Software Practices

Rob Kooper,
Luigi Marini,
Jong Lee

National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign

Software Practices

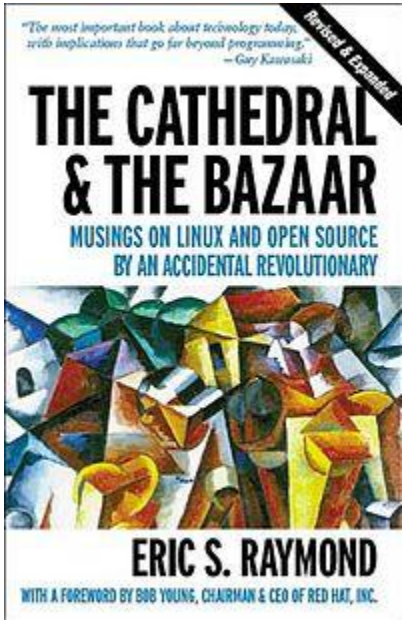
- “a set of informal rules that the software development community has learned over time to improve the quality of applications and simplify their maintenance”

  Wikipedia

- Goal today is to make these informal rules explicit

The Cathedral & The Bazaar

- The cathedral = source between releases only available to small group of software developers
- The baazar = source available to the public
- Top-Down vs Bottom-Up



19 Lessons

- Good programmers know what to write. Great ones know what to rewrite (and reuse)
- Release early. Release often.
- Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.

wikipedia

Software Development
is a…

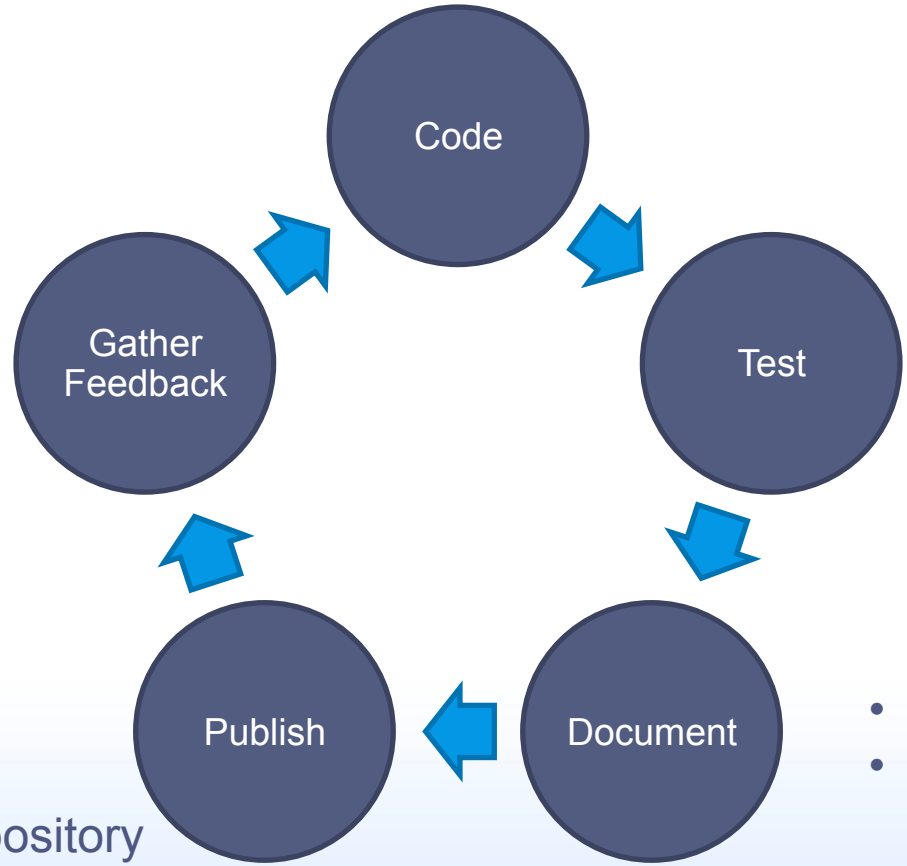
Collaborative Effort

Share Code

Many eyes looking at the same code

Iterative Development

- Mailing list
- Code reviews
- User



- Write new test
- Check old tests

- The code
- The manuals

- Code to shared repository
- Artifacts to central repository

SOFTWARE TOOLS

Software Tools

- IDE
  - Eclipse, Sublime, NodeJS, Emacs,vi, …
- Languages
  - JAVA, SCALA, PHP, …
- Source Code Management
  - GIT, SVN, …
- Build Tools
  - Maven, ANT, …

Eclipse

- Integrated Development Environment (IDE)
- Swiss Army Knife

- Good set of tutorials can be found at:
- http://www.vogella.com/eclipseide.html

Eclipse History

- Developed by IBM Canada
  - Java based
  - Replaced Smalltalk based IDE
- 2001 become open-source
- Eclipse 3.0 started to use OSGi
- Eclipse 3.2 and later have codenames and release trains
  - Current version is Kepler (4.3)

Eclipse Extensibility

- Eclipse Platform is plugin based
  - Really thanks to OSGi
- Eclipse is a small kernel with many plugins
- Eclipse plugins installed using P2
  - Dependency management
  - Upgrades
- Plugins can change all aspects of Eclipse

Eclipse Common Workflows

- Support for Code Editing
  - Opening/saving multiple files
  - Code completion
  - Function documentation
  - Code outline
- Support for Debugging
  - Variable inspection
  - (Conditional) breakpoints
- Support for Source Code Control
  - Build in support for GIT/CVS
  - Extra download for SVN

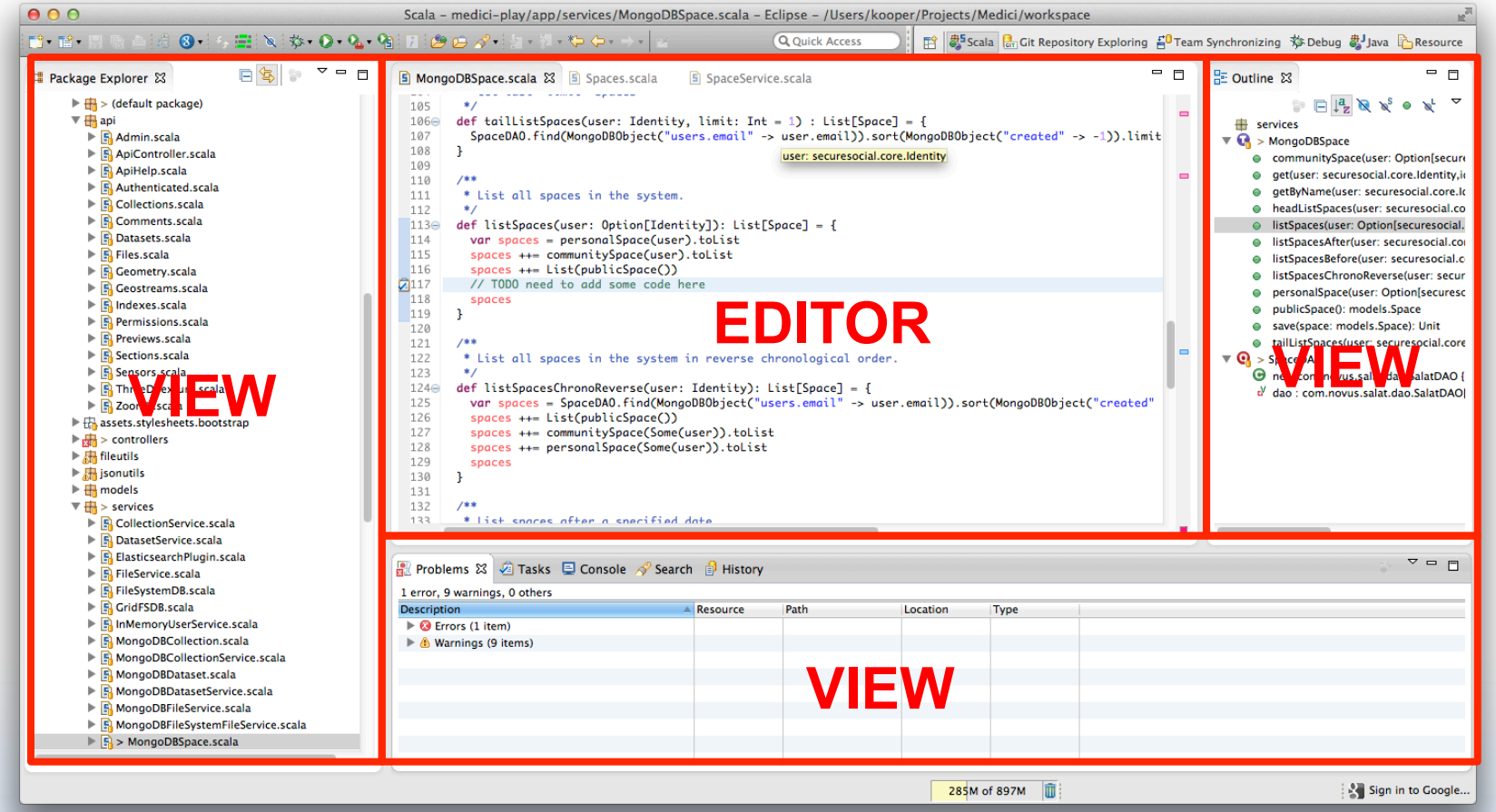
Eclipse Language Support

- Many languages supported
  - JDT is for java development
  - CDT is for C/C++ development
  - PDT is for PHP development
  - SCALA IDE is for SCALA development
  - StatET is for R development
  - PyDev is for Python development
  - RDT and RadRails is for Ruby and Ruby on Rails
  - Android Development toolkit
  - Google Development toolkit (GWT, App Engine)
  - …

Eclipse Definitions

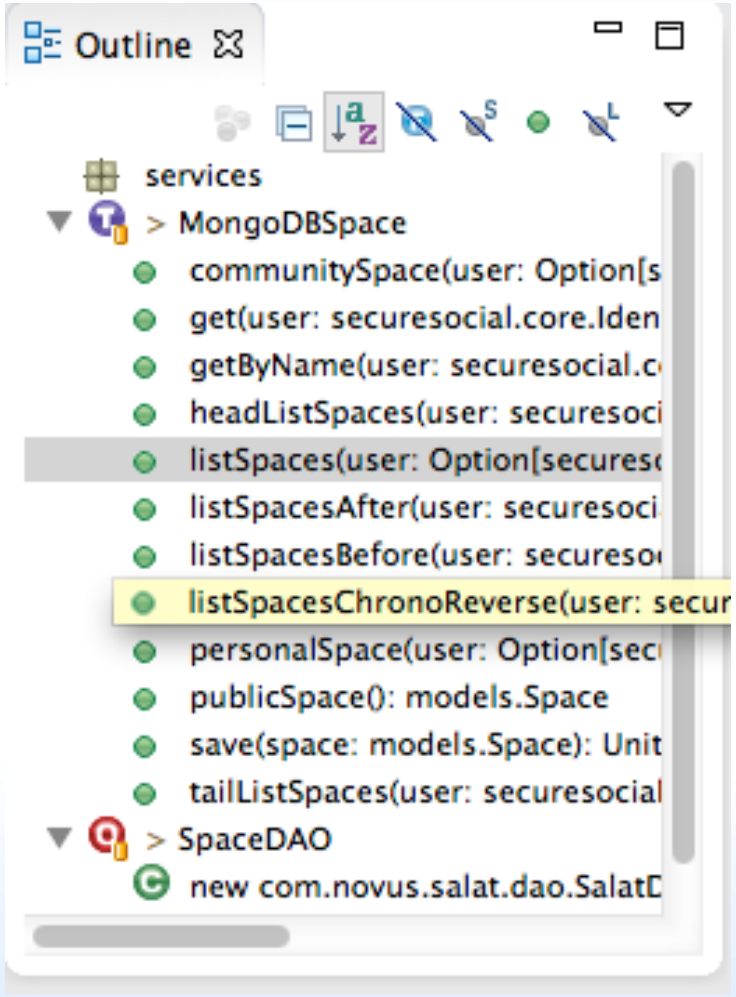
- **View**: used to work on data, show a hierarchical overview of data (Code Outline)
- **Editor**: used to modify the data shown to user (Java Editor)
- **Perspective**: grouping of views and editors to accomplish a task (Code editing, Debug)
- **Project**: contains source, binaries, etc. often a buildable and reusable unit (medici, polyglog, versus-core)
- **Workspace**: physical location on disk to store preferences, plugin meta-data, logs (versus, medici, browndog)

Eclipse Perspective



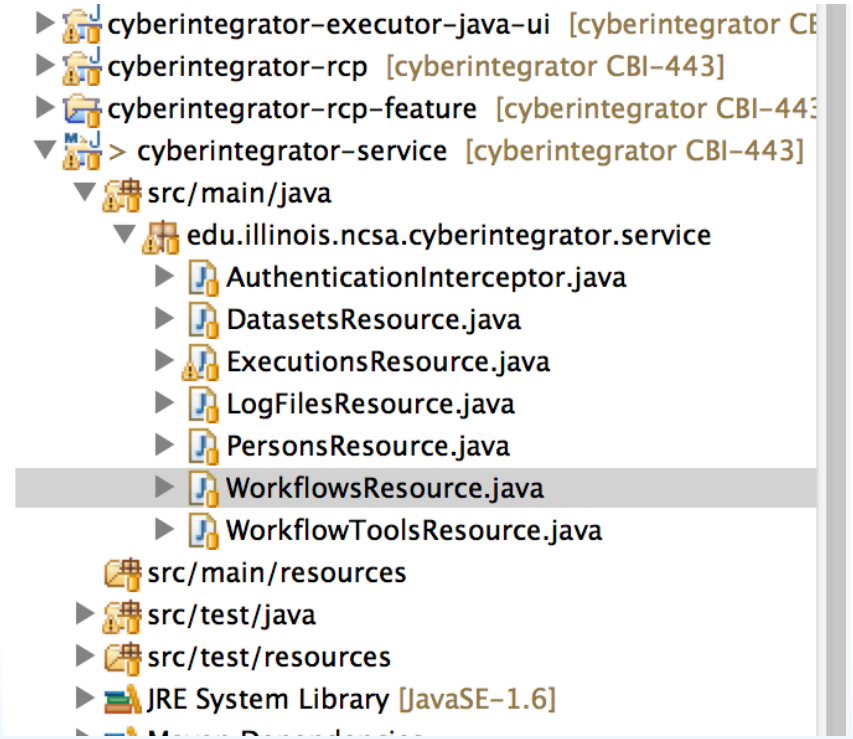
Eclipse Views

- Selections influence other views
  - Often editor
- Open files in project
  - Package Explorer
- Jump in source code
  - Outline
  - Tasks



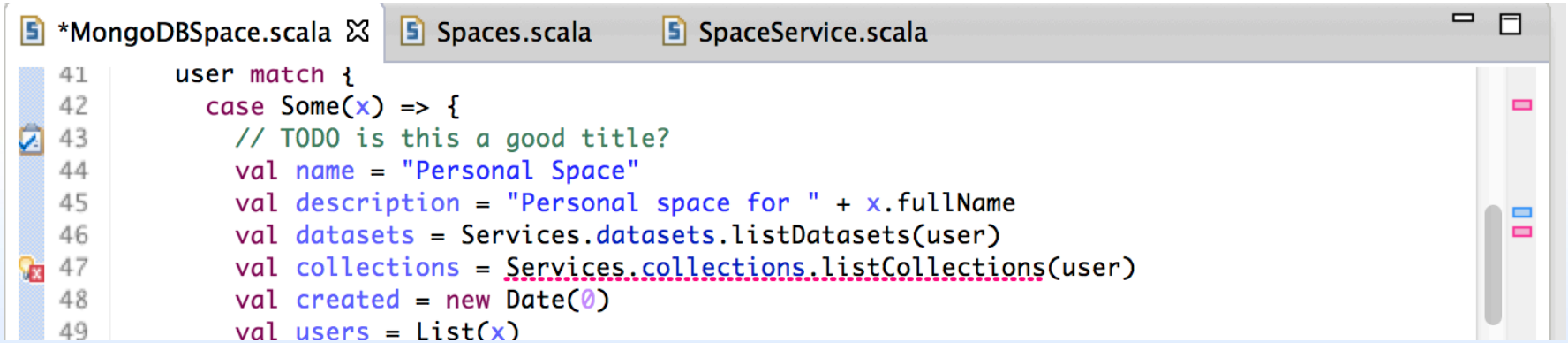
Eclipse Workspace and Projects

- Package Explorer shows workspace
  - Multiple projects (cyberintegrator-XYZ)
  - Contents in project
- Double click item to open (expand or in editor)



Eclipse Editor

- Editor can change files
  - * in front of name indicates not saved
  - Blue ticks in left and right indicate TODO items
  - Red ticks indicate errors in code
  - Right will show all marks in file, left only what is visible
  - Code completion (ctrl-space)



Languages

- Primarily JAVA code development
  - http://docs.oracle.com/javase/tutorial/java/
  - http://stackoverflow.com/questions/tagged/java
- Starting to develop using SCALA
  - http://docs.scala-lang.org/tutorials/
  - http://blog.tmorris.net/posts/scalaoption-cheat-sheet/

Source Code Management (SCM)

- SCM is NOT an option
- All our code should be in a repository from day 1
  - NO EXCUSES!
- SCM's have existed for decades
  - SCCS released in 1972
  - SCCS, RCS, CVS, SVN centralized systems
    - Can have a single server to checkin/checkout from
  - GIT/HG/BAZAAR distributed version control systems
    - developed around same time (2005)
    - Everybody has all code at all times
    - No single master
- We use GIT (and sometimes SVN)

GIT (Distributed)

- All repositories have all history
- Can work offline
- Small size
- Need to checkout everything
- Easy to branch
- Is the standard

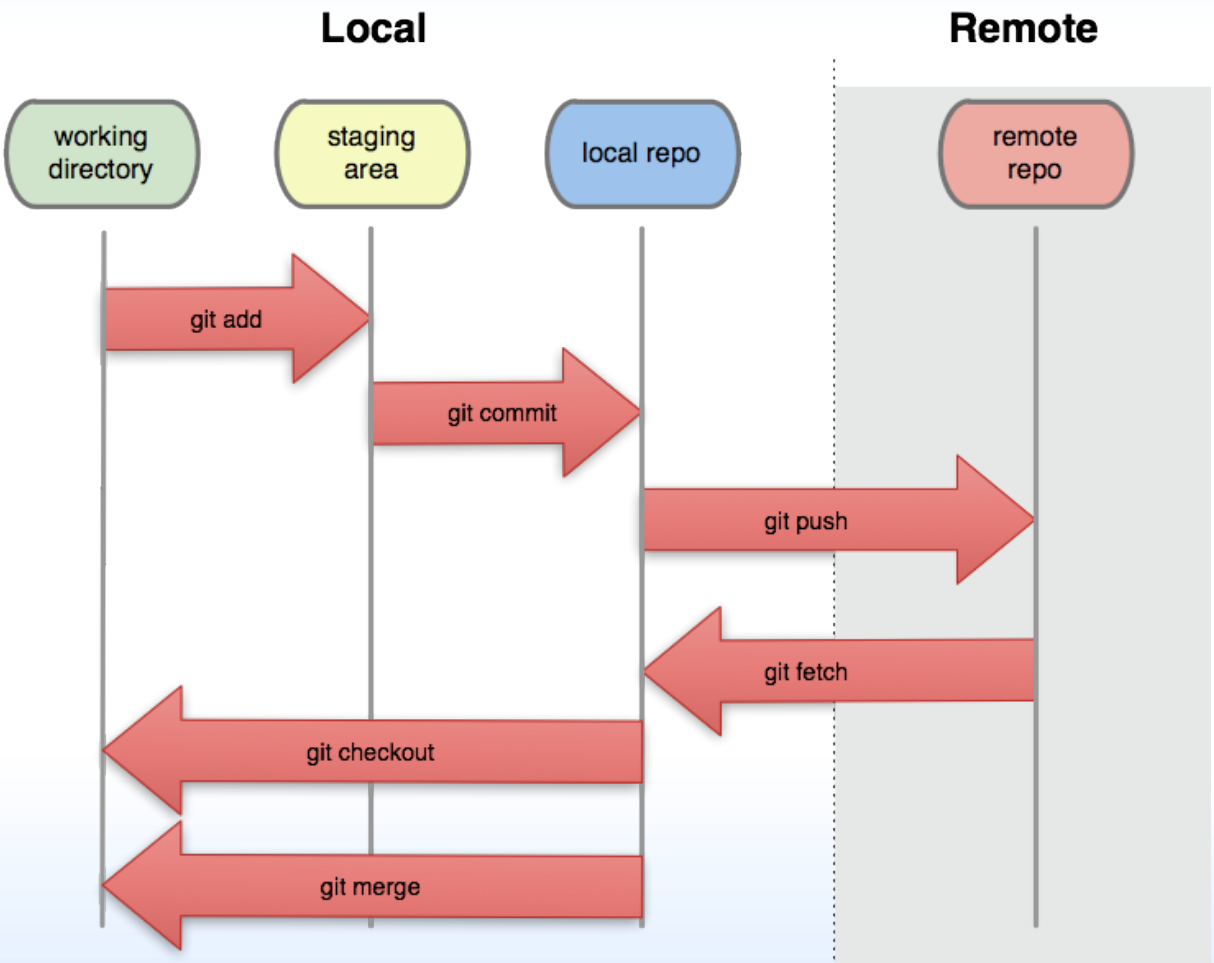
SVN (Centralized)

- Only centralized repository has all history
- Can only work online
- Large size
- Can checkout one item/ folder
- Branching is hard
- Was the standard

https://git.wiki.kernel.org/index.php/GitSvnComparison

http://thkoch2001.github.io/whygitisbetter/

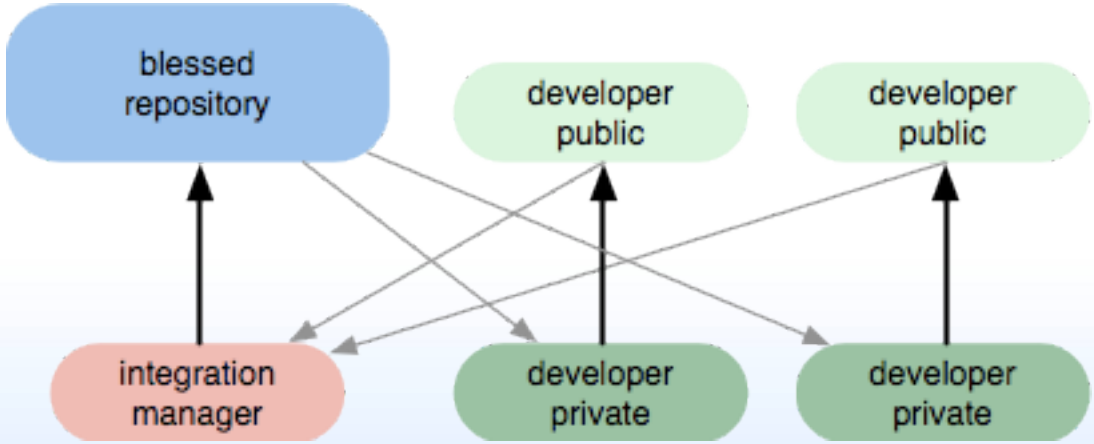
GIT local and remote



http://thkoch2001.github.io/whygitisbetter/

Proposed GIT Workflow

- Integration manager
  - Many developers cloning central/blessed repository
  - Many developers writing to their own repository
  - Many developers doing pull requests
  - One (or two) people that process pull requests
  - One (or two) people that write to blessed repository



http://thkoch2001.github.io/whygitisbetter/

Branching and Tagging

- When to branch?
  - New feature
- When to tag?
  - Always!
  - … when you have stable code
  - Lightweight tags are just a pointer to a commit

Getting and updating GIT repositories

- Clone a repository from remote to local
  - git clone <URL> [<reponame>]
- Fetch changes from remote to local
  - git fetch
- Merge changes from one branch to another
  - git merge [<branch name>]
    - master branch will merge with origin/master
    - Other branches will merge with what a specific branch
- Fetch and Merge changes
  - git pull

Branching GIT repositories 1/2

- Branches are cheap and quick, use them often!
- Branch from master (after an update)
- Show all branches
  - git branch
- Create a branch
  - git branch <branchname>
    - This will not checkout branch use git checkout
- Switch to another branch
  - git checkout <branchname>
- Create branch and check it out
  - git checkout –b <branchname>

Branching GIT repositories 2/2

- Rename a branch
  - git branch –m [<old name>] <new name>
    - If no <old name> give it will rename current branch
- Push a branch to remote
  - git push origin <branch name>
- Delete a branch
  - git branch --delete <branch name>
- Delete remote branch
  - git push --delete origin <branch name>

Other GIT commands

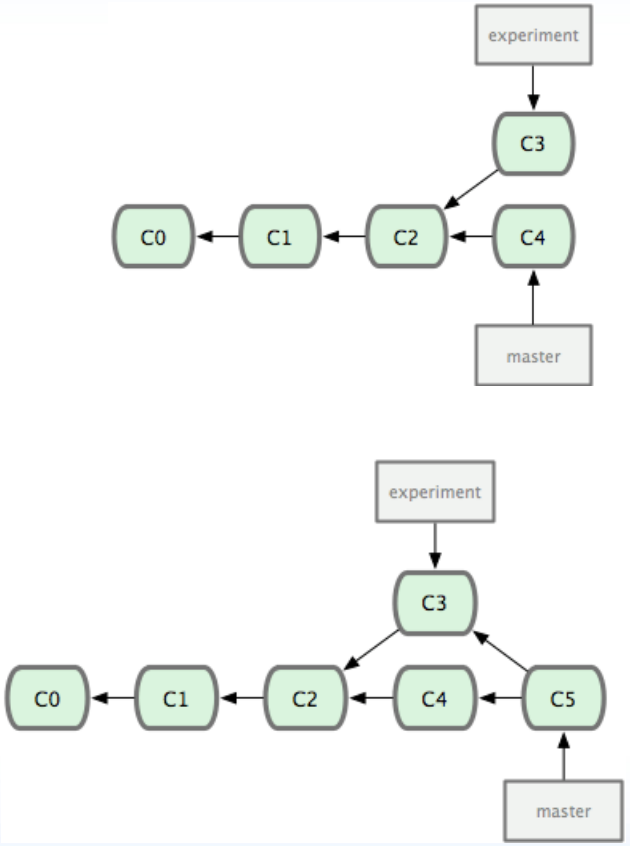
- Create a graph of all log messages
  - git log --graph --oneline –all
- Show, add and delete tags
  - git tag
  - git tag –a –m “message” <tagname>
  - git push origin <tagname>
  - git tag --delete <tagname>
  - git push --delete <tagname>

Eclipse and GIT

- Use GIT perspective to clone GIT repository
  - Right click repository to import projects into eclipse
- All operations under Team menu
  - Fetch, pull, merge, commit, add, push, …
- When creating branch can select remote branch to follow as well as what branch to clone from.
- History will show graph of commits and branches

GIT Merge vs. Rebase 1/2

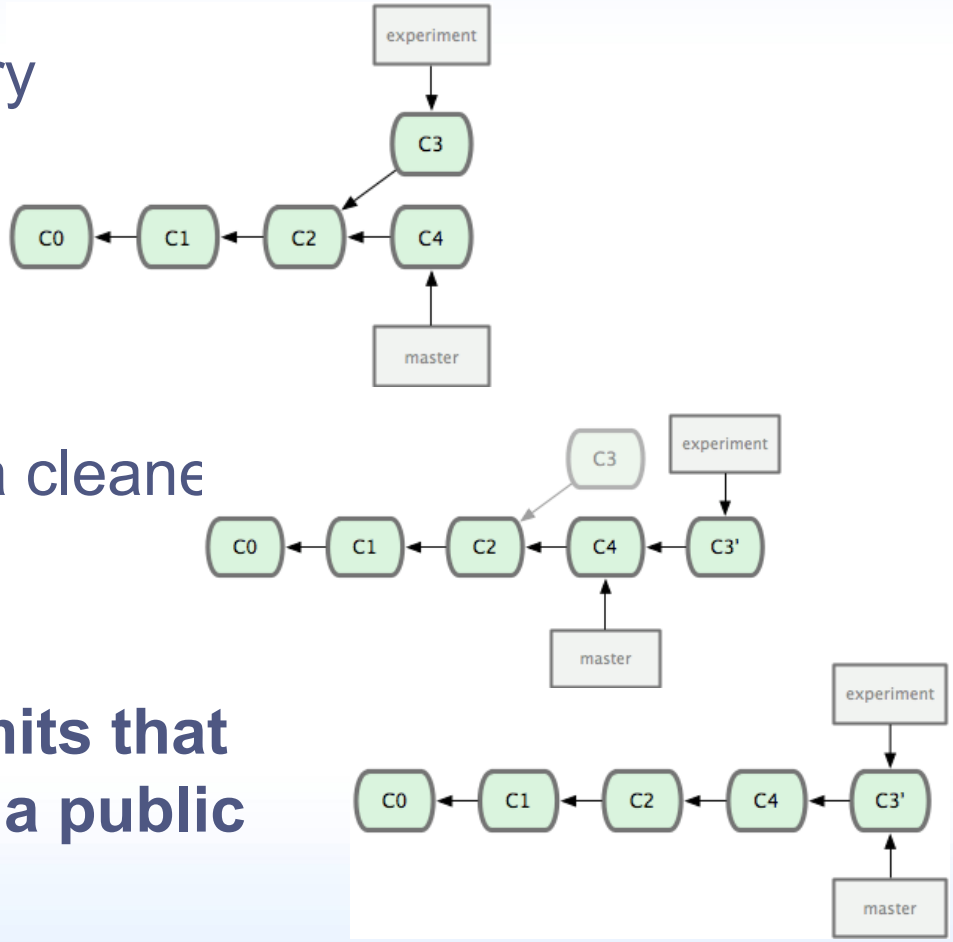
- Distributed SCM has to merge commits
  - I can have multiple commits before I push
  - Need to merge changes
- 2 options merge and rebase
  - Same result at the end
- Merge creates new commit
  - 3 way merge between branches and most recent ancestor
  - Create new commit with all changes



http://git-scm.com/book/en/Git-Branching-Rebasing

GIT Merge vs. Rebase 2/2

- Rebase replays history
  - Replay all patches
  - Check each patch
  - Resolve conflicts
  - Continue
- Rebasing makes for a cleane history.
- **Do not rebase commits that you have pushed to a public repository.**



http://git-scm.com/book/en/Git-Branching-Rebasing

Build Tools

- Build tools make it easier for others build
  - No more messy readme's with missing steps
- Build tools are needed for continuous integration
  - Automatic builds to test compilation of checkins

- C/C++ : Make and Makefiles
- JAVA : Maven and ANT
- SCALA : SBT

Maven

- Uses a pom.xml file to describe build and dependencies
  - Can specify specific version of dependencies
  - Can download all jar files needed
  - Build a single zip/jar/war file for distribution
  - Push build results to central server for others
  - Documentation of dependencies
- No need to include libraries
  - Significantly reduce size of repository

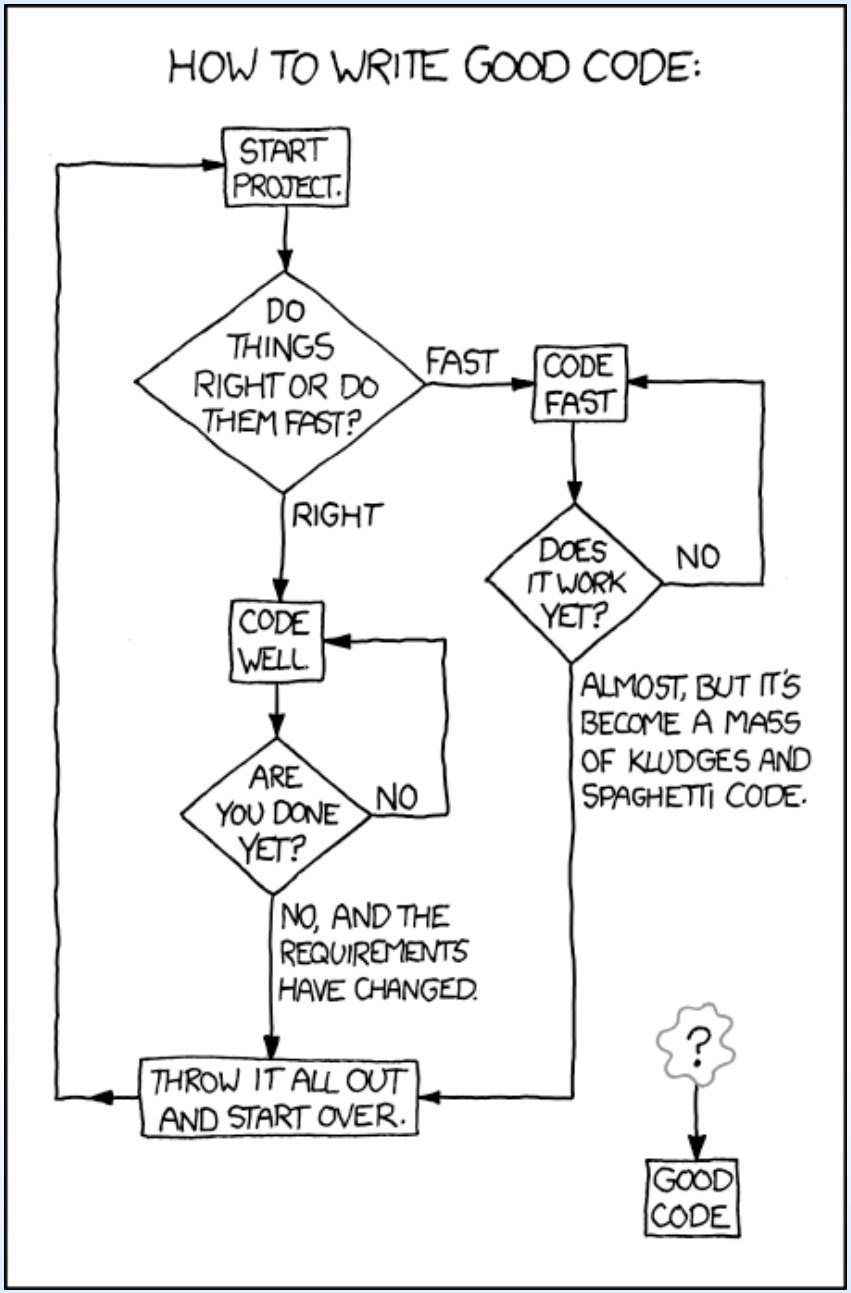
Example Maven, GIT and libraries

- medici-gwt-web
  - Source code for old medici
- 6322 files, 599 jar files

- GIT repository is 617,534,291 bytes

- jar = 365,920,330 bytes
- war = 96,157,342 bytes
- java = 31,262,928 bytes

WRITING GOOD CODE



http://xkcd.com/844/

Writing good code

- Documentation
  - Javadoc
  - Comments
  - Manuals
- Coding Style
  - Tabs/Spaces/braces/Line length
  - Logging, exceptions
- Testing
  - Unit/integration/regression/user
- Bug reporting/fixing

Documentation Minimum Requirements

- README
  - General information
  - Installation instructions
- LICENSE
  - NCSA

Why do we need comments

- Most projects groups are larger than 1 person
- Even if you are the only person, will you understand your code next month? Next year?

- Comments and documentation are a requirements for good code!

- Don’t wait till the end to comment/document, it will not happen!

Comments 1/3

- Javadoc, Roxygen, ScalaDoc, etc
  - http://docs.scala-lang.org/style/scaladoc.html
- Document functions
  - Short title
  - Longer description
  - List parameters
  - Return value
  - Author

- Eclipse will use function documentation to show what function does and what parameters it takes when it is used!

Comments 2/3

- Document algorithms, not basic code
- Try not to use the following comment:

  /* now comes magic, not sure what it does */
- Others will read your comments, including potentially your next employer.

GOOD

/* Following code will find all primes and multiply them */

BAD

/* add 1 to i */

Comments 3/3

- If you know something is not ready comment it
  - Use TODO (HACK, FIXME)
  - If you will fix it add your name with the TODO

- Example

// TODO RK : not implemented needs to be done

- Eclipse will highlight this for you
  - Also will show it in TASK view

Documentation

- Function documentation is not the end
- Document classes
  - What does the class do?
  - Example code
- Document packages/modules
  - What is the purpose of this package?
  - Why do these files belong together?
- Documentation
  - What does the software do?
  - How do I start it?
  - How does a user use it?

Manuals

- User manuals are boring to write, but we need them!
- Describe what software does
  - What use cases does it solve?
  - How can the user work with the software?
  - Any known defects?

Needs Doc desperately.

Very difficult to implement

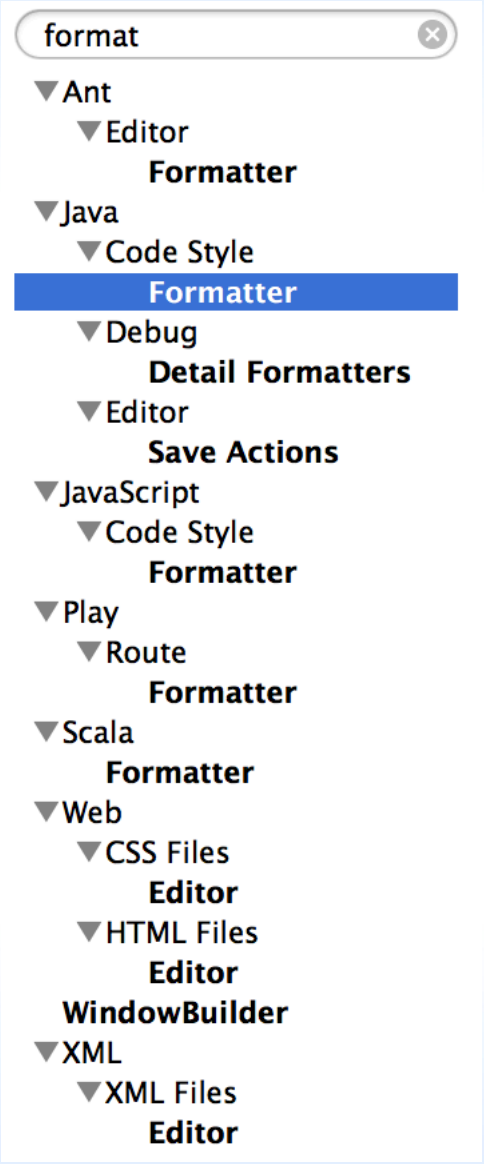
So, I have decided to focus on concrete support requests and have just embedded basic docu

https://marketplace.atlassian.com/plugins/de.polscheit.jira.plugins.gantt

Code Styles

- Python enforces good indentation
- Come up with a style for the project
  - Where do the braces go?
  - Always use braces?
  - Use spaces or tabs?
  - How many spaces per indentation level?
- Use tools for code formatting
  - Use code formatter in eclipse
  - Project lead should setup a style to use
  - Save style with project as .settings

- **BE CONSISTENT!**



Logging

- Don’t use System.out/err.println()
  - Maybe for quick debugging, but remove it afterwards
- Use logging facade packages
  - commons-logging
  - SLF4J
- Allow for modification of logger used
  - Log4j, JDK logger, something else
- Loggers allow to change loglevel
  - TRACE,DEBUG, INFO, WARN, ERROR

Exceptions

- When catching an exception log it, even it you rethrow it!

- Don’t catch an exception and do nothing with it!

```
try {
   // do something
} catch(Exception e) { }
```

- When logging add exception

```
} catch(Exception e) {
    log.error(“Could not open file.”, e);
}
```

Testing

- Software testing can be stated as the process of validating and verifying that a computer program/ application/product:
  - meets the requirements that guided its design and development,
  - works as expected,
  - can be implemented with the same characteristics,
  - and satisfies the needs of stakeholders.

wikipedia

When to test

- Now!
- Use cases will help to define test cases
  - Use cases will also help you sell your software
- Think of test cases and write them down.
  - Documentation!
- Create automatic tests
  - Allow you to do many tests continuously
- Facilitates refactoring
- Don’t think of test cases that pass, think of test cases that will break your code!

Types of Testing

- Unit tests
  - Test small units of code, functions
  - Short tests
- Integration tests
  - Test combination of smaller units
  - Tests complete system
- Regression tests
  - Test for reoccurrence of old bugs
- User testing
  - Does the software what the user wants/expects?
  - Don’t tell the user what to do, give them a task.

Bug reporting

- Include as much detail as possible
  - What version of the software
  - What browser and/or OS?
  - What version of Java?
  - Do you have a stacktrace?
  - Do you have an example dataset?
- Include detailed steps (if possible)
  1. I opened application
  2. I clicked on login link
  3. I typed in username with a space and password
  4. I clicked on login and got 500 error

Bug fixing

- Create a test case first
  - A simple test case that shows the bug
  - Check in the test to all branches
  - Test is used for regression testing
- Fix the bug
- Commit fix to all branches where bug exists
- Notify user of fix to bug and say “Thank you”

SOFTWARE DEVELOPMENT

Agile Development

- Lightweight development (not software)
  - Been around for long time (1957)
- Lightweight agile software development
  - Evolved mid 1990s
  - Counter to waterfall methods

The Agile Manifesto

- We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
  - **Individuals** and interactions over processes and tools
  - **Working** software over comprehensive documentation
  - **Customer** collaboration over contract negotiation
  - **Responding** to change over following a plan
- That is, while there is value in the items on the right, we value the items on the left more.

Wikipedia

Scrum

- A key principle of Scrum is its recognition that during a project the customers can change their minds about what they want and need.

Wikipedia

- Things will change in software projects!
- How do/will we react to this?

Scrum (definitions)

- Product Owner
  - The person responsible for representing the interests of the stakeholders.
- Scrum Master
  - The person responsible for the Scrum process.
- Development Team
  - A cross-functional group of people responsible for delivering potentially shippable increments of Product.
- Sprint
  - A time period (typically 1–4 weeks) in which development occurs on a set items that the team has committed to.

Scrum Meetings

- Standup meeting (15 minutes)
  - What have you done since <last meeting>?
  - What are you planning to do <this period>?
  - Any impediments/stumbling blocks?
  - **NO DISCUSIONS OF PROBLEMS!**
- Sprint planning meeting (2 part meeting, 2x4 hours?)
  - What tasks will we work on this sprint?
  - How shall we accomplish these tasks?
- End of cycle (2 meetings, 4 hours each)
  - After sprint is over.
  - Sprint review meeting (what was done)
  - Sprint retrospective (what worked, what can be improved)

NCSA OPENSOURCE

Available Software Resources

- Full Atlassian Stack
  - CONFLUENCE, JIRA, STASH, BAMBOO, FISHEYE, CROWD
- Sonatype Nexus repository
  - Maven artifact repository
- Jenkins
  - Another build system

- All available on https://opensource.ncsa.illinois.edu/
  - Intel I7 processor, 2.8Ghz
  - 16GB memory
  - 1TB of storage (700GB free)
  - Continuous backup using crashplan

Opensource Software 1/2

- CONFLUENCE
  - https://opensource.ncsa.illinois.edu/confluence
  - Wiki
- JIRA
  - https://opensource.ncsa.illinois.edu/jira
  - Bug tracking software
- STASH
  - https://opensource.ncsa.illinois.edu/stash
  - Source code management
- BAMBOO
  - https://opensource.ncsa.illinois.edu/bamboo
  - Continuous build software

Opensource Software 2/2

- CROWD
  - https://opensource.ncsa.illinois.edu/crowd
  - Account management
- Jenkins
  - https://opensource.ncsa.illinois.edu/jenkins
  - Continuous build software (migrating to BAMBOO)
- Nexus
  - https://opensource.ncsa.illinois.edu/nexus
  - Maven artifact repository

Documentation

- Request to have webspace per project
  - https://opensource.ncsa.illinois.edu/project/XYZ
  - Javadoc?
  - Web pages (checked out from stash)

Questions?

- This is a living workflow!
- If it does not work for you, or you know a better way please let us know.

- Rob Kooper (kooper@illinois.edu)
- Luigi Marini (lmarini@illinois.edu)
- Jong Lee (jonglee1@illinois.edu)
- Kenton McHenry (mchenry@illinois.edu)

ADDITIONAL SLIDES

Additional Slides

- Following slides were not discussed during presentation.
  - Workflow with GIT commands
  - Stash review process
  - Demo GIT Site

Proposed Workflow (Command Line)

1) Checkout master

- `git checkout master`
- `git pull`
- `git status`

2) Create a branch

- use JIRA issue as branch name such as MMDB-1234
- `git checkout -b <branch name>`

Proposed Workflow (Command Line)

3a) commit your changes to your local repository

- `git commit`

3b) push your changes

- only need to specify remote if first time pushing
- `git push [origin <branch name>]`

3c) update your git repository from remote

- `git fetch`

3d) update you branch with respect to the remote master

- `git rebase origin/master`

Proposed Workflow (Command Line)

4a) update your branch from remote by rebase

- `git fetch`
- `git rebase origin/master`

4b) push to the branch on remote

- Use –f if you rebased and already pushed
- `git push origin <branch name>`

4c) goto stash website and issue a pull request

Proposed Workflow (Command Line)

5a) once branch is merged (make sure it is!)

- `git checkout master`

5b) delete branch on local

- `git branch --delete <branchname>`

5c) delete branch on remote

- `git push --delete origin <branchname>`

Proposed Workflow (Eclipse)

1) Checkout master
   - Team -> Switch To -> master
   - Team -> Pull

2) Create a branch
   - Team -> Switch -> New Branch …
   - Source ref: master (either origin or not)
   - Pull strategy: rebase

Proposed Workflow (Eclipse)

3a) Commit your changes to your local repository

- Team -> commit

3b) push your changes

- Team -> Push to upstream

3c) update your git repository from remote

- Team -> Fetch from upstream

3d) update you branch with respect to the remote master

- Team -> rebase and rebase with origin/master

Proposed Workflow (Eclipse)

4a) update your master from remote by rebase

- Team -> rebase and rebase with origin/master

4b) push to the branch on remote

- (Not clear how to force push from eclipse)
- Team -> push to upstream

4c) goto stash website and issue a pull request

Proposed Workflow (Eclipse)

5a) once branch is merged (make sure it is!)

- Team -> Switch To -> master

5b) delete branch on local

- Team -> Advanced -> Delete Branch

5c) delete branch on remote

- Goto stash and remove branch

Stash Review Process

- Stash review allows a second set of eyes
  - After you finish code for issue
  - Request a pull request to the master
    - Goto your branch in stash
    - Click on Pull Request button
  - Add reviewers that know the code and problem
  - Add reviewer that can push to master
    - Medici : Luigi and Rob
    - Cyberintegrator : Chris and Rob
    - Polyglot : Kenton
    - Versus : Luigi and Smruti

Demo GIT Site

- Please checkout from the following site:
  - https://opensource.ncsa.illinois.edu/stash/users/kooper/repos/demo/browse
- Do the following:
  - Look at the code
  - Run unit test
    - Branch, fix, commit and push
  - Look at maven pom.xml
  - Look at pull request (on stash)
    - Comment, approve