Does Code Decay? Assessing the Evidence from Change Management Data

Stephen G. Eick, Todd L. Graves, Alan F. Karr, J.S. Marron, and Audris Mockus

Presenter: Zhifeng Lai
September 18, 2009
Introduction

• Inevitability of Changes
  – Addition of new functionalities
  – Adaptation to new environment
  – Fixes of reported bugs
  – Redesign of existing architecture …..

• Effects of Changes
  – “There are anecdotal reports of systems that have reached a state from which further change is not possible.”
Introduction

• Investigate the effects of changes to software.
  – Do historical changes make future changes harder (code decay)?
  – Diagnose code decay from some symptoms? How?
  – Predict some responses to code decay?
  – Take actions to prevent code decay?
Software Changes

• A Data-Driven Definition
  – A change is any alteration to the software recorded in the version management database
Software Changes

• A Data-Driven Definition
  – A change is any alteration to the software recorded in the version management database.

• A Well-Defined Process
Software Changes

• A Data-Driven Definition
  - A change is any alteration to the software recorded in the version management database.

• A Well-Defined Process

Diagram:
- Feature
  - IMR
    - Description
    - MR
      - Time Date
        - delta
          - File, Module
          - Developer
          - lines add., del.
Software Changes

• A Data-Driven Definition
  – A change is any alteration to the software recorded in the version management database.

• A Well-Defined Process

Initial Modification Requests (IMRs): general descriptions on how to implement features
Software Changes

• A Data-Driven Definition
  – A change is any alteration to the software recorded in the version management database.

• A Well-Defined Process

Modification Requests (MRs): descriptions on work to be done to each module
Software Changes

• A Data-Driven Definition
  – A change is any alteration to the software recorded in the version management database.

• A Well-Defined Process

```
Feature
IMR
  Description
  MR
    Time Date
    delta
      Developer
      #lines add., del.
    File, Module

Track added lines and deleted lines.
```
A Conceptual Model

• Change-Based Definition
  – *Code is decayed if it is more difficult to change than it used to be.*
A Conceptual Model

• Change-Based Definition
  – *Code is decayed if it is more difficult to change than it used to be.*

• Conceptual Model
Causes of Code Decay

• Reflect the nature of the software and the organizational environment
  1. Inappropriate architecture
  2. Violations of the original design principles
  3. Imprecise requirements
  4. Time pressure
  5. Inadequate programming tools
  6. Organizational environment
  7. Programming variability
  8. Inadequate change processes
Causes of Code Decay

- Reflect the nature of the software and the organizational environment
  1. Inappropriate architecture
  2. Violations of the original design principles
  3. **Imprecise requirements**
  4. Time pressure
  5. Inadequate programming tools
  6. Organizational environment
  7. Programming variability
  8. Inadequate change processes
Causes of Code Decay

• Reflect the nature of the software and the organizational environment
  1. Inappropriate architecture
  2. Violations of the original design principles
  3. Imprecise requirements
  4. **Time pressure**
  5. Inadequate programming tools
  6. Organizational environment
  7. Programming variability
  8. Inadequate change processes
Risk Factors for Code Decay

• Causes for concern that can increase the likelihood of code decay
  1. The size of a module
  2. The age of code
  3. Inherent complexity
  4. Organizational churn
  5. Ported or reused code
  6. Requirement load
  7. Inexperienced developers
Risk Factors for Code Decay

• Causes for concern that can increase the likelihood of code decay
  1. The size of a module
  2. The age of code
  3. Inherent complexity
  4. Organizational churn
  5. Ported or reused code
  6. Requirement load
  7. Inexperienced developers
Symptoms of Code Decay

• Measurable manifestations of code decay
  1. Excessively complex code
  2. A history of frequent changes
  3. A history of faults
  4. Widely dispersed changes
  5. Kludges in code
  6. Numerous interfaces
Symptoms of Code Decay

• Measurable manifestations of code decay
  1. Excessively complex code
  2. A history of frequent changes
  3. A history of faults
  4. Widely dispersed changes
  5. Kludges in code
  6. Numerous interfaces
Code Decay Indices

- Mathematical Model
  - Quantify causes, risk factors, and symptoms
  - Enable prediction
  - Design for handy version management database
Causes of Code Decay

- Reflect the nature of the software and the organizational environment

1. Inappropriate architecture
2. Violations of the original design principles
3. Imprecise requirements
4. Time pressure
5. Inadequate programming tools
6. Organizational environment
7. Programming variability
8. Inadequate change processes
Code Decay Indices

• Mathematical Model
  – Quantify causes, risk factors, and symptoms
  – Enable prediction
  – Design for handy version management database

**Code Changes**

**Code Decay**
Code Decay Indices

• Mathematical Model
  – Quantify causes, risk factors, and symptoms
  – Enable prediction
  – Design for handy version management database

![Diagram showing relationships between Code Changes, Size, Age, and Code Decay]
Code Decay Indices

- Mathematical Model
  - Quantify causes, risk factors, and symptoms
  - Enable prediction
  - Design for handy version management database

**Code Changes**
- Size
- Age

**Code Decay**
- History of Frequent Changes
- Span of Changes
- Fault Potential
- Effort
Code Decay Indices

- Code Changes
  - Size
  - Age
  - ...

- Number of MRs
- History of Frequent Changes
- Span of Changes
- Fault Potential
- Effort
- ...

Code Decay
Code Decay Indices

- Number of non-commentary source lines
- Code Changes
  - Size
  - Age
  - ...
- Code Decay
- History of Frequent Changes
- Span of Changes
- Fault Potential
- Effort
- ...

……
Code Decay Indices

AGE(m): The average age of a module’s constituent lines

Code Changes

- Size
- Age
- ......

Code Decay

- History of Frequent Changes
- Span of Changes
- Fault Potential
- Effort
- ......
Code Decay Indices

- Code Changes
  - Size
  - Age
  - …

The number of files a change touches

- History of Frequent Changes
- Span of Changes
- Fault Potential
- Effort
- …
Evaluation

• Objective
  – Collect evidences for code decay
  – Validate the change-based model

• Subject
  – A real-time telecommunication system
    • Fifteen-year development, 100,000,000 LOC
  – The analysis is based on a subsystem
    • 100 modules, 2,500 files, 500 login names
Increase in Span of Changes

• Span of Changes
  
  $$\text{FILES}(c) = \sum_{f} 1\{c \rightarrow f\}.$$  
  
  – The number of files a change touches

• FILES(c) is a symptom
  
  – Get expertise for unfamiliar files
  – Breakdown of modularity
  – Touching more files increases the size of a change
Increase in Span of Changes

Highlighted smooths

\[
\text{Prob}\{\text{FILES}(c) > 1\}
\]

Date
Breakdown of Modularity

• Historical Change Count

\[ \text{CHNG}(m, I) = \sum_{c \rightarrow m} 1\{\text{DATE}(c) \in I\}, \]

- The number of changes to a module \( m \) in the time interval \( I \)
- Frequent changes indicate code decay

• Common Historical Change Count

\[ \text{CHNG}(m, m', I) = \sum_{c \rightarrow m, c \rightarrow m'} 1\{\text{DATE}(c) \in I\} \]

- The number of changes touching both \( m \) and \( m' \)
- Large value indicates modularity breakdown
Breakdown of Modularity

1988

1989

1996
Breakdown of Modularity

\[ w(m, m') = \frac{\text{CHNG}(m, m', I)}{4\sqrt{\text{CHNG}(m, I) \times \text{CHNG}(m', I)}} \]
Breakdown of Modularity

\[ w(m, m') = \frac{\text{CHNG}(m, m', I)}{\sqrt[4]{\text{CHNG}(m, I) \times \text{CHNG}(m', I)}} \]
Breakdown of Modularity

Fault Potential

• Fault Predictors
  – Predict the number of faults that will have to be fixed in module $m$ in a future interval of time.
  – Quality response to code decay (due to changes)

• Model $^{[1]}$
  – Weighted time damp model
    \[
    \text{FP}_{\text{WTD}}(m, t) = \gamma_1 \sum_{c \rightarrow m, \text{DATE}(c) < t} e^{-\alpha[t-\text{DATE}(c)]} \times \log[\text{ADD}(c, m) + \text{DEL}(c, m)]
    \]
  – Generalized linear model
    \[
    \text{FP}_{\text{GLM}}(m, t) = \gamma_2 \sum_{c \in \Delta} \mathbb{1}\{c \rightarrow m\} \times \beta^{\text{AGE}(m)}
    \]

Fault Potential

- Weighted Time Damp Model

\[
FP_{WTD}(m) \propto \sum_{c \sim m} e^{0.75 \times \text{DATE}(c)} \times \\
\log[\text{ADD}(c, m) + \text{DEL}(c, m)]
\]
Fault Potential

- Weighted Time Damp Model

\[ \text{FP}_{\text{WTD}}(m) \propto \sum_{c \sim m} e^{0.75 \times \text{DATE}(c)} \times \log[\text{ADD}(c, m) + \text{DEL}(c, m)] \]
Fault Potential

- Weighted Time Damp Model

\[ FP_{\text{WTD}}(m) \propto \sum_{c \rightarrow m} e^{0.75 \times \text{DATE}(c)} \times \log[\text{ADD}(c, m) + \text{DEL}(c, m)] \]

ADD(c, m): number of lines added by c in module m
Fault Potential

• Weighted Time Damp Model

\[
\text{FP}_{\text{WTD}}(m) \propto \sum_{c \sim m} e^{0.75 \times \text{DATE}(c)} \times \\
\log[\text{ADD}(c, m) + \text{DEL}(c, m)]
\]

DEL \((c, m)\): number of lines deleted by \(c\) in module \(m\)
Fault Potential

• Weighted Time Damp Model

\[ FP_{\text{WTD}}(m) \propto \sum_{c \sim m} e^{0.75 \cdot \text{DATE}(c)} \times \log[\text{ADD}(c, m) + \text{DEL}(c, m)] \]

– Primary evidence that changes induce faults: \( \alpha \neq 0 \)

“When \( \alpha = 0 \), past changes of the same size would be indistinguishable from one another and, hence, none could be posited to have any specific effect.”
Fault Potential

- Weighted Time Damp Model

\[ FP_{WTD}(m) \propto \sum_{c \rightarrow m} e^{0.75 \times \text{DATE}(c)} \times \log[\text{ADD}(c, m) + \text{DEL}(c, m)] \]

- Primary evidence that changes induce faults: \( \alpha \neq 0 \)

\[ FP_{WTD}(m, t) = \gamma_1 \sum_{c \rightarrow m, \text{DATE}(c) < t} e^{-\alpha [t - \text{DATE}(c)]} \times \log[\text{ADD}(c, m) + \text{DEL}(c, m)] \]
Fault Potential

• Weighted Time Damp Model

\[ FP_{WTD}(m) \propto \sum_{c \rightarrow m} e^{0.75 \times \text{DATE}(c)} \times \log[\text{ADD}(c,m) + \text{DEL}(c,m)] \]

  – Primary evidence that changes induce faults: \( \alpha \neq 0 \)

• Generalized Linear Model

\[ FP_{GLM}(m) = 0.017 \times \sum_{c} 1\{c \rightarrow m\} \times 0.64^{\text{AGE}(m)} \]

  – Code having many lines that have survived for a long time is likely to be relatively free of faults
Fault Potential

• Other Findings
  – Module size and software complexity metrics do not improve fault prediction
  – Number of developers touching a module do not have effects
  – Concurrent changes with large number of other modules do not contribute to fault potential

  Code changes are more responsible for fault potential.
Model for Effort

• Effort Predictor
  – Predict the efforts (human hours) required to implement a change from symptoms and risk factors
  – A Regression Model:

\[
\log(1 + \text{EFF}(c)) = 0.32 + 0.13 \left(\log[1 + \text{FILES}(c)]\right)^2 - 0.09 \left(\log[1 + \text{DEL}(c)]\right)^2 + 0.12 \log[1 + \text{ADD}(c)] \log[1 + \text{DEL}(c)] + 0.11 \log[1 + \text{INT}(c)] - 0.47 \log[1 + \text{DELTAS}(c)].
\]
Model for Effort

• Effort Predictor
  – Predict the effort (human hours) required to implement changes from symptoms and risk factors
  – A Regression Model:

\[
\log(1 + \text{EFF}(c)) = .32 + .13 (\log[1 + \text{FILES}(c)])^2 \\
- .09(\log[1 + \text{DEL}(c)])^2 \\
+ .12 \log[1 + \text{ADD}(c)] \log[1 + \text{DEL}(c)] \\
+ .11 \log[1 + \text{INT}(c)] \\
- .47 \log[1 + \text{DELTAS}(c)].
\]

\text{INT}(c): \text{the time required to implement } c
Model for Effort

- Effort Predictor
  - Predict the effort (human hours) required to implement changes from symptoms and risk factors
  - A Regression Model:

\[
\log(1 + EFF(c)) = 0.32 + 0.13 \left( \log[1 + FILES(c)] \right)^2 \\
- 0.09 \left( \log[1 + DEL(c)] \right)^2 \\
+ 0.12 \log[1 + ADD(c)] \log[1 + DEL(c)] \\
+ 0.11 \log[1 + INT(c)] \\
- 0.47 \log[1 + DELTAS(c)].
\]

DELTAS(c): number of deltas associated with c
Model for Effort

- Effort Predictor
  - Predict the effort (human hours) required to implement changes from symptoms and risk factors
  - A Regression Model:

\[
\log(1 + \text{EFF}(c)) = 0.32 + 0.13 (\log[1 + \text{FILES}(c)])^2 \\
- 0.09 (\log[1 + \text{DEL}(c)])^2 \\
+ 0.12 \log[1 + \text{ADD}(c)] \log[1 + \text{DEL}(c)] \\
+ 0.11 \log[1 + \text{INT}(c)] \\
- 0.47 \log[1 + \text{DELTAS}(c)].
\]

The span of changes is a symptom of decay.
Model for Effort

• Effort Predictor
  – Predict the effort (human hours) required to implement changes from symptoms and risk factors
  – A Regression Model:

\[
\log(1 + \text{EFF}(c)) = 0.32 + 0.13 (\log(1 + \text{FILES}(c)))^2 \\
- 0.09 (\log(1 + \text{DEL}(c)))^2 \\
+ 0.12 \log(1 + \text{ADD}(c)) \log(1 + \text{DEL}(c)) \\
+ 0.11 \log(1 + \text{INT}(c)) \\
- 0.47 \log(1 + \text{DELTAS}(c)).
\]
Model for Effort

• Effort Predictor
  – Predict the effort (human hours) required to implement changes from symptoms and risk factors
  – A Regression Model:

\[
\log(1 + \text{EFF}(c)) = .32 + .13 \left( \log[1 + \text{EFF}(c)] \right) \\
- .09 \left( \log[1 + \text{DEL}(c)] \right)^2 \\
+ .12 \log[1 + \text{ADD}(c)] \log[1 + \text{DEL}(c)] \\
+ .11 \log[1 + \text{INT}(c)] \\
- .47 \log[1 + \text{DELTAS}(c)].
\]
Model for Effort

- **Effort Predictor**
  - Predict the effort (human hours) required to implement changes from symptoms and risk factors
  - A Regression Model:

\[
\log(1 + EFF(c)) = 0.32 + 0.13 \left( \log[1 + FILES(c)] \right)^2 \\
- 0.09 \left( \log[1 + DEL(c)] \right)^2 \\
+ 0.12 \log[1 + ADD(c)] \log[1 + DEL(c)] \\
+ 0.11 \log[1 + INT(c)] \\
- 0.47 \log[1 + DELTAS(c)].
\]
Model for Effort

• Effort Predictor
  – Predict the effort (human hours) required to implement changes from symptoms and risk factors
  – A Regression Model:

\[
\log(1 + \text{EFF}(c)) = 0.32 + 0.13 \left( \log[1 + \text{FILES}(c)] \right)^2 \\
- 0.09 \left( \log[1 + \text{DEL}(c)] \right)^2 \\
+ 0.12 \log[1 + \text{ADD}(c)] \log[1 + \text{DEL}] \\
+ 0.11 \log[1 + \text{INT}(c)] \\
- 0.47 \log[1 + \text{DELTAS}(c)].
\]
Summary

• Conceptual Model
  – Causes, symptoms, risk factors
• Code Decay Indices
  – Quantify the conceptual model
• Evaluation
  – The span of changes increases over time
  – The modularity declines over time
  – Code changes and their recency contribute to fault potential
  – The span and size of changes are important factors for predicting future efforts
Discussion

• Is the span of changes informative?
  – Major evidence for the existence
    • $\text{Prob}\{\text{FILES}(c) > 1\}$ \(\rightarrow\) breakdown of modularity
  – Contradictory finding
    • “Concurrent changes with large number of other modules did not contribute to fault potential.”
Discussion

• Is the span of changes informative?

• Can refactoring rejuvenate code?
  – Categorizes of changes
    • Adaptive changes
    • Corrective changes
    • *Perfective changes*
  – Categorization according to their effects to code decay?
Discussion

• Is the span of changes informative?
• Can refactoring rejuvenate code?
• Can design patterns prevent code decay?
Discussion

• Is the span of changes informative?
• Can refactoring rejuvenate code?
• Can design patterns prevent code decay?

Causes of Code Decay

• Reflect the nature of the software and the organizational environment
  1. Inappropriate architecture
  2. Violations of the original design principles
  3. Imprecise requirements
  4. Time pressure
  5. Inadequate programming tools
  6. Organizational environment
  7. Programming variability
  8. Inadequate change processes
Discussion

• Is the span of changes informative?
• Can refactoring rejuvenate code?
• Can design patterns prevent code decay?

Introduction

• Investigate the effects of changes to software.
  – Do historical changes make future changes harder (code decay)?
  – Diagnose code decay from some symptoms? How?
  – Predict some responses to code decay?
  – Take actions to prevent code decay?
Discussion

• Is the span of changes informative?
• Can refactoring rejuvenate code?
• Can design patterns prevent code decay?

Thank you!