Implementation

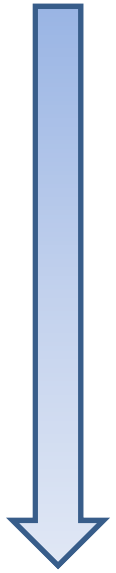
Computing & Information Sciences

W. H. Bell

Software development lifecycle

One iteration

- Requirements definition.
- Software and systems design.
- **<u>Implementation and unit testing.</u>**
- Integration and system testing.
- Operation and maintenance.



Types of development

- New build.
  - Structure not completely defined.
- Steady state development.
  - Well understood.

New build

- Build prototypes.
  - Understand any issues in architecture choices.
- Start from something similar.
  - Use experience from existing software development.
- Define interfaces.
  - Difficult until framework becomes established.
  - Small number of initial developers.

Steady state

- Adding features to all layers of an application.
- Well-defined interfaces.
- Stable architecture and associated test framework.
  - Refined and re-optimised program structure.
- Easier to include more developers.
  - Need more control over developer changes.
  - Need a code review process.

Modular approach

- Express implementation as a set of modules.
  - A module is a set of functions and classes.
  - Define APIs between modules.
  - Follow existing low-level design documentation.
- Continue factorise implementation into modules.
  - Coupling between modules allows testing to occur.

Modular approach

- Improve code reusability.
- Allow developers to work in parallel.
- Easier to debug and test.

Feature-driven development

- Use for steady state agile development.
- Implement a vertical slice of functionality.
  - Update many layers/modules within the application.
- Difficult to apply when there is no code skeleton.

Design updates

- Implementing software causes evaluation of design.
  - Code describes design.
  - Realise that the design needs to be updated.
    - Expect small changes.
    - Functions or interactions could be different.
- Redesign part of development process.
  - Constrained by cost.
  - Referred to as technical debt in steady-state development.
    - Apply refactoring as needed.

Refactoring

- Initial code designed to fulfil requirements.
  - May have limited expandability.
- Continuous development may cause:
  - Bigger classes or functions.
  - Code duplication.

Refactoring

- Simplify functions and classes.
- Look for patterns and factorise out functions or classes.
- Optimise performance – improved use of syntax.
- Use IDE refactoring tools.

https://code.visualstudio.com/docs/editor/refactoring

Coding for others

- Clearly written classes, functions and variable names.
  - Efficient and readable logic.
  - Shorter functions with clear purpose.
  - Thinking about testing units of software.
- Succinct comments that are informative.
  - State purpose of code.
  - Kept up to date with respect to code.

Coding for others

- Well-organised directory and file structure.
  - Limit number of lines of code in file.
  - Limit number of files in directory.
- Lightweight maintenance documentation.
  - Automatically generated reference documentation.
- Architecture should be obvious from implementation.

Development approaches

Development and testing

- Software must pass tests.
  - Unit tests – verification of API calls.
  - Acceptance tests – top-level features.
- Develop and then write tests.
  - Design ideas may evolve during development.
- Write tests and then develop – test-driven development.
  - More time consuming to re-work tests if tests are wrong.

Test-driven development

- Describe functionality using test framework.
  - Unit tests against API definition.
  - Use same language or alternative language.
- Build software to pass tests.
  - Can fulfil tests with many implementation approaches.
  - Long-term resilience to software obsolescence.

Generating documentation

Generating documentation

- Use code structure.
  - Find classes, functions and types.
  - Follow inheritance.
- Use comments.
  - Rely on special comment formatting.
  - Comment states intention, inputs and return values.

Generating documentation

- Doxygen
  - C, Objective-C, C#, PHP, Java, Python, Fortran.
  - https://www.doxygen.nl/
- Javadoc
  - Java.
  - https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html
- Dartdoc
  - Dart (Flutter)
  - https://pub.dev/packages/dartdoc
- Pydoc
  - Python
  - https://docs.python.org/3/library/pydoc.html
- Godoc
  - Golang
  - https://tip.golang.org/doc/comment

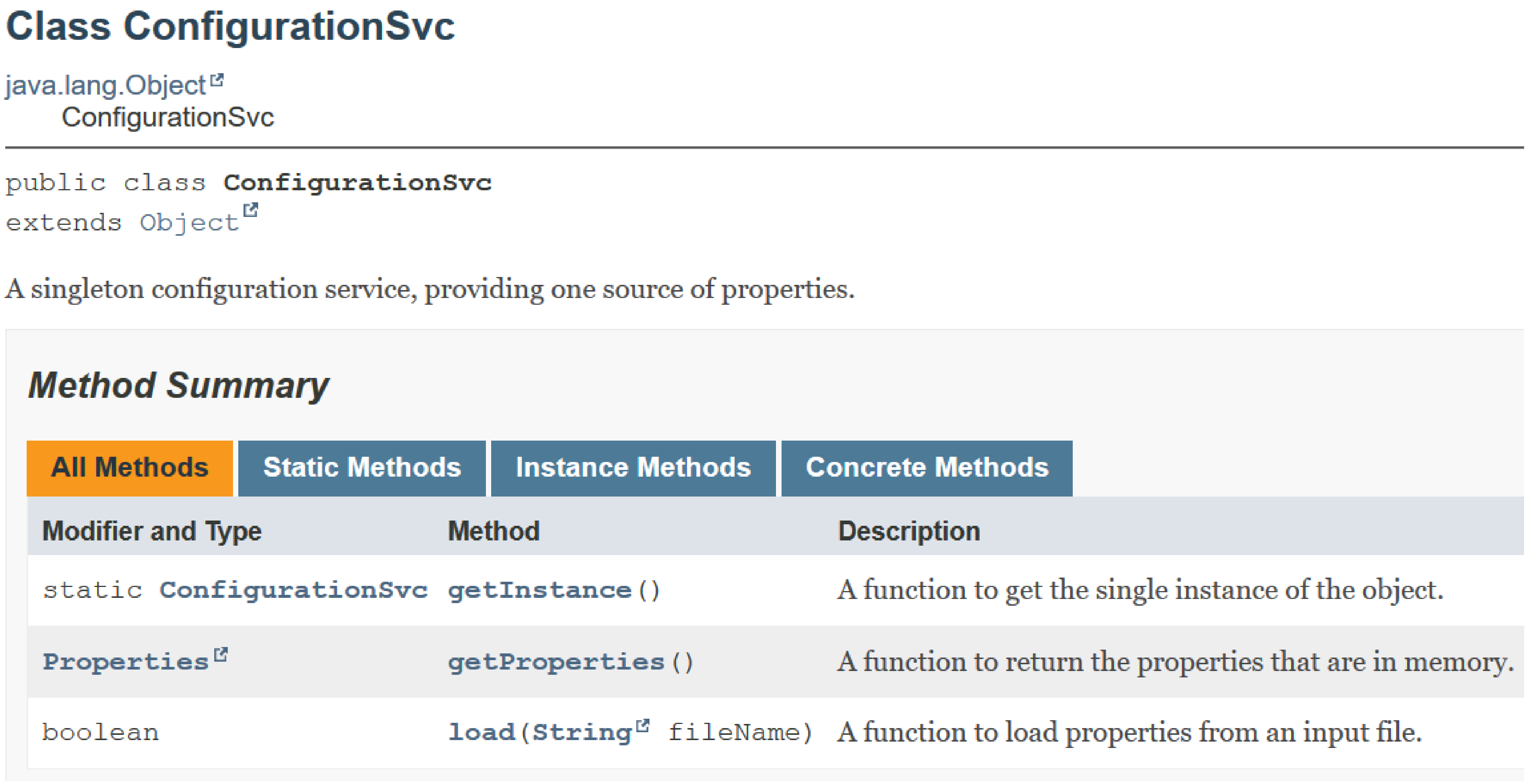
Javadoc

Follow commenting recipe, stating what input and return values are.

```
/**
 * A singleton configuration service, providing one source of properties.
 */
public class ConfigurationSvc {

    ...

    /**
     * A function to load properties from an input file.
     * @param fileName  name of the file that contains the properties.
     * @return  true if the file is successfully loaded.
     */
    public boolean load(String fileName) {
```

Javadoc

Generates HTML pages from code and comments.



Class ConfigurationSvc

java.lang.Object
    ConfigurationSvc

```
public class ConfigurationSvc
extends Object
```

A singleton configuration service, providing one source of properties.

Method Summary

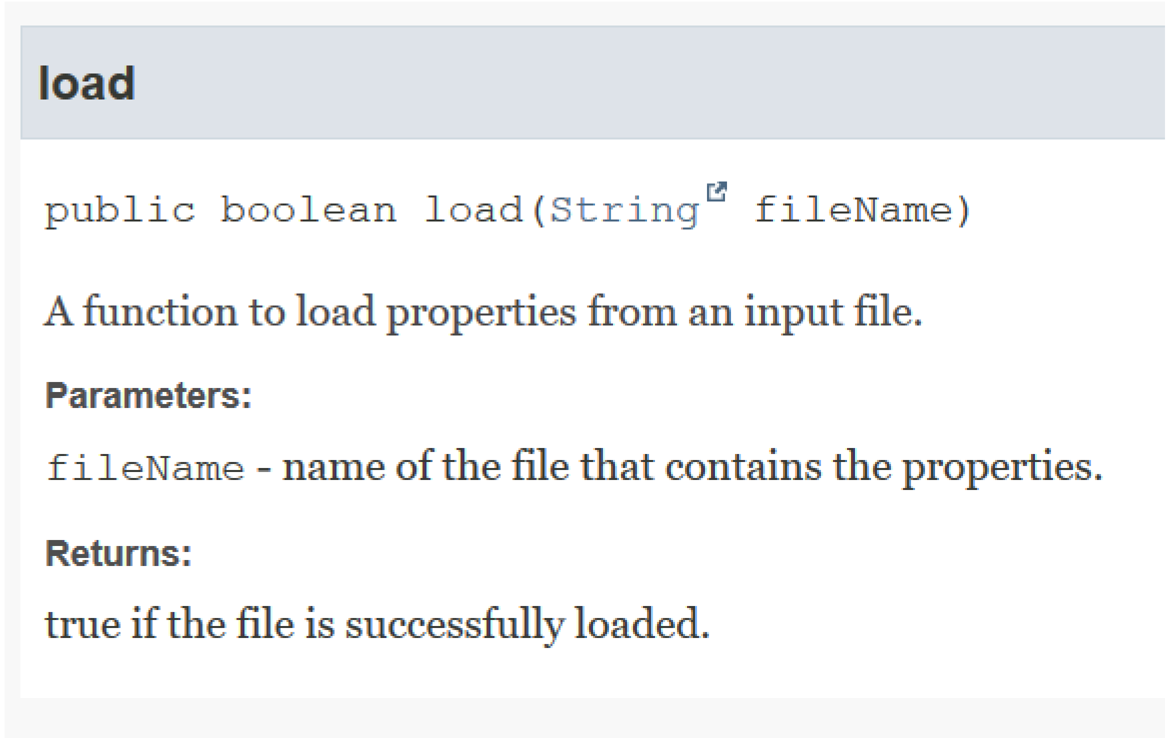
All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
`static ConfigurationSvc`	`getInstance()`	A function to get the single instance of the object.
`Properties`	`getProperties()`	A function to return the properties that are in memory.
`boolean`	`load(String fileName)`	A function to load properties from an input file.

Javadoc

Generates HTML pages from code and comments.

```
/**
 * A function to load properties from an input file.
 * @param fileName  name of the file that contains the properties.
 * @return  true if the file is successfully loaded.
 */
public boolean load(String fileName) {
```



load

```
public boolean load(String fileName)
```

A function to load properties from an input file.

Parameters:

`fileName` - name of the file that contains the properties.

Returns:

true if the file is successfully loaded.

Pydoc

- Generates online help.
  - Manual pages.
    - Manual page for sys module:

```
python -m pydoc sys
```

  - HTML pages.
    - Running web server on local host:

```
python -m pydoc -p 7000
```

https://docs.python.org/3/library/pydoc.html

Pydoc

Many docstring formatting standards exist.
Using Google style:

```python
def factorial(x: int) -> int:
    """
    Calculate the factorial of an input integer.

    Args:
        x (int): An input value.

    Returns:
        int: The factorial of the input integer.
    """
    result = 1
    while x > 1:
        result *= x
        x -= 1
    return result
```

https://google.github.io/styleguide/pyguide.html

Pydoc

```
$ python -m pydoc algorithm
Help on module algorithm:

NAME
    algorithm

FUNCTIONS
    factorial(x: int) -> int
        Calculate the factorial of an input integer.

        Parameters:
            x (int): An input value.

        Returns:
            int: The factorial of the input integer.

FILE
    c:\users\xxyb1234\algorithm.py
```

https://google.github.io/styleguide/pyguide.html

Software repositories

Tracking changes

- Need to verify what was changed.
    - Associate with code modification comments.
    - Many layers of changes – one file many changes.
- Associate feature implementation with changes.
    - Update several files.

Code changes

Initial version

```python
def factorial(x: int) -> int:
    return x


if __name__ == "__main__":
    print(factorial(3))
```

algorithm_org.py

Updated version

```python
def factorial(x: int) -> int:
    result = 1
    while x > 1:
        result *= x
        x -= 1
    return result


if __name__ == "__main__":
    print(factorial(3))
```

algorithm_1.py

Code differences

```
diff -Naur algorithm_org.py algorithm_1.py
```

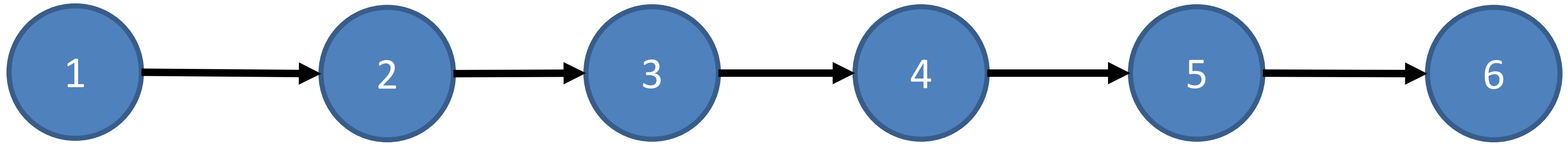
```
--- algorithm_org.py    2023-02-24 21:47:17.176878500 +0000
+++ algorithm_1.py  2023-02-24 21:47:07.664964100 +0000
@@ -1,5 +1,9 @@
 def factorial(x: int) -> int:
-    return x
+    result = 1
+    while x > 1:
+        result *= x
+        x -= 1
+    return result


 if __name__ == "__main__":
```

Storing many changes

- Store changes in repository.
  - Includes files and differences.
- Many repository solutions exist.
  - Concurrent Versions System (CVS) – 1990 onwards.
  - Subversion (SVN) – 2000 onwards.
  - Git – 2005 onwards.
    - Originally written to manage Linux kernel source code.

Storing many changes

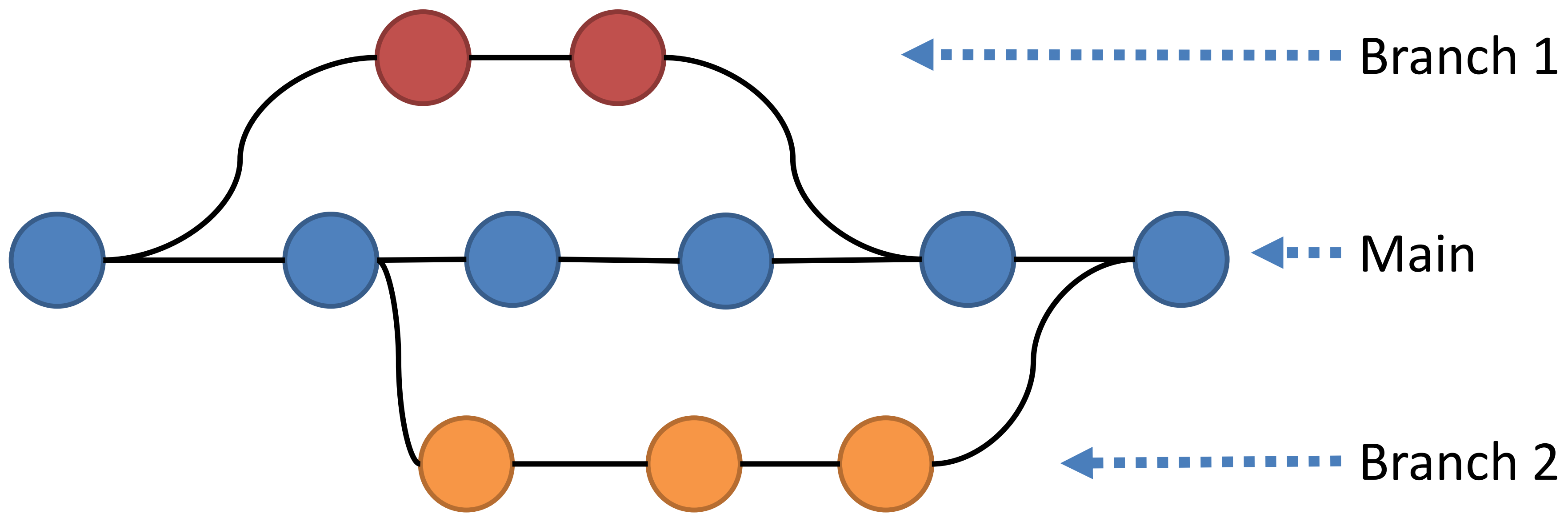
- Commit set of file changes to repository.
  - Track file versions and commit points.
  - Files change over time.
- Allows single developer to track changes.
  - More difficult to include parallel development.
  - May brake working version in repository.



Branches

- Allow parallel development branches.
  - Split off from the main development thread of versions.
  - Merge back into the main branch on task completion.
- Allows other developers to work in parallel.
  - Merging back into the main branch can require work.
  - Need automated testing and code review before merge.
  - Must avoid breaking the main branch.

Branches



Previous repositories (SVN, CVS)

- Previous repositories keep differences in server.
  - Client checks out a live copy.
  - Updating to another branch requires server interaction.
  - Changes are committed back to server.
- Server becomes a bottleneck to development.
  - Cannot work if server is too busy.
  - Cannot commit when not connected to the Internet.

Git

- Clone repository to local file system.
  - Origin – remote Git repository, probably hosted in server.
    - Can change between origins or have no origin.
  - Local repository – version information in `.git/` directory.
- Work with local repository.
  - Commit, branch and change between branches.
    - Internet access not needed.
  - Push changes back to origin.

https://git-scm.com/doc

Git workflow: single origin

Application: commercial projects.

- Clone repository.
- Create a branch or checkout existing branch.
- Commit changes to branch.
- Push changes to origin.
- Raise a pull request.
- Code review and approval needed for merge.

Git workflow: forked origin

Application: open source development.

- Fork main repository into another remote repository.
- Clone forked repository.
- Create branch or checkout existing branch.
- Commit changes.
- Push changes back to forked origin.
- Raise pull request from forked repository to original.
- Code review and approval needed for merge.
- Update forked repository.

Git servers

- GitHub
  - https://github.com/
- Gitlab
  - https://about.gitlab.com/
- Bitbucket
  - https://bitbucket.org/product
- AWS CodeCommit
  - https://aws.amazon.com/codecommit/
- Microsoft Azure DevOps.
  - https://azure.microsoft.com/en-us/products/devops
- SourceForge.
  - https://sourceforge.net/

Git pull request review

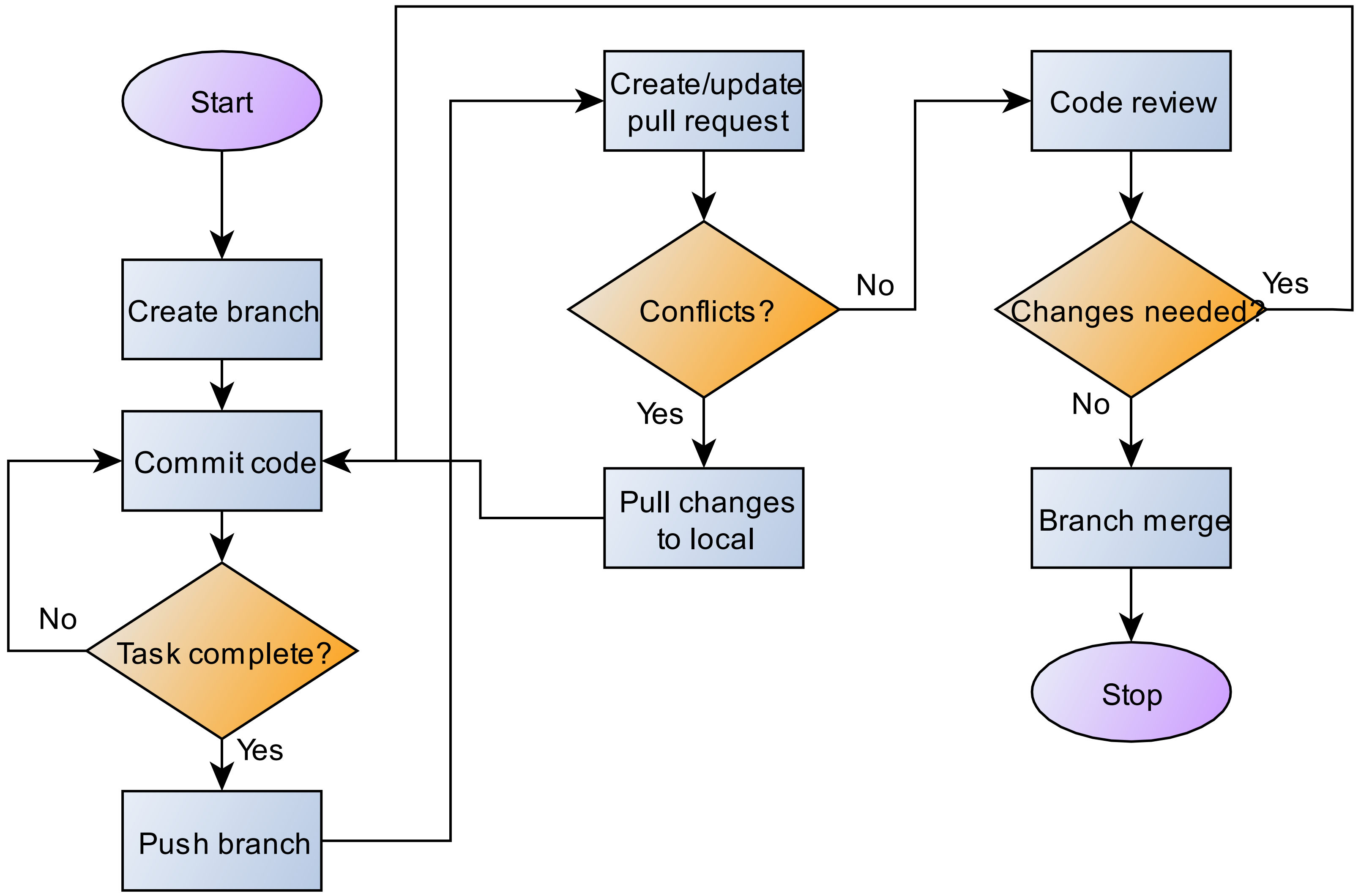
- Review code changes.
  - Understand if changes are safe.
  - Main may have changed during branch development.

https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/reviewing-changes-in-pull-requests/reviewing-proposed-changes-in-a-pull-request

Git pull request review

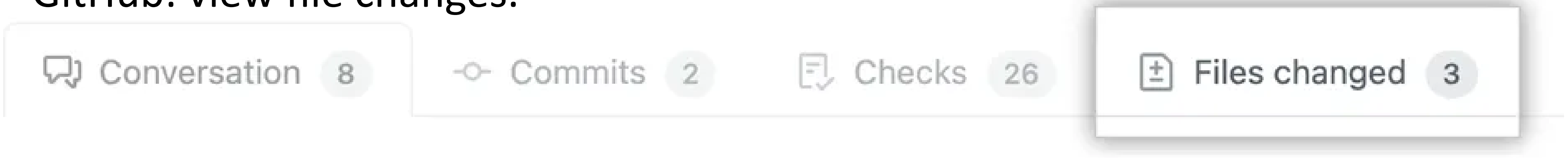
- Main branch may have changed during development.
  - Resolve conflicts.
- Review code changes.
  - Understand if changes are safe.

Git pull request review

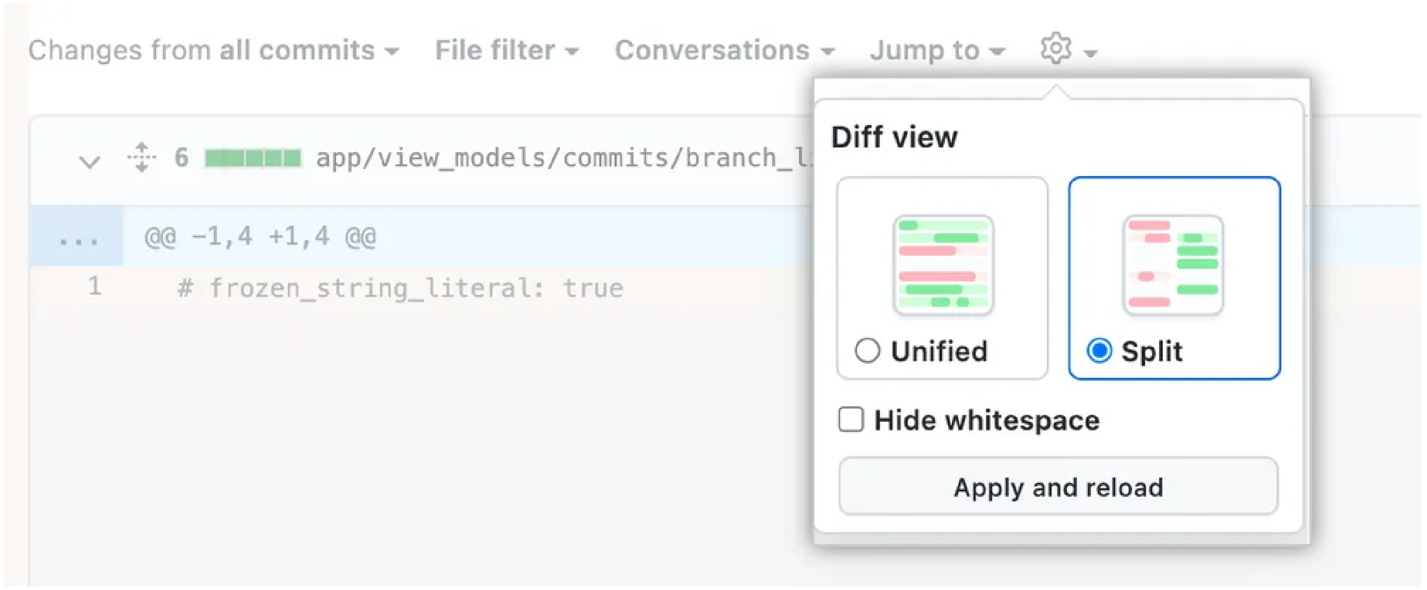


Git pull request review

GitHub: view file changes.



GitHub: selecting difference view.



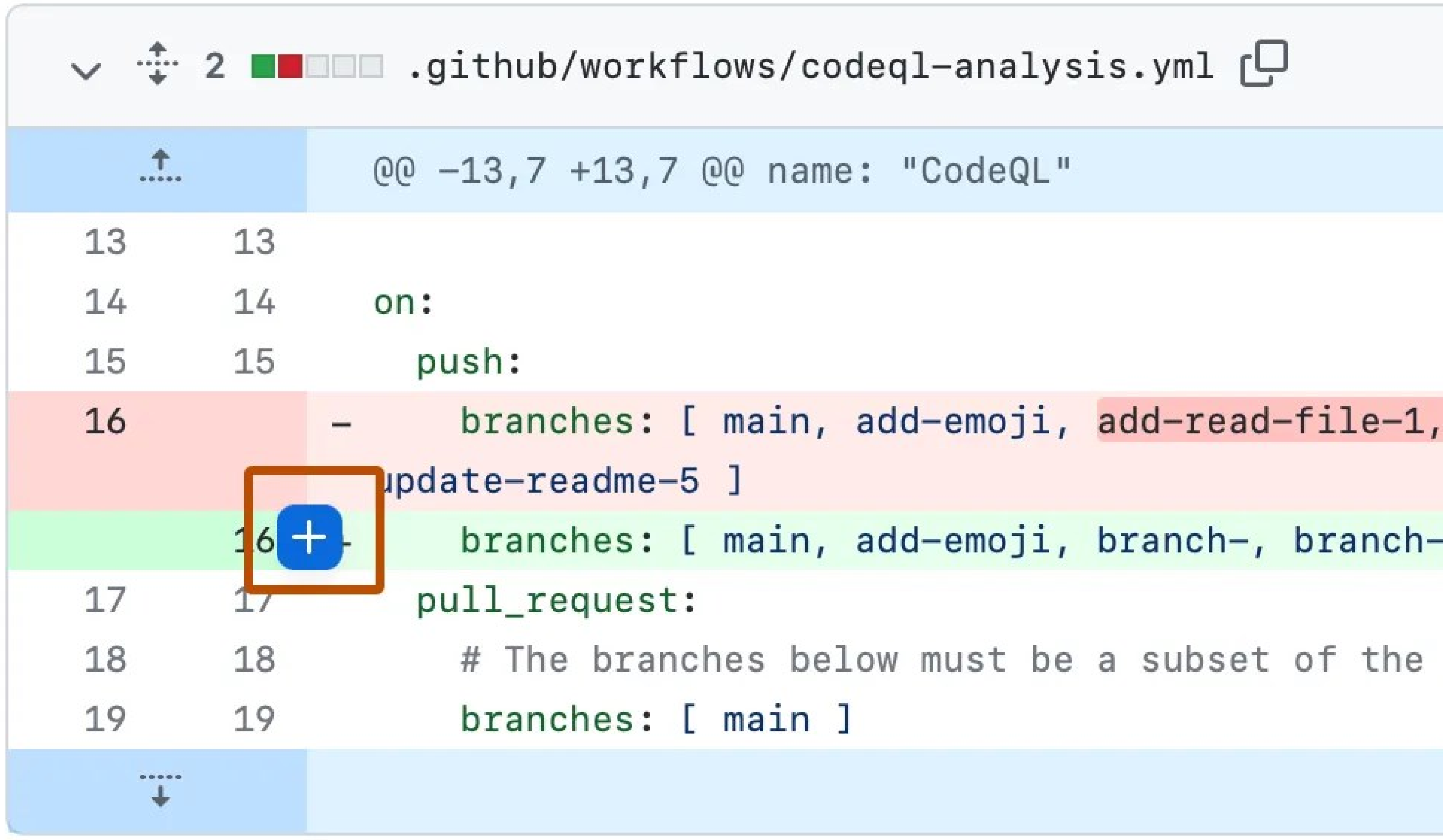
```
6 app/view_models/commits/branch_l
@@ -1,4 +1,4 @@
1   # frozen_string_literal: true
```

Diff view: Unified [✓ Split]; Hide whitespace ☐

https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/reviewing-changes-in-pull-requests/reviewing-proposed-changes-in-a-pull-request

Git pull request review

GitHub: Add comment concerning change.

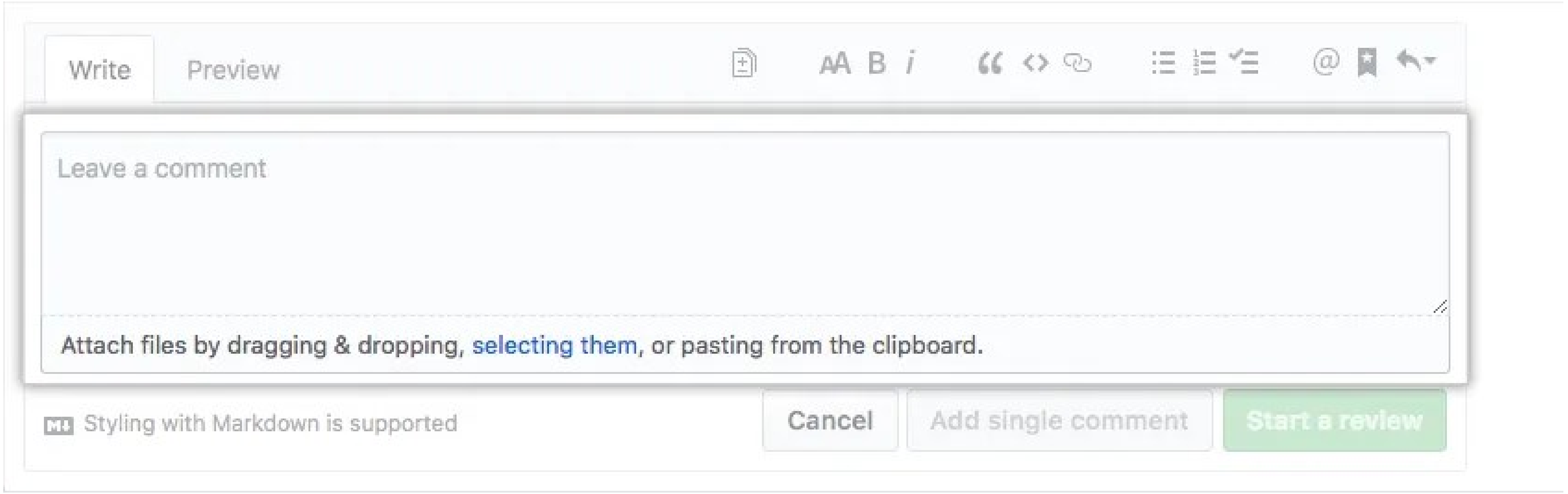


```
@@ -13,7 +13,7 @@ name: "CodeQL"

on:
  push:
-   branches: [ main, add-emoji, add-read-file-1,
update-readme-5 ]
+   branches: [ main, add-emoji, branch-, branch-
  pull_request:
    # The branches below must be a subset of the
    branches: [ main ]
```

https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/reviewing-changes-in-pull-requests/reviewing-proposed-changes-in-a-pull-request

Git pull request review

GitHub: Add comments concerning changes.



Leave a comment

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Styling with Markdown is supported

https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/reviewing-changes-in-pull-requests/reviewing-proposed-changes-in-a-pull-request

Conclusions

- Design/redesign continues during implementation.
  - Implement software as modules and features.
- Need to implement well-written, clear code.
  - Doc string comments and automatic documentation generation.
- Version control is vital, especially when working in teams.

University of Strathclyde Glasgow