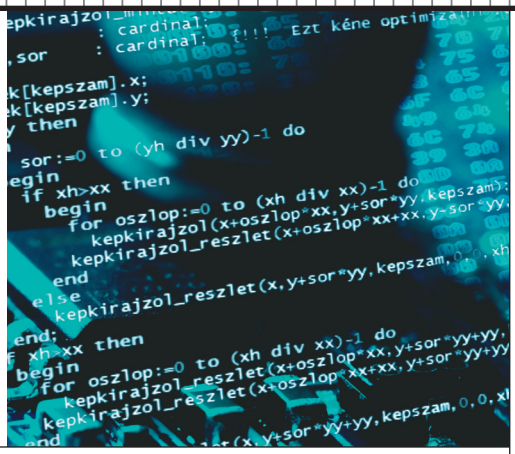
Simplicity as a Driver for Agile Innovation

Tiziana Margaria, *Potsdam University*
Bernhard Steffen, *TU Dortmund University*



Software and hardware vendors long avoided interoperation for fear of opting out of their own product lines. Yet decisive change came to the automobile industry from a holistic evolution and maturation on many fronts.

Looking at software system production and use today, we can easily compare the industry's current life cycle to that experienced by the automobile industry 80 years ago. The following statement, attributed to Gottlieb Daimler, characterizes carmakers' expectations at that time: "The market for automobiles will never grow beyond one million cars, for a very simple reason: Who would educate all those chauffeurs?"

This skepticism is understandable—back then, cars were handcrafted and cost more than a house. At the time, they were technically amazing—they could go up to 100 kph—but they had a hefty downside—the mean distance between flat tires averaged 30 km thanks to nail damage from horses and carts.

Not surprisingly, the number of extra tires constituted a status symbol: two full wheels were normal, with some cars carrying up to eight extra wheels to weather longer trips. But those who could afford a car were neither willing to change tires nor eager to maintain the engine, making well-trained chauffeurs an indispensable commodity in the 1920s.

So it goes with software. Despite the promises and effort, working with software products still offers a comparable adventure, one that rarely proceeds as expected. Difficulties with deployment and use lead to enormous system, organizational, and personal performance losses, not only at first deployment but even more so when we factor in the inevitable upgrades, migrations, and version changes.

THE PRICE FOR THE PACE

Millions of users suffer when standard software with a large market share evolves. Maybe it undergoes a radical redesign of the graphical user interface (GUI) or offers a new generation of tools not readily compatible with previous versions. Users must then desperately search for previously well-understood functionality, spending hours or even days bringing perfectly designed documents to a satisfactory state within this changed technical environment.

This frustrating catch-up phase causes an enormous productivity loss that can force customers to shy away from updates and migrations, sticking instead with old and even outdated or discontinued products or versions. In many situations, customers fear any kind of innovation involving IT because they immediately associate a change with enormous disruptions and long periods of instability. With technology-driven innovations, this fear is justified thanks to the new technologies themselves. However, even small and technically simple adaptations to a business process typically require a major IT project, with all its involved risks.

Thus, decision makers act conservatively, preferring patches and exchanging functionality only when it's absolutely necessary. Even the automobile industry fails when it comes to IT adoption and, particularly, IT agility. Much of a car's control software runs on specific hardware, which limits the software's applicability, especially after the hardware becomes obsolete—the software can't be ported elsewhere, meaning the manufacturer is more or less stuck with that hardware.

It takes engineers years to innovate, which the product life cycle then outlives by factors beyond that of the electronics and software within. The central problem is the IT lock-in at design time: decisions on which technology to use and long-term deals with the manufacturers are frozen before production starts and often last beyond the facelifts that periodically refresh these products.

 Published by the IEEE Computer Society

In the aerospace industry, this lifetime mismatch is even more evident: it takes decades to plan and design a mission, which leaves the IT used in the field in a typically decades-old state. IT innovation is the fastest we observe, and it systematically outpaces the life cycle of the products built using it. Inevitably, the products' life spans shorten to those of the IT they embody, as in consumer electronics, but this is unacceptable for expensive products.

Today, we have a similar situation in IT: singularly taken, the technologies and products are well-designed and innovative, but aren't made for working together and can't evolve independently. Consequently, we work with systems whose stability isn't proven and in which we can thus pose only limited trust. Once a bearable situation is achieved, and a constellation works, we tend to stick to it, bending the business and procedures to fit the working system, then running it until support is discontinued, if then. This shows that even pure software-based IT is often caught in the platform lock-in trap: business needs too often outpace the life cycle of the IT platforms that steer a company's organization and production.

STATES OF THE ART

Various factors contributed to our current state of the art. Some are rooted in the business models of major software and hardware vendors, who long avoided interoperation for fear the consequences of opting out of their own product lines would be dire. The frantic pace of technology provides its own chaos: before a certain technology reaches maturity and can repay the enormous investments for its development and production, a newer option attracts attention with novelty and fresh promises.

Decisive change came to the automobile industry not from the isolated improvement of single elements but from a holistic evolution and maturation on many fronts, with the interplay of numerous factors:

- *Better, more robust components.* The modern car platform approach builds on comparatively few well-engineered individual components, such as the tires, motor, and the chassis.
- *Better streets.* Today, we hardly need worry about flat tires.

Even pure software-based IT is often caught in the platform lock-in trap.

- *Better driving comfort.* Cars run smoothly, reliably, and safely, even if maltreated. User orientation has made a huge difference: drivers don't need to be mechanics.
- *Better production processes.* Modern construction supports cars tailored to their customers, even if all are built on platforms. Essentially, no two delivered cars are identical, but all are bound to only a few well-developed platforms.
- *Better maintenance and support.* Drivers have access to support worldwide, which can even include home transportation.

These modern developments have a straightforward match to the situation in IT, while also revealing the weaknesses of today's IT industry:

- *Better, more robust components.* Today's components are typically too complicated and fragile, and therefore are difficult to integrate in larger contexts. Service orientation seems to be a potentially strong step in the right direction, but it must be combined with a clear policy.
- *Better connection and interoperation.* We still lack seamless connection and integration, with numerous mismatches at the protocol, interface, or behavioral level. Meanwhile, the intended semantics and accompanying security provide an everlasting concern and a hot research topic.
- *Better user comfort.* Experts might know various specifically optimized solutions, but normal users find none. Even getting a modern phone to simply make a call can be rather frustrating, with many perceived extra steps and commands.
- *Better production processes.* Application development and quality assurance should be directly steered by user requirements, controlled via user experience, and continuously subject to modification during development.
- *Better maintenance and support.* Established scenarios and often-used functionality should continue to work, while support should be immediate and integrated into the normal workflow.

The transition to overcoming these weaknesses will depend on adopting economical principles that favor dimensions of maturity and simplicity over sheer novelty. In our analogy, Formula One car racing is an attractive platform for high-end research, but is unsuited for the needs and requirements of mass driving due to different skills, costs, and traffic conditions. Taking ideas and results from the high-end and specialized laboratory product requires diverse and extensive research to succeed. Transferred to the IT domain, this kind of research spans several dimensions:

- *Human-computer interaction* has led to GUIs that provide an intuitive user interface.
- *Domain modeling and semantic technologies* can establish a

user-level understanding of the involved entities.
- *Cloud computing* and other forms of platform virtualization provide stable user-level access to functionality.
- *Service orientation and process technologies* offer easy interactive control at the user process level.
- *Integrated product line management and quality assurance* requires validation and monitoring to guarantee correctness criteria at design, orchestration, and runtime.
- *Rule-based control* helps developers react flexibly to unforeseen situations.
- *Security and safety* affect not only business-critical applications but also technologies for establishing a high level of fault tolerance, be it at the infrastructural, software, or human level.
- *Major application domains*, such as business, biology, or medicine, keep the focus on constant awareness of the primary issue—user requirements.

The contributions of these individual research areas must be combined holistically to successfully control, adapt, and evolve systems composed of mature components.

THE PRICE FOR MATURITY

Achieving a sufficient level of maturity across components, connections, interoperation, and evolution is a complex and highly interdisciplinary task that requires technological knowledge and deep domain modeling expertise.

In this setting, standard investigation topics in IT such as complex architectural design and computational complexity are only of secondary and ancillary importance. The key to success is application of the "less is more" principle, with the goal of treating simple things simply, by a correspondingly simple design reminiscent of Lego blocks: primitive and well-defined blocks combine to reliably create complex solutions.

Developers might argue that there is no universal approach, but several domain-, purpose-, and profile-specific approaches within their scope are possible that capture the vastness of today's programming problems much more simply, reliably, and economically than most people think. This approach trades generality, which must be complex to accommodate diverse and sometimes antagonistic needs, with simplicity.

Companies such as Apple have successfully adopted simplicity as a fundamental design principle—for example, insights that simplify its users' lives concern both the handling of its products and their maintenance and robustness. Users adopted these innovations enthusiastically and pay a premium price for this "IT simply works" experience. Similarly, Windows 7 attempts to overcome the tendency to provide cutting-edge and increasingly complicated technology in favor of a more user-driven philosophy. Combining extensive interviews and agile methods in its development accelerated this paradigm shift.

While promising beginnings, these initiatives fall short of making mature technologies that simply work a widespread reality. We need extensive research and a clear engineering approach tailored to simplicity.

IT SIMPLY WORKS

The potential of a slogan like "IT simply works" offers vast opportunities unrestrained by the physical limitations of classical engineering. In principle, every software component can be exchanged at any time, almost everywhere, without leaving any waste—an ideal situation for truly component-based engineering.

Leveraging this potential would economically surpass the impact of producing new products based on leading-edge IT. Studies of product innovation show that technological leadership corresponds only to a relatively small fraction of market success and new market creation.

Most often, technology-driven innovation accompanies risk caused by the new technologies themselves. Innovations rooted in the *business purpose*, such as the service to the user or customer, have a much higher chance of success because user-level advantages are easier to communicate in the market, especially if detached from technological risks.

Improved levels of maturity can enable a new culture of innovation on the application side. Once we overcome the fear of change, true agility will guide the application experts, leading to new business models and new markets. History shows that with the availability of reliable cars, totally new forms of transportation and business arose.

For the software industry, maturity could revolutionize software's mass construction and mass customization far beyond our experience in the automotive industry. Theoretically, we can easily "change wheels while driving" and decompose and reassemble the entire car or bring new passengers aboard at the speed of light without being bound to specific hardware.

From a higher perspective, drawing adequate lines here can be considered a distinguishing trait for this new line of research and play a central role in the evolution of our economy and society.

Tiziana Margaria *is chair of service and software engineering at the Institute of Informatics, Potsdam University, Germany. Contact her at margaria@cs.uni-potsdam.de.*

Bernhard Steffen *is chair of programming systems in the Department of Computer Science, TU Dortmund University, Germany. Contact him at steffen@cs.tu-dortmund.de.*

Editor: Mike Hinchey, Lero—The Irish Software Engineering Research Centre; mike.hinchey@lero.ie