Proposed Software Workflow Spring 2020

Time and Date: TBD

The way we want to do software development

Outline of Workshop

- Why Be Concerned With Software Development
  - Things to Consider
  - Motivating Examples
- The Github Workflow
- Tests and TDD
- Recommended Tools
- Future Tools - Not Yet Fully Implemented
- Documentation
- Deliverables
- Practical Session

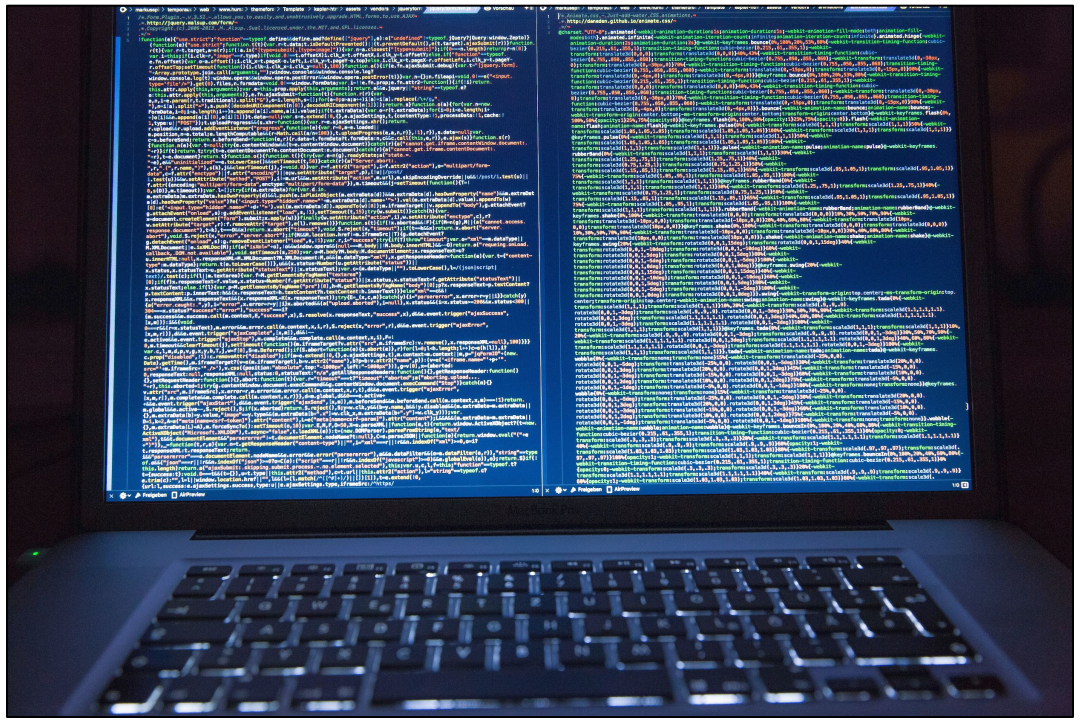
[Link to Proposed Workflow for Software Development Document]

Software Development - Things to Consider

- Developing Code for Larger Project
- Readability and Conformity
- Keep it General
- Keep it Simple
- Keep it Sufficient
- Avoid Special Cases
- keep it modular
- share code as much as possible

Software Development - Motivating Examples

- Glitch causes SolarCity Corp to be undervalued by $400 million in acquisition
- Hawaii Sends Out a State-Wide False Alarm About a Missile Strike
- Glitch in F-35 fighter planes causes target detection problems
- Medicine infusion pumps recalled for deadly flaw
- Frenchman sues uber over a software bug
- The Equifax social security hack
- Bug assists in bank heist
- ARIANE 5 Failure



Detailed Description

Software Development - How Not to Do It

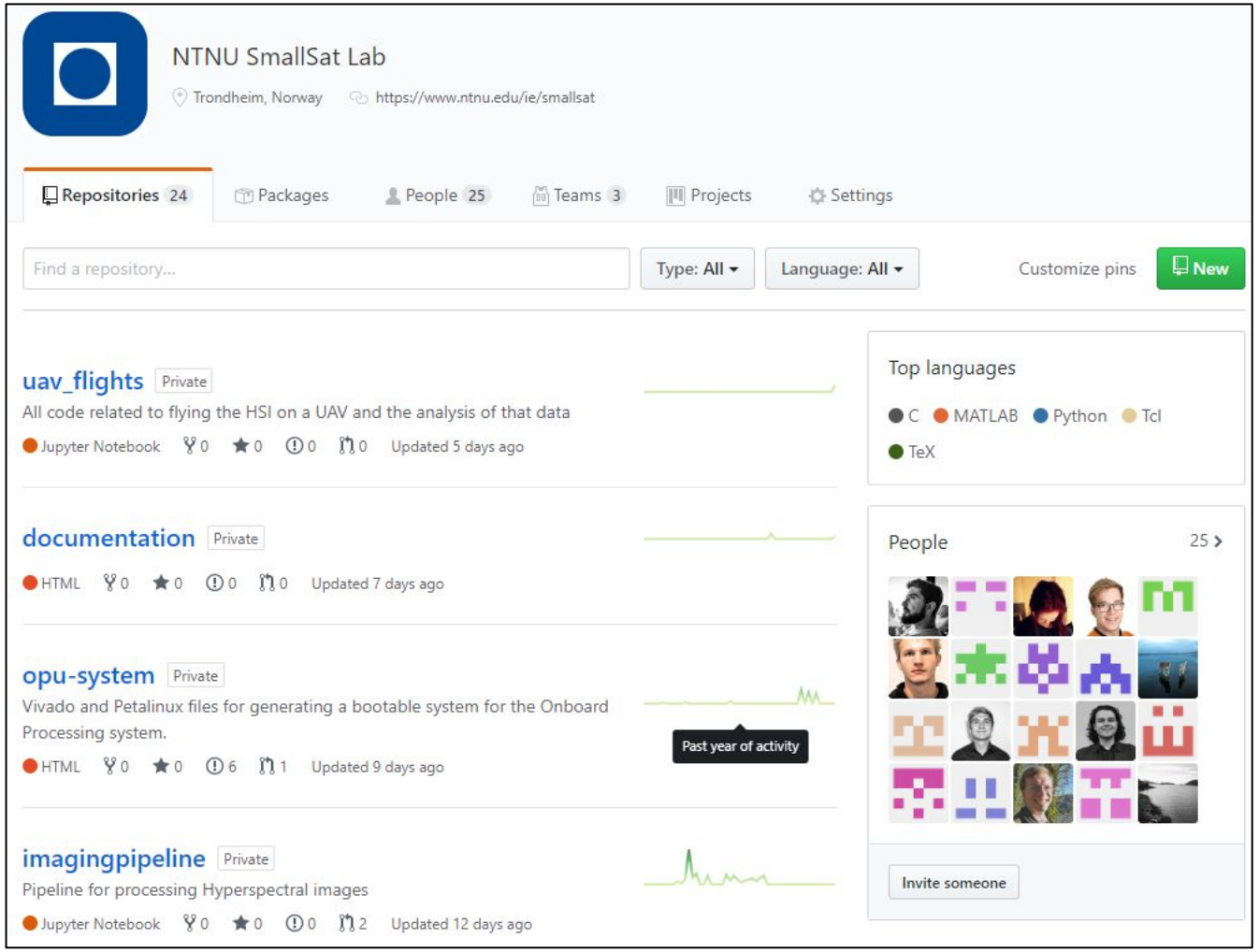
LocalHawk

KONGSBERG

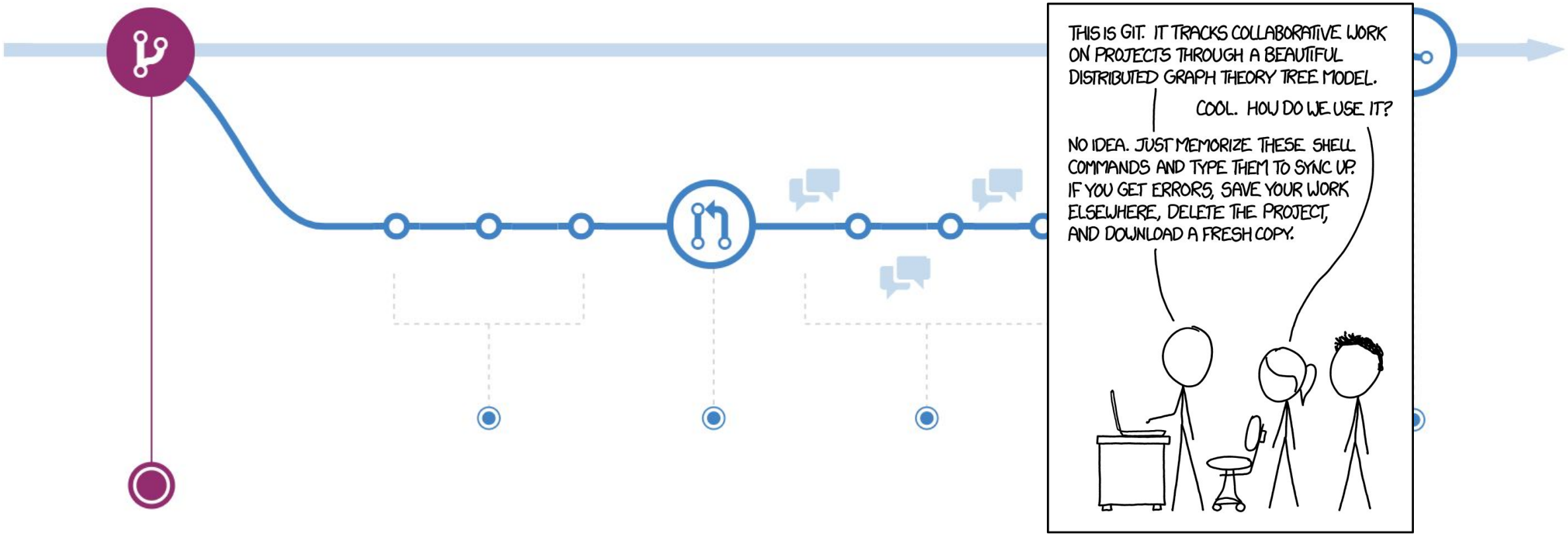
http://www.localhawk.net/localhawk-2017/
http://www.localhawk.net/intro-2/

The GitHub Workflow - Overview

- GitHub
  - hosting service for development and version control using Git.
- Github Flow
  - GitHub flow is a lightweight, branch-based workflow that supports teams and projects where deployments are made regularly.
- Docker
  - OS-level virtualization to deliver software in containers
- Test frameworks
  - cpputest, check, pytest

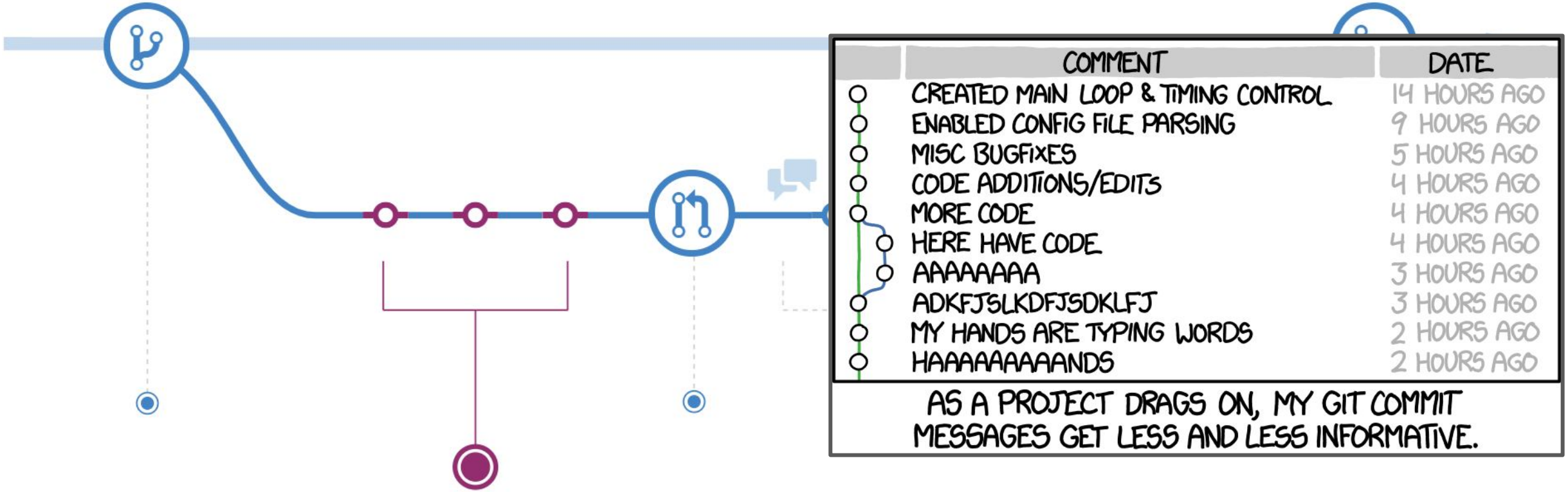


The GitHub Workflow - Branching



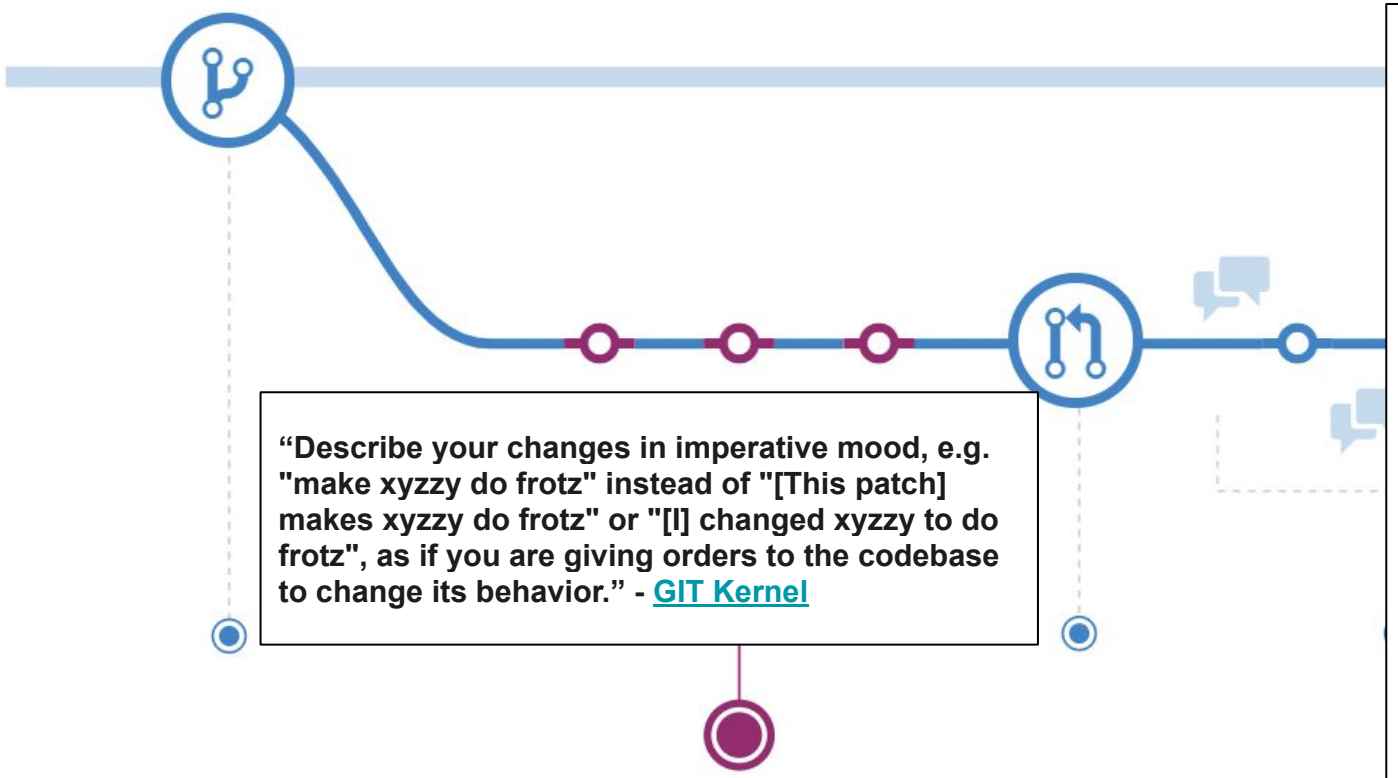
- Make a properly named branch for the feature, bug fix, test you are developing
  - Good: dev_module_name, dev_temp_sensor, test_module_name, test_temp_sensor, etc.
  - Bad: dev_sivert, new_feature, dev_feature, test, etc.

The GitHub Workflow - Commits

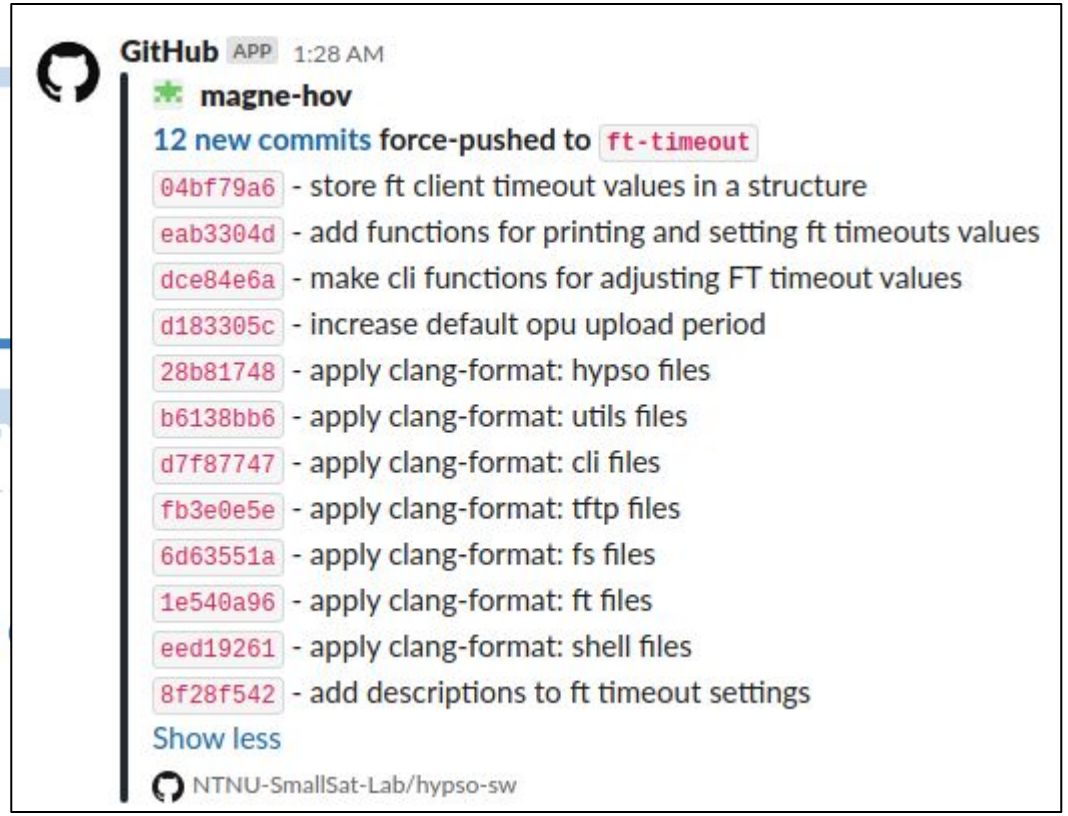


- Make commits a lot! Have coherent commit messages, rubber duck
  - Good: “specified functionality for module”, “test for max limit of temperature guard”, etc.
  - Bad: “bug fixes”, “edit of module”, “code”, “asindajdpopsa”, etc

The GitHub Workflow - Commits

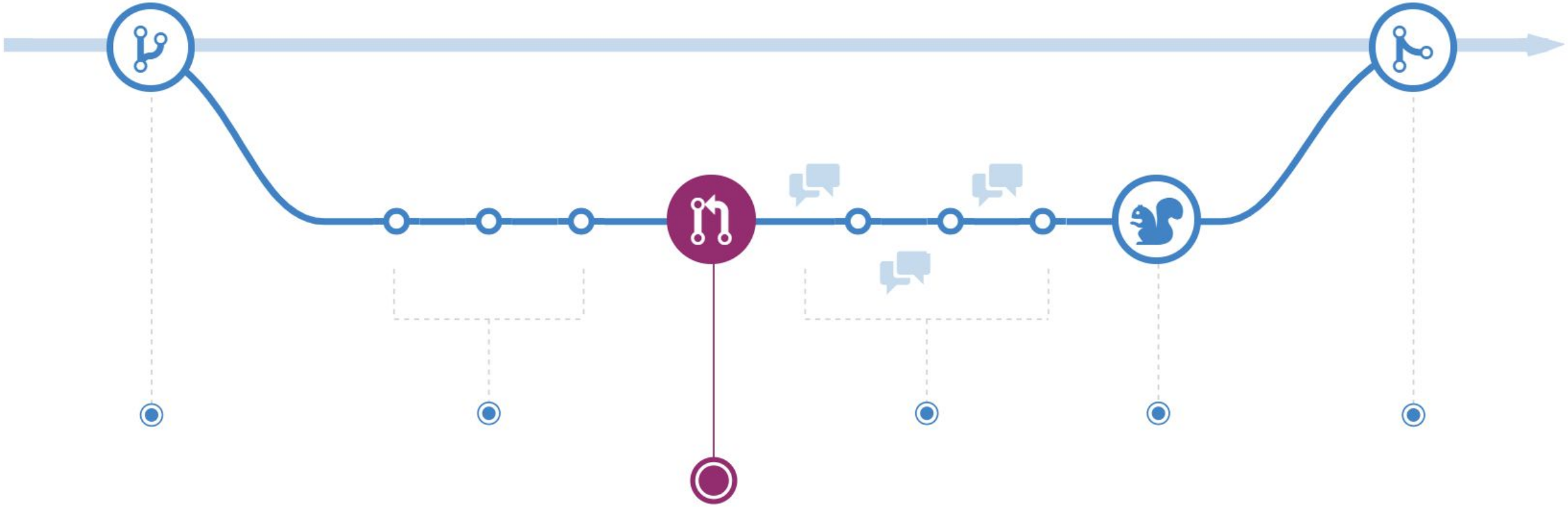


“Describe your changes in imperative mood, e.g. "make xyzzy do frotz" instead of "[This patch] makes xyzzy do frotz" or "[I] changed xyzzy to do frotz", as if you are giving orders to the codebase to change its behavior.” - GIT Kernel



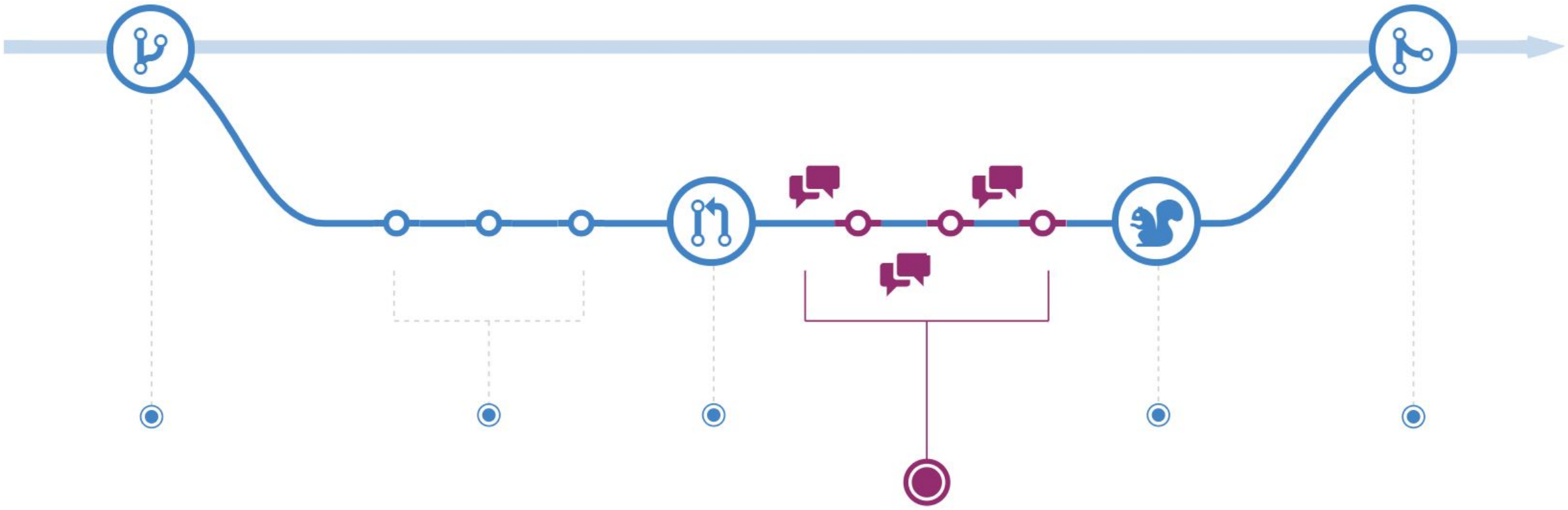
- Make commits a lot! Have coherent commit messages, rubber duck
  - Good: “specified functionality for module”, “test for max limit of temperature guard”, etc.
  - Bad: “bug fixes”, “edit of module”, “code”, “asindajdpopsa”, etc

The GitHub Workflow - Pull Request



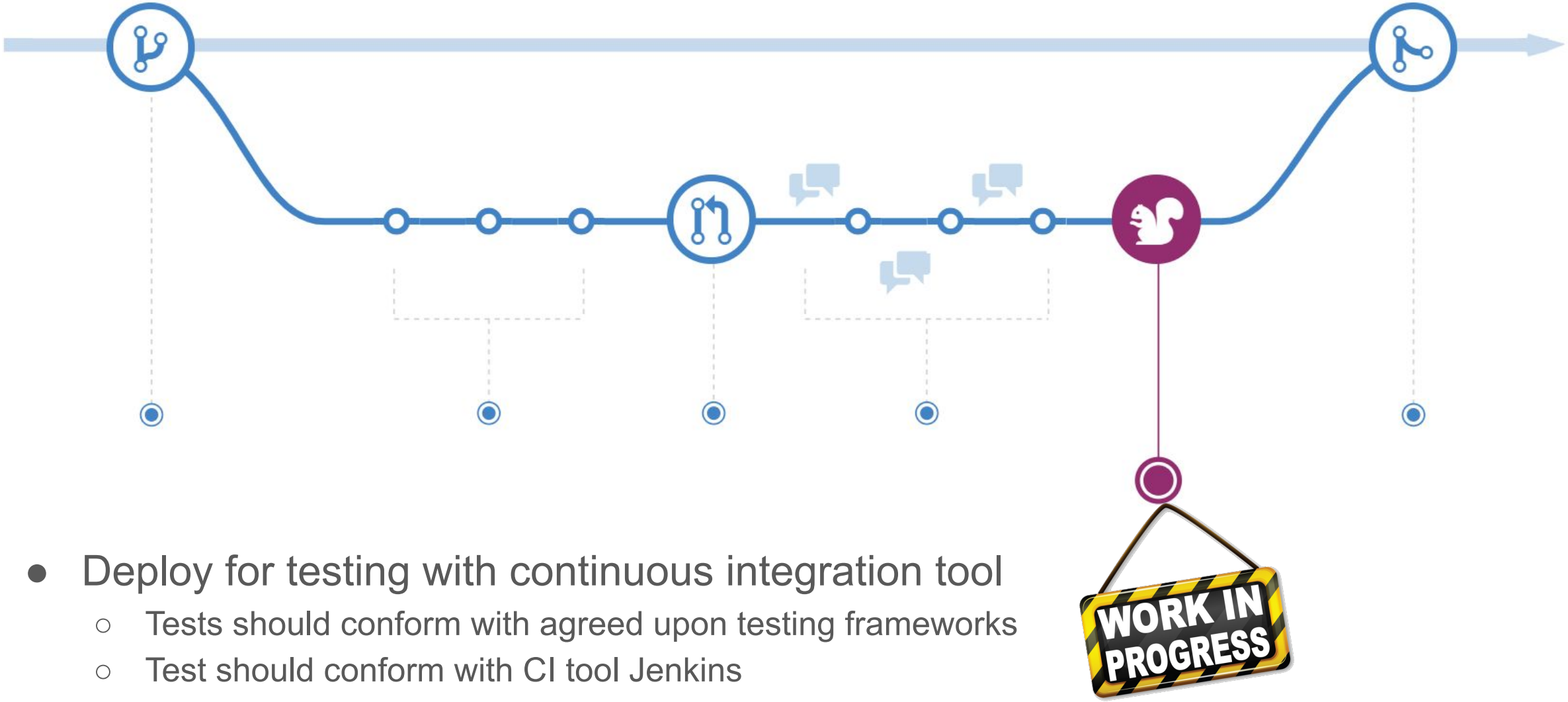
- Pull Requests, make sure that your code is ready to be merged with master
  - Look at each other's work, **Test it**, make suggestions, commend good work
  - Help each other, be friends

The GitHub Workflow - Overview



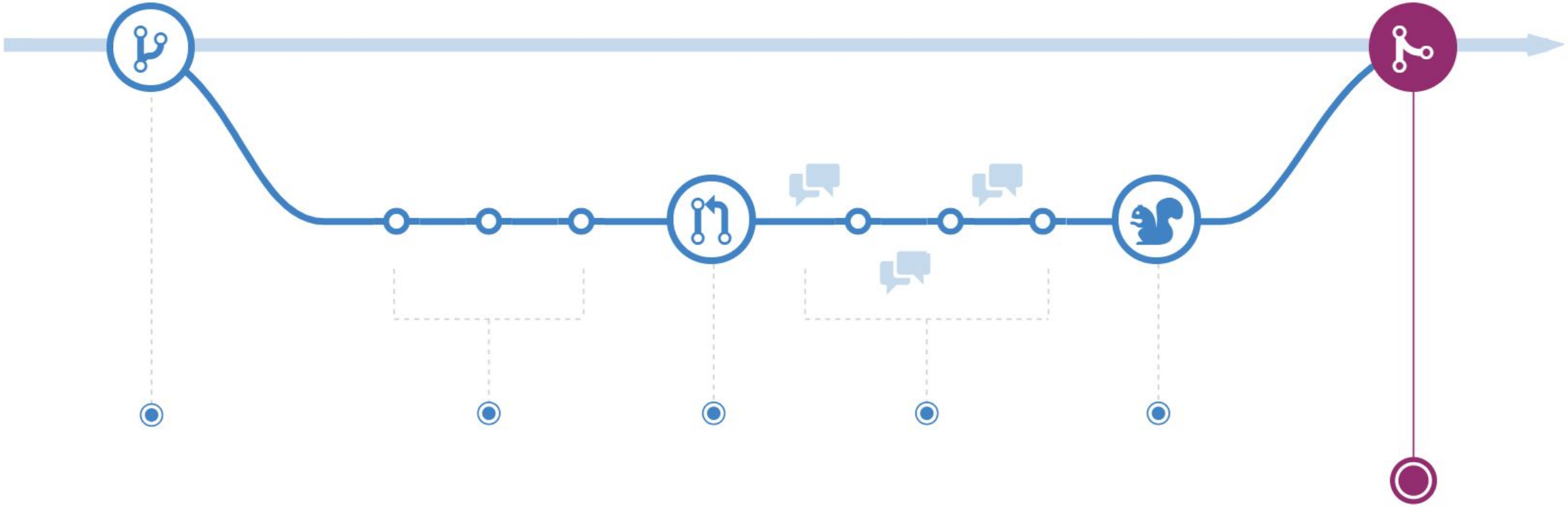
- Discuss and review code
  - Help each other identify strengths and weaknesses of implementation, **Test it**
  - Make it happen since there is a requirement of 2 reviewers

The GitHub Workflow - Overview



- Deploy for testing with continuous integration tool
  - Tests should conform with agreed upon testing frameworks
  - Test should conform with CI tool Jenkins

The GitHub Workflow - Overview



- Merge
  - Should merge back into master, enabling a new feature, test or bug fix to the main product
  - Only have one master branch in hopes to reduce complexity

Tests and TDD

- Unit Testing
  - a testing method by where individual units of the source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use.
- Integration testing
  - software testing in which individual software modules are combined and tested as a group. Integration testing is conducted to evaluate the compliance of a system or component with specified functional requirements. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

Tests and TDD

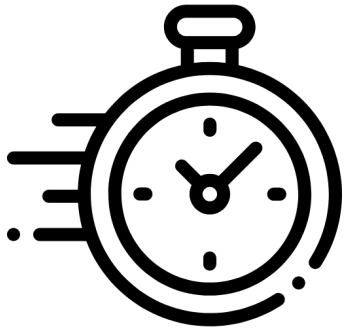
- Regression testing,
  - re-running functional and non-functional tests to ensure that previously developed and tested software still performs after a change. If not, that would be called a regression. Changes that may require regression testing include bug fixes, software enhancements, configuration changes, and even substitution of electronic components.
- Acceptance tests
  - testing with respect to user needs, requirements, and processes needed to determine whether or not a system satisfies the acceptance criteria and to enable the user to determine whether or not to accept the system.
- Performance testing
  - testing to determine how a system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage. Build performance standards into the implementation, design, and architecture of the software.

Test Driven Development - Warning!

- Potential Pitfalls
- Focus on Correctness, not passing tests
- Quality is made by design, not testing
- TDD is Time Consuming and Costly
    - both Short Term and Long Term

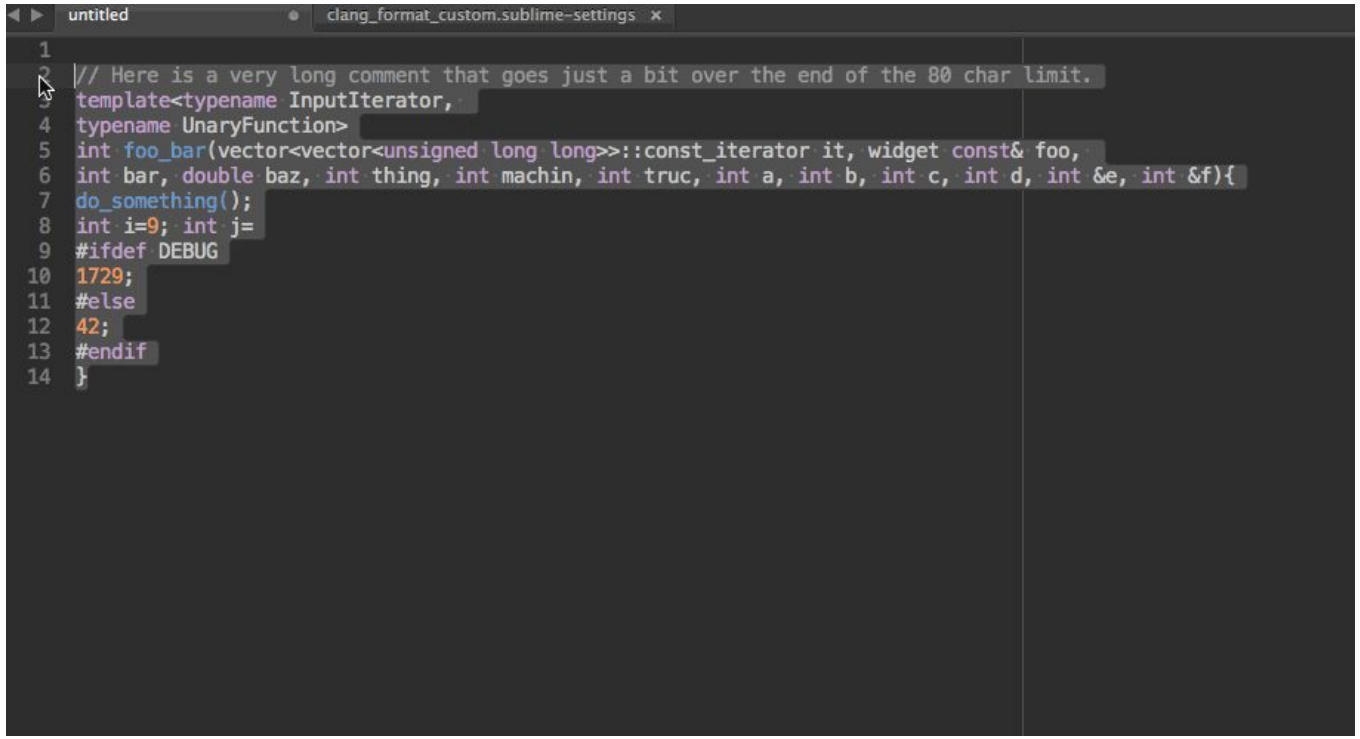
Recommended Tools

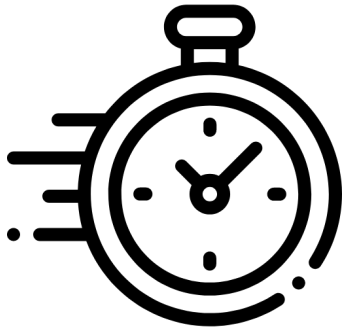
- Revision Control e.g. GitHub
  - Does most of the job for you
- Docker
  - Uniform development environment
  - OS-level virtualization to deliver software in containers
- Programming syntax aids
  - Use lint e.g. pylint for python
  - Use static code analysis e.g. clang
  - Use style guide e.g. PEP8 and clang-format
  - Use each other
  - Do reviews



Recommended Tools - Examples

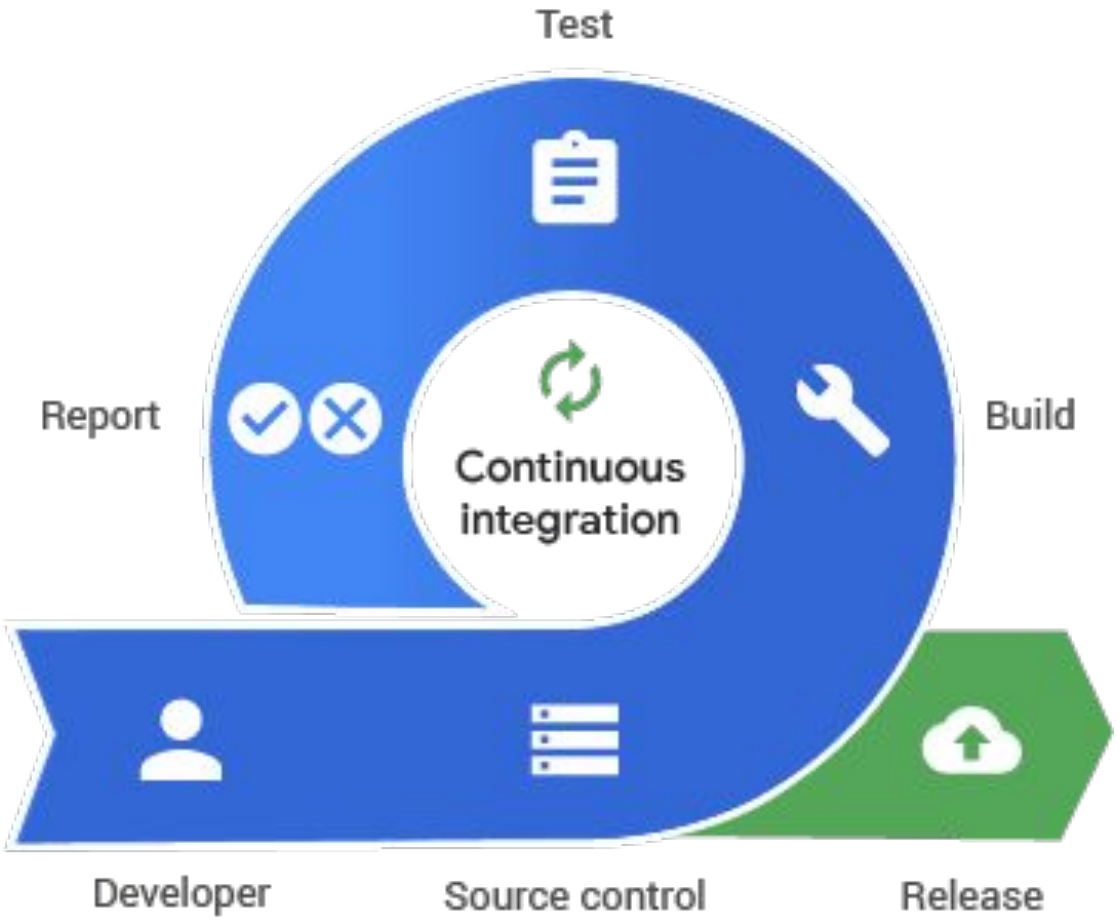
- Automated application of style guide

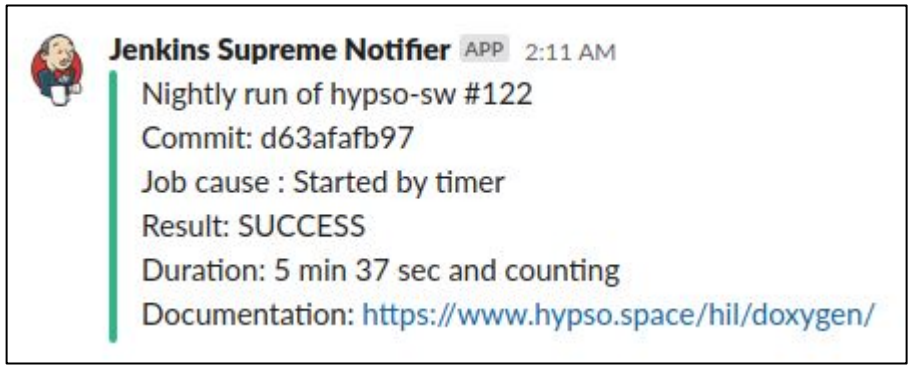




Future Tools - Not Yet Implemented

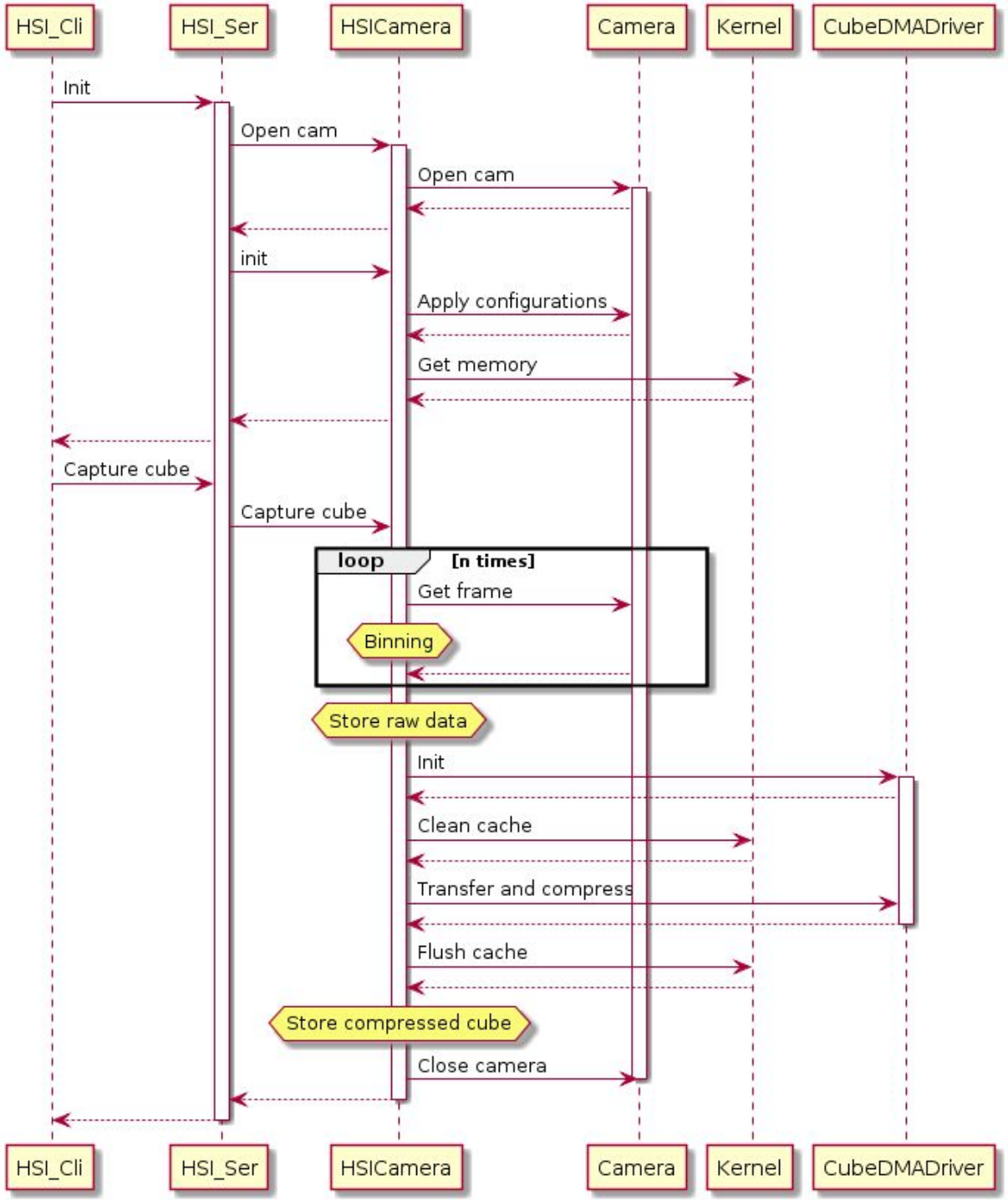
- Continuous Integration
  - Version Control System tool
  - Build Tool
  - Artifacts Repository Manager
- More?





Documentation

- The importance of standards
- Uniform Modelling Language
  - PlantUML
  - Good abstraction
  - Easier than just code
  - Not perfect
  - Still good

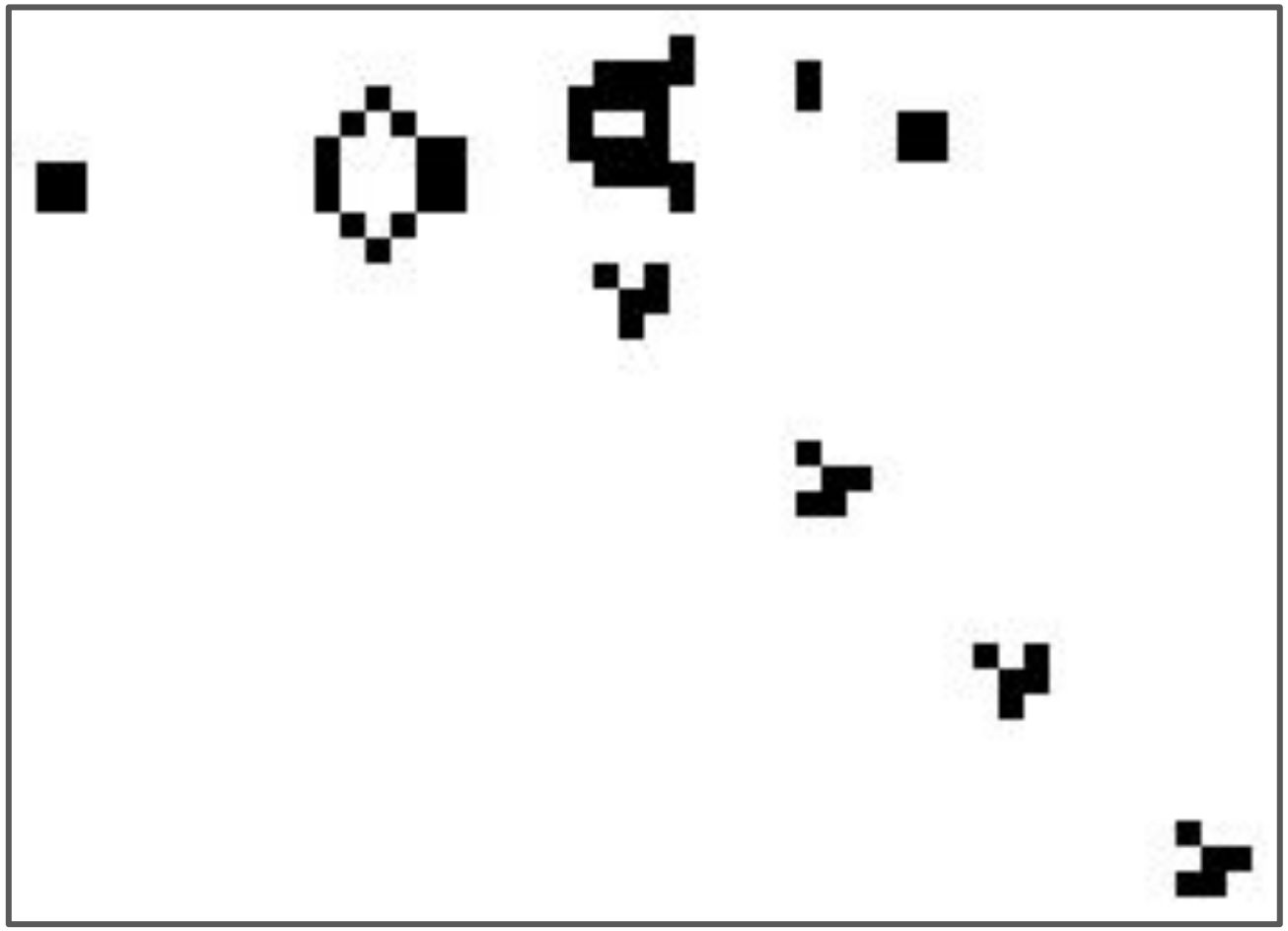


Deliverables

- Please read [Proposed Workflow for Software Development]
  - Add comments
  - Include recommendations
  - Identify unclear or ambiguous elements
- Software Description
  - What will you make?
  - How will it work?
  - How will it be integrated?
- verification and validation / test-plan description
  - How will you test the feature(s)?
  - How will it be verified?
  - How will it be validated?
- Implementation

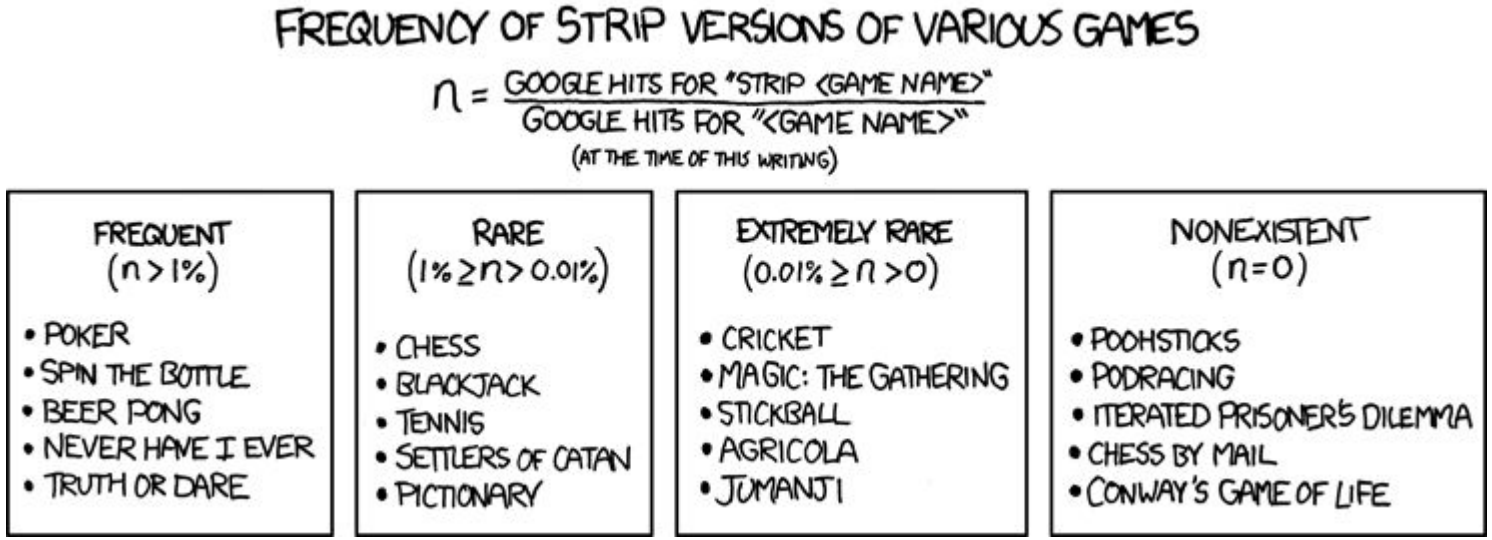
Practical Session - Conway's Game of Life

- Use Github
  - Fork this repo
- Use Provided Tools
  - Docker
  - Pylint
  - autopep8
  - Each other
  - Github Flow
- Plan functions
- Make tests first!
- Use Github Flow



Practical Session - Conway's Game of Life

- Rules
  - Any cell can be either alive or dead
  - Any live cell with fewer than two live neighbours dies, as if by underpopulation
  - Any live cell with two or three live neighbours lives on to the next generation
  - Any live cell with more than three live neighbours dies, as if by overpopulation
  - Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction
- Make a 40x40 grid of cells, randomly initialized
- Simulate 100 iterations
- What to do on the edge?



Time 1+ hr

It’s okay if you don’t finish :)