Model of the System's Evolution Through Structural Erosion and Reorganization

Alexander Prutzkow

***Abstract*—A program system is created based on a problem solution using a model. The program system changes, including due to changes in its structure. The purpose of the study is to clarify the evolution of a system with a modifiable structure, which is necessary for understanding the program life cycle. We introduce a model of the system's evolution in aspect of changes in its structure. During the life cycle, the system exists in states of structured and unstructured. Erosion makes the system unstructured, and reorganization makes it structured. Some modifications don't change the state of the system. The measure of structuredness determines the state of the system. The measure can be the difference between the current and intended structures, structure parameters – the number of components, the relationships between them, the target indicator of the structure – for example, the complexity of changing the system or its structure. Technical debt and structural erosion negatively affect the structuredness of the system. Technical debt occurs after a change of a target and distortion of a system. Technical debt must be eliminated after the value of the measure of the system's structuredness critically exceeds the specified threshold. The causes of structural erosion are technical debt, lack of understanding of the system's structure, lack of experience in changing the system or its structure, and substituting the target indicator after the environment change. There are a number of approaches to measuring structural erosion, but none are accurate.**

***Keywords*—System, structure, structuredness, erosion, technical debt, measure.**

I. Introduction

Let the problem be solved by a technical system (fig. 1). The problem solution isn't formalized. A model is used to formalize the solution. Moreover, it is often impossible to describe the solution without a model. The model transforms and formalizes the solution, makes it model-driven, and is mapped into the solution's structure. A technical system is developed based on the model-driven solution. The system's structure mapped into the model-driven structure solution. Yourdon and Constantine state, "Implementation, maintenance, and modification generally will be minimized when each piece of the system corresponds to exactly one small, well-defined piece of the problem, and each relationship between a system's pieces corresponds only to a relationship between pieces of the problem" [1].

We give an example to explain the relationship between the concepts depicted in fig. 1.

Problem: The load located at point A must be found at point B.

Solution: move the load from point A to point B.

Models are (1) a crane and a truck, (2) 100 men, ropes, and rollers, or (3) a tractor and a sledge.

You can easily suggest model-driven solution. Model-driven solution isn't right if a model will be substituted.

A program system or program is a technical computer-based system [2]. The program system changes and evolves for a variety of reasons [3]. Lehman formulated the laws of software evolution in [4]. Evolution of these laws is researched thoroughly in [5]. If the system has architecture, then the architecture determines the evolution of the system [6]. When the system is modified, its structure may be changed [7]. Changes in the structure don't always make the system more structured.

II. Terminology

We define the terms (marked in sparse font) necessary for further statement and discuss them (marked ■). Definitions of some terms are based on the IEEE 1471-2000 standard ones.

System is a set of components organized to accomplish a specific function or set of functions.

■ We consider a system in a broad sense.

Structure is an organization of a system embodied in its components, relationships to each other and to the environment, component functions.

■ Not all systems can change structure, for example a biological organism.

System architecture is an abstract structure and the principles guiding design and evolution of a system.

■ The term "structure" is more general than the term "architecture". For example, the phrase "class structure" makes sense, but the phrase "class architecture" – no.

A system is structured when the value of the measure of its structuredness is higher than the specified threshold, the system is unstructured otherwise.

■ To determine whether a system is structured, a precise criterion is needed. The criterion is the value of the measure of its structuredness is higher than the specified threshold. The complexity of the system may become the measure. Other attributes of the system have their own measure and specified threshold. In [1, p. xi] "structured" and "organized" are synonyms for concept of "design process".

Manuscript received April 18, 2026.

A. Prutzkow is with Ryazan State Radio Engineering University, 390005, Gagarin str., 59/1, Ryazan, Russia, and with Lipetsk State Pedagogical University, 398020, Lenin str., 42, Lipetsk, Russia (e-mail: mail@prutzkow.com).

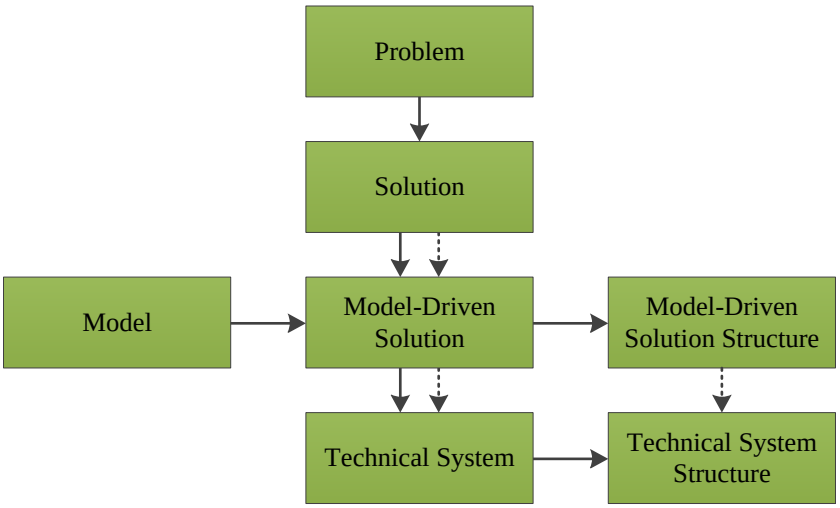

Fig. 1. A solution to a problem in the form of a technical system.
The solid arrow means "affects". The dotted arrow means "maps into"

Structural erosion is a change in system's structure after which the value of the measure of the structuredness critically exceeds the specified threshold.

- The term "structural erosion" has synonyms such as "architectural erosion", "architectural degradation", "architectural decay". Other synonyms are collected in [8, table 7; 9, p. 133]. The term "architectural erosion" was coined by Perry and Wolf [10].

Technical debt is a change in a system or its structure for the short-term improvement of another indicator, accompanied by deterioration in the original target indicator.

- "Software systems accumulate technical debt when short-term goals in software development are traded for long-term goals" [11]. Cunningham introduces the term "technical debt" as a metaphor in [12], but the term has become persistent. In contrast to Li et al. [13] and Lilienthal [14], we believe that technical debt arises after a deliberate system change. Ambiguities in the understanding of this term are explored in [15, 16]. Improving another indicator doesn't imply worsening the original target indicator. Formal and practical relationships between program indicators (attributes) are collected in [17].

III. Motivation

Lilienthal [14, fig. 4–1] depicts origination and effect of technical debt and collects concepts related to the evolution of a system with a modifiable structure. Lilienthal doesn't focus on and systematize the evolution.

IV. The Purpose of the Study

The purpose of the study is to clarify the evolution of the system with the modifiable structure, which is necessary for understanding the program life cycle.

V. Model of the Evolution of a System with a Modifiable Structure

We introduce a model of the system evolution in aspect of changes in system's structure (fig. 2). Our model is more abstract and general than the Lilienthal's chart.

We describe the model using an example. A company is developing a program. Due to a shift in the planned release date to an earlier date, it was necessary to speed up the developing of the program. It was decided not to carefully discuss the structure of the classes and their relationships (constructing with disorganizing and technical debt), but to ensure that the requirements were met. The first version of the program was released on time. The developers revised the class structure and their relationships, and modified the program ([re-]organizing). A new architect was invited to develop the second version of the program. Due to inexperience, the architect decided to inject new functions into a part of the program not intended for this purpose (erosion). After several correct and incorrect function injections, the complexity of making changes increased significantly. After the release of the second version of the program, its structure was reduced to the structure of the first version with some updates ([re-]organizing).

The evolution of the system with the modifiable structure is determined by the following elements:

- States of the system: structured system, unstructured system, system start, system end.
- Processes of transition of the system from one state to other: constructing with organizing, constructing with disorganizing [and technical debt], (re-)organizing, erosion, modifying, and ending.
- A measure of the system's structuredness that determines the state of the system: structured or unstructured.

The measure can be defined in at least three ways:

(1) the difference between the current and intended structures [18];
(2) parameters of the structure: the number of components, relationships between them;
(3) the target indicator of the structure; for example, the complexity of changing the system or its structure [19].

There is no precise measure of structuredness or unstructuredness. The degree of system's structuredness is defined in the following ways (fig. 3):

- one threshold and offset Δx;
- two thresholds: one threshold defines a structured system, and the other an unstructured system [14].

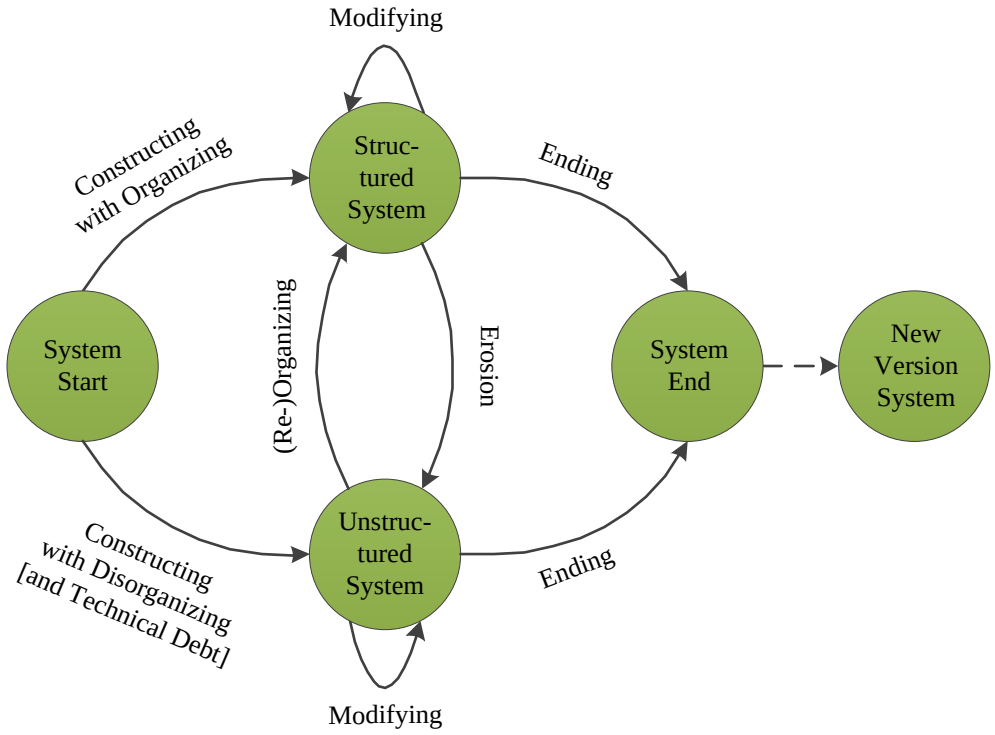

Fig. 2. States of the evolution of a system with a modifiable structure and transitions between them

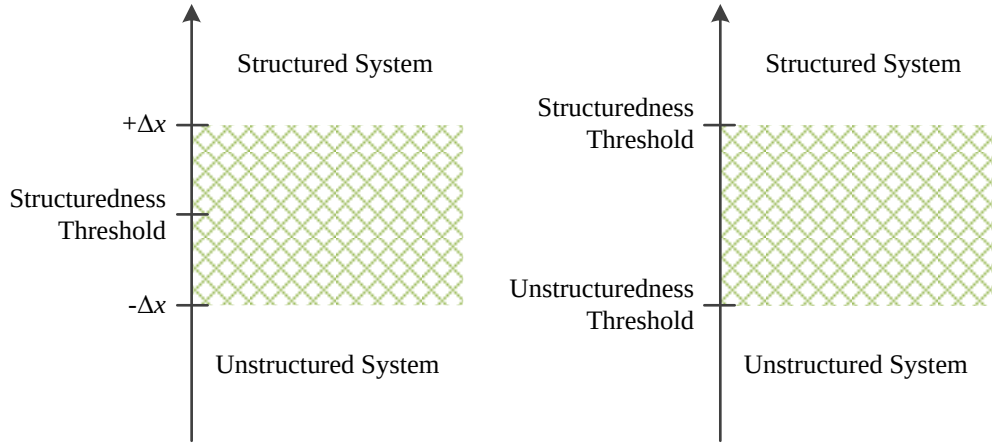

Fig. 3. Measure of the system's structuredness with one (left) or two (right) thresholds

After its termination, the system can be reborn in a new version written from scratch.

Technical debt and structural erosion negatively impacts the system's structure. We discuss them in details.

VI. Technical Debt

A slogan of technical debt is "Don't do it good now, but do it good later".

Technical debt arises after performing the following actions:

(1) a change of a target: the original target indicator is substituted by another target indicator or the range of the original target indicator is expanded on short term;
(2) distortion of a system: a change of a system or its components that doesn't meet the original target indicator but does meet another target indicator.

The major difference between technical debt and financial debt is that technical debt not necessarily pay [20].

Moreover, technical debt isn't always a bad practice. For instance, technical debt can speed up the development of new features [13, 16]. Technical debt must be kept visible and under control [13]. It's a duty of technical debt management.

Technical debt can increase or decrease during its lifetime [21]. For instance, the introduction of a new technology can reduce the technical debt without any changes of a system [15, 21].

Technical debt falls into levels (from top to bottom) [22]:

- process debt: insufficient documentation;
- architectural/structural debt: ill-considered change of structure, denormalization of the database structure;
- code debt: code duplication, redundant dependencies on global objects.

Technical debt types are identified in [23].

The original target indicator that is sacrificed with arising technical debt is one of the quality properties of the ISO / IEC 25010-2011 standard. Li et al. [13] calculated their

frequencies of mention in studies (in descending order):
(1) maintainability;
(2) portability;
(3) reliability,
(4) security;
(5) performance efficiency, compatibility;
(6) functional suitability, usability.

Main ways to technical debt payment are refactoring [14, 24] and design improvements [24].

Distortion of a system with technical debt is indistinguishable from distortion resulting from developer errors. Estimates of technical debt [15, 21], which cannot be clearly separated from developer errors, isn't accurate. Technical debt interest is inherently difficult to estimate or measure as well [11].

Technical debt must be eliminated after the value of the measure of the system's structuredness critically exceeds the specified threshold.

VII. Structural Erosion

The causes of structural erosion are
(1) technical debt [14];
(2) lack of understanding of the system's structure [25];
(3) lack of experience in changing the system or its structure [25];
(4) substituting the target indicator after the environment change.

Causes 1–3 are active changes in structure, while cause 4 is passive.

Cause 1 is conscious structural erosion, causes 2–3 are unconscious structural erosion. Malicious structural erosion is beyond the scope of the article.

Causes 1–4 lead to obsolescence of the system [26] or its structure.

In [8], more causes were found, but they reduce to listed above.

Symptoms of architectural erosion are [8]
- structural symptom;
- violation symptom refers to the violations of prescribed design decisions or constraints (abstraction and encapsulation);
- quality manifest symptoms in quality attributes of products;
- evolution symptom.

de Silva and Balasubramaniam [9] classify strategies for controlling architectural erosion:
- minimize strategies are targeted towards limiting the occurrence and impact of architectural erosion but isn't effective in eliminating it;
- prevent strategies to avoid erosion;
- repair strategies fix the damage caused by erosion and reconcile the implementation with its architecture.

There are practical examples of structural erosion in [27].

VIII. Measurement of Structural Erosion

The degree of system's structuredness is determined by a measure, so the choice of this measure is an important aspect. The measure of system's structuredness can be based on the measure of structural erosion.

Ahmad et al. [22] collects 54 metrics and 31 techniques to measure architectural degradation. Measures are classified by types technical debt.

Architectural debt measures evaluate architectural quality with reflection models, architecture smell detection, consistency checks, and coupling/cohesion analysis. Design decision methods focus significantly on structural change detection and high-level architectural analyses.

Code debt measures primarily emphasizes implementation and code quality, focusing on proactive anomaly detection, and maintenance-centric metrics like stability and evolution tracking, complemented by static and rule-based analyses.

Process debt measures center on development practices, employing team and community analysis methods for proactive monitoring and reactive management of architectural alignment through team practices.

Conclusion of this research is effective monitoring requires metrics that capture both architectural integrity and temporal dynamics.

Baabad et al. [28] classify approaches to identify and measure architectural erosion: (1) architectural change approach, (2) historical data revision approach, (3) architectural cohesion approach, (4) architectural modularization approach, (5) architectural technical debt approach, (6) software size architecture approach, (7) architectural complexity approach, (8) architectural bad smells approach, and (9) architectural dependency coupling approach.

Li et al. [8] divide the structural erosion measures into two types and a combination of these types:
- preventive measure (architecture conformance checking, architecture monitoring and evaluation, establishing traceable mechanism, evolving architecture as changes occur, explicitly defining architecture);
- remedial measure (architecture maintenance, architecture restoration);
- both (architecture refactoring, management optimization).

IX. Conclusions

We illustrated the evolution of the system with the modifiable structure using a state diagram. We first clarified the terminology. We intended to provide only definitions of the terms. However, we were forced to discuss them with explanations.

Throughout its life cycle, the system exists in both structured or unstructured states. The states change after some system modifications.

The evolution is described in a general sense. The subject field of studies referenced in the article is software engineering, but isn't restricted by the field or merely technical systems:
(a) The evolution of the system illustrates the process of learning the Java language in our course [29]. Students send the teacher completed practical assignments with an incorrect structure (constructing with disorganizing) or technical debt (constructing with disorganizing and technical debt). The teacher makes comments, and the students correct them ([re-]organizing).
(b) Conway states, "There is predictable relationship between the graph structure of a design organization and the graph structure of the system it designs" [30]. So the

evolution's model may be extended on the organization's structure.

(c) The model of the system's evolution initiates a historical question: What were the reforms of the state structure of Russia under Peter I: reorganizing the old structure or transition to a new system (new version system)?

(d) The article is written in accordance with the evolution as well. (1) Compiling the structure of the article and writing the primary text (constructing with organizing). (2) Expanding an article with the results of existing studies; the results were not always included in the part of the article that corresponded to the structure (erosion). (3) Editing the text and structure of the article ([re-]organizing).

We deliberately don't use the term "decision" for the following reasons:

- more important manifestation of decision is a change of a system;
- the decision is determined by the knowledge, experience, and psychology of the person, that are beyond the scope of the article; see [31].

The article has theoretical significance, as it binds the concepts of the system, its structure, erosion, technical debt as the evolution of this system. The article has practical significance, as it clarifies the reasons for and the need for system reorganization, which is essential in industrial programming and teaching.

REFERENCES

[1] Yourdon E., Constantine L. (1978) Structured Design. Fundamentals of a Discipline of Computer Program and Systems Design // Yourdon Press.

[2] James K.L. (2016) Software Engineering // PHI Learning.

[3] Rajlich V. (2014) Software Evolution and Maintenance // Software Engineering / DOI: 10.1145/2593882.2593893.

[4] Lehman M. (1996) Laws of Software Evolution Revisited // Software Process Technology.

[5] Herraiz I. et al. (2013) The Evolution of the Laws of Software Evolution // ACM Computing Surveys (CSUR), 46(2) / DOI: 10.1145/2543581.2543595.

[6] Jazayeri M. (2002) On Architectural Stability and Evolution // Reliable Software Technologies.

[7] Breivold H.P. et al. (2012) A Systematic Review of Software Architecture Evolution Research // Information and Software Technology, 54 / DOI: 10.1016/j.infsof.2011.06.002.

[8] Li R. et al. (2022) Understanding Software Architecture Erosion: A Systematic Mapping Study // Journal of Software: Evolution and Process, e2423 / DOI: 10.1002/smr.2423.

[9] de Silva L., Balasubramaniam D. (2012) Controlling Software Architecture Erosion: A Survey // Journal of Systems and Software, 85(1) / DOI: 10.1016/j.jss.2011.07.036.

[10] Perry D.E., Wolf A.L. (1992) Foundations for the Study of Software Architecture // ACM SIGSOFT Software Engineering Notes, 17(4).

[11] Zazworka N. et al. (2014) Comparing Four Approaches for Technical Debt Identification // Software Quality Journal, 2(3).

[12] Cunningham W. (1992) The WyCash Portfolio Management System // Object-Oriented Programming Systems, Languages, and Applications / DOI: 10.1145/157710.157715.

[13] Li Z. et al. (2015) A Systematic Mapping Study on Technical Debt and Its Management // Journal of Systems and Software, 101 / DOI: 10.1016/j.jss.2014.12.027.

[14] Lilienthal C. (2022) Improve Your Architecture with the Modularity Maturity Index // Software Architecture Metrics.

[15] Kruchten P. et al. (2013) Technical Debt: Towards a Crisper Definition // ACM SIGSOFT Software Engineering Notes, 38(5).

[16] Tom E. et al. (2012) A Consolidated Understanding of Technical Debt // Information Systems.

[17] Prutzkow A. (2026) What Is a Good Program? A Space Way // System Engineering and Information Technologies, 8(2) / DOI: 10.54708/SIIT-2026-no2-p29.

[18] Merkle B. (2010) Stop the Software Architecture Erosion // Systems Programming Languages and Applications: Software for Humanity / DOI: 10.1145/1869542.1869621.

[19] Jaktman C.B. et al. (1999) Structural Analysis of the Software Architecture – A Maintenance Assessment Case Study // Software Architecture / DOI: 10.1007/978-0-387-35563-4 35.

[20] Guo Y. et al. (2014) Exploring the Costs of Technical Debt Management – A Case Study // Empirical Software Engineering Journal, (1) / DOI: 10.1007/s10664-014-9351-7.

[21] Lavazza L. et al. (2018) Technical Debt as an External Software Attribute // Technical Debt / DOI: 10.1145/3194164.3194168.

[22] Ahmad N. et al. (2025) Architectural Degradation: Definition, Motivations, Measurement, and Remediation Approaches // arXiv, 2507.

[23] Alves N.S.R. et al. (2016) Identification and Management of Technical Debt: A Systematic Mapping Study // Information and Software Technology, (70).

[24] Pérez B. (2021) Technical Debt Payment and Prevention Through the Lenses of Software Architects // Information and Software Technology, (140) / DOI: 10.1016/j.infsof.2021.106692.

[25] Bandara V., Perera I. (2018) Identifying Software Architecture Erosion Through Code Comments // Advances in ICT for Emerging Regions.

[26] Parnas D.L. (1994) Software Aging // Software Engineering.

[27] van Gurp J., Bosch J. (2002) Design Erosion: Problems and Causes // Journal of Systems and Software, 61(2).

[28] Baabad A. (2025) An Empirical Analysis of Approach-Based Metrics Model for Architectural Erosion Detection // Software: Practice and Experience, 55(9).

[29] Prutzkow A.V. (2024) Teaching Java Programming at the University: Pedagogical Aspects [Prepodavanie yazyka Java v Vuze: Pedagogicheskie Aspekty] // Humanitarian Studies of Central Russia [Gumanitarnye Issledovaniya Tsentralnoy Rossii], (4 (33)) / DOI: 10.24412/2541-9056-2024-433-62-68.

[30] Conway M. (1968) How Do Committees Invent? // Datamation, 14(4).

[31] Shneiderman B. (1980) Software Psychology. Human Factors in Computer and Information Systems // Winthrop Publishers.