

# Before we start...

Go to the shared Google Doc,  
follow the set up instructions:

<http://bit.ly/CESSDA-CES2018>

cessda eric

Consortium of European Social Science Data Archives  
European Research Infrastructure Consortium

# CESSDA Technical Infrastructure

Developing, Testing, and Deploying Tools  
and Services

CESSDA Expert Seminar (CES2018)

Ljubljana, Slovenia  
September 26 2018

Steve Crouch, Research Software Group  
Lead, Software Sustainability Institute  
[s.crouch@software.ac.uk](mailto:s.crouch@software.ac.uk)

cessda eric



# Software Sustainability Institute



Software  
Sustainability  
Institute

*Cultivate world-class research with software*

cessda eric

# What we'll do Today

- Brief recap of CESSDA's Technical Architecture and Software Maturity Levels
- Use CESSDA's infrastructure and development practices to
  - Contribute a fix to an example Java application repository
  - Test our fix and extend the existing tests
- For those new to version control and continuous integration
- But also to give you experience with CESSDA infrastructure

# Need Help?

- Say hi to your neighbours
- Ask in the GoogleDoc
  - <http://bit.ly/CESSDA-CES2018>
- Sticky notes
- Our helpers – Peter, Wilko



***Session 1:***  
***Summary of CESSDA***  
***Technical Infrastructure***

# Why have a defined Technical Architecture?

- Community development at scale is *hard*
  - Have to balance innovation with usability, sustainability, and stability
- You need more than a powerful infrastructure
  - Community must develop in a defined, understood, and consistent manner
  - Community must know what is expected of them
- Interoperability between infrastructure, components, and *people*

# The CESSDA Technical Architecture

- A set of principles and guidelines based on established best practice
  - Useful in any software development
- Many infrastructure documents a hard read – not CESSDA's!
  - Suitably high-level, common sense
  - A good read for developers
  - Not too prescriptive
- Following the Technical Architecture helps ensure compliance
  - But also genuinely useful for you
- Become familiar with it



# Interoperability Characteristics

- Again, embody established development best practices
- Five characteristics
  1. Loosely coupled but coordinated
  2. Sustainable
  3. Extensible
  4. Maintainable
  5. Standards based

# Software Development Guidelines I

- Things you should do to improve acceptance
- Code structure
  - Solid, commented, minimum complexity, DRY
  - Use code conventions, string localisation, outside config
- Environment specific information
  - Deploy new developments quickly
  - Use same tools throughout development, testing, deployment
  - Use containers

# Software Development Guidelines II

- Use Technical Architecture as primary guide for decisions
- Document throughout lifecycle
  - Make operational, development, end-user docs available
  - Keep them maintained!
- Source code management
  - Bitbucket SCM mandated
  - Each development has own CESSDA repository
  - Each repo linked to CESSDA Continuous Integration
- Security and privacy: privacy impact assessment expected

# *Exercise: Evaluate your (Potential) CESSDA Software*

- To familiarise yourselves with the CESSDA Maturity Levels
- Use the CESSDA Maturity Levels questionnaire
- Assess software that either
  - You are developing (or will develop) for CESSDA
  - You are generally familiar with
- They get you thinking early about software quality in a number of useful dimensions!

*Session 2:*

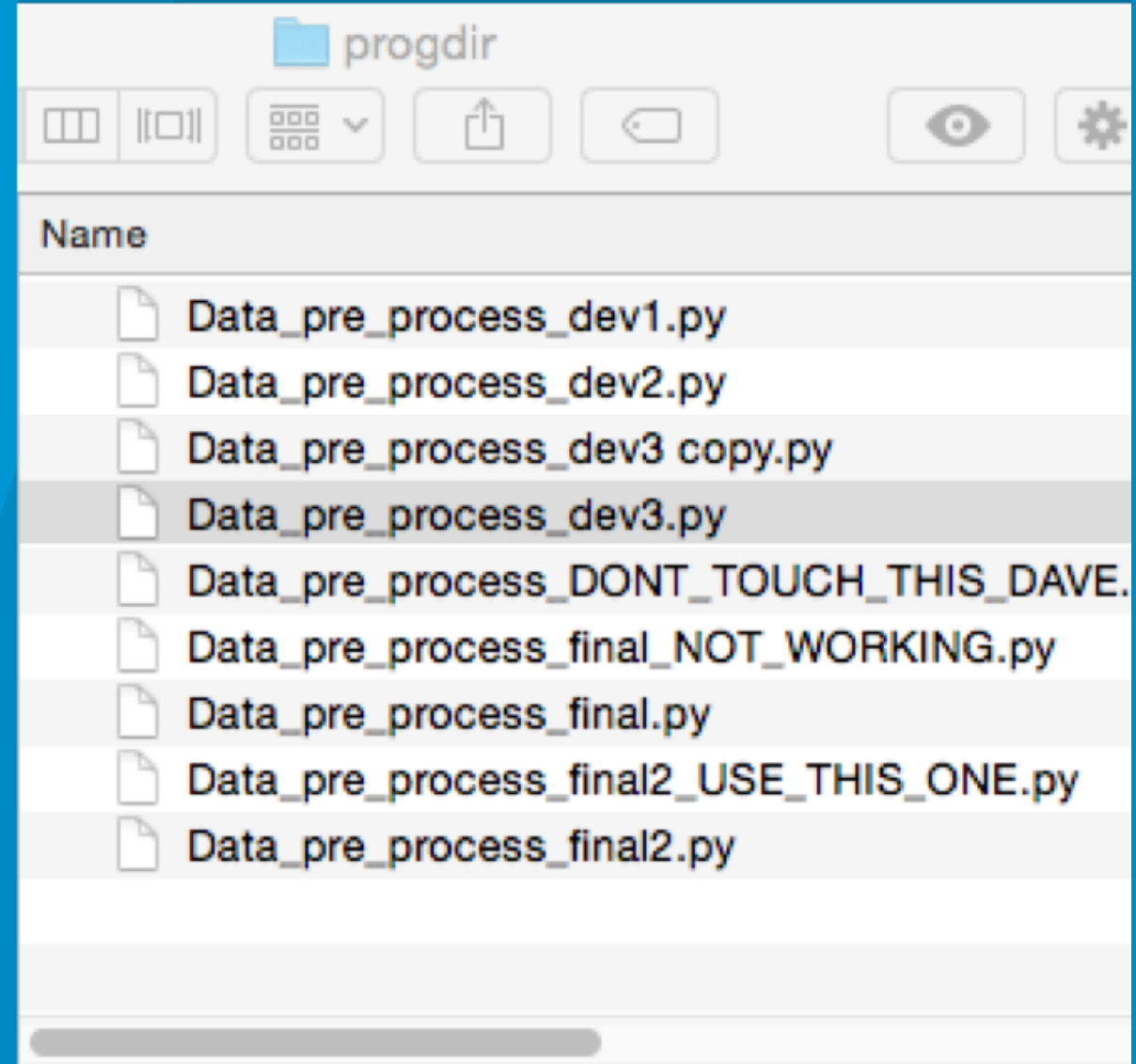
*Application development in  
CESSDA*

# Brief introduction to Version Control

- AKA revision control, source control, source code management (SCM)
- Track changes to files
- Maintain complete history of development

# Key Advantages I: A More Efficient Backup

- Who's been in this situation?



# Key Advantages II: Reproducibility

- Complete history – so can retrieve any version of any files
  - Or sets of files, e.g. a software package
- Reproducibility becomes simpler
  - Reproducing results of publications

*“If you’re not using version control, whatever else you may be doing with a computer, it isn’t science”*

- Greg Wilson, co-founder of Software Carpentry



# Key Advantages III: Aids Collaboration

- **Crucial** tool in team development
- Professional software developers use VC
  - They know who has changed what and when
- Every large software development project relies on VC
  - Most programmers use it for their small jobs as well
- Version control is not just for software
  - Papers, small data sets
  - Anything that changes over time, or needs to be shared

# Distributed or Centralised SCM?

- Two types of SCM
  - Centralised: only one master copy of the repository exists, which has all history, e.g. Subversion or CVS
  - Distributed: each developer in team has own copy of repo, which you synchronise with the authoritative central repo, e.g. Git or Mercurial

# So what SCMs does Bitbucket support?

- Bitbucket uses Git (also Mercurial)
- Written by Linus Torvalds to manage Linux source code development
- Very powerful
- Distributed SCM

# Requesting a CESSDA code repository

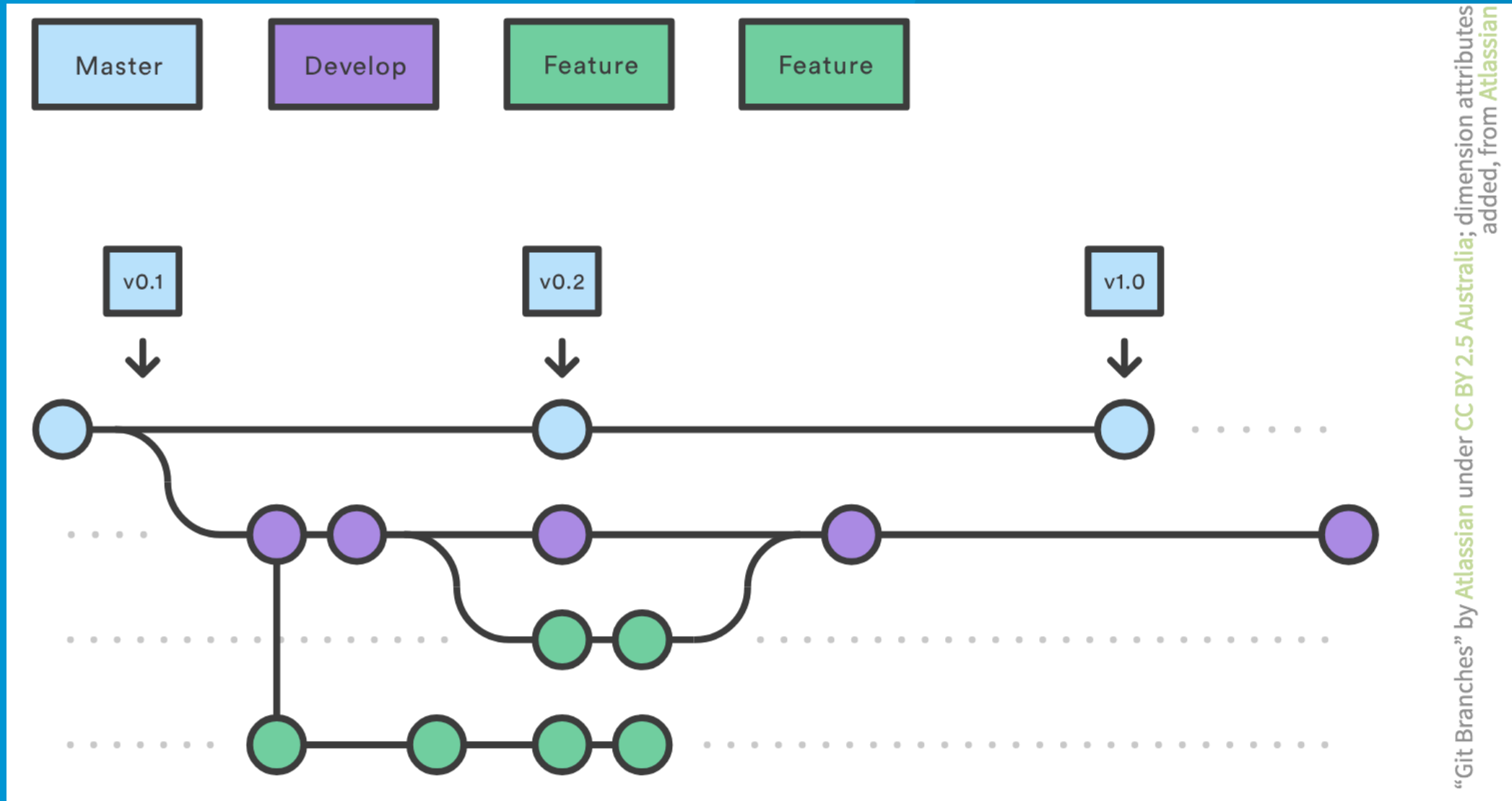
- If developing own code from scratch, need to request CESSDA Bitbucket repository
  - Form:  
[CESSDA Research Infrastructure Contributor License Agreement form](#)
  - Supply name, email, Bitbucket user account, project/component name, repository description, coding language, Bitbucket account names of other contributors
- You'll get an email confirming its creation

# *Session 2: Practical...*

# The Fibonacci scenario

- A mathematically generated sequence of numbers
- Starts with 0 and 1, each following number being an addition of the previous two numbers in the sequence:
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ..
- We'll use a Java program that gives you the Fibonacci number at a particular index (starting with 0) in the sequence
  - e.g. asking for the 6th number gives us 8

# Feature Branch Workflow



# *Session 2: Practical continued...*



# Exercise: Diagnose and Fix the Problem

- Have a look at the source code for Fibonacci in `src/main/java/math/Fibonacci.java`
- See if you can find the problem and correct it, then save the file
- Recompile the code (*`mvn compile`*) and then manually run the code a few more times with different arguments to check the results of other inputs, e.g:
  - `java -cp target/classes math.Fibonacci 1`
  - `java -cp target/classes math.Fibonacci 2`
  - `java -cp target/classes math.Fibonacci 6`
  - ...

***Session 3:***  
***Using automation to build,  
test and deploy our code***

# Benefits of Automation

- Productivity: automating code build and test saves time
- Improvement: running automatic build and test frequently can find more bugs
- Reproducibility: automated process reduces mistakes from manual error, and potentially incorrect results
- Reuse: automation shows others how to build and test your software

# The 'Holy Grail' of Automation

1. Provide automated build process: easier and quicker to validate changes
  2. Provide unit tests: check if changes break anything
  3. Join together – automated build and test: a *fail fast* environment for community development
  4. Use Continuous Integration (CI): automate building, testing, even deployment, of code *as changes are made*
- Each step awards new benefits immediately! cessda eric

# How does CI Work?

1. You make and commit code changes to your repository
  2. A CI server is configured to notice these commits and independently checks out your code from revision control and performs a number of predefined steps automatically (like build, test, deploy).
  3. Reports produced, e.g.
    - Ongoing progress as build, test, deploy are attempted
    - Final success/failure summary, along with errors encountered
- Expected to use CI from the start for CESSDA

# CI within CESSDA

- CESSDA uses Jenkins
  - Popular, extensible, open source CI server
  - When changes are committed, it runs a Jenkins *job*
  - Very flexible deployment
- You can define automated steps within a Jenkins *pipeline*
  - Defined within a *Jenkinsfile* file in repository root
  - Two approaches:
    - Declaratively: define high-level steps for Jenkins job
    - Scripted: more flexible, define steps using Groovy language
- We'll show enough to get started - declarative method  
@cessda eric

# Using Jenkins to Build and Deploy Code

- Jenkinsfile invokes the Maven commands we used previously
  - NB: depends on Jenkins CI infrastructure having same tools installed
- Structured around pipeline, containing
  - At least one *agent*: environment to run job, e.g. Docker image, or *any*
  - Number of *stages*: each stage (e.g. build, test) contains sequence of commands, called *steps*
- Let's take a look at our repo's Jenkinsfile now

# *Session 3: Practical...*



# *Question: Confess!*

- Why don't you write tests?
  - “I don't write buggy code”
  - “It's too hard”
  - “It's not interesting”
  - “It takes too much time and I've research to do”

# What Testing Gives You

- What testing gives you:
  - Confidence that your code does what it is supposed to
  - That your research is built on a solid foundation
  - Ability to detect, and fix, bugs more quickly
  - Correct code (bugs caught early in the cycle)
  - Confidence to refactor or fix bugs without creating new bugs
  - Examples of how to use your code

*“If it's not tested, it's broken”*  
- bittermanandy, 10/09/2010

# Examples of Unit Testing Frameworks

- Fortran: FRUIT, pFUnit
- R: RUnit, testthat
- MATLAB: Unit Testing Framework
- .NET: csUnit
- PHP: PHPUnit, PHP Unit Testing Framework
- Python: Nose, Autotest, PyTest
- Java: *jUnit*

# *Session 3: Practical continued*

# *Exercise: Write your own Unit Tests*

- Consider what aspects of your code should be tested
- Write 2-3 jUnit tests, adding to the FibonacciTest class, to check your code handles them correctly
- Whilst writing these, also create a test that deliberately fails so you can see what happens to the results

# *Session 3: Practical continued...*

# Test Driven Development (TDD)

- Writing tests good way to define how functions behave
- Instead of writing tests afterwards, write them first
  - Tests become a 'contract' for how function should work
- Process
  - Write some unit tests for a function that doesn't exist yet
  - Write that function
  - Modify it until it passes all of the tests
  - Clean up (or refactor) the function

# TDD: Red, Green, Refactor

1. Get a red light (i.e., some failing tests)
  2. Make it turn green (i.e., get something working)
  3. Then clean it up by refactoring
- This cycle should take anywhere from a couple of minutes to an hour or so. If it takes longer than that, the change being made is probably too large, and should be broken down into smaller (and more comprehensible) steps.



# *Exercise: Implement a new feature using TDD*

- Think of a new feature you would like to add to your code
- Write the unit tests for it first
- Then implement the feature in the code, and rerun the tests.
- Once successful, refactor your code as necessary
  - Make it more readable
  - Add comments for other developers (including yourself)
  - Ensure your tests still pass after refactoring!

# *Session 3: Practical continued...*

# *Wrap-up*

cessda eric

# Key Points

- We've covered quite a bit!
- Develop according to CESSDA's Technical Architecture
- Adhering to Technical Architecture recommendations greatly increases chances of acceptance
- Use CESSDA Software Maturity Levels questionnaire to assess readiness
- Take huge advantage of CESSDA Continuous Integration
- Include suitable suite of tests for your software [cessda eric](https://www.eric.ed.gov/?q=CESSDA)

# Start as you mean to go on

- Many projects deal with things like software documentation, licensing, and sustainability way too late
  - SSI and Southampton's RSG deal with these frequently
  - Much harder to address things later
- CESSDA's guidelines embody best practice
  - Work on them – and adopt good habits – early
  - Greatly increases chances of acceptance into CESSDA RI
  - Just use them for any development!

# Closing remarks

- Training materials online (with slides)
  - <https://softwaresaved.github.io/2018-09-26-cessda-training/index.html>
- Huge thanks to
  - Local hosts
  - John Shepherdson
  - Our volunteer helpers, Peter, Wilko
- Thank you!

# Thanks for listening

- Any questions?