
Does Code Decay? Assessing the Evidence from Change Management Data

Álvaro Navarro
Universidad Rey Juan Carlos
anavarro _at_ gsync.escet.urjc.es

*Libre Software Engineering Lab, November 2nd
2005*

(cc) 2005 Álvaro Navarro

Some rights reserved. This work licensed under Creative Commons Attribution-ShareAlike License. To view a copy of full license, see <http://creativecommons.org/licenses/by-sa/2.0/>

Summary

- What is code decay
- Changes of Software
- A Conceptual Model for Code Decay
- Causes of Software Decay
- Symptoms of Code Decay
- Risk Factors for Code Decay
- Code Decay Index
- Model for predicting faults

What is 'code decay' ?

code that is harder to change than it should be

Evidences of code decay:

- the span of changes becomes bigger in time (interval)
- there is less modularity (quality)
- the fault potential increases (quality)
- the effort to change the software raises (cost)

Changes to Software

Any alteration to the software recorded in the change history data base.

- Adaptive Changes: new functionality
- Corrective Changes: fix bugs
- Perfective Changes: maintain software without new functionalities

Changes to Software (II)

Process that is followed to introduce changes to the software:

- customers send Initial Modification Requests
- If accepted, an IMR generates a number of Modification Requests (MR)
- A change to a file is a delta (with meta-information related)

Store information in a database and answer questions.

A Conceptual Model for Code Decay

Three key responses:

- **COST**: cost for the developers
- **INTERVAL**: time to complete the change
- **QUALITY**: changed software

A Conceptual Model for Code Decay (II)

Limitations and comments that should be take into account:

- Code decay should be reworded as a temporal phenomenon.
- Increased difficulty may be intrinsic of the changes.
- Code can be correct and still be decayed.
- Decaying software may have more value.
- Implicitly decay is caused by changes, but code that is not changed may also decay because it's affected by changes in other code parts of the software.
- Attributing hard code changes to decay is misleading. There is code that is inherently hard to change.

Causes of Software Decay

- Inappropriate architecture
- Violations of the original design principles
- Imprecise requirements
- Time pressure
- Inadequate programming tools
- Organizational environment
- Programmer variability
- Inadequate change process

Symptoms of Code Decay

- Excessively complex (bloated) code
- History of frequent changes
- Code with history of faults
- Widely dispersed changes. In modularized code, changes should be local
- changes that could be done more elegantly or efficiently (Kludged)
- Numerous interfaces

Risk Factors for Code Decay

- Size of a module (in NCSL, i.e. SLOC)
- The age of the code.
- Inherent complexity.
- Organizational churn (degrading knowledge, more inexperienced developers change code...)
- Ported or reused code. Written for another system or platform
- Requirements load
- Inexperienced developers

Code Decay Indices

Three main issues:

- Select appropriate levels of aggregation (take MRs as they find that they are the most informative)
- Scaling (conversion into a rate per unit or per unit of software size)
- Transformation (taking logarithms or powers or roots to improve comprehension)

Indexes can be divided into two groups:

- Direct Indexes: number of deltas, lines added, lines

Code Decay Indices (II)

Derived Indexes:

- History of Frequent Changes
- Span of changes
- Size (in SLOC)
- Age, defined as the average age of its constituent lines
- Fault potential
- Effort

Code Decay Indices (III)

Strong evidence for code decay:

- The size of changes increases over time
- Network-based visualizations (with the network visualization tool, NicheWorks) show breakdown in modularity
- Other results give evidence of quality and effort being affected.

Model for predicting faults

- Code a year older than otherwise similar code tends to have only two-thirds as many faults
- changes to code are more responsible for faults than complexity
- number of developers touching a file also has no effect on its fault potential