

Software Engineering

Simon Scheidegger
simon.scheidegger@gmail.com

July 27th, 2017

Open Source Macroeconomics Laboratory – BFI/UChicago

Outline

- Replicable results
- Git repositories
- Unit testing

Replicable results

See e.g. Stodden et. al (2014) – Implementing Reproducible Research



Implementing reproducible research

- Computational tools are at the core of modern research.
- In addition to experiments & theory, the notions of **simulation** and **data-intensive discovery** are often referred to as **“third” (and fourth) pillars of science**.
- **For all its importance, computing receives perfunctory attention in training of new scientists and in the conduct of everyday research.**
- It is treated as an **inconsequential task** that students and researchers learn “on the go” with little consideration for ensuring computational results are trustworthy, comprehensible, and ultimately a secure foundation for **reproducible outcomes**.

Implementing reproducible research (II)

- Software and data are often stored with poor organization.
- Little documentation.
- Few tests.
- **Haphazard patchwork of software tools** is used with limited attention paid to capturing the complex work flows that emerge.
- **The evolution of code is not tracked over time**, making it difficult to understand what iteration of the code was used to obtain any specific result.
- Many of the software packages used by scientists in research are proprietary and closed source, preventing complete understanding and control of the final scientific results.
- **One node hour on a HPC system costs around >1 USD**

Result – status of the community

<http://www.federalreserve.gov/econresdata/feds/2015/files/2015083pap.pdf>

Is Economics Research Replicable? Sixty Published Papers from Thirteen Journals Say “Usually Not”

Andrew C. Chang* and Phillip Li†

September 4, 2015

Abstract

We attempt to replicate 67 papers published in 13 well-regarded economics journals using author-provided replication files that include both data and code. Some journals in our sample require data and code replication files, and other journals do not require such files. Aside from 6 papers that use confidential data, we obtain data and code replication files for 29 of 35 papers (83%) that are required to provide such files as a condition of publication, compared to 11 of 26 papers (42%) that are not required to provide data and code replication files. We successfully replicate the key qualitative result of 22 of 67 papers (33%) without contacting the authors. Excluding the 6 papers that use confidential data and the 2 papers that use software we do not possess, we replicate 29 of 59 papers (49%) with assistance from the authors. Because we are able to replicate less than half of the papers in our sample even with help from the authors, we assert that economics research is usually not replicable. We conclude with recommendations on improving replication of economics research.






JEL Codes: B41; C80; C82; C87; C88; E01

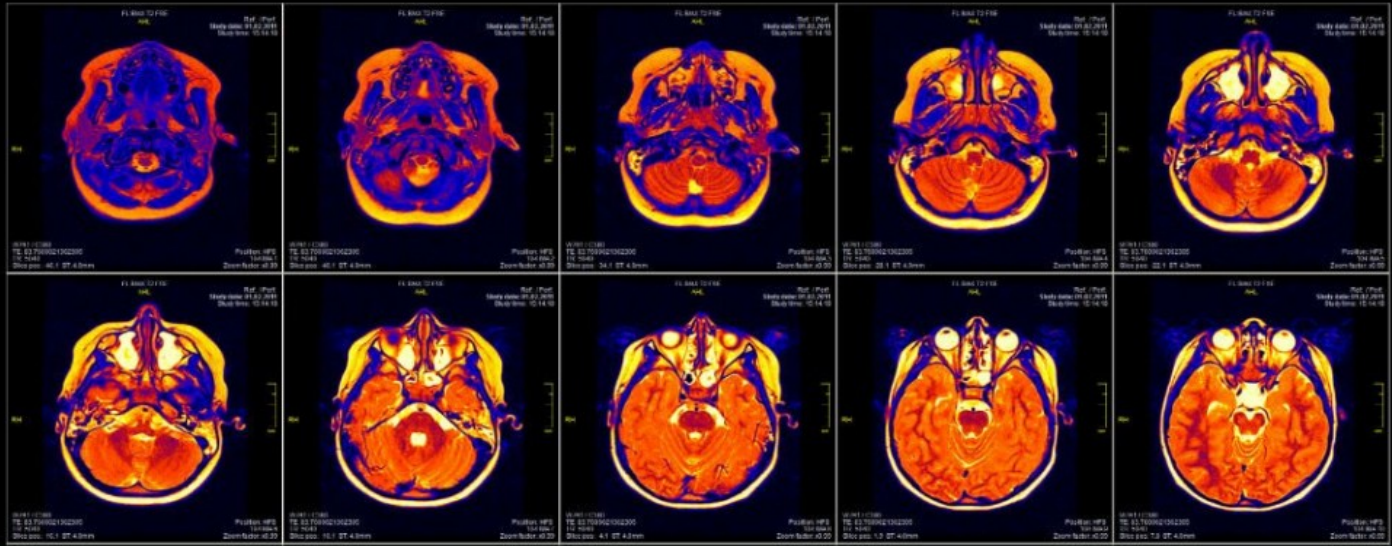
Keywords: Data and Code Archives; Gross Domestic Product; GDP; Journals; Macroeconomics; National Income and Product Accounts; Publication; Research; Replication

Flaws in other fields

<http://www.pnas.org/content/113/28/7900.abstract>

science
alert







Kondor8/Shutterstock.com

A bug in fMRI software could invalidate 15 years of brain research

This is huge.

BEC CREW 6 JUL 2016



There could be a very serious problem with the past 15 years of research into human brain activity, with [a new study](#) suggesting that a bug in fMRI software could invalidate the results of some 40,000 papers.

That's massive, because functional magnetic resonance imaging (fMRI) is one of the best tools we have to measure brain activity, and if it's flawed, it means all those conclusions about what our brains look like during things like [exercise](#), [gaming](#), [love](#), and [drug addiction](#) are wrong.

Common comp. research work-flow

- Matlab, Python etc. for **prototyping**.
- **Developing high-performance code in C++, Fortran,...**
- Postprocessing (e.g. Matlab – manually twiddling data & Figs.)
 - What if a couple of months later the researcher realizes there is a problem with the results? Will he be able to remember what buttons he clicked to reproduce the work flow to generate updated plots, paper manuscripts, etc.
 - **Publish-or-perish** mindset encourages to charge forward chasing the goal of an accepted manuscript.
 - “reproducibility” implies the repetition and thus the requirement also to move back, to retrace one's steps.

Computational research life cycle

- **Open source community has cultivated tools and practices** that, if embraced and adapted by the scientific community, will greatly enhance the ability to achieve reproducible outcomes.
 - *Individual exploration*: a single investigator tests an idea, algorithm, or question, likely with a small-scale test, dataset, or simulation.
 - *Collaboration*: if the initial exploration appears promising, more often than not some kind of collaborative effort ensues to bring together complementary expertise from colleagues.
 - *Production-scale execution*: large datasets and complex simulations often require the use of clusters, supercomputers, or cloud resources in parallel.
 - *Publication*: whether as a paper or an internal report for discussion with colleagues, results need to be presented to others in a coherent form.
 - *Education*: ultimately, research results become part of the corpus of a discipline that is shared with students and colleagues, thus seeding the next iteration in the cycle of research.

a) Individual work – open source

- For individual work, researchers use various **interactive computing environments**: Python, MATLAB, Mathematica, R,...
- These environments combine interactive, high-level programming languages with a rich set of numerical and visualization libraries. The impact of these environments cannot be overstated; **researchers use them for rapid prototyping, interactive exploration, and data analysis, as well as visualization.**
- However, they have limitations:
 - (a) some of them are **proprietary and/or expensive** (e.g. MATLAB, Mathematica);
 - (b) most are focused on coding in a single, relatively slow, programming language.
- While the use of **proprietary tools** is not a problem per se and may be a good solution in industry, it is **a barrier to scientific collaboration and to the construction of a common scientific heritage where anyone can validate the work of others and build upon it.**
- Scientists cannot share work unless all colleagues can purchase the same package.
- **Students are forced to work with black boxes they are legally prevented from inspecting.** Furthermore, because of their limitations in performance and handling large, complex code bases, these tools are mostly used for prototyping: researchers eventually have to switch tools for building production systems.

b) Collaboration

- For collaboration, researchers tend to use a mix of e-mail, version control systems (VCSs) and shared network folders (Dropbox, etc.) → Slack (slack.com)
- **VCSs are critically important** in making research collaborative and reproducible.
- They allow groups to work collaboratively on documents and track how they evolve over time. Ideally, all aspects of computational research would be hosted on publicly available version control repositories, such as **GitHub, Bitbucket or Google Code**.
- Unfortunately, the common approach is for researchers to **e-mail documents to each other with ad hoc naming conventions** that provide a poor man's version control.
- **While a small group can make it work, this approach most certainly does not scale beyond a few collaborators**, as painfully experienced by anyone who has participated in the madness of a flurry of e-mail attachments with oddly named files such as **paper-final-v2-REALLY-FINAL-john-OCT9.doc**.

c) Production-scale execution

- For production-scale execution, researchers typically turn away from the convenience of interactive computing environments to **compiled code (C,C++, Fortran) and libraries** for distributed and parallel processing.
- **These tools are specialized enough that their mastery requires a substantial investment of time.** We emphasize, that before production-scale computations begin, the researchers already have a working prototype in an interactive computing environment.
- **Therefore, turning to new parallel tools means starting over and maintaining at least two versions of the code moving forward.** Furthermore, data produced by the compiled version are often **imported back into the interactive environment** for visualization and analysis.
- The resulting back-and-forth work-flow is nearly impossible to capture and put into VCSs, making the computational research difficult to reproduce.
- Obviously the alternative, taken by many, is **simply to run the slow serial code** for as long as it takes. This is hardly a solution to the reproducibility problem, as run times in the weeks or months become in practice **single-shot** efforts that no one will replicate.

d & e) Publication & Education

- For publications and education, researchers use tools such as L A TEX, Google Docs, or Microsoft Word and PowerPoint.
- The most important attribute of these tools in this context is that, L A TEX excepted, they integrate poorly with VCSs and are ill-suited for work flow automation.
- Digital artefacts (code, data, and visualizations) are **often manually pasted** into these documents, which easily leads to a divergence between the computational outcomes and the publication.
- **The lack of automated integration** requires manual updating, something that is error-prone and easy to forget.

Lessons learned out of those issues

- The common approaches and tools used today introduce discontinuities between the different stages of the scientific work flow. Forcing researchers to switch tools at each stage, which in turn makes it difficult to move fluidly back and forth.
- A key element of the problem is the gap that exists between what we view as “final outcomes” of the scientific effort (papers and presentations that contain artefacts such as figures, tables, and other outcomes of the computation) and the pipeline that feeds these outcomes. Because most work flows involve a **manual transfer of information** (often with unrecorded changes along the way), the chances that these final outcomes match what the computational pipeline actually produces at any given time are low.
- **The problems listed earlier are both technical and social.** It is critical to understand that at the end of the day, only when researchers make a conscious decision to adopt improved work habits will we see substantial improvements on this problem. Obviously, higher-quality tools will make it easier and more appealing to adopt such changes; but other factors—from the inertia of ingrained habits to the pressure applied by the incentive models of modern research—are also at play.

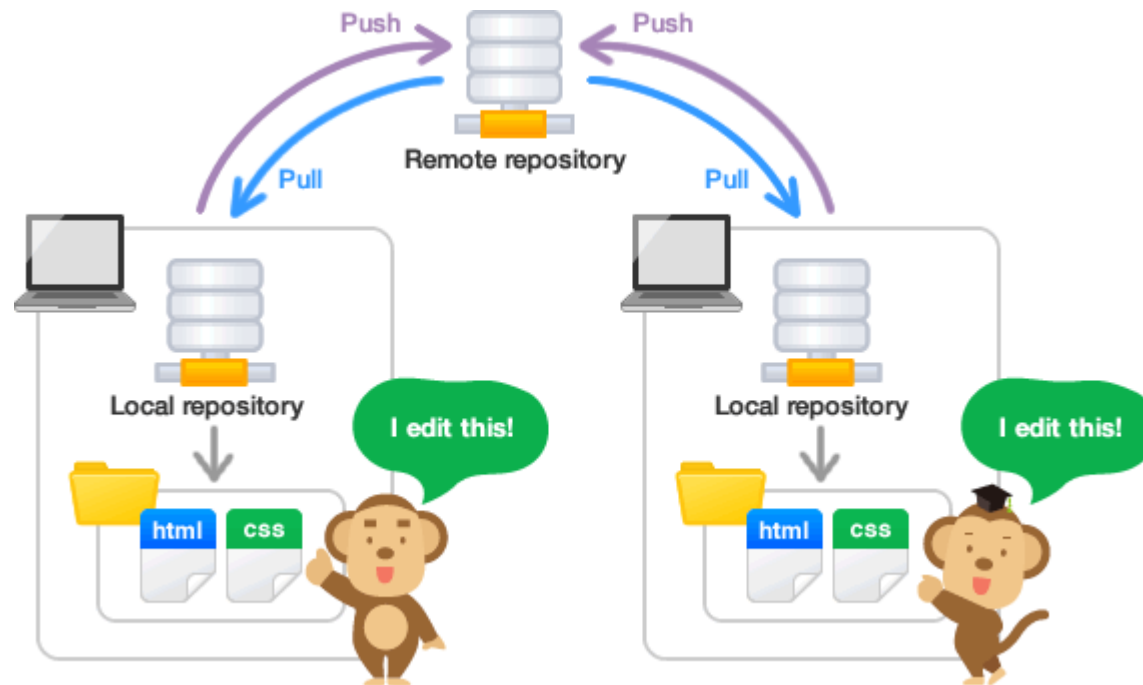
Simple possible remedies

- use open source tools (I don't claim to make all code open source).
- Version control with clear commit messages.
- Execution automation (use makefiles, scripts to plot,...).
- Testing (incl. unit tests).
- Readability (well commented framework – write README,...)

Git repositories

The official website: <https://git-scm.com/>

Free git book: <https://progit2.s3.amazonaws.com/en/2016-03-22-f3531/progit-en.1084.pdf>



How I use git :)



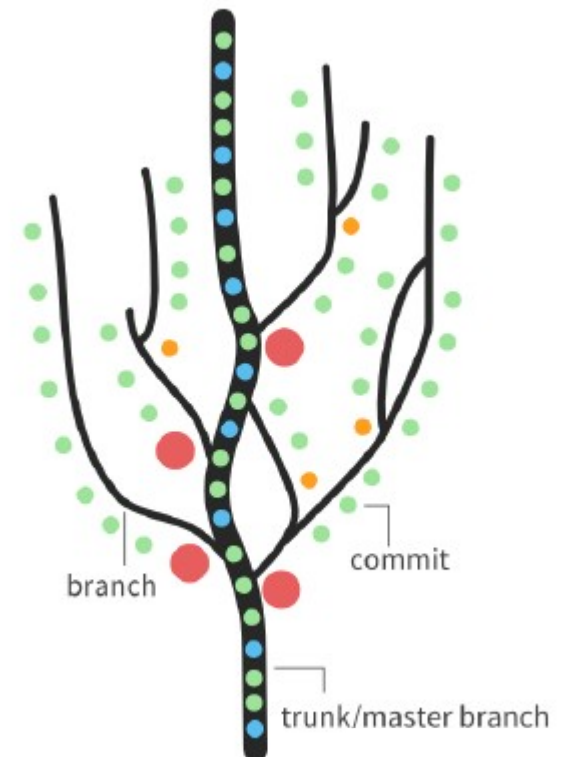
<https://xkcd.com/1597/>

Git overview

- As you develop software and make changes, add features, fix bugs, etc. it is often useful to have **a mechanism to keep track of changes and to ensure that your code base** and artefacts are well-protected by **being stored on a reliable server** (or multiple servers).
- This allows you **access to historic versions of your application's code** in case something breaks or to **"roll-back"** to a previous version if a critical bug is found.
- The solution is to use a **revision control system** that allows you to "check-in" changes to a code base.
- It **keeps track of all changes and allows you to "branch" a code base** into a separate copy so that you can develop features or enhancements in isolation of the **main code base** (often called the **"trunk"** in keeping with the tree metaphor).
- **Once a branch is completed** (and well-tested and reviewed), **it can then be merged back into the main trunk** and it becomes part of the project.

Version control system

- Git essentially **keeps track of all changes made to a project** and allows users to work in **large teams on very complex projects** while minimizing the conflicts between changes.
- These systems are not only used for organizational and backup purposes, but are absolutely essential when developing software as part of a team.
- **Each team member can have their own working copy of the project** code without interfering with other developer's copies or the main trunk.
- Only when separate branches have to be merged into the trunk do conflicting changes have to be addressed.
- Otherwise, such a system allows multiple developers to work on a very complex project in an organized manner.



Git – decentralized

- Git is a decentralized system.
- Multiple servers can act as repositories, but each copy on each developer's own machine is also a complete revision copy.
- Code commits are committed to the local repository.
- Merging a branch into another requires a push/pull request.
- Decentralizing the system means that anyone's machine can act as a code repository and can lead to wider collaboration and independence since different parties are no longer dependent on one master repository.
- Git itself is a version control system that can be installed on any server.

Getting access to a repository

Various commercial providers:

<https://github.com>

<https://bitbucket.org>

<https://gitlab.org>

Be careful: Some repositories are public (versus private)

→ various user choices (~10 USD/month → more functionality).

Collaboration on a project

- You can then **grant read/write access to your others** by making them collaborators on the project. You can easily do this in Github by following the instructions at this link:

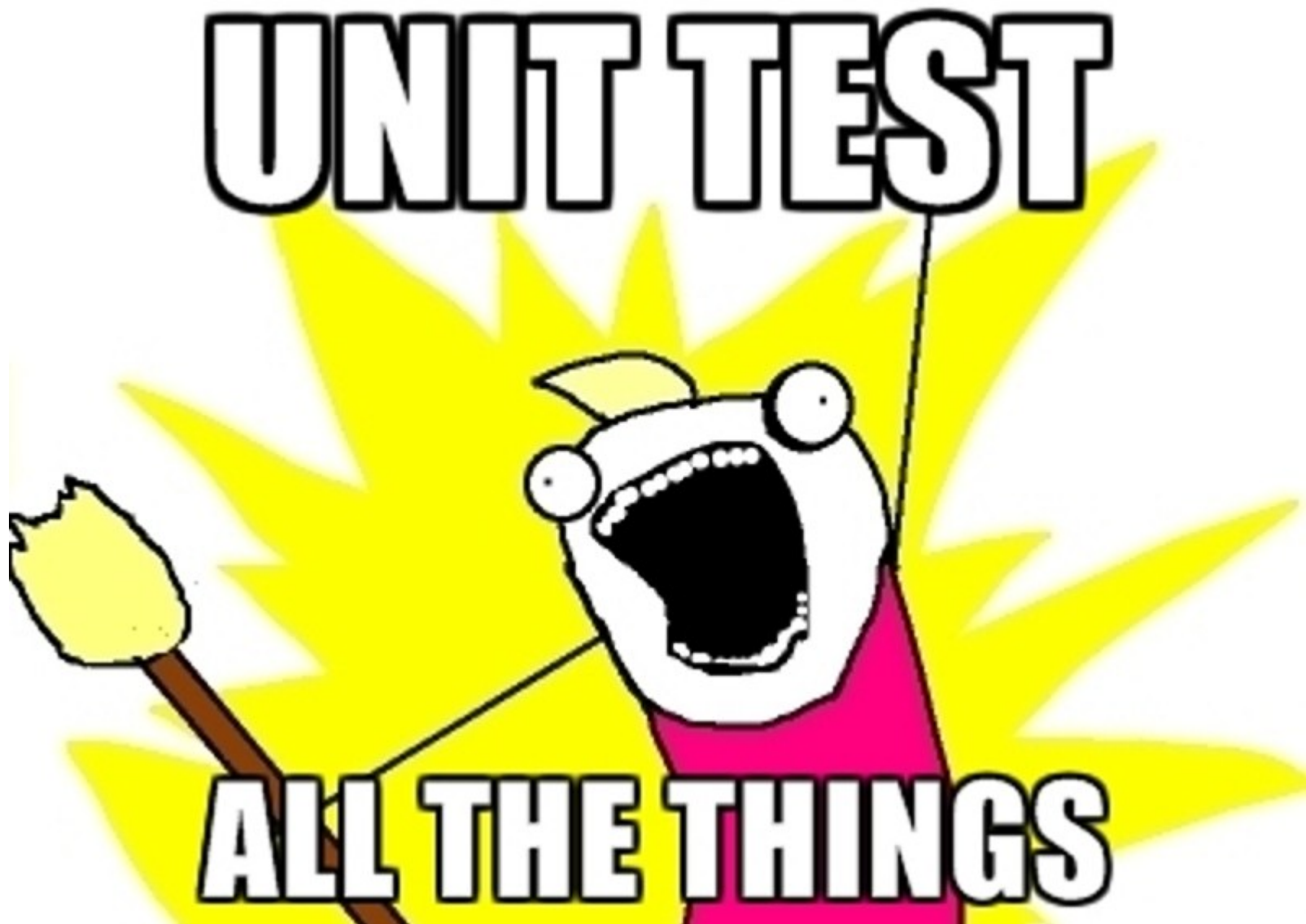
<https://help.github.com/articles/adding-collaborators-to-a-personal-repository/>

- Once you've all been added, everyone should be able to push/pull from the same repository.
- Git has many more options like branching etc. → check the man pages...
- Git Reference: <http://gitref.org/>
- Git Glossary: <https://help.github.com/articles/github-glossary/>
- GitHub's walkthrough for both Windows and Mac:

<https://help.github.com/articles/set-up-git/>

<https://services.github.com/kit/downloads/github-git-cheat-sheet.pdf>

Unit testing



What is unit testing

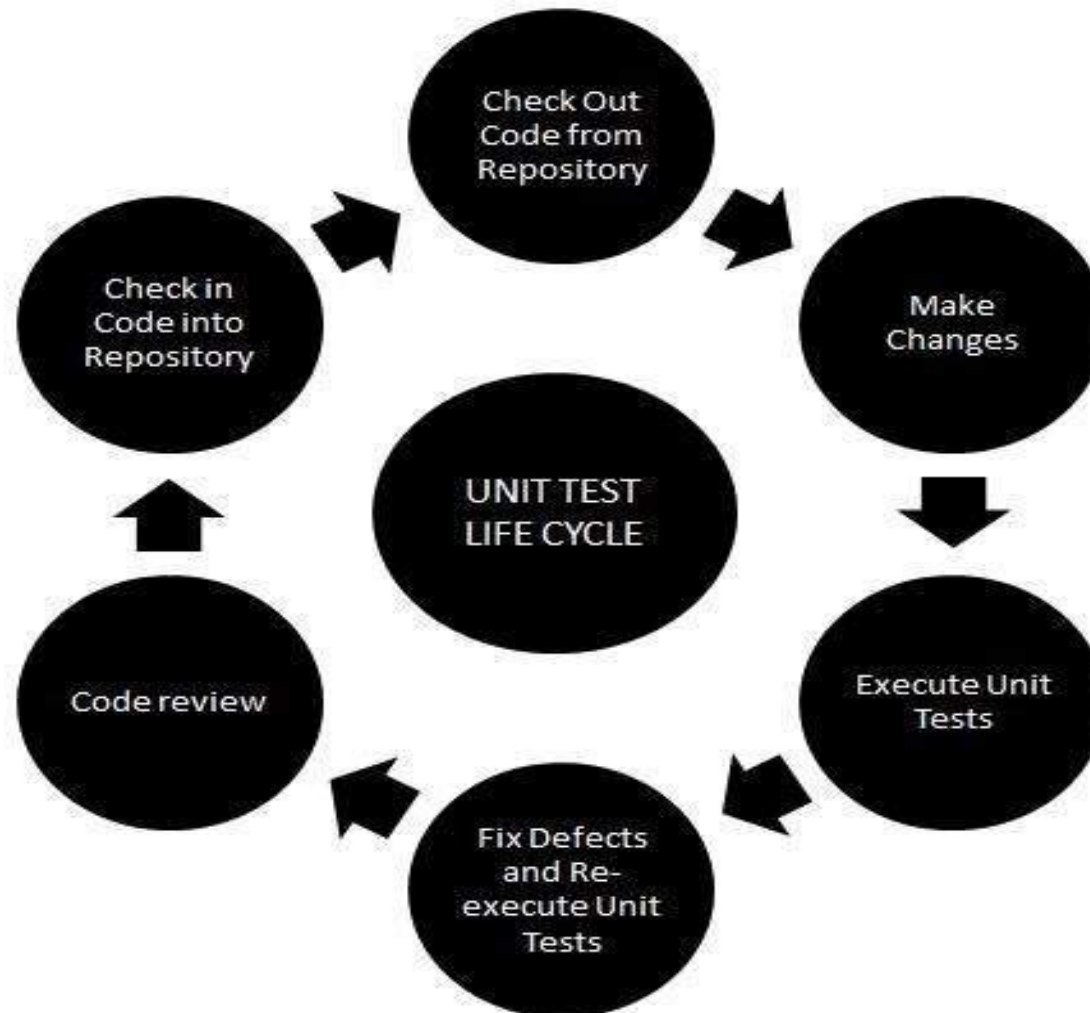
Unit testing:

- **a testing technique** using which individual modules are tested to determine if there are any issues by the developer himself.
- It is concerned with **functional correctness** of the standalone modules.
- The main aim is to **isolate each unit of the system** to **identify, analyse and fix the defects**.

Unit Testing - Advantages

- Reduces defects in the newly developed features or reduces bugs when changing the existing functionality.
- Reduces cost of testing as defects are captured in very early phase.
- Improves design and allows better re-factoring of code.
- Unit Tests, when integrated with build gives the quality of the build as well.

Unit testing life cycle



Google helps you

[**https://github.com/google/googletest**](https://github.com/google/googletest)



Questions?

1. Advice – RTFM

<https://en.wikipedia.org/wiki/RTFM>

2. Advice – <http://imgtfy.com/>

<http://imgtfy.com/?q=unit+testing>

<http://imgtfy.com/?q=git+repository>

