Project 1: Milestone 1

ECE 46100 - Team 4

As a Boilermaker pursuing academic excellence, we pledge to be honest and true in all that we do.
Accountable together – We are Purdue.

Prepared for: Dr. James Davis
Prepared by: Mohammed Fashola, Owen Prince, Ryan Villarreal
Sep 12, 2021

Tool selection and preparation

Programming language	We will be coding in Python for this project.
linter	We will be using PyLint as a linter for this project.
Git-hooks	We will not be using git hooks
CI	None for this project
Testing framework	We will be using pytest as the testing framework
Logging library	We will be using the Python inbuilt logging library to track events in our program (LogRecord)
Libraries	We will be using the following tools, libraries and APIs: ● GitHub Rest API ● PyGitHub to interact with the GitHub Rest API ● Regex library ● Sys ● Os ● Numpy ● Pandas ● Dotenv ● Pprint ● Coverage.py to test coverage ● Pyinstaller to create executable
Statement that GitHub tokens are obtained	GitHub tokens are to be obtained and stored in a “.env” file by each member. This “.env” file will not be stored on GitHub nor shared with anyone other than the developer.

Team contract

- Attend all weekly meetings on time
    - If meetings are missed for any reason, individual must make up work and find out what was missed
- Look at Slack daily/often
- Always give descriptions of any commits or changes to the project repository such that
- Google Drive will be utilized only as a method for editing documents simultaneously. All files related to project must be uploaded to the team repository
- Meet deadlines or clearly communicate if not possible
- Github:
    - Each use different "feature" branches
    - Keep main branch clean
- Style guide:
    - Split into small chunks
    - DRY
    - Document confusing lines, make code readable
    - Try to use "pythonic" practices
    - Utilize VSCode as an IDE for development
    - Utilize Debug statements for each module for easy error tracing
    - Frequently Commit Code updates to Github as a backup
        - Only push to your individual branch, unless consulted with entire group
- Tests have 80-100% code coverage
- Maintain a separate file with test cases and log calls
- Keep modules separate in order to allow for easier debugging

Team synchronous meeting times

Mid-week sync on Thursdays from 4:30 - 5:30 PM EST, end of week sync on Saturdays for one hour, time TBD. Meeting on Mondays for 30 minutes to review and turn in a weekly report.

Requirements

Documentation requirements	● Code is well documented and readme clearly documents the proper usage of the program. ● Each class and their respective methods have comments that clearly define their purposes.

Program Requirements	● Design is completely open source and adheres to GNU Lesser General Public License v2.1 ● Program takes less than a minute for 10 URLS between starting execution and printing the score of each URL. ● An executable file called "run" is located in the root directory of the project. ● Program will install dependencies when "./run install" is executed within the root directory of the project and exit 0. ● Program will analyze a variable-length list of URLs when the command "./run URL_FILE" is executed, where URL_FILE is the list of URLs, and then exit 0. ● The list of URLs is formatted as an ASCII-encoded newline-delimited string. ● Program analyzes packages under the npmjs.com domain or the GitHub domain. ● Program should access the GitHub API programmatically ● Program calculates 5 separate metrics-- each metric returning a score in the interval [0, 1]-- and a net score. The five metrics are listed below. ○ Ramp up score ○ Correctness score ○ Bus factor score ○ Responsive maintainer score ○ License score ● At least one metric includes data from the source code repository. ● Net score is a weighted sum within the interval of [0, 1] with weights that reflect Sarah's priorities. ● Program prints them in the following format: "URL_1 net_score ramp_up_score correctness_score bus_factor_score responsive_maintainer_score license_score" ● Program prints URL scores in the order of trustworthiness (highest net score to lowest net score). ● Program does not upload tokens to a publicly-visible location.
Testing requirements	● Program will execute a test suite when the command "./run test" is entered on the command line. ● Program has a corresponding test suite with at least 20 test cases. ● Test suite achieves 80% code (line) coverage. ● Each test case within the test suite prints results in the form "X/Y test cases passed. Z% line coverage achieved." ● Software will produce a log file in the location $LOG_FILE. ● Verbosity of log file is described by the environment variable $LOG_LEVEL, where a value of 0 is silent, 1 is information messages, 2 is debug messages. ● Program should include descriptive informational and debug messages.
Score requirements	

Ramp-up score requirements	● Ramp-up score should reflect the learning curve of respective module: ○ Ramp-up factor should depend on descriptiveness of module wiki (larger wiki corresponds to higher score). ○ Ramp-up factor should depend on ratio of the number of lines of comments to the total number of source lines of code (slocs)
Correctness score requirements	● Correctness should reflect the overall correctness of the program: ○ High level of correctness should correspond with a low number of outstanding issues. ○ High level of correctness should correlate with the date of the latest release of the module. ○ Patches are reviewed by a high number of people before they're implemented. ○ Higher number of stars/downloads contributes to a higher correctness. ○ Module has been checked for correctness by a third party and has passed
Bus-factor score requirements	● Bus-factor score should reflect the number of critical developers on the project: ○ Large group of contributors reflects a better bus factor. ○ Higher number of contributors that contributed to the last release corresponds to a better bus factor. ○ An even workload distribution among developers reflects a better bus factor.
Responsive maintainer score requirements	● Responsive maintainer score should reflect the time it would take to post a security patch to the module: ○ Responsive maintainer score should correspond to the time between releases ○ The turnaround time for resolving issues should be low. ○ There should be a high number of contributors in the project.
License compatibility score requirements	● License compatibility score requirement should reflect the compatibility of the license with the LGPLv2.1 license: ○ License is one of the licenses listed in this table: (see the end of the document if the link doesn't work)

Preliminary design

Metric operationalizations and net score formula

Metric Calculations:	
Ramp-up score requirements	● Ramp-up score should reflect the learning curve of respective module: ○ EQN1: Readme includes install section, usage section, run section ■ ⅓ if only 1 section included ■ ⅔ if 2 sections included ■ 1 if all sections included ○ EQN2: Ramp-up factor should depend on ratio of the number of lines of comments to the total number of source lines of code (slocs) ■ # of lines of comments / # slocs (capped at 1) ● **Overall Ramp-up score = (EQN1 + EQN2) / 2**
Correctness score requirements	● Correctness should reflect the overall correctness of the program: ○ EQN1: High level of correctness should correspond with a low number of outstanding issues. ■ # outstanding issues / total # of issues ○ EQN2: High level of correctness should correlate with the date of the latest release of the module. ■ 1 if latest release is within the last year; 0 otherwise ○ EQN3: Higher number of stars/downloads contributes to a higher correctness. ■ # of clones this week / # of clones this week ○ EQN4: Module has been checked for correctness by a third party and has passed ■ 1 if passed ■ 0 otherwise ● **Overall Correctness score = (EQN1 + EQN2 + EQN3 + EQN4) / 4**
Bus-factor score requirements	● Bus-factor score should reflect the number of critical developers on the project: ○ EQN1: Large group of contributors reflects a better bus factor: formula is ■ **1 - 1/# of contributors** ○ EQN2: Higher number of contributors that contributed to the last release corresponds to a better bus factor. ■ **1- 1/# of contributors for most recent release** ○ EQN3: An even workload distribution among developers reflects a better bus factor. ■ **1 - top contributor commits / all commits** ● **Overall Bus-factor score = (EQN1 + EQN2 + EQN3) / 3**
Responsive maintainer score	● Responsive maintainer score should reflect the time it would take to post a security patch to the module:

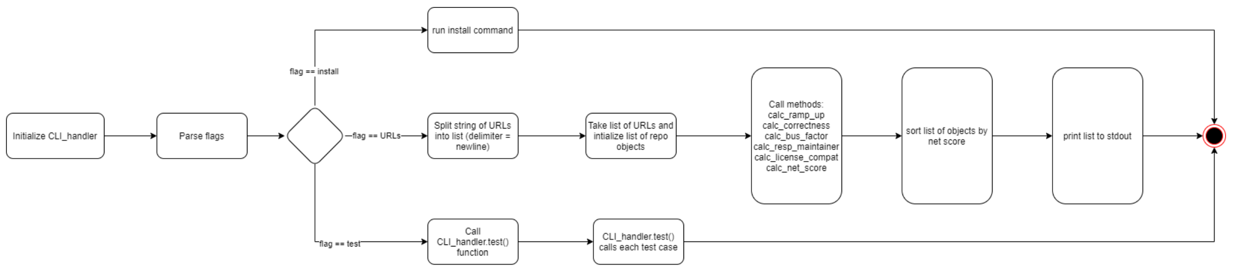
requirements	○ EQN1: Responsive maintainer score should correspond to the time between releases ■ **1 if 2 or more releases per year** ○ EQN2: The turnaround time for resolving issues should be low. ■ 0 if turnaround time is longer than 1 week ■ 1 if turnaround time <= 1 week ○ EQN3: There should be a high number of contributors in the project. ■ **1/10 * # contributors if below 10, otherwise 1** ● **Overall responsive maintainer score = (EQN1 + EQN2 + EQN3) / 3**
License compatibility score requirements	● License compatibility score requirement should reflect the compatibility of the license with the LGPLv2.1 license: ○ 1 if license is located in this table of compatible licenses ○ 0 otherwise ● **Overall License compatibility score is either 0 or 1**

Diagrams to support planning

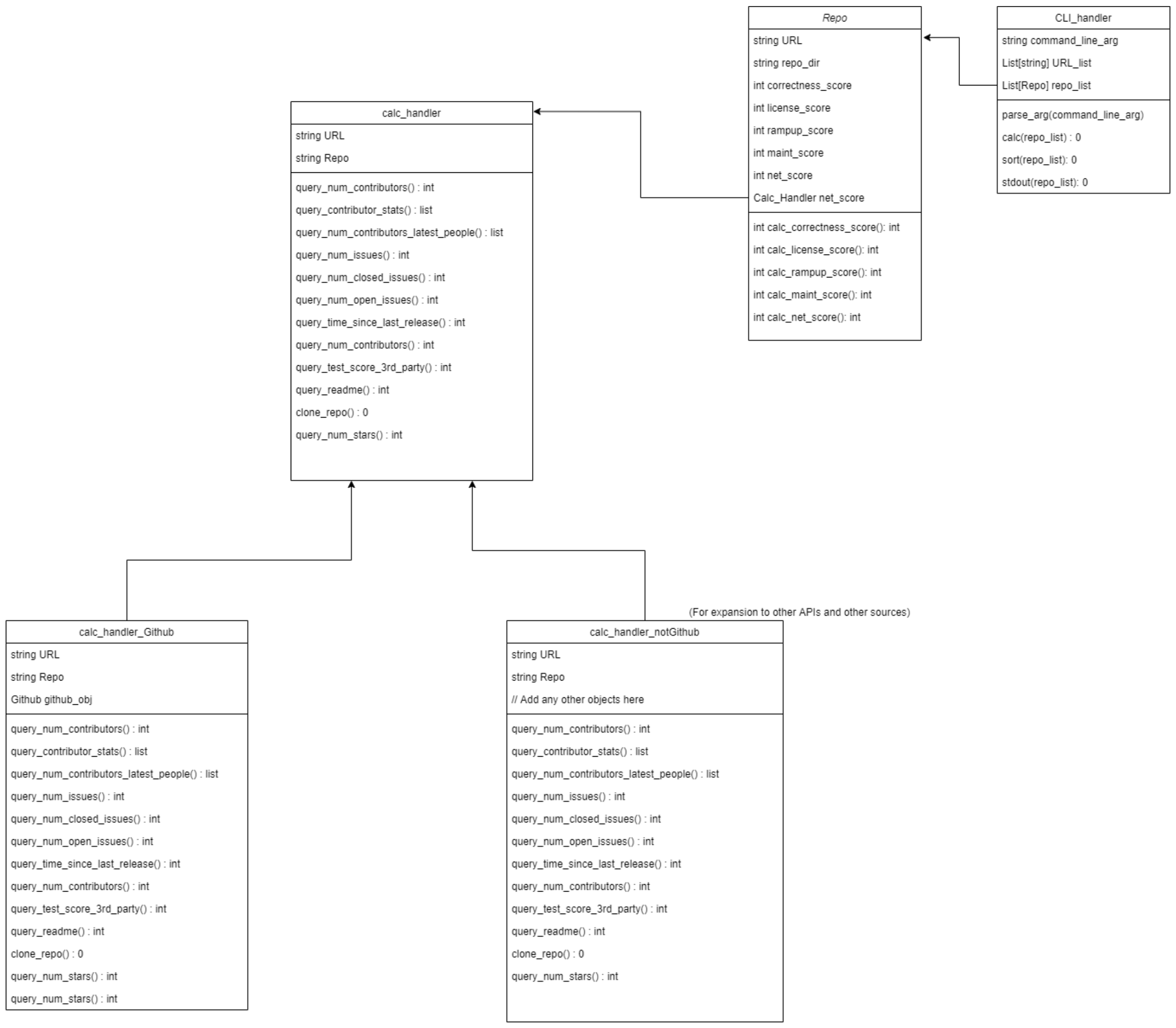
After evaluating Sarah's needs, we decided on the following score breakdown.

$$net\_score = license\_score * (0.4 * maintenance + 0.25 * bus\_factor + 0.25 * correctness + 0.1 * ramp\_up)$$

Below is a simple activity diagram that describes all the activities of the system:


run install command
flag == install
Initialize CLI_handler
Parse flags
flag == URLs
Split string of URLs into list (delimiter = newline)
Take list of URLs and intialize list of repo objects
Call methods: calc_ramp_up calc_correctness calc_bus_factor calc_resp_maintainer calc_license_compat calc_net_score
sort list of objects by net score
print list to stdout
flag == test
Call CLI_handler.test() function
CLI_handler.test() calls each test case

This is our simplified UML model:


Repo
string URL
string repo_dir
int correctness_score
int license_score
int rampup_score
int maint_score
int net_score
Calc_Handler net_score
int calc_correctness_score(): int
int calc_license_score(): int
int calc_rampup_score(): int
int calc_maint_score(): int
int calc_net_score(): int
CLI_handler
string command_line_arg
List[string] URL_list
List[Repo] repo_list
parse_arg(command_line_arg)
calc(repo_list) : 0
sort(repo_list): 0
stdout(repo_list): 0
calc_handler
string URL
string Repo
query_num_contributors() : int
query_contributor_stats() : list
query_num_contributors_latest_people() : list
query_num_issues() : int
query_num_closed_issues() : int
query_num_open_issues() : int
query_time_since_last_release() : int
query_num_contributors() : int
query_test_score_3rd_party() : int
query_readme() : int
clone_repo() : 0
query_num_stars() : int
(For expansion to other APIs and other sources)
calc_handler_Github
string URL
string Repo
Github github_obj
query_num_contributors() : int
query_contributor_stats() : list
query_num_contributors_latest_people() : list
query_num_issues() : int
query_num_closed_issues() : int
query_num_open_issues() : int
query_time_since_last_release() : int
query_num_contributors() : int
query_test_score_3rd_party() : int
query_readme() : int
clone_repo() : 0
query_num_stars() : int
query_num_stars() : int
calc_handler_notGithub
string URL
string Repo
// Add any other objects here
query_num_contributors() : int
query_contributor_stats() : list
query_num_contributors_latest_people() : list
query_num_issues() : int
query_num_closed_issues() : int
query_num_open_issues() : int
query_time_since_last_release() : int
query_num_contributors() : int
query_test_score_3rd_party() : int
query_readme() : int
clone_repo() : 0
query_num_stars() : int

We designed the metrics feature such that the “repo” class will contain the formulas to calculate each metric, and the calc_handler class will contain the methods that interact with the GitHub API. The “calc_*” methods (calc_bus_factor, for example) will only implement math or call functions from the calc_handler class to return relevant numbers (query_num_contributors, for example). This makes it very easy to implement new metrics. All that is needed is to add a new function to do the calculations, determine which numbers you will need from the API, then update the formula for the net_score calculation.

The handle URLs feature is designed as modularly as possible. We implement a ‘template’ design pattern by only handling URLs in our calc_handler class. Each type of URL will be handled by a different child class of calc_handler. We will be using the calc_handler_git class for our current implementation. The methods of the child classes will be kept standard, so both the calc_handler_git and calc_handler_notGithub classes will have all the same methods and return the same results. This way, the repo classes can make the same function calls no matter the URL type and extending the functionality only requires implementing one new class and updating the methods by which you query methods from the remote repository.

Planned milestones for weeks 2-4

Week 2 Planning/Implementing:

Task	**Owner**	**Time to Complete (hrs)**	**Success**
Create class skeletons	Owen	6	successful creation of classes
Populate repo class functions	Owen	1	repo class functions work as expected
Create a test skeleton (Log File, coverage)	Mohammed	3	test function that can be used to test program
CLI Handler	Ryan	8	Working CLI w/ correct formatting
CLI Installer	Mohammed	2	File created that will install all necessary libraries
Look into permissions, getting values from API	Ryan	2	Understand why we can't access certain information
Dummy values in calc_handler	Owen	1	Ability to test program with dummy values

Week 3 Implementing:

Task	**Owner**	**Time to Complete (hrs)**	**Success**
Calc_handler created	TBD	15	3 out of 5 score functions complete and tested
Clone repository	TBD	10	Ability to clone and parse repository

system, parse			
Parsing licenses	TBD	4	Ability to score and parse the licenses
15 test cases implemented to test coverage	TBD	3	15 test cases are created that cover 80% or more of code

Week 4 Validation/Delivery:

Task	**Owner**	**Time to Complete (hrs)**	**Success**
Calc_handler completed and tested	TBD	TBD	Calc_handler correctly returns all values as needed from API
Package up final product	TBD	TBD	Created .exe file that allows for interacting with program
Ensure code is well documented	TBD	TBD	Descriptive comments clearing stating what is happening in that portion of code
Complete README, configuration steps and how to use our program	TBD	TBD	Completed README that explains all necessary steps to use program
20 Test cases completed	TBD	TBD	20 test cases are created that cover 80% or more of code

Validation and Assessment plan

Requirement Category	**Requirements**	**Validation**
Documentation requirements	● Code is well documented and readme clearly documents the proper usage of the program. ● Each class and their respective methods have comments that clearly define their purposes.	README covers all necessary and needed external actions from the user. README explains how to interact and properly use the program
Program Requirements	● Program takes less than a minute for 10 URLS between starting execution and printing the score of each URL. ● An executable file called "run" is located in the root directory of the project. ● Program will install dependencies when "./run install" is executed within the root directory of the project and exit 0. ● Program will analyze a variable-length list of URLs when the command "./run URL_FILE" is executed, where URL_FILE is the list of URLs, and then exit 0. ● The list of URLs is formatted as an ASCII-encoded newline-delimited string. ● Program analyzes packages under the npmjs.com domain or the GitHub domain. ● Program should access the GitHub API programmatically ● Program calculates 5 separate metrics-- each metric returning a score in the interval [0, 1]-- and a net score. The five metrics are listed below.	● Program will be considered to satisfy Sarah's requirement when: ○ "./run URL_LIST" takes less than 1 minute per 10 URLs, and should print the respective scores to the terminal window ○ The program should not crash for any URL input ○ Each score will be a number between 0 and 1 ○ Program successfully prints in order from most to least trustworthy

	○ Ramp up score ○ Correctness score ○ Bus factor score ○ Responsive maintainer score ○ License score ● At least one metric includes data from the source code repository. ● Net score is a weighted sum within the interval of [0, 1] with weights that reflect Sarah's priorities. ● Program prints them in the following format: "URL_1 net_score ramp_up_score correctness_score bus_factor_score responsive_maintainer_score license_score" ● Program prints URL scores in the order of trustworthiness (highest net score to lowest net score). ● Program does not upload tokens to a publicly-visible location.	
Testing requirements	● Program will execute a test suite when the command "./run test" is entered on the command line. ● Program has a corresponding test suite with at least 20 test cases. ● Test suite achieves 80% code (line) coverage. ● Each test case within the test suite prints results in the form "X/Y test cases passed. Z% line coverage achieved." ● Software will produce a log file in the location $LOG_FILE. ● Verbosity of log file is described by the environment variable	● Program will be considered to satisfy Sarah's requirement when: ○ ./run test runs the test suite ○ There are at least 20 test cases ○ Test cases achieve 80% line coverage in coverage file ○ Each test case prints output to stdout in the form "X/Y test cases passed. Z% line coverage achieved." ○ Debug and informational messages are printed to a log file.

	$LOG_LEVEL, where a value of 0 is silent, 1 is information messages, 2 is debug messages. ● Program should include descriptive informational and debug messages.	
Score requirements		
Ramp-up score requirements	● Ramp-up score should reflect the learning curve of respective module: ○ Ramp-up factor should depend on descriptiveness of module wiki (larger wiki corresponds to higher score). ○ Ramp-up factor should depend on ratio of the number of lines of comments to the total number of source lines of code (slocs)	Proper implementation of metrics in software.
Correctness score requirements	● Correctness should reflect the overall correctness of the program: ○ High level of correctness should correspond with a low number of outstanding issues. ○ High level of correctness should correlate with the date of the latest release of the module. ○ Patches are reviewed by a high number of people before they're implemented. ○ Higher number of stars/downloads contributes to a higher correctness.	Proper implementation of metrics in software.

	○ Module has been checked for correctness by a third party and has passed	
Bus-factor score requirements	● Bus-factor score should reflect the number of critical developers on the project: ○ Large group of contributors reflects a better bus factor. ○ Higher number of contributors that contributed to the last release corresponds to a better bus factor. ○ An even workload distribution among developers reflects a better bus factor.	Proper implementation of metrics in software.
Responsive maintainer score requirements	● Responsive maintainer score should reflect the time it would take to post a security patch to the module: ○ Responsive maintainer score should correspond to the time between releases ○ The turnaround time for resolving issues should be low. ○ There should be a high number of contributors in the project.	Proper implementation of metrics in software.
License compatibility score requirements	● License compatibility score requirement should reflect the compatibility of the license with the LGPLv2.1 license: ○ License is one of the licenses listed in this table: (see the end of the document if the link doesn't work)	Proper implementation of metrics in software.

Appendix:

Licenses compatible with LGPLv2.1:

GNU General Public License (GPL) version 3 (#GNUGPL) (#GNUGPLv3)
GNU General Public License (GPL) version 2 (#GPLv2)
GNU Lesser General Public License (LGPL) version 3 (#LGPL) (#LGPLv3)
GNU Lesser General Public License (LGPL) version 2.1 (#LGPLv2.1)
GNU All-Permissive License (#GNUAllPermissive)
Artistic License 2.0 (#ArtisticLicense2)
Clarified Artistic License (#ClarifiedArtistic)
Berkeley Database License (a.k.a. the Sleepycat Software Product License) (#BerkeleyDB)
Boost Software License (#boost)
Modified BSD license (#ModifiedBSD)
CeCILL version 2 (#CeCILL)
The Clear BSD License (#clearbsd)
Cryptix General License (#CryptixGeneralLicense)
eCos license version 2.0 (#eCos2.0)
Educational Community License 2.0 (#ECL2.0)
Eiffel Forum License, version 2 (#Eiffel)
EU DataGrid Software License (#EUDataGrid)
Expat License (#Expat)
FreeBSD license (#FreeBSD)
Freetype Project License (#freetype)
Historical Permission Notice and Disclaimer (#HPND)
License of the iMatix Standard Function Library (#iMatix)
License of imlib2 (#imlib)
Independent JPEG Group License (#ijg)
Intel Open Source License (#intel)
NCSA/University of Illinois Open Source License (#NCSA)
License of Netscape JavaScript (#NetscapeJavaScript)
OpenLDAP License, Version 2.7 (#newOpenLDAP)
License of Perl 5 and below (#PerlLicense)
Public Domain (#PublicDomain)

License of Python 2.0.1, 2.1.1, and newer versions (#Python)
License of Python 1.6a2 and earlier versions (#Python1.6a2)
License of Ruby (#Ruby)
SGI Free Software License B, version 2.0 (#SGIFreeB)
Standard ML of New Jersey Copyright License (#StandardMLofNJ)
Unicode, Inc. License Agreement for Data Files and Software (#Unicode)
Universal Permissive License (UPL) (#UPL)
The Unlicense (#Unlicense)
License of Vim, Version 6.1 or later (#Vim)
W3C Software Notice and License (#W3C)
License of WebM (#WebM)
WTFPL, Version 2 (#WTFPL)
WxWidgets Library License (#Wx)
WxWindows Library License (#Wxwind)
X11 License (#X11License)
XFree86 1.1 License (#XFree861.1License)
License of ZLib (#ZLib)
Zope Public License, versions 2.0 and 2.1 (#Zope2.0)