ADA: Embracing technology change acceleration

Ondřej Dvořák, Robert Pergl, and Petr Kroha

Czech Technical University in Prague,
Faculty of Information Technology, Czech Republic,
{ondrej.dvorak, robert.pergl, petr.kroha}@fit.cvut.cz,
WWW home page: http://ccmi.fit.cvut.cz

Abstract. The pace of technology change has accelerated in the past decade. Conceptually similar technologies are introduced on nearly a daily basis. On one hand, IT experts call for applying the most modern approaches and technologies to software projects, on the other, companies suffer from liabilities to a technology used in their legacy solutions. This seems to result in a disturbing situation when a specific technology of an ongoing software project becomes legacy almost before the project successfully hits a production. This poses a continual challenge for software development, and effective ways of technology transition are sought. Affordance-Driven Assembling (ADA) represents such an effort from the standpoint of enterprise engineering theories. In this paper, we formulate a high-level architecture of a software system based on ADA. We demonstrate the architecture on an example of an object-oriented system. We evaluate the qualities of such architecture from the perspective of evolvability using Normalized Systems Theory, and we formulate conclusions on potential of this approach.

Keywords: component-based systems, semantic descriptions, software architecture, EE theories, ADA, O-ADA, design patterns, evolvability

1 Introduction

In the past decades, a range of new software technologies and frameworks have been developed to provide various solutions to software development challenges. For instance, we can find tens of frameworks and libraries for Dependency Injection, Object-Relational-Mapping, Service-Bus, we can dabble with hundreds of User Interface (UI) technologies, or rummage thousands of controls in JavaScript. Although there are considerable differences among them, many of them address the same class of problems. At the same time, it feels like the pace at which these frameworks are introduced is accelerating.

We commonly refer to the famous Moore's Law which observes this phenomenon in computing. This widely known exponential doubling of transistors is powering the advances in the past five decades. However, it seems there is a similar common force driving information technology forward. In [5], Ray Kurzweil introduces the so-called Law of Accelerating Returns which shows that the technological change is exponential, as well, and every decade the overall rate of progress is doubling [5]. He demonstrates that: "We wont experience 100 years

of progress in the 21st century it will be more like 20,000 years of progress (at todays rate)" [5].

If Kurwzeil is right, we can expect that more and more frameworks and libraries will be introduced even quicker. This brings a serious problem to deal with *legacy software*, i.e., software which has been developed using already outdated technology. Due to the technology pace, the software systems can thus become legacy almost before hitting a production – as the pace of development technology increases, so too does the pace of technology obsolescence [8]. Therefore, the software industry must address this problem of adopting to new technologies quickly. Otherwise, it will be much earlier facing the painful issues when maintaining legacy software, e.g., lack of IT resources, lack of skilled man power, lack of up-to-date documentation, costly to support and maintain [9]. This problem is even worsen with the evidence showing that companies already spend most of their available budgets on maintenance [1].

In this paper, we focus on technologies used for UI development where the problem of a technology transition is arguably the most observable in the proliferation and dynamics of UI frameworks. For instance, the JavaScript community coined a term the "JavaScript fatigue". Regardless the progress and popularity of JavaScript, developers report that they are dealing with a fatigue which refers to an inability to keep up with the latest libraries, there is a fear of becoming obsolete and outdated due to a constant change of the ecosystem and an overwhelming number of choices [2].

We argue that by following certain architectural patterns in a software system, we can adapt software artefacts to the latest technologies more efficiently. In this paper, we elaborate on the ADA (Affordance-Driven Assembling) approach that is founded in formal EE (enterprise engineering) theories. In [4], we introduced the way of thinking about software systems aligned with EE theories lens, in this paper we elaborate on this and show how a way of working looks in ADA to demonstrate the desired benefits of easing the technology transition.

2 Evolvability and Formal Foundations

To measure an improvement in helping technology transition, we work with the term "evolvability". We understand "evolvability" in terms of Normalized Systems Theory (NST) [6] that explains it as the "ability for software to be easily changed" [7]. In our research, the "technology change" is a so-called NST change driver. We measure it from a perspective of combinatorial effects. Thus, the main criterion used to evaluate the quality of a certain software architecture is a *bounded impact of the technology transition*.

When migrating a software from one technology to another, we suffer from incompatibilities between them. In practice, we usually tackle the problem of moving to another framework, or library based on the same platform or language, e.g., .NET, Java, C++, JavaScript. Thus, we typically identify parts which can be left untouched while the rest is adopted to the new technology. This rises a question of what are the concepts that can be easily reused, and which make it harder to move. Additionally, in which way we have to capture these concepts

in a software system in order to optimize it for future upgrades. In [4], it was shown that such concepts can be found in Enterprise Engineering (EE) τ-theory and β-theory [3].

3 Affordance-Driven Assembling

In [4], it was clarified that the concept of EE theories can be observed in Software Engineering (SE) as well. It was shown that software systems founded in these theories must be composed of components that suit the needs of a given user with a specific *purpose*. This relationship between users with purposes and components with properties can be captured by the term *affordance*:

Definition 1. ***Affordance*** *is a user-component relationship, which can be represented by the following formula:*
*affordance: (user * purpose) * (component * properties).*

When building software systems based on this definition, we must be able to identify so-called ADA-users, describe their ADA-purposes, and based on that build the final solution composed of ADA-components. All of these ADA artifacts are formally defined in [4]. Fig. 1 depicts basic high-level components and their relations in an ADA component-based system.

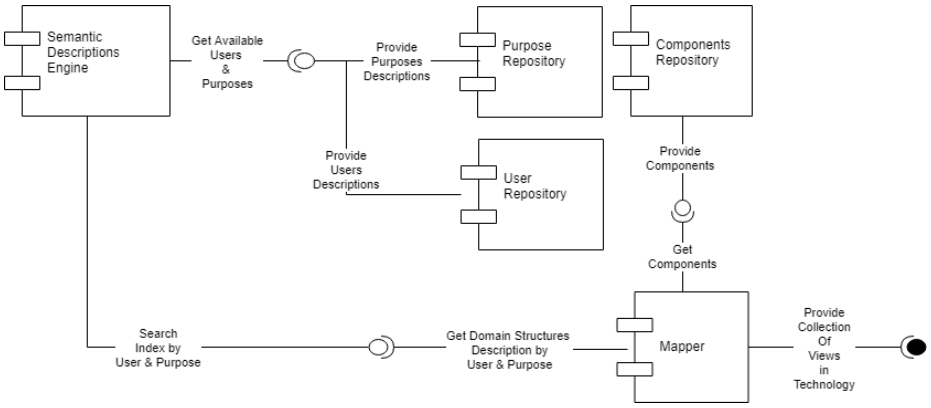

Fig. 1. A possible high-level architecture of a system applying ADA

4 Case Study: Evaluating ADA in Practice – `corima`

The COPS company has implemented an application framework `corima`. It is an example of ADA in practice. The system `corima` is capable of hosting applications in various business domains[1]. By now, the biggest application suite is focused on banking and corporate treasury management. It encompasses around 50

[1] Under the business domain, we understand the area of a business, e.g., finance, health-care, etc.

data-centric web and desktop applications used across tens of customers mainly in Europe.

The system `corima` implements ADA concepts captured in Fig. 1. We evaluated its code-base in order to measure the impact of a technology or domain transition.

Our preliminary calculations show that if the change-driver is a technology (e.g., a move to another UI framework), the impact of such a change is bounded. We only need to change a code which corresponds to the *Mapper* and *Component Repository* in Fig. 1. This only makes ∼2.2% of the entire code-base.

On the other hand, if the change driver is in the domain. The adjustment of *Purpose* and *User repository* is bounded to ∼1.7% of the whole code-base and it won't change with the amount of new applications.

5 Conclusion and Future Work

In this paper, we started by observing that the pace of introducing new technologies is exponentially accelerating, posing a serious challenge for the software industry with systems becoming quickly legacy ones much sooner. We addressed this challenge by showing how ADA can ease technology transitions.

We formulated a possible high-level architecture for software systems following ADA and demonstrated it on a case study of a real system. Using NST, we evaluated that ADA improves the evolvability of software in situations of technological transitions by putting clear boundaries of change impact.

Although our measurements indicated the potential of ADA, further statistical evaluation on a large number of cases is as subject of a future work.

References

1. Alija, N.: Justification of software maintenance costs 7, 15–23 (03 2017)
2. Clemmons, E.: Javascript fatigue. https://medium.com/@ericclemmons/javascript-fatigue-48d4011b6fc4 (2018)
3. Dietz, J., Hoogervorst, J.: Theories in Enterprise Engineering Memorandum - TAO, BETA
4. Dvořák, O., Pergl, R., Kroha, P.: Affordance-driven software assembling. In: Enterprise Engineering Working Conference. pp. 39–54. Springer (2018)
5. Kurzweil, R.: The law of accelerating returns. In: Alan Turing: Life and legacy of a great thinker, pp. 381–416. Springer (2004)
6. Mannaert, H., Verelst, J., De Bruyn, P.: Normalized Systems Theory, From Foundations for Evolvable Software Towards a General Theory for Evolvable Design. Normalized Systems Institute (2016)
7. Oorts, G., Huysmans, P., De Bruyn, P., Mannaert, H., Verelst, J., Oost, A.: Building evolvable software using normalized systems theory: A case study. In: 2014 47th Hawaii International Conference on System Sciences. pp. 4760–4769. IEEE (2014)
8. Seacord, R.C., Plakosh, D., Lewis, G.A.: Modernizing legacy systems: software technologies, engineering processes, and business practices. Addison-Wesley Professional (2003)
9. Srinivas, M., Ramakrishna, G., Rao, K.R., Babu, E.S.: Analysis of legacy system in software application development: A comparative survey. International Journal of Electrical & Computer Engineering (2088-8708) 6(1) (2016)