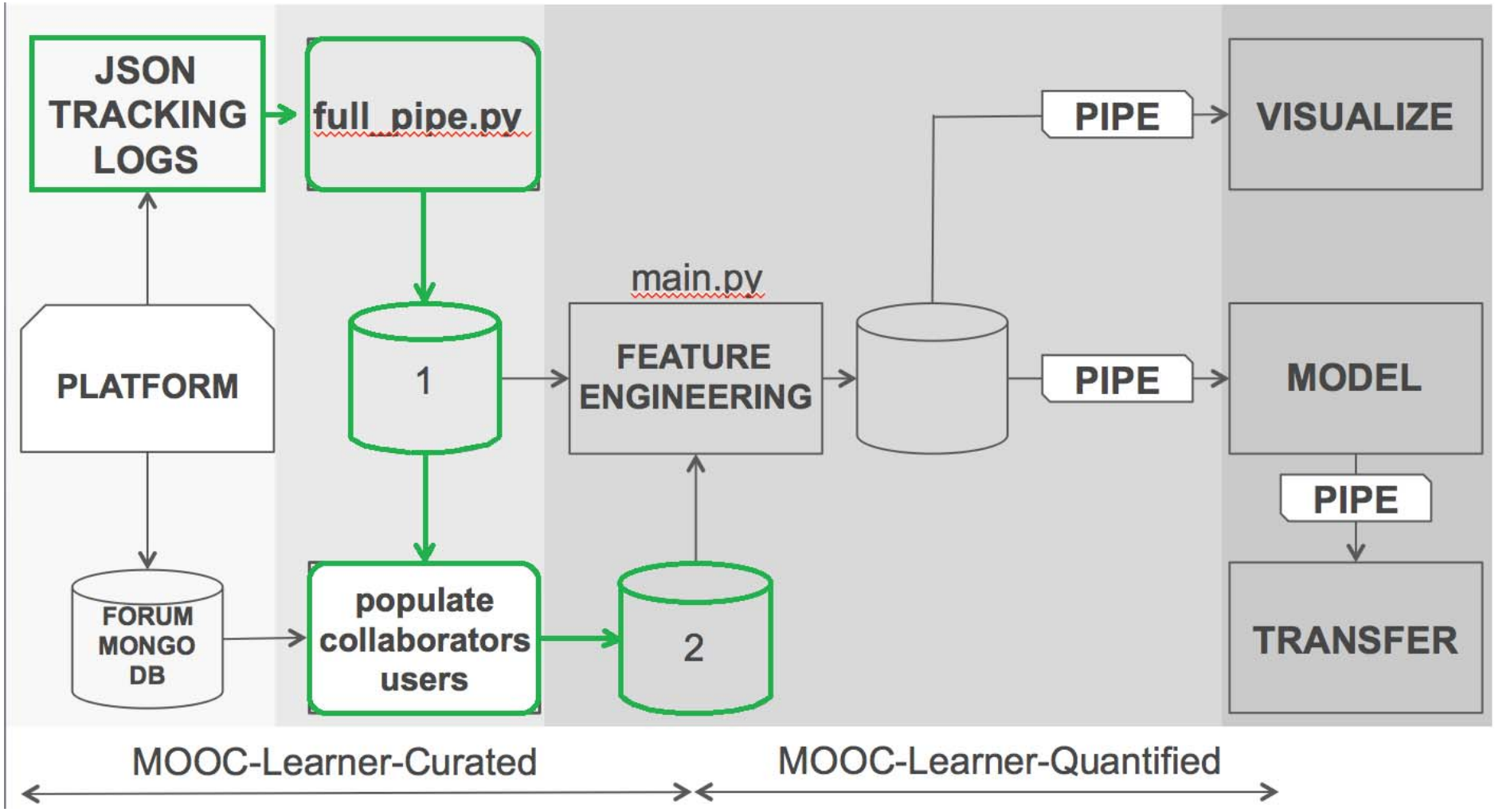
# Extending MOOCdb Slide Deck

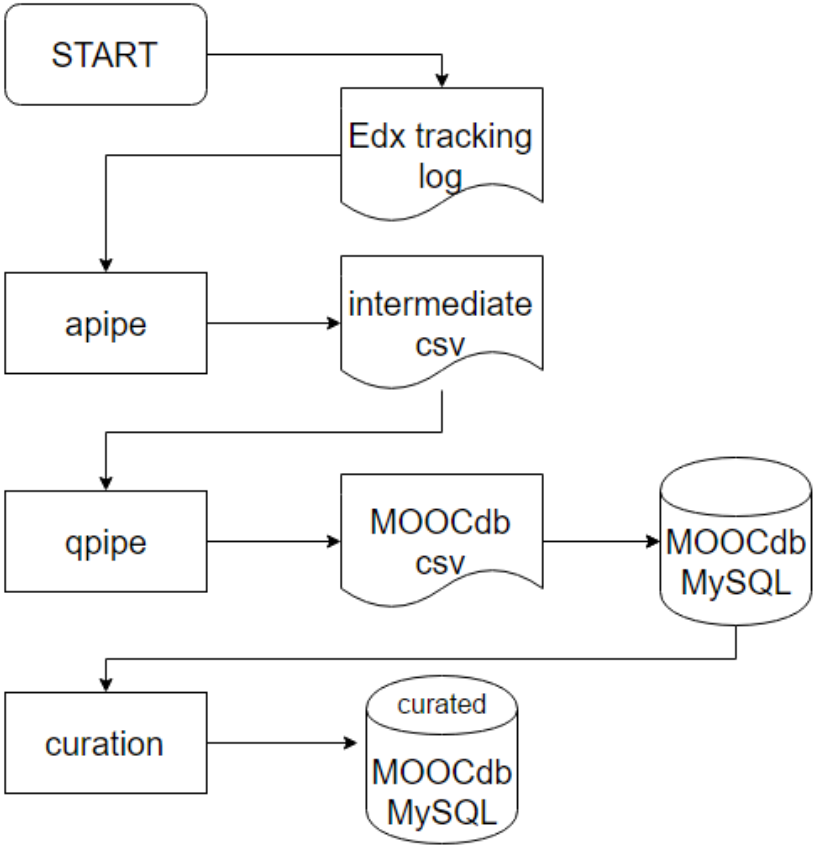
- You'll notice that all the references to the most recent version of translation software are for v1.5 which is the pre-release which was pushed soon after the thesis was turned in. No major changes were made that invalidate the figures (or they have been re-done to reflect any minor differences).
- References in the "Measuring Improvements" section to v1.4.X are intentional since that was the version of discussion in communication with the VisMOOC team.
- Not all figures included here are actually on the final poster slide but they're included as extra potentially useful raw materials for you
- The user-facing schema on page 6 does differ from that of v1.4.4 however (changes available on the v1.5 release notes), which is why I chose to make the poster represent v1.5. That figure didn't actually make it into the final poster because it's a bit too unwieldly.
- The actual poster slide is on the last slide of this deck, as requested.

# Workflow Curate Flowchart

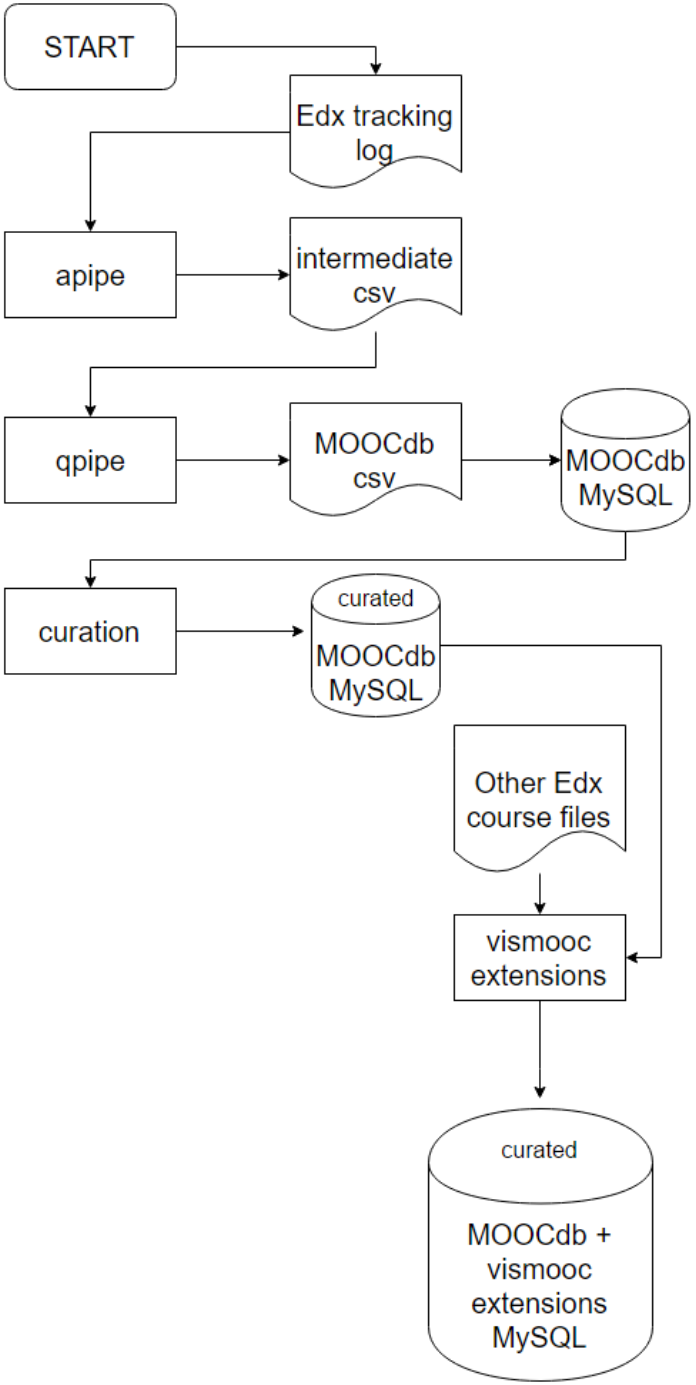


# Code Flowchart

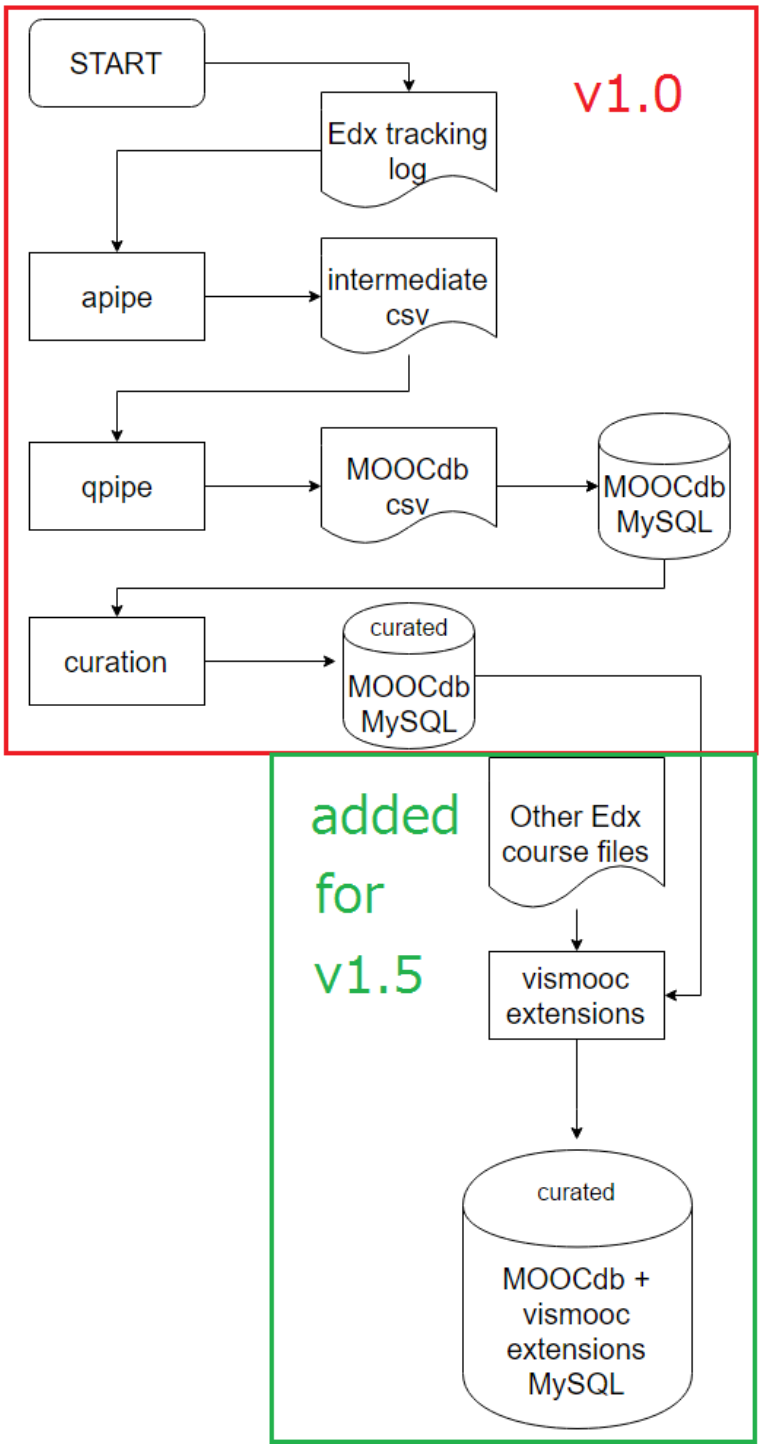
translation
software v1.0




translation
software v1.5

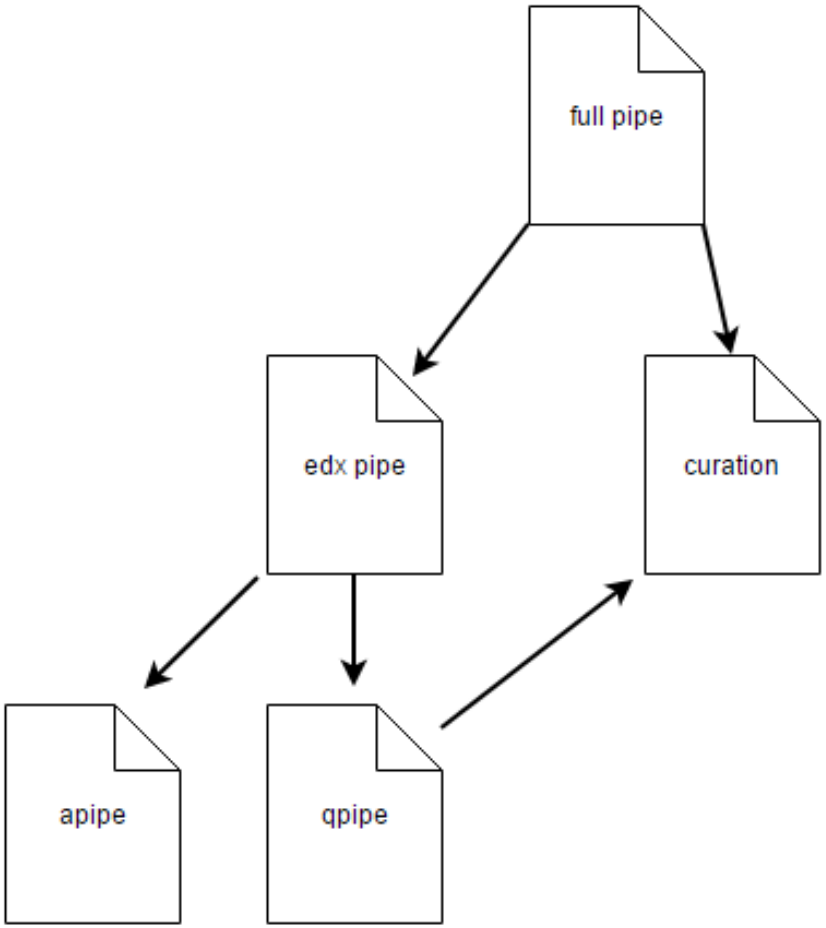


# Code Flowchart annotated

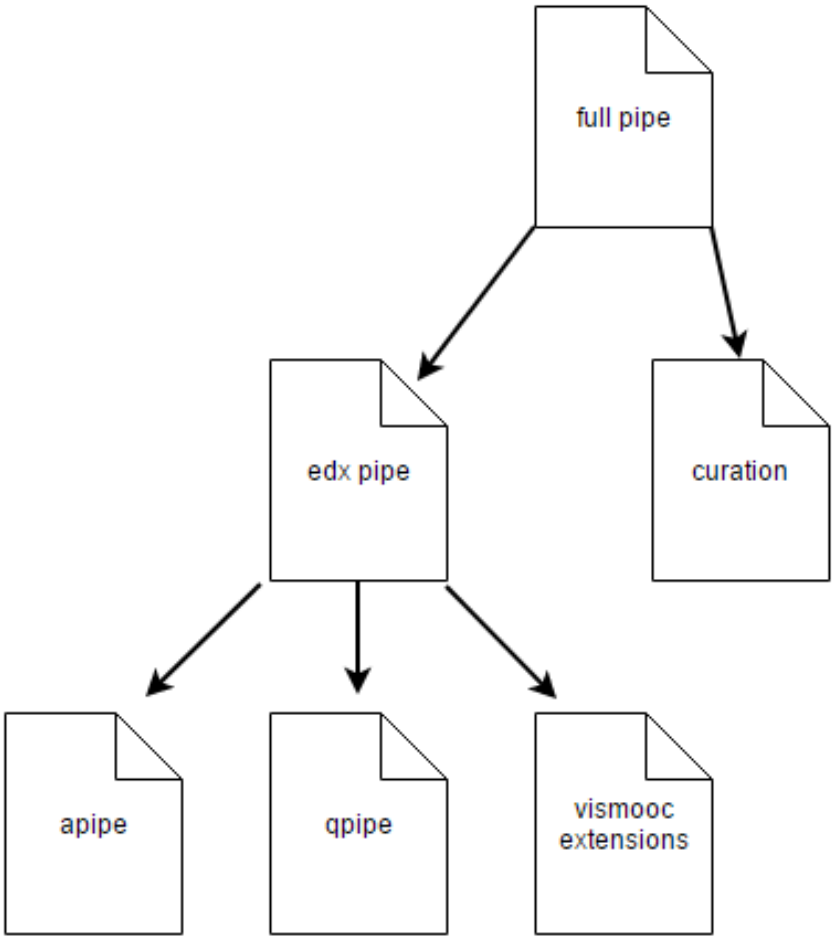


# Code Organization

translation
software v1.0



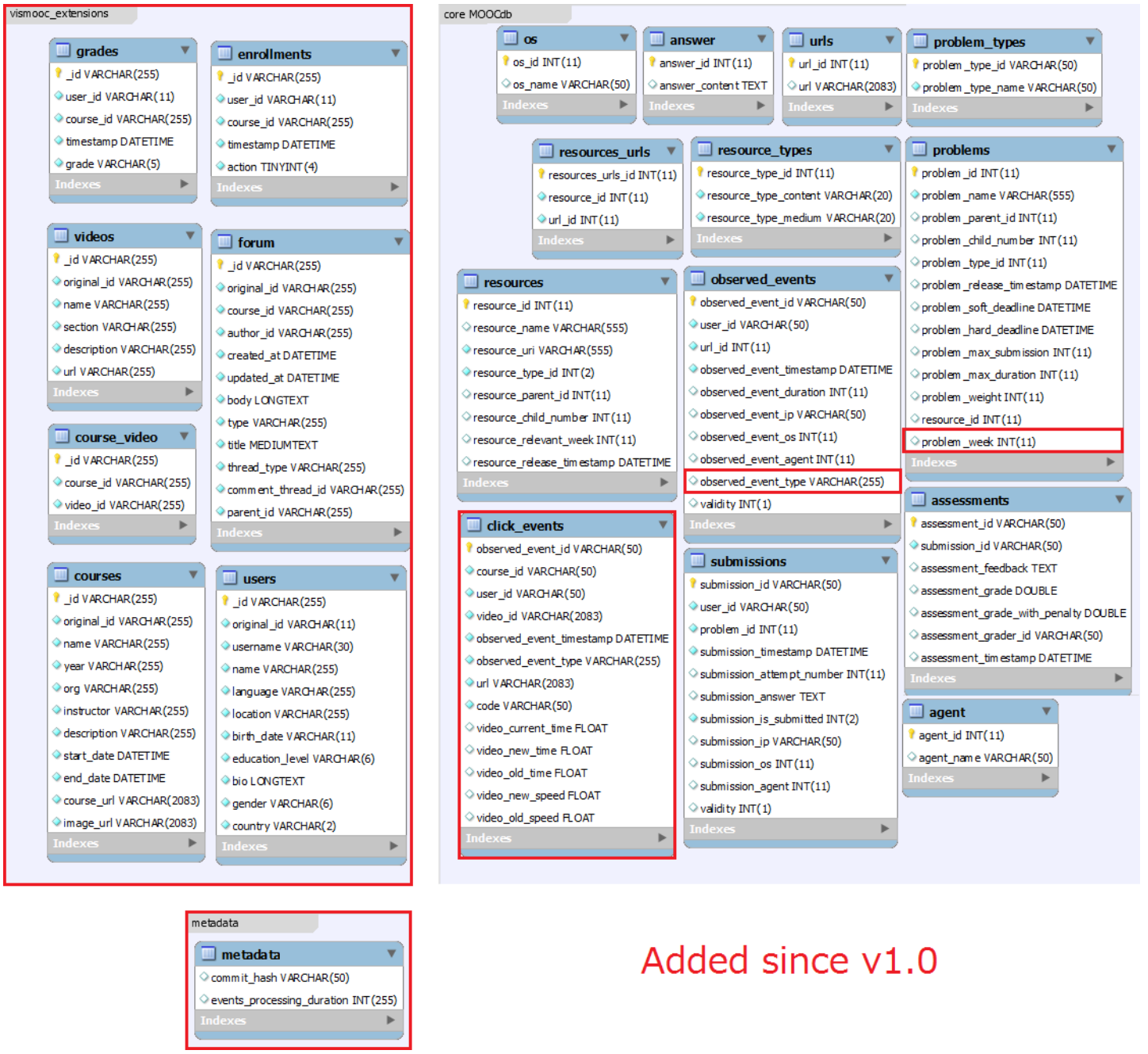

translation
software v1.5



# Schema

translation software v1.5 EER

(i.e. user facing schema)

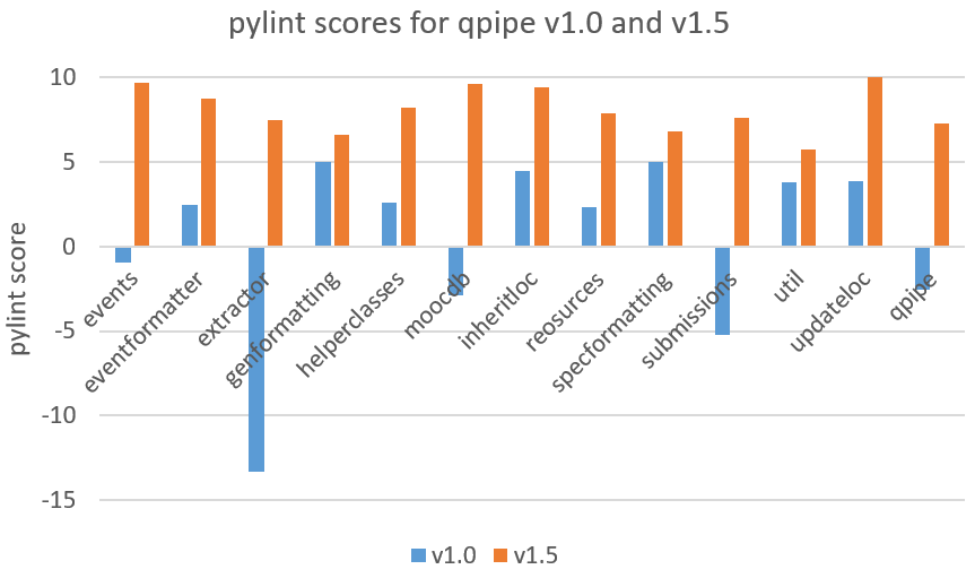


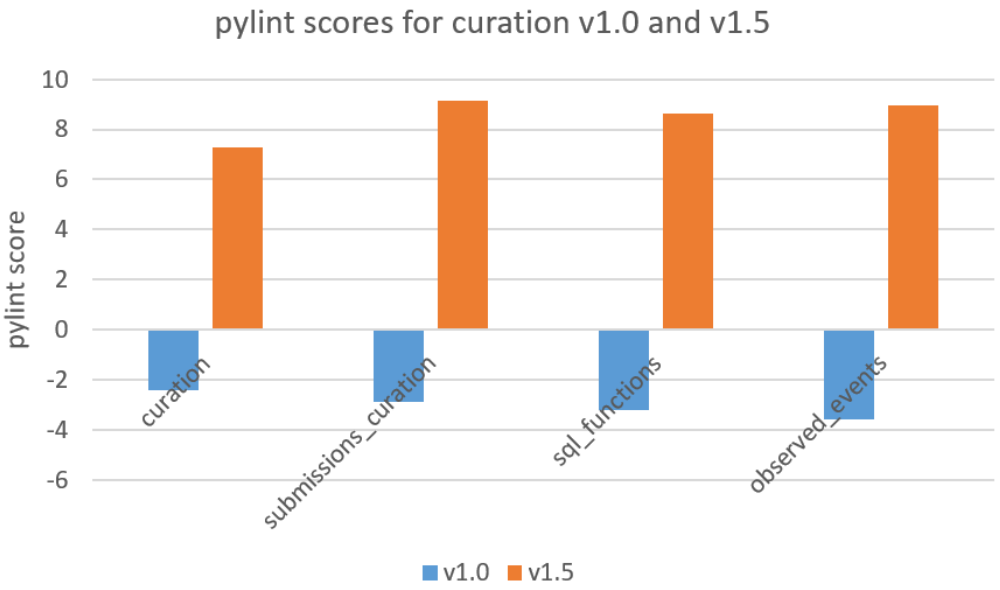
# pylint tables

| module | v1.0 | v1.5 |
|---|---|---|
| events | -0.98 | 9.69 |
| eventformatter | 2.46 | 8.78 |
| extractor | -13.33 | 7.5 |
| genformatting | 5 | 6.58 |
| helperclasses | 2.61 | 8.2 |
| moocdb | -2.92 | 9.6 |
| inheritloc | 4.44 | 9.44 |
| reosurces | 2.31 | 7.86 |
| specformatting | 5 | 6.82 |
| submissions | -5.2 | 7.62 |
| util | 3.83 | 5.71 |
| updateloc | 3.89 | 10 |
| qpipe | -2.54 | 7.27 |

| module | v1.0 | v1.5 |
|---|---|---|
| curation | -2.41 | 7.27 |
| submissions_curation | -2.88 | 9.15 |
| sql_functions | -3.22 | 8.61 |
| observed_events | -3.61 | 8.97 |

# pylint plots



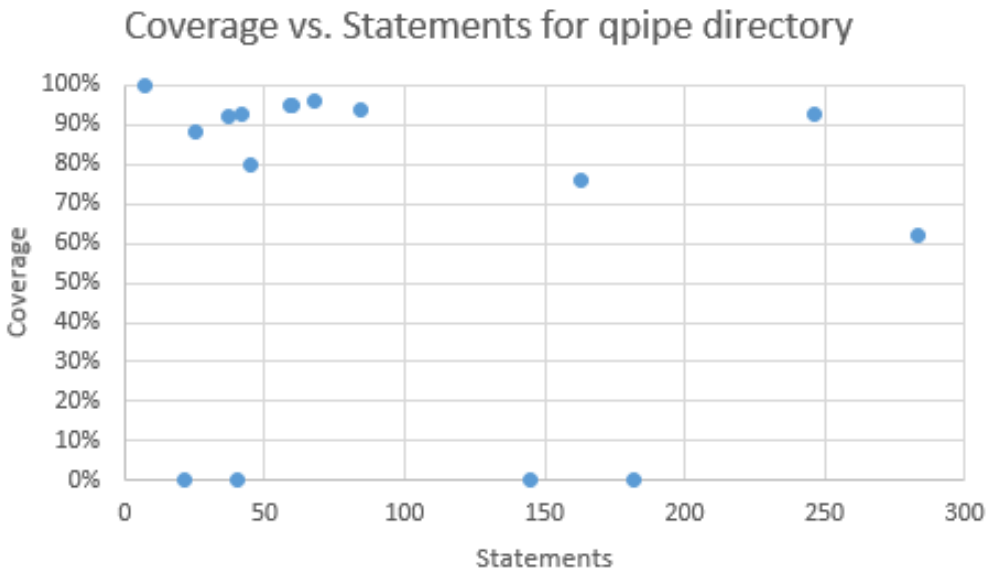



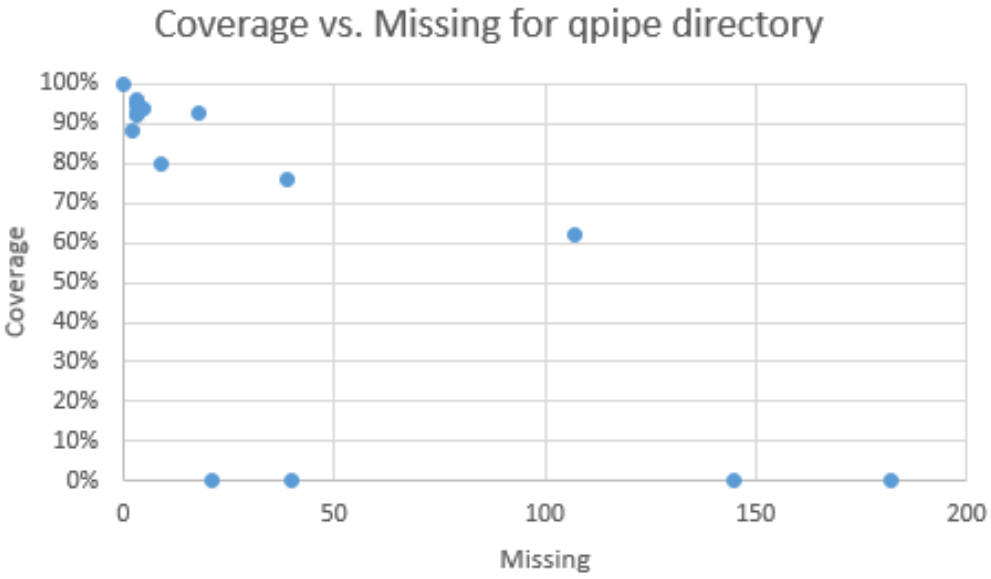
# Coverage tables

| Module | Statements | Missing | Coverage |
|---|---|---|---|
| clickevents | 60 | 3 | 95% |
| eventformatter | 163 | 39 | 76% |
| eventmanager | 45 | 9 | 80% |
| events | 246 | 18 | 93% |
| extractor | 40 | 40 | 0% |
| genformatting | 84 | 5 | 94% |
| helperclasses | 283 | 107 | 62% |
| inheritloc | 42 | 3 | 93% |
| moocdb | 25 | 2 | 88% |
| qpipe | 145 | 145 | 0% |
| resources | 182 | 182 | 0% |
| specformatting | 59 | 3 | 95% |
| submissions | 21 | 21 | 0% |
| updateloc | 37 | 3 | 92% |
| userids | 68 | 3 | 96% |
| util | 7 | 0 | 100% |
| Total | 1532 | 584 | 62% |

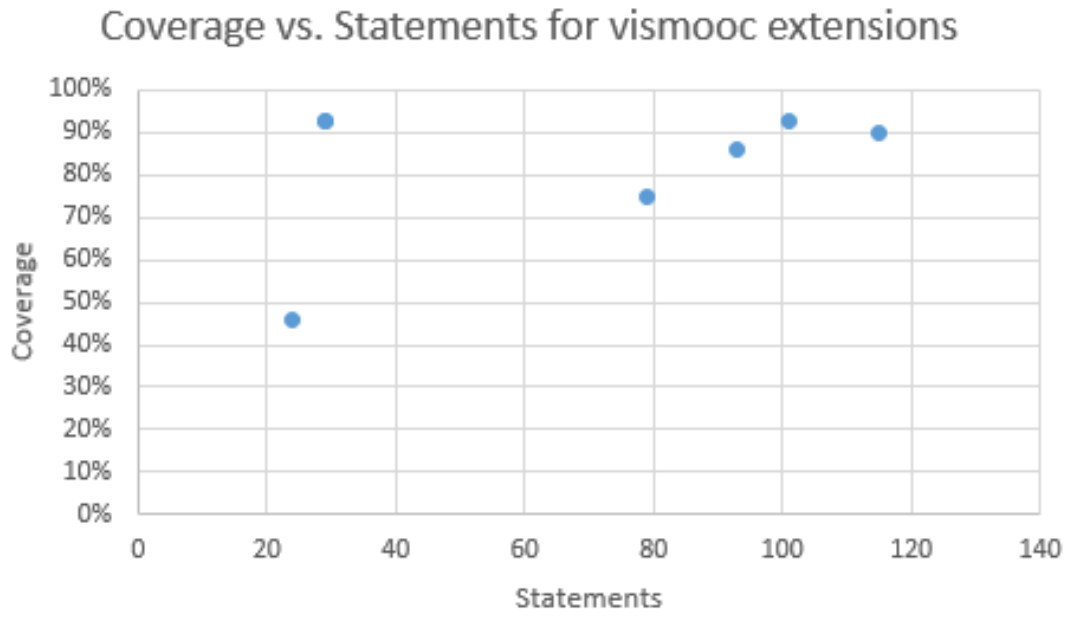
| Module | Statements | Missing | Coverage |
|---|---|---|---|
| courses | 115 | 12 | 90% |
| enrollments | 29 | 2 | 93% |
| forum | 101 | 7 | 93% |
| grades | 29 | 2 | 93% |
| users | 79 | 20 | 75% |
| util.py | 24 | 13 | 46% |
| videos | 93 | 14 | 86% |
| Total | 476 | 76 | 84% |

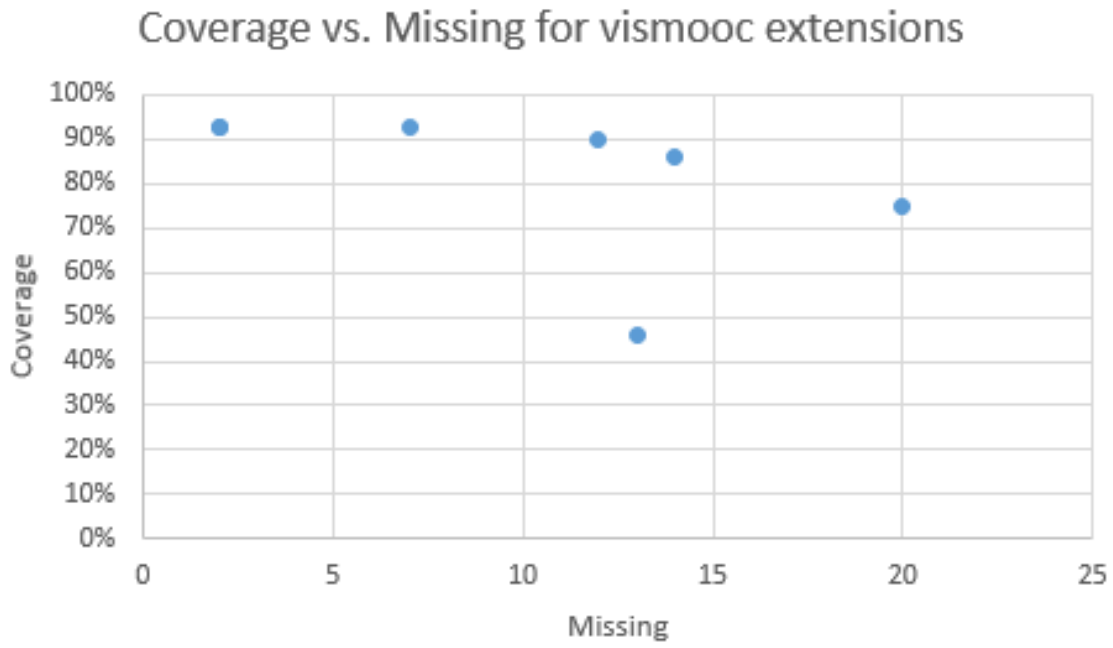
# Coverage qpipe



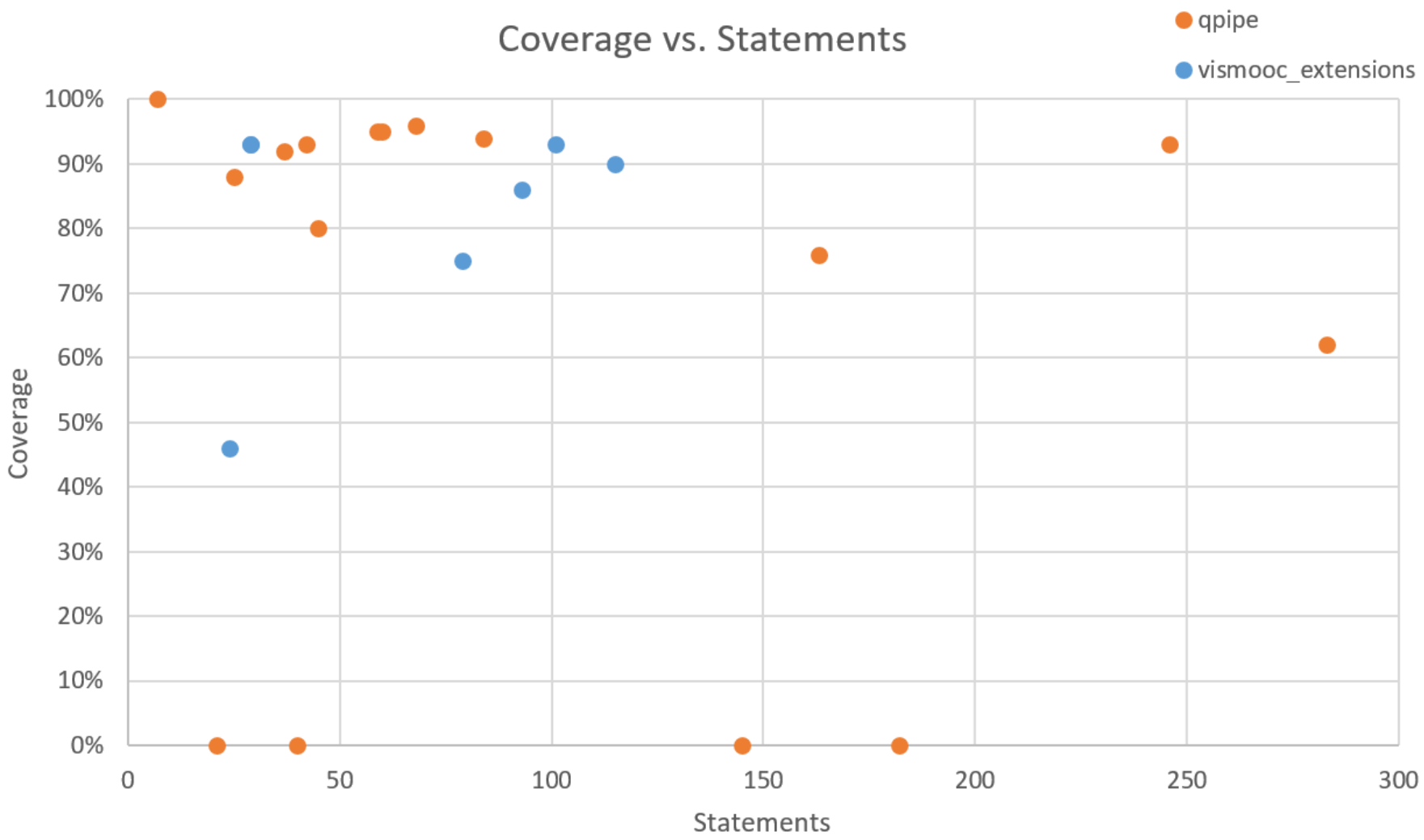



# Coverage vismooc extensions



# Coverage qpipe and vismooc



# Extending MOOCdb

Austin Liew, MEng, ALFA Group
CSAIL MIT

## Abstract

The MOOCdb project aims to unify course data from different MOOC (Massive Open Online Course) providers into a single format named MOOCdb [1]. However, the translation software [2][3] under the MOOCdb project lacked stability and up to date documentation. Furthermore, we were tasked with implementing new functionality in order to interface with HKUST's VisMOOC software [4]. This study presents the application of the LCCA (Legacy Code Change Algorithm) [5] on the legacy MOOCdb translation software project to solve its code rot and make it maintainable. The evaluation is based on the rate at which issues were resolved during different stages of LCCA, and an analysis of the key differences after the application of the LCCA. We see that the project saw much more frequent changes in response to discovered issues once the project was significantly refactored and restructured, and that the stability is greatly increased with the addition of a test harness to the core modules. Such changes enabled industry accepted good practices and alleviate the bulk of the code rot that prevented the project from moving forward.

## Evolution of the Translation Software

**Code Organization:**
Originally the translation software was organized as in **Figure 5**. We wanted to break any dependencies from lower levels of the directory tree to higher levels. Specifically, qpipe originally imported curation code which sat higher up in the project hierarchy. However given that curation by design is able to operate on any MOOCdb MySQL instance, the code in qpipe related to curation was moved up the project hierarchy which eliminated the import dependency to change the project structure to that of **Figure 6**.

**Code Formatting:**
The formatting of the code throughout all modules in the translation software was inconsistent. For the continued maintenance of the project it was important to try and unify as much as possible the formatting of the code throughout the project. The first major step taken was to run an automatic refactoring tool yapf [6] which is specifically a formatter for Python source files. Besides using yapf, minor tweaks were made by hand, such as removing superfluous whitespace.

**Metadata Table and Test Harness:**
A modification was made to the translation software so that upon creating a MOOCdb MySQL instance, a metadata table is also added that contains the commit hash of the translation software used to generate the MOOCdb MySQL instance. Attaching the commit hash to the MOOCdb MySQL instance also indicates the necessary versioning information to reproduce the same instance with the translation software should someone else attempt to do so. A sample generation tool and a test harness was also added for qpipe modules to improve the stability of the project, as per LCCA.

**vismooc_extensions:**
Since MOOCdb's ultimate goal is to act as a unifying MOOC data format, there will inevitably be shortcomings that different third party groups will want to address via their respective modifications to the translation software. We introduced the concept of extensions as our choice for providing separation from the core translation software logic while still being easy to interface with the translation software. Using the TDD methodology [7], vismooc_extensions modules were implemented for translation software v1.5 now generates tables for grades, enrollments, videos, course_video, courses, forum and users tables to interface with VisMOOC. **Figure 4** shows the added pipe for the vismooc_extensions modules in relation to core MOOCdb. A test harness was also added for the vismooc_extensions modules, as per the TDD methodology.

**Figure 1.** pylint score comparison for qpipe

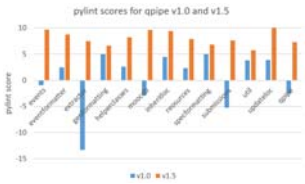
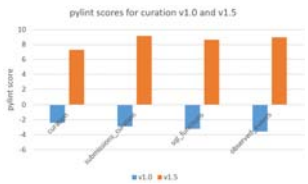



**Figure 2.** pylint score comparison for curation




## Measuring Improvements

**Understandability**:
We use pylint [8] scores to get a quantitative estimate of readability (and hence understandability). pylint is a static code checker for Python that analyzes code for violations of defined style guides. We ran pylint on v1.0 and v1.5 of translation software to get a comparison of readability. **Figures 1 and 2** summarize the score differences between selected modules from translation software v1.0 and translation software v1.5. The huge improvement from extractor was a consequence of eliminating non-standard Python class attribution setters and getters. Of note 9 modules have changed from negative to positive scores. As indicated by the table, v1.5 scored much higher than v1.0 across the board, with fairly large score differentials for most of the individual modules. This gives us high confidence that v1.5 is much more readable than v1.0.

**Resistance to Change:**
given the back and forth with the VisMOOC team from versions v1.4.0 to v1.4.4, which was developed over a fairly tight 1 to 2 week period, we think that the current version of the translation software has a much lower resistance to change compared to v1.0. v1.0 was littered with numerous syntax error or unclear requirements for input filenames and input directory structure. After combing through the code to resolve those issues just to get the code to run, it took a few weeks before an actual end-to-end execution of translation software v1.0 could even be reached. At this point, even though translation software v1.0 could be execute it took a whole day to actually complete because it would process an entire semester of Edx course data. Hence it took a whole day just to even run v1.0 to see if changes worked as intended. On the other hand, the sample generation tool and fixes of v1.4 enabled much quicker and bug-free end-to-end executions. The quicker executions finish in less than a minute on generated samples and new bugs can be detected and squashed in a matter of minutes as a result of this much shorter development cycle.

**Stability:**
A strong test harness is necessary for ensuring that a software project adheres to the expected outputs for a particular set of inputs. The coverage data for both qpipe and vismooc_extensions is plotted in **Figure 3**. qpipe now has 9 module tests with 54 unit tests spread across them. vismooc_extensions has 6 module tests with 56 unit tests spread across them. There is no comparison to v1.0 because all of its tests either had syntax errors that prevented it from executing, or did not have any assertions to automatically check output. Based on brief comments it appears that many of the tests were designed in such a way that a developer just observed console output to see if it looked reasonable. Hence v1.0 had effectively 0 unit tests (and hence 0% coverage). Noticeably, the vismooc_extensions coverage data is high across the board regardless of the number of statements. This is because this was new code developed mostly using TDD which requires tests to be written before implementation.

**Reproducibility:**
Reproducibility is also crucial for debugging errors that were not covered by an existing test suite. Hence, our stance is that reproducibility is still important means of supporting the claims of academic work, and that is why the metadata table was added. During collaboration towards the end of this study the VisMOOC team found that our v1.4.0 release which implemented the prototype of the vismooc_extensions containing an implementation of the courses module actually crashed on qpipe and curation on different Edx courses' respective tracking logs. The VisMOOC team sent us console output error logs and sample inputs that created the error and we were able to successfully reproduce the exact same error logs on our side. Once we were able to do this, tracking down the source of the bug became much quicker.

**Figure 3.** Coverage vs. Statements Scatterplot for qpipe and vismooc extensions

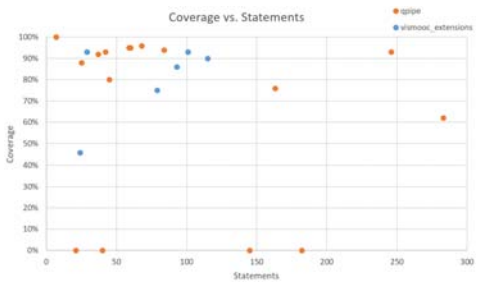



**Figure 4.** Evolution of Translation Project Functionality

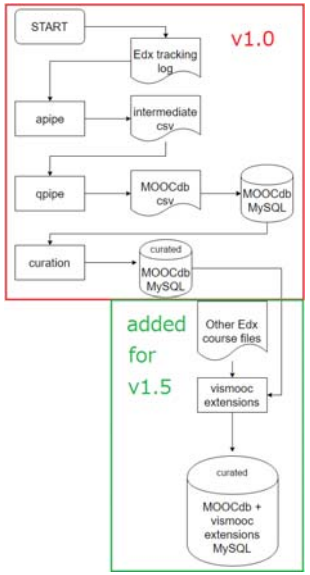



**Figure 5.** translation project v1.0 organization

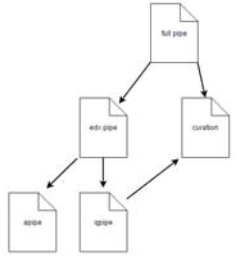
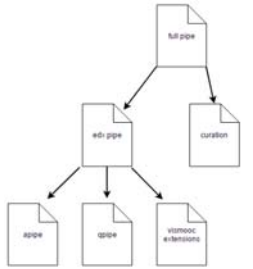



**Figure 6.** translation project v1.5 organization




## Conclusions

This study has presented an application of LCCA on the legacy translation software project developed originally as part of the MOOCdb project. We find that by following LCCA, we were able to improve the understandability, stability, and reproducibility of the MOOCdb translation software while also reducing its resistance to change. We see the overall pylint score of the translation software go from -1.27 to 8.79 from v1.0 to v1.5. We also see the test count essentially go from 0 to 110 from v1.0 to v1.5. We also find that the test coverage for qpipe and vismooc_extensions is 62% and 84% respectively, a significant improvement from its previous 0%. These statistics give us confidence that the translation software has indeed improved noticeably according to our evaluation focuses.

One big lesson we learned is that automation is key for keeping code clean. Using automatic code refactoring to shift the inconsistent visual style throughout the codebase to a consistent one was much quicker and prone to less errors than trying to perform such a refactoring manually. Also, the test harness itself provides automatic verification upon simply executing the test itself. There is no room for human error on the part of manually observing the output. Since automation plays such a big factor keeping the code clean, we highly recommend the same use of automation is adopted by future maintainers of the translation software to avoid relapse into an ugly codebase like that of v1.0.

Besides that, adherence to TDD during LCCA when possible was also a big factor in cleaning up the code. The higher coverage score for vismooc_extensions was a result of following TDD for most of the development of that code. Future maintainers of the translation software are highly encouraged to follow TDD as much as possible, and otherwise use LCCA if they cannot follow TDD.

## References

1. MOOCdb. http://moocdb.csail.mit.edu
2. Austin Liew. *Overcoming Code Rot in Legacy Software Projects*. Masters thesis for the Electrical Engineering and Computer Science department of MIT, 2017.
3. Quentin Agren. *From clickstreams to learner trajectories*. Masters thesis for École Normale Supérieure de Lyon, 2014.
4. VisMOOC. http://ihome.ust.hk/~qchenah/vismooc/
5. Michael C. Feathers. (2004). *Working Effectively with Legacy Code*.
6. Yapf. https://github.com/google/yapf
7. Robert Cecil Martin. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*.
8. Pylint. https://docs.pylint.org/en/1.6.0/faq.html