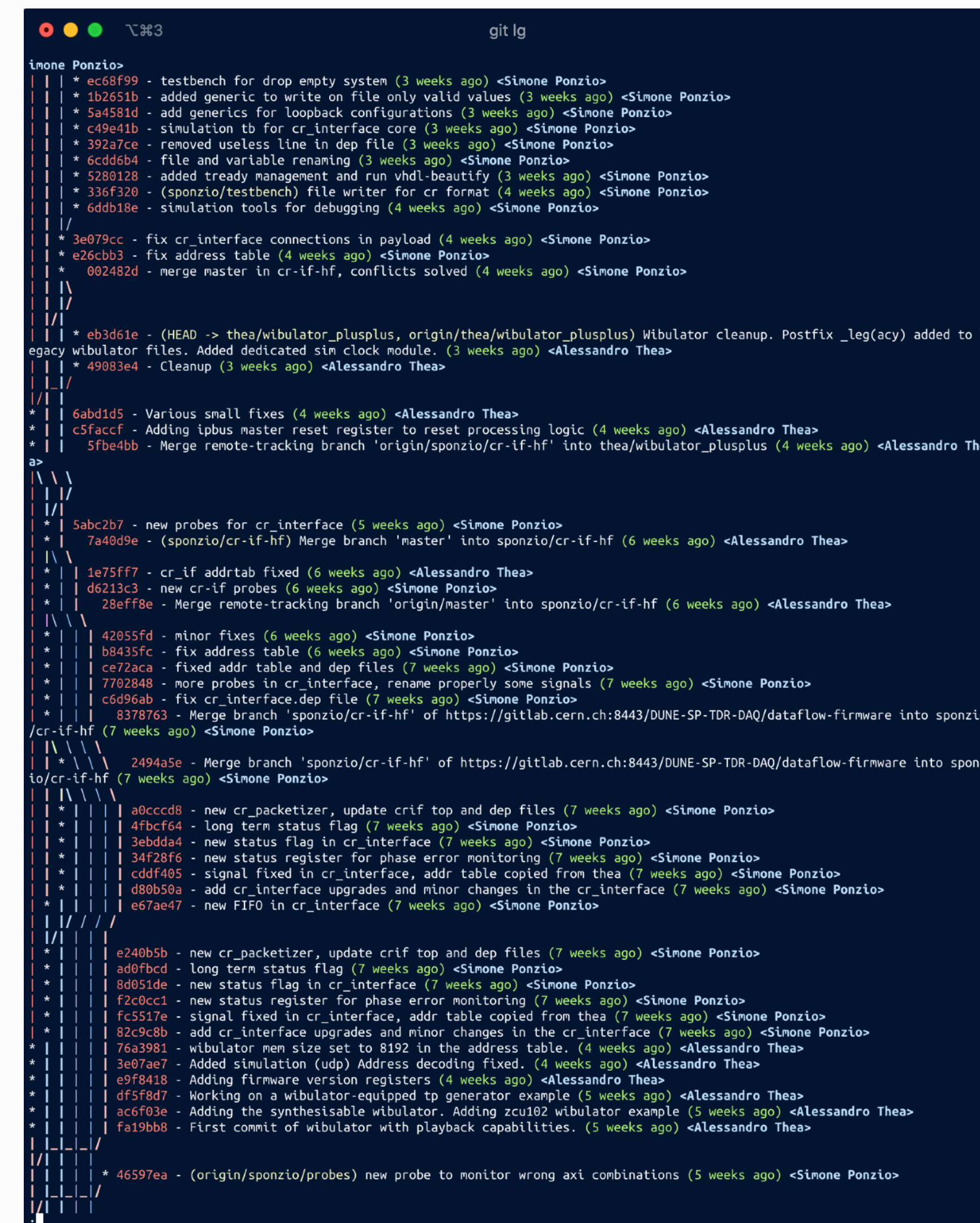
Git & collaborative programming

Sometimes git command line is just not enough

- Multiple developers, many branches, parallel developments
- Non-trivial merge conflicts
- Even mid-sized project become hard to navigate

Git GUIs can make a significant difference when in trouble

- SourceTree, Git-Kraken, Sublime Merge, Git Up

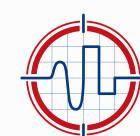


```
git lg
Simone Ponzio>
* ec68f99 - testbench for drop empty system (3 weeks ago) <Simone Ponzio>
* 1b2651b - added generic to write on file only valid values (3 weeks ago) <Simone Ponzio>
* 5a4581d - add generics for loopback configurations (3 weeks ago) <Simone Ponzio>
* c49e41b - simulation tb for cr_interface core (3 weeks ago) <Simone Ponzio>
* 392a7ce - removed useless line in dep file (3 weeks ago) <Simone Ponzio>
* 6cdd6b4 - file and variable renaming (3 weeks ago) <Simone Ponzio>
* 5280128 - added tready management and run vhdl-beautify (3 weeks ago) <Simone Ponzio>
* 336f320 - (sponzio/testbench) file writer for cr format (4 weeks ago) <Simone Ponzio>
* 6ddb18e - simulation tools for debugging (4 weeks ago) <Simone Ponzio>
* 3e079cc - fix cr_interface connections in payload (4 weeks ago) <Simone Ponzio>
* e26cb3 - fix address table (4 weeks ago) <Simone Ponzio>
* 002482d - merge master in cr-if-hf, conflicts solved (4 weeks ago) <Simone Ponzio>
* eb3d61e - (HEAD -> thea/wibulator_plusplus, origin/thea/wibulator_plusplus) Wibulator cleanup. Postfix _leg(acy) added to legacy wibulator files. Added dedicated sim clock module. (3 weeks ago) <Alessandro Thea>
* 49083e4 - Cleanup (3 weeks ago) <Alessandro Thea>
* 6abdd5 - Various small fixes (4 weeks ago) <Alessandro Thea>
* c5faccf - Adding ipbus master reset register to reset processing logic (4 weeks ago) <Alessandro Thea>
* 5f8e4bb - Merge remote-tracking branch 'origin/sponzio/cr-if-hf' into thea/wibulator_plusplus (4 weeks ago) <Alessandro Thea>
* 5abc2b7 - new probes for cr_interface (5 weeks ago) <Simone Ponzio>
* 7a4099e - (sponzio/cr-if-hf) Merge branch 'master' into sponzio/cr-if-hf (6 weeks ago) <Alessandro Thea>
* 1e75ff7 - cr_if addrtab fixed (6 weeks ago) <Alessandro Thea>
* d6213c3 - new cr-if probes (6 weeks ago) <Simone Ponzio>
* 28eff8e - Merge remote-tracking branch 'origin/master' into sponzio/cr-if-hf (6 weeks ago) <Alessandro Thea>
* 42055fd - minor fixes (6 weeks ago) <Simone Ponzio>
* b8435fc - fix address table (6 weeks ago) <Simone Ponzio>
* ce72aca - fixed addr table and dep files (7 weeks ago) <Simone Ponzio>
* 7702848 - more probes in cr_interface, rename properly some signals (7 weeks ago) <Simone Ponzio>
* c6d96ab - fix cr_interface.dep file (7 weeks ago) <Simone Ponzio>
* 8378763 - Merge branch 'sponzio/cr-if-hf' of https://gitlab.cern.ch:8443/DUNE-SP-TDR-DAQ/dataflow-firmware into sponzio/cr-if-hf (7 weeks ago) <Simone Ponzio>
* 2494a5e - Merge branch 'sponzio/cr-if-hf' of https://gitlab.cern.ch:8443/DUNE-SP-TDR-DAQ/dataflow-firmware into sponzio/cr-if-hf (7 weeks ago) <Simone Ponzio>
* a0cccd8 - new cr_packetizer, update crif top and dep files (7 weeks ago) <Simone Ponzio>
* 4fbcf64 - long term status flag (7 weeks ago) <Simone Ponzio>
* 3ebdda4 - new status flag in cr_interface (7 weeks ago) <Simone Ponzio>
* 34f28f6 - new status register for phase error monitoring (7 weeks ago) <Simone Ponzio>
* cddf405 - signal fixed in cr_interface, addr table copied from thea (7 weeks ago) <Simone Ponzio>
* d80b50a - add cr_interface upgrades and minor changes in the cr_interface (7 weeks ago) <Simone Ponzio>
* e67ae47 - new FIFO in cr_interface (7 weeks ago) <Simone Ponzio>
* e240b5b - new cr_packetizer, update crif top and dep files (7 weeks ago) <Simone Ponzio>
* ad0fbcd - long term status flag (7 weeks ago) <Simone Ponzio>
* 8d051de - new status flag in cr_interface (7 weeks ago) <Simone Ponzio>
* f2c0cc1 - new status register for phase error monitoring (7 weeks ago) <Simone Ponzio>
* fc5517e - signal fixed in cr_interface, addr table copied from thea (7 weeks ago) <Simone Ponzio>
* 82c9c8b - add cr_interface upgrades and minor changes in the cr_interface (7 weeks ago) <Simone Ponzio>
* 76a3981 - wibulator mem size set to 8192 in the address table. (4 weeks ago) <Alessandro Thea>
* 3e07ae7 - Added simulation (udp) Address decoding fixed. (4 weeks ago) <Alessandro Thea>
* e9f8418 - Adding firmware version registers (4 weeks ago) <Alessandro Thea>
* df5f8d7 - Working on a wibulator-equipped tp generator example (5 weeks ago) <Alessandro Thea>
* ac6f03e - Adding the synthesisable wibulator. Adding zcu102 wibulator example (5 weeks ago) <Alessandro Thea>
* fa19bb8 - First commit of wibulator with playback capabilities. (5 weeks ago) <Alessandro Thea>
* 46597ea - (origin/sponzio/probes) new probe to monitor wrong axi combinations (5 weeks ago) <Simone Ponzio>
```

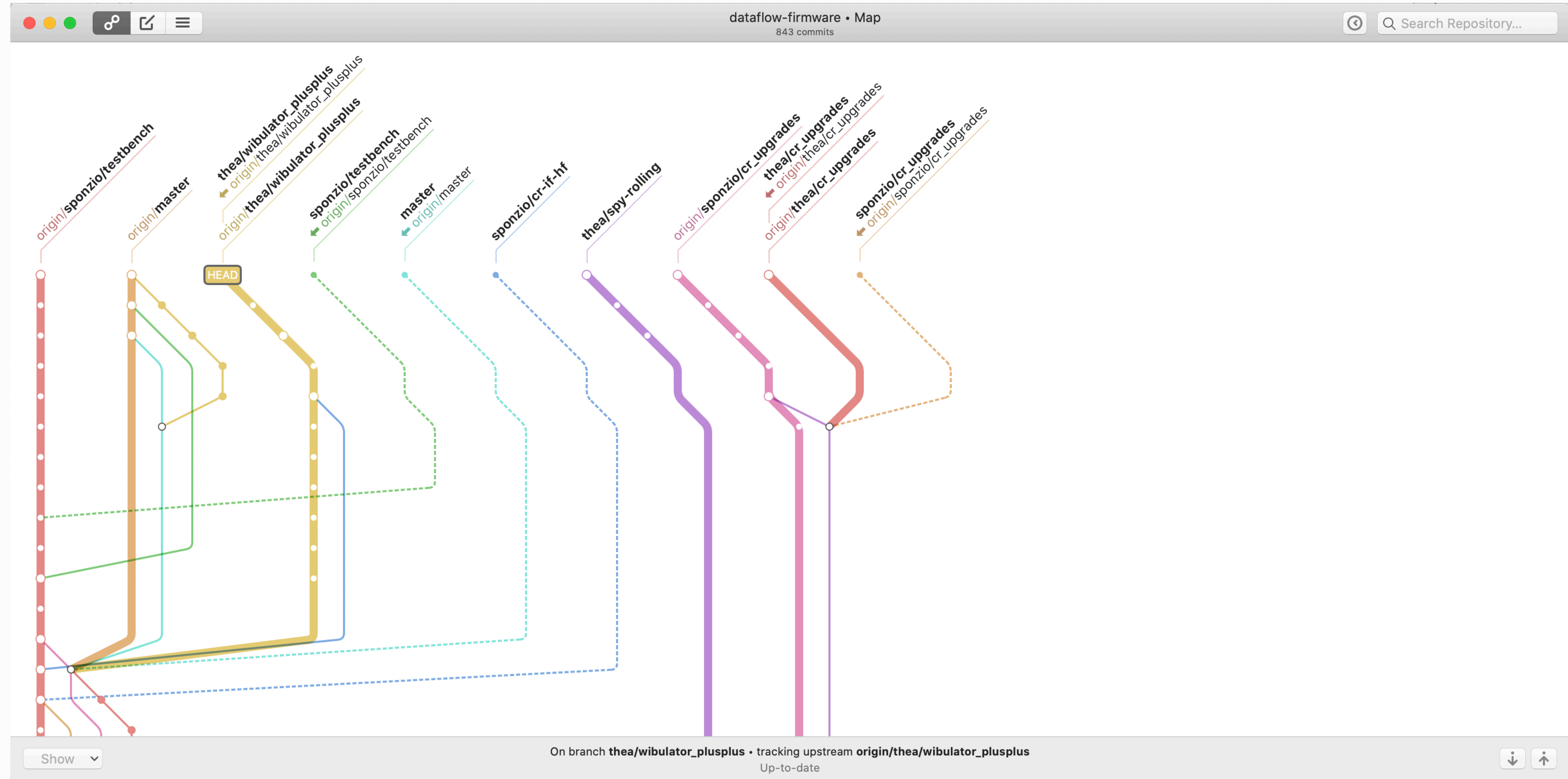
Example : Sublime Merge - repository overview

The screenshot displays the Sublime Merge interface for the repository `~/Development/dune/dataflow/dataflow-firmware`. The current branch is `thea/wibulator_plusplus`. The interface is divided into several sections:

- Left Panel (Locations):** Shows a tree view of branches and remotes. The `thea` branch is selected, with `wibulator_plusplus` as the current branch.
- Commit History:** A list of recent commits with their descriptions, authors, and dates. The most recent commit is "Merge branch 'sponzio/cr-if-hf' into 'master'" by Alessandro Thea.
- Commit Details:** A detailed view of the selected commit, showing the commit hash, tree hash, author, date, parent, and branches. The commit message is "Wibulator cleanup. Postfix _leg(acy) added to legacy wibula...".
- Diff View:** A comparison of the selected commit against its parent, showing changes in files like `components/wibulator/firmware/cfg/wibulator.dep` and `components/wibulator/firmware/cfg/wibulator_2g.dep`.



Another example : Git Up - focus on branch structure



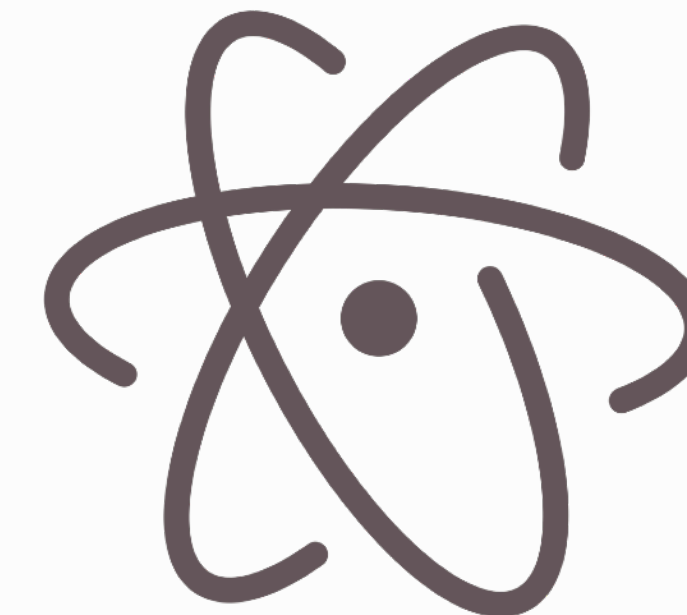
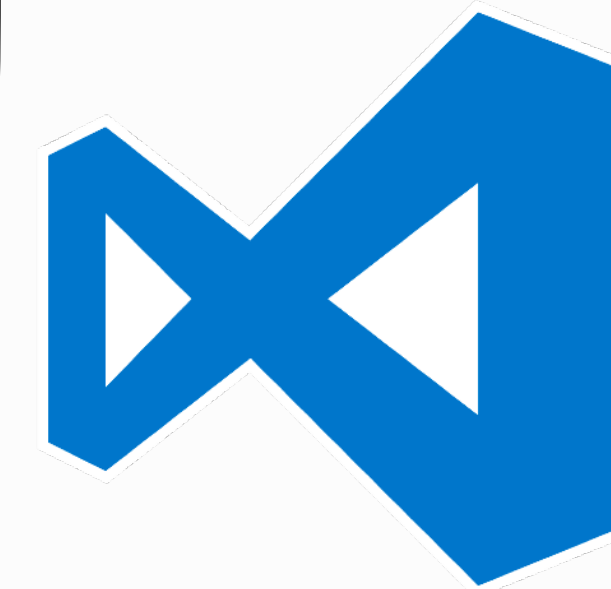
Reprise: Text editors

Large projects can be trying for some editors

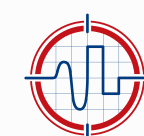
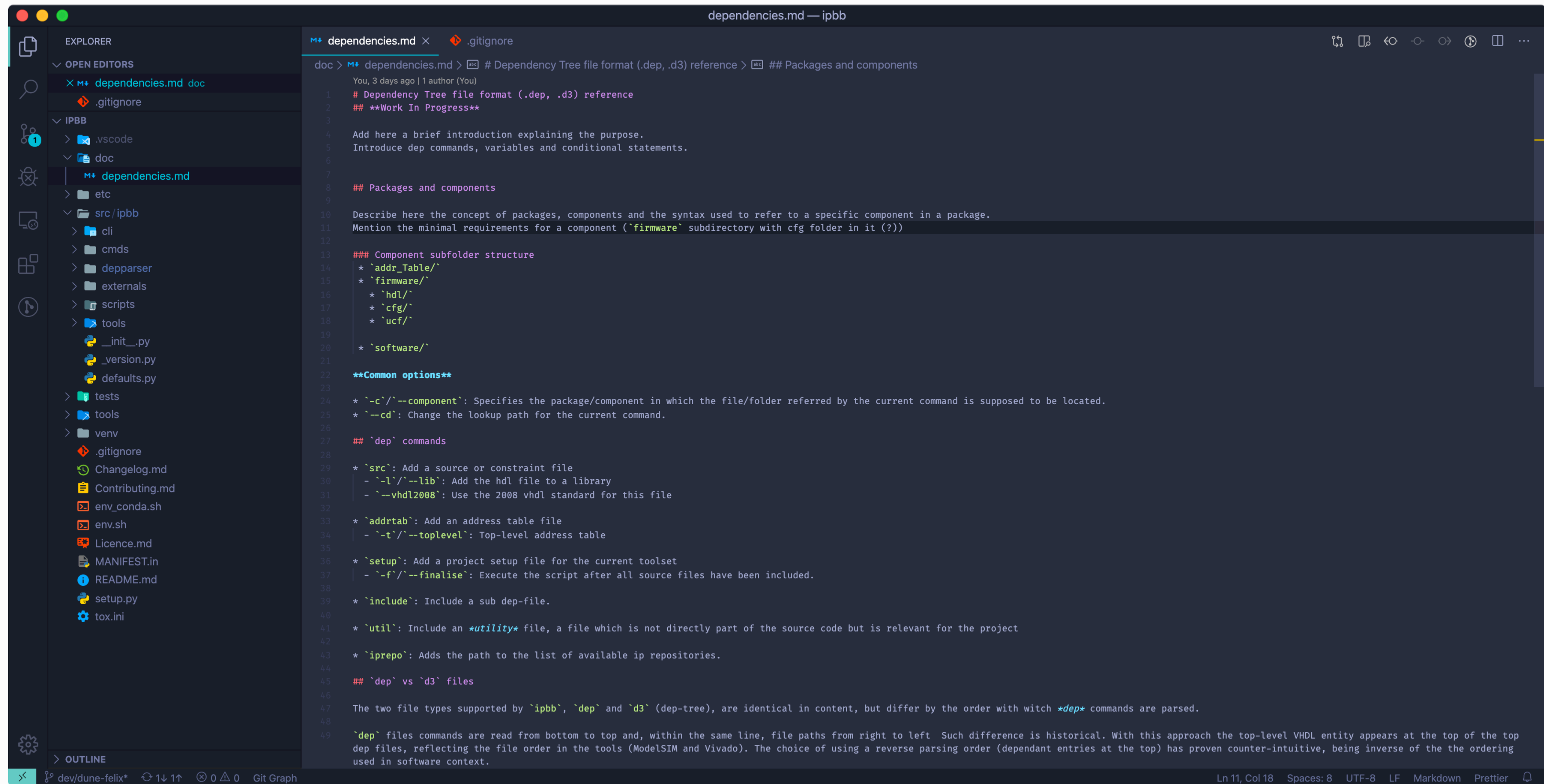
- and for you
- large codebase, files, folders, branches, tags, etc...
- Your favourite vintage editor may just not be up to the job

Modern editors offer non-negligible advantages

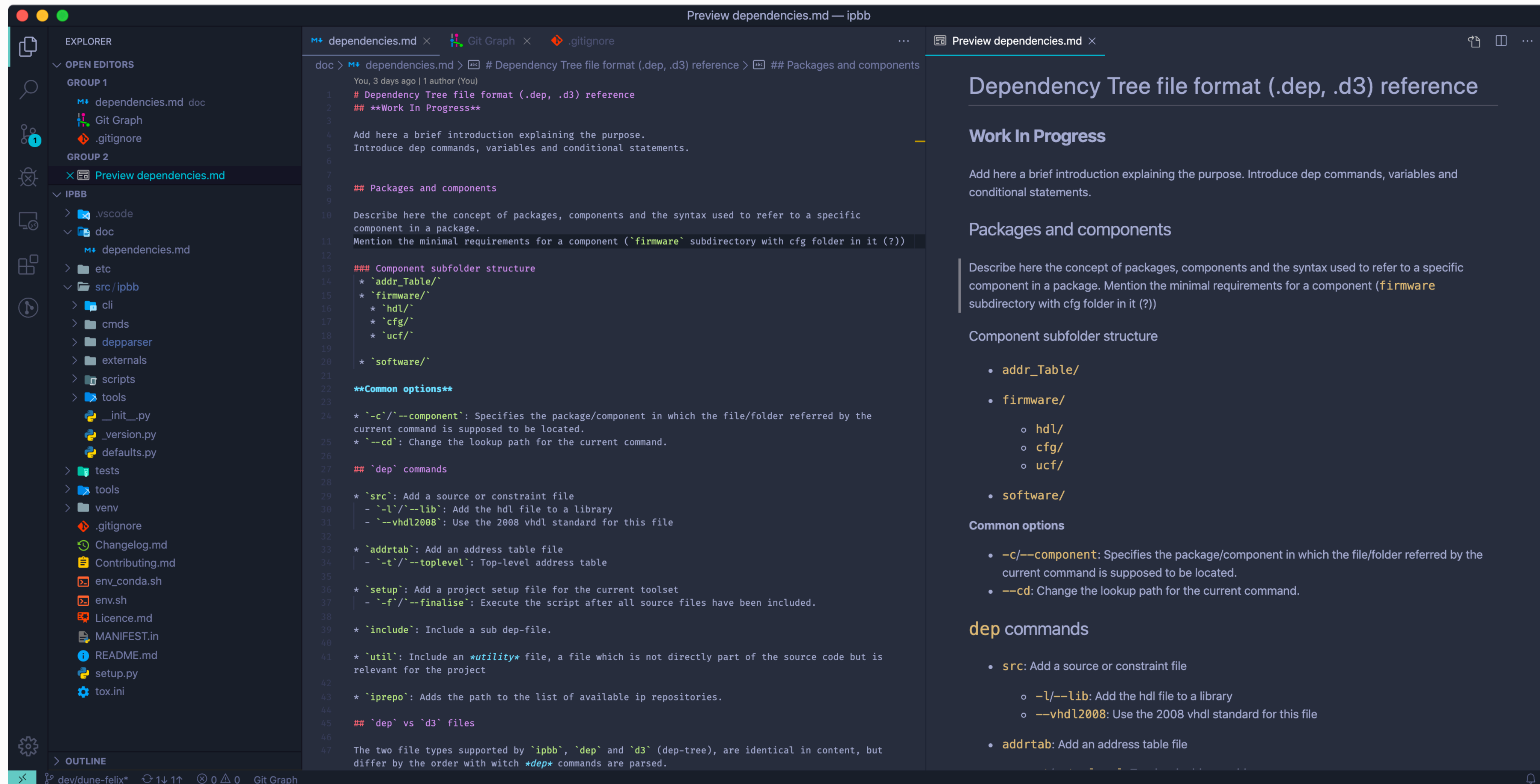
- Modern GUIs, flexible plugin system, large user community
- Syntax highlighting, Integration with VCS', preview, terminal integration and more



Text Editors++: Visual Studio Code



Text Editors++: Visual Studio Code

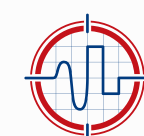


Text Editors++: Visual Studio Code

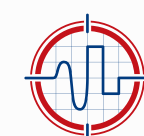
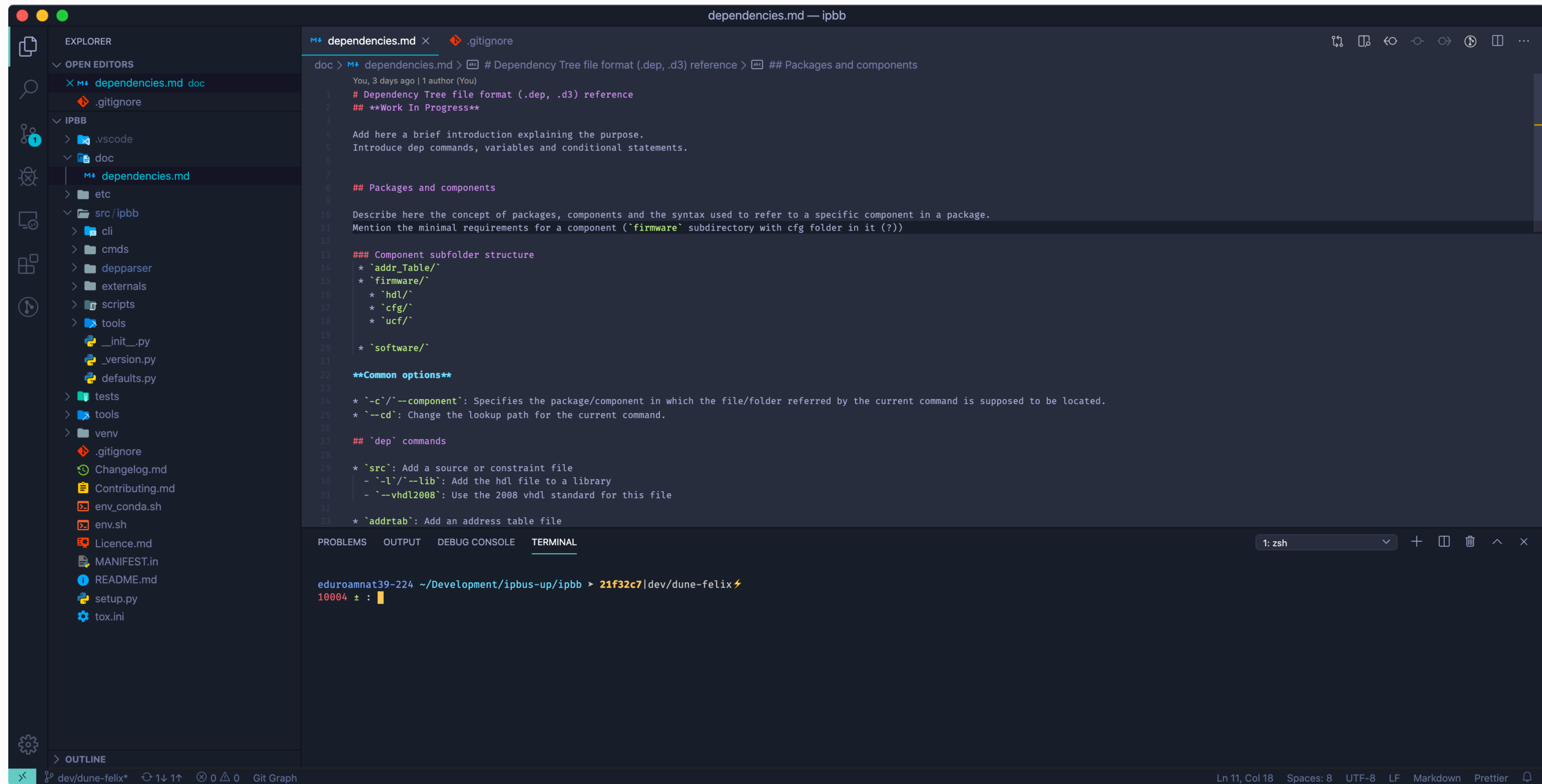
The screenshot displays the Visual Studio Code interface. The Explorer sidebar on the left shows the project structure, including folders like 'src/ipbb' and files like 'dependencies.md'. The main editor area shows the content of 'dependencies.md', which is a dependency tree file format. The Git Graph sidebar on the right shows a commit history table with columns for Branches, Description, Date, Author, and Commit.

```
doc > M+ dependencies.md > # Dependency Tree file format (.dep, .d3) reference > ## Packages and components
You, 3 days ago | 1 author (You)
1 # Dependency Tree file format (.dep, .d3) reference
2 ## **Work In Progress**
3
4 Add here a brief introduction explaining the purpose.
5 Introduce dep commands, variables and conditional statements.
6
7
8 ## Packages and components
9
10 Describe here the concept of packages, components and the syntax used to refer to a specific
11 component in a package.
12 Mention the minimal requirements for a component (`firmware` subdirectory with cfg folder in it (?))
13
14 ### Component subfolder structure
15 * `addr_Table/`
16 * `firmware/`
17 * `hdl/`
18 * `cfg/`
19 * `ucf/`
20
21 * `software/`
22
23 **Common options**
24 * `--component`: Specifies the package/component in which the file/folder referred by the
25 current command is supposed to be located.
26 * `--cd`: Change the lookup path for the current command.
27
28 ## `dep` commands
29
30 * `src`: Add a source or constraint file
31   - `--lib`: Add the hdl file to a library
32   - `--vhdl2008`: Use the 2008 vhdl standard for this file
33
34 * `addrtab`: Add an address table file
35   - `--toplevel`: Top-level address table
36
37 * `setup`: Add a project setup file for the current toolset
38   - `--finalise`: Execute the script after all source files have been included.
39
40 * `include`: Include a sub dep-file.
41
42 * `util`: Include an utility file, a file which is not directly part of the source code but is
43 relevant for the project
44
45 * `iprepo`: Adds the path to the list of available ip repositories.
46
47 ## `dep` vs `d3` files
48
49 The two file types supported by `ipbb`, `dep` and `d3` (dep-tree), are identical in content, but
50 differ by the order with which dep commands are parsed.
```

Branches	Description	Date	Author	Commit
	Uncommitted Changes (1)	13 Jan 2020 15:38	*	*
dev/dune-felix	Working on dep documentation	10 Jan 2020 09:...	Alessandro T...	21f32c7b
github/dev/dune-felix	Working on dep documentation	10 Jan 2020 09:...	Alessandro T...	efbcbbbe
	Bugfix: added missing `_` in front of utils package. First i...	9 Jan 2020 23:02	Alessandro T...	f27123ec
dev/master github	Merge branch 'feature/compone...	16 Dec 2019 13:...	Alessandro T...	afa9e74d
feature/components-autocompletion github	Fixed va...	16 Dec 2019 13:...	Alessandro T...	f79e6cb1
	Updated `proj create` and `toolbox checkdep` to be mo...	3 Dec 2019 13:04	Alessandro T...	5d099e52
	Working on autocompletion of depfile arguments/option...	2 Dec 2019 06:15	Alessandro T...	d46a4f76
	Spreading autocompletion around. See #56	30 Nov 2019 06:...	Alessandro T...	567c6526
	Initial implementation of package and component autoc...	30 Nov 2019 05:...	Alessandro T...	7b83a003
	Merge branch 'dev/dep-troubleshooting' into dev/master	3 Dec 2019 14:06	Alessandro T...	95a794f8
	Fixed few bugs in deparser formatter	29 Nov 2019 01:...	Alessandro T...	573fed08
	Adding migration script to convert old build areas to the ...	19 Nov 2019 13:...	Alessandro T...	4fd7a68c
	Bugfix	13 Nov 2019 17:36	Alessandro T...	6af5877d
	Factored code used to display missing files and compon...	5 Nov 2019 22:36	Alessandro T...	f016ce63
	Removed 'top.bit' from programmer's code.	24 Oct 2019 12:19	Alessandro T...	8228abcf
	Adding `srcs reset` command	23 Oct 2019 07:...	Alessandro T...	b403ecf2
	Merge branch 'dev/depparser' into dev/master	3 Dec 2019 14:04	Alessandro T...	494aa982
	Bugfix in `add symlink`	19 Oct 2019 22:05	Alessandro T...	722b9192
	Various minor bugfixes	14 Oct 2019 14:56	Alessandro T...	d15be733
	Switching to the new parser	4 Oct 2019 00:54	Alessandro T...	81aeb3ac
	Adding flag to continue on timing error (vivado impl) Wo...	2 Oct 2019 14:34	Alessandro T...	04e0c93e
	depparser: Improved the classes to store the file structure	1 Oct 2019 23:41	Alessandro T...	936f42f6
	Fixed several `vivado` subcommands still using `top` as...	1 Oct 2019 14:05	Alessandro T...	46075343
	Working on the new version of depparser	1 Oct 2019 14:05	Alessandro T...	923e4fc4
	Adding a repo-generator script for testing purposes	30 Sep 2019 06:...	Alessandro T...	a97847a1
	Reverse parsing working	29 Sep 2019 02:...	Alessandro T...	64477246
	Refactoring continues	28 Sep 2019 11:39	Alessandro T...	d3e8e624
	Refactoring depfile parser class.	28 Sep 2019 11:29	Alessandro T...	5f8b0552
	Small tweaks	28 Sep 2019 09:...	Alessandro T...	1150f660
	Adding support for string replacement in ipbb commands.	24 Sep 2019 22:...	Alessandro T...	f140ad4a
	programmer: added option to ad probe file	23 Sep 2019 17:...	Alessandro T...	f35763e9
	Introduced ipbb setup files	23 Sep 2019 17:...	Alessandro T...	7490b887



Text Editors++: Visual Studio Code



The git ecosystem

Easy to host & share your projects:

- Setting up a shared repo can be done via any cloud service, e.g. dropbox
- Many open-source hosting sites, biggest: github.com
- Not open to public but CERN users: [GitLab.cern.ch](https://gitlab.cern.ch)
 - ▶ Both include fairly usable issue-tracking
- The beauty of pull-requests*:
 - ▶ Do builds on pull-requests (combine with CI)
 - ▶ Review contributed code on pull-requests



Git is widely used — **de-facto community standard**

- Exception: Python uses Mercurial

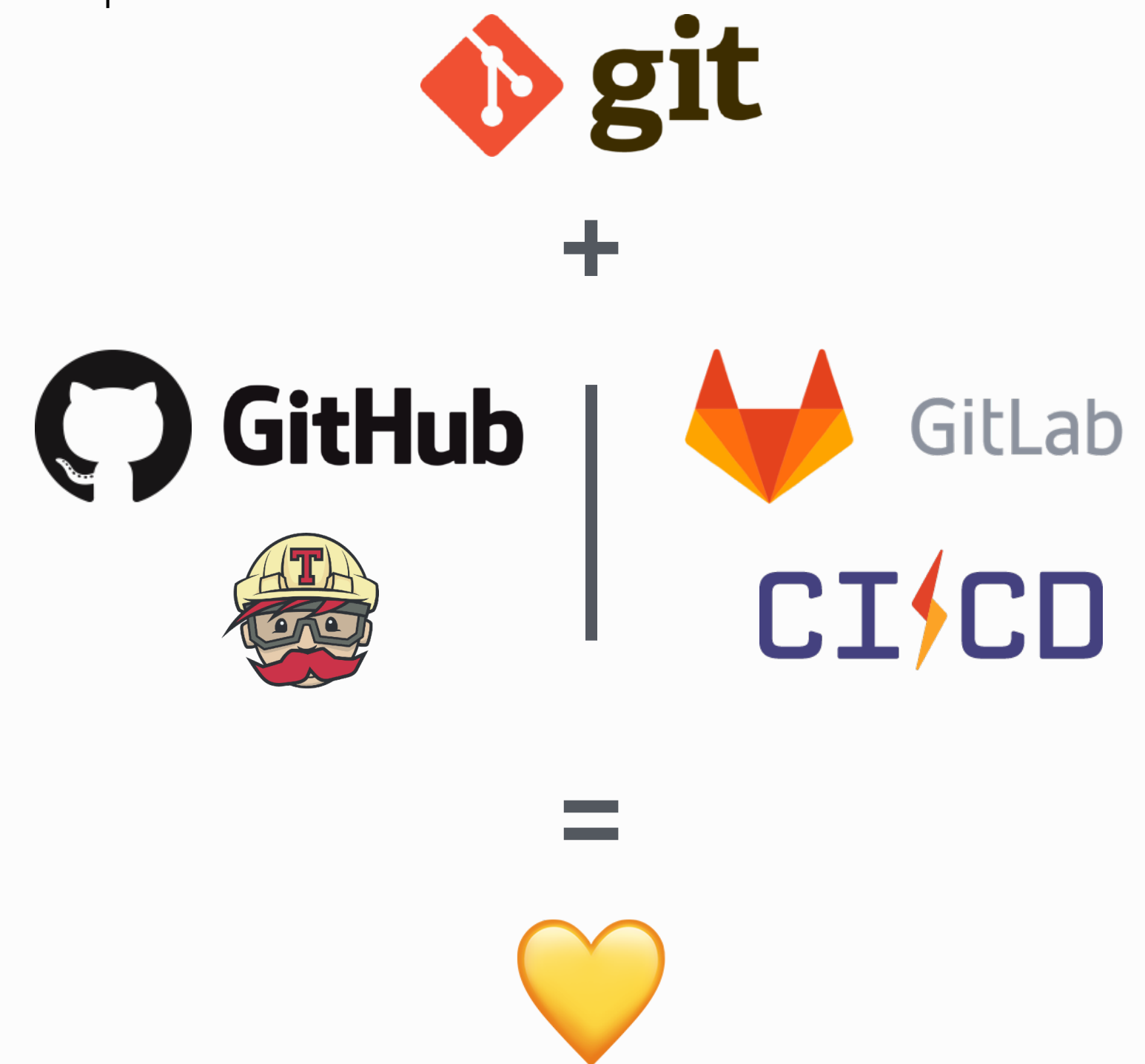
The more you learn the more you'll like it!

* merge-request in GitLab

The git ecosystem

Easy to host & share your projects:

- Setting up a shared repo can be done via any cloud service, e.g. dropbox
- Many open-source hosting sites, biggest: github.com
- Not open to public but CERN users: [GitLab.cern.ch](https://gitlab.cern.ch)
 - ▶ Both include fairly usable issue-tracking
- The beauty of pull-requests*:
 - ▶ Do builds on pull-requests (combine with CI)
 - ▶ Review contributed code on pull-requests



Git is widely used — **de-facto community standard**

- Exception: Python uses Mercurial

The more you learn the more you'll like it!

* merge-request in GitLab

Use the right tools for the job

Look around, you won't regret it

General Tips & Pointers

Learning about software development

Coursera — courses by universities (Caltech, Johns Hopkins, Stanford and more)

- <https://www.coursera.org/courses>
- Large variety of courses
 - ▶ Not only technology / programming
 - ▶ Also physics, biology, economics... and more
 - ▶ Also in different languages

Udacity — courses from industry (Google, Intel, Autodesk)

- <https://www.udacity.com/courses#!/all>
 - ▶ Mixed courses: Some free, recently switched to a payed model with monthly fees

University Homepages — have a gander... many courses available through YouTube etc.

- e.g.: [Programming Paradigms, Stanford University](#)

<http://ureddit.com/> — University of Reddit

Closing Advice

Before you write trigger / DAQ software (and firmware!), you should know the ins and outs:

- What is: compiler, interpreter, linker, terminal, object, class, pointer, reference
- If these concepts are not clear: Excellent material on the web (previous slide)

Before (and while) implementing: Think

- Smart solutions can take significant amount of time...
put it on the back-burner if you have other things to work on

Read! Ask! Write! The internet is full of information... Blogs, tutorials, StackOverflow, also Wikipedia can be very useful to get a grasp of new concepts

Conclusion

These slides were full of starting points: You have to follow up to get something out of it

- Most of it are tools to make your life easier
 - ▶ Bonus: If you know them you'll have an easier time to follow nerd-talk
- Nothing is free
 - ▶ You'll have to invest some effort to learn
 - ▶ If you do that this week: We'll be here to help!

Homework:

- Install git, start a repository. Try branching on the web
- Run tmux, kill the connection, reconnect and see if you can continue where you left off
- Tune your .bashrc / .bash_profile to get a more useful prompt
- Try out vim / emacs / atom / vscode and learn what suits you best
 - ▶ Download a shortcut summary...
 - ▶ Learn how to block-select, indent multiple lines, rename occurrences of text

Master by doing

Don't forget: Have fun while doing so!

Random Things

6 Stages of Debugging:

1. That can't happen.
2. That doesn't happen on my machine.
3. That shouldn't happen.
4. Why does that happen?
5. Oh, I see.
6. How did that ever work?
 - <http://plasmasturm.org/log/6debug/>

Go-language: Designed with threading in mind
<http://tour.golang.org/welcome/1>

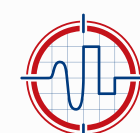
Want to try your programming skills?
Google code jam (registration open):
codingcompetitions.withgoogle.com/codejam
Also you can just practice
by solving nice problems.

Guru of the Week: (Not any more)
regular C++ programming problems
with solutions by Herb Sutter
<http://www.gotw.ca/gotw/>

“Debugging is like being the
detective in a crime novel where
you are also the murderer.”
– @fortes

About JavaScript:
<https://www.destroyallsoftware.com/talks/the-birth-and-death-of-javascript>
<https://www.destroyallsoftware.com/talks/wat>

2014 lecture has complementary stuff:
<http://indico.cern.ch/event/274473/session/21/material/0/0.pdf>



More Random Things

In HEP probably no way around ROOT / RooFit

- Maintained at CERN, used in LHC experiments

GNU R — www.r-project.org

- Used widely among statisticians (including finance and others)
- Interpreted language + software for analysis and graphical representation
- ROOT bindings now available (use it through TMVA)

SciPy — <http://www.scipy.org/>

- Collection of python libraries for numerical computations, graphical representation and containing additional data structures

Sci-kitlearn: — <http://scikit-learn.org/stable/>

- Python library for machine learning
- ROOT bindings available (usable through TMVA)

Data visualisation:

Matplotlib (part of SciPy)

- histograms, power spectra, scatterplots and more.. extensive library for 2D/3D plotting

ROOT

- Again, probably no way around it... Sometimes a little unintuitive

Other:

JaxoDraw — <http://jaxodraw.sourceforge.net/>

- Feynman graphs through “axodraw” latex package

tex2im — <http://www.nought.de/tex2im.php>

- Need formulas in your favourite WYSIWG presentation tool?

GraphViz — <http://www.graphviz.org/> or MacOS: <http://www.pixelglow.com/graphviz/>

- Diagrams / Flowcharts with auto-layout

SAGE — www.sagemath.org

- Open source alternative to Matlab, Maple and Mathematica

GNUPlot — <http://www.gnuplot.info/>

- Quick graphing and data visualisation

Wolfram Alpha — <http://www.wolframalpha.com/>

- Wolfram = Makers of Mathematica.. A... ask me anything?:
 - ▶ <http://www.wolframalpha.com/input/?i=how+much+does+a+goat+weigh>
 - ▶ Answer: Assuming “goat” is a species specification. Result: 61 kg