Predicting Maintainability for Software Applications

Early in the Life Cycle

Cara Cuiule

ICEAA 2018

Agenda

- Introduction
- Definitions, scope of research
- Maintainability measurement models
- Effort versus maintainability
- Metrics for maintainability at the start of the life cycle
- Recommendations for future research

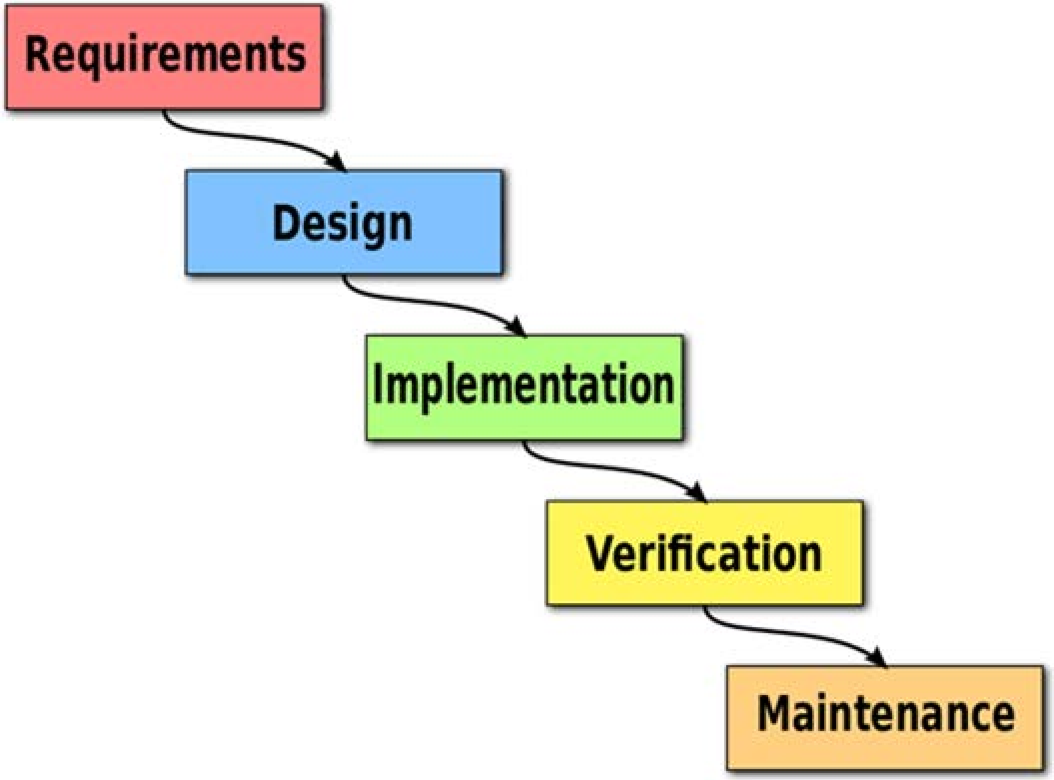
Definitions

- Maintainability – how easy it is to change a system
  - Sub-Characteristics
    - *Modularity*
    - *Reusability*
    - *Analyzability*
    - *Modifiability*
    - *Testability*
  - Internal vs. External Attribute

- Maintenance – changes made to a system after initial release
  - Corrective
  - Adaptive
  - Perfective
  - Preventive

Goal

Find maintainability models that could be measured in Requirements phase of life cycle



Maintainability Models

- No standard parametric model
  - Especially not for estimating near the beginning of the lifecycle
- Models researched used combination of the following:
  - Source Code Metrics
  - Design Metrics
  - Expert opinion

Maintainability Models that use one or more of the following:

- Source code metrics
  - Maintainability Index (Oman and Hagemeister)
- Design metrics
  - Software Maintainability Index (Muthanna et al.)
- Expert opinion
  - Software Improvement Group (SIG) Model

Source Code Models

- Maintainability Index
  - Oman and Hagemeister

$$= 171 - 5.2 * ln(aveV) - 0.23 * aveV(g') - 16.2 * ln(aveLOC) + 50.0 * sin\sqrt{2.46 * aveCM}$$

  - Inputs are average for each module:
    - *aveV = Halstead Vol*
    - *aveV(g') = McCabe's Cyclomatic Complexity*
    - *aveLOC = Lines of Code*
    - *aveCM = % of lines of comments*
  - Average maintainability between 65 and 85

Software Maintainability Index

- Developed by Muthanna et al.
- Design phase metrics
- Formula for each module:

$$SMI = 125 - 3.989 * FAN - 0.954 * DF - 1.123 * MC$$

- *SMI = Software Maintainability Index (0-125)*
- *FAN = Average number of external calls*
- *DF = Data flow*
- *MC = Average McCabe Cyclomatic complexity*

Expert Opinion Models

- SIG Model
  - Developed due to criticism about MI model
  - Gives each sub characteristic of maintainability a score

ISO 9126 maintainability	source code properties: volume	complexity per unit	duplication	unit size	unit testing
analysability	x		x	x	x
changeability		x	x		
stability					x
testability		x		x	x

Conclusions From Model Research

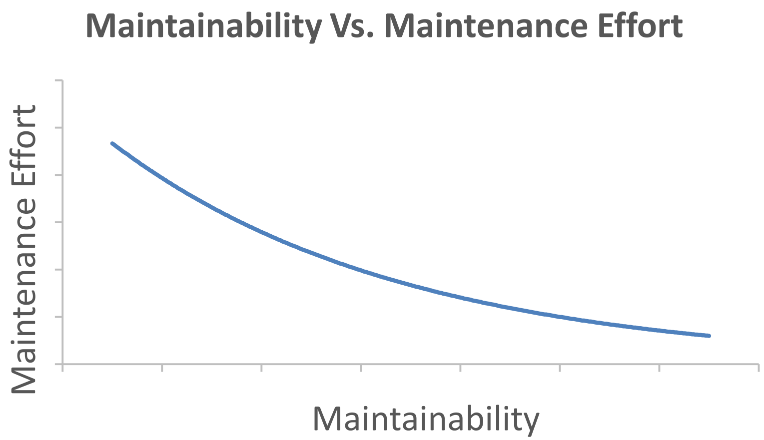
- Earliest we can use models is design phase

Goals

- Determine relationship between maintenance effort and maintainability
- Find other metrics that could be measured or estimated before implementation phase

Maintainability and Maintenance Effort

- Theory
  - Maintainability and effort have a negative relationship



- Practice
  - Experts assume the same relationship to measure maintainability
    - *Riaz et al., Hayes et al., Sjøberg et al.*
  - Studies actually tried to measure relationship
    - *Maintainability Prediction Model (MainPredMo)*
    - *SIG Model Validation*

Maintainability and Maintenance Effort, pt. 2

- Maintainability Prediction Model (MainPredMo)
  - $RDCRatio = \frac{Reqt\ Effort + Design\ Effort}{Coding\ Effort}$
  - $Maintainability = 3.795 + 1.652 * RDCRatio$
- SIG Model
  - Higher maintainability, less time for corrective maintenance and enhancements (perfective maintenance)
- For now, assume maintainability and effort have a negative relationship

Promising Metrics for the Beginning of the Life Cycle

1. Size related
2. Use of Object Oriented Language
3. Good Coding practices
4. Use of Organizational Guidelines

Size Related Metrics

- Size related metrics can be estimated very early in life cycle
  – SLOC, LOC, LLOC, Function Points, etc.

- Studies looked at LOC

- Various claims it has relationship with maintainability
  – Sjøberg et al., Heitlager et al., Nishizono et al., Riaz et al.

- Some works claim weak relationship
  – Hayes and Zhao, Hegedűs et al. (“Source Code Metrics”)

Object Oriented Language

- Object oriented language leads to system with better maintainability
  - Lim et al.
  - Dash et al.

- Unclear if this is dependent on the use of good design practices such as UML

Coding Practices

- Recommendations made by Hayes et al.
  - Good coding/architecture practices
  - Clear rules/standards
  - Focus work on important parts of system

- What are "good" coding standards?
  - No clear answer

Frameworks for Organizational Practices

- Capability Maturity Model Integration (CMMI)
  - Use in the development phase
  - Study shows it improves quality and productivity
    - ***Not** strictly software related data*
- Software Maintenance Maturity Model
  - Developed by April and Abran
  - Claims of improvement not yet backed by research

Recommendations for Future Research

- **First develop a standard maintainability metric**

- Verify relationship between maintainability & maintenance effort

- Determine how the following factors influence maintainability
  - Size
  - Object Oriented language tied with design practices
  - Coding practices
  - Organizational processes

- Realistic route: keep looking into design based metrics

Conclusion

- Studies use different ways to measure maintainability

- More research needs to be done

Questions?

Cara Cuiule
cara.cuiule@pricesystems.com

Backup

CMMI Study

- Improved quality by 50% and productivity by 60%
- Issues
  - Not all organizations were software
  - No standard way to measure quality and productivity
    - *Ex – lines of code written versus number of units produced*

Backup

Software Maintenance Maturity Model (SMmm)

- Meant to be supplemental to CMMI
- Vetted by academia and industry professionals
- No research claiming effect on productivity and organizational processes