# Structural Quality & Software Evolution

Alison Major

Lewis University

2022

# Introduction

Maintainability Index and Pylint Refactor Scores

1. Areas of concern: cost, timeline, quality
2. Quality is hard to understand
3. Pylint is a static analysis tool
4. Refactor violations point out code smells

# Keeping Users Engaged Long Term

Why does software evolution matter?

1. Users find bugs
2. Users want new features
3. New security threats

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

# Keeping Users Engaged Long Term

How do we ensure software evolution?

1. Keep the project maintainable
2. — bugs should be quick and easy to fix
3. — new features should be easy to add

# The Impact of Structural Quality

Software Maintenance

1. ADD TO PAPER: don't touch too many files (SOLID principal)
2. Consistent standards (naming, small methods, etc)
3. 90% cost typical software system is in the maintenance phase

# The Impact of Structural Quality

Software Evolution

1. Maintenance — bug fixes and minor functional improvements
2. Evolution — new laws from governing bodies
3. Evolution — new user needs (system must adapt!)
4. Lehman’s laws

# The Impact of Structural Quality

Measuring Maintainability

1. easy to maintain = easy to evolve
2. Pylint Maintainability Index (MI)

# The Impact of Structural Quality

Maintainability Scores

1. Refactor Score (Pylint) — code smells
2. TODO: List type of checks for Refactor
3. Code smells point out problems in Architecture
4. PEP 8 is a set of Python standards

# The Impact of Structural Quality

Other Maintainability Characteristics

1. low coupling, high cohesion
2. confidence that metrics around software structure provide value in keeping systems maintainable (and therefore can evolve)
3. readability — big commits reduce maintainability
4. PEP 8 enforces readability

# The Impact of Structural Quality

Documentation and Maintainability

1. documentation holds the results of significant design decisions
2. can influence the ability to evolve because. . .
3. — enhances code understanding
4. — comprehensibility impacts maintainability in a positive way

# Related Work

Considering Data Sets (TODO: New title?)

1. 23 — which language prone to defects
2. 24 — OOP metrics and maintainability

IS THIS SECTION WORTH HAVING?

# Related Work

Design Patterns and Software Quality

1. design patterns provide flexibility
2. classes with frequent changes
3. — easy to extend or
4. — correlates to other classes (red flag!)
5. keeping this in mind, we focus on changes for system extensions and adaptiation, not bug fixes
6. — we look at refactor score (code smell) not error score (bugs)

# Related Work

Software Architecture and Maintainability

1. maintainability
2. extensibility
3. simplicity, understanding
4. re-usability
5. performance

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

# Methodology

Initial Repository Set

1. previous project collected open source Python projects
2. — popular
3. — long development history
4. — multiple release cycles

# Methodology

Filtered Respository Set We chose only those with 80% Python and then top 20th percentile in these categories:

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

Results in 46 repositories for further research.

# Results

We found that...

# Conclusions and Recommendations

We recommend that...