Development procedures and tools

Benedikt Hegner
(CERN)

Revision control

- Imagine hundreds of developers doing thousands of changes to the code base
  - How to track what was changed when and why?

- Imagine you have some code you are optimizing for performance
  - You recall yesterday your code was better
  - But the code is already gone...

- Revision control systems are there to help you:
- svn, git, hg, ...

- Please give it a few minutes and try:
- http://aymanh.com/subversion-a-quick-tutorial

Release Models

- The final goal of writing code should be a release
  - Correct
  - Self-consistent
  - Actually existing (not “in 5 years there will be...”)

- In the LHC experiments we have hundreds of people contributing semi-independently. How to sync their activities?

- Basically two models
  - Milestone based
  - Time based

- Of course hot fix releases here and there...

Milestone based releases

- A big chunk of functionality to be provided
- Not 100% sure when that will be completed
- Everything else relies on the changes

- Once all functionality is there, cut the release

- Works for prototypes or well-defined big migrations

- Doesn’t work for code in maintenance mode

Time based releases

- Define a time table of ‘release trains’
- Everything ready for a certain release deadline gets integrated
- Missed the deadline? Take the next release!
- Works well if there is a huge set of independent subsystems which all have their own schedule
- Requires more releases than a milestone based model

In reality

- Reality is a mixture of both concepts

- Full Releases are time based

- Individual subsystems set roadmaps for functionality
  - Once ready go into next available release

- Limiting factor is usually a wrong or too tight coupling of systems

Release integration process

- Don’t defer building the release up to the very last moment
  - Provide and **test** snapshot at least once a night (a.k.a. nightly or integration build)

- Add updates as soon as they are considered ready

- In theory, every integration build should be releasable
  - ... and a release only a snapshot at a certain day

- There is a special role to ensure that all this is happening - the release manager

Testing

Unit testing

- The code is not yet correct when the compiler accepts it..

- Developers usually test the latest functionality they implemented, but don’t check that old things aren’t broken

- Unit testing ensures you don’t break old things

- Make the testing as easy as possible
  - Otherwise you will always skip that testing step!

- Many good tools around there
  - You’ll try one later in an exercise

Unit testing

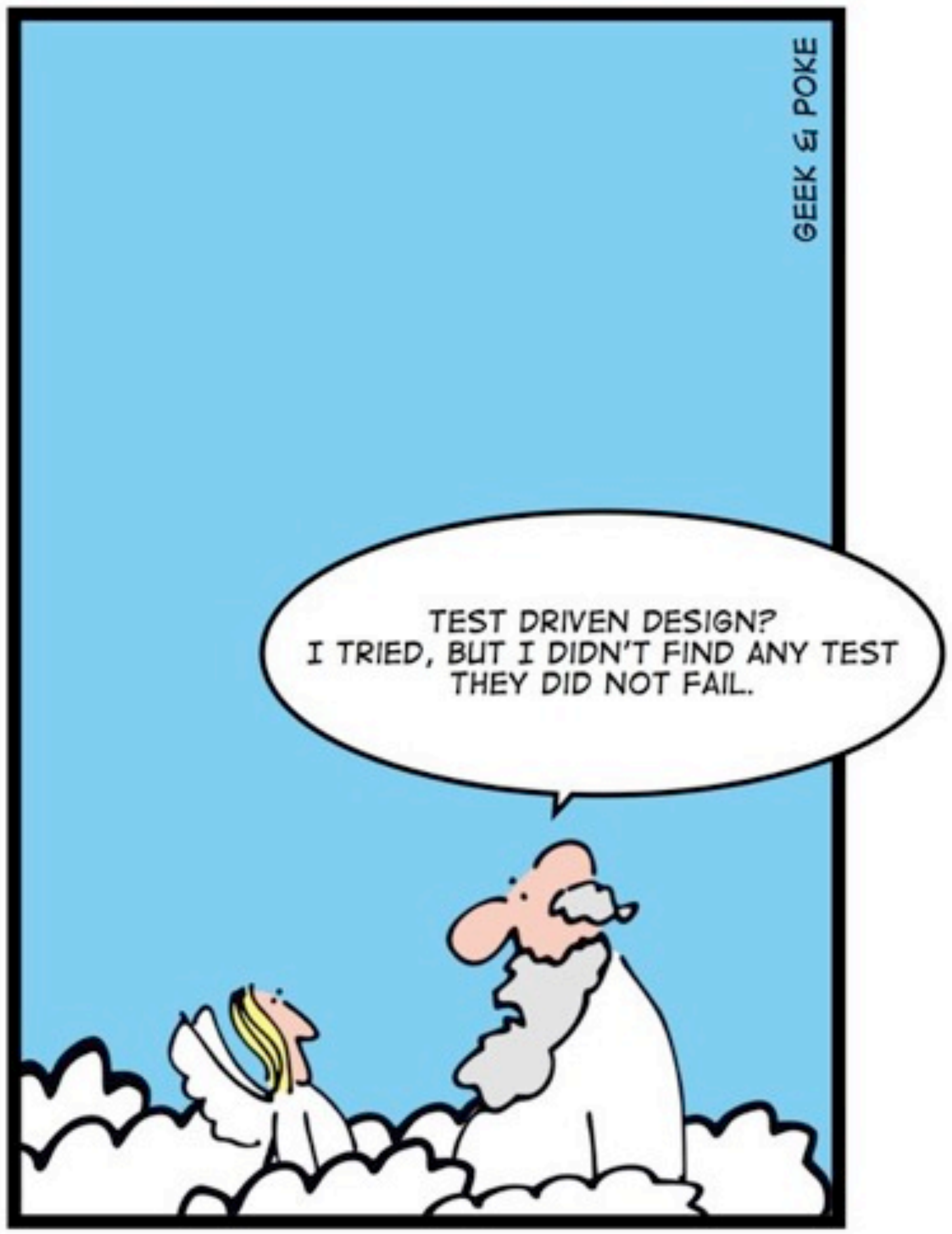
- You can put the testing to the very extreme

  Test Driven Design

- Encode the "contract" your code needs to fulfill and then develop against this

- Split writing tests and code among different people
  - You need a nasty test writer!

- Be sure that you separate interface tests from implementation specific tests
  - Important for refactoring!


GEEK & POKE
TEST DRIVEN DESIGN? I TRIED, BUT I DIDN'T FIND ANY TEST THEY DID NOT FAIL.

Testing at bigger scale

- Testing with unit tests is only a small piece
- Doing full workflows once in a while (i.e. daily) to check for consistency across components
- Release validation checking various use cases
  - at CMS offline releases have to survive a release validation production & procedure
- If you skip this procedure, Murphy's law will hit you right away

Static code analyzers

- Unit tests and others check code for proper runtime behaviour
- If code isn't getting executed during the tests it is not getting tested
  - coverage tools help you to spot such problems (http://lcgapp.cern.ch/spi/aaLibrarian/nightlies/x86_64-slc5-gcc43-cov/dev.Thu_CORAL-preview-x86_64-slc5-gcc43-cov/index.html)

- Static code analyzers help there
  - Don't care about how likely a certain execution branch is. Just tries every single one.

- Live demo:
  - https://coverity.cern.ch/projects/index.htm

Continuous integration tools

- If you want to really ensure that there is testing, automatize it

- Again, there are nice tools out there
  - The first you should look at is Jenkins

- Might e.g. be triggered by checkins to the repository

- The same setup can be used for the integration builds

Back to release planning... setting goals

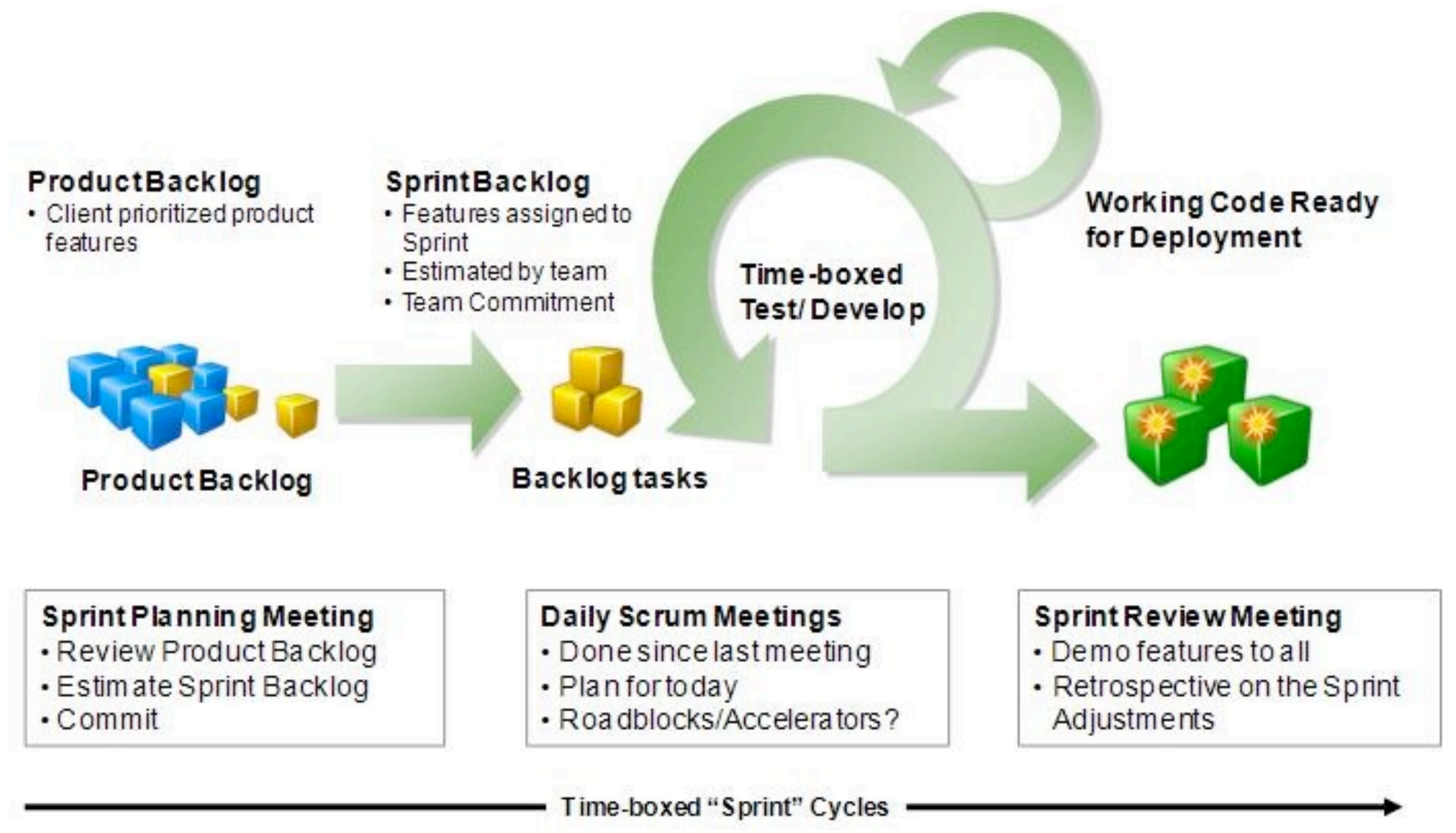
- One of the many mistakes done in development is playing around with features but never converging

- So how to define and keep track of goals:
  - on paper (private)
  - on whiteboard (office / small group)
  - central places

- There are tools at CERN to support you
  - Savannah now
  - Jira in the not too far future

- Live demo...
  - https://savannah.cern.ch/

An extreme - SCRUM



Documentation

- Of course we should to document the code properly
- There are many tools helping you there
  - Code cross-referencing
    - LXR, OpenGrok
  - Documentation creation tools
    - E.g. doxygen
- Live demo...
  - http://opengrok.web.cern.ch/opengrok

Exercise:

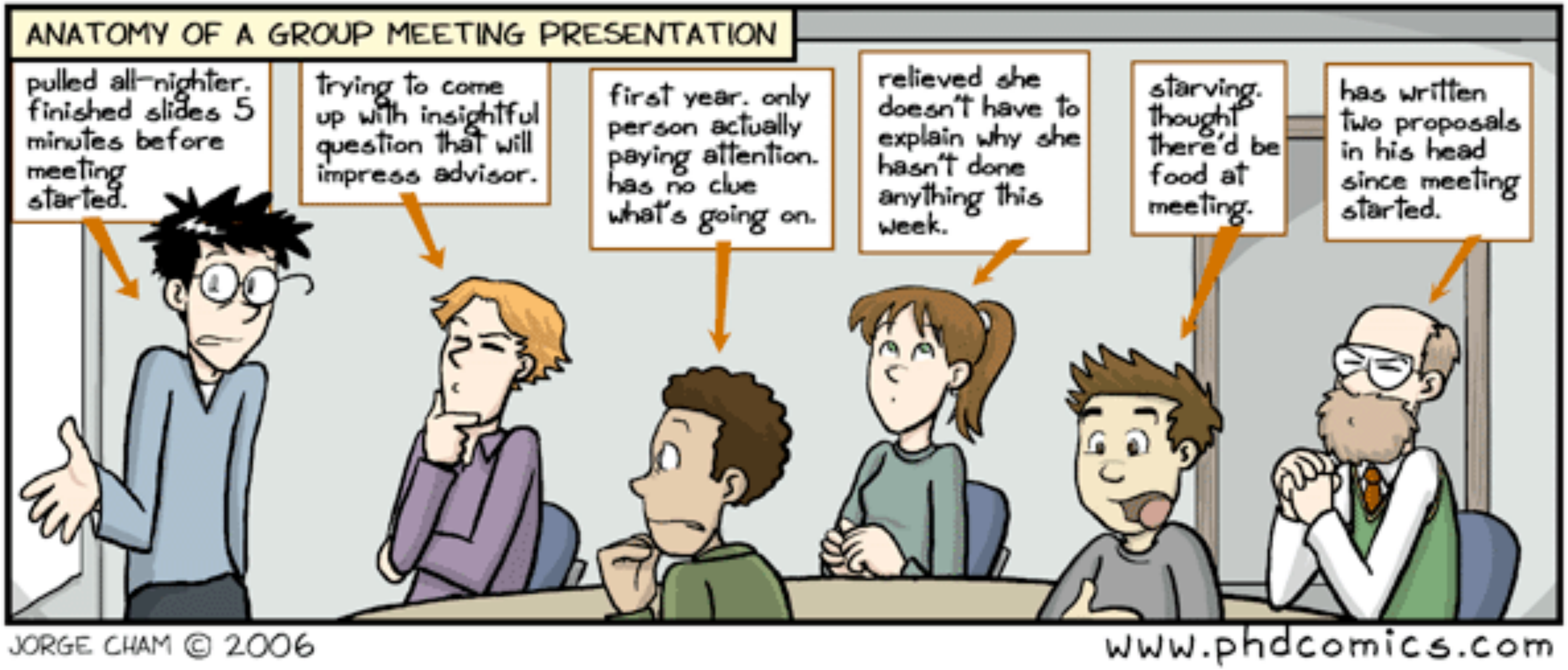
Start with the example code provided

Put it in revision control

Fix the problem

And use the revision control while doing that.

That’s it :-)


ANATOMY OF A GROUP MEETING PRESENTATION
pulled all-nighter. finished slides 5 minutes before meeting started.
trying to come up with insightful question that will impress advisor.
first year. only person actually paying attention. has no clue what's going on.
relieved she doesn't have to explain why she hasn't done anything this week.
starving. thought there'd be food at meeting.
has written two proposals in his head since meeting started.
JORGE CHAM © 2006
www.phdcomics.com