Good Practices

Chris, ~~Chris~~ & Karina do Data Science

DAP
Capability
And
Training
Support

Agenda

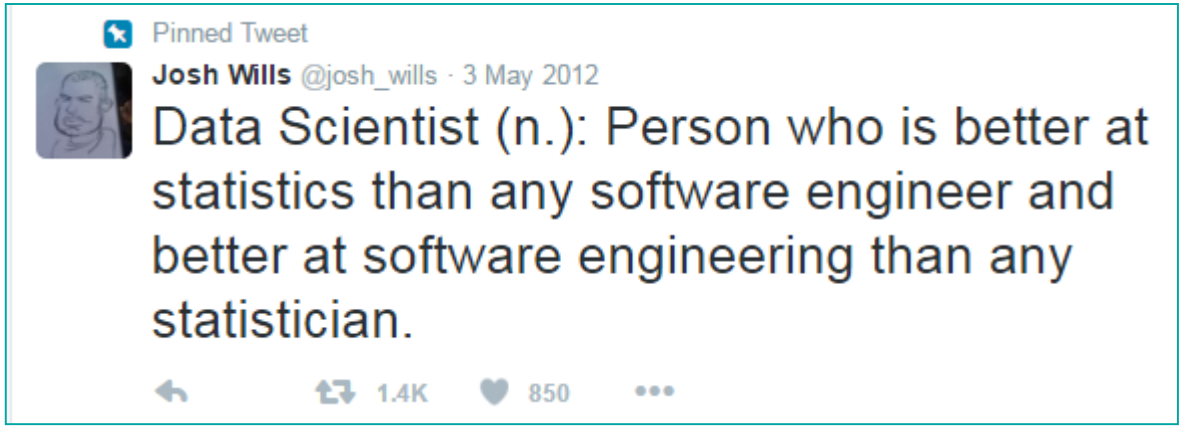
DAP Capability And Training Support

- Introduction to Good Practices (why and what)
- Key Themes
- Walkthrough Levels of Good Practices
- Anything Else? – Your Questions.

Why Good Practices?

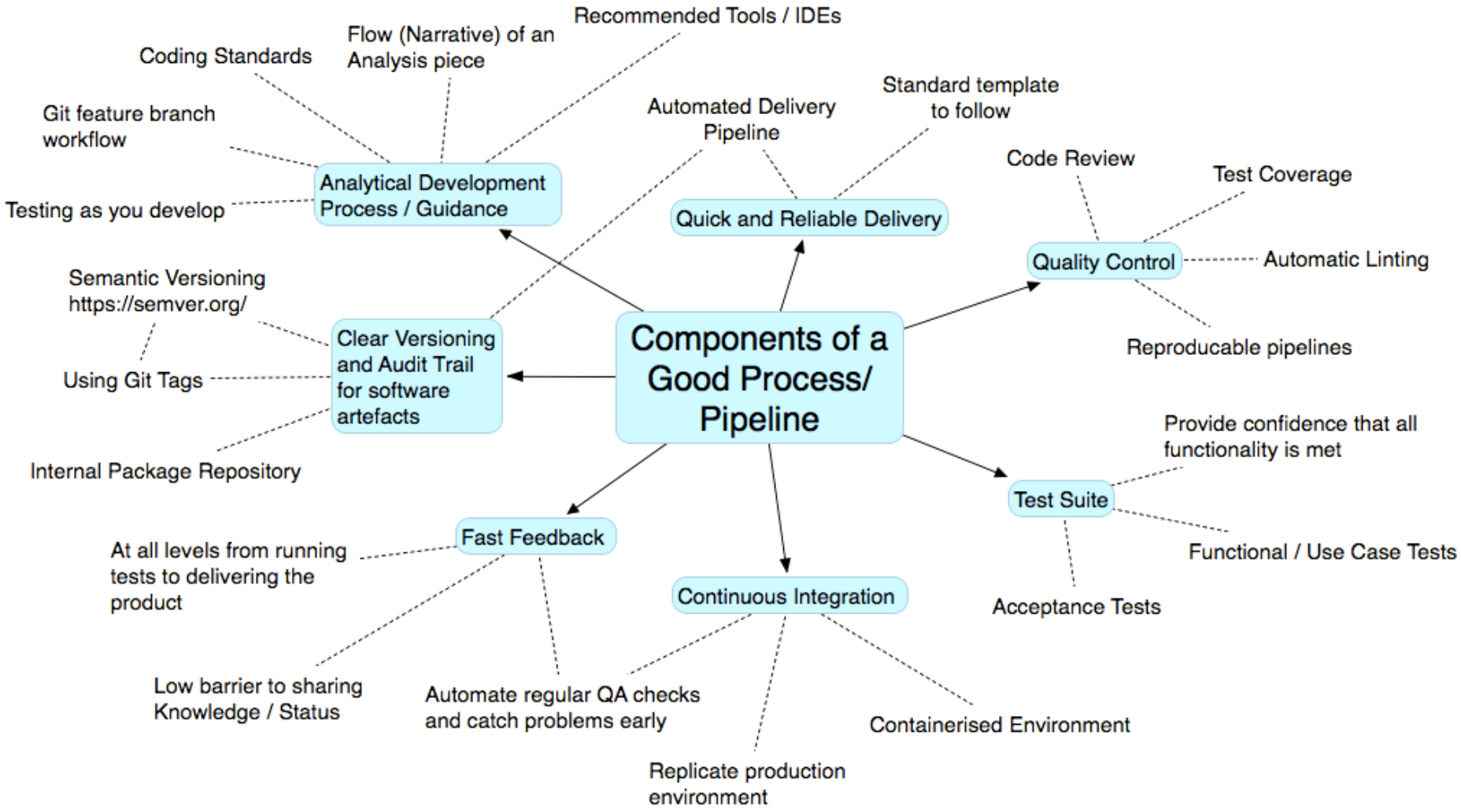
DAP
Capability
And
Training
Support

- Help inform more effective ways of working to reduce the friction in collaborative working and boost productivity.
- Provide guidelines and advice born from hard earned experience.
- Many programmers/analysts are self taught. Lots to learn on that journey.



What is Good Practice?

- It can cover **a lot** of things all working together!
- An end goal to iterate towards.
- No one size fits all, and doesn't come for free.
- We will focus on a few of the key themes, and show how to build up to them.



Key Themes

DAP
Capability
And
Training
Support

Consistency in Process

- A standard workflow to provide a common high level language and understanding.
- Easier collaboration & communication of ideas, reporting progress etc.

Consistency in Code

- Writing “Good Code”:
  - Works as intended, and comes with proof that it does so (tests).
  - Clearly structured so that it is easy to comprehend and maintain.
- Also: following conventions, peer reviewing code, good layout, regular refactoring.

Key Themes

DAP
Capability
And
Training
Support

Reproducibility

- Quality assurance and confidence building as you develop.
- Managing and tracking changes in:
  - Source code,
  - Data,
  - Dependencies,
  - Parameters / Configuration

Reusability

- Building reusable tools / components that others can use or build upon.
- Documentation and packaged releases, writeups of analysis, plan for ongoing development and contributions.

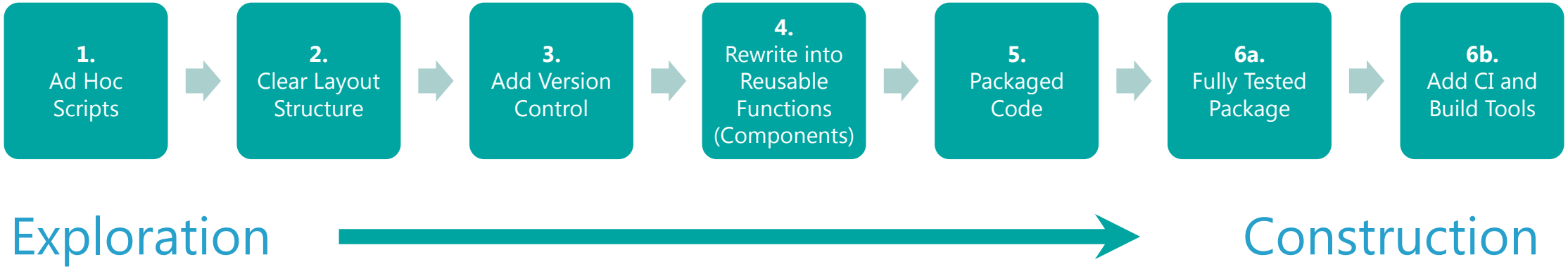
Levels of Good Practices

DAP
Capability
And
Training
Support

Reproducible Analytics Pipelines in DAP

DAP Capability And Training Support

- Reproducible Analytical Pipelines (Article Link)
  - Wider GSS initiative to streamline the production of regular statistical reporting
  - Focused on R tooling and free to use external infrastructure (GitHub / Travis etc.)
- Adapted below to be more general and DAP compatible



1. Ad Hoc Scripts

- Everything starts somewhere
- In exploratory mode, investigating, building familiarity.
- Focus is on learning and reducing uncertainties
- Code may not be well named or organised.
- Certainly not ready for sharing!

```
|- project_name/
   |- myscript.py
   |- myscript2.py
   |- eda.py
   |- untitled.py
```

2. Clear Layout Structure

- Use meaningful folder/filenames
- Separate inputs, code and the outputs.
- Separate stand alone scripts from 'ad hoc' exploratory code.
  - `analysis/` for experiments and ad hoc work
  - `scripts/` for scripted repeatable jobs / tasks
- Have one place to go to:
  - Find out more about the project
  - Be able to rerun the analysis
- Begin to implement good coding standards

```
|- project_name/
   |- analysis/
   |- data/
   |- scripts/
   |- results/
   |- run.py
   |- README.md
```

3. Add Version Control

- Encouraged to have all code under version control from the start.
- As project is now much more suitable for sharing with others, version control is a must.
- Allows use of GitLab for collaborative development.
  - Tracking Issues
  - Raising merge requests
  - Peer Reviewing code

```
|- project_name/
   |- analysis/
   |- data/
   |- scripts/
   |- results/
   |- run.py
   |- README.md
   |- .gitignore
```

4. Re-write to Reusable Functions

DAP
Capability
And
Training
Support

- Refactor analysis scripts to a higher level
- Lower level implementation wrapped into functions and called by analysis scripts.
- Benefits:
  - More Reusable
  - More Testable
  - Easier to reason / follow analysis scripts
  - More space to focus on documenting the analysis performed.

```
|- project_name/
   |- analysis/
      |- experiment.py
   |- data/
   |- scripts/
      |- __init__.py
      |- load.py
      |- clean.py
      |- plot.py
   |- results/
   |- run.py
   |- README.md
   |- .gitignore
```

5. Package up the Code

- Packaging is about wrapping up the set of functions and scripts into a standard format.
- Makes it possible to:
  - Track library dependencies and versions
  - Easily install code + dependencies onto another system
- Different Languages have different structures and tooling
  - R Packages
  - Python Packages

```
|- project_name/
  |- analysis/
     |- experiment.py
  |- data/
  |- package/
     |- project_name/
        |- __init__.py
        |- load.py
        |- clean.py
        |- plot.py
     |- setup.py
   |- results/
   |- run.py
   |- README.md
   |- .gitignore
```

6a. Tested Package

- Develop a set of tests to provide a proof that your functions work as intended.
- Testing framework libraries:
  - Pytest in Python
  - testthat in R
- Tests should be:
  - Fast
  - Independent
  - Repeatable (deterministic)
  - Self-validating (no manual steps)
  - Thorough (How much do you trust they cover everything?)

```
|- project_name/
  |- analysis/
     |- experiment.py
  |- data/
  |- package/
     |- project_name/
     |- tests/
        |- test_load.py
        |- test_clean.py
     |- setup.py
   |- results/
   |- run.py
   |- README.md
   |- .gitignore
```

6a. CI and Build Process

- Use a continuous integration tool to automate the running and reporting of tests as you develop.
- Combined with merge requests in GitLab, provides a robust, automated QA check for the code.
- Can also automate follow on deployment tasks using CI tools.
- Not yet available in DAP, several teams are currently looking at the best way to add this.
- Build recipe kept alongside the code.

```
|- project_name/
  |- analysis/
     |- experiment.py
  |- data/
  |- package/
     |- project_name/
     |- tests/
        |- test_load.py
        |- test_clean.py
     |- setup.py
   |- results/
   |- run.py
   |- README.md
   |- .gitignore
   |- Jenkinsfile
```

Levels of Good Practices

DAP
Capability
And
Training
Support

Level	Description	Output Produced using R/Python	Coding Standards	Version Control	Peer Review	Documentation of Functions	Automated Data QA	Dependency Packages Managed	Unit Testing of Functions	Unit Testing Guaranteed
1	Ad hoc Scripts	✓	✗	✗	✗	✗	✗	✗	✗	✗
2	Clear Layout Structure	✓	✓	✗	✗	✗	✗	✗	✗	✗
3	Add Version Control	✓	✓	✓	✓	✗	✗	✗	✗	✗
4	Rewrite into Reusable Functions (Components)	✓	✓	✓	✓	✓	✓	✗	✗	✗
5	Packaged Code	✓	✓	✓	✓	✓	✓	✓	✗	✗
6a	Fully Tested Package	✓	✓	✓	✓	✓	✓	✓	✓	✗
6b	CI and Build	✓	✓	✓	✓	✓	✓	✓	✓ (+75%)	✓

Questions?

References

- Data Explorers Support Page for Good Practices
- RAP Paper
- Slides to be made available on Data Explorers Support Page

- General enquires mailbox: DAPCATS@ons.gov.uk
- Feedback survey to follow.