CrossDoc

Team: Octo-Docs

Team Members
Garrison Smith
Peter Huettl
Kristopher Moore
Brian Saganey

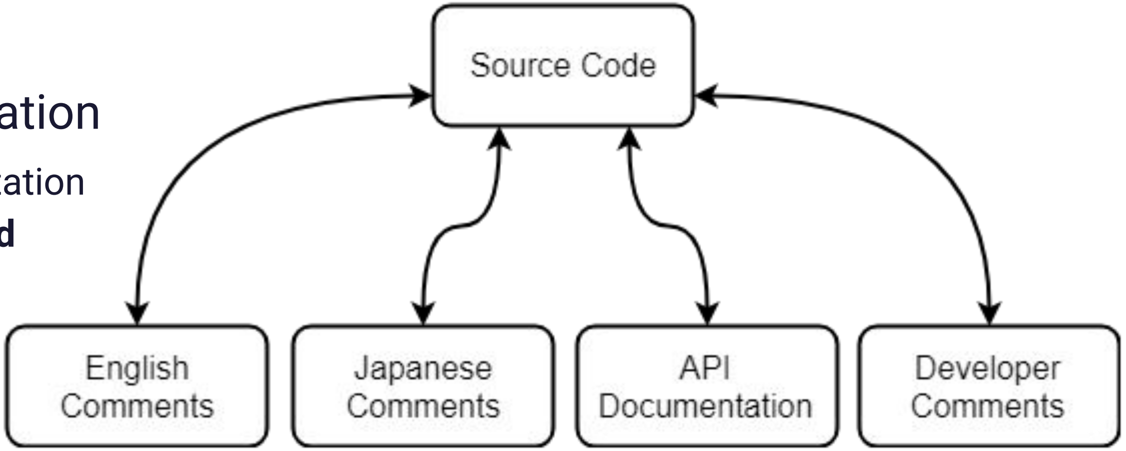
Client/Mentor

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

Problem Statement

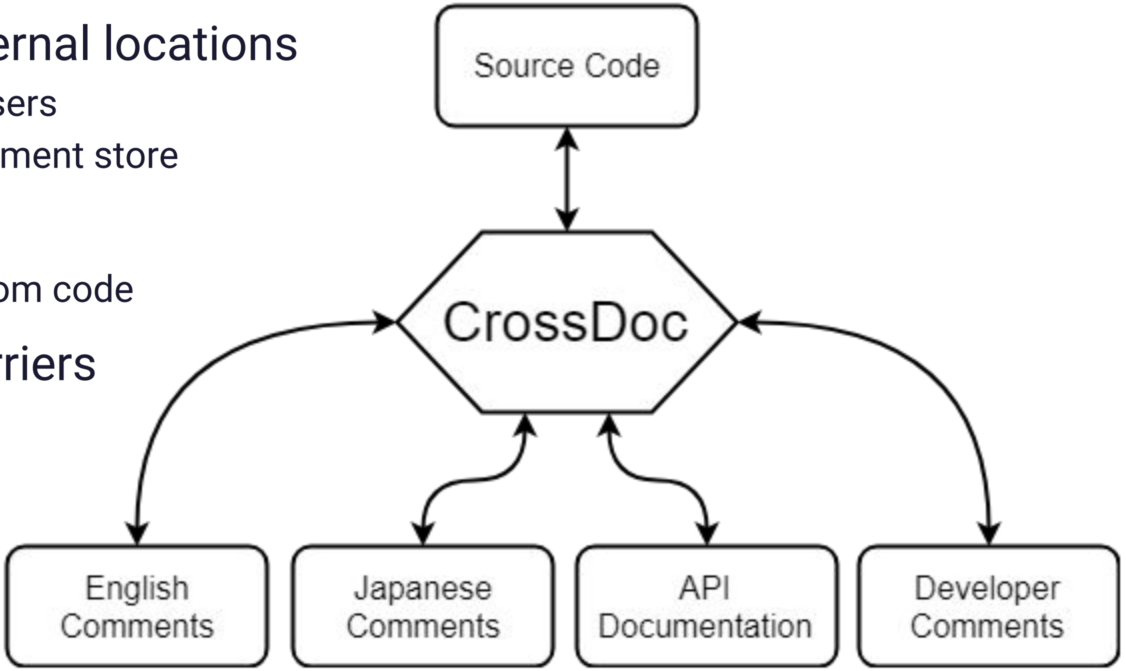
The Problem

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



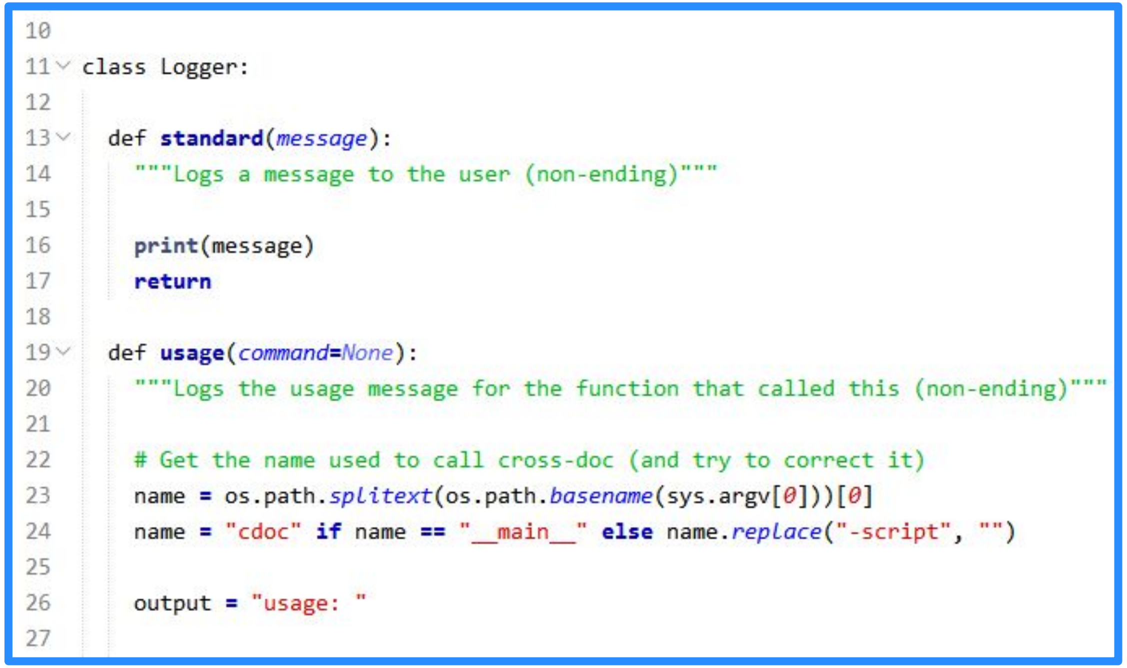
The Solution: CrossDoc

- 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

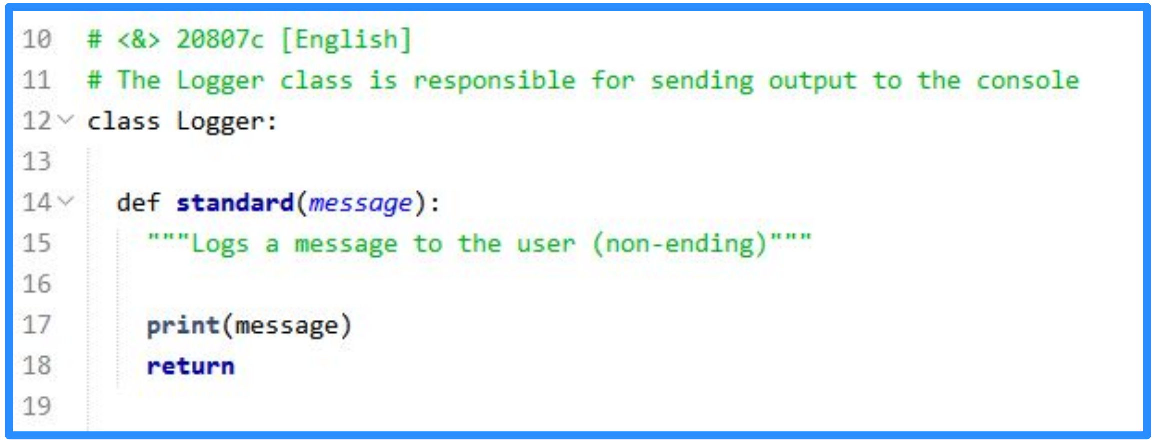
- Documentation is buried and coupled with the codebase
- Disorganized comments with jumbled information



```
class Logger:

  def standard(message):
    """Logs a message to the user (non-ending)"""

    print(message)
    return

  def usage(command=None):
    """Logs the usage message for the function that called this (non-ending)"""

    # Get the name used to call cross-doc (and try to correct it)
    name = os.path.splitext(os.path.basename(sys.argv[0]))[0]
    name = "cdoc" if name == "__main__" else name.replace("-script", "")

    output = "usage: "

```

Solution Visualized

- Provide a better commenting method with CrossDoc
- Scalable, external storage, and enhanced functionalities



```python
# <&> 20807c [English]
# The Logger class is responsible for sending output to the console
class Logger:

  def standard(message):
    """Logs a message to the user (non-ending)"""

    print(message)
    return

```

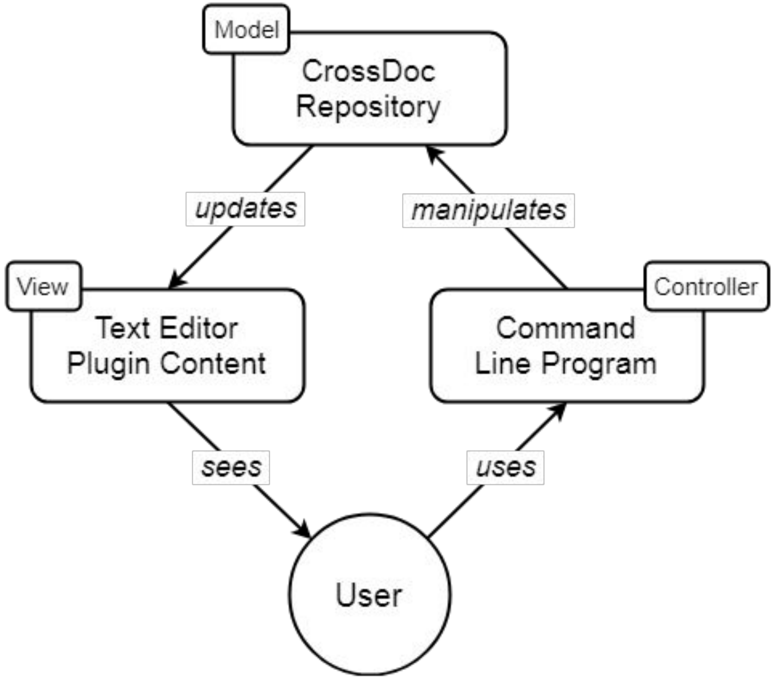
CrossDoc Key Requirements

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

Architecture and Implementation

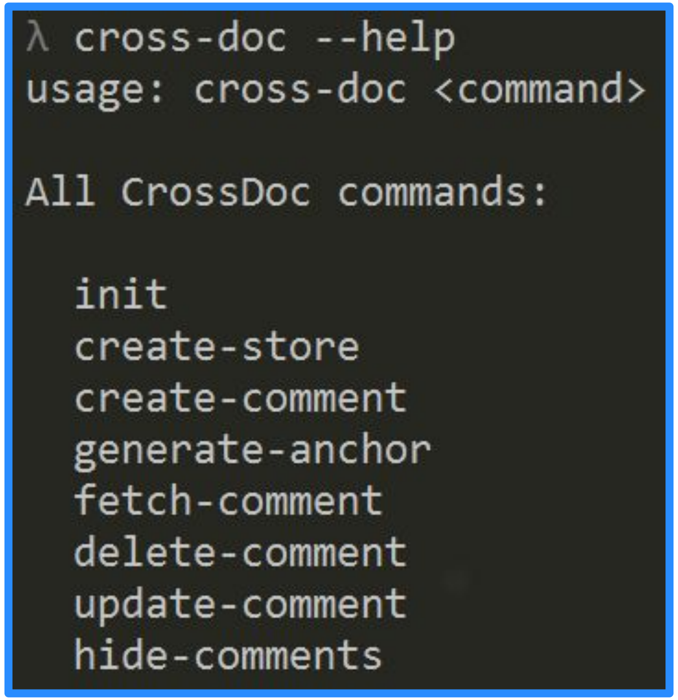
High Level Overview

- 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

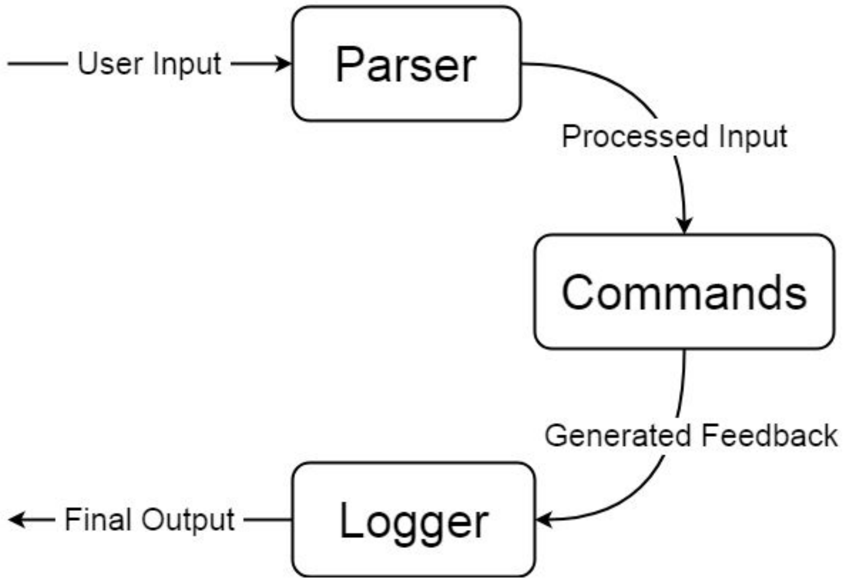
- Provides API to interact with tool
- Text editor agnostic
- Implements core functionality
    - Create comments
    - Read comments
    - Delete comments
    - Etc..



```
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
```

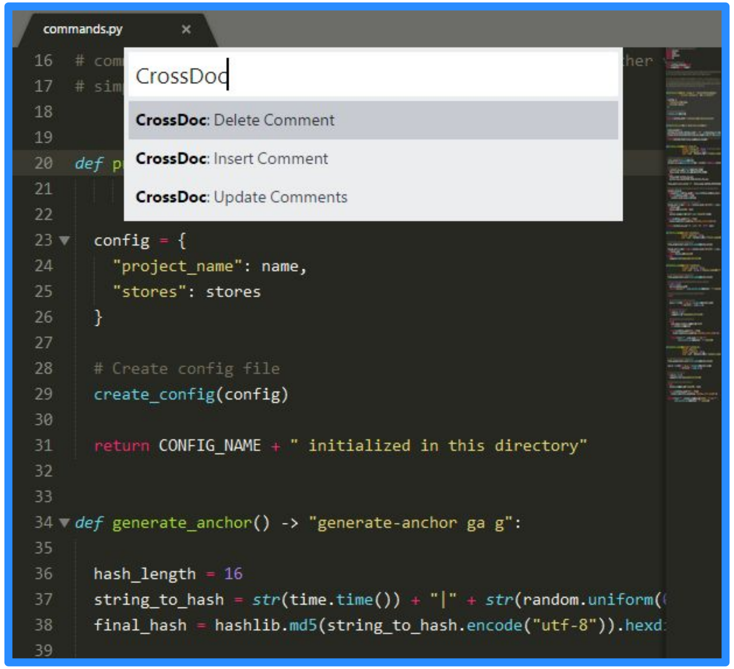
Command Line Program

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



Text Editor Plugins

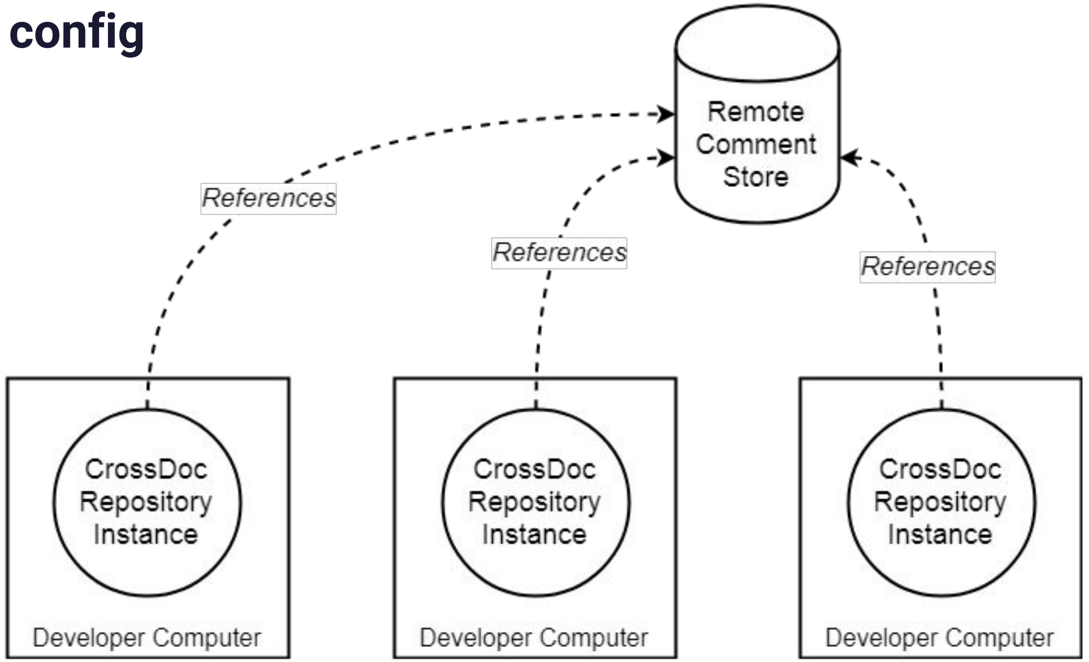
- CrossDoc user interface
- Intuitive commands and hotkeys
- Support for multiple text editors
  - Atom
  - Emacs
  - Sublime
  - Vim



```
# com                                                ther
# sim


def p


    config = {
        "project_name": name,
        "stores": stores
    }

    # Create config file
    create_config(config)

    return CONFIG_NAME + " initialized in this directory"


def generate_anchor() -> "generate-anchor ga g":

    hash_length = 16
    string_to_hash = str(time.time()) + "|" + str(random.uniform(
    final_hash = hashlib.md5(string_to_hash.encode("utf-8")).hexd
```

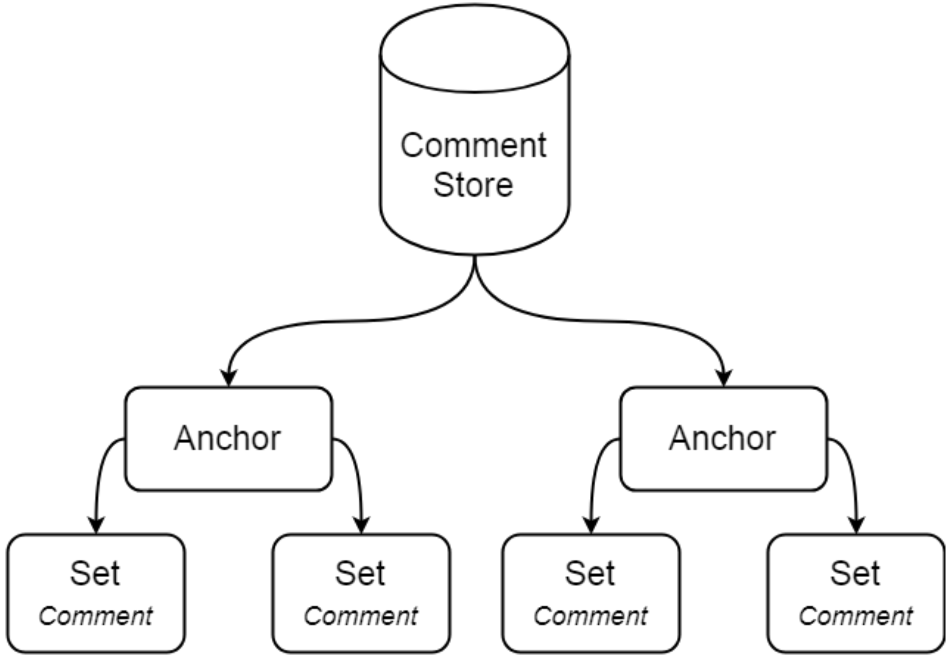
CrossDoc Repository

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



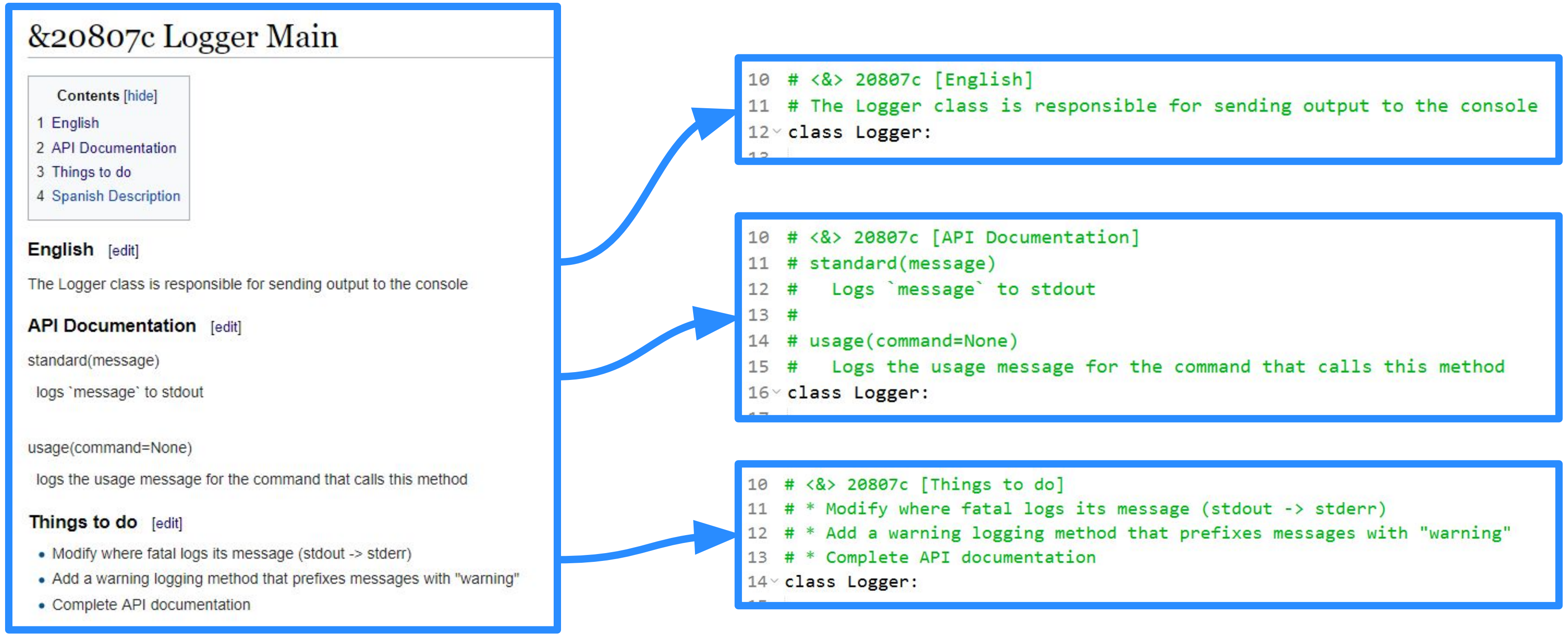
Comment Storage

- Comment stores
  - Directory of anchors
  - Local and remote
- CrossDoc anchors
  - Comment identifier
- Comment sets
  - Distinct categories
  - Stores comment text



Prototype Review

External Comment Storage



&20807c Logger Main

Contents [hide]

English [edit]

The Logger class is responsible for sending output to the console

API Documentation [edit]

standard(message)

logs `message` to stdout

usage(command=None)

logs the usage message for the command that calls this method

Things to do [edit]

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

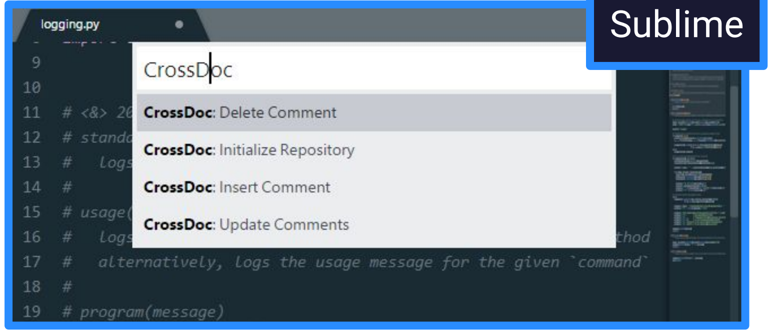
```
# <&> 20807c [English]
# The Logger class is responsible for sending output to the console
class Logger:
```

```
# <&> 20807c [API Documentation]
# standard(message)
#   Logs `message` to stdout
#
# usage(command=None)
#   Logs the usage message for the command that calls this method
class Logger:
```

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

Text Editor Plugins

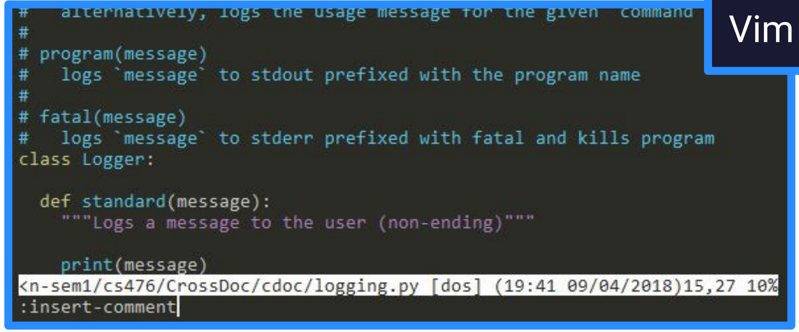
Sublime



```
logging.py

# <&> 20
# standa
#   logs
#
# usage(
#   logs                                                    thod
#   alternatively, logs the usage message for the given `command`
#
# program(message)
```

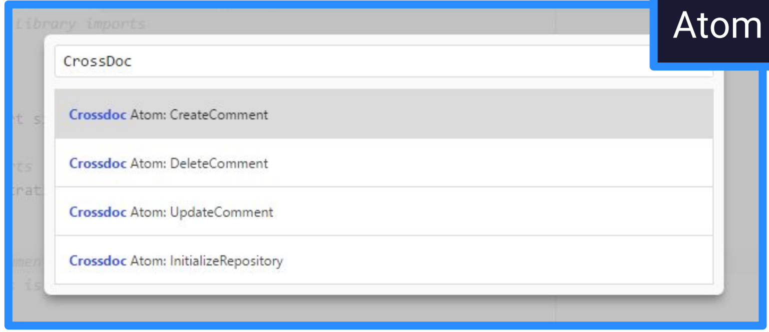
```
CrossDoc
CrossDoc: Delete Comment
CrossDoc: Initialize Repository
CrossDoc: Insert Comment
CrossDoc: Update Comments
```

Vim



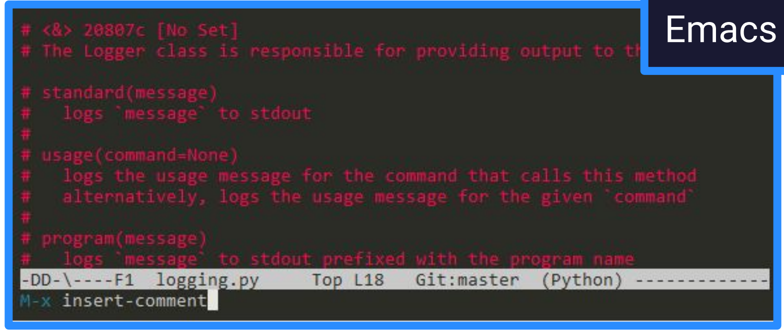
```
#   alternatively, logs the usage message for the given `command`
#
# program(message)
#   logs `message` to stdout prefixed with the program name
#
# fatal(message)
#   logs `message` to stderr prefixed with fatal and kills program
class Logger:

  def standard(message):
    """Logs a message to the user (non-ending)"""

    print(message)
<n-sem1/cs476/CrossDoc/cdoc/logging.py [dos] (19:41 09/04/2018)15,27 10%
:insert-comment
```

Atom



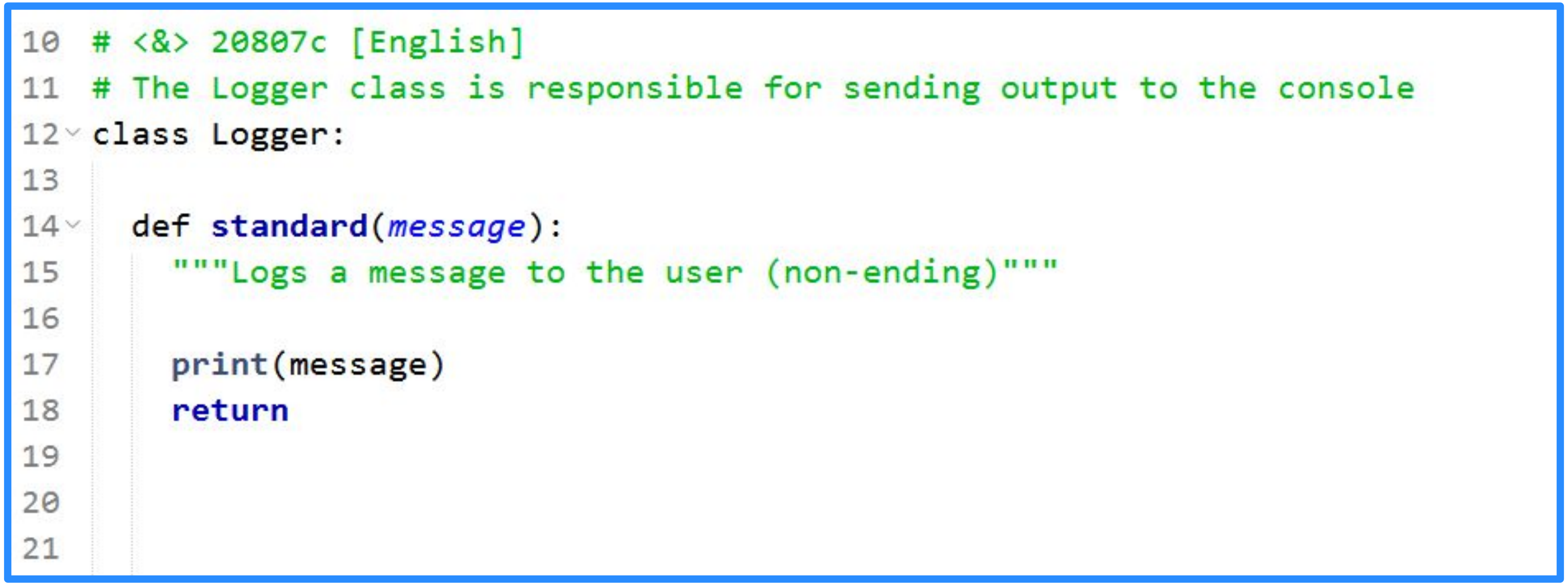
```
CrossDoc
Crossdoc Atom: CreateComment
Crossdoc Atom: DeleteComment
Crossdoc Atom: UpdateComment
Crossdoc Atom: InitializeRepository
```

Emacs



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

# 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
-DD-\----F1  logging.py      Top L18    Git:master  (Python) ------------
M-x insert-comment
```

Comment Categories

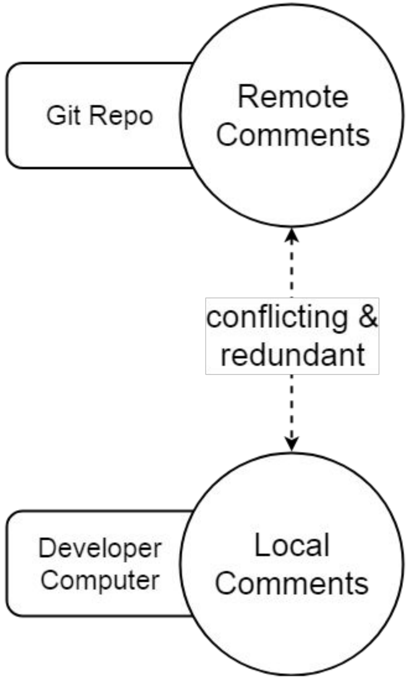


```python
# <&> 20807c [English]
# The Logger class is responsible for sending output to the console
class Logger:

    def standard(message):
        """Logs a message to the user (non-ending)"""

        print(message)
        return



```

Development Challenges

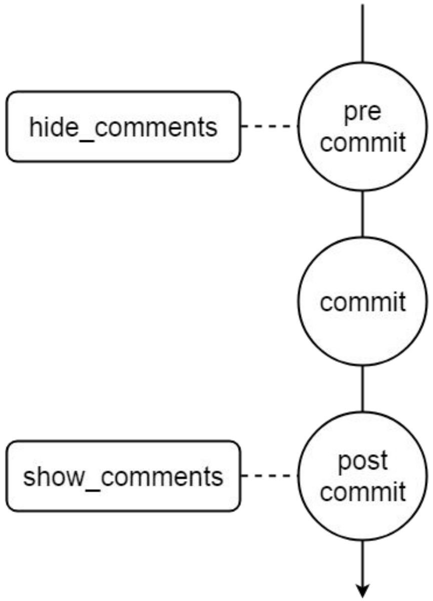
Development Challenges

- Consistent functionalities across editors
  - Managing limitations of text editor APIs
  - Developing a consistent user experience
- Managing multiple storage platforms
  - Remote and local storage
  - Internal platform validation
- Decoupling comments from version control
  - Removing redundancy from commits
  - Encapsulation of comment text



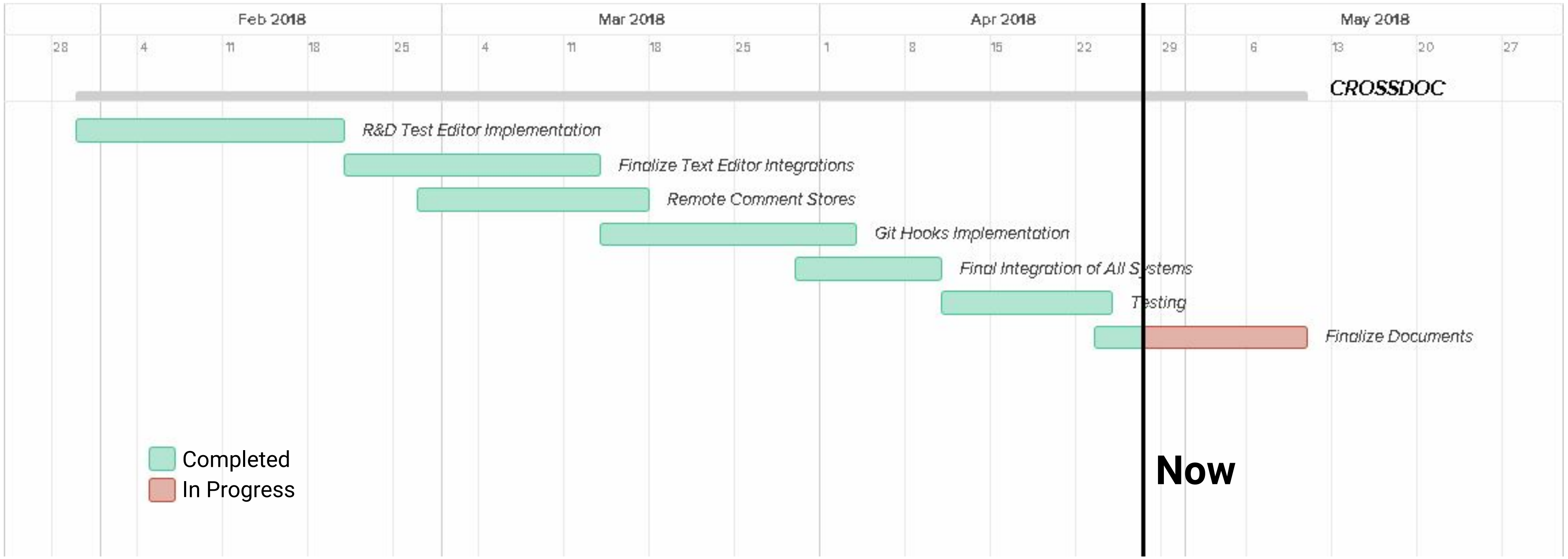
Development Solutions

- ~~Consistent functionalities across editors~~
  - Provided unified API through command line program
  - Integrated commands directly into each editor's command API
- ~~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)



Development Schedule

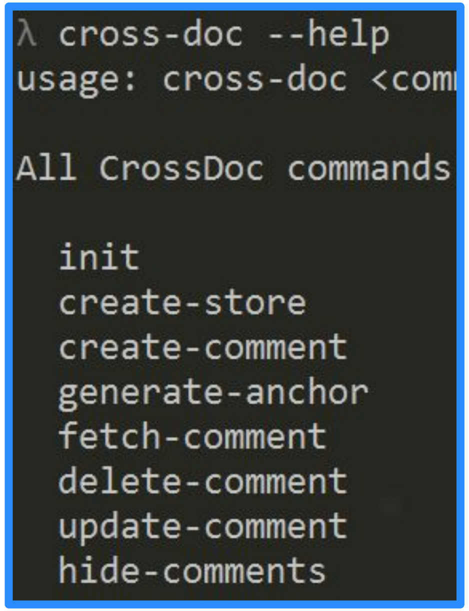
Gantt Chart



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

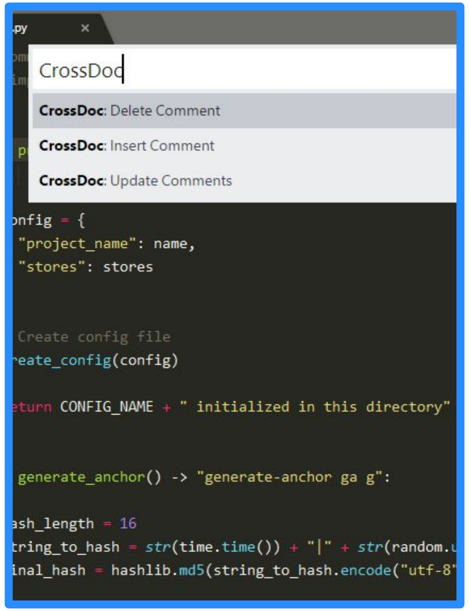
Schedule Milestones

Command-Line Program



```
λ cross-doc --help
usage: cross-doc <comm

All CrossDoc commands

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

Text-Editor Plugins



CrossDoc

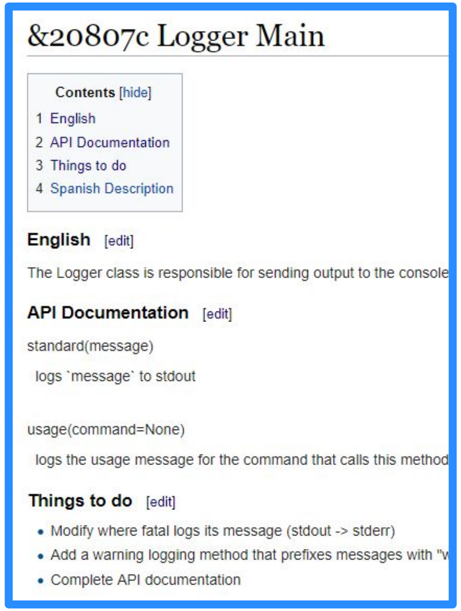
CrossDoc: Delete Comment

CrossDoc: Insert Comment

CrossDoc: Update Comments

```
onfig = {
 "project_name": name,
 "stores": stores

 Create config file
reate_config(config)

eturn CONFIG_NAME + " initialized in this directory"

 generate_anchor() -> "generate-anchor ga g":

ash_length = 16
tring_to_hash = str(time.time()) + "|" + str(random.u
inal_hash = hashlib.md5(string_to_hash.encode("utf-8"
```

Remote Storage



&20807c Logger Main

Contents [hide]
1 English
2 API Documentation
3 Things to do
4 Spanish Description

English [edit]

The Logger class is responsible for sending output to the console

API Documentation [edit]

standard(message)

logs `message` to stdout

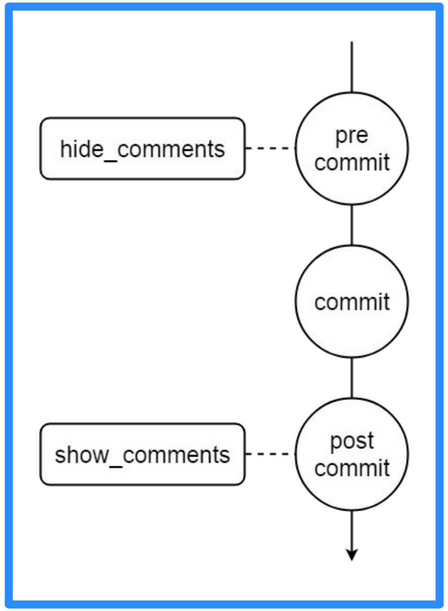
usage(command=None)

logs the usage message for the command that calls this method

Things to do [edit]

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

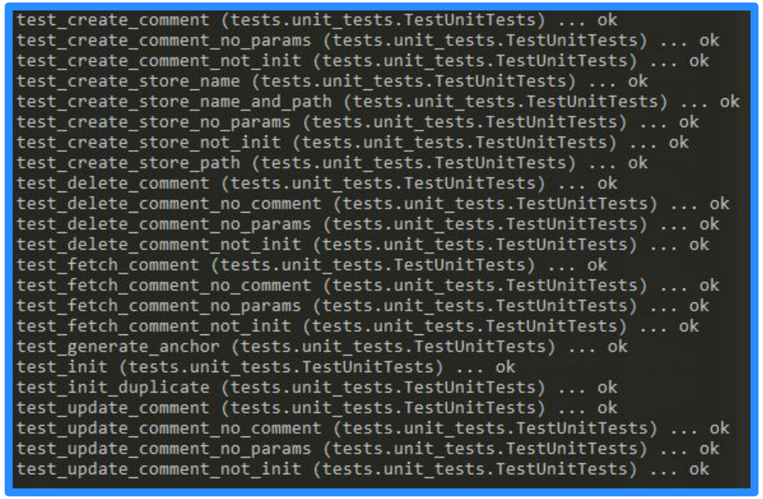
Git-Hooks



System Tests

- Unit Testing
  - 124 Equivalence Partitions
  - Command Coverage: 100%
  - Branch Coverage: 100%
  - Python's **unittest** library

- 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**



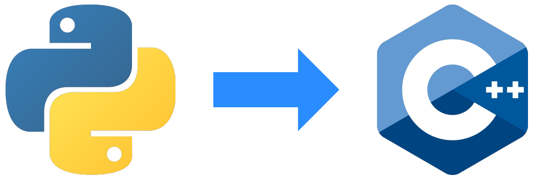
```
test_create_comment (tests.unit_tests.TestUnitTests) ... ok
test_create_comment_no_params (tests.unit_tests.TestUnitTests) ... ok
test_create_comment_not_init (tests.unit_tests.TestUnitTests) ... ok
test_create_store_name (tests.unit_tests.TestUnitTests) ... ok
test_create_store_name_and_path (tests.unit_tests.TestUnitTests) ... ok
test_create_store_no_params (tests.unit_tests.TestUnitTests) ... ok
test_create_store_not_init (tests.unit_tests.TestUnitTests) ... ok
test_create_store_path (tests.unit_tests.TestUnitTests) ... ok
test_delete_comment (tests.unit_tests.TestUnitTests) ... ok
test_delete_comment_no_comment (tests.unit_tests.TestUnitTests) ... ok
test_delete_comment_no_params (tests.unit_tests.TestUnitTests) ... ok
test_delete_comment_not_init (tests.unit_tests.TestUnitTests) ... ok
test_fetch_comment (tests.unit_tests.TestUnitTests) ... ok
test_fetch_comment_no_comment (tests.unit_tests.TestUnitTests) ... ok
test_fetch_comment_no_params (tests.unit_tests.TestUnitTests) ... ok
test_fetch_comment_not_init (tests.unit_tests.TestUnitTests) ... ok
test_generate_anchor (tests.unit_tests.TestUnitTests) ... ok
test_init (tests.unit_tests.TestUnitTests) ... ok
test_init_duplicate (tests.unit_tests.TestUnitTests) ... ok
test_update_comment (tests.unit_tests.TestUnitTests) ... ok
test_update_comment_no_comment (tests.unit_tests.TestUnitTests) ... ok
test_update_comment_no_params (tests.unit_tests.TestUnitTests) ... ok
test_update_comment_not_init (tests.unit_tests.TestUnitTests) ... ok
```

Usability Tests

- **Group A:** Software Developers
  - Main goal: Devs find it easy to create, push, and pull comments with CrossDoc
  - Should also feel like normal commenting with our extended systems
- **Group B:** Technical Writers
  - Main goal: Non-programmers able to modify comment-base from Wiki location
  - Testing the consistency of remote stores and ease-of-use for writers

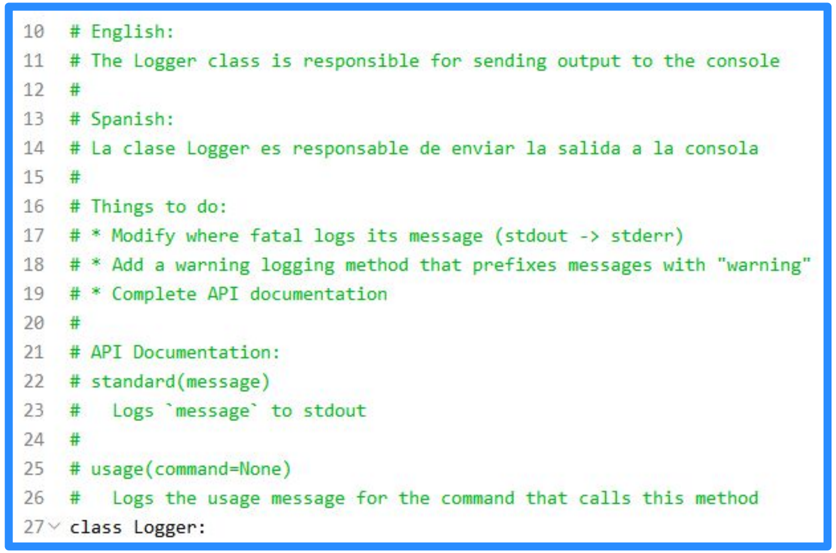
Future Work

- Further optimize command line program performance
- Integrate with comment formats such as Doxygen and Javadoc
- Mechanism that flags out-of-date documentation

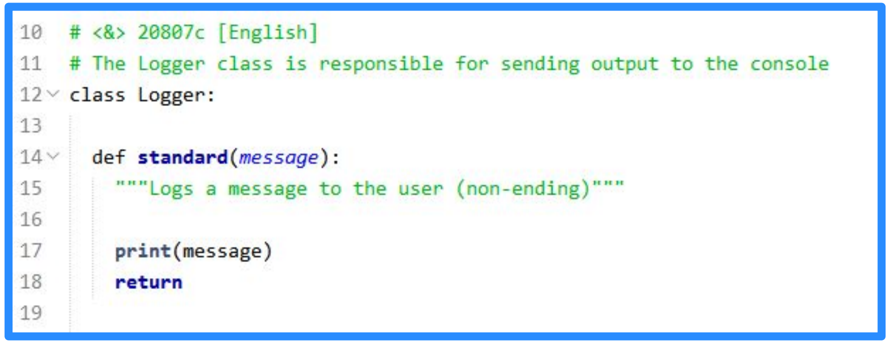


Summary

Problem & Solution Summary



```
# English:
# The Logger class is responsible for sending output to the console
#
# Spanish:
# La clase Logger es responsable de enviar la salida a la consola
#
# Things to do:
# * Modify where fatal logs its message (stdout -> stderr)
# * Add a warning logging method that prefixes messages with "warning"
# * Complete API documentation
#
# API Documentation:
# standard(message)
#   Logs `message` to stdout
#
# usage(command=None)
#   Logs the usage message for the command that calls this method
class Logger:
```



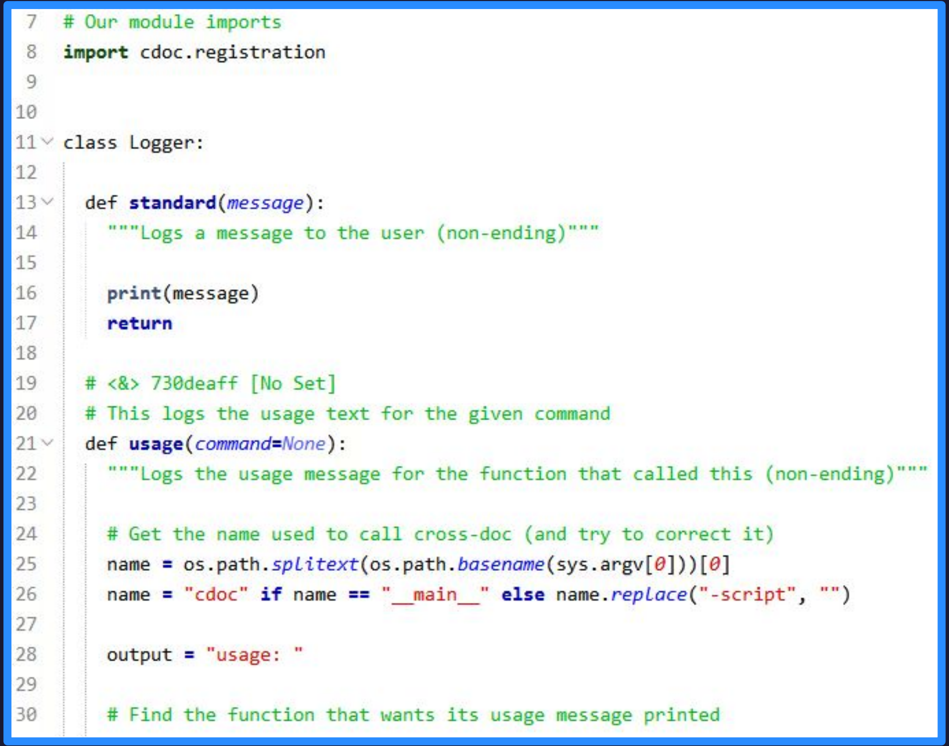
```
# <&> 20807c [English]
# The Logger class is responsible for sending output to the console
class Logger:

  def standard(message):
    """Logs a message to the user (non-ending)"""

    print(message)
    return

```

Without CrossDoc

With CrossDoc

In Conclusion

- Design
  - MVC style architecture
  - Modular components
- Implementation
  - Robust command tool
  - Text editor plugins
- Impact
  - Decouple comments/code
  - Remote comment editing
  - Comment categories



```python
# Our module imports
import cdoc.registration


class Logger:

  def standard(message):
    """Logs a message to the user (non-ending)"""

    print(message)
    return

  # <&> 730deaff [No Set]
  # This logs the usage text for the given command
  def usage(command=None):
    """Logs the usage message for the function that called this (non-ending)"""

    # Get the name used to call cross-doc (and try to correct it)
    name = os.path.splitext(os.path.basename(sys.argv[0]))[0]
    name = "cdoc" if name == "__main__" else name.replace("-script", "")

    output = "usage: "

    # Find the function that wants its usage message printed
```

Questions/Comments

Poster Presentation 2-4pm

Walkup Skydome