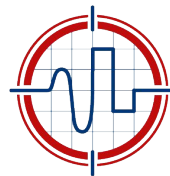


Programming for today's physicists and engineers

D. Rabady, CERN

ISOTDAQ 2023: 13th International School of Trigger and Data Acquisition
Istanbul University, Istanbul, Turkey, 21 June 2023



ISOTDAQ

International School of Trigger
and Data Acquisition



Opening words

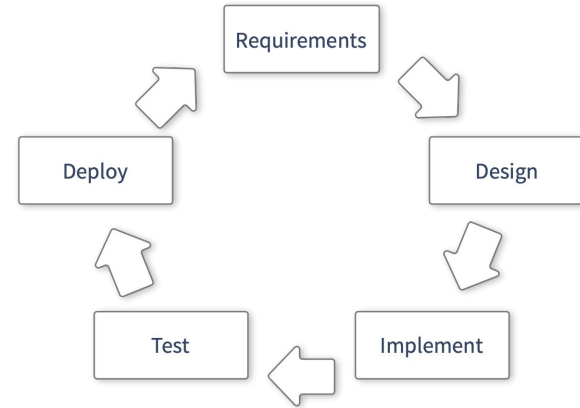
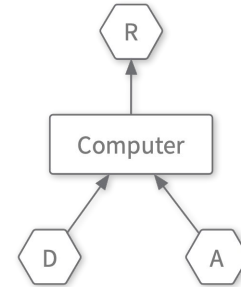
Disclaimer: This is more a collection of pointers* than a tutorial, it's a starting point...
(Almost) no code but a bias towards C++ and Python

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

Acknowledgment: Slides are based on previous lectures by Alessandro Thea, Joschka Poettgen (Lingemann) and Erkan Ozcan

*further reading and tips in these boxes

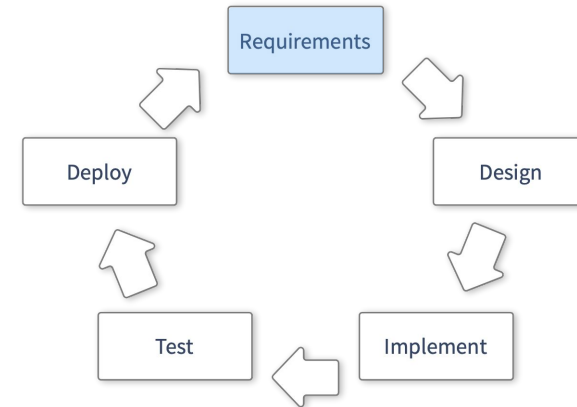
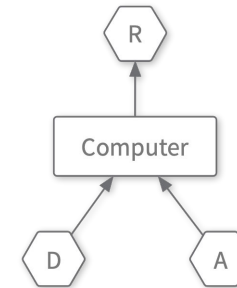
What is programming?



https://en.m.wikipedia.org/wiki/Systems_development_life_cycle

What is programming?

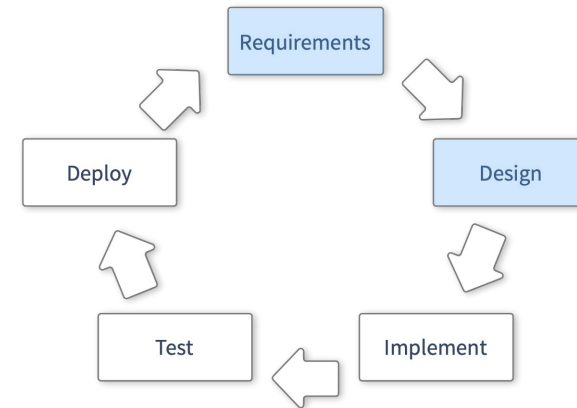
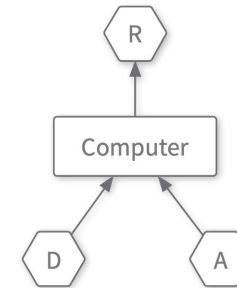
- Understand & define the problem to solve
 - Define the requirements for your software
 - Choose the language



https://en.m.wikipedia.org/wiki/Systems_development_life_cycle

What is programming?

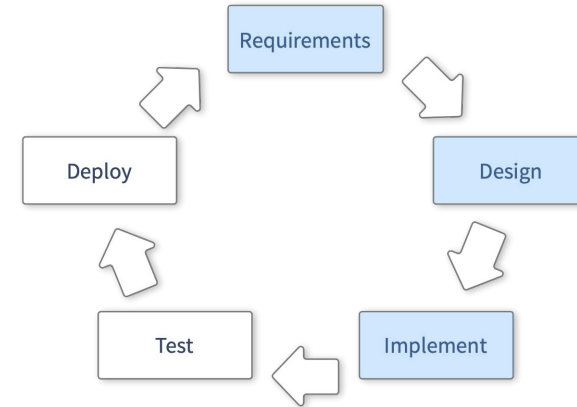
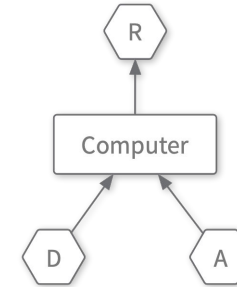
- Understand & define the problem to solve
 - Define the requirements for your software
 - Choose the language
- Formulate a possible solution (design)
 - Identify key functionalities and features



https://en.m.wikipedia.org/wiki/Systems_development_life_cycle

What is programming?

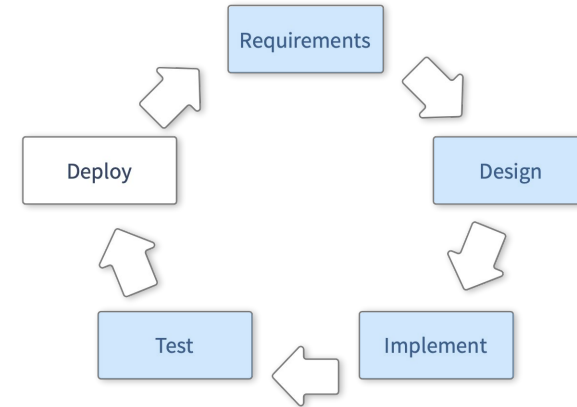
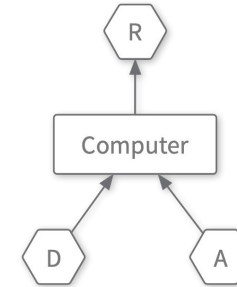
- Understand & define the problem to solve
 - Define the requirements for your software
 - Choose the language
- Formulate a possible solution (design)
 - Identify key functionalities and features
- Implement the design
 - Write code, debug it
 - Prepare documentation



https://en.m.wikipedia.org/wiki/Systems_development_life_cycle

What is programming?

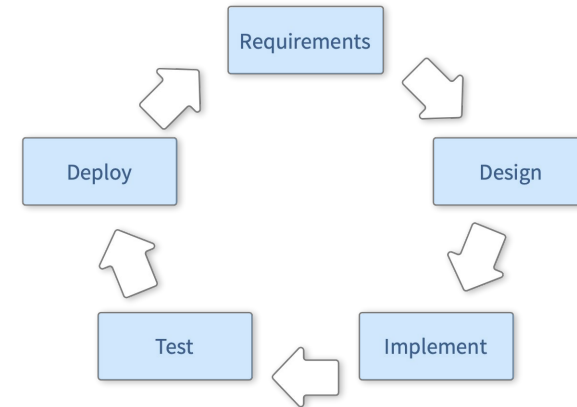
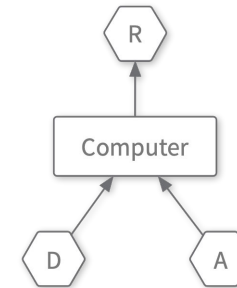
- Understand & define the problem to solve
 - Define the requirements for your software
 - Choose the language
- Formulate a possible solution (design)
 - Identify key functionalities and features
- Implement the design
 - Write code, debug it
 - Prepare documentation
- Validate the code
 - Perform thorough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



https://en.m.wikipedia.org/wiki/Systems_development_life_cycle

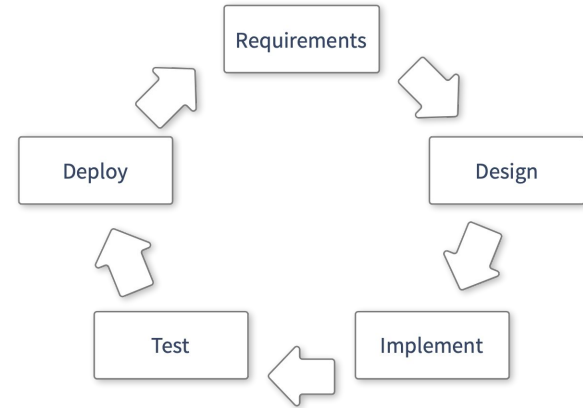
What is programming?

- Understand & define the problem to solve
 - Define the requirements for your software
 - Choose the language
- Formulate a possible solution (design)
 - Identify key functionalities and features
- Implement the design
 - 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



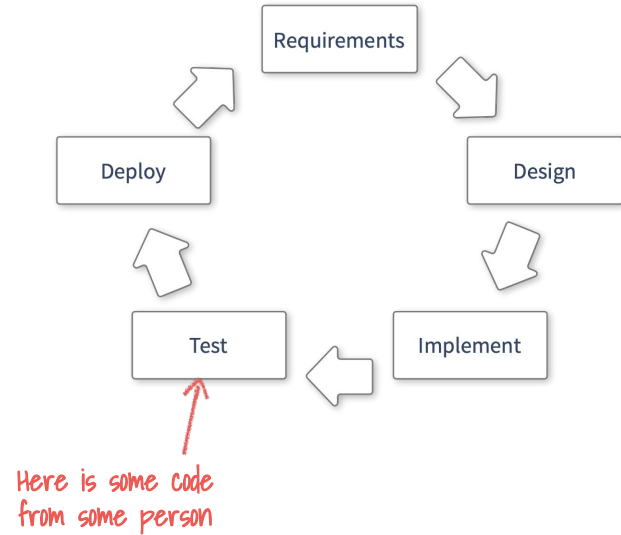
https://en.m.wikipedia.org/wiki/Systems_development_life_cycle

What can programming be like...



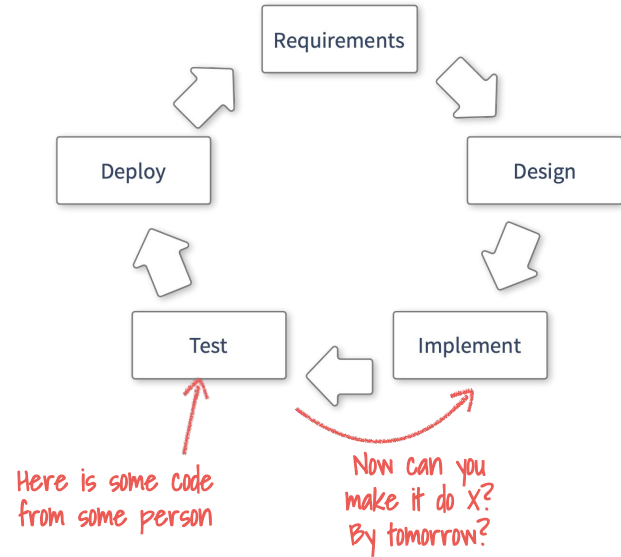
What can programming be like...

- Inherit some code
 - Poke at it to get the hang of it



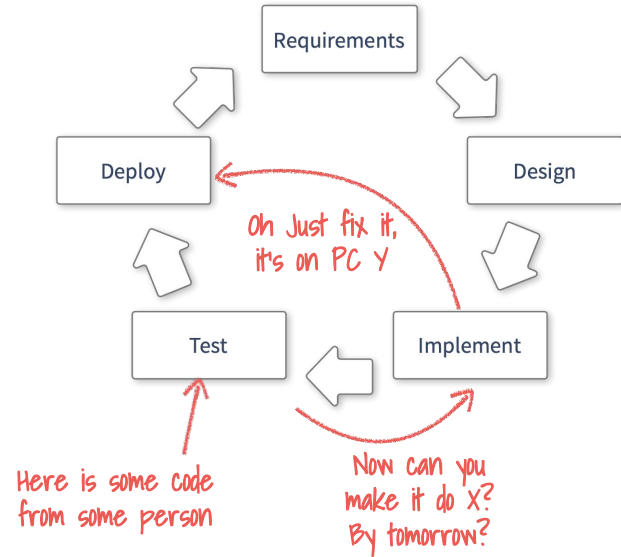
What can programming be like...

- Inherit some code
 - Poke at it to get the hang of it
- Add some features
 - The purpose of which is not completely clear
 - By hack... patching some files



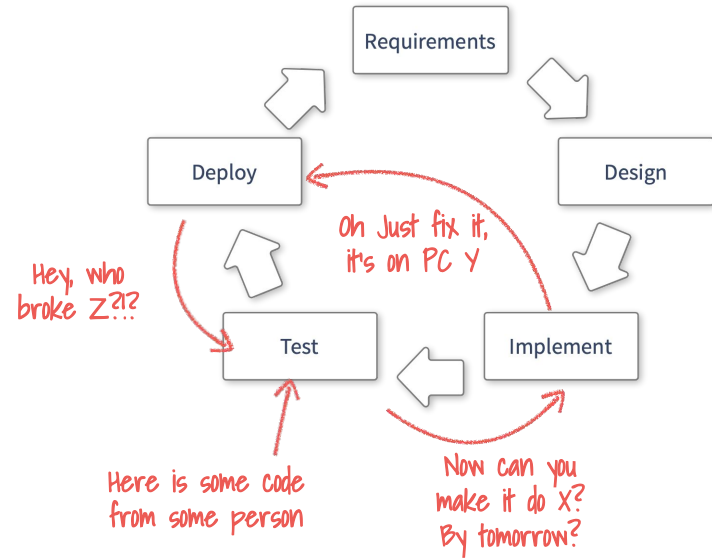
What can programming be like...

- Inherit some code
 - Poke at it 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?



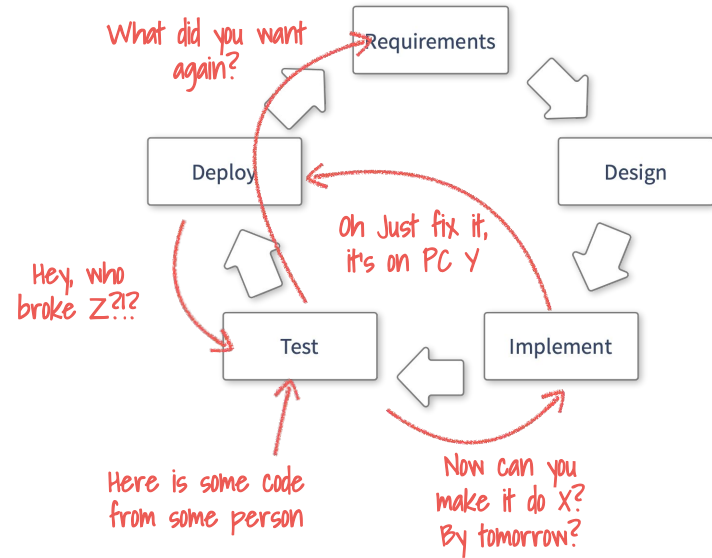
What can programming be like...

- Inherit some code
 - Poke at it 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.



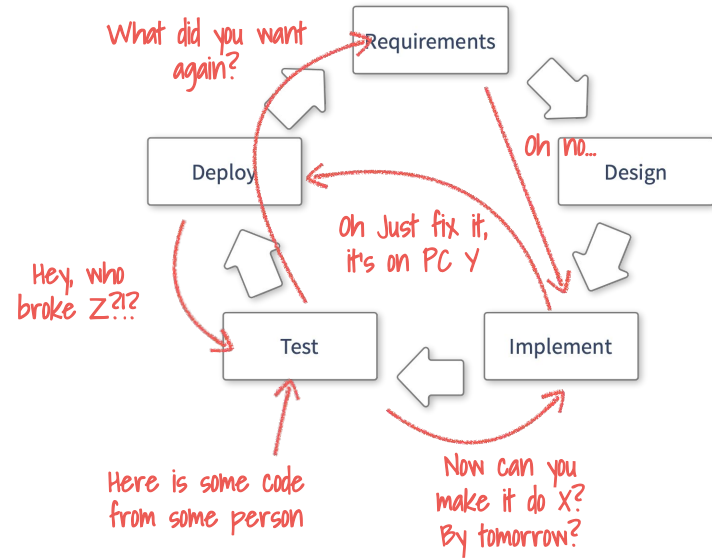
What can programming be like...

- Inherit some code
 - Poke at it 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 can programming be like...

- Inherit some code
 - Poke at it 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...

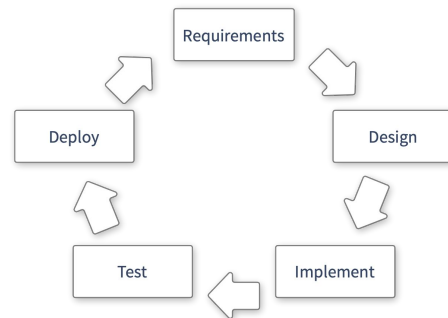


But it doesn't have to be...

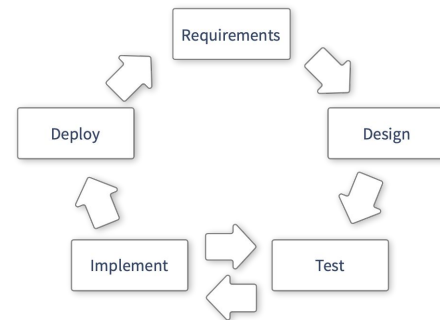
- Many ways to do this right
 - Agile/waterfall project management
 - Extreme programming
 - Test driven development
 - Pair programming
 - ...
 - Some of these have ideas/techniques that are even useful in isolation

- General guidelines
 - Avoid duplication of work
 - Avoid feature bloat
 - Ensure code quality
 - Keep in mind: Software engineering relies heavily on communication!
 - Communication with the computer
 - Communication with your peers

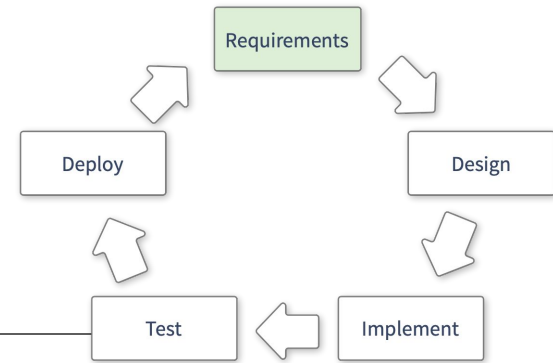
Iterative Development



Test-Driven Development

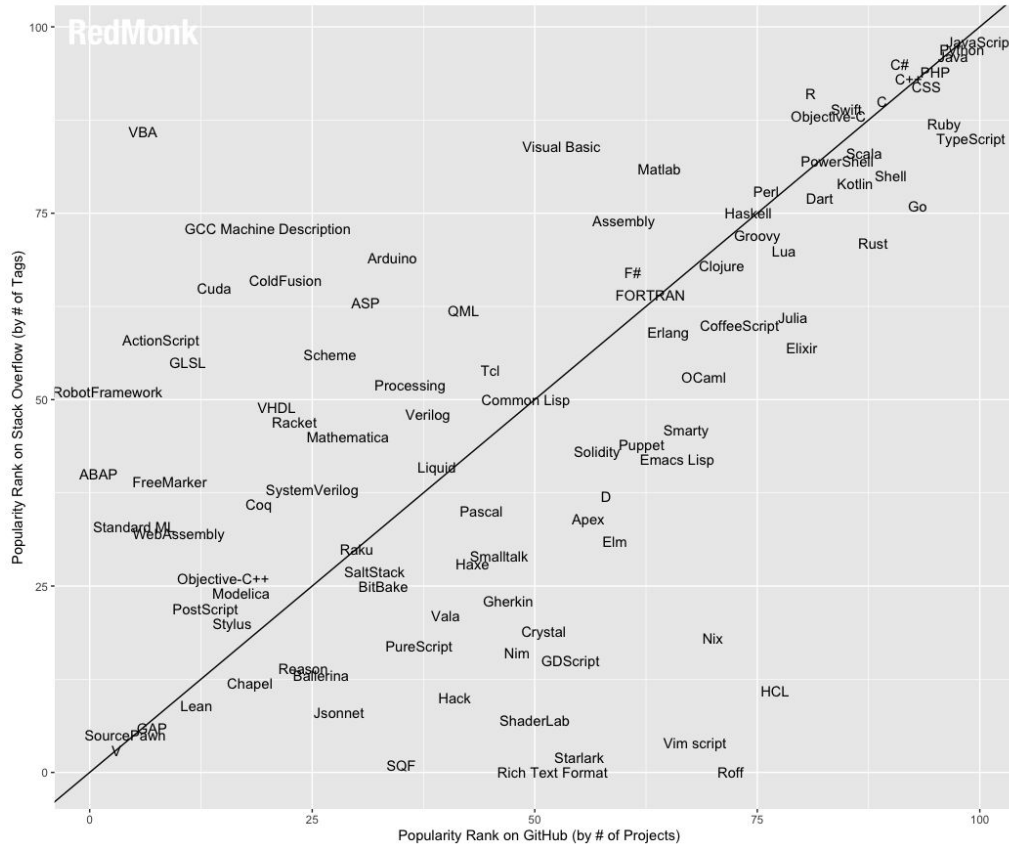


Requirements



Choosing the programming language

RedMonk Q122 Programming Language Rankings



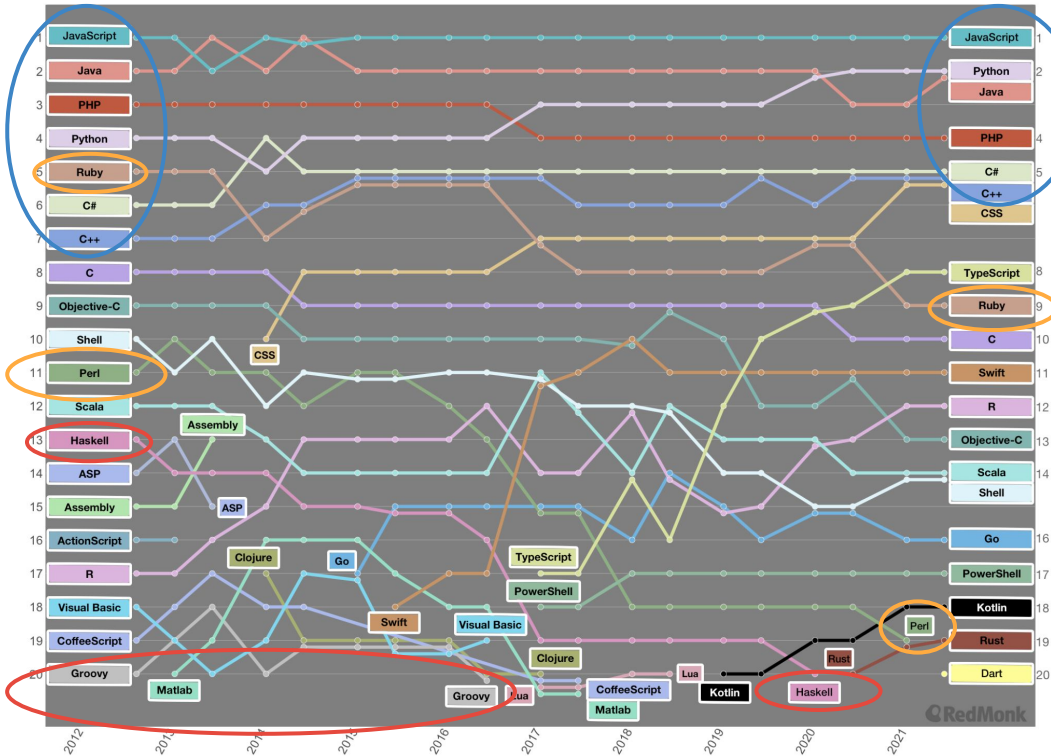
Things to consider:

- Problem domain
 - Analysis?
 - DAQ?
- Popularity
 - The more people using the language the more questions already asked (and hopefully answered)

Choosing the programming language

RedMonk Language Rankings

September 2012 - June 2021



Things to consider:

- Problem domain
 - Analysis?
 - DAQ?
- Popularity
 - The more people using the language the more questions already asked (and hopefully answered)
 - But it shouldn't be short lived...

Choosing the programming language



Things to consider:

- Problem domain
 - Analysis?
 - DAQ?
- Popularity
 - The more people using the language the more questions already asked (and hopefully answered)
 - But it shouldn't be short lived...
- Documentation and support
 - Don't underestimate the utility of useful official docs

Choosing the programming language

Things to consider:

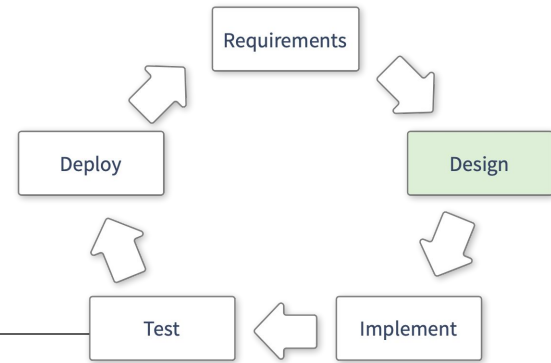
- Problem domain
 - Analysis?
 - DAQ?
- Popularity
 - The more people using the language the more questions already asked (and hopefully answered)
 - But it shouldn't be short lived...
- Documentation and support
 - Don't underestimate the utility of useful official docs
- **Can you choose?**
 - **Existing expertise in group**
 - Language support at workplace
- ...

```
; g++ -std=c++17
g++: error: unrecognized command line option '-std=c++17'
```

```
; python3.9
zsh: command not found: python3.9
```

Do you really
have to
program?
Or has somebody already done it for you?

Design



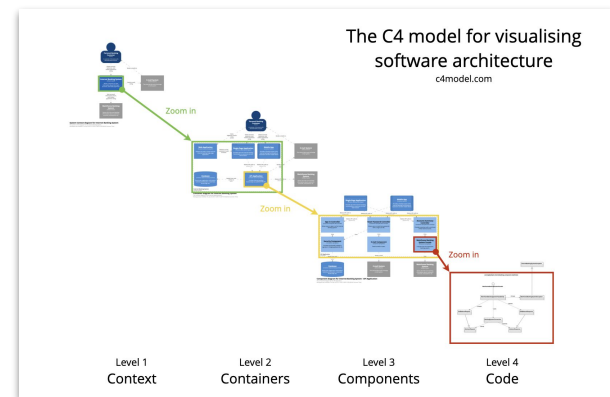
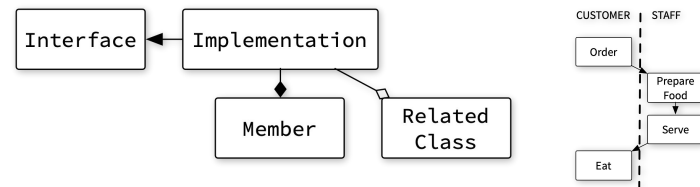
Plan ahead

Sketch out what you want to design

- Unified modeling language (**UML**) diagrams
 - Support for many types of diagrams
 - Structural, behaviour, interaction diagrams,...
 - Very powerful
 - Not very widespread nowadays, though...
- **C4 model**
 - Rather new arrival, but gaining popularity
 - Four levels of diagrams from "birds eye view" to "code level" diagram
 - UML used for lowest level code description

Write your first ideas out in **pseudo-code**

- Don't need to think about scope, initialisation, etc.
- Allows you to quickly put your thoughts to paper
 - Can be useful e.g., for algorithm design, when you want to spend your brain cycles on the problem, not on the tools



```
function find_local_optimum()  
    while solution is not locally optimal do  
        find s with f(s) < f(solution)  
    return s
```


Things to keep in mind when designing

Goal

- Are you still building what was required?
- Are you adding unneeded features?

Maintainability

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

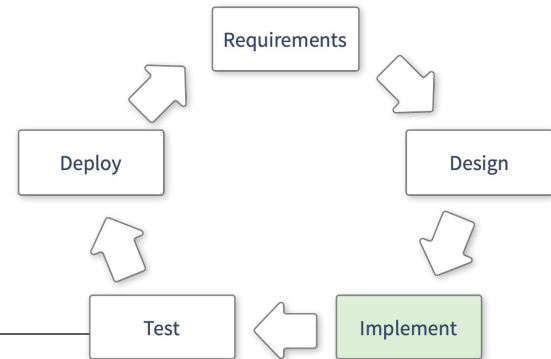
Scalability

- 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?

Implementation



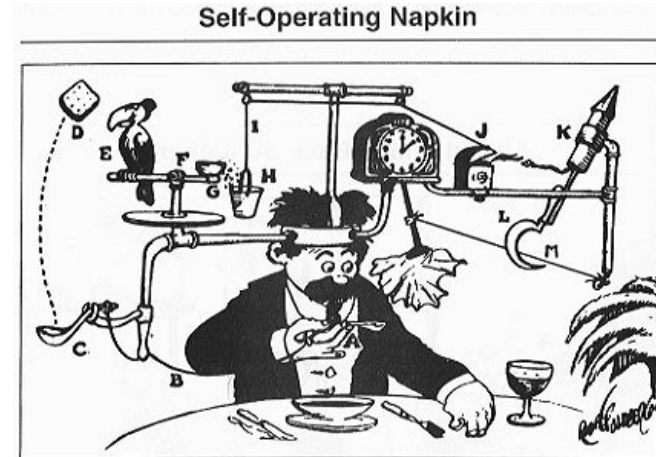
Do not reinvent the wheel

Look around for existing solutions

- Many problems have already been solved
- Look for libraries where:
 - Active community? Well maintained? Tested?
 - Rule of thumb: Last commit a few days ago, at most a year old
- Be wary of libraries with many features
 - Often come with lots of dependencies, increasing your attack surface, overall complexity, might slow down your code, ...
- Caveat: **Don't pull in external code for trivialities**
 - Almost no reason to use e.g., <https://pypi.org/project/isEven/>

Getting to know new frameworks:

- Read the docs
 - Investing time in the beginning will pay off
- python packages: try the ipython “help”
- Start with a simple test
 - Modify existing examples to do what you want to do
 - Does the code do what you expect?



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

Before searching for external libraries: Check out e.g., STL and Python standard library!

When coding: Avoid feature bloat

If you squeeze every 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 as simple as possible!



Tools of the Trade: Editor, Terminal and IDEs

Whatever you do, you'll end up using (at least)

- Editor
 - Know* at least one ubiquitous editor: [nano](#), [vi\(m\)](#), [emacs](#), etc.
 - More modern solutions: can make your life a lot easier..
 - Depending on the language/platform (e.g., Java): IDEs are a better choice (Eclipse, Netbeans, etc.)
- Terminal
 - Learn about shortcuts (minimal set: [Tab](#), [Ctrl+r](#), [Ctrl+e](#), [Ctrl+a](#),... have a look**)
 - Knowing about some basic command line-tools can come in handy

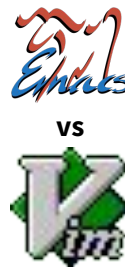
*at least how to exit them :-)

**<https://ostechnix.com/list-useful-bash-keyboard-shortcuts/>
or <https://gist.github.com/tuxfight3r/60051ac67c5f0445efee>

Some words on editors: Choose what suits you

The choice of editor is yours to make...

- Do you want
 - “a great operating system, lacking only a decent editor”
 - Or one with two modes: “beep constantly” and “break everything”*
- **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 (reasonably) open: e.g., Microsoft VSCode, Adobe Brackets,...
 - 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
- Learn about: Linters, version control system integrations, and other plugins

*from [The Editor War](#)

The terminal

Initially: 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:
 - `$ grep -R -A 3 "foo" *`
 - Displays all matches of “foo” (+3 lines below) in all files in the current directory and its subfolders
 - Can be extended to only search files with certain ending
 - `$ find . -name "*.cc" -exec grep -A 3 "foo" {} +`
- 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 be put into a script**
- **Alternative:** Can solve most things more conveniently with an interpreted language
 - **Con:** Might not always be available where you need it

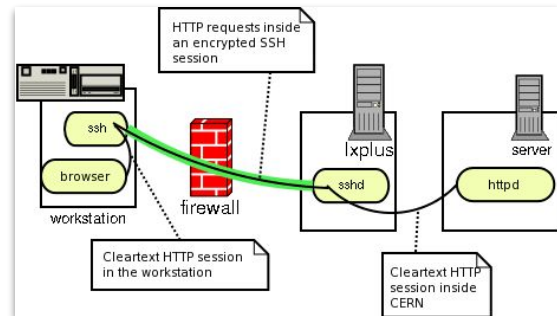
*to dump e.g., the last 100 commands:
`$ history | tail -n 100 > history.txt`

Bite size bash (few dollars, but quite useful):
<https://wizardzines.com/zines/bite-size-bash/>

Interlude: Working on the road — SSH

SSH — very, very versatile:

- **Tunneling**
 - Secure connections to other machines
 - Can e.g., establish secure connections to machines behind firewalls
- Keys for authentication instead of passwords
 - Makes life a bit more comfortable
 - **Alternative:** Login via Kerberos token if available*
- To work around shaky connections
 - Always use tmux/screen or a similar terminal multiplexer
- **SSHFS**
 - Files on remote host but "pretend" to be local



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/>

* see e.g., <https://linux.web.cern.ch/docs/kerberos-access/>

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 these are 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!

```
import click

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}

@click.command()
@click.option('--month', prompt='Which month are you interested in?',
              help='Print the number of days this month has.')
def print_days_in_month(month):
    """Simple program that states how many days MONTH has."""
    if month in months:
        print(f"{month} has {months[month]} days.")
    else:
        print(f"Sorry. Month {month} not known.")

if __name__ == '__main__':
    print_days_in_month()
```

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 these are 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!

```
import click

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}

@click.command()
@click.option('--month', prompt='Which month are you interested in?',
              help='Print the number of days this month has.')
def print_days_in_month(month):
    """Simple program that states how many days MONTH has."""
    if month in months:
        print(f"{month} has {months[month]} days.")
    else:
        print(f"Sorry. Month {month} not known.")

if __name__ == '__main__':
    print_days_in_month()
```

```
$ python month_script.py --help
Usage: month_script.py [OPTIONS]

    Simple program that states how many days MONTH has.

Options:
  --month TEXT  Print the number of days this month has.
  --help        Show this message and exit.
```

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 these are 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!

```
import click

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}

@click.command()
@click.option('--month', prompt='Which month are you interested in?',
              help='Print the number of days this month has.')
def print_days_in_month(month):
    """Simple program that states how many days MONTH has."""
    if month in months:
        print(f"{month} has {months[month]} days.")
    else:
        print(f"Sorry. Month {month} not known.")

if __name__ == '__main__':
    print_days_in_month()
```

```
$ python month_script.py --help
Usage: month_script.py [OPTIONS]

    Simple program that states how many days MONTH has.

Options:
  --month TEXT  Print the number of days this month has.
  --help        Show this message and exit.
$ python month_script.py --month november
november has 30 days.
$ python month_script.py
Which month are you interested in?: november
november has 30 days.
```

Static Code Checking

While writing your code:

- There are static code analysis tools that can help you
 - Like spell-check for your code
- Try out a linter for your preferred editor*
 - Highlights potentially problematic code
 - Your code will be more reliable

Static checking at compile time:

- Clang has a nice suite of static checks implemented**
 - 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 early on!

```
1  #!/usr/bin/env bash
2
3  set -e # Exit on error
4
5  if [ $# -lt 4 ]; then
6      echo "#### Usage:"
7      echo "## " $Q "[CERN username] [bitfile version] [scouting_board_type] [input_system]"
8      echo "## Example " $Q "dinyar vx.y.z kcu1500 ugmt"
9      echo "## Exiting.."
10     exit 1
11 fi
12
13 if [ "$(command -v vivado_lab)" ]; then
14     echo "Found Xilinx Vivado Lab Edition at" $(command -v vivado_lab)
15 else
16     echo "vivado_lab could not be found in your path. Exiting."
17     exit 1
18 fi
19
```

Quote this to prevent word splitting. shellcheck(SC2046)
View Problem Quick Fix... (Ctrl+.)

You, 2 months ago • Fix bug stopping

```
10
19
20 public void addConfigSetting(String id, String val) {
21     configSettings_.put(id, val);
22 }
23
24 public String getHost() {
25     return host_;
26 }
27
28 public int
29     return "some_string";
30 }
31
32 public Map<String, String> getConfigSettings() {
33     return configSettings_;
34 }
35
```

Type mismatch: cannot convert from String to int Java(16777235)
View Problem Quick Fix... (Ctrl+.)

You, 1 second ago • Uncommitted changes

* Examples for VSCode:

Python: <https://code.visualstudio.com/docs/python/linting>

Shell: <https://github.com/vscode-shellcheck/vscode-shellcheck>

** see <http://clang-analyzer.llvm.org>

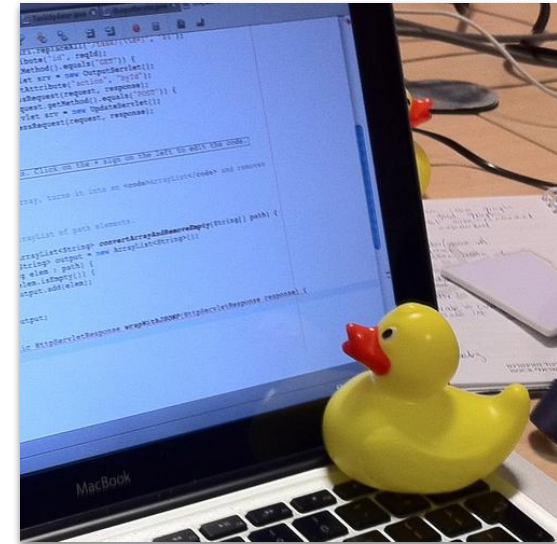
Debugging

- While running your code:
 - **Printing to console/log**: especially convenient if working in unfamiliar programming environment
 - This works for a surprisingly long time...
 - Sooner or later you'll want to use a debugger: e.g. **gdb (GNU debugger)**
 - basic commands: run, bt, info <*>, help
 - Very useful documentation in the Red Hat Developer Guides for e.g. [RHEL7](#) and [RHEL9](#)
 - Python debugger (pdb, ipdb*, pygdb)
 - Under Linux: **strace**
 - To determine which system calls the program uses (and where they might be failing)
 - Use with: `strace <program name>`
 - Nice tutorials in the internet (e.g. <https://opensource.com/article/19/10/strace>)

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

General hints for debugging

- Segmentation violations due to memory management
 - Life-time vs. scope
 - Look at smart pointers (part of C++11/14 standards, alternative: boost)
- Even if you don't have crashes: Memory Leaks, try [valgrind](#)
- When all else fails: [Use a rubber duck](#)
 - or invite your colleague for a coffee...
 - Verbalising a problem one has can often lead to a moment of epiphany



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”
 - Can at worst lead to “comment bugs”
- Clean code (almost) documents itself: **You write it once but you read it many times**

```
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”
 - Can at worst lead to “comment bugs”
- Clean code (almost) documents itself: **You write it once but you read it many times**

```
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!

Standalone documentation:

- Again: Explain your interfaces (can be derived from internal, e.g. doxygen.org)
- For large projects: **Explain the big picture**
 - Give use-cases and examples
 - Consider using UML (unified modelling language) or other graphical notation techniques (e.g., the C4 model)

Introduction

SWATCH is a C++ library designed to provide a generic interface for controlling and monitoring the 2016 Level-1 trigger upgrade hardware.

The generic interface is based on the following classes:

- Processor (`swatch::processor::Processor`): Represents the uTCA cards that process trigger primitive data
- DaqTTCManager (`swatch::dtm::DaqTTCManager`): Represents the connection to the central TCDS and DAQ networks (i.e. the AMC13)
- System (`swatch::system::System`): Represents a trigger subsystem composed of one or more processors and DaqTTCManagers (e.g. calo layer-1, calo layer-2, uGT, etc)

Each processor contains the following interfaces:

- TTC interface (`swatch::processor::TTCInterface`)
- Readout interface (`swatch::processor::ReadoutInterface`)
- Input optical port (`swatch::processor::InputPort`)
- Output optical port (`swatch::processor::OutputPort`)
- Algorithm interface (`swatch::processor::AlgoInterface`)

These generic interface classes contain a built-in common monitoring and control framework. There are three concepts of actions for controlling hardware in SWATCH:

- Commands: One shot action - e.g. reset board, configure links
- Command sequences: Multiple commands executed in succession
- Operations: FSMs (Finite State Machines); each transition is typically a single command or command sequence

The monitoring framework is based around two main types of classes:

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!

Standalone documentation:

- Again: Explain your interfaces (can be derived from internal, e.g. doxygen.org)
- For large projects: **Explain the big picture**
 - Give **use-cases and examples**
 - Consider using UML (unified modelling language) or other graphical notation techniques (e.g., the C4 model)

Step 2: Register subsystem-specific metrics

In the subsystem classes for processors and processor component interfaces (TTC, readout, input port, output port, and algorithm), you can add subsystem-specific metrics, which will then be accessible from the generic SWATCH API, and updated along with that object's common metrics. The new metrics are registered in the constructor of the subsystem-specific class, by calling one of the `registerMetric` methods, and their values must be set in the subsystem class' implementation of the `retrieveMetricValues()` method (in the same way as the values of the common metrics are set).

For example, a `uint16_t` counter for an input port - with "good" status for value of zero, error if the value is above 8, and warning otherwise - could be registered in the `MyInputPortClass` as follows:

- Add a reference to that type of metric as member data:

```
class MyInputPort {  
    // ...  
    // ...  
    // ...  
  
    swatch::core::Metric<uint16_t> mMetricMySpecialCounter;  
};
```

- Register that metric in the constructor:

```
MyInputPort::MyInputPort ( ... ) :  
    swatch::processor::InputPort(stub),  
    // ...  
    mMetricMySpecialCounter( registerMetric<uint16_t>("specialCounter", swatch::core::GreaterTha  
{  
    // ...  
}
```

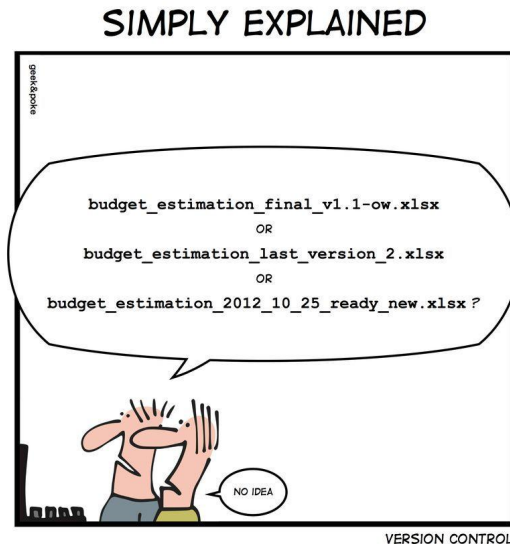
Always track code changes - Revision Control

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 is shattered 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

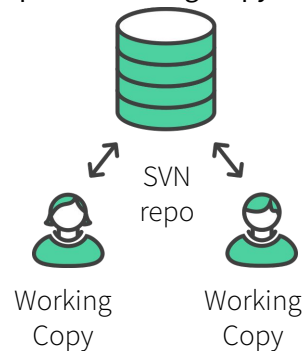
- Learn to track (and comment) every code change
- RCS is essential (and unavoidable) for collaboration



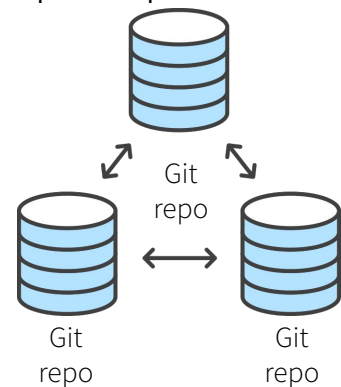
Revision Control Software

- Once upon a time: CVS and Subversion
- **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
 - Other distributed solutions are: Mercurial, bazaar...
 - Easy to get started...

Central-Repo-To-Working-Copy Collaboration



Repo-To-Repo Collaboration



Interlude: git basics

```
$ git init
```

```
Initialized empty Git repository in /TestDirectory/.git/
```

Interlude: git basics

```
$ git init
```

```
Initialized empty Git repository in /TestDirectory/.git/
```

```
$ touch README.md # Create a readme file
```

```
$ # Edit the file...
```

Interlude: git basics

```
$ git init
```

Initialized empty Git repository in /TestDirectory/.git/

```
$ touch README.md # Create a readme file
```

```
$ # Edit the file...
```

```
$ git add README.md # Add the file to the index
```

Interlude: git basics

```
$ git init
```

```
Initialized empty Git repository in /TestDirectory/.git/
```

```
$ touch README.md # Create a readme file
```

```
$ # Edit the file...
```

```
$ git add README.md # Add the file to the index
```

```
$ git commit -m "Initial commit to my new repo"
```


Interlude: git basics

```
$ git init
```

```
Initialized empty Git repository in /TestDirectory/.git/
```

```
$ touch README.md # Create a readme file
```

```
$ # Edit the file..
```

```
$ git add README.md # Add the file to the index
```

```
$ git commit -m "Initial commit to my new repo"
```

```
$ # Create more files and edit them..
```

```
$ git add firstFile.txt sourceFile.cc headerFile.h
```

```
$ git commit -m "Short descriptive summary
```

```
dquote>
```

```
dquote> More expansive explanation of the commit,
```

```
dquote> if necessary."
```

How-to write good commit messages:
<https://chea.ms/git-commit/>

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

Interlude: git basics, avoid this:

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Git in a nutshell

Learn basic concepts and commands

- Create repository, add file, commit new versions: `git (init, add, commit)`

Familiarise yourself with parallel development concepts

- Switch to branches or create new ones: `git (switch [-c]/checkout [-b])`
- Merging and rebasing: `git (merge, rebase)`

Learn how to interact with remote repositories and users

- Retrieve and share code: `git (clone, pull, push, fetch)`

Very powerful system means there is inherent complexity

- It is entirely possible to use restricted feature set and be very productive!

Still worth spending some time reading about it!*

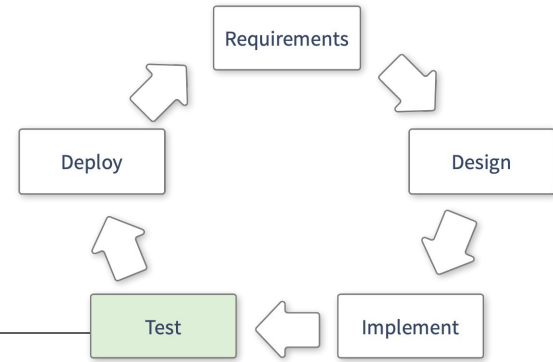


Git tutorials:

<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/>

Testing



What do we mean with tests?

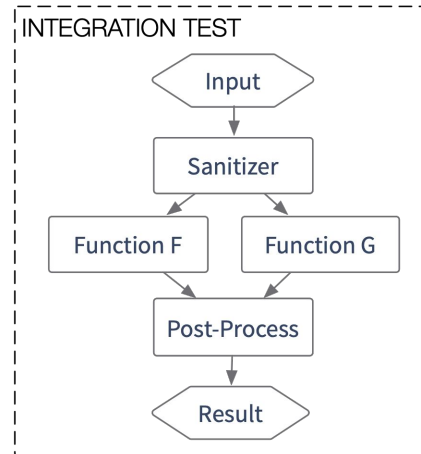
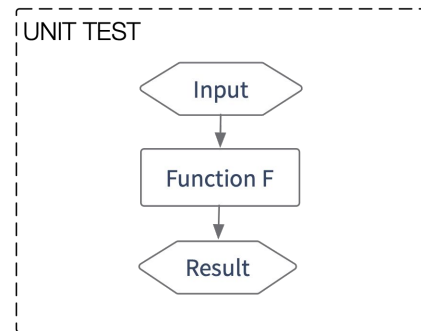
Different tests, different purposes:

- 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

Modifying code base that is well-covered by tests is infinitely more fun than one that is almost untested!

Do not confuse it with verification

Checks if specifications are met



Writing good tests is hard

How to come up with tests?

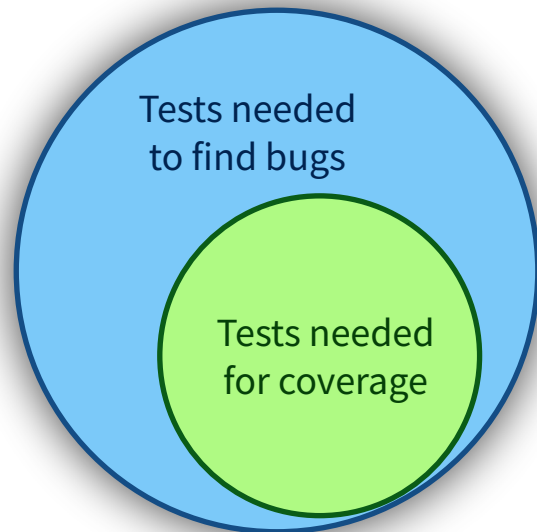
- What should the algorithm do?
 - Check if well defined input produces correct result
- How should the algorithm fail?
 - Check if wrong input fails in the way you want

You'll ~~probably~~ miss corner cases

- Once you discover them, implement a test!
 - **Only let a bug hit you once**
- Have 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
```

```
$ # No error → All tests passed!
```

```
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

1 items had no tests:

```
__main__
```

1 items passed all tests:

```
2 tests in __main__.fib
```

2 tests in 2 items.

2 passed and 0 failed.

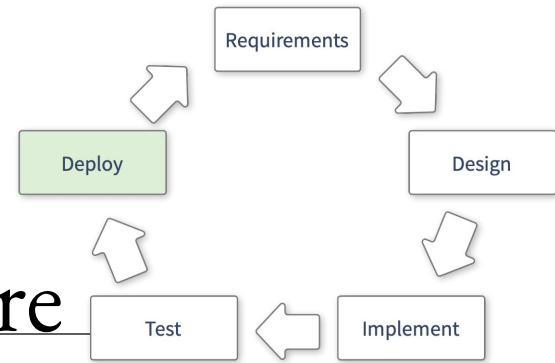
Test passed.

```
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()
```

See <https://docs.python.org/3/library/doctest.html> for more examples and explanations!

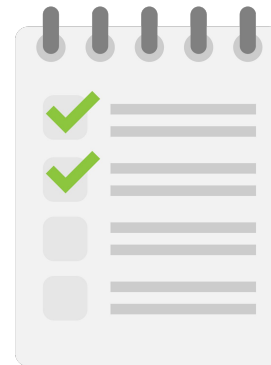
Deploying your software



Releasing the Software

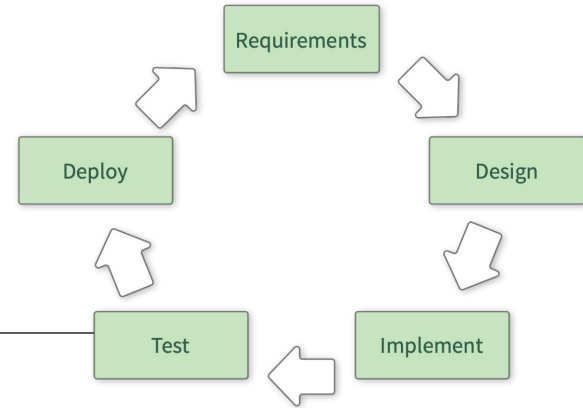
When you release your software:

- Tag the repository
 - Defines a common "checkpoint"
- Test in the target environment
 - In e.g., fresh virtual machine or docker container
 - If you test locally you might be unknowingly using private files/variables
- Create/release accompanying documentation
 - Produce e.g., Doxygen pages
 - Update wikis or whatever documentation pages you use (tag a new version)
 - Make sure all examples work



Ideal case: All this is done automatically!

Automate

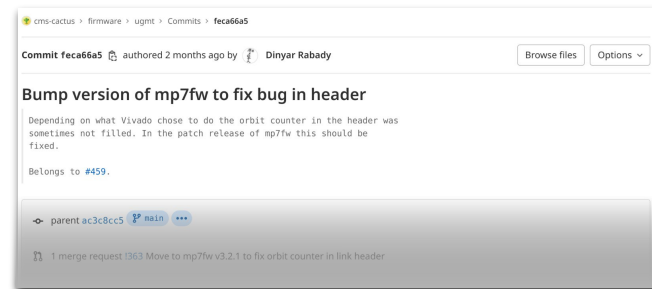


Goal

We would like to

- Commit some code

and then **automatically** make it ready for deployment.



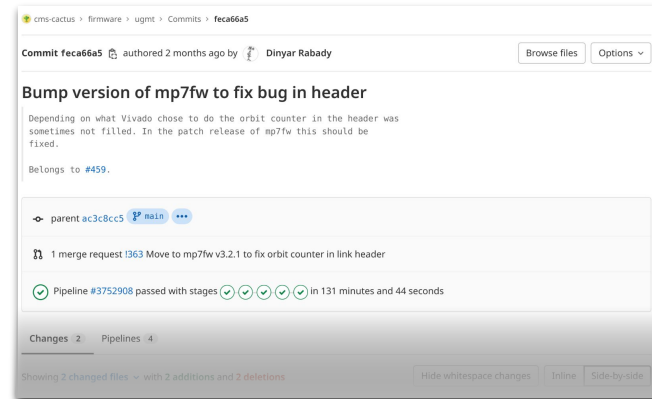
Goal

We would like to

- Commit some code

and then **automatically** e.g.,

- Test
 - e.g., Unit tests
- Build
- Package the products and name them
 - e.g., RPM, JAR, tar/zip archives, ...
- Test again
 - e.g., Integration tests
- Provide the packages for deployment
 - OR... deploy them immediately



uGMT firmware

Repository of bitfiles for the upgraded Global Muon Trigger (point 5).

uGMT bitfiles

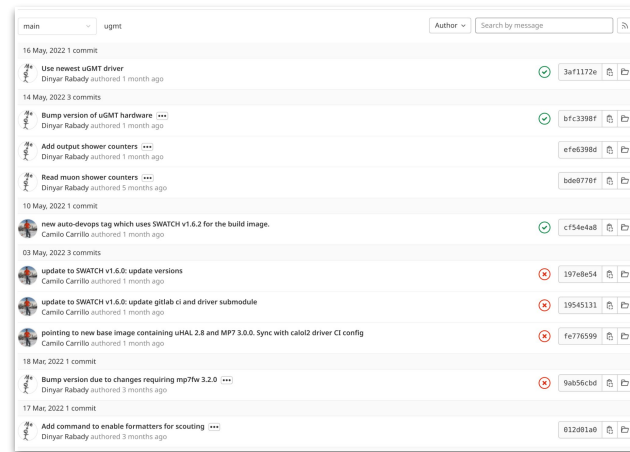
Bitfiles for MP7 XE

[uGMT_XE_v6_1_0_unstableMP7FW](#)
[uGMT_XE_v6_1_0_patch2](#)
[uGMT_XE_v6_1_0_patch1](#)
[uGMT_XE_v6_1_0](#)
[uGMT_XE_v6_0_1](#)
[uGMT_XE_v6_0_0_patch2](#)
[uGMT_XE_v6_0_0_patch1](#)
[uGMT_XE_v6_0_0](#)
[uGMT_XE_v5_0_1_patch4](#)
[uGMT_XE_v5_0_1_patch3](#)
[uGMT_XE_v5_0_1_patch2](#)
[uGMT_XE_v5_0_1_patch1](#)
[uGMT_XE_v5_0_1](#)
[uGMT_XE_v5_0_0](#)
[uGMT_XE_v4_1_0](#)
[uGMT_XE_v4_0_0_patch2](#)

[uGMT_XE_njshuly](#)
[uGMT_XE_branch_vivado-2018p3](#)
[uGMT_XE_branch_timing-experiments](#)
[uGMT_XE_branch_test-mp7fw-v3p2p0](#)
[uGMT_XE_branch_speed-up-synth](#)
[uGMT_XE_branch_scouting-counter-fix](#)
[uGMT_XE_branch_patterns-from-ellab](#)
[uGMT_XE_branch_no-zero-suppression](#)
[uGMT_XE_branch_no-valid-bits-from-masked-chans](#)
[uGMT_XE_branch_name-bitfiles-from-branches](#)
[uGMT_XE_branch_modified-mp7-zero-suppression](#)
[uGMT_XE_branch_latest-in-ci](#)
[uGMT_XE_branch_kalman-con](#)
[uGMT_XE_branch_full-jbb](#)
[uGMT_XE_branch_fix-use-lagged-patterns](#)
[uGMT_XE_branch_fix-orbitcounter-for-header](#)
[uGMT_XE_branch_fix-dsv-position](#)
[uGMT_XE_branch_fix-counters](#)
[uGMT_XE_branch_feature-mp7fw-scout-link-header](#)
[uGMT_XE_branch_feature-hwtest-via-direct-connection](#)
[uGMT_XE_branch_feature-hadronic-showers](#)
[uGMT_XE_branch_feature-emt-displ-info](#)
[uGMT_XE_branch_feature-dockers-builds](#)
[uGMT_XE_branch_environments-for-b40-deployment](#)
[uGMT_XE_branch_displaced-muons](#)
[uGMT_XE_branch_chore-u8mscripts6p1p1](#)
[uGMT_XE_branch_chore-jpbh-dev2021](#)
[uGMT_XE_branch_chore-disable-iso-bit-computation](#)

But why?

- Reduces work
 - **Committing** to repository is enough
 - Tests are run
 - Builds are performed
 - Product provided for download
- Possibility for **non-experts** to **build complex or resource intensive** software or firmware
 - **Added bonus:** Per construction there is a recipe to build your code in the repository!
- **Consistent record** of firmware development
 - With **indication** whether particular **revision is known to be good**
- Gives **confidence**
 - **"Spatial"**: Does my work build correctly on other machines?
 - e.g., Did I commit everything required to my repository?
 - **"Temporal"**: Is this particular commit from two years ago supposed to work?
- Ensures **consistency** of a given package
 - We **know** that this was built with *these* libraries and in *this* environment



Interlude: Gitlab CI

```
$ cat .gitlab-ci.yml
image: gitlab-registry.cern.ch/scouting-demonstrator/scone/rpm-builder:v1_0_0

build_rpm_tag:
  stage: build
  only:
    - tags
  script:
    - bash build.sh
    - bash package.sh
  artifacts:
    name: "scone-${CI_JOB_NAME}_${CI_COMMIT_TAG}"
    paths:
      - cms-scone*.rpm
      - info.json
    expire_in: 1 week
```

Interlude: Gitlab CI

scouting-demonstrator > SCONE > jobs > #22307454

passed

Job build_rpm_tag triggered 2 weeks ago by Dinyar Rabady

1 Running with gitlab-runner 14.10.1 (f761588f)

2 on default-runner-86545f9dd6-zgpcd 8xy--Bqv

3 feature flags: FF_DISABLE_UMASK_FOR_DOCKER_EXECUTOR:true

4 Resolving secrets

5

6 Preparing the "docker" executor

7 Using Docker executor with image gitlab-registry.cern.ch/scouting-demonstrator/scone/rpm-builder:v1_0_0 ...

8 Authenticating with credentials from job payload (GitLab Registry)

9 Pulling docker image gitlab-registry.cern.ch/scouting-demonstrator/scone/rpm-builder:v1_0_0 ...

10 Using docker image sha256:3ffe3f3335dfb435125cb4ab798d0f3555a38660abc382f21ce7ca3b38f9ca4c for gitlab-registry.cern.ch/scouting-demonstrator/scone/rpm-builder@sha256:f488c85e101f16af006fd14a0316560ec95920a85ad5dded55ebd9830f507c51 ...

11 Not using umask - FF_DISABLE_UMASK_FOR_DOCKER_EXECUTOR is set!

12

13 Preparing environment

14

15

16 Getting source from Git repository

17

18

19

20

21

22

23 Executing "step_script" stage of the job script

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

00:00

00:12

00:01

00:01

00:28

00:03

00:01

build_rpm_tag

Duration: 51 seconds

Finished: 2 weeks ago

Timeout: 1h (from project)

Job artifacts

These artifacts are the latest. They will not be deleted (even if expired) until newer artifacts are available.

Keep Download Browse

Commit e603803d

Make multiple read/write requests possible

Pipeline #4059136 for v1.2.0

build

→ build_rpm_tag

For resource intensive builds/tests or other special requirements for the running machine: Can set up specialised runners (i.e., "computers that run your jobs").

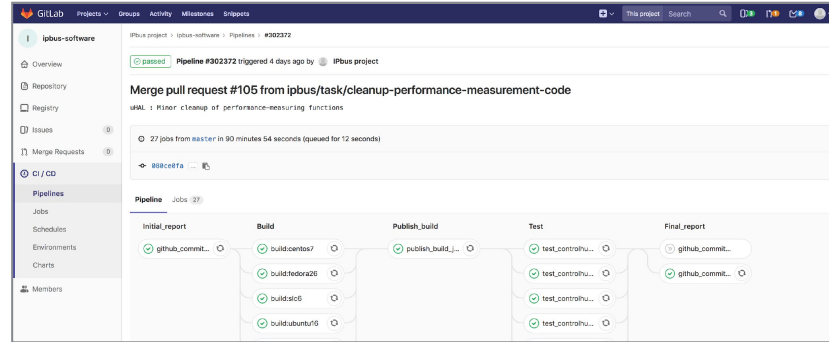
See <https://docs.gitlab.com/ee/ci/> for a complete documentation!

D. Rabady (CERN)

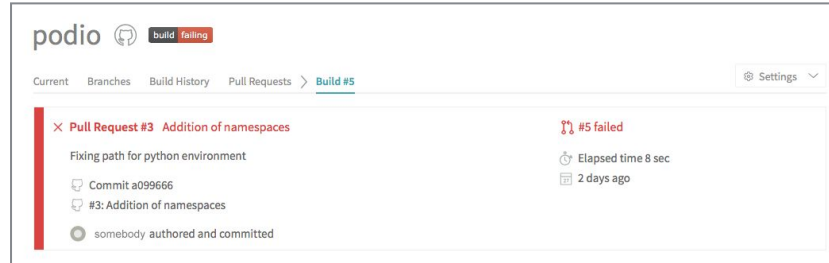
ISOTDAQ 2023. Istanbul University, Istanbul, Turkey. 21 June 2023

66

Many solutions available for this...



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



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

Also <https://www.jenkins.io/>, <https://circleci.com/>,
<https://docs.github.com/en/actions/automating-builds-and-tests/about-continuous-integration>, ...

Words of advice

- Well designed **build tools are crucial**
 - Invaluable even when not using continuous integration
 - Essential when using it
- Some measure of discipline required to get the most out of this
 - Build system requires some time to set up
 - **Very good investment**, but initially somewhat frustrating
 - Continuous integration most useful if actually used regularly
 - If you put each code change into your repository you can later come back and check what was the breaking change
- Spend time finding the right development process for you(r team)
 - Use of tags?
 - Use of branches?
 - Pull/Merge requests?
 - ...
 - Lots of "standard" processes around (Gitflow, trunk-based,...)
 - Significant experience probably also around your lab (and coffee is cheap)
- Above all: **Don't get overwhelmed by the possibilities!**
 - Start simple!

General tips & pointers

Learning about software development

Coursera and Udacity

- <https://www.coursera.org/courses>
 - Large variety of courses
 - Not only technology / programming
 - Also physics, biology, economics... and more
 - Also in different languages
- <https://www.udacity.com/courses/all?price=Free>
 - Mixed courses: Some free, recently switched to a paid model with monthly fees

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

- e.g.: [Programming Paradigms, Stanford University](#)
- <https://www.edx.org/search?tab=course>

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
- Nothing is free
 - You'll have to invest some effort to learn, but most tools shown here bring benefits very early in the learning curve!
- Many more tools to discover, lots of fun to be had with them!

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
 - **Beware:** If you use a service like lxplus you get a machine from a pool (e.g. lxplus769) → Tmux/screen is running on that one!
- Tune your .bashrc / .bash_profile to get a more useful prompt
 - You can try e.g. [oh-my-bash](#) or [oh-my-zsh](#), depending on the shell you are using
- Try out vim/emacs/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!

More Random Things

Making git (more) fun to use

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>

```
[user]
  name = Dinyar Rabady
  email = dinyar@cern.ch
[core]
  excludesfile = /home/dinyar/.gitignore_global
  editor = vim
[color]
  status = auto
  branch = auto
[push]
  default = matching
[merge]
  conflictstyle = diff3
[http]
  emptyAuth = true
[init]
  defaultBranch = main
[pull]
  ff = only
```

```
* feca66a - (tag: ugmt_v6_1_0_patch1) Bump
* ac3c8cc - Move hw-test to host connected
| * c09255d - (refs/stash) WIP on hadroni
| \
| | * 5754477 - index on hadronic_showers:
| /
| * d5ce4a1 - Modified opt design directive
| * 23e0151 - Change route design directive
| * cc1944f - Improve artifact handling for
| * 4daf292 - Add counter for output shower
| * ff8f7e3 - Add rate counters for input s
| * 2053ae2 - Extend tesetbench to read sho
| * 9157704 - Implement hadronic showers (3
| /
* eb39fe4 - (origin/feature/mp7fw-scout_lin
```

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
```

Integrate git with your shell

- Tab-completion and git information shell prompt

Git bash shell integration:

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

Or try git integration in oh-my-zsh:

<https://github.com/ohmyzsh/ohmyzsh>

```
; █ src/gt-algorithm (feature/vu13p_dc ⚡) lxplus737
```

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>