Structural Quality & Software Evolution

Alison Major

Lewis University

2022

Introduction

Maintainability Index and Refactor Scores

- Areas of concern: cost, timeline, quality
- Quality is hard to understand
- Pylint & Radon are a static analysis tools
- Refactor violations point out code smells

Keeping Users Engaged Long Term

Why does software evolution matter?

- Users find bugs
- Users want new features
- New security threats
- New laws from governing bodies

Need a thriving community of engaged users in order to keep apps and games successful.

In an open source system, need a thriving community of engaged developers in order to continue evolving.

Keeping Users Engaged Long Term

How do we ensure software evolution?

Keep the project maintainable.

- Bugs should be quick and easy to fix
- New features should be easy to add
- Consistent standards (naming, small methods, etc)

Software Maintenance

Large portion of project cost in a typical software system is in the maintenance phase.

The Impact of Structural Quality

Measuring Maintainability

- Easy to maintain = Easy to evolve
- Pylint & Radon Maintainability Index (MI)
- PEP 8 is a set of Python standards
- Refactor Messages (Pylint)
  - Refactor warnings are generally "code smells"
  - Code smells point out problems in Architecture

The Impact of Structural Quality

Other Maintainability Characteristics

- Low coupling, high cohesion
- Readability
  - Big commits reduce maintainability
  - PEP 8 enforces readability
- Confidence that metrics around software structure provide value in keeping systems maintainable (and therefore can evolve)

The Impact of Structural Quality

Documentation and Maintainability

- Documentation holds the results of significant design decisions
- Can influence the ability to evolve because. . .
  - Enhances code understanding
  - Comprehensibility impacts maintainability in a positive way

Related Work

Design Patterns and Software Quality

- Design patterns provide flexibility
- Classes with frequent changes are either...
  - Easy to extend (okay)

    ...or...
  - Correlate to other classes (high coupling... red flag!)
- We look at refactor score (code smell) not error score (bugs)

Keeping this in mind, we focus on *changes for system extensions and adaptiation*, not bug fixes.

Related Work

Software Architecture and Maintainability

- Maintainability
- Extensibility
- Simplicity, understanding
- Re-usability
- Performance

Keep these in mind for easier future development when adding or changing code.

Methodology

Initial Repository Set

- Popular
- Long development history
- Multiple release cycles

Filtered Respository Set

At least 80% Python code and top 20th percentile in these categories:

- Long history of commits (2,968+ commits)
- Large number of contributors (90+ contributors)
- Many releases (44+ releases)
- Substantial Age (66.4+ months)

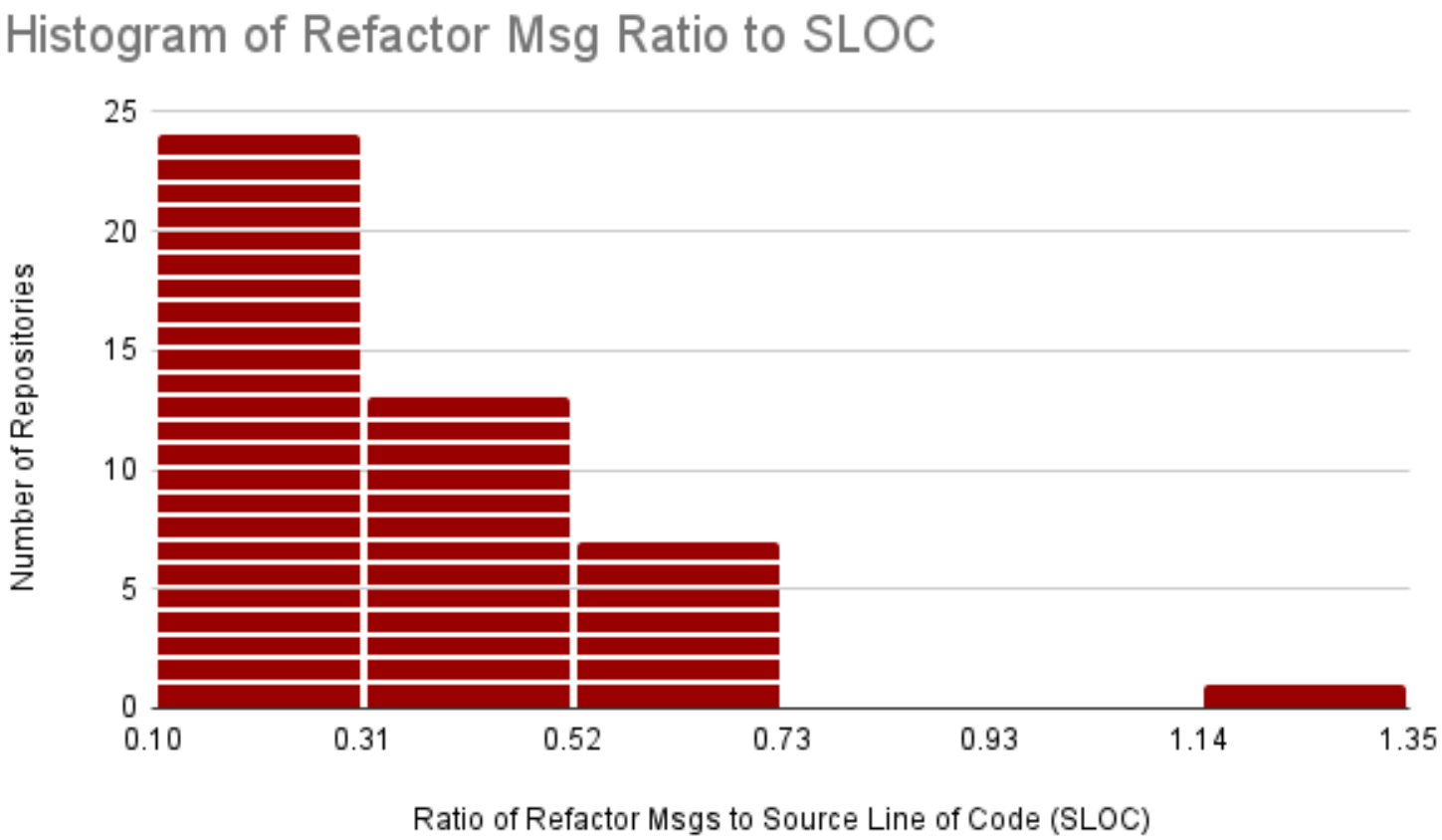
Results in 46 repositories for further research.

Results

- Radon MI for all repositories rank as grade "A" which is considered "very high maintainability"
- Open source systems with engaged community of developers tend to have higher scores
- For comparison, calculated ratio of refactor message count to SLOC as well as the average MI for a project.

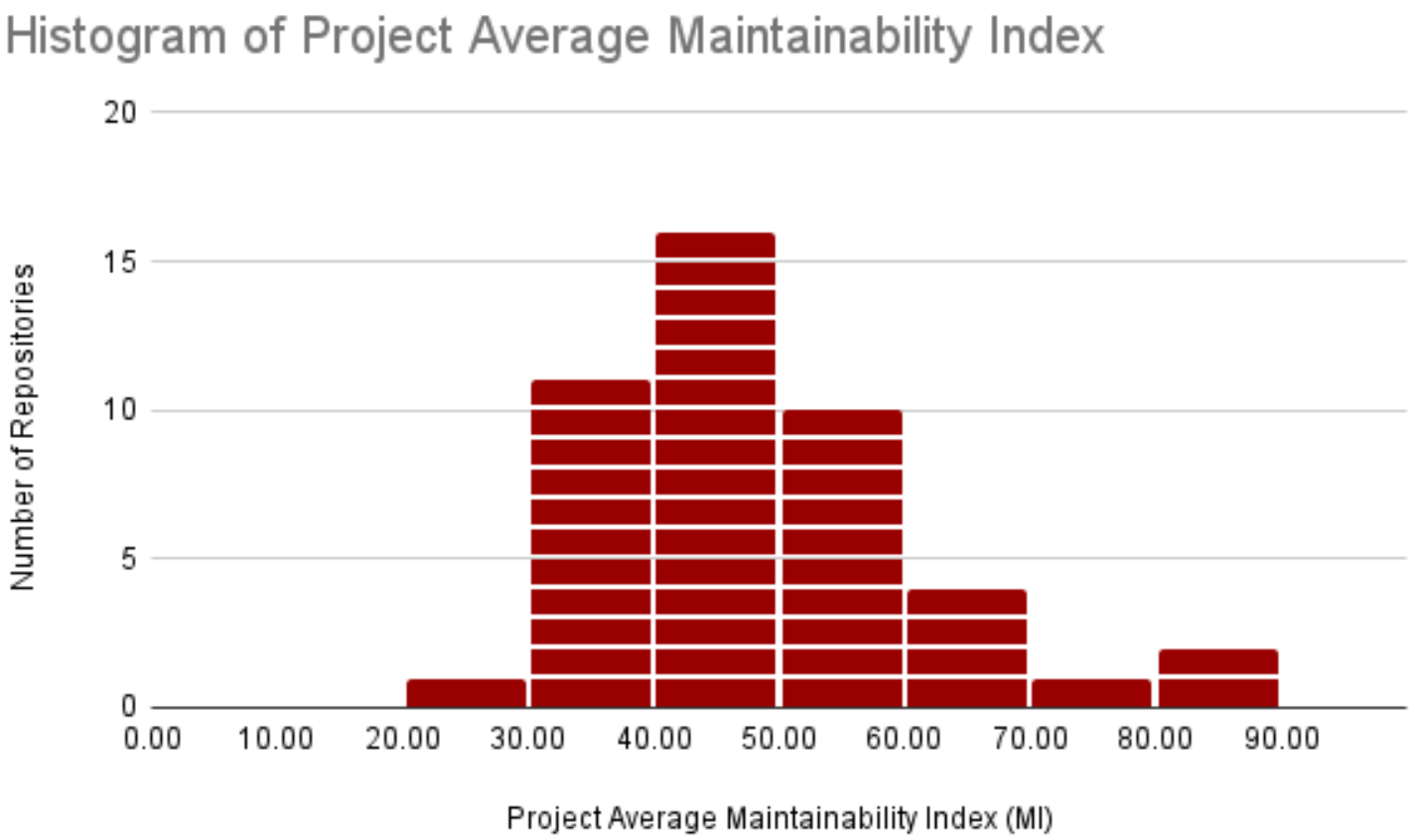
Repo	Ratio	Avg MI	Status
cython	0.14	31.0	active
youtube-dl	0.15	54.16	active
electrum	0.16	39.41	active
numba	0.62	62.55	active
scrapy	0.64	64.47	active
raven-python	1.35	87.02	deprecated

Results



Diligent development communities can keep refactor warnings low, regardless of system size (lines of code).

Results



Many repositories average in the mid-score to high-score.
Radon considers 20 points and up to be very maintainable.

Conclusions

- Structural quality impacts software evolution.
- Good projects will grow and evolve.
- Poor structure leads to deprecation.
  - If the development community is engaged, deprecation of the project may lead to a fresh, improved code base.
- Open source and projects with many contributors are vulnerable to degrading maintainability.
  - Popular repositories with a long history of commits and releases (i.e. our repository data set) tend to have good maintainability.
  - The high maintainability is a testament to their longevity.

Recommendations

- Good architecture is important for evolution.
- Reliable quality metric can be a useful way to measure maintainability, which promotes ability to evolve a project.
- Pick a set of standards to maintain good architecture even with a large, open source community.
  - Limit complexity as project changes and grows.
  - SOLID principles.
  - Keep it DRY.
  - And other design patterns known to be best practice.
- Auto-enforce by using quality measurements for desired standards in the project's CI/CD pipeline.