CrossDoc

Team: Octo-Docs

Team Members:
Garrison Smith
Peter Huettl
Kristopher Moore
Brian Saganey

Client/Mentor

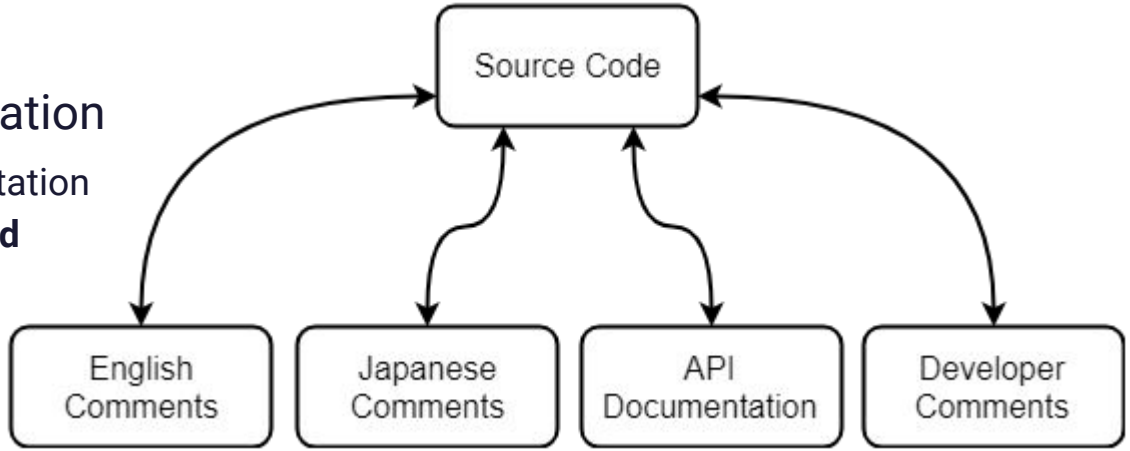
- Dr. James Palmer
  - Associate Professor at NAU - SICCS
- Dr. John Georgas
  - Associate Professor at NAU - SICCS
- Nakai McAddis
  - Lecturer at NAU

NORTHERN ARIZONA UNIVERSITY

Problem Statement

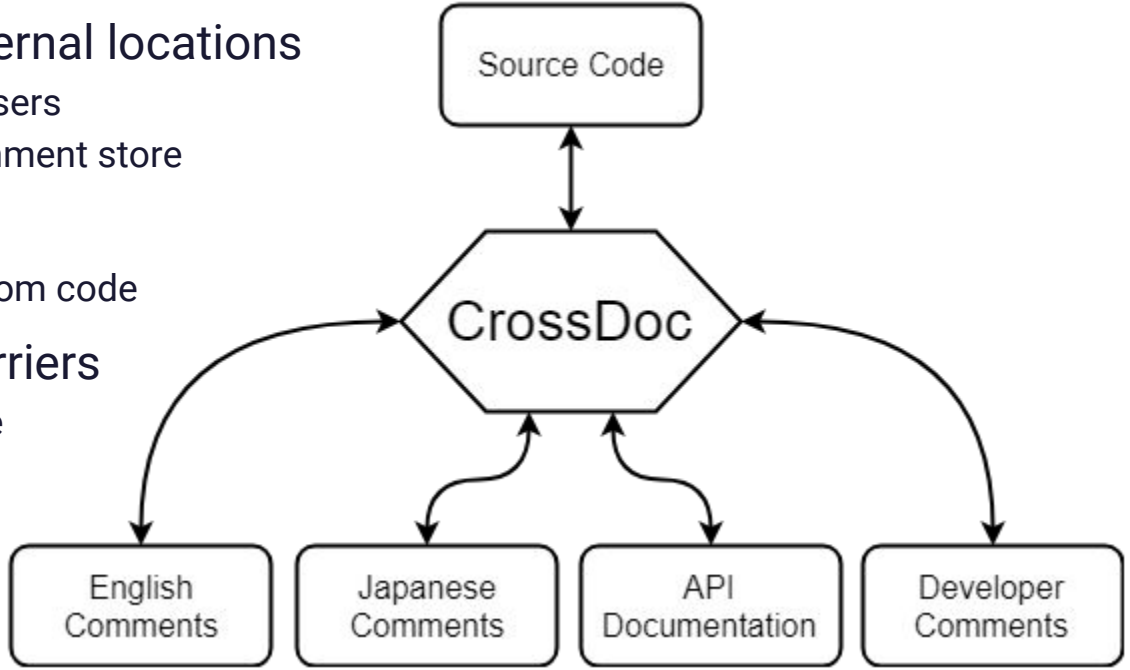
The Problem

- Large companies with large projects
  - Culturally diverse developers
  - Language barrier
- Software and Documentation
  - Misunderstood documentation
  - Comments **tightly coupled** with the codebase



The Solution: CrossDoc

- Comments stored in external locations
  - Easily accessible for all users
  - Editable in code or in comment store
- Scales alongside teams
  - Expands independently from code
- Breaks down cultural barriers
  - Easily store and reference comments in different languages



Problem Visualized

- Documentation is buried and too reliant on the codebase
- Jumbled comments with excess information

```
10  # The Logger class is responsible for providing output to the console
11  #
12  # API documentation
13  # standard(message)
14  #   logs `message` to stdout
15  #
16  # usage(command=None)
17  #   logs the usage message for the command that calls this method
18  #   alternatively, logs the usage message for the given `command`
19  #
20  # program(message)
21  #   logs `message` to stdout prefixed with the program name
22  #
23  # fatal(message)
24  #   logs `message` to stdout prefixed with "fatal" and kills program
25  #
26  # Things to do
27  # * Modify where fatal logs its message (stdout -> stderr)
28  # * Add a warning logging method that prefixes messages with "warning"
29  class Logger:
30
31    def standard(message):
```

Solution Visualized

- Provide a better way to comment with CrossDoc!

```
31  # <&> 20807c [No Set]
32  # The Logger class is responsible for providing output to the console
33  class Logger:
```

- Scalable, external storage, and enhanced comment functionalities.

```
19  # <&> 20807c [TODO]
20  # * Modify where fatal logs its message (stdout -> stderr)
21  # * Add a warning logging method that prefixes messages with "warning"
22  class Logger:
```

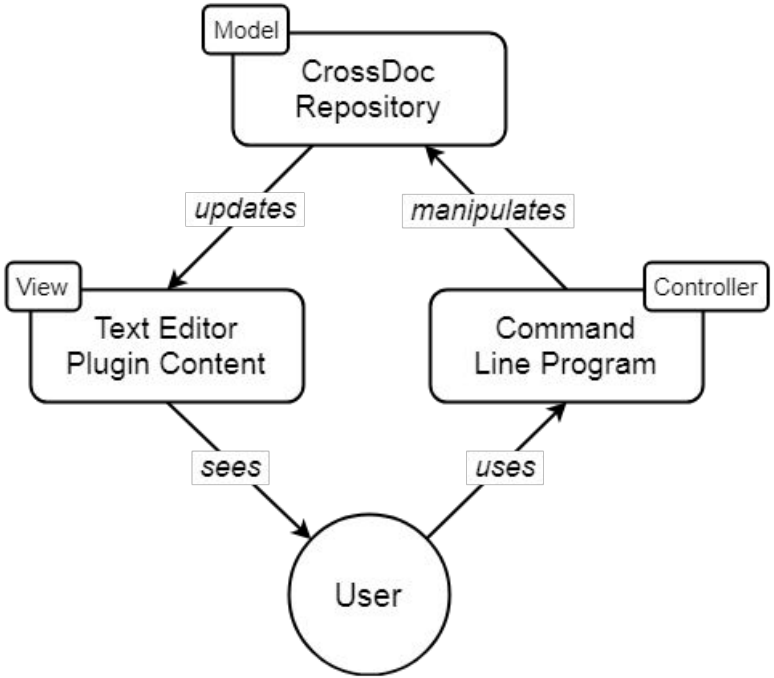
CrossDoc Key Requirements

- Simple setup process
- External comment storage
- Intuitive comment editing
- Functional text-editor plugins
  - Atom
  - Emacs
  - Sublime
  - Vim

Architecture and Implementation

High Level Overview

- MVC Architecture
  - Model: CrossDoc Repository
  - View: Text Editor Plugin Content
  - Controller: Command Line Program
- Frameworks/Tools
  - Python setuptools
  - Text editor APIs
  - MediaWiki API



Command Line Program

- Provides API to interact with tool
- Text editor agnostic
- Implements core functionality
  - Create comments
  - Read comments
  - Delete comments
  - Etc..

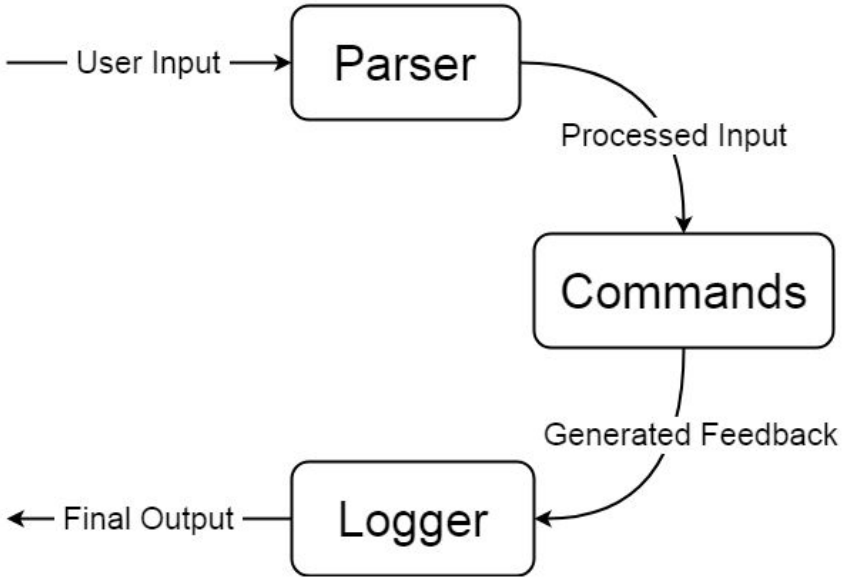
```
λ cross-doc --help
usage: cross-doc <command>

All CrossDoc commands:

  init
  create-store
  create-comment
  generate-anchor
  fetch-comment
  delete-comment
  update-comment
  hide-comments
```

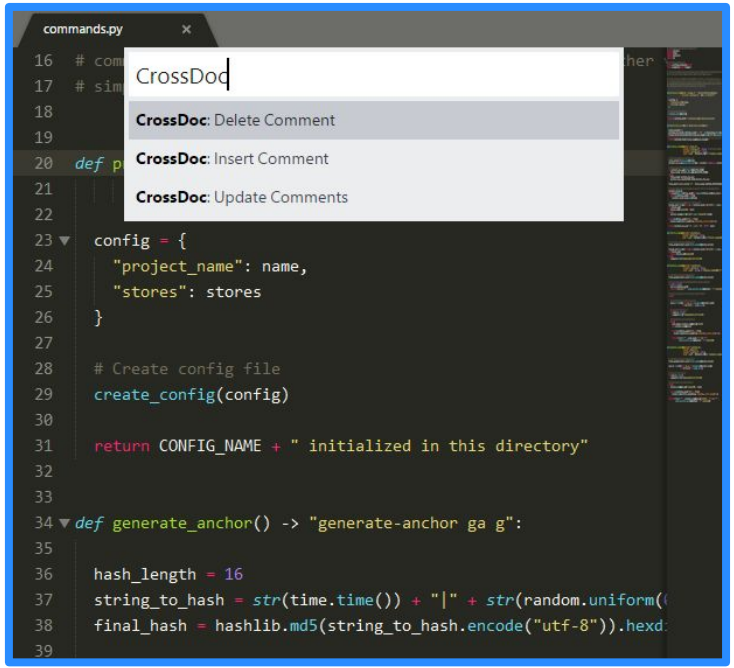
Command Line Program

- Parser
  - Reads input
  - Delegates to commands
- Commands
  - Implements CrossDoc functionality
- Logger
  - Provides concise output
  - Outputs help text where necessary



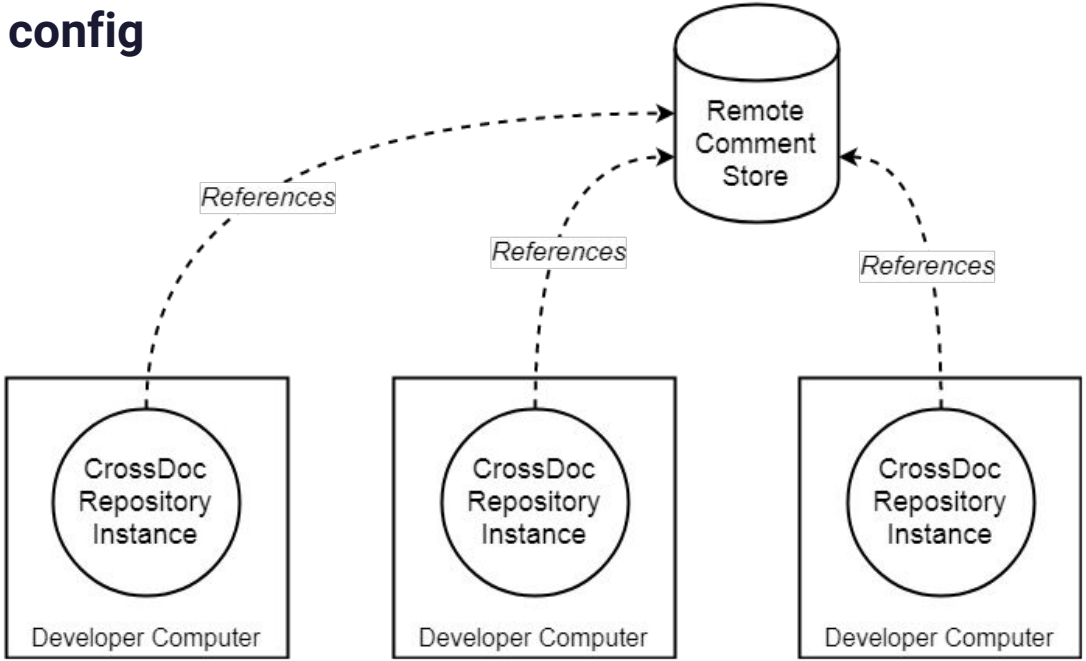
Text Editor Plugins

- CrossDoc user interface
- Intuitive commands and hotkeys
- Support for multiple text editors
  - Atom
  - Emacs
  - Sublime
  - Vim



CrossDoc Repository

- Identified by a **custom config** file *(cdoc-config.json)*
- Stores **references** to comment stores
- Persistent **meta-data** storage



Prototype Review

External Comment Storage

&20807c Logger Main

Contents [hide]

No Set [edit]

The Logger class is responsible for providing output to the console

Documentation [edit]

standard(message)

```
logs `message` to stdout
```

usage(command=None)

```
logs the usage message for the command that calls this method
alternatively, logs the usage message for the given `command`
```

program(message)

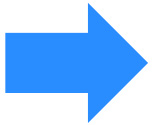
```
logs `message` to stdout prefixed with the program name
```

fatal(message)

```
logs `message` to stderr prefixed with fatal and kills program
```

TODO [edit]

- Modify where fatal logs its message (stdout -> stderr)
- Add a warning logging method that prefixes messages with "warning"

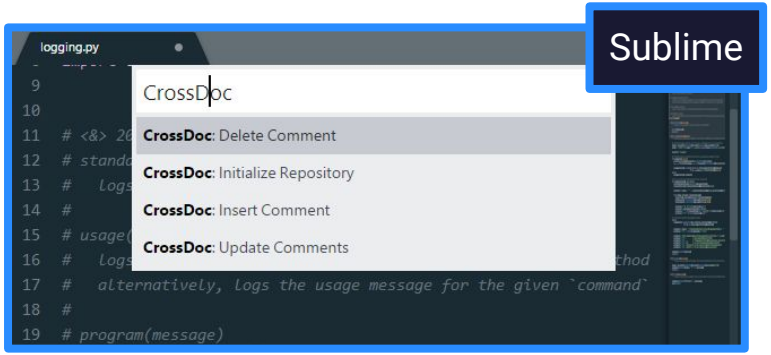


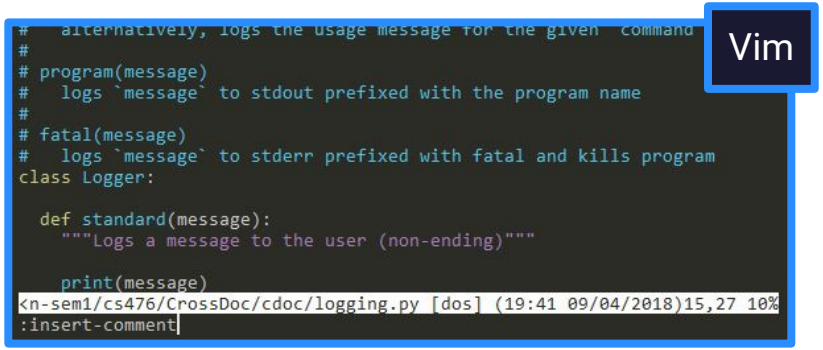
```
31  # <&> 20807c [No Set]
32  # The Logger class is responsible for providing output to the console
33  class Logger:
```

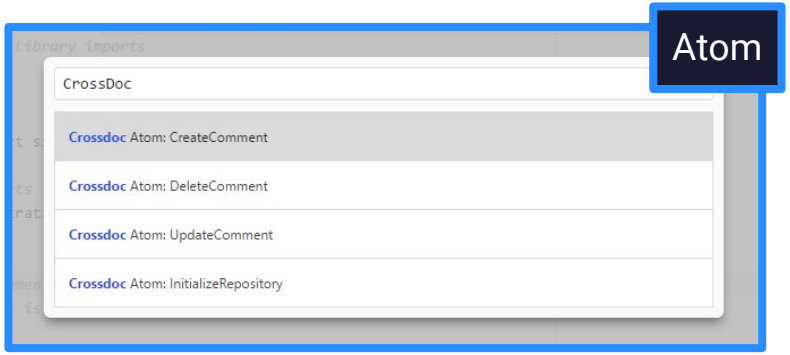
```
20  # <&> 20807c [Documentation]
21  # standard(message)
22  #   logs `message` to stdout
23  #
24  # usage(command=None)
25  #   logs the usage message for the command that calls this method
26  #   alternatively, logs the usage message for the given `command`
27  #
28  # program(message)
29  #   logs `message` to stdout prefixed with the program name
30  #
31  # fatal(message)
32  #   logs `message` to stdout prefixed with "fatal" and kills program
33  class Logger:
```

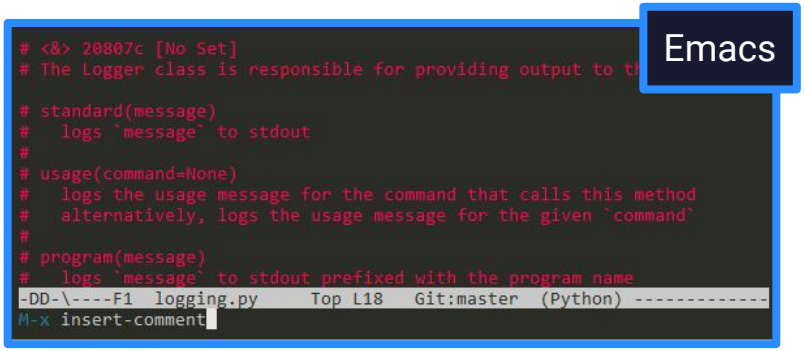
```
19  # <&> 20807c [TODO]
20  # * Modify where fatal logs its message (stdout -> stderr)
21  # * Add a warning logging method that prefixes messages with "warning"
22  class Logger:
```

Text Editor Plugins

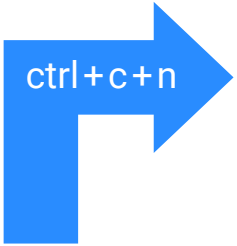




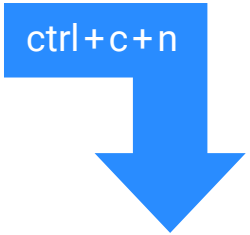




Comment Categories




```
# <&> 20807c [No Set]
# The Logger class is responsible for providing output to the console
class Logger:
```



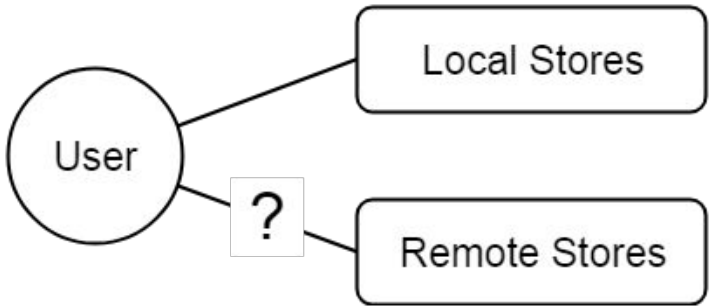

```
# <&> 20807c [TODO]
# * Modify where fatal logs its message (stdout -> stderr)
# * Add a warning logging method that prefixes messages with "warning"
class Logger:
```


```
# <&> 20807c [Documentation]
# standard(message)
#   logs `message` to stdout
#
# usage(command=None)
#   logs the usage message for the command that calls this method
#   alternatively, logs the usage message for the given `command`
#
# program(message)
#   logs `message` to stdout prefixed with the program name
#
# fatal(message)
#   logs `message` to stderr prefixed with fatal and kills program
class Logger:
```

Development Challenges

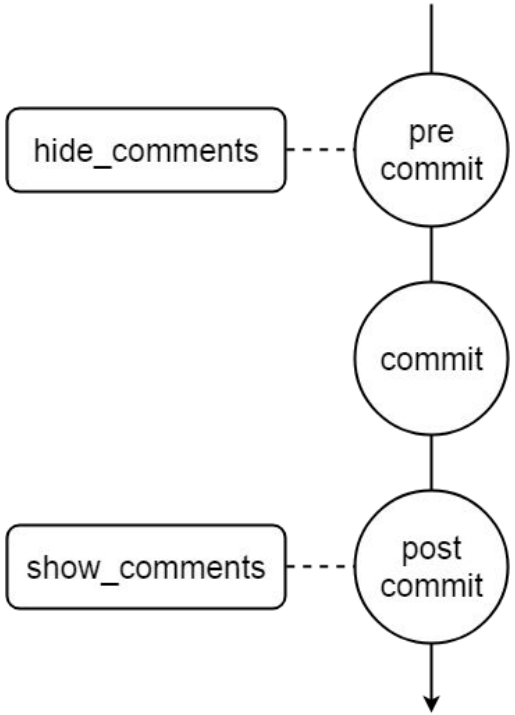
Development Challenges

- Managing multiple storage platforms
  - Remote and local storage
  - Internal platform validation
- Decoupling comments from version control
  - Removing redundancy from commits
  - Encapsulation of comment text



Development Solutions

- ~~Managing multiple storage platforms~~
  - Implementation of Wiki storage
  - Seamless integration with command line tool
- ~~Decoupling comments from version control~~
  - Git Hooks (pre and post commit)



Testing Plan

System Tests

Testing of the CrossDoc platform will leverage the use of Python's "unittest" library

- Unit Testing
  - Rigorous testing of CrossDoc command systems with all feasible inputs
  - 124 Equivalence Partitions
  - Function Coverage: 95%
  - Branch Coverage: 100%

- Integration Testing
  - Ensure functionality of the Text Editor Plugins to Command Line Program Chain
  - Atom, Emacs, Sublime, and Vim will utilize testing classes in the CL Program

Usability Tests

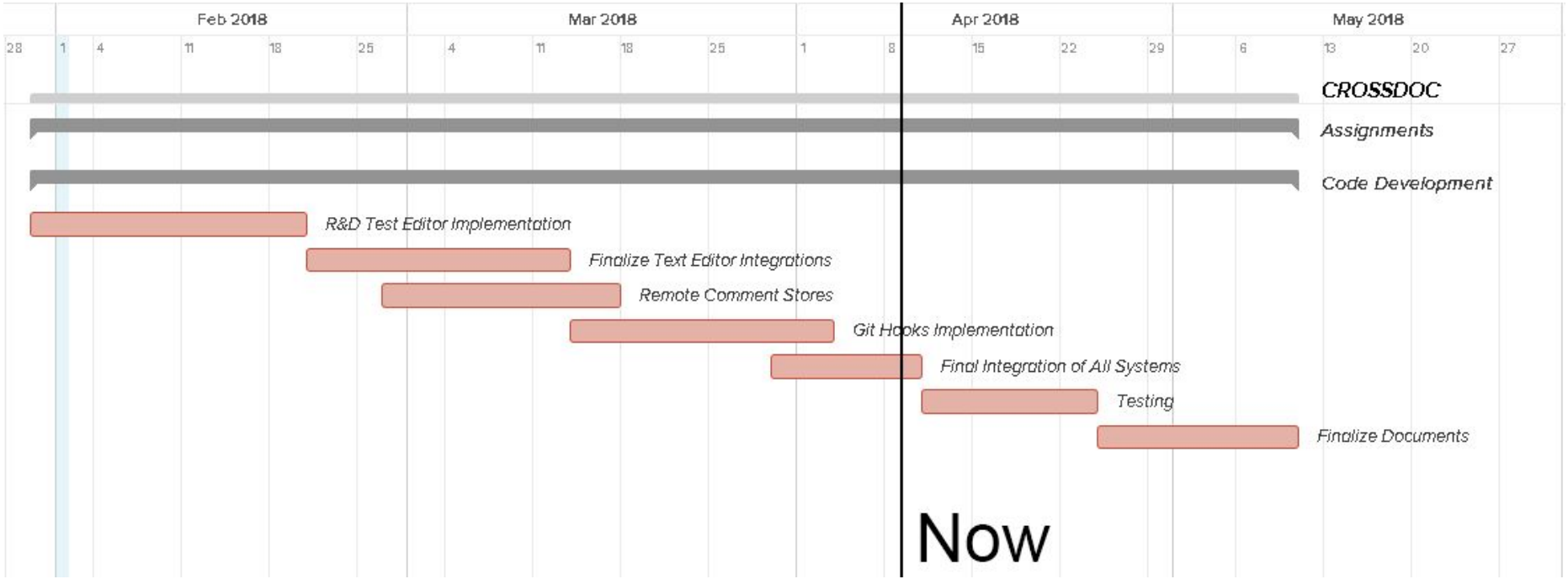
Testing the CrossDoc application with its two main user groups

- Software Developers
  - Main goal: Devs find it easy to create, push, and pull comments into the repository
  - Should also feel like normal commenting with our extended systems

- Technical Writers
  - Main goal: Non-programmers able to modify comment-base from Wiki location
  - Testing the functionality of Remote Stores and Ease of Use for writers

Development Schedule

Gantt Chart


Feb 2018
Mar 2018
Apr 2018
May 2018
28
1
4
11
18
25
4
11
18
25
1
8
15
22
29
6
13
20
27
CROSSDOC
Assignments
Code Development
R&D Test Editor Implementation
Finalize Text Editor Integrations
Remote Comment Stores
Git Hooks Implementation
Final Integration of All Systems
Testing
Finalize Documents
Now

Development Milestones

Previously Completed (DR2):

- Command-Line Program
- Text-Editor Plugins

Newly Completed (DR3):

- Testing Plan
- Developed Wiki extension for Remote Stores
- Began Foundation of Git-Hook pre and post commit systems

&20807c Logger Main

Contents [hide]
1 No Set
2 Documentation
3 TODO
4 Spanish Description

No Set [edit]

The Logger class is responsible for providing output to the console

Documentation [edit]

standard(message)

```
logs `message` to stdout
```

usage(command=None)

```
logs the usage message for the command that calls this method
alternatively, logs the usage message for the given `command`
```

program(message)

```
logs `message` to stdout prefixed with the program name
```

fatal(message)

```
logs `message` to stderr prefixed with fatal and kills program
```

TODO [edit]

- Modify where fatal logs its message (stdout -> stderr)
- Add a warning logging method that prefixes messages with "warning"

Summary

Problem & Solution Summary

```
10  # The Logger class is responsible for providing output to the console
11  #
12  # API documentation
13  # standard(message)
14  #   logs `message` to stdout
15  #
16  # usage(command=None)
17  #   logs the usage message for the command that calls this method
18  #   alternatively, logs the usage message for the given `command`
19  #
20  # program(message)
21  #   logs `message` to stdout prefixed with the program name
22  #
23  # fatal(message)
24  #   logs `message` to stdout prefixed with "fatal" and kills program
25  #
26  # Things to do
27  # * Modify where fatal logs its message (stdout -> stderr)
28  # * Add a warning logging method that prefixes messages with "warning"
29  class Logger:
30
31      def standard(message):
32          """Logs a message to the user (non-ending)"""
33
34          print(message)
35          return
```

Without CrossDoc

```
31  # <&> 20807c [No Set]
32  # The Logger class is responsible for providing output to the console
33  class Logger:
```

```
19  # <&> 20807c [TODO]
20  # * Modify where fatal logs its message (stdout -> stderr)
21  # * Add a warning logging method that prefixes messages with "warning"
22  class Logger:
```

With CrossDoc

In Conclusion

- Prototypes
  - Text Editor Plugins: Atom, Emacs, Sublime, Vim
  - Comment Categories within Editors
  - Remote Stores integration through Wiki
- Testing Plan
  - Unit Testing of CrossDoc core commands
  - Integration Testing of Chain between TE Plugins and CrossDoc
  - Usability Testing with Software Developers and Technical Writers
- Our Path Ahead
  - Finalize Git Hooks implementation
  - Write and execute tests according to Testing Plan
  - Creation of Easy to Use Documentation for End-Users

Questions/Comments