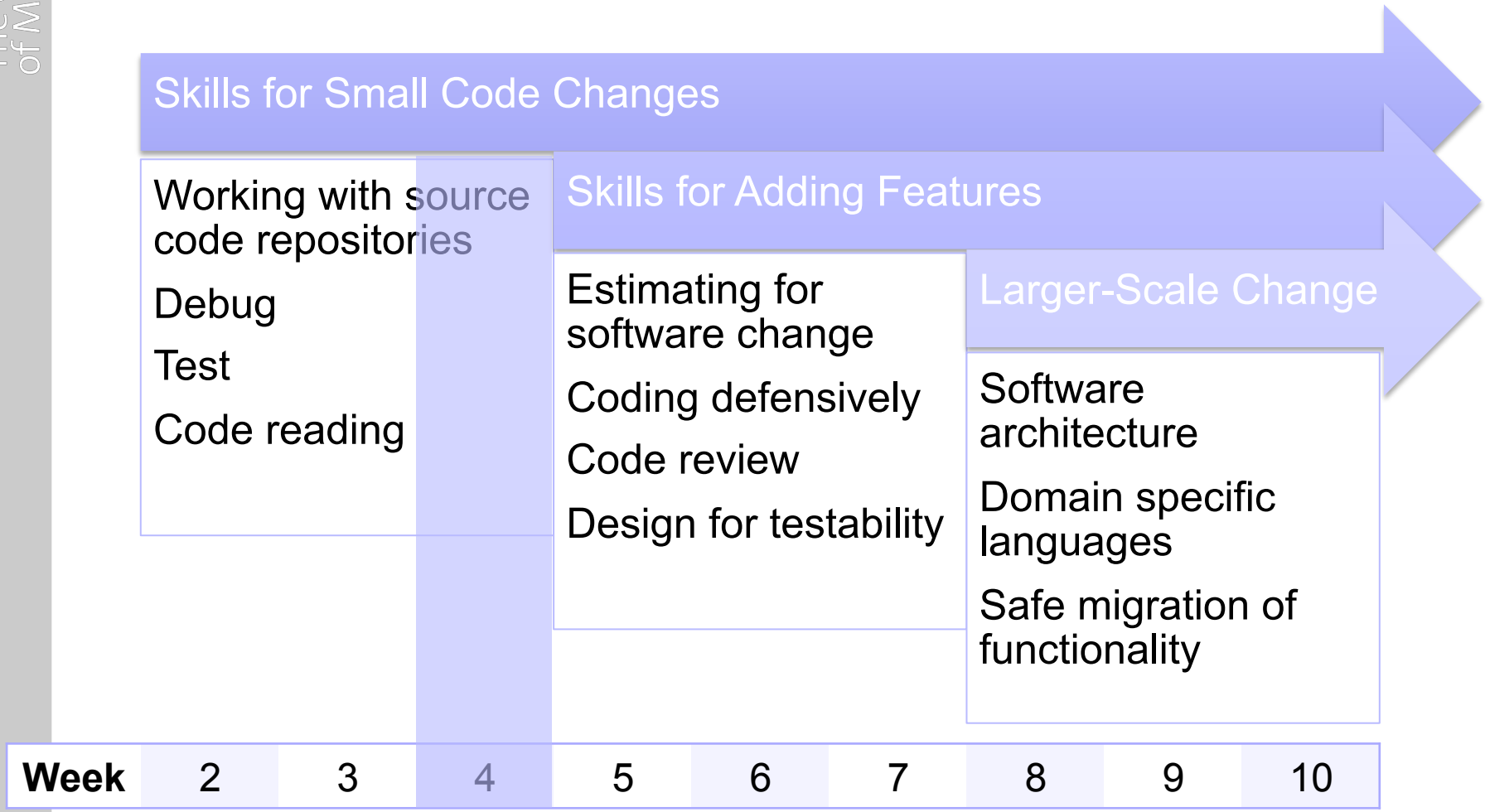
Buddy Review & Git workflows

COMP23420: Software Engineering

Week 4

Markel Vigo

Course Unit Roadmap (Weeks 2-10)



Coursework deadlines

- Deadline is Friday 5.00PM
- You are going to be marked based on the contents of the repo
- Marking will happen the next week
  - Face-to-face
  - With TAs
  - At the team study sessions
- Schedule on Moodle

Link to the Coursework/Exam

- We will learn basic Git workflows including
  - Branch creation
  - Branch merging
  - Conflict resolution
- On the command line, Eclipse and GitLab
- We will learn how important code reviews are
- How code reviews can be incorporated onto distributed version control systems and their workflows

Outline

Times are estimated

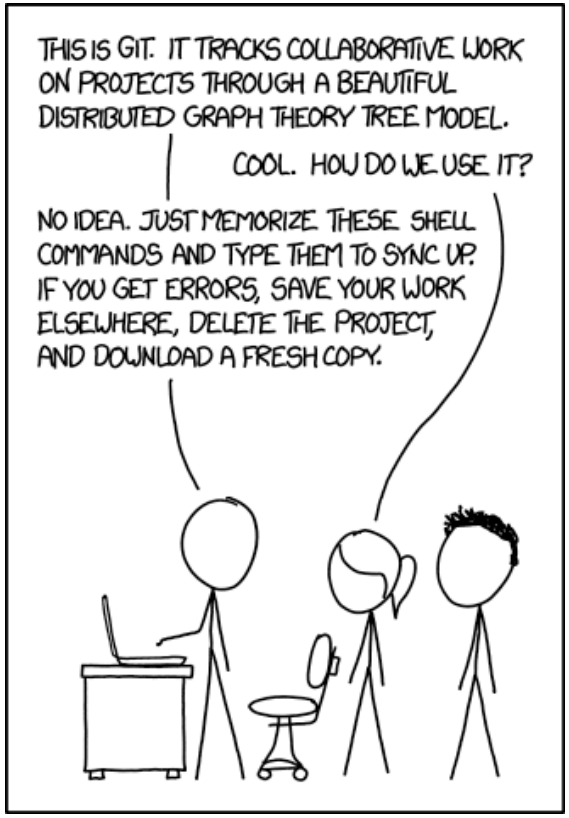
1. Motivation 5’
2. Git workflows 15’
3. Git workflows: branch creation 35’
4. Break? 10’
5. Code reviews 10’
6. Git workflows: branch merging 35’

Motivation

Motivation

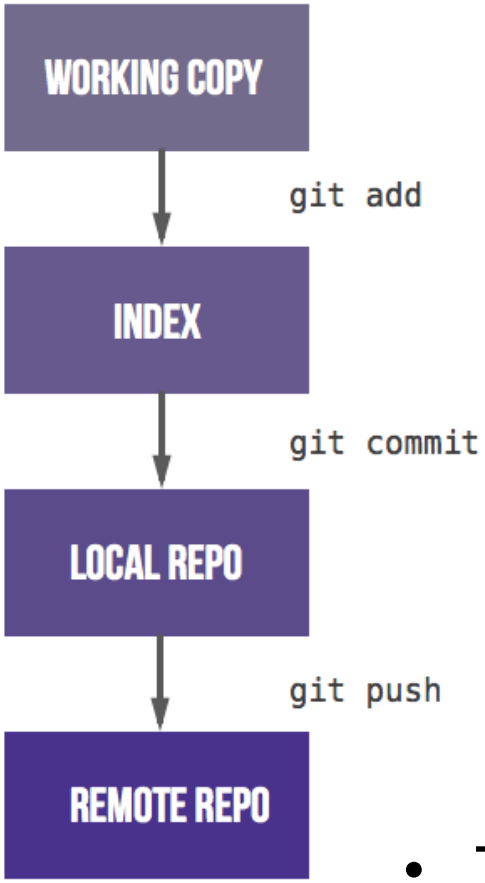
- Software engineering is not an individual activity
- Challenges:
  - Working with people
  - Working in distributed teams
  - Working with others' code
- Key challenges (and opportunities):
  - Group coordination
  - Codebase synchronisation
  - Quality assurance

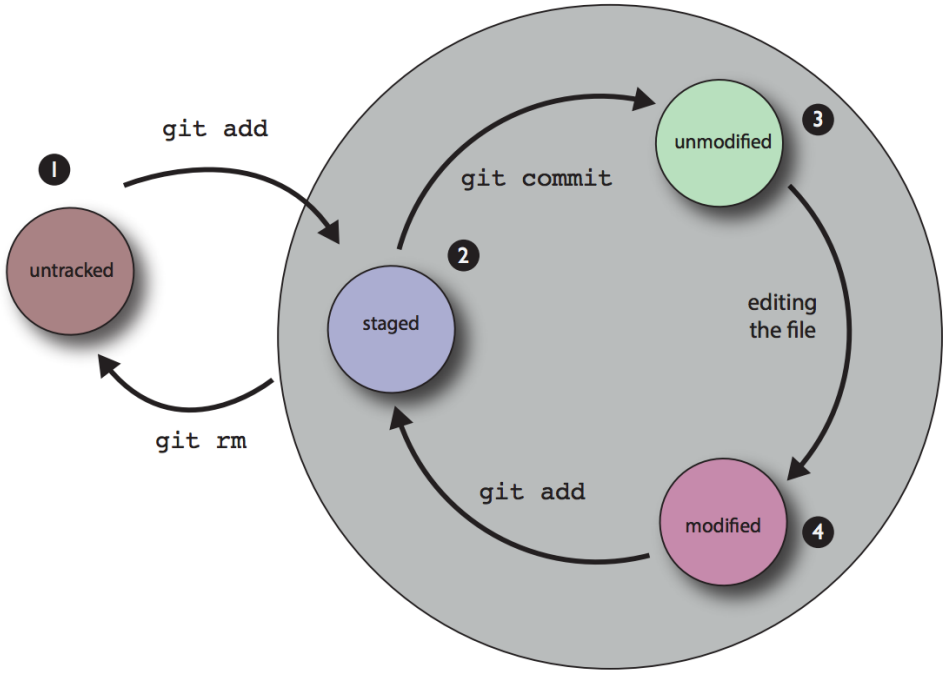
Git workflows



Basic Git: what you *should* know

- Git is a distributed version control system

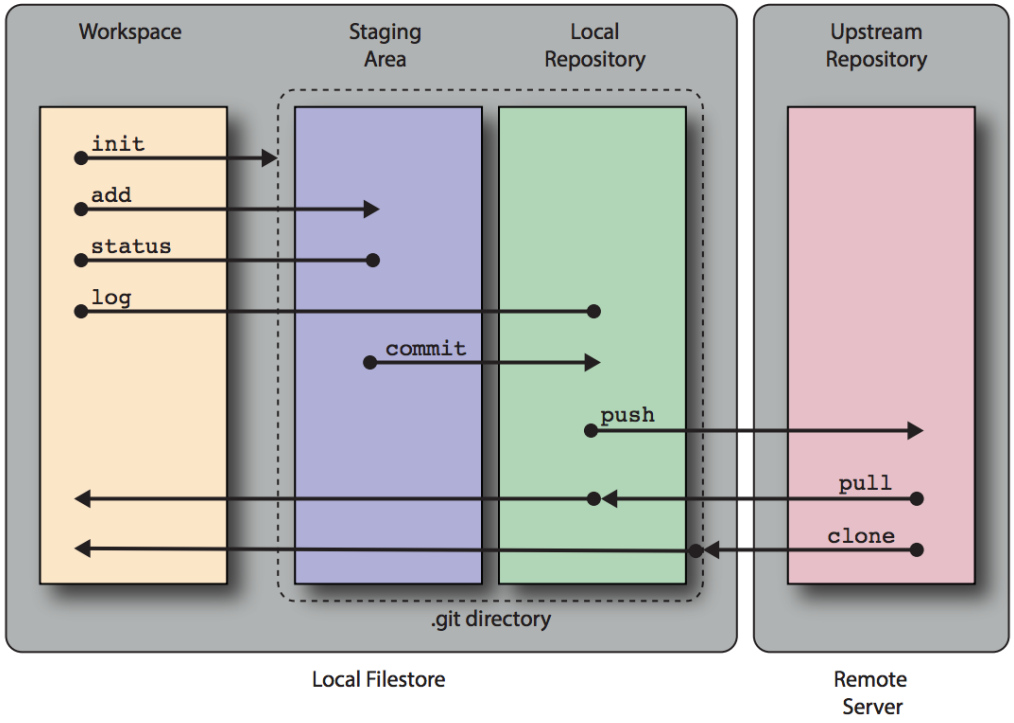




- The life cycle of a file under git's control

Git: what you *should* know

- Basic git commands and their interaction with various repositories

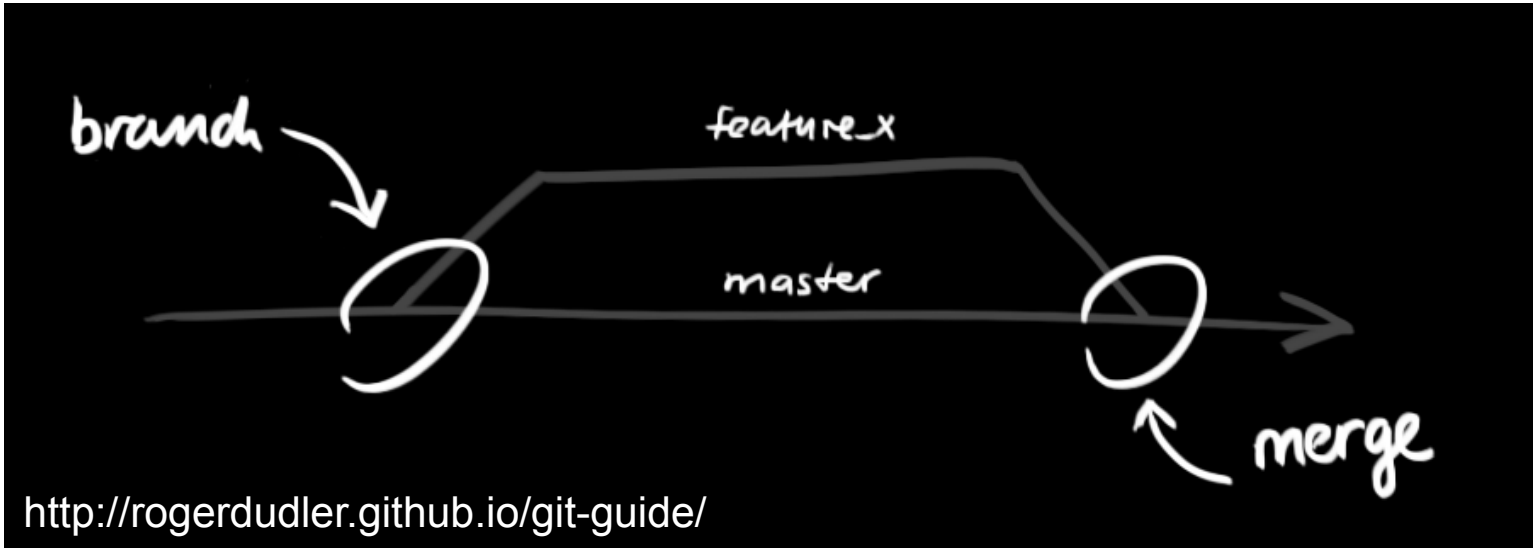


Git: best practice

- A `commit` should represent one conceptual change to your work
  - Expressed in one sentence
  - One `commit` per bug
- Do `commit` frequently
- Do not `commit` unfinished things
- Write **meaningful** messages

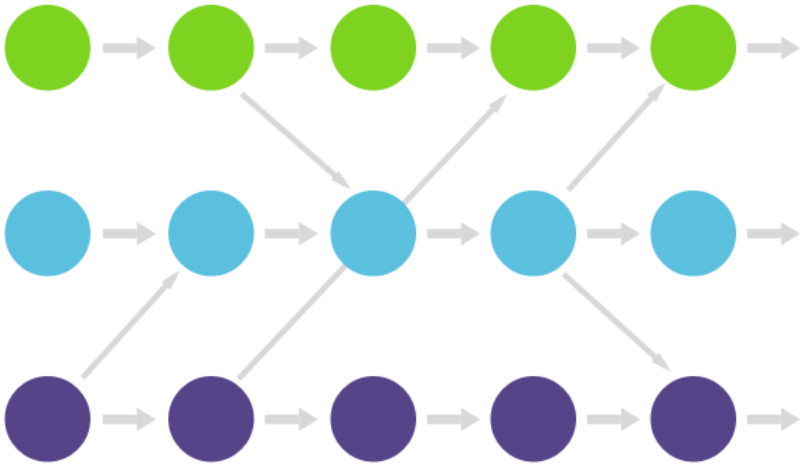
Branching

- Branches are used to develop features isolated from each other.
- The master branch is the “default” branch when you create a repository
- Use other branches for development and merge them back to the master branch upon completion.



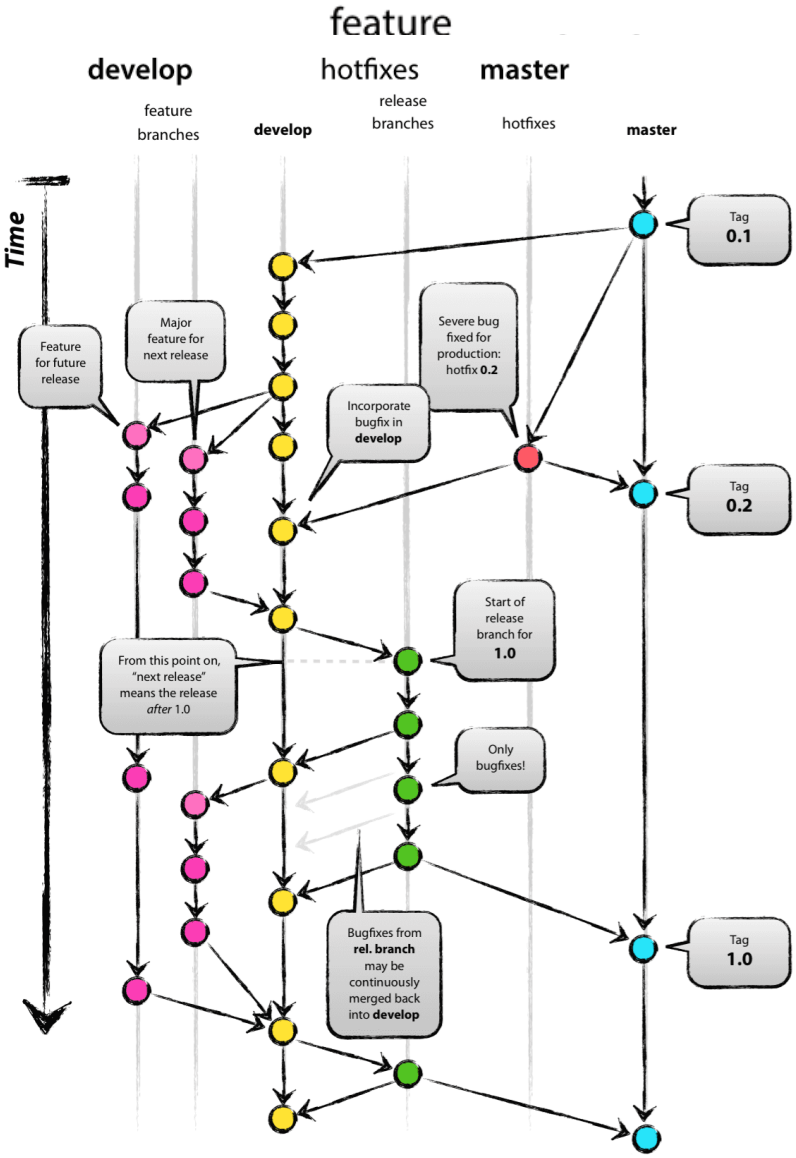
Git workflows

- Branching allows a wide variety of strategies
- This flexibility can result in complex, intertwined and messy ways of developing code
- Using Git stops being efficient
- A '**code of conduct**' or **protocol** or **conventions** are needed
- Patterns for Git use: Git workflows



Git workflows

- The **GitFlow** model
- Two main branches
  - *Master*: production code
  - *Develop*: latest development
- Supporting branches
  - Feature branches
  - Hotfix branches
  - Release branch
- Positive aspects
  - Popular workflow
- Negative aspects
  - Still complex!



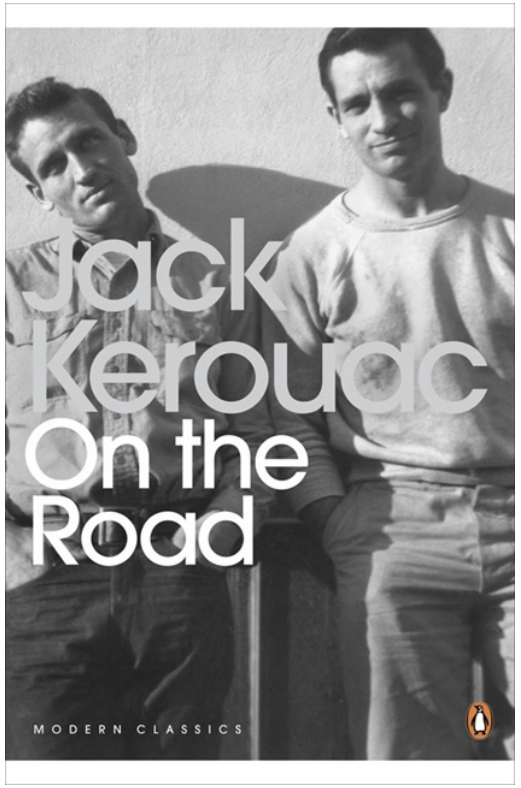
Git workflows

- The **GitHub Flow** model as a reaction
  - Master branch
  - Feature branches
- **GitHub Flow** assumes you are able to deploy every time you merge a feature branch
- Simplification and reduction of branch types
- **GitLab Flow** simplifies this even more
- Confusion: different repository managers use different terminology for merging!
  - Gitlab and Gitorious: merge request
  - GitHub and BitBucket: pull request

Practicing Git workflows: branching

- ‘On the Road’ by Jack Kerouac
- Job description: you are the publisher of the book and you have received the first chapter.

1. You will request different chapters to complete the book.
2. We will have the `master` branch for the title and chapter 1
3. We will **create** one branch for
   I. chapter 2 in the command line
   II. chapter 3 in GitLab
   III. chapter 4 in Eclipse



Break: 10 minutes

Code reviews

```
448
449         filter_mock = mock_sql.get_session().query().filter()
450         self.assertFalse(filter_mock.limit.called)
451         self.assertTrue(filter_mock.delete.called_once)
452
453     def test_flush_expired_tokens_batch_mysql(self):
454         # test mysql dialect, we don't need to test IBM DB SA separately, since
455         # other tests below test the differences between how they use the batch
456         # strategy
457         with mock.patch.object(token_sql, 'sql') as mock_sql:
458             mock_sql.get_session().query().filter().delete.return_value = 0
459             mock_sql.get_session().bind.dialect.name = 'mysql'
460             tok = token_sql.Token()
461             expiry_mock = mock.Mock()
462             ITERS = [1, 2, 3]
463             expiry_mock.return_value = iter(ITERS)
464             token_sql._expiry_range_batched = expiry_mock
465             tok.flush_expired_tokens()
466
467             # The expiry strategy is only invoked once, the other calls are via
468             # the yield return.
469             self.assertEqual(1, expiry_mock.call_count)
470             mock_delete = mock_sql.get_session().query().filter().delete
```

```
448
449         filter_mock = mock_sql.get_session().query().filter()
450         self.assertFalse(filter_mock.limit.called)
451         self.assertTrue(filter_mock.delete.called_once)
452
453     def test_flush_expired_tokens_batch_mysql(self):
454         # test mysql dialect, we don't need to test IBM DB SA separately, since
455         # other tests below test the differences between how they use the batch
456         # strategy
457         with mock.patch.object(token_sql, 'sql') as mock_sql:
458             mock_sql.session_for_write().__enter__(
459             ).query().filter().delete.return_value = 0
460
461             mock_sql.session_for_write().__enter__(
462             ).bind.dialect.name = 'mysql'
```

Steve Martinelli: not clear on these changes and why they were needed — Feb 7 6:35 PM
Morgan Fainberg: to identify this is a writer session and use the context manager corre... — 3:45 PM

```
463
464             tok = token_sql.Token()
465             expiry_mock = mock.Mock()
466             ITERS = [1, 2, 3]
467             expiry_mock.return_value = iter(ITERS)
468             token_sql._expiry_range_batched = expiry_mock
469
470             tok.flush_expired_tokens()
471
472             # The expiry strategy is only invoked once, the other calls are via
473             # the yield return.
474             self.assertEqual(1, expiry_mock.call_count)
475
476             mock_delete = mock_sql.session_for_write().__enter__(
477             ).query().filter().delete
478
```

Code reviews

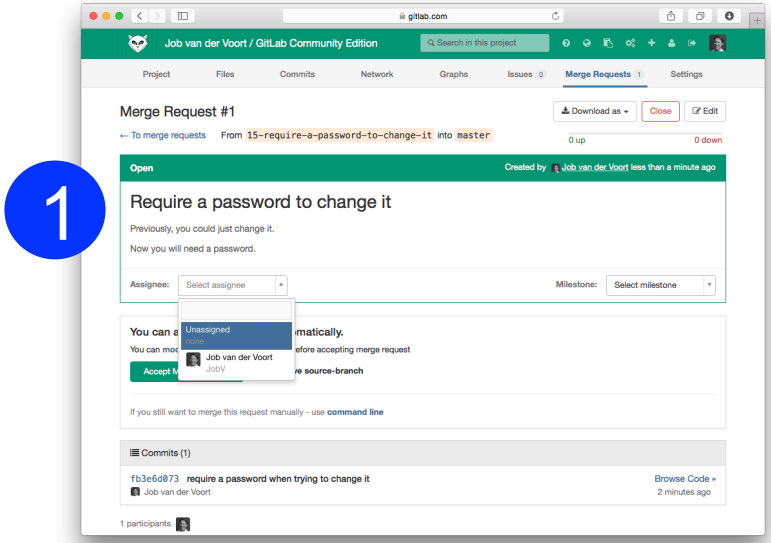
- Having the coder reviewed by somebody else is good
- It can be done in several ways:
  - Formal meeting with a projector and code is checked line by line
  - 'Over the shoulder'
  - Email
  - Pair programming
  - Tool based
- It improves the quality, readability and maintainability of software.
- It has an average of 60% of defect removal rate (even 85%) vs. 25% of unit testing

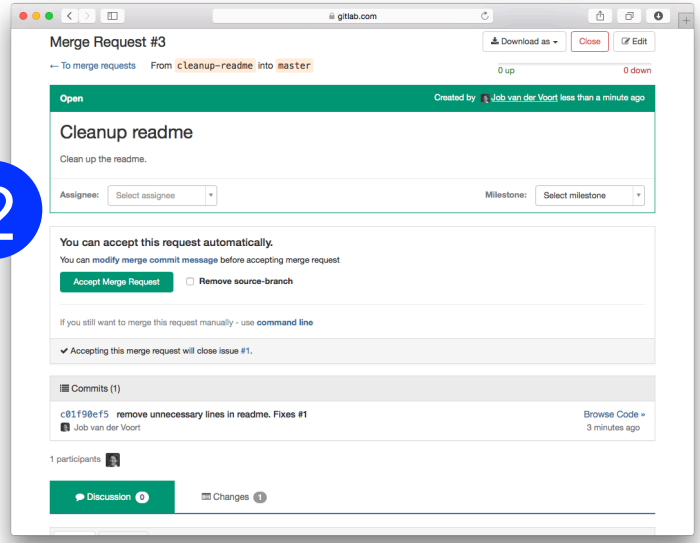
Code review: what to report

- Code review rates should be between 200 and 400 lines of code per hour
- What sort of issues?
  - Design issues
  - Possible errors
  - Coding style issues
  - Testing issues
  - Rewards: positive comments
- How to do the reviews?
  - Be nice
  - Be constructive (somebody else will look at yours!)
- It's good for your own soft skills

Code reviews in the GitLab workflows

- GitLab provides a online platform to discuss the code
- Code reviews on merge requests
  1. Assign the request to somebody else
  2. This person will verify(through code reviews)
     I. If happy with request they accept the merge
     II. If not they may require further work

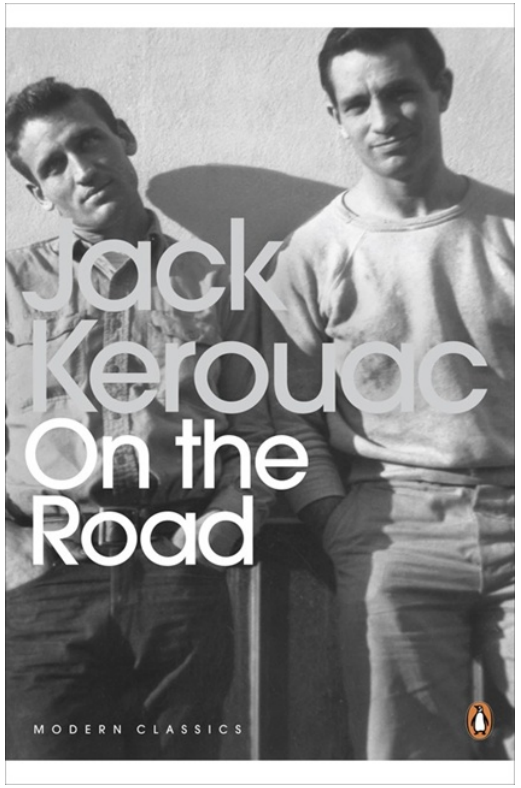




Practicing Git workflows: merging

- 'On the Road' by Jack Kerouac
- Job description: you are the publisher of the book and you have received the first chapter.

1. You will request different chapters to complete the book.
2. We will have the `master` branch for the title and chapter 1
3. We will **merge**
   I. chapter 2 from GitLab
   II. chapter 3 from the command line
   III. chapter 4 from Eclipse



Next Week

- In the team study sessions you will work on the coursework

- In the workshop we will learn to estimate the cost of bug fixing and adding new features