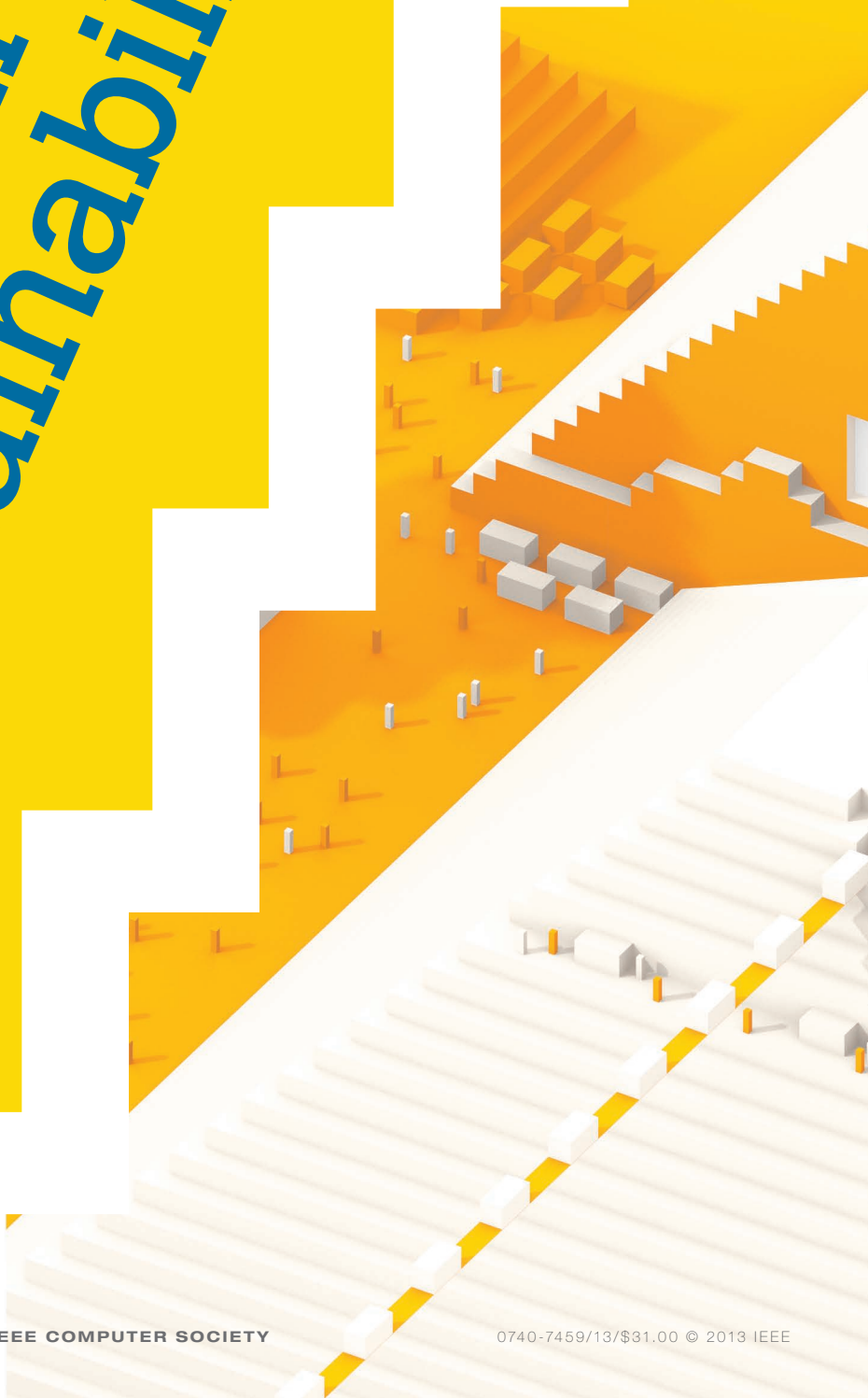
Architecture Sustainability

Avgeriou, Paris; Stal, Michael; Hilliard, Rich

Published in:
Ieee software

Publication date:
2013

Architecture Sustainability



0740-7459/13/$31.00 © 2013 IEEE

Paris Avgeriou
University of Groningen

Michael Stal
Siemens AG

Rich Hilliard
Freelance software systems architect

SOFTWARE ARCHITECTURE IS the foundation of software system development, encompassing a system's architects' and stakeholders' strategic decisions. These decisions are often made in an unsystematic manner, incurring grave consequences for the system's development, future operation, and maintenance, thus leading to design erosion and to a decrease of internal and external qualities. In turn, this leads to rework and renovations and therefore to increased costs and dissatisfied stakeholders. (We have witnessed such examples in the safety-critical embedded systems domain, such as power plants, trains, and medical imaging devices.) The problem is aggravated because such systems' lifetimes can span several decades. Changing or extending these systems requires as systematic a process as possible.

Systematic architecting considers a system in its total environment, where *environment*, according to ISO/IEC/IEEE 42010, is the "context determining the setting and circumstances of all influences upon a system including developmental, technological, business, operational, organizational, political, economic, legal, regulatory, ecological and social influences."[1] Such influences include business goals and strategies, prioritized system requirements, in-house experience and expertise, operation, development and deployment constraints, and other realities. Taken together, these form the basis for establishing the forces that architects must consider when making decisions and identifying the risks to be mitigated throughout the system's life cycle. Systematic architecting also implies incremental growth and strict separation of concerns; otherwise, architects would be repeatedly faced with the need to make simultaneous complex decisions (which bears the risk of accidental complexity).

Changes Affecting Architecture Sustainability

One of the primary goals of systematic architecting is to increase architecture sustainability—that is, the architecture's capacity to endure different types of change through efficient

maintenance and orderly evolution over its entire life cycle. Sustainability should not only consider the system within its boundaries but should also be understood in relation to the systems' total environment. Consequently, we can't limit systematic architecting to initial architecture creation but must observe it throughout software architecture evolution and improvement to encourage architecture sustainability during change. Furthermore, sustainability isn't a matter of big-design-up-front versus agile or lean—our attention to sustainability is crucial to the success of a software system under either style of development. Among the causes of change, several are particularly significant for architecture sustainability.

New Requirements Emerge while Older Requirements Change

Every iteration includes both new architecture elements to satisfy new strategic and tactical requirements as well as invasive architecture changes in existing components that extend and refine earlier architecture decisions. This is particularly important in product line and platform development, where decisions can impact multiple systems.

Interdependence between Requirements and Architecture

As Frederick Brooks noted, it's common in a system's early stages for architects to not understand the requirements in detail, or perhaps the requirements are preliminary, conflicting, ambiguous, or incomplete, and customers and product managers might not understand whether architects are creating an appropriate solution for the problems.[2] Thus, architecture and requirements engineering should happen in parallel and in negotiation (as the Twin Peaks model advocates[3]), which inevitably leads to architectural changes.

Changes in Business Strategies and Goals

Software projects' turnaround times are subject to business pressures. Within a project's timeline, the business strategy might change to address new markets or mergers and acquisitions. The architecture must reflect such business changes.[4] For example, rescoping a product family to reflect a new business strategy will induce significant modifications to the architecture.

Environment Changes

Changes to a system's environment can have a dramatic effect on that system—for example, changes to a bank's business model significantly affect the supporting enterprise applications. Architects must adapt systems to such changes to stay aligned with their environment. The safest and most effective way to achieve this is to create system architectures that anticipate or are open to such changes.

> We can't limit systematic architecting to initial architecture creation.

Architecture Erosion or Drift

Often, architecture isn't adequately maintained or kept in sync with the system; as a result, numerous choices creep into the implementation that are contrary to explicit architecture decisions. In contrast to other causes of change, architecture erosion is usually due to an architect's inaction rather than intentional factors.

Accidental Complexity

Architects face a high degree of inherent complexity when they design medium to large software systems. As a result, they tend to introduce accidental complexity[5] by creating "design pearls"—solutions that are more complex than necessary—or by leveraging suboptimal solutions such as inappropriate patterns or algorithms, with negative impact on quality.

Technology Changes

Current technology platforms and tools are constantly changing, and new technologies emerge at a rapid pace. Architects must manage changes resulting from these underlying technologies and tools, especially for technologies with strategic impact.

Deferred Decisions to Meet Near-Term Goals

Architects must balance competing concerns, including the ability to meet near-term production cost limitations and delivery schedules at the risk of longer-term sustainability. One metaphor for the impact of such decisions is technical debt, which implies that the architecture will need to be changed in the future to pay back the debt created.

To Err Is Human

Architects, engineers, and other decision-making stakeholders of any software system will sometimes make wrong, unnecessary, or less-than-optimal decisions. When we fail to detect such issues early, a chain of subsequent design decisions rooted in bad design decisions can result. Rectifying such decisions will induce architecture changes, the magnitude of which will depend on how early bad decisions are discovered. These issues are even more crucial in software product line engineering and platform development,

TABLE 1

A comparison of approaches to handling change.

	Refactoring	Renovating	Rearchitecting
Change in components	Existing ones modified	New ones built from scratch	Components reused, modified, built, or rebuilt
Effort	Medium	Medium	Substantial
Reverse engineering required	Some components	Some components	Entire system
Frequency	Regular	Medium	Seldom

where every issue affects several applications rather than just one.

Approaches to Architecture Sustainability

Changes in the architecture often don't fit into one or even a few phases of the life cycle but range over the entire life cycle. Some changes will only affect local parts of a system; others will have more widespread impact. Systemic changes might involve various stakeholders and disciplines. We distinguish three types of approaches for handling change systematically, listed in an order of increasing severity: refactoring, renovating, and rearchitecting (see Table 1).

Refactoring

An architecture follows a piecemeal growth, but after each refinement step, it's subject to architecture evaluation. If the architecture evaluation identifies and reveals smells such as dependency cycles or overly generic design, possible architectural improvements are identified. We'll typically perform these improvements in a tactical setting, modifying certain elements but not the offered functionality of the system. Architecture refactoring can also help open up the architecture for extension or change.

Renovating

Sometimes parts of the architecture will be in such poor condition that refactoring is no longer effective. In such cases, renovating the architecture—rebuilding one or more essential elements from scratch—might be a better choice. Renovating is complementary to refactoring because it also deals with only parts of the system: often a decision must be made between changing these parts (refactoring) or building or rebuilding them from scratch (renovating).

Rearchitecting

When an architecture is subject to significant changes, refactoring or renovating won't always suffice. This might be the case when a technology platform is replaced by a newer one, when there is a significant change in business scope, or when the architecture is in such bad shape that errors keep emerging. In such cases, rearchitecting is necessary. The rearchitecting process usually analyzes the existing architecture (for example, via a SWOT [Strength, Weakness, Opportunity, Threat] analysis) and results in a new architecture by reusing components that are worth keeping, modifying some of the existing components (refactoring), rebuilding the rest of the existing components (renovating), or building some entirely new components that offer new functionality.

Support

Architects need support in the form of concepts, methods, techniques, and tools for recognizing, confronting, and managing architecture sustainability concerns. We advocate making explicit the differences among refactoring, renovating, and rearchitecting approaches, even though these are usually treated as a single topic of maintenance and evolution. Mixing the three types obfuscates the already challenging problems of architecture sustainability for practicing architects and offers no clear direction to researchers. We encourage the research community to pursue approaches focusing on each of these types and provide targeted architectural changes according to the characteristics of each type. When making decisions, architects must be clear about choices among the three types or combinations thereof, and they need support when deciding on a sustainability strategy and its implementation. Furthermore, refactoring, renovating, and rearchitecting must take place within a context of architecture governance that establishes who is allowed to check or change the architecture (including the what, how, and when) as well as to check architecture compliance and conformance.

In This Issue

An important recent trend in software architecting has been the decision viewpoints: recognizing architects' need to capture and record decisions and the rationale for those decisions as first-class ingredients of architecture descriptions.[6,7] Following that trend, in "Making Architectural Design Decisions Sustainable: Challenges, Solutions, and Lessons Learned," Uwe Zdun, Rafael Capilla, Huy Tran, and Olaf Zimmermann address the topic of sustainable design decisions and their documentation over the course of software evolution. The authors summarize five criteria for sustainability of decisions based on lessons they've

ABOUT THE AUTHORS

PARIS AVGERIOU is a professor of software engineering at the University of Groningen. His research interests include software architecture with a strong emphasis on architecture modeling, knowledge, evolution, patterns, and links to requirements. Avgeriou received a PhD in software engineering from the National Technical University of Athens. He's a senior member of IEEE. Contact him at paris@cs.rug.nl.

MICHAEL STAL is senior principal engineer at Siemens AG's Corporate Research and Technology division. His research interests include software architecture, middleware, service-oriented architecture, concurrent and networked systems, and product line engineering. Stal received a PhD in computer science from the University of Groningen. Contact him at michael.stal@siemens.com.

RICH HILLIARD is a freelance software systems architect. His research interests include software engineering, system architecture, and requirements analysis. Hilliard received a degree in mathematics and linguistics from MIT. He's also a member of the IEEE Computer Society. Contact him at r.hilliard@computer.org.

learned from their experiences in various industrial projects. Their proposed solutions include minimal decision documentation, extraction, and distillation of recurring decisions as reusable guidance models, traceability between requirements and decisions and between decisions and implementation, and the use and application of design rationale. These solutions span the three approaches of refactoring, renovating, and rearchitecting

In "Measuring Architecture Sustainability," Heiko Koziolek, Dominik Domis, Thomas Goldschmidt, and Philipp Vorst explore sustainability as economical longevity, arguing that architectural sustainability, like architecture itself, is influenced by many factors that rise from many sources. The authors describe experiences over a two-year period with an approach called Morphosis in applying metrics for tracking architecture sustainability to the development of an industrial control system. The Morphosis approach consists of evolution scenario analysis, technology choice scoring, architecture compliance checks, and architecture-level code metric tracking, which are all useful for assessing erosion and modularization. The authors demonstrate that such assessments can be carried out with limited effort and that through regular assessment, developers can improve their code through refactoring to achieve improved scores.

In "Implementing Long-Term Product Line Sustainability through Planned Staged Investments," Juha Savolainen, Nan Niu, Tommi Mikkonen, and Thomas Fogdal deal with sustainability in a setting of long-lived systems: the software product line (SPL), where changes for individual products over time produce architecture drift. To address this, the authors introduce planned staged investments for sustainable rearchitecting. Their strategy involves two alternating activities—investment when core assets are realigned with current and anticipated needs, and harvesting to create products from core assets with minimum effort required—as ways to balance the conflicting needs that result from redesign and reuse activities. They demonstrate this approach through a case study of an actual SPL (a frequency converter). Many practitioners, not only those working on SPLs, will recognize the problems discussed in this article.

We hope this special issue will raise awareness of architecture sustainability issues and increase interest and work in the area. We thank the authors, everyone who submitted manuscripts, our reviewers, and the *IEEE Software* editor in chief and editorial staff for their efforts throughout the development and preparation of this issue.

Acknowledgments

A special thanks goes to David Emery for exchanges on the causes of architecture change.

References

1. *Systems and Software Engineering—Architecture Description*, IEEE Std. ISO/IEC/IEEE 42010, Dec. 2011.
2. F.P. Brooks, *The Design of Design: Essays from a Computer Scientist*, Addison-Wesley Professional, 2010.
3. B. Nuseibeh, "Weaving Together Requirements and Architectures," *Computer*, vol. 34, no. 3, 2001, pp. 115–119.
4. P. Clements and L. Bass, "The Business Goals Viewpoint," *IEEE Software*, vol. 27, no. 6, 2010, pp. 38–45.
5. F.P. Brooks, *The Mythical Man-Month*, Addison-Wesley, 1975.
6. U. van Heesch, P. Avgeriou, and R. Hilliard, "A Documentation Framework for Architecture Decisions," *J. Systems and Software*, vol. 85, no. 4, 2012, pp. 795–820.
7. P.B. Kruchten, R. Capilla, and J.C. Dueñas, "The Role of a Decision View in Software Architecture Practice," *IEEE Software*, vol. 26, no. 2, 2009, pp. 36–42.