Agile & Architecture

Geert Haerens

Department of Management Information Systems
Faculty of Business and Economics
University of Antwerp, Belgium
Email: geert.haerens@uantwerpen.be

Enterprise Architect @ Digital and IT Consulting
Engie
Email: geert.haerens@engie.com

***Abstract*—The Agile & Architecture special track at Patterns 2024 aims at collecting knowledge on how to build truly agile software systems, being systems that have the ability to change. We have collected four papers that address different aspects of agile architecture, including the comparison of the heuristic method SOLID with the scientific-based software development method of Normalized Systems (NS), a use case of evolvability of an application built according to NS over a period of ten years, a use case where NS is used to help software migration and agile software factories.**

***Index Terms*—Agile; Software Architecture; NS; SOLID; Software Rejuvenation; Software Migration, Software Factory**

I. Introduction

During his life, Marcus Vitruvius Pollio (85-20 BC) [1], a Roman engineer, published about 10 books about "Architectura". For him, architecture is about applying three principles: Firmita (strong, firm), Utilitas (useful, user-friendly), and Venustat (aesthetic, beautiful). Since then, the concept of architecture has extended beyond the construction of physical structures. It is used in technologies and social structures. The IEEE1471 standard [2] defines architecture as "the fundamental organization of a system embodied in its components, their relationships to each other and the environment, and the principles guiding its design and evolution". This definition of architecture applies to many kinds of architecture, be it physical, technical or sociological.

We focus on software architecture, where software systems are made up of components that relate (interact) with each other and interact with the environment (technical and socio-technical (aka the users)). The software architecture should include or be made in accordance with some principles that guide the evolution of the software system. The software architect is responsible for ensuring that the software has an architecture that conforms to the requirements.

Today, software creation is dominated by the Agile approach. Agile, as a word, signifies the ability to move with quick, easy grace and a resourceful and adaptive character [3]. As architecture is about the structure of a system and agility is about the ability to change, we can say that an agile system would have a structure that allows and facilitates change. Agile architecture is thus a property of a system. A system must be consciously defined to become agile, as it does not become agile by itself.

Agile, in the context of software creation and delivery, is inspired by the Agile Manifesto [4], which states:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work, we have come to value:

- *Individuals and interactions **over** processes and tools*
- *Working software **over** comprehensive documentation*
- *Customer collaboration **over** contract negotiation*
- *Responding to change **over** following a plan*

Many Agile Methods have been developed based on this manifesto, which aims to deliver software in small iterations with quick stakeholder feedback compared to the waterfall methodology [5], where the first piece of working software is delivered after having gone through a big upfront design phase. When looking at an agile methodology such as SCRUM, it must be noted that the architect's role is not mentioned. It is assumed that the development team can absorb this role completely and make the system according to requirements. Recall that, based on the In the Agile Manifesto, requirements will change; thus, the software will change over time. The iterative nature of the agile software delivery method requires the software to be able to change and thus exhibit agility as a property. Otherwise, at some point, new requirements can no longer be implemented or implemented within an interaction, effectively killing the agile methodology.

The SCRUM method does not scale well. For creating large systems, the Scaled Agile Framework, SAFe [6], is better suited. SAFe recognizes the need for architectural oversight and has defined different architectural roles for the purpose. SAFe defines Agile Architecture as:

- *Systems evolve over time while supporting the needs of current users*

- *Avoid overhead and delays associated with phase-gated processes*
- *Ensure Systems Always run*
- *Supports the continuous flow of value*
- *Balances intention architecture and emerging design*

According to SAFe, Agile Architecture is about the process/method of performing Architecture. It is no longer a phase-gate process but a process focusing on the continuous flow of value. What SAFe calls Agile Architecture is more Agile Architecting, a process and guidance on how an architect should behave, but includes little guidance on making a truly agile system except for the need for intentional architecture.

The fundamental question that emerges is: how do we make agile systems? How to make agility part of the intentional architecture.

II. Submissions

For this special track on Agile & Architecture, we have collected four papers that are related to agile systems. One paper compares two software development methodologies aiming at creating agile systems. The second paper is a use case of an application created with a method that promises system agility, and the application's evolvability is tracked over ten years. A third paper is about the integration of system agility during software migration. The fourth and last paper is about extending the agility of the software system toward the software factory, where applications are not only created but also deployed to the working environment.

The first paper, by Gerko Koks [7], studies the convergence between SOLID [8] and Normalized Systems [9] [10]. SOLID is an acronym for five software design principles that deliver a so-called Clean Architecture when combined in a certain form. SOLID has been put forward by Robert C. Martin and is extensively explained in his Clean Code book series. The SOLID principles are known in the software development community and frequently serve as guidelines for software creation. Following the SOLID and Clean Architecture approach will provide cleaner code, which is more easily adaptable and thus provides the agility we seek. Normalized Systems theory (NS) provides four theorems derived from system stability and statistical entropy that aim to make software evolvable. A software system is considered evolvable when a bounded functional change leads to a bounded amount of work to implement this change and is independent of the system's size. A system is considered non-evolvable when change ripples through the system, making the amount of work for a single functional change dependent on the system's size. This is called a Combinatorial Effect. In his paper, Gerco Koks investigate to what extent the SOLID and NS principles converge; what are the differences and similarities between the two? The principles are compared, and the architecture resulting from applying these principles is also compared. The conclusion is that the SOLID principles can be mapped to NS theorems but that SOLID lacks equivalent principles for some of the NS principles. The same is true for the resulting architecture. The conclusion is that applying SOLID is valuable yet insufficient for an evolvable system, as not all the NS theorems are covered.

The second paper, by Jan Verelst [11], studies the evolvability of an application over ten years. The application has been built in accordance with the NS principles using software elements and software expansion. Software elements are the architectural construct in which the NS principles are applied, and software expansion ensures that the architectural code skeleton is generated to ensure compliance with the NS principles. Building applications this way provides evolvability with respect to different axes of anticipated changes, such as functional changes, changes to the elements and underlying technological changes. Each time a change happens in these dimensions, the software gets re-expanded, aka rejuvenated, and promises not to introduce a CE in the code skeleton. The paper tracks the different kinds of changes over 10 years and tracks the rejuvenation effort. If NS lives up to its promise, the rejuvenation efforts should be limited. The conclusion is that major changes have happened to the application over 10 years, both functional and technological, but the effort to implement these changes is small. This demonstrates that building software systems with NS provides the agility we seek.

The third paper, by Christophe De Clercq [12], is a software migration use case where NS is used to create the target application and support the migration. The migration applies a phased approach where certain functionalities are gradually migrated from the source to the target application while keeping the functionality accessible in the source and destination until a pre-defined cut-over point. Such an approach requires gateways between the target and source application and data migration at the cut-over point. Classically, such gateways are delicate pieces of software that easily break, are difficult to maintain, and require removal from the target system after the cut-over point. By considering the need for gateways as a cross-cutting concern and integrating it into the expandable software elements, the maintenance of the gateways is restricted to the templates and the rejuvenation capabilities' use. After the final migration of the data from the source to the target system, the gateways can be easily removed by rejuvenating the application with element templates that no longer include the gateway cross-cutting concern. On the one hand, the new software is created with the NS principles and expansion capabilities; on the other hand, NS helps with migration. By doing so, the target system will exhibit evolvability, and a high degree of flexibility (and evolvability) will be provided during the migration phase.

The fourth and last paper, by Herwig Mannaert [13], concerns evolvability in software factories. Today, the DevOps [14] and the CI/CD philosophy are put forward as the default method of linking development and production. The idea is that the development and production people work together and have a mutual understanding of the challenges they face to take them into account in their respective domains, thus improving the flow from software development to production. Unfortunately, the philosophy is often reduced to an abstract

infinity sign and the usage of tools. Making software is hard. Making a software factory will be even harder. The continuous changes and proliferation of technologies and tools in the CI/CD-pipeline do not help either. The paper put forward that DevOps and CI/CD are cross-cutting concerns in the software factory and should be treated as such. Having agile software systems without an agile software factory is the same as having a great product that cannot be shipped to where it is needed.

III. Conclusion

The four papers of this Agile & Architecture special track on Patterns 2024 all relate nicely to the subject we wanted to draw attention to: how to make agile systems. Gerco Koks's paper discusses the necessary conditions for agility (NS), and he compares them with existing heuristical methods (SOLID). A use case of rejuvenation, demonstrating agility, has been discussed in Jan Verelst's paper. Christophe De Clercq demonstrated agility in a migration scenario, and Herwig Mannaert showed us the direction toward evolving software factories.

References

[1] Marcus Vitruvius Pollio on Wikipedia, [Online], Available: https://en.wikipedia.org/wiki/Vitruvius, [retrieved: May, 2024].

[2] M. W. MAIER, D. EMERY, R. HILLIARD, "ANSI/IEEE 1471 and systems engineering. Systems engineering", 2004.

[3] Agile on Websers Dictinairy, [Online], Available: https://www.merriam-webster.com/dictionary/agile, [retrieved: May, 2024].

[4] Agile Manifesto, [Online], Available: https://agilemanifesto.org/, [retrieved: May, 2024].

[5] Waterfall method on Wikipedia, [Online], https://en.wikipedia.org/wiki/Waterfall_model, [retrieved: May, 2024].

[6] SAFe, [Online], https://scaledagileframework.com/agile-architecture/, [retrieved: May, 2024].

[7] G. Koks, "Converging Clean Architecture with Normalized Systems", in Special Track: Agile & Architecture, along with PATTERNS 2024, IARIA XPS Press, 2024.

[8] R. C. Martin, "Clean architecture: a craftsman's guide to software structure and design", London, England: Prentice Hall, 2018, OCLC: on1004983973, ISBN: 978-0-13-449416-6.

[9] H. Mannaert, J. Verelst and P. De Bruyn, "Normalized Systems Theory: From Foundations for Evolvable Software Toward a General Theory for Evolvable Design", ISBN 978-90-77160-09-1, 2016

[10] H. Mannaert, J. Verelst and K. Ven, "The transformation of requirements into software primitives: Studying evolvability based on systems theoretic stability", Science of Computer Programming, Volume 76, Issue 12, pp. 1210-1222, 2011

[11] J. Verelst, "Systematic Rejuvenation of a Budgeting Application over 10 years: A Case Study", in Special Track: Agile & Architecture, along with PATTERNS 2024, IARIA XPS Press, 2024.

[12] C. De Clercq, "Using Normalized Systems Expansion to Facilitate Software Migration - a Use Case", in Special Track: Agile & Architecture, along with PATTERNS 2024, IARIA XPS Press, 2024.

[13] H. Mannaert, "Toward a Rejuvenation Factory for Software Landscapes", in Special Track: Agile & Architecture, along with PATTERNS 2024, IARIA XPS Press, 2024.

[14] R. T. Yarlagadda, "Devops and its practices," International Journal of Creative Research Thoughts (IJCRT), vol. 9, no. 3, 2021, pp. 111–119.