Working in Teams: Version Control

September 8, 2022

Byung-Gon Chun

(Slide credits: George Candea, EPFL and Armando Fox, UCB)

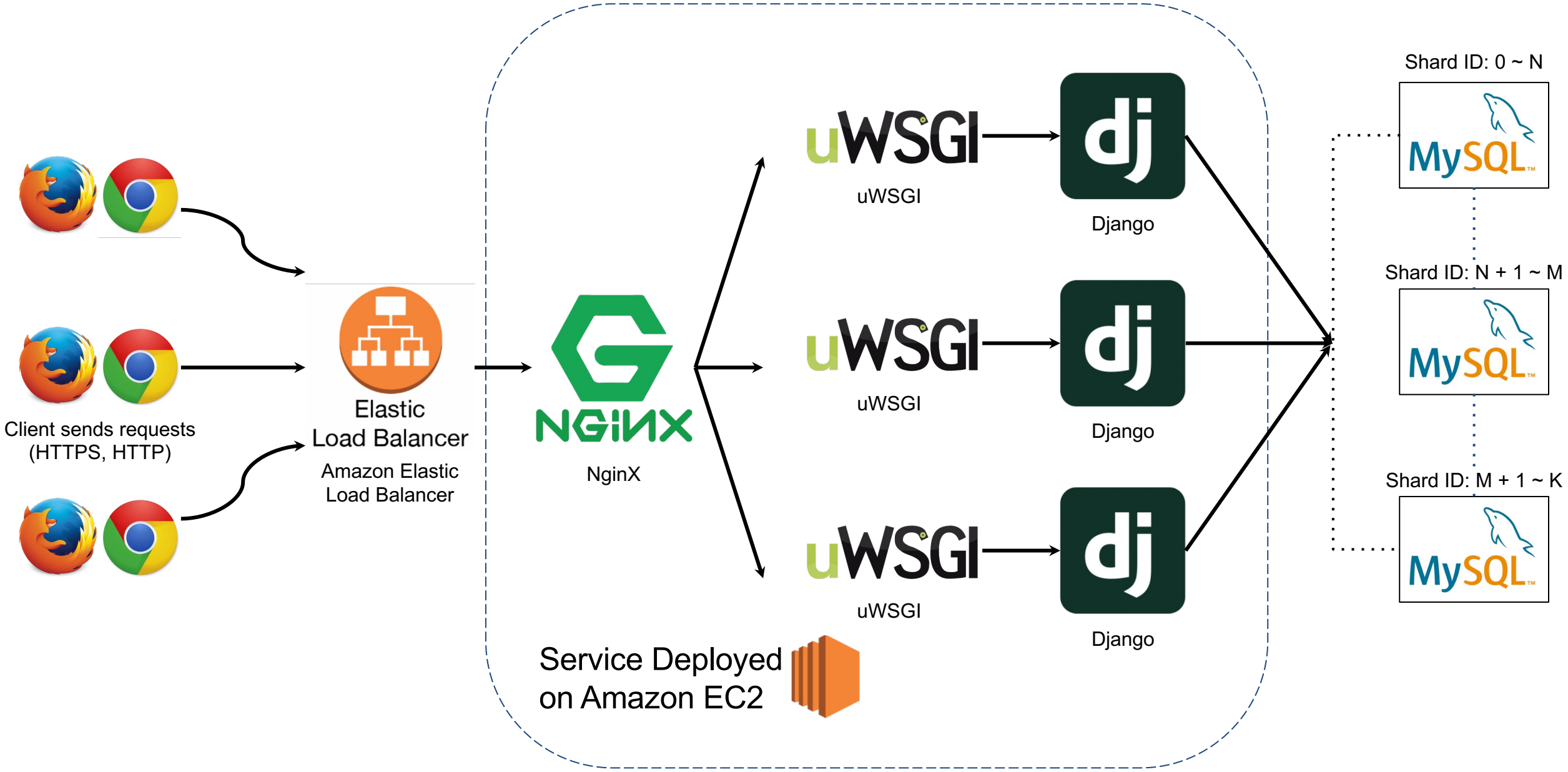
Deployment

Development

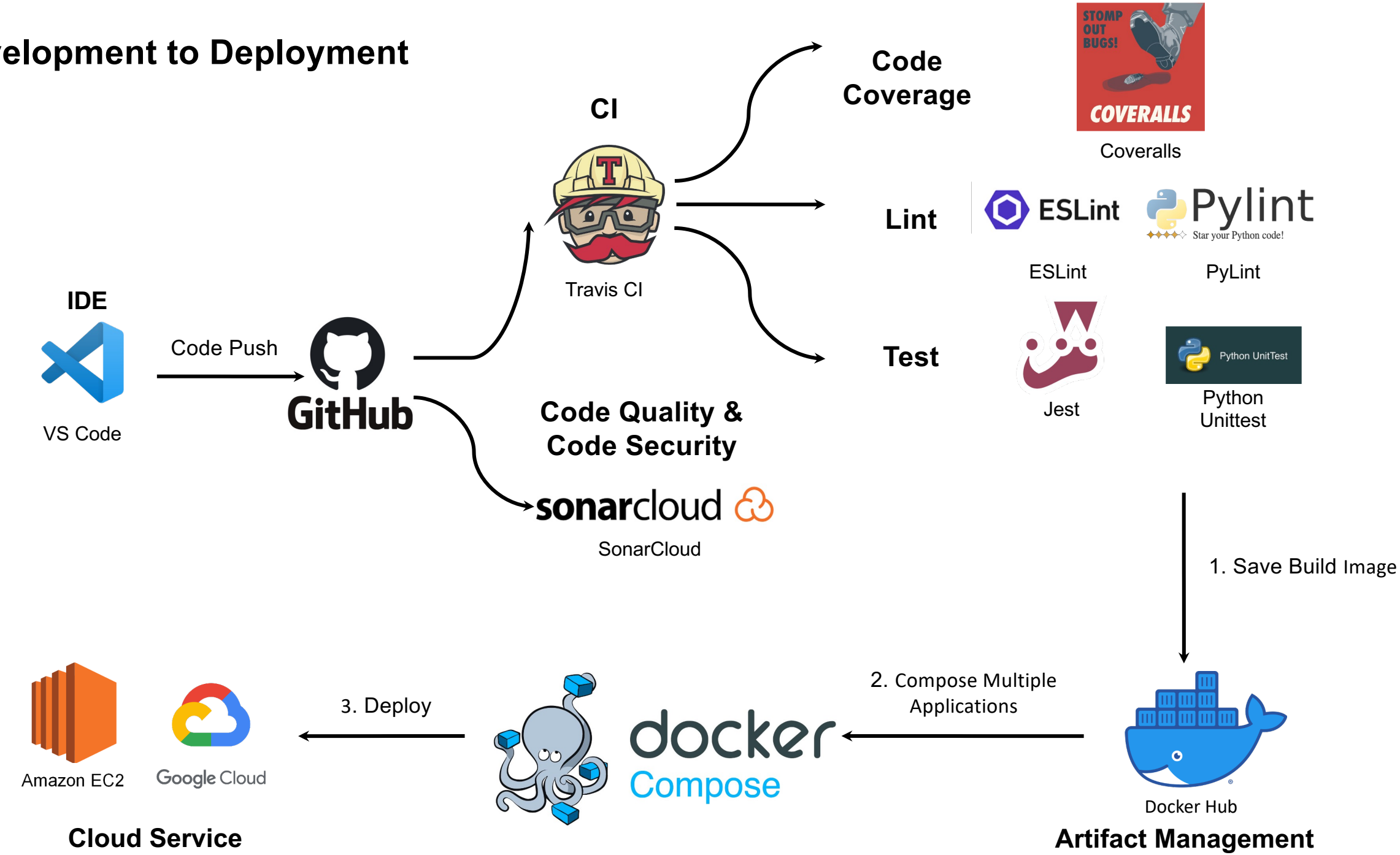
	Framework	Language	Testing	Lint Tool
Frontend	React, Redux	TypeScript	Jest	ESLint
Backend	Django	Python	Python Unittest	PyLint

IDE	Environment	Static Analysis	Continuous Integration	Code Coverage
VS Code	Linux, Docker	SonarCloud	Travis CI	Coveralls

Deployment with multiple Backend instances



Development to Deployment



Work in Teams

- Develop code, docs, config files, etc. over time
- Multiple developers touch upon code, docs, config files, etc. concurrently
- How to make sure files are “consistent”?

Versions

How to make sure updated files are “consistent”?

A Naïve Approach

- Before editing a file, lock the file
- The file cannot be updated by any other members concurrently.
- After done with editing, unlock the file

- This avoids conflicts, but developers **cannot** work on the same file at the same time

Version Control

- What is it?
  - *Version* (snapshot) code, docs, config files, etc. at key points in time
  - Complete copy of every versioned file per snapshot
  - Implementation: deltas? complete file copy?

- Why do it?
  - Roll back if introduce bugs
  - Separate deployed from development version of code
  - Keep separate *branches* of development
  - Documented history of who did what when
  - Track what changed between revisions of a project

40 Years of Version Control

SCCS & RCS (1970s)

CVS (1986)

Subversion (2001)

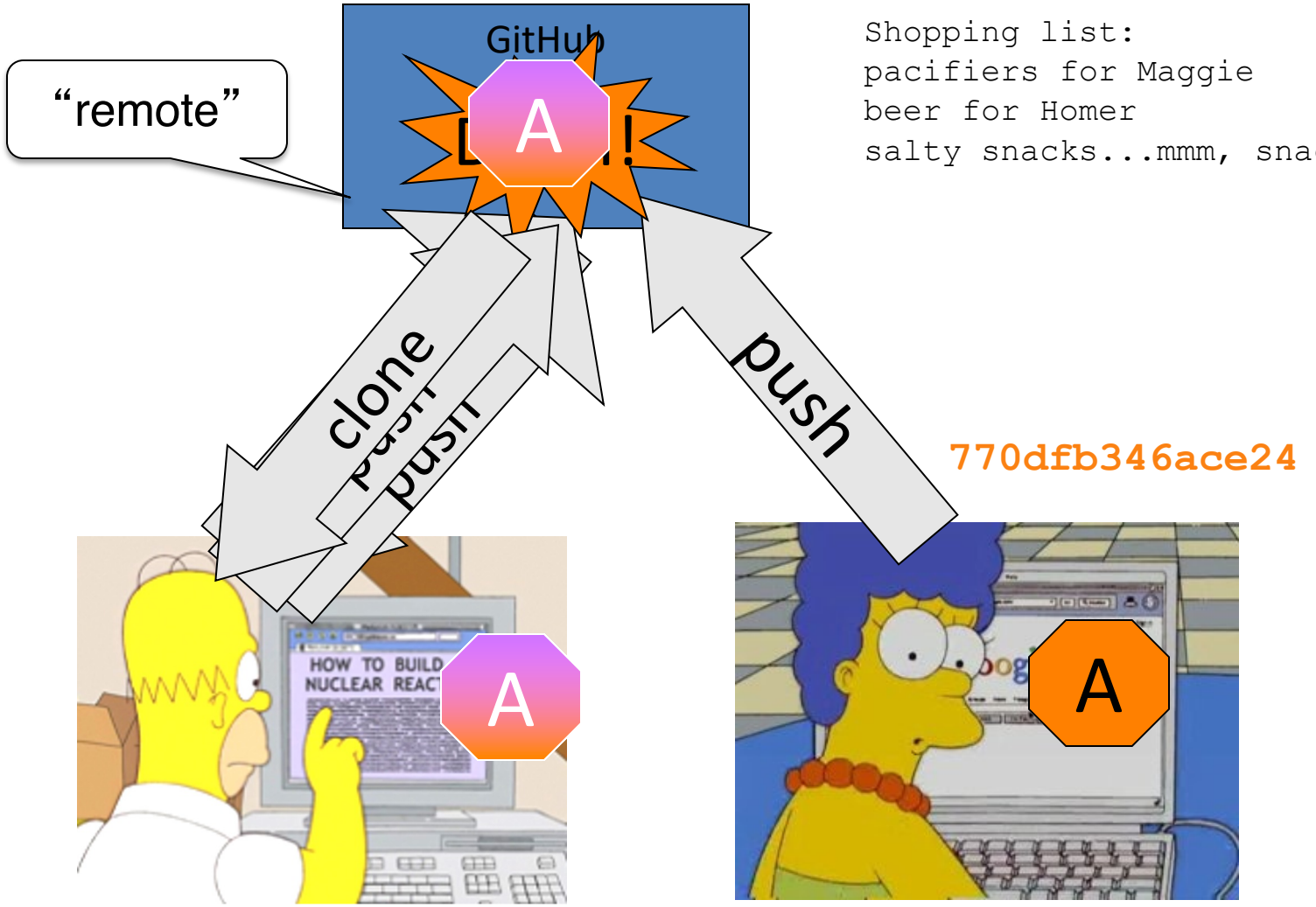
Git (2005)

Image © TheSun.au

40 Years of Version Control

- RCS – a single server
- CVS – a different server, a single master
- Subversion – a different server, a single master
- Git – decentralized, any repo pushes or pulls from any other

How Simpsons Use Git




```
Shopping list:
pacifiers for Maggie
beer for Homer
salty snacks...mmm, snacks
```

Pull = (Fetch + Merge) & Push

- Fetch = copies new commits from the origin
- Merge two repos = try to apply commits in both
  - Conflict if different changes to same file "too close" together
  - `git pull` = `git pull origin master (master -> main)`
- Successful merge implies commit!
- Always commit your changes before merging/pulling
- Commit early & often—small commits OK! `git commit`
- Push - `git push` when all done

Identifying a Commit

- A commit: a snapshot of your repo at a specific point in time
- 40-digit hex hash (SHA-1), unique in the universe…but a pain
  - use unique (in this repo) prefix, eg `770dfb`

  `HEAD`: most recently committed version on current branch

  `ORIG_HEAD`: right after a merge, points to pre-merged version

  `HEAD~`***n***: n' th previous commit

  `770dfb~2`: 2 commits before `770dfb`

  `"master@{01-Sep-2012}"`: last commit on master branch prior to 1-Sep-2012

Undo!

```
git reset --hard ORIG_HEAD
git reset --hard HEAD
git checkout commit-id -- files...
```

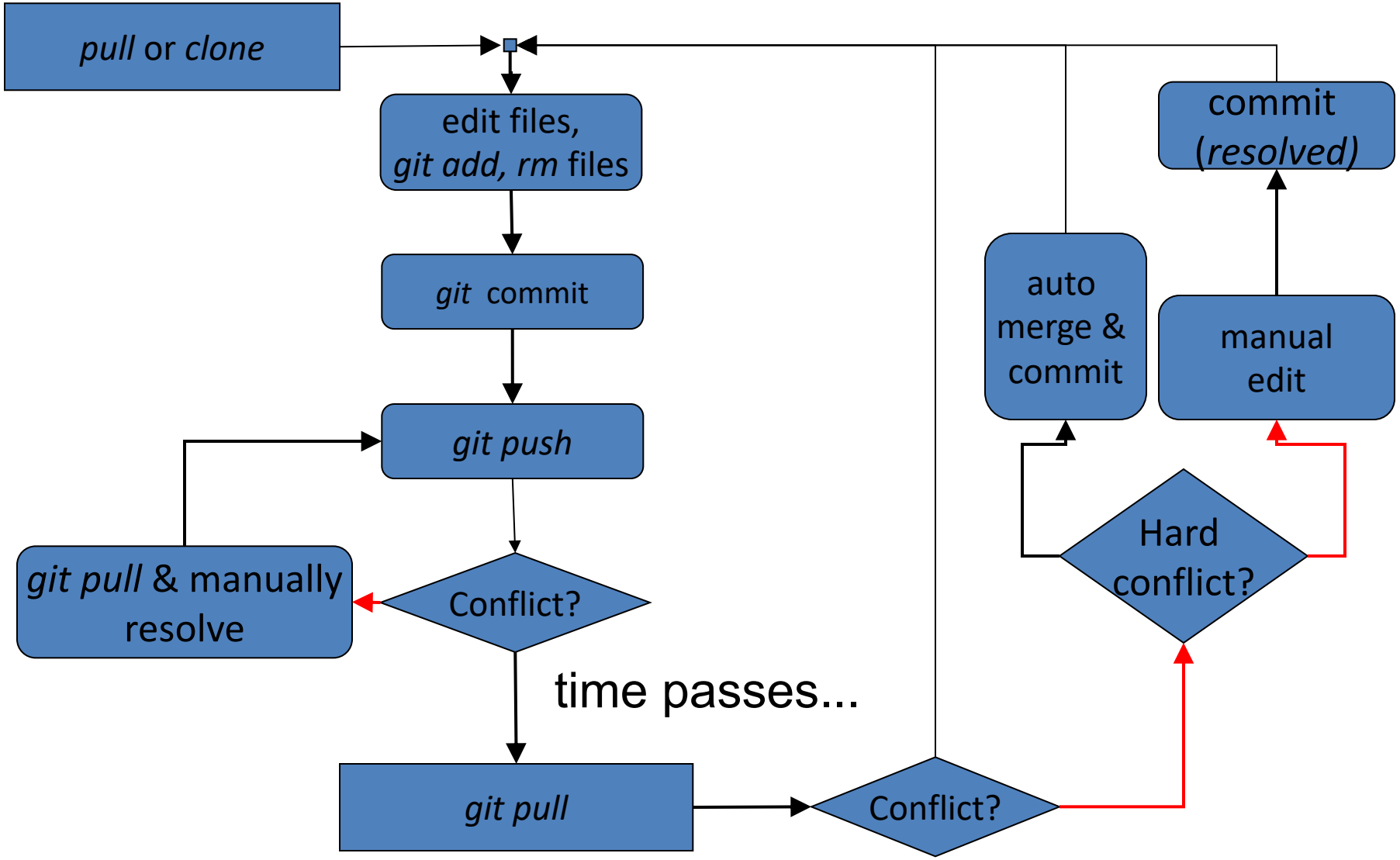
Track who changed what file and when

```
git blame files

git diff files
git diff branch files
git diff "master@{01-Sep-12}" files

git log ref..ref files
git log –since="date" files
```

Recap: Version Control with Conflicts



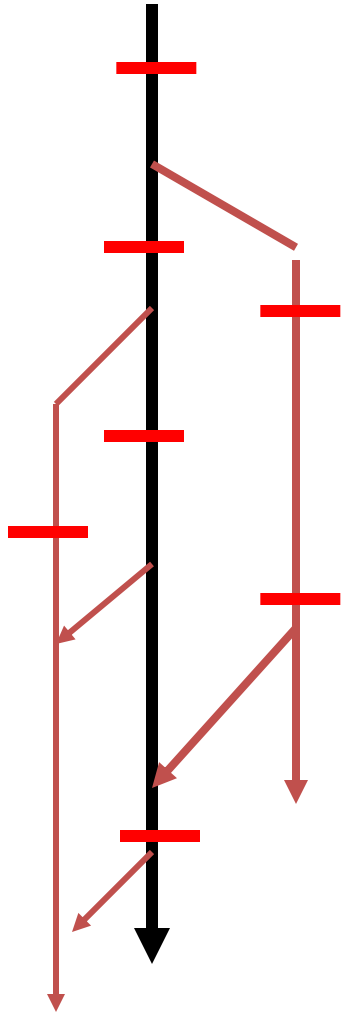
Branches

- To allow part of the team to work on an experimental new feature without disrupting working code
- To fix a bug in a previously-released version of the code that some customers are still using
- To create a release of code

Branches

- Development **master (main)** vs. **branches**
  - Creating a branch is *cheap!*
  - switch among branches: *checkout*
- Separate commit histories per *branch*
- *Merge* branch back into master
  - ...or with *pushing* branch changes
  - Most branches eventually die
- Two common branch management strategies: feature branch, release branch
- Killer use case for agile SaaS: *branch per feature*

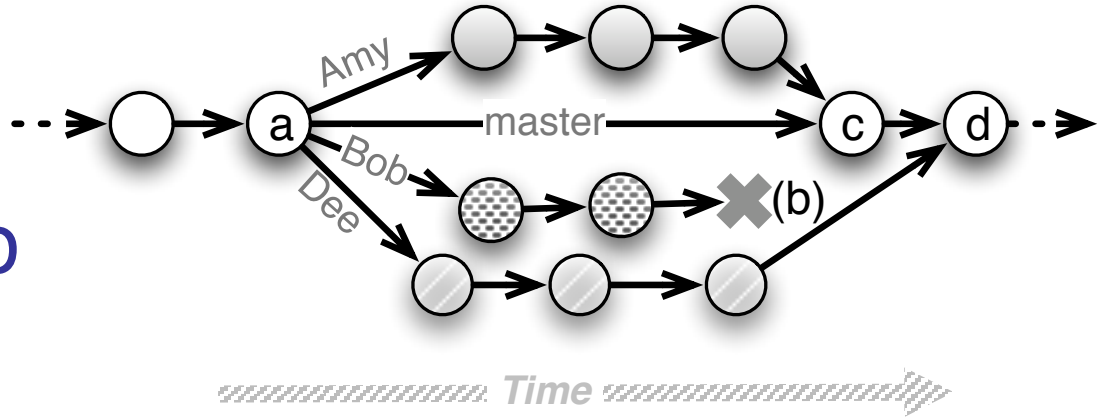
  release branch uncommon in SaaS



Creating new features without disrupting working code

1. To work on a new feature, create a new branch *just for that feature*
   - many features can be in progress at the same time
2. Use the branch *only* for changes needed for *this feature*, then merge into master
3. Back out this feature ⇔ undo this merge

In well-factored app,
1 feature should not
touch many parts of app



Mechanics

- Create a new branch & switch to it

```
git branch CoolNewFeature
git checkout CoolNewFeature
```

← *current branch*

- Edit, add, make commits, etc. on the branch
- Push branch to origin repo (optional):

```
git push origin CoolNewFeature
```

  – creates a *tracking branch* (a local branch that is connected to a remote branch)

- Create a pull request, do code review, and merge into master in origin repo
- Switch back to master, and pull:

```
git checkout master
git pull
```

Branches & Deployment

- Feature branches should be short-lived
  - otherwise, drift out of sync with master, and hard to reconcile
  - `git rebase` can be used to "incrementally" merge

```
          A---B---C topic
         /
D---E---F---G master
```

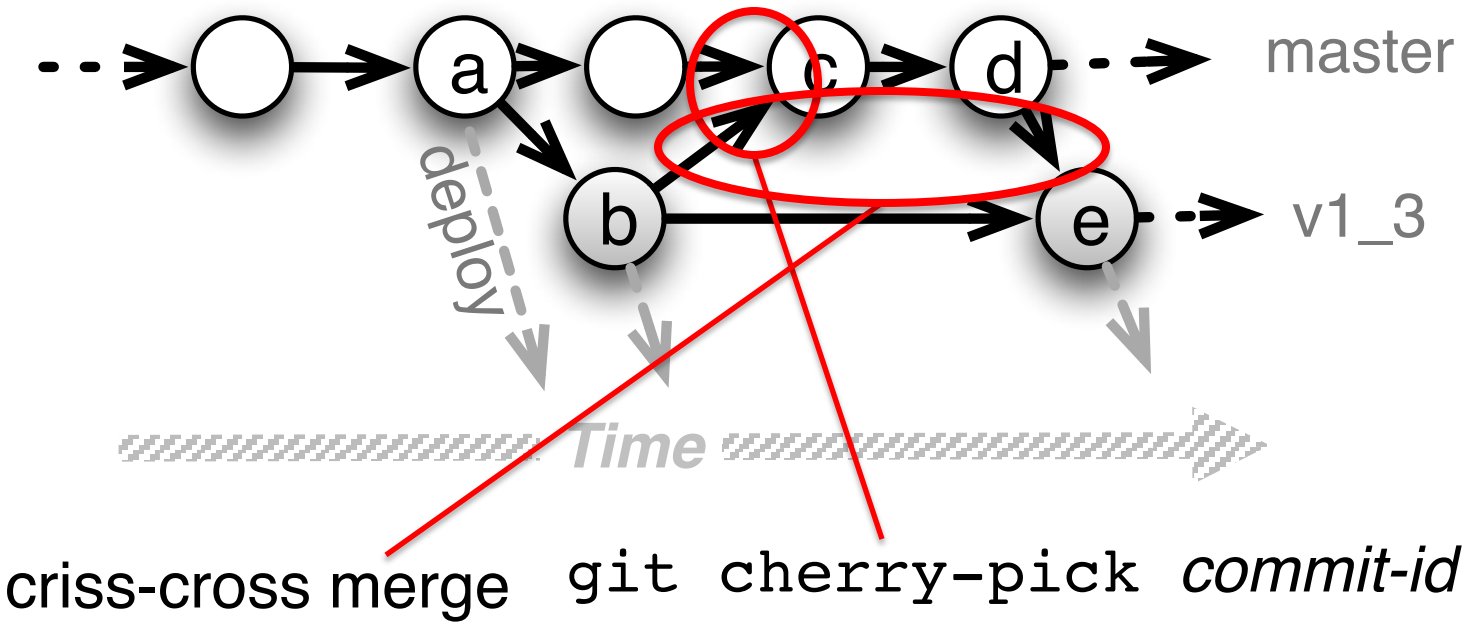
git rebase master

```
                    A'--B'--C' topic
                   /
D---E---F---G master
```

Current branch is topic

  - `git cherry-pick` can be used to merge only specific commits (next slide)
- "Deploy from master" is most common

Release/bugfix branches and cherry-picking commits



Rationale: release branch is a stable place to do incremental bug fixes

Branch vs. Fork

- Git supports *fork & pull* collaboration model
- If you have **push/admin access** on repo:
  - branch: create branch in *this repo*
  - merge: fold branch changes into master (or into another branch)
  - **Create a pull request from the branch for code review rather than folding changes to master**
- If you don't:
  - **fork**: clone *entire repo* on GitHub to one that **you can branch, push, etc.**
  - Finalize your work on its own branch
  - pull request asks owner of original repo to pull specific commits from my forked repo

Open Source Project Case Study:
Apache REEF,
Apache Nemo (incubating)

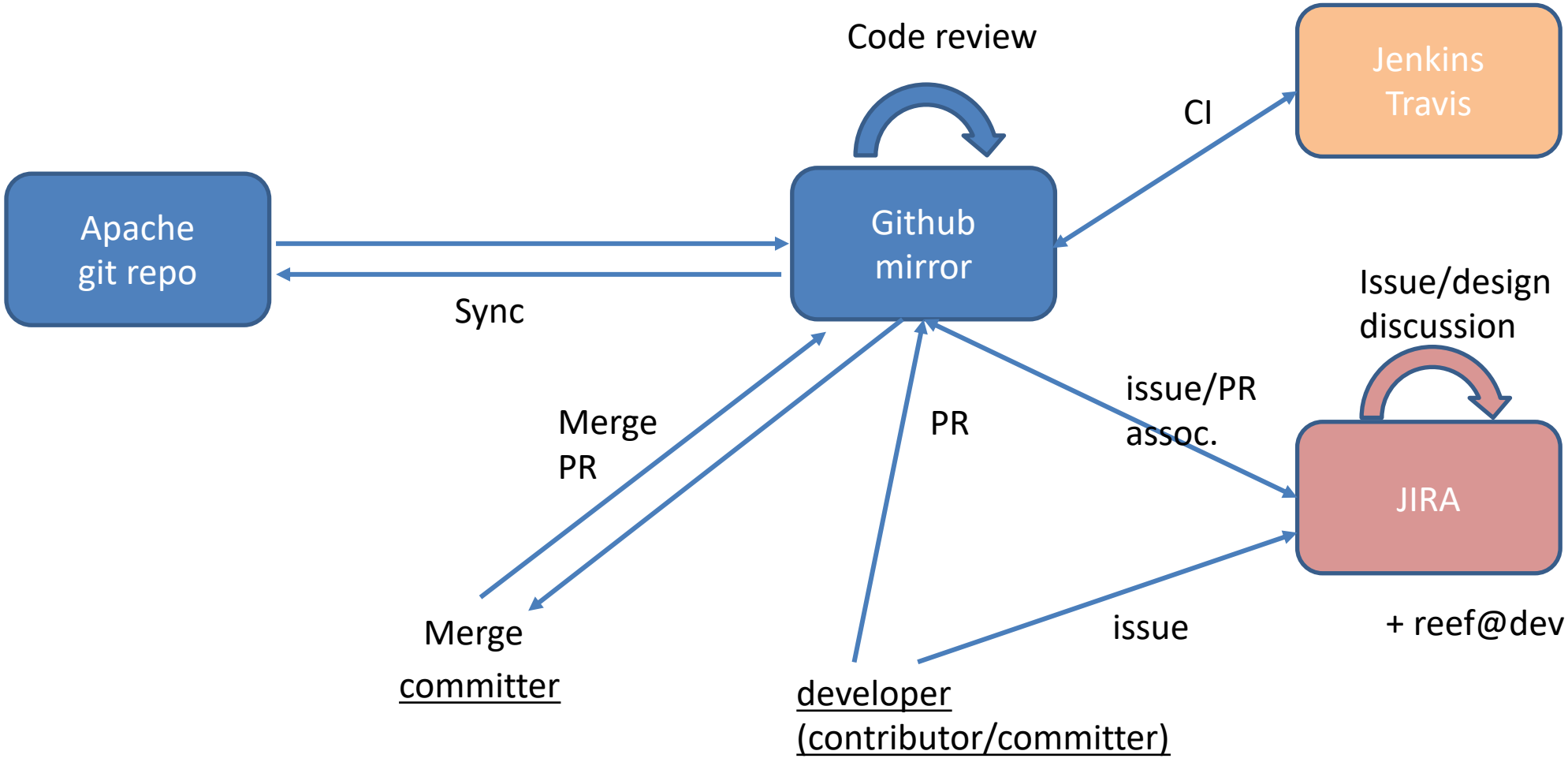
REEF

- Apache REEF™ (Retainable Evaluator Execution Framework) is a library for developing portable applications for cluster resource managers such as Apache Hadoop™ YARN or Apache Mesos™. Apache REEF drastically simplifies the development of applications on those resource managers.
- Started in late 2012
  Apache Incubating in August 2014
  Apache TLP in November 2015
- http://reef.apache.org
- In production use at Microsoft, Twitter, …
- Apache REEF TOCS paper (Sep. 2017) – 27 authors!
- Moved to Attic in 2022

Apache Way

- Incubating => Top-level
- “Meritocracy”
- Project organization
  - Project Management Committee (PMC)
  - Committer
  - Contributor

Revised REEF Project Management Structure



Issue (Feature/Bug) Handling

Class Project

- Github issue registration
- Discussion (design / impl)
- Coding
- Github pull request
- Github code review
- Github code merge
- Close the issue

Apache REEF

- JIRA issue registration
- Discussion (design / impl) through JIRA
- Coding
- Github pull request
- Github code review
- Github code merge
- Close the JIRA issue

Developer Documentation

Created by Markus Weimer, last modified on Oct 02, 2015

Example: Commit Message

- Whichever method you choose, the following should be included i
  - Pull requ
  - A link to
  - The text er of the pull

Example Commit message (80 columns)

Commit message template for contributions from Comitters

```
[REEF-33] Allow Tasks to initiate an Evaluator Heartbeat

  This adds the class HeartBeatTriggerManager which can be used by a Task to
  initiate an Evaluator Heartbeat. It is used by injecting it into the Task.

JIRA:
  [REEF-33] https://issues.apache.org/jira/browse/REEF-33

Pull Request:
  Closes #24
```

Example: Squashing Commits

- Typically, one PR addresses one JIRA issue.
- Github PR may contain multiple commits.
- In the rebase process, the committer must squash commits to have one commit to address the JIRA issue in the Apache git repo

```
pick 7387a49 Comment for first commit
squash 3371411 Comment for second commit
squash 9bf956d Comment for third commit
```

Design Discussion Example

- Tang Namespace (Gyewon Lee) https://issues.apache.org/jira/browse/REEF-31

Gyewon Lee added a comment - 10/Nov/14 17:29

In 6, I said

"I think getBinding() could take a namespace's name for its argument" but that's not what I intended. What I suggest is getBinding() contains only name for the target namespace so it does not have direct reference to the ConfigurationBuilder.

Thanks
Gyewon
Reply

Markus Weimer added a comment - 10/Nov/14 18:17

Regarding #6: How about adding a method like this to `ConfigurationBuilder`:

```
public void shareBinding(Name<?> whatIsBeingShared, String...
nameSpaces);
public void shareBinding(Class<?> whatIsBeingShared, String...
nameSpaces);
```

Add utility function in Tang to parse command lines

This would allow users to tell Tang to share the given instance with the given namespaces. If more than one of the given namespaces already binds this name or interface, the call throws a `BindException`. Another idea would be to expose a source namespace:

```
public void shareBinding(Name<?> whatIsBeingShared, String
```

Code Pull Request Example (JIRA: REEF-30)

[REEF-30] Run REEF on Mesos #52

Closed **johnyangk** wants to merge 14 commits into `apache:master` from `johnyangk:REEF-30`

Conversation 33 | Commits 14 | Files changed 35

johnyangk commented on Jan 19

JIRA: [REEF-30]: https://issues.apache.org/jira/browse/REEF-30

reef_on_mesos

What's in it

- reef-runtime-mesos package
- bin/runmesostests.sh (for running reef-tests on Mesos clusters)
- HelloREEFMesos in reef-examples
- MesosTestEnvironment in reef-tests
- pom.xml changes(adding reef-runtime-mesos) in root, reef-tests, reef-examples

How to test(reef-tests) it

Pretty similar to running reef-tests on YARN.

1. run mesos daemons (http://mesos.apache.org/gettingstarted/)
2. At $REEF_HOME, "mvn clean install -DskipTests"
3. At $REEF_HOME/reef-tests, "mvn jar:test-jar"
4. At $REEF_HOME, "bin/runmesostests.sh $YOUR_MESOS_MASTER_IP"

Code Review Example
(JIRA: REEF-30)

.../runtime/mesos/util/HDFSConfigurationConstructor.java View full changes

((11 lines not shown))

```
+ *
+ * Unless required by applicable law or agreed to in writing,
+ * software distributed under the License is distributed on an
+ * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
+ * KIND, either express or implied.  See the License for the
+ * specific language governing permissions and limitations
+ * under the License.
+ */
+package org.apache.reef.runtime.mesos.util;
+
+import org.apache.hadoop.conf.Configuration;
+import org.apache.reef.tang.ExternalConstructor;
+
+import javax.inject.Inject;
+
+public final class HDFSConfigurationConstructor implements ExternalConstructor<Cor
```

markusweimer added a note on Jan 20

We have a very similar class in the YARN runtime. It might be a good idea to move it to `reef-utils-hadoop` and reuse it here instead of this one.

johnyangk added a note on Jan 20

Can I create a separate JIRA issue for this? It may be good to move this and the one in the YARN runtime together.

Contributing to Nemo

https://github.com/apache/incubator-nemo/blob/master/.github/CONTRIBUTING.md

Contributing to Nemo

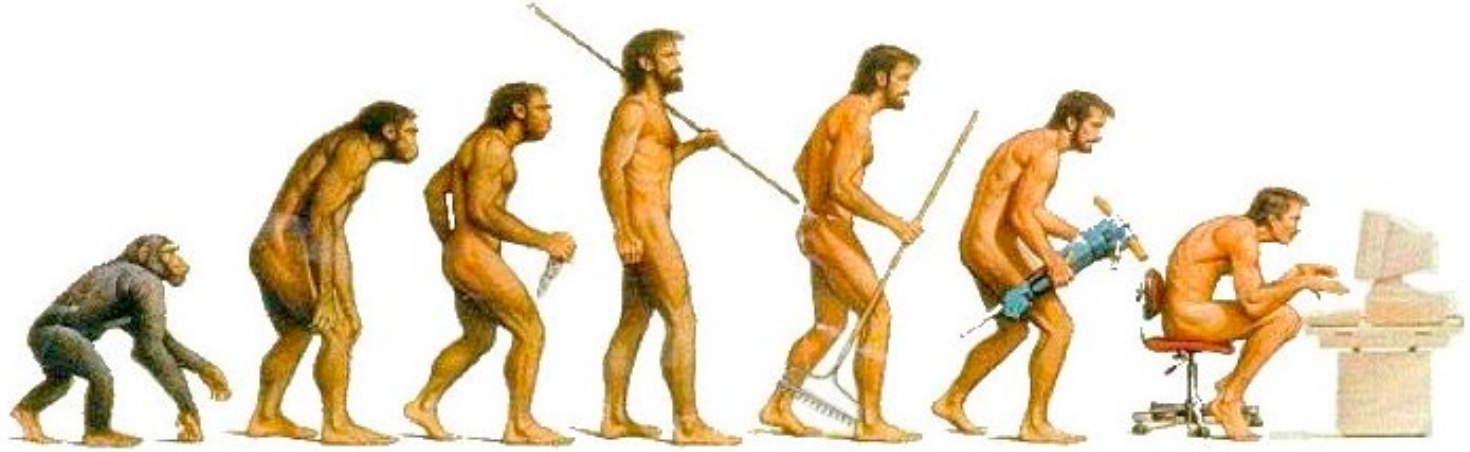
🎉 Thanks for taking the time to contribute! 🎉

This project and everyone participating in it is governed by the Code of Conduct.

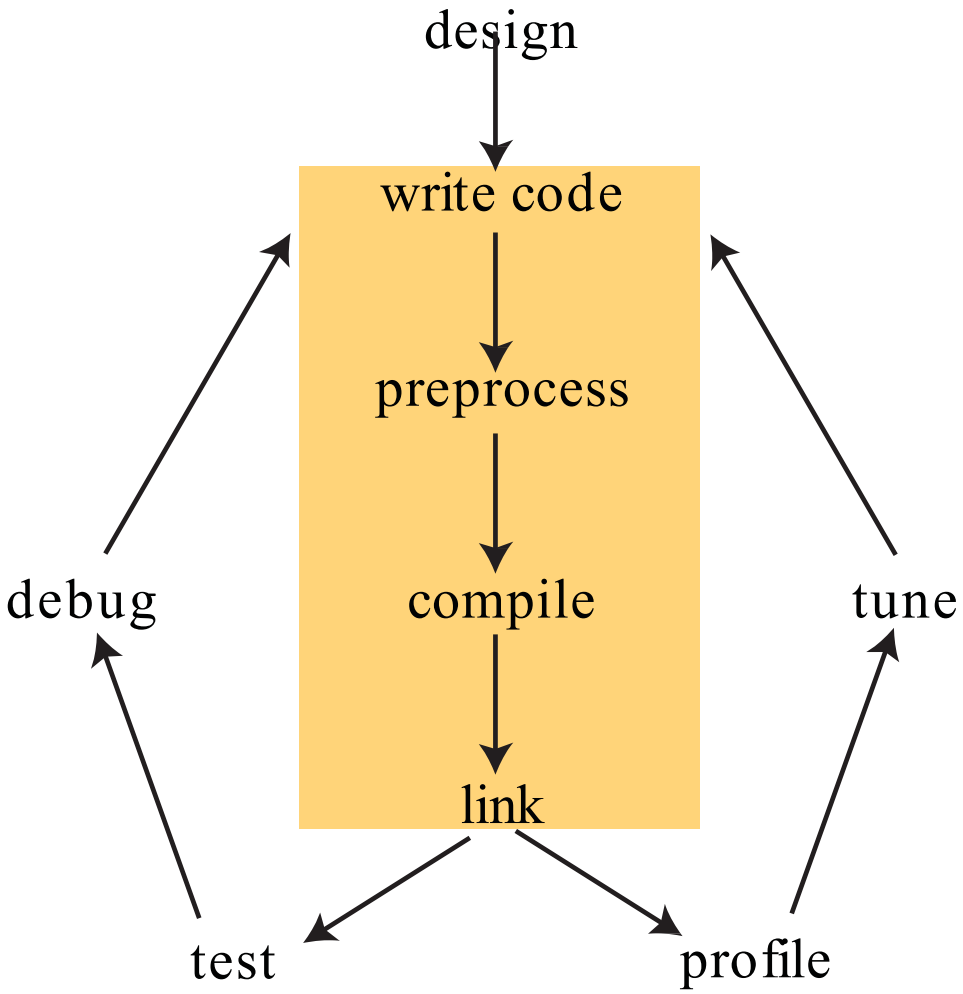
Before contributing to our project, keep in mind that we go through the following simple steps:

- Identify the change required for the project.
- Search and check for existing, related JIRA tickets and pull requests. Make a new JIRA ticket if the problem is not pointed out.
- Make sure that the change is important and ready enough for the community to spend time reviewing
- Open the pull request following the PR template, clearly explaining and motivating the change.

The Software Developer’s Toolbox

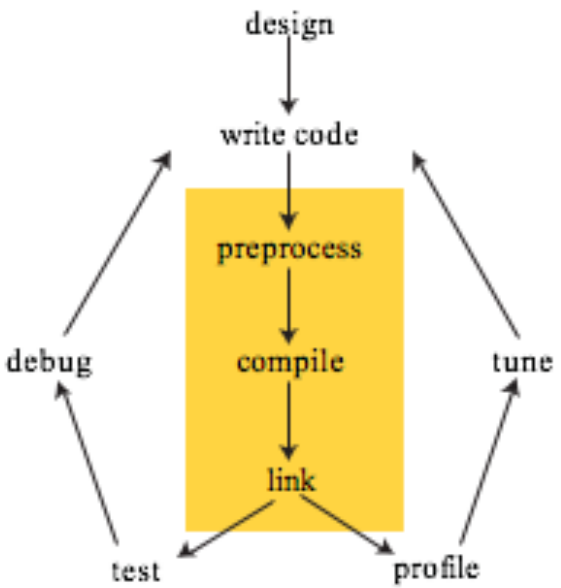


- Humanity progresses(?) when it gets new tools
- Always choose the right tools for the job
- Keep them sharp
  - Leading-edge tools -> coding productivity improves > 50%


design
write code
preprocess
compile
link
debug
tune
test
profile

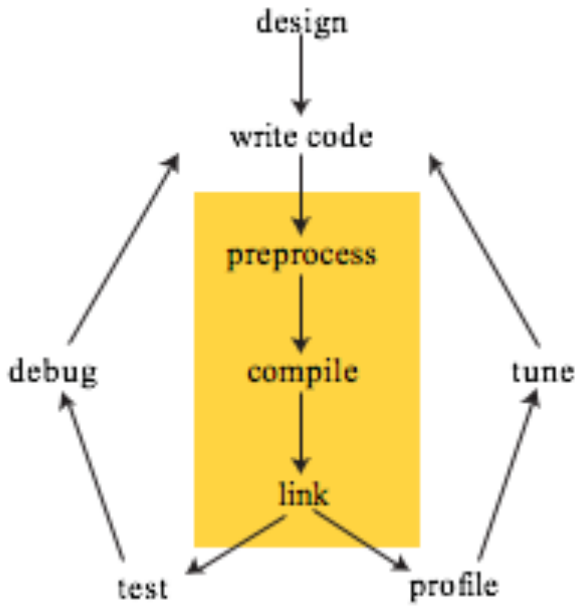
Build Tools

- Preprocessor
  - *macros → save typing, provide consistency*
  - *helps exclude debug code from shipped code*
- Compiler
  - *preprocessed source code → object code*
  - *static analyses (syntax, semantics, warnings)*
  - *can compute complexity metrics*
- Linker (static or dynamic)
  - *pieces together object code into executable software*



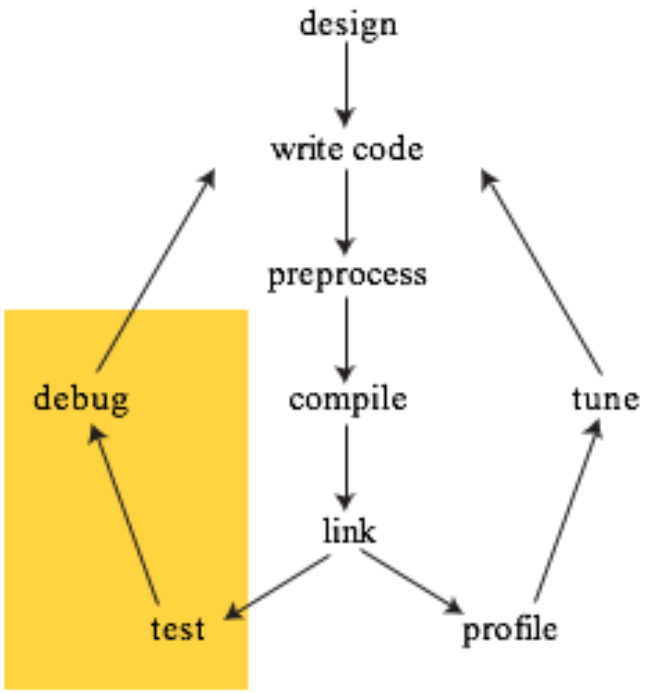
Build Tools

- Builders
  - *examples: maven, make, bazel, buck, etc.*
  - *declaratively specify how to build the software*
  - *minimizes the time needed to build the executable*
  - *Incremental build*



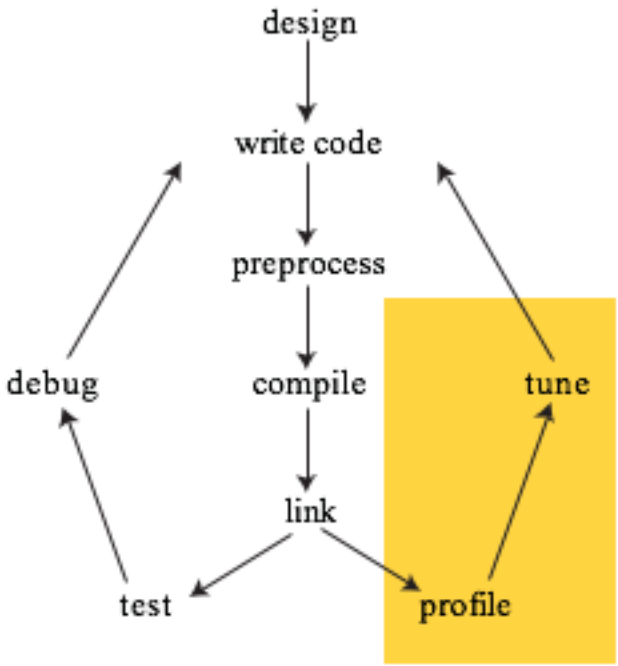
Testing/Debugging

- Test framework
  - unit test (JUnit, CppUnit, unittest, etc.)
  - test generators, record/playback tools
  - fault injectors
  - coverage tools: *coverage*
- Debugging
  - logging frameworks, trace monitors, diff tools
  - interactive debuggers



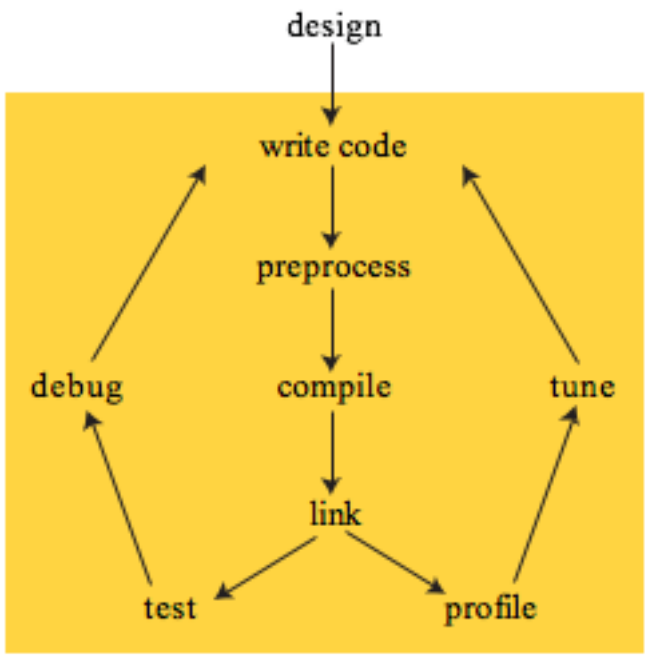
Profiling

- Profiler is like a stethoscope
  - count # execs of statements, functions, etc.
  - time spent, identify bottlenecks
  - trace library calls, system calls, etc.
  - resource (CPU, memory, I/O, network) profiling
  - APM (Application Performance Monitoring)
  - E.g., cProfile, gperftools



Development Environment

- Integrated Development Environments (IDEs)
  - *single portal into development sequence*
  - *edit code, refactor, navigate the code*
  - *auto-formatting and coloring code*
  - *compile, analyze, find errors, interactive help*
  - *templates, search-and-replace*



Collaboration Tools

- Version control system
  - *ideally integrated with defect tracking*
  - *standardized configurations*
  - *make backups (and test your backup plan)*

More Tools

- Continuous integration servers (e.g., Jenkins, Travis)
- Static analysis tool (sonarcloud)
- Documentation automators (Javadoc, Doxygen, ...)
- CASE (Computer-Aided Software Engineering)
  - *use visual representations to describe program logic*
  - *code generators (beware of maintainability)*
- *...*