FOCUS

DESIGN THINKING

21 Guest Editors’ Introduction
Design Thinking
Annie Combelles, Christof Ebert, and Percival Lucena

25 On Integrating Design Thinking for Human-Centered Requirements Engineering
Jennifer Hehn, Daniel Mendez, Falk Uebernickel, Walter Brenner, and Manfred Broy

32 Migrating a Software Factory to Design Thinking: Paying Attention to People and Mind-Sets
Nolwen Mahé, Bram Adams, Josianne Marsan, Mathieu Templier, and Sylvie Bissonnette

41 Three Phases of Transforming a Project-Based IT Company Into a Lean and Design-Led Digital Service Provider
Seppo Kuula, Harri Haapasalo, and Juha-Matti Kosonen

FEATURES

52 Emerging Perspectives of Application Programming Interface Strategy
Juho Lindman, Jennifer Horkoff, Imed Hammouda, and Eric Knauss

60 The Online Controlled Experiment Lifecycle
Aleksander Fabijan, Pavel Dmitriev, Helena Holmström Olsson, and Jan Bosch

68 Closing the Gap Between Software Engineering Education and Industrial Needs
Vahid Garousi, Görkem Giray, Eray Tüzün, Cagatay Catal, and Michael Felderer

78 The Sound of Software Development
Laura Barton, Gülipék Candan, Thomas Fritz, Thomas Zimmermann, and Gail C. Murphy

MISCELLANEOUS

4 Contact Us
COV 3 IEEE Computer Society Information
DEPARTMENTS

3 From the Editor
Building Blocks of Software Design
Ipek Ozkaya

6 Requirements
When Does Design Help Thinking, and When Does Design Thinking Help?
Birgit Penzenstadler

10 Reliable Code
Predicting the Past
Gerard J. Holzmann

13 Software Technology
Security Test
Christof Ebert, Youssef Rekik, and Rahul Karade

49 Point–Counterpoint
Design Thinking in Industry
Talita Rozante de Paula, Tatiane Santana Amancio, and Jose Adan Nonato Flores

86 Practitioners' Digest
Requirements Engineering Research: News From the Trenches
Jeffrey C. Carver and Birgit Penzenstadler

89 Impact
Bringing Semantic Knowledge Graph Technology to Your Data
Bob van Luijt and Micha Verhagen

95 Sounding Board
Anaya's Journey: A Vision for a Future Software Academy
Austen Rainer

98 Software Engineering Radio
Boris Cherny on TypeScript
Nate Black
Building Blocks of Software Design

Ipek Ozkaya

SOFTWARE DESIGN INVOLVES the process of understanding the requirements and creating the artifacts that specify these requirements as the product to be built. The specification of the requirements ultimately happens in code. Intermediate abstraction mechanisms, such as domain modeling languages, software design and architecture patterns, programming paradigms, and design fragments, assist software engineers to specify requirements further into the final designs as implementations. However, in the absence of commonly agreed-upon building blocks that assist software engineers in tracing the design specification across software elements, these abstraction mechanisms become sources of unintended errors. Consequently, despite the availability of many software development lifecycle processes and implementation tool support, designs erode and drift from their intent quicker than anticipated.

Software design refers to both the process of creating the software product as well as the characteristics of the product itself. Design thinking and similar approaches assist software engineers during the design process through prototyping, testing, and experimentation of concepts. Techniques such as visualization or use of metaphors provide software engineers tools to further progress their designs. However, ultimately the vocabulary—the building blocks—that software design gets expressed and delivered to its end users is through code.

The ease of creating and building code, compared to creating and building any other engineering product that requires manufacturing steps, lies at the heart of the software complexity, design drift, rework, and catastrophic failure challenges of the software industry. The software design process gets cut short, code is not utilized effectively to design but to quickly implement the changes, creating a gap between the optimal design and the actual deployed design. In an effort to bridge this gap, we need better building blocks that assist software engineers to design and communicate the design as well as construct the design.

**Code as Software Design?**
In his 1992 essay, “What Is Software Design?” Reeves suggests that one of the reasons C++ as a programming language had become popular was because C++ made it easier to design software and program it at the same
Reeves also argues that the only documentation that satisfies the engineering criterion of the ability to build software to a specification is the source code as an engineering document itself. The act of constructing (manufacturing) in software is taken care of by compilers. Reeves observes that one reason why software gets complex very quickly is a consequence of the fact that building the design in software is a simple push-button compile action, with much less overhead of manufacturing. Today, building and deploying the end product is further simplified by the increasing capabilities of DevOps automation pipelines. Ease of making and deploying changes results in a focus on delivering the changes, as opposed to making the changes with sound design. Because creating the source code as the document and its construction by the compiler is perceived to be cheaper activities, subsequent changes, iterations, and evolutions are often done reactively, many times resulting in adding unintended complexity to software.

Building blocks that are expressive enough to represent software design as well as its construction should provide ability to specify information about the components, such as how elements communicate, what states elements are in, and what states persist as well as strong type checking that allows for detecting errors. If the ultimate representation of the design becomes the code, next-generation programming languages should make a targeted effort in their expressive power for representing such design characteristics.

Models as Software Design?
A key criticism against “the design is in the code” perspective is often the inability of the code to express runtime properties, the static and dynamic communications, and cross-cutting concerns across different elements of the software. Architecture thinking assists in capturing these kinds of specifications and system properties, and architecture modeling languages provide a vocabulary to express them. However, the high-level architecting process can be several steps removed from the act of programming. Model-based software engineering approaches use formal modeling languages that generate code to fill this gap; however, these techniques are still yet to be robust enough to represent the software product comprehensively. General-purpose modeling languages, such as UML, have failed in providing the necessary level of formalism in an effort to provide general enough design representation. Other modeling languages that provide tighter formal specifications and code generation capabilities, such as Architecture Design and Analysis Language (AADL), have limited scope.

Ironically, the needs expressed by software developers and architects from software architecture and modeling languages are no different than their needs from programming languages that they should be expressive enough to represent design. A study conducted with 48 practitioners from 40 IT companies revealed that architecture languages are not closing the software design expressiveness gap any better. The top gap software engineers found as a barrier in industry for using existing architecture modeling languages was their limited formalism to support design analysis. The most commonly cited limitations included lack of ability to express quality attribute properties, such as latency, throughput, propagation of change,
and lack of formality resulting in languages with no precise semantics.

**Closing the Gap**

Designing and delivering a software product, like any other engineering product, is a complex process that involves many activities, artifacts, and stakeholder communication techniques to collect, implement, and verify the requirements. Agile software development processes and the availability of better deployment tools have resulted in the reduction of errors that stem from communication barriers and delays introduced due to high-ceremony processes. The ultimate challenge in improving software products remains to be figuring out how to avoid the design errors and inconsistencies introduced among the many layers of abstraction that are essential in managing the complexity of the process.

Programming is not solely about constructing software—programming is about designing software. Thinking about the source code as the design does not imply don’t design, just code. Good architecture and abstractions are essential. Similarly, architecting is not solely about designing software, architecting is about constructing software. Software engineers increasingly require high-level programming languages that are closer to how software engineers think and design software as well as modeling languages that are closer to how detailed designs can be realized in code. While we continue to groom architects that think in code and developers that think in design, there are also opportunities for developing better programming languages that can express design and better tools that provide automated support for iterative design and design conformance.

**References**

When Does Design Help Thinking, and When Does Design Thinking Help?

Birgit Penzenstadler

From the Editor

Every requirements engineering practitioner has heard the adage, “Requirements are the what, design is the how.” This view of design describes the work of specifying how the product under design is to be implemented: development specifications, demonstrably traceable up to the requirements to indicate that the “what” has been satisfied. In this issue’s “Requirements” column, Birgit Penzenstadler invites us to take a step back and explore the question of “why” we are undertaking a particular project or program. Design thinking affords opportunities to creatively explore the broadest systems’ context of our work, and fully understand the struggles and challenges that our profession affords us the opportunity—and maybe even the imperative—to collaboratively engage with and address. —Sarah Gregory

As I’m Sitting

In the audience of a software development day, where Johan Sanneblad (hiQ developer and business consultant) raves about the possibilities of ultrawideband (IEEE Standard for Information Technology, IEEE Standard 802.15.4a, 2007) and wireless personal area networks and the new chip that Apple put in their new iPhone, I realize that the focus on the technical perspective is really asking us to take a deeper look at the social implications of the technology we are designing. How can we design technology for people, together with people, considering impacts on other people? As often in the requirements corner, the answer includes taking multiple perspectives, a diverse set of stakeholders, ethical considerations, and creativity.

Many of these questions can be answered in part by design thinking (DT) approaches. DT refers to the cognitive, strategic, and practical processes by which design concepts (proposals for new products, buildings, machines, etc.) are developed by designers and/or design teams. Many of the key concepts and aspects of DT have been identified through studies across different design domains and by design cognition and activity in both laboratory and natural contexts.1

The American psychologist and sociologist Herbert Simon described the
word “design” as “changing existing circumstances into preferred ones.” Together with other scientists, he laid the foundation for DT. DT is linked to improving the future in a creative process, without judgment, which consists of five basic steps: empathize, define, ideate, prototype, and test. Crazy ideas are welcomed as they can lead to creative solutions. In his book *Design Thinking*, Nigel Cross takes on the perspective of understanding how designers think and work and thereby gives us food for thought in a different direction. He challenges us to reflect upon whether we are designing to win, please, or use, and he collected evidence in deconstructing and watching what designers do. Cross quotes the Spanish engineer-architect Santiago Calatrava, who describes his iterative approach to design doing layer after layer as “very much a dialogue” (see Chapter 1). This dialogue is something I’ve come to experience in all my instances of working with DT, whether as a very literal interpretation of the term or applying the approach proposed and made popular by David Kelley, the founder of IDEO and the Stanford d.school. IDEO’s tool kit offers iterative steps we manage to learn a little framework and follow the steps that were already prescribed in a 1969 article. It is whether we manage to get out of our heads, manage to leave the constraints that we have gotten used to, and take the time to really feel our way into future users and customers and what they struggle with and delight in.

The IDEO perspectives that lead to new products and services are desirability (what might the user want?), feasibility (is that technically possible?) of empathy and experimentation and thereby facilitates a dialogue between developer and future customers (see Figure 1). Although taught linearly, they will happen in an interconnected, iterative, and looped manner.

Everyone is creative, says Kelley—and I agree. In my experience, though, the question is not whether we manage to learn a little framework and follow the steps that were already prescribed in a 1969 article. It is whether we manage to get out of our heads, manage to leave the constraints that we have gotten used to, and take the time to really feel our way into future users and customers and what they struggle with and delight in.

The IDEO perspectives that lead to new products and services are desirability (what might the user want?), feasibility (is that technically possible?) of empathy and experimentation and thereby facilitates a dialogue between developer and future customers (see Figure 1). Although taught linearly, they will happen in an interconnected, iterative, and looped manner.

Everyone is creative, says Kelley—and I agree. In my experience, though, the question is not whether we manage to learn a little framework and follow the steps that were already prescribed in a 1969 article. It is whether we manage to get out of our heads, manage to leave the constraints that we have gotten used to, and take the time to really feel our way into future users and customers and what they struggle with and delight in.

The IDEO perspectives that lead to new products and services are desirability (what might the user want?), feasibility (is that technically possible?) of empathy and experimentation and thereby facilitates a dialogue between developer and future customers (see Figure 1). Although taught linearly, they will happen in an interconnected, iterative, and looped manner.

Everyone is creative, says Kelley—and I agree. In my experience, though, the question is not whether we manage to learn a little framework and follow the steps that were already prescribed in a 1969 article. It is whether we manage to get out of our heads, manage to leave the constraints that we have gotten used to, and take the time to really feel our way into future users and customers and what they struggle with and delight in.

The IDEO perspectives that lead to new products and services are desirability (what might the user want?), feasibility (is that technically possible?) of empathy and experimentation and thereby facilitates a dialogue between developer and future customers (see Figure 1). Although taught linearly, they will happen in an interconnected, iterative, and looped manner.

Everyone is creative, says Kelley—and I agree. In my experience, though, the question is not whether we manage to learn a little framework and follow the steps that were already prescribed in a 1969 article. It is whether we manage to get out of our heads, manage to leave the constraints that we have gotten used to, and take the time to really feel our way into future users and customers and what they struggle with and delight in.

The IDEO perspectives that lead to new products and services are desirability (what might the user want?), feasibility (is that technically possible?) of empathy and experimentation and thereby facilitates a dialogue between developer and future customers (see Figure 1). Although taught linearly, they will happen in an interconnected, iterative, and looped manner.

Everyone is creative, says Kelley—and I agree. In my experience, though, the question is not whether we manage to learn a little framework and follow the steps that were already prescribed in a 1969 article. It is whether we manage to get out of our heads, manage to leave the constraints that we have gotten used to, and take the time to really feel our way into future users and customers and what they struggle with and delight in.

The IDEO perspectives that lead to new products and services are desirability (what might the user want?), feasibility (is that technically possible?) of empathy and experimentation and thereby facilitates a dialogue between developer and future customers (see Figure 1). Although taught linearly, they will happen in an interconnected, iterative, and looped manner.

Everyone is creative, says Kelley—and I agree. In my experience, though, the question is not whether we manage to learn a little framework and follow the steps that were already prescribed in a 1969 article. It is whether we manage to get out of our heads, manage to leave the constraints that we have gotten used to, and take the time to really feel our way into future users and customers and what they struggle with and delight in.

The IDEO perspectives that lead to new products and services are desirability (what might the user want?), feasibility (is that technically possible?) of empathy and experimentation and thereby facilitates a dialogue between developer and future customers (see Figure 1). Although taught linearly, they will happen in an interconnected, iterative, and looped manner.

Everyone is creative, says Kelley—and I agree. In my experience, though, the question is not whether we manage to learn a little framework and follow the steps that were already prescribed in a 1969 article. It is whether we manage to get out of our heads, manage to leave the constraints that we have gotten used to, and take the time to really feel our way into future users and customers and what they struggle with and delight in.

The IDEO perspectives that lead to new products and services are desirability (what might the user want?), feasibility (is that technically possible?) of empathy and experimentation and thereby facilitates a dialogue between developer and future customers (see Figure 1). Although taught linearly, they will happen in an interconnected, iterative, and looped manner.

Everyone is creative, says Kelley—and I agree. In my experience, though, the question is not whether we manage to learn a little framework and follow the steps that were already prescribed in a 1969 article. It is whether we manage to get out of our heads, manage to leave the constraints that we have gotten used to, and take the time to really feel our way into future users and customers and what they struggle with and delight in.

The IDEO perspectives that lead to new products and services are desirability (what might the user want?), feasibility (is that technically possible?) of empathy and experimentation and thereby facilitates a dialogue between developer and future customers (see Figure 1). Although taught linearly, they will happen in an interconnected, iterative, and looped manner.

Everyone is creative, says Kelley—and I agree. In my experience, though, the question is not whether we manage to learn a little framework and follow the steps that were already prescribed in a 1969 article. It is whether we manage to get out of our heads, manage to leave the constraints that we have gotten used to, and take the time to really feel our way into future users and customers and what they struggle with and delight in.
doable?), and viability (can we economically make a profit?). These perspectives have also received their fair share of criticism as they are perceived by some as wanting to teach entire methodologies (ethnography) by simply throwing out the tip to empathize with a future user. In addition, Vinsel argues that ideation sessions encourage positive thinking at the expense of critical thinking, such that DT is framed as fun work rather than the serious kind that leads to effective product design and development.

In a different line of argumentation, Helen Walters from Business Week critiques the article “Design Consultancies Hoped That a Process Trick Would Produce Change,” cited by Bruce Nussbaum in his advocating for Creative Intelligence as the next big framework. Mind you, this was already in 2011. While I have come across the term a few times since then, the framework is less simplistic and might therefore be harder to spread. Nevertheless, it is based on the same principles from Simon back in 1969. It also still falls for the same shortcomings that Jon Kolko points out:

Instead of empathy as the result of long-term immersion in a culture, as is the case of Pelle Ehn’s work in Scandinavia, we have two-hour “subject-matter expert” interviews where we gain a scratch-the-surface understanding of business needs. Instead of Osborn’s view of structured brainstorming, we have chaotic “working sessions.” Instead of Simon’s methodical understanding of how the human brain works, we have a “grip it and rip it” culture of test and iterate, abdicating proactive reflection for reactive alterations. Instead of a view of design as a way of understanding culture and carefully shaping it through craft and care, we appropriate it as a way of driving innovation through a relentless pursuit of newness. And instead of beautiful, usable, significant, and relevant designed things, we have “canvases” and “playbacks” and “design sprints”—and lots and lots of Post-it notes.

Looking at some of the big challenges humankind is facing, such as protecting the natural environment and ecosystems, further improving healthcare, equity, and social justice as detailed in the recent United Nations General Assembly sustainable development goals, we wonder whether DT can offer additional help and insights, as promoted by IDEO in their “Field Guide to Human-Centered Design.” While the danger of similar shortcomings is evident—we don’t learn to be an expert in a foreign culture in a matter of days—the Field Guide is a valuable first set of instructions for initial engagement, even if by no means encompassing or a replacement for domain experts.

The discussion that I would like to be the focus of this column is the search for alternatives and innovation for new solutions, and thereby, we come full circle back to creativity in general (Figure 2). Because, as Einstein famously said, “You cannot solve a problem from the same frame of consciousness that created it. You must learn to see the world anew.”

In 2017, we ran the First International Workshop on Design in Software Engineering, which included a design thinking prototyping workshop according to the d.school. It was on redesigning the gifting experience, the example d.school uses for their free crash course, and gave us plenty of discussion input. We enjoyed the sense of playfulness it brought but also questioned its applicability in serious scenarios as described in Vinsel’s critique, and we discussed the issue of efficiency versus creativity.

In 2018, we included a similar crash course workshop in the LUT summer school, and found that some students were focused on more traditional technical engineering constraints. In computer science education, we are more prone to teach our students computational thinking instead of systems thinking, although they could and should make use of both. By using only computational thinking, it is much harder to see broader systems contexts that can afford opportunities to investigate creative solution alternatives.

In 2019, my colleague Francisco Gomes ran a Design Jam Hackathon at the Chalmers and Gothenburg University Sustainability Week with 60 undergraduate students that was based on the rough design thinking process referenced above. The students had a great time developing ideas that would contribute to reaching the sustainable development goals. Creativity abounded, and now we hope they are following up and implementing some of their ideas.
For 2020, we are planning an evaluation session with Anneli Selvefors for her design thinking tool kit for circular economy that she beta tested with a self-selected group of participants at the Sustainability Week. Maybe approaches like hers, which specifically connect DT to sustainability-oriented business models, will help reorient toward meaningful impact.

**References**


Shortly after the end of the Second World War, there was a strong interest in improving the accuracy of weather forecasts. The importance of this was clear: if you can accurately predict storms, or any type of adverse weather, you have a much better chance to prepare for its effects and stay safe. George Dyson\textsuperscript{1} wrote about this in *Turing’s Cathedral*: “In 1945 meteorology had become a science, while forecasting remained an art....On average, forecasts beyond twenty-four hours were still no better than ‘persistence’—predicting that the weather tomorrow will be the same as it was today.”

By dividing a larger area into a grid of small cells, with weather data like wind speed, pressure, and temperature available for each cell, one could iteratively compute changing weather patterns over the entire area. Each cell would, step by step, only take the information from its neighboring cells into account to compute how the parameters gradually change over time. For sufficient detail, the cell sizes would have to be relatively small, which means that the amount of required computation would be massive, far exceeding what could be done with pen and paper by large numbers of human computers at the time.

Dyson quotes from a book written in 1922 by Lewis Richardson,\textsuperscript{2} who had been studying the potential of this type of method. The book, *Weather Prediction by Numerical Process*, predicted that “perhaps some day in the dim future, it will be possible to advance the computations faster than the weather advances, and at a cost less than the savings to mankind due to the information gained.” At the time, Richardson estimated that about 64,000 human computers would be needed to make reasonably accurate weather predictions in this way across the globe. Although Richardson called that “a staggering figure,” he probably had no idea just how far off the mark he still was.

The potential benefit of improved accuracy in weather predictions was not lost on John von Neumann either. In 1945, he used it as one of the selling points for his design of a new computer: the Institute for Advanced Studies (IAS) machine. At the time, von Neumann worked at the Institute for Advanced Studies in Princeton, where most work was focused on serious theoretical studies, so designing and building the hardware for an actual machine probably needed some strong justification.

Von Neumann recognized the need for more accurate weather predictions, and he realized that he could use it to motivate an initially perhaps somewhat skeptical audience that work on the development of computing machinery could be worth it, in due time. A sobering thought, perhaps, is that today, with the availability of massively more weather data, and virtually unlimited amounts of computer power compared to von Neumann’s days, we can still feel that the accuracy of weather prediction is often lacking. If the forecast says that it will be sunny for the next two weeks, few of us would be surprised if it still rained a few days later. Similarly, if the path of a hurricane is predicted to stay clear of your city one week out, it may still be wise to prepare for a hurried evacuation later, just in case. We can all agree that the task is both very important and very difficult. The trouble in this case is that there is inevitably an element of randomness in weather patterns, which is sometimes jokingly referred to as the *butterfly effect*.
The cumulative effects of numerous small amounts of randomness can make it very hard to make precise long-term predictions, no matter how fast our machines are or how much data on current and historical weather patterns are available.

**Predicting Software Failures**

There's another type of problem that is of similar importance, and that is to predict more accurately how software or software systems may fail. Also in this case, especially in the last few years, through sites like GitHub and so forth, we have gained access to massive amounts of data on source code, and through databases like the Common Weakness Enumeration website (https://cwe.mitre.org), we now also have access to large repositories of software defects with documented security implications. Should all this not allow us to build better and better predictors for flagging vulnerable code? Again in this case, the answer is mixed, but for somewhat different reasons than for accurate weather forecasting. Yes, we can scan code efficiently for the presence of any number of known vulnerabilities, and by doing so we can prevent the same types of vulnerabilities from causing harm. But no, we are far from the point where we can give an accurate metric of an application's safety. That is, if we exclude the easy answer to the questions: Can this code fail in unpredictable ways? Or is this system vulnerable to attack by a determined adversary? In both cases, the answer is, of course, with virtual certainty, yes.

With software and cybersecurity vulnerabilities, we do not just have the problem that human designers and software developers can and do make both small and large random mistakes in the applications they develop. There is an added obstacle that weather forecasters do not have to worry about: the presence of highly skilled and determined hackers around the world, sometimes directly sponsored by foreign governments, that make it their life's mission to overcome every obstacle put in their way to disrupt critical systems or to retrieve insufficiently protected data from them. They can afford to spend the time to pick apart software systems and to find just those nasty little random mistakes that make systems vulnerable to a takeover.

In his somewhat alarming new book, *Sandworm*, Andy Greenberg spells out the consequences of this phenomenon of cyberhacking when it is performed on a massive scale and directed at large organizations or even at entire countries. Once hackers gain access to the critical infrastructure of industrial control systems, and they've shown repeatedly that they can do so with apparent ease, the potential damage is no longer restricted to some stolen credit card information or Social Security numbers. The hackers can knock out power systems and paralyze the control systems of hospitals, banks, and election systems. Famously, in 2016, a group named the *Shadow Brokers* even succeeded in breaking into the highly protected systems at the National Security Agency to steal much of its secret arsenal of tools. Would they have any trouble breaking into your home computer?

**More Reliable Systems**

To return to our analogy with weather forecasting: it is important to know if the area where you live is in the path of an oncoming hurricane, even if the predictions cannot always be perfect. Any type of reasonable forecast is better than none. With cyberattacks, all we need to know is that this new type of hurricane is indeed coming, and it is coming our way with near certainty. But this is also a good reminder that reliability is fundamentally a systems property and not a component property. It is always wise to assume that any software application can fail or can be hacked, but that does not mean that the system of which the software is part is just as vulnerable. We must learn to design (more) reliable systems from unreliable components. Also in this case, even small improvements can be very meaningful.

The principles of building reliable systems from unreliable parts were learned long ago in the design of mechanical systems. A good example is the use of the centrifugal governor on steam engines, as popularized by James Watt. It is a small device that detects overpressure in the steam engine’s boiler and automatically opens a valve to release that pressure to prevent the boiler from exploding. Similarly, in optical systems design, we can use the flaws of one type of glass to correct for those of another and thus create a system of lenses that is better than any of its components. In electrical systems, for instance, in the design of signal amplifiers, we can duplicate the circuit to reduce the noise and improve the signal quality. The noise will be the one thing that differs in the output of the two subsystems that process the same input. Therefore, if we add the two outputs together, the original signal doubles in strength, but the noise signals (being uncorrelated) do not.

A simple method to set up your work system in such a way that it is protected better against complete
data loss in a targeted attack by a malicious adversary is to make sure that you always keep at least one trusted backup of your critical data completely offline, disconnected from all other devices and networks. Put more simply: don’t leave your backup disk conveniently connected to your computer.

How we can protect our key infrastructures, like power, telecommunications, financial, or even election systems, from silent hostile takeovers is still an open question. All of these systems rely on software to operate correctly, software that in most cases was not designed to be incorruptible. How can we apply the principles of reliable system design in these cases in particular? If Greenberg\textsuperscript{3} is right, we may not have a lot of time left to figure that out. ☹

References

Gerard J. Holzmann works on developing stronger methods for the design and analysis of safety-critical software as a consultant and researcher at Nimble Research. Contact him at gholzmann@acm.org.

IEEE Computer Society Has You Covered!

WORLD-CLASS CONFERENCES — 200+ globally recognized conferences.
DIGITAL LIBRARY — Over 700k articles covering world-class peer-reviewed content.
CALLS FOR PAPERS — Write and present your ground-breaking accomplishments.
EDUCATION — Strengthen your resume with the IEEE Computer Society Course Catalog.
ADVANCE YOUR CAREER — Search new positions in the IEEE Computer Society Jobs Board.
NETWORK — Make connections in local Region, Section, and Chapter activities.

Explore all of the member benefits at www.computer.org today!
Security Test

Christof Ebert, Youssef Rekik, and Rahul Karade

From the Editor
The Internet of Things connects devices with each other and cloud services to create new user experiences. Connectivity, however, invites cyberattacks, which are growing with the use of more standardized equipment, such as Ethernet and Linux software stacks. Risk-oriented security testing through a mix of methods and tools facilitates the transparent hardening of critical systems. Youssef Rekik, Rahul Karade, and I provide an overview of industry-scale technologies for security testing. I look forward to hearing from readers and prospective authors about this column and the technologies you want to know more about. — Christof Ebert

CYBERCRIMINALS CAN BREAK into any connected system. Traditionally, IT systems with their many open interfaces had been in the focus of attackers, while embedded systems were perceived to be too difficult to hack and not worth the time and energy required. But as systems have added Ethernet, WLAN, USB, Bluetooth, GPS, and other connectivity features, the number of attack surfaces has increased. The most popular hacking method involves attacking a diagnosis port, or otherwise open interface, which can give a malevolent party access to functions or, at least, the ability to corrupt data and prohibit performance such as denial-of-service attacks.

The pressure to deliver products as fast as possible, combined with increasingly open architectures and overwhelming complexity, has further weakened the quality and security of Internet of Things (IoT) systems across industries. Today, medical devices, such as insulin pumps and pacemakers, are at risk, as are cars, industry production facilities, and wide, distributed utility systems. The more our society depends on connectivity the more we are at risk of being hit by major attacks that have the potential not only to damage single systems but entire cities and countries. Imagine a major breakdown of electric utilities. Such an event would immediately disable the water supply and, thus, threaten life in the impacted area.

Risk-oriented security with dedicated test methods and appropriate tools is the call of the day. Security testing must start with static code analysis, proceed with unit tests, and further advance through dedicated methods, such as fuzzing and robustness evaluations up to the level of penetration testing (PenTest). Let us briefly introduce risk-oriented security engineering and delve into the appropriate test methods and tools.
We will focus on novel gray-box penetration techniques that bridge threat analysis and more efficient and effective testing.

As with all verification and validation methods, cybersecurity testing requires deep experience and competence to select the best methods and test the end criteria as well as a lean yet effective regression strategy capable of continuous integration and deliveries. Often, we see companies that test components and their interfaces while overlooking security threats in networking and services. We have enriched this article with experiences from our security-consulting projects.

Convergence and Cybersecurity

The convergence of IT and embedded systems has opened many doors for criminals, literally speaking.1–3 Hacking tools are easily available, even in online shops. They are sold on the normal web. Software-defined radio technologies for man-in-the-middle attacks that mimic and intercept signals are available, as are code grabbers. Forums provide complete tutorials on breaking into utilities and stealing vehicles. For instance, automotive hacking tools and online support are available on websites including Omerta.cc, Nulled.to, ffffff.ru, and Chipadla.ru.

Converging IT and embedded systems and devices, such as cars, transport vehicles, medical equipment, industry automation, and utilities networks, might not easily be broken while running in a stable, disconnected mode. Yet the connectivity infrastructure can and will be used to attack any connected device, such as an automotive electronic control unit (ECU). By breaching the IT servers used for software updates, remote control, and maintenance, hackers can load malware and corrupt data. Attackers have even manipulated cellular networks through built-in subscriber-identification-module cards, which many IT companies use to connect with real-time information and update firmware.

The major problem with these attacks concerns not only data security and privacy but, in most cases, functional safety. Embedded devices that have control systems, such as cars and medical implants, are by definition safety critical. The basic principle, therefore, is simple to understand yet difficult to achieve: There is no functional safety without security. Based on the specific challenges of automotive security, original equipment manufacturers and suppliers must realize effective protection against manipulations of the converging embedded and IT systems.

Let us look to automotive systems since they exhibit the meeting of IT and embedded systems most prominently. Cars demand functional safety. The engine, steering, and braking are influenced by numerous embedded computers. Assistance systems enhance drivers’ capabilities through features such as advanced cruise control, platooning, and automatic parking. In the future, we will face fully autonomous cars that will depend even more on IT systems inside and outside. Intercepting their communications or, maybe accidentally, corrupting their data would mean that their initial safety case was no longer valid, which is reason enough to explore technologies for security testing.

Risk-Oriented Cybersecurity

Risk-oriented security helps to balance the growing threats against increasing complexity during the entire lifecycle. Unlike many previous attempts, our research and many practice projects indicate that while security by design is good, it is not good enough. Effective security must handle the entire lifecycle. To cover the major safety hazards that result through security misuse and abuse scenarios for automobiles, we combine safety and security engineering (Figure 1). While safety and security need their own respective methodologies, we suggest that organizational infrastructures and governance schemes at present should be combined for efficiency and effectiveness. We see this as an evolution toward a full (independent) cybersecurity organization in the lifecycle.4,5

Cybersecurity testing in this space connects security requirements directly to design decisions, following the triple peak model of combined requirements analysis, solution modeling, and test-oriented requirements engineering.6 This ensures full traceability from the initial threat analysis and risk assessment (TARA) and definition of security requirements. From a compliance and governance perspective, we see this approach as helpful since it illustrates the necessity to prove that security requirements and decisions have been
adequately verified through each regression. For instance, at Vector Consulting, we introduced this risk-oriented cybersecurity methodology to a leading supplier in a highly safety-critical environment and ensured that the road-test cycle time could be dramatically reduced.

Security testing can never be complete. The different test strategies, from white-box static analysis to unit tests, fuzzing, and the PenTest, must balance the cost of not having enough security and being attacked, with all of the damaging consequences, versus the expense to implement appropriate security mechanisms and keep them updated through the lifecycle.

**Security-Test Technologies**

Security testing is divided into phases, each requiring different tools. Therefore, it is essential to know the tools’ strengths and limitations to choose the best one per our requirements. Table 1 provides an overview of test tools that are in wide use. As is the tradition of this column, we examine established tools and newer ones that are gaining popularity. Some tools that are used in more than one industry are chosen to understand their collective impact. There are open-source and commercial tools, some that are fully automated, and some that are offered in the form of software as a service. The tools were rated “essential” because they were effective and well supported. Specifically, we examined quality attributes such as usability, scalability, and update availability.

**Usability**

A tool should provide an interactive GUI and command line to better facilitate understanding and training. It should support the preparation of detailed reports and graphs to show the risks and exploits associated with each weakness found during the PenTest. Automating some common procedures should be possible. Frameworks such as the popular Metasploit are introducing all of the methodologies required to carry out each phase of the PenTest process, helping to make obsolete the tools dedicated to a single phase. The Nmap tool, which is exclusively used for reconnaissance scanning, is improving in its methodology to provide information about vulnerabilities that is as detailed as possible.

**Scalability**

It is essential for the penetration testing tools to support a variety of programming languages and network and application protocols. These tools must provide APIs to extend penetration testing to multiple hardware and software targets. The portability of the software helps in reaching a greater number of researchers. The necessary argument one can make is that the system provided by a company should be

![Diagram](image.png)

**FIGURE 1.** Efficiently connecting safety and security.

1) Assets and their value are the starting point.
2) Threats are classified according to required attack potential and severity.
3) Goals are the output of both techniques.
Table 1. The security-testing tools.

<table>
<thead>
<tr>
<th>Tool name</th>
<th>Application domain</th>
<th>Methodology</th>
<th>URL</th>
<th>Supplier</th>
<th>Portability</th>
<th>Usage scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metasploit</td>
<td>IT, IoT, medical</td>
<td>Scanning and exploitation</td>
<td><a href="https://www.metasploit.com/">https://www.metasploit.com/</a></td>
<td>Rapid7</td>
<td>✓ ✓ ✓</td>
<td>• Test and exploit vulnerabilities in operating systems and applications</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Develop and execute exploit code against a remote target</td>
</tr>
<tr>
<td>CANoe</td>
<td>Automotive</td>
<td>Testing and analysis of controllers and networks, including dedicated security testing, such as fuzzing</td>
<td><a href="https://www.vector.com/int/en/products/products-a-z/software/canoe/">https://www.vector.com/int/en/products/products-a-z/software/canoe/</a></td>
<td>Vector</td>
<td>✓</td>
<td>• ECU and network of ECUs development, simulation, and testing</td>
</tr>
<tr>
<td>BurpSuit Professional</td>
<td>IT</td>
<td>Web security and network scanning</td>
<td><a href="https://portswigger.net/burp/pro">https://portswigger.net/burp/pro</a></td>
<td>Portswigger</td>
<td>✓ ✓ ✓</td>
<td>• Web security and network scanner</td>
</tr>
<tr>
<td>Scapy</td>
<td>IT, automotive</td>
<td>Scanning and exploitation</td>
<td><a href="https://scapy.net/">https://scapy.net/</a></td>
<td>Scapy</td>
<td>✓ ✓ ✓</td>
<td>• Scanning and exploitation of networked devices and automotive ECUs</td>
</tr>
<tr>
<td>BreakingPoint</td>
<td>IT, automotive, energy</td>
<td>Applications and network-security testing</td>
<td><a href="https://www.ixiacom.com/products/network-security-testing-breakingpoint">https://www.ixiacom.com/products/network-security-testing-breakingpoint</a></td>
<td>Ixia</td>
<td></td>
<td>• Scanning and exploitation to validate network performance and security posture</td>
</tr>
<tr>
<td>AppScan</td>
<td>IT</td>
<td>Dynamic and static security testing</td>
<td><a href="https://www.hcltechsw.com/wps/portal/products/appscan/home">https://www.hcltechsw.com/wps/portal/products/appscan/home</a></td>
<td>HCL</td>
<td>✓</td>
<td>• Exploitation of web and mobile applications</td>
</tr>
<tr>
<td>Netsparker</td>
<td>IT</td>
<td>Dynamic analysis</td>
<td><a href="https://www.netsparker.com/">https://www.netsparker.com/</a></td>
<td>Netsparker</td>
<td>✓</td>
<td>• Scanning of web interfaces</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• New protocol definition using XML</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Test applications and hardware, multiprotocol fuzzer</td>
</tr>
<tr>
<td>Defensics</td>
<td>IT, automotive</td>
<td>Comprehensive fuzzing and vulnerabilities-scanning framework</td>
<td><a href="https://www.synopsys.com/software-integrity/security-testing/fuzz-testing.html">https://www.synopsys.com/software-integrity/security-testing/fuzz-testing.html</a></td>
<td>Synopsys</td>
<td>✓ ✓ ✓</td>
<td>• Black-box fuzzing to test APIs and services, discovering and remediating unknown vulnerabilities in software and devices</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Software, telecommunication, CAN, and IP protocols fuzzing</td>
</tr>
</tbody>
</table>

API: application programming interface; CAN: controller-area network.
<table>
<thead>
<tr>
<th>Tool name</th>
<th>Domain</th>
<th>Application</th>
<th>Testing and analysis</th>
<th>Usage scheme</th>
<th>Methodology</th>
<th>Strengths</th>
<th>Limitations</th>
<th>License</th>
<th>Cost</th>
<th>Notable qualities</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppScan</td>
<td>IT, automotive</td>
<td>IT</td>
<td>Scanning and exploitation of software, telecommunication, energy</td>
<td>• Independent of operating system</td>
<td>• Embedded device-protocol fuzzing</td>
<td>• Up-to-date database of known vulnerabilities and exploits</td>
<td>• Limited capabilities for the free version</td>
<td>Open source, commercial</td>
<td>Low</td>
<td>Comprehensive GUI improves the usability. Uses dedicated security test methods.</td>
</tr>
<tr>
<td>BurpSuit</td>
<td>IT, IoT, medical</td>
<td>IT</td>
<td>Scanning of web interfaces</td>
<td>• Difficult to configure</td>
<td>• Embedded device-protocol fuzzing</td>
<td>• Automated as well as manual testing</td>
<td>• Currently limited to automotive security testing</td>
<td>Commercial</td>
<td>High</td>
<td>Ease of use and effective vulnerability scanning</td>
</tr>
<tr>
<td>CANoe</td>
<td>IT</td>
<td>Automotive, IT</td>
<td>Dynamic and static analysis</td>
<td>• Complex test language</td>
<td>• Supports multiple network protocols</td>
<td>• Large number of supported application protocols</td>
<td>• Can’t handle a large number of packets simultaneously</td>
<td>Open source</td>
<td>Low</td>
<td>Modular and extensible to other protocols</td>
</tr>
<tr>
<td>Defensics</td>
<td>automotive</td>
<td>IT, automotive</td>
<td>Black-box fuzzing to test APIs</td>
<td>• Partial support for certain complex protocols</td>
<td>• Supports multiple network protocols</td>
<td>• It highlights, with several grades of severity, the types of vulnerabilities.</td>
<td>• Scans may become slow on large websites.</td>
<td>Commercial</td>
<td>Medium</td>
<td>Reliable with low number of false positives</td>
</tr>
<tr>
<td>BESTORM</td>
<td>IT</td>
<td>IT</td>
<td>Efficient threat detection</td>
<td>• Using the recommended Ixia hardware makes it expensive.</td>
<td>• Large number of supported application protocols</td>
<td>• It identifies a wide area of vulnerabilities</td>
<td>• Scans may become slow on large websites.</td>
<td>Commercial</td>
<td>High</td>
<td>Highly customizable and in-depth reporting</td>
</tr>
<tr>
<td>HCL</td>
<td>IT, automotive</td>
<td>IT</td>
<td>Scan applications and hardware, including networks and automotive ECUs</td>
<td>• Requires personnel training</td>
<td>• Embedded device-protocol fuzzing</td>
<td>• It identifies a wide area of vulnerabilities</td>
<td>• Scans may become slow on large websites.</td>
<td>Commercial</td>
<td>Medium</td>
<td>Efficient threat detection</td>
</tr>
<tr>
<td>Ixia</td>
<td>IT, automotive</td>
<td>IT</td>
<td>Efficient threat detection</td>
<td>• Using the recommended Ixia hardware makes it expensive.</td>
<td>• Embedded device-protocol fuzzing</td>
<td>• It identifies a wide area of vulnerabilities</td>
<td>• Scans may become slow on large websites.</td>
<td>Commercial</td>
<td>High</td>
<td>Highly customizable and in-depth reporting</td>
</tr>
<tr>
<td>Metasploit</td>
<td>IT, automotive</td>
<td>IT</td>
<td>Efficient threat detection</td>
<td>• Using the recommended Ixia hardware makes it expensive.</td>
<td>• Embedded device-protocol fuzzing</td>
<td>• It identifies a wide area of vulnerabilities</td>
<td>• Scans may become slow on large websites.</td>
<td>Commercial</td>
<td>High</td>
<td>Efficient threat detection</td>
</tr>
<tr>
<td>Netsparker</td>
<td>IT</td>
<td>IT</td>
<td>Efficient threat detection</td>
<td>• Using the recommended Ixia hardware makes it expensive.</td>
<td>• Embedded device-protocol fuzzing</td>
<td>• It identifies a wide area of vulnerabilities</td>
<td>• Scans may become slow on large websites.</td>
<td>Commercial</td>
<td>Medium</td>
<td>Reliable with low number of false positives</td>
</tr>
<tr>
<td>Scapy</td>
<td>IT, automotive</td>
<td>IT</td>
<td>Efficient threat detection</td>
<td>• Using the recommended Ixia hardware makes it expensive.</td>
<td>• Embedded device-protocol fuzzing</td>
<td>• It identifies a wide area of vulnerabilities</td>
<td>• Scans may become slow on large websites.</td>
<td>Open source</td>
<td>Low</td>
<td>Modular and extensible to other protocols</td>
</tr>
<tr>
<td>Synopsys</td>
<td>IT, automotive</td>
<td>IT</td>
<td>Efficient threat detection</td>
<td>• Using the recommended Ixia hardware makes it expensive.</td>
<td>• Embedded device-protocol fuzzing</td>
<td>• It identifies a wide area of vulnerabilities</td>
<td>• Scans may become slow on large websites.</td>
<td>Commercial</td>
<td>High</td>
<td>Efficient threat detection</td>
</tr>
<tr>
<td>Vector</td>
<td>IT</td>
<td>IT</td>
<td>Efficient threat detection</td>
<td>• Using the recommended Ixia hardware makes it expensive.</td>
<td>• Embedded device-protocol fuzzing</td>
<td>• It identifies a wide area of vulnerabilities</td>
<td>• Scans may become slow on large websites.</td>
<td>Commercial</td>
<td>High</td>
<td>Efficient threat detection</td>
</tr>
</tbody>
</table>

compatible with other systems. This enhances the scope of the testing tools.

### Availability

A tool’s license and cost are important aspects. Tools that continuously update their vulnerability and exploit databases are recommended. Those that contain an extensive threat library and multivector testing capabilities are better choices for security testing.

Black-box testing has been the norm for security analysis across all industries. Although risk identification is almost always desirable through the PenTest, the tools rarely support the gray-box PenTest because the supply chain is very complex in automotive and aerospace industries, and multiple suppliers provide specific embedded devices for dedicated functionalities. To protect their intellectual property, suppliers are reluctant to hand over their device architectures, which makes it difficult for them to use the white- and gray-box PenTest. As a result, most of the tools follow the black-box PenTest methodology. Our gray-box PenTest methodology, which combines risk analysis, architecture evaluation, and security testing, has proven highly efficient and effective to detect vulnerabilities.

Cybersecurity for connected systems has gained huge relevance with the convergence of IT and embedded systems.

### Table 1. The security-testing tools (cont.).

<table>
<thead>
<tr>
<th>Tool name</th>
<th>Application domain</th>
<th>Methodology</th>
<th>URL</th>
<th>Supplier</th>
<th>Portability</th>
<th>Usage scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nmap</td>
<td>IT</td>
<td>Information gathering</td>
<td><a href="https://nmap.org/">https://nmap.org/</a></td>
<td>Nmap</td>
<td>✓</td>
<td>• Network scanning • Port scanning</td>
</tr>
<tr>
<td>AttifyOS</td>
<td>IoT</td>
<td>IoT PenTest framework</td>
<td><a href="https://www.attify.com/attifyos">https://www.attify.com/attifyos</a></td>
<td>Attify</td>
<td></td>
<td>• IoT devices and connectivity PenTests</td>
</tr>
<tr>
<td>IT, IoT, medical</td>
<td></td>
<td>Network and application PenTest framework</td>
<td><a href="https://www.kali.org/">https://www.kali.org/</a></td>
<td>Kali by Offensive Security</td>
<td></td>
<td>• Network and applications PenTest</td>
</tr>
<tr>
<td>CANBadger</td>
<td>Automotive</td>
<td>ECU PenTest</td>
<td><a href="https://github.com/Gutenshit/CANBadger/wiki/Getting-the-board-ready">https://github.com/Gutenshit/CANBadger/wiki/Getting-the-board-ready</a></td>
<td></td>
<td>✓</td>
<td>• ECU hacking using man-in-the-middle attack and hijacking security access</td>
</tr>
<tr>
<td>Saleae Logic 8</td>
<td>Embedded devices</td>
<td>Firmware and logic analyzer</td>
<td><a href="https://www.saleae.com/">https://www.saleae.com/</a></td>
<td>Saleae</td>
<td>✓</td>
<td>• Embedded device logic analysis</td>
</tr>
</tbody>
</table>
Because of the introduction of classic IT attack surfaces and vulnerabilities to critical infrastructures, the amount of attacks is fast growing. Since they are used across industries that have great relevance to our society, such systems must be thoroughly protected and hardened. Safety requires security as a mandatory condition, which means that any safety-critical system, at a minimum, needs to be protected. Security must be integrated early in the design phase to understand the threats and risks to embedded functions. Security today is mandatory due not only to its safety impact but product liability. It is not excusable anymore to say that hacking is inevitable. We must protect connected systems as best we can and prove that we have taken the necessary actions in terms of processes, education, management, and technology. Testing plays a critical role in this process.

In this article, we looked to the testing environments for hardening. Specifically, we investigated current test tools, such as Metasploit, and evaluated how they can be used for security testing in converging IoT systems. While traditional security testing took a black-box approach, we recommend a grey-box methodology building upon a TARA and known component, interface, and network vulnerabilities. We showed how an initial security analysis and technical concept based on a given reference architecture shows threats and risks...
and is used to guide mitigation. We focused specifically on typical Ethernet-protection mechanisms, such as firewall policies, IDS/IPS and VLAN. Adding intelligence to testing tools by introducing machine- and deep-learning concepts will benefit the process by improving metrics including speed, the response rate, the vulnerabilities found, and so forth.

We strive not only to provide guidance for specific misuse cases but to change the mentality of embedded-systems engineers toward designing for security rather than functionality. With the convergence of IT and embedded systems among industries, cybersecurity is a major requirement. Isolated mechanisms, such as distributed functionality in proprietary subsystems, protection at the component level, gateways and firewalls between components, and the validation of critical functions, is insufficient. Software-process evangelist Tom Gilb once observed, “If you don’t actively attack risks, they will actively attack you.” That mind-set should guide us toward improving security. Cybersecurity can never be comprehensive, but it can be vastly improved through risk-oriented testing with an optimized mix of strategies and tools.

References

Design Thinking

Annie Combelles, Inspearit
Christof Ebert, Vector Consulting Services
Percival Lucena, IBM Research
CONTINUOUSLY INNOVATING
THEIR portfolio is a key challenge for practically all companies. Regardless of size, companies today rely on the ability to innovate to maintain their competitiveness. Their performance depends on efficiency when mobilizing internal knowledge and fostering collective intelligence. In that sense, design thinking, with its mix of analytics and intuition, provides valuable support to product management and is a necessary step to improve user experience.

Design thinking is an innovation and problem-solving method. It is based on iterative usage of creative design methods, such as problem reframing and early prototyping, and involves different stakeholders (Figure 1). Across industries, it has shown its value as a methodology for generating innovative ideas. Currently, the method extends to software and IT and is part of the agile method framework. Building upon these early experiences, design thinking is used in software engineering because it pragmatically connects problem understanding, solution exploration, and innovation.

With its growing relevance in agile software development, in this column, we present a snapshot of the underlying design thinking methodology along with an analysis on the state of its practice, usage schemes, lessons learned, risks, and benefits. This theme issue describes the underlying methods, context, and practical application of design thinking.

For a long time, prototypical problem solving has been a bridge between customer/market needs and product engineering. Japanese marketing expert Noriaki Kano realized as early as the 1970s that often times customer needs cannot be articulated. At the same time, we waste energy in product development to build consecutive releases using predictable enhancements of functions. With its lean development, car manufacturer Toyota focused on what matters most: controlling costs and keeping things simple, while at the same time enhancing user experience. Industry icon Steve Jobs followed these principles and redefined existing red-ocean products, such as MP3 players and mobile phones, by leaving out many familiar functions and adding a few exciting features. Design thinking further extends these principles to explore the possibilities of what could be and to create outcomes that benefit the user.

The products of tomorrow must provide continuous choices that balance the changing needs/usage of individuals, markets, and society. Industrial software development is still driven by huge feature lists that are frequently extended. Market excitement (in terms of innovative products) is not commensurate with the growing complexity of delivered systems. Here is where design thinking extends current software engineering practices.

Using the Twin Peaks model of problem analysis and solution creation, design thinking looks to control risk and uncertainty by developing a series of prototypical frames to progressively meet customer expectations. Along with this Twin Peaks analogy, design thinking connects analytical problem solving with creative idea finding and iterative solution modeling. It helps to capture design rationales and builds upon the reflective techniques used in agile development. By drawing upon prototyping and agile teamwork using the iterative reframing of problems, design thinking enhances more traditional techniques such as Scrum and feature-driven development. The multidimensional perspectives of design thinking benefit agile scaling methods such as Large-Scale Scrum, Scaled Agile Framework, and Scrum of Scrums in their corporate and portfolio-oriented approaches.

Cocreation, as a core element of design thinking, maps to data exploration in big data and data analytics. As such, design thinking is holistic by nature and brings various stakeholders to the table to redefine value creation processes and to continuously explore how to build the best product.

Because product development-related artifacts such as code, data, and specs are continually shaped in a process of learning, recalling, and accepting, the materialized artifacts derived from prototyping can capture an agreement on terms while

FIGURE 1. The design thinking method.
remaining comprehensive to all parties. For instance, when best practices for user interface (UI) designers suggest the use of sketched paper prototypes to discuss UIs with end users rather than polished screen-shots or even the actual UI. These software engineering-driven design thinking techniques are currently enhanced to provide a more comprehensive media modeling, which are used in various tool extensions such as business process modeling and design modeling.

Design thinking makes it possible to not rush into the product design but to dedicate upstream time to generate a maximum number of ideas against the identified needs and to propose the elements upon which to build a response element to users. Multidisciplinary groups that guarantee a diversity of points of view must also be set up. Then, a prototyping phase allows for checking the feasibility of ideas and involving users, their feedback being a central element of design thinking. Finally, with the transition from prototype to product development being a pivotal step full and inherently risky, the design thinking approach limits the loss of information due to the involvement of development teams in the product design phase.

This theme issue on design thinking presents the principles of design thinking and examines how they are practically applied to software development. We received 21 submissions, from which we selected three manuscripts. In line with the IEEE Software mission that embraces leading practitioners, we selected manuscripts that provide hands-on content based on actual industry experiences. The submitted metastudies and mere research articles could not be used because they did not offer enough takeaways for those who want to use design thinking. To amplify industry relevance, we invited a point–counterpoint debate among three practitioners from IBM to share their good and not-so-good experiences (see the “Point–Counterpoint” column on page 49 in this issue).

The three articles plus the “Point–Counterpoint” column included in this theme issue highlight some key aspects of the design thinking methodology, which could be useful to combine with existing techniques. One article describes an industrial case and proposes interesting lessons learned.

Integrating design thinking and requirements engineering is proposed in the first article, which was written by a group authors from different institutes of technology. Hehn et al. insist on the complementary aspects of both techniques; for example, design thinking may help in the requirements-elicitation phase, capturing how ideas and functionalities have emerged is beneficial. The authors suggest that using continuous design thinking becomes a good procedure. The article concludes with an interesting section for practitioners: “Open Challenges.” In it, the authors structure their observations in a table according to three approaches: upfront design thinking, infused design thinking,
and continuous design thinking, including a case study for each.

The second article, authored by a team from Montréal, is an experience report from a company that transformed its software development factory to integrate the design thinking technique. Mahé et al.’s article complements the first, and its primary intent is to improve the requirement-elicitation processes. But in addition, due to the company’s rapid growth, design thinking was considered to facilitate efficient collaboration, knowledge transfer, and foster team engagement. The article lists a few prerequisites and impacts derived from the experiment, concluding in positive human interactions and demonstrated business advantages.

Finally, Kuula et al. article can be viewed as a contribution to digital transformation, one of the popular current focuses of many organizations. Being methods oriented only, the article addresses a real case of design thinking implementation for continuous value creation. It describes a systematical service design approach where digitalization drives service business models. The authors provide interesting observations and feedback about the cyclic and iterative journey and close with a thorough evaluation of the solution proposed.

The three articles and “Point-Counterpoint” form a constructive learning package for practitioners, providing an overview of the design thinking technique first, followed by thought-provoking ideas about the design thinking mindset and human interactions for iterative cocreation activities. It is probably too early in the industrial deployment of such techniques to expect more than experience testimonies and a set of good practices; however, the time will come when quantitative impact measures emerge to fully back up this framework.

For more information on paper submission, featured articles, calls for papers, and subscription links visit: www.computer.org/tsusc
On Integrating Design Thinking for Human-Centered Requirements Engineering

Jennifer Hehn, University of St. Gallen
Daniel Mendez, Blekinge Institute of Technology
Falk Uebernickel, University of Potsdam/HPI
Walter Brenner, University of St. Gallen
Manfred Broy, Technical University of Munich

// We elaborate on the possibilities and needs to integrate design thinking into requirements engineering, drawing from our research and project experiences. We suggest three approaches for tailoring and integrating design thinking and requirements engineering with complementary synergies and point at open challenges for research and practice. //

REQUIREMENTS ENGINEERS OFTEN face the challenge of discovering and satisfying the fuzzy needs and volatile requirements of the various stakeholders involved. Design thinking (DT), as a human-centered, rapid-prototyping method for innovative design, is one promising approach to address this challenge.1,2

We postulate that we need an effective integration of DT and requirements engineering (RE). However, little is known about how an integration could be realized considering a holistic view, also due to existing misconceptions. In DT, too many times, we tell ourselves problem solving ends with understanding the problem and building a nontechnical prototype, leaving open the seamless transition into software development endeavors. In RE, we pretend too often that software requirements are somehow “just there” and simply need to be elicited, missing the potential of fully exploring the problem space. One difficulty to be taken into account is that, like with other “agile” approaches, DT can appear as a set of single methods, tools, or even as a holistic approach.3 RE, in turn, is an engineering discipline encompassing various principles, tools, and even more methods—all to be selected depending on given project situations and software process models.4 To make effective use of the full potential of DT, we first need a better understanding of what it is and how it relates to RE, in which situations it might be suitable, and how it could be properly integrated.

For years, we accompanied and researched organizations that adopted DT in their industrial settings. In this article, we share our experiences and outline synergies and differences between DT and RE with a model of artifacts that emerges from industrial adoptions. We recommend three integration strategies before concluding with open questions for research and practice.

DT in a Nutshell
DT is a structured problem-solving approach to develop innovative
products, services, and business models. It builds upon the exploration of human needs, nontechnical prototyping, iterative problem reframing, and interdisciplinary teamwork. DT is primarily intended to be applied in project settings known as *wicked*, which are characterized by volatile and partially unidentified/hidden requirements. Our own scope and experiences are centered in the development of software-intensive products in industrial projects as well as in practical courses at the University of St. Gallen. There, we employ an iterative and multidisciplinary approach widely known as the DT double diamond shown in Figure 1.

We distinguish the problem space from the solution space, each with exploratory (diverging) and defining (converging) activities. The problem space contains methods to capture the problem in a human-centric, empathic manner. The terms *define* and *needfinding* explore the user and business environment, while the term *synthesis* condenses the gathered information to potential opportunities for the discovered needs. The solution space contains methods to develop ideas, build prototypes, and systematically test them, termed *ideation*, *prototyping*, and *testing*, respectively. Iterations are carried out wherever necessary in the process. Prototypes evolve from rudimentary, and often paper-based, low-fidelity prototypes to more sophisticated, technical ones at later stages. The ability to conduct this stepwise improvement of assumptions, ideas, and prototypes fundamentally relies on an open communication environment that is leveraged by harnessing selected tools and techniques like the ones summarized in our online material compendium (www.dt4re.org).

**Cross Comparison of DT and RE**

**Two Complementary Approaches**

To compare both approaches, it needs to be clear what is being achieved through DT and through RE and how the output can be transformed into an actual product. Based on our experiences as practitioners who train companies in these methods, we created a blueprint of relevant artifacts from both approaches (Figure 2). The artifacts describe the produced work results and their dependencies and, thus, abstract from complex development processes, which are barely comparable across projects.

Our model structures the artifacts according to why a system is needed (named the *context layer*), which user-level requirements and features are necessary (referred to as the *requirements layer*), and how the system is to be realized (called the *system layer*). The full model consists of 40 artifacts: 16 attributed to DT, 16 to RE, and eight to both (for details, see the supplementary material “Description of Content Items in the Combined Artifact Model” that accompanies this article in IEEE Xplore).

We see various commonalities between DT and RE, at least if the latter is understood as an iterative approach. The differences should be seen as complementary activities. Design thinking expands the toolbox for RE by emphasizing artifacts describing the relevance of the system vision (context layer). They
complement the more technical RE artifacts with a human-centered perspective.10 Also, RE expands the toolbox of DT. The technical realization of the functionalities is not within the scope of DT and is rather attributed to the artifacts produced in RE (requirements and system layers). Based on our model, we suggest the following.

- **Use DT to guide your requirements elicitation:** The DT process model (Figure 1) can help produce relevant context artifacts for a comprehensive understanding of the problem. Use the high-fidelity prototype to visualize the system, including key functionalities and the general form of user interaction. In addition, the prototype serves usability-driven demonstration purposes to customers rather than technically driven feasibility studies.

- **Embrace the learning curve of DT to inform implementation:** From an implementation point of view, understanding how ideas and functionalities have emerged helps design the solution vision in the intended form. For example, field study results and insights can inform use cases and scenarios—typically defined in RE—or user stories when working in a more agile way.

---

**FIGURE 2.** The combined artifact model of DT and RE.
Table 1. Strategies for integrating DT and RE.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Upfront DT</th>
<th>Infused DT</th>
<th>Continuous DT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used at an early project stage to provide clarity for unclear user needs and</td>
<td>Used within an existing RE process to gain new ideas or clarify fuzzy</td>
<td>Continuous usage of DT and RE elements to realize end-to-end view from</td>
<td></td>
</tr>
<tr>
<td>to define a solution vision</td>
<td>requirements</td>
<td>user need to solution vision to functional solution</td>
<td></td>
</tr>
<tr>
<td>Uses DT as guiding process</td>
<td>Uses DT as toolbox in RE</td>
<td>Uses DT as guiding mind-set</td>
<td></td>
</tr>
<tr>
<td>Requires setup of a (mini) project</td>
<td>Requires setup up of focused workshops</td>
<td>Combines upfront and infused strategy and setup of new role</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Outcome</td>
<td>Clear solution vision in form of a mockup</td>
<td>Outcome situation-dependent [e.g., (new) features, user requirements, and</td>
<td>Software requirements specification based on and traceable to customer needs</td>
</tr>
<tr>
<td></td>
<td>Comprehensive set of DT artifacts (Figure 2) as basis to perform further RE</td>
<td>test with users, all along the RE process]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>activities</td>
<td>Selected set of DT artifacts (Figure 2)</td>
<td>Comprehensives set of DT and RE artifacts as shown in Figure 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Duration</td>
<td>Approximately 3–12 weeks</td>
<td>Approximately 1–10 days</td>
<td>More than six months</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key roles</td>
<td>DT team (4–6) from different areas of expertise</td>
<td>Workshop participants (5–20) from different areas of expertise</td>
<td>See upfront and infused strategies</td>
</tr>
<tr>
<td></td>
<td>Extended team of experts and project sponsor: offers internal expertise and</td>
<td>DT coach: provides process guidance</td>
<td>Human-centric requirements engineer: New role incorporating DT expertise</td>
</tr>
<tr>
<td></td>
<td>defines initial design challenge</td>
<td>Requirements engineer as mediator between DT team and software development</td>
<td>as well as RE expertise and mediating between both schools of thoughts.</td>
</tr>
<tr>
<td></td>
<td>DT coach: provides process guidance</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Benefits</td>
<td>Full potential of DT leveraged while no changes to RE are necessary</td>
<td>Only minimal changes in existing RE practices are required</td>
<td>Seamless integration including development-critical artifacts</td>
</tr>
<tr>
<td></td>
<td>Solution concept with traceable links to user customer needs</td>
<td>Resource and time friendly</td>
<td>Human-centered mind-set throughout entire project</td>
</tr>
<tr>
<td></td>
<td>Deep context understanding achieved</td>
<td>Low adoption hurdle for DT</td>
<td>Precise and traceable (user) requirements through continuous identification of new requirements and testing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Challenges</td>
<td>Resource and time intense</td>
<td>Risk of neglecting problem context</td>
<td>Commitment, resource, and time intense</td>
</tr>
<tr>
<td></td>
<td>Lost implicit knowledge when handing over prototype</td>
<td>Missing sustainable impact of DT</td>
<td>Highly team dependent</td>
</tr>
<tr>
<td></td>
<td>No to little attention on development-critical artifacts</td>
<td>No to little attention on development-critical artifacts</td>
<td>Requires organizational mind-shift and support</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case example</td>
<td>The international Alpha Insurance company wanted to develop a new service</td>
<td>Beta Enterprises is an international electronics group that wanted to</td>
<td>Gamma Energy is a large energy provider with subsidiaries worldwide. A</td>
</tr>
<tr>
<td></td>
<td>for their new target group of young professionals. A project team from</td>
<td>evaluate the possibilities of smartphone applications for container ships</td>
<td>diverse project team applied an upfront DT approach to explore the potential of platforms in the utility sector. The outcome was a solution vision for a digital home improvement platform to advance lead generation. To ensure a human-centered mind-set throughout specification and development, a new role was instantiated to use selected DT tools to enhance the prototype and fill the backlog with new features. Produced DT artifacts included high-fidelity prototypes</td>
</tr>
<tr>
<td></td>
<td>five different business functions (marketing, IT, actuary, product manager, claims) with 40% capacity followed the DT process in an iterative manner for three months. The solution vision resulted in a tested medium-fidelity prototype for a digital on-demand insurance that could be activated and deactivated based on the user’s preferences. The DT team handed over</td>
<td>in a marine context. The main goal was to define requirements from a user point of view and foster creativity for solution finding. In a highly regulated environment, a DT infusion was chosen to support the ongoing RE activities with selected tools from needfinding and prototyping. Five DT infusion sessions (one–two days) were conducted within five months. Produced artifacts included field studies for precise</td>
<td></td>
</tr>
</tbody>
</table>
• **Define your content focus**: Depending on what a team needs to learn more about, each approach emphasizes a different content. If you need to understand the user and business context in detail, concentrate on DT artifacts; if you focus on the technical perspective and feasibility questions, concentrate on the RE artifacts at the requirements and system layers. Teams may jump back and forth between both approaches if new questions come up in one or the other area.

• **Find the right balance**: A balance needs to be found between creative and corporate requirements. Experienced practitioners are able to observe different demands and adapt the approach accordingly. The differences in DT and RE are therefore a great chance to customize projects along the way.

### Three Integration Strategies

We have identified three valuable strategies to customize projects based on different ways to integrate DT into RE:

1. **run DT prior to performing RE practices (known as upfront DT)**

2. **infuse the existing RE process with selected DT tools and artifacts (known as infused DT)**

3. **integrate DT into RE practices on an ongoing basis (known as continuous DT).**

Table 1 provides an overview of when and how each strategy can best be employed. To choose the right strategy, consider your objective and project context.

• **Use the upfront strategy** when you are facing a high level of uncertainty about the problem and the solution. Applying DT helps understand the problem deeply and define the overall concept of an idea.

• **Use targeted infusions** to support existing RE activities, for example, if you already have a solution idea but still need to sharpen it or if you want to clarify fuzzy requirements. Applying RE helps to focus on details or features of an idea.

• **Use continuous DT** when integrating DT principles into your RE routine, for example, as part of an organizational change program. Instantiate roles (human-centric requirements engineers) and use the DT principles as a long-term strategy.

We have also found the maturity level of DT in an organization as an influencing factor. While RE is typically a common practice, DT is still relatively new. Thus, the decision to integrate both approaches also depends on the level of courage, given time, and dedicated resources. As a rough guideline, the infusion strategy provides a reasonable starting point as it applies focused DT interventions within established practices. While the upfront strategy also keeps existing procedures, it requires more time and resources. Finally, the continuous strategy demands commitment by management to foster mind-set change in an organization or department.

### Open Challenges

We summarized three strategies to effectively connect DT and RE. Applying DT in an upfront manner or in a way in which it coexists with engineering activities is what we typically encounter in practice. Fully integrated and continuous DT, however, is needed to facilitate seamless
ABOUT THE AUTHORS

JENNIFER HEHN is a research associate pursuing her Ph.D. in business innovation at the Institute of Information Management, University of St. Gallen, Switzerland. She is also a senior manager at the innovation consultancy IT Management Partner St.Gallen AG (ITMP) with a focus on managing design thinking projects in a variety of industries. Her research interests are design thinking, requirements engineering, and agile development techniques. Hehn received an M.A. and a Dipl.-Kffr. in business and art history/ethnography. Contact her at jennifer.hehn@unisg.ch.

WALTER BRENNER is a full professor at the University of St. Gallen, Switzerland, and acts as managing director of the Institute of Information Management. Prior to joining academia, he worked as the head of Application Development with Alusuisse-Lonza AG. His research focuses on IT management, management of IT service providers, and innovation and technology management. Brenner received a Ph.D. Contact him at walter.brenner@unisg.ch.

DANIEL MENDEZ is an associate professor for software engineering at the Blekinge Institute of Technology, Karlskrona, Sweden, and senior scientist at fortiss, the research institute of the Free State of Bavaria, Germany, for software-intensive systems and services. His research focuses on empirical software engineering with a particular focus on interdisciplinary, qualitative research in requirements engineering and its quality improvement. Mendez received a Dr. habil. Contact him at daniel.mendez@bth.se.

FALK UEBERNICKEL is a professor at the Hasso-Plattner-Institute in Potsdam, Germany, and an adjunct professor at the University of St. Gallen, Switzerland, responsible for design thinking and business innovation. He has worked together with and consulted several international organizations in implementing design thinking. Uebenickel received a Ph.D. His research interests include human-centered design and design thinking for software-intensive systems and quantification of human-centered design. Contact him at falk.uebernickel@hpi.de.

MANFRED BROY is a full professor emeritus for software and systems engineering at the Technical University of Munich, Germany. His research interests include software and systems engineering comprising both theoretical and applied aspects including system models, specification and refinement of system components, specification techniques, development methods, and verification. Broy received a Dr. habil. Contact him at broy@in.tum.de.
transitions into engineering activities (and back). We showed one such integration at an artifact level, which raises us to the next level of challenges for research and practice. We cannot yet unfold a complete picture of how principles, work results, and methods found in DT and other software engineering practices (beyond RE) exactly relate to each other.

At a conceptual level, we still need to better understand two major aspects of DT: which principles can also be found in other more holistic human-centered software engineering disciplines and how they differ and what their boundary objects are.

These questions become apparent in our artifact model. What are the same or similar artifacts with which purposes? When are they interchangeable? The same holds when reflecting upon the methods used to create the artifacts. Which methods in DT can be used for other software engineering disciplines? How do these methods differ and how can they be integrated? How can milestones be effectively defined, for instance, as interfaces between different software process models? Finally, we also need to reflect on project roles. What seems trivial at first becomes challenging when considering competencies and responsibilities. How can, for instance, multidisciplinary DT teams be integrated with traditional roles such as those of a requirements engineer or a business analyst? How must existing responsibilities be modified when coexisting and collaborating?

Further questions arise when putting an integration into action at the project level. What are typical project situations that influence the choice of a strategy? How do these situations and the class of systems influence the choice of a strategy and/or single methods? How can these situations be characterized and assessed (with which confidence)? The latter is essential to build a holistic approach tailored to the needs of individual software project settings and, thus, ready for adoption in industry.

We have drawn from our experiences to discuss how DT can be used effectively for RE. Both approaches aim at discovering goals and requirements. While both DT and RE are very distinct when it comes to their underlying philosophies, many artifacts are complementary or even overlapping. Yet, while in RE the measurements of success are often the documented requirements as a foundation for development and quality assurance, in DT we follow a philosophy of domain understanding and the learning curve leading to it, regardless the surrounding processes. We showed different combination strategies depending on the project context before we laid out a road map for future research and practice. With this article, we hope to foster an important and overdue discussion, and we cordially invite researchers and practitioners to join this endeavor with their own ideas of integrating DT for a human-centered approach to RE.

**References**

Migrating a Software Factory to Design Thinking: Paying Attention to People and Mind-Sets

Nolwen Mahé, Design Thinking Montréal
Bram Adams, Polytechnique Montréal
Josianne Marsan and Mathieu Templier, Université Laval
Sylvie Bissonnette, Proaction Technologies/Design Thinking Montréal

// Design thinking (DT) has found its way into software engineering, promising better requirements elicitation, customer relations, and cohesion within the development team. We report on Proaction Technologies’ migration toward DT and evaluate the process through interviews with employees and clients. //

DESIGN THINKING (DT) is an approach “that uses the designer’s sensibility and methods to match people’s needs with what is technologically feasible and what a viable business strategy can convert into customer value and market opportunity” (Brown, p. 85). It combines processes, tools, and a mind-set that has roots in architecture and product design and is typically used to elicit and prove-in user requirements for the development of products or services. DT is adopted by many major software providers and promises to become a key success factor by enhancing the customer experience.

Since its adoption has been slow, research on the application of DT is scarce. The literature shows that implementing and using innovations can lead to unexpected issues, challenges, and outcomes. Hence, more studies of real-life experiences are needed to offer insights to guide organizations through implementing and using DT. We present an experience report from Proaction Technologies, a company that transformed its software-development factory to integrate DT. Apart from the DT approach, we provide insights from the trenches regarding the prerequisites for success, elements to consider during the process, and expected and unexpected impacts.

Use-Case Context
Proaction Technologies and its flagship product, UTrakk, are presented in Figure 1. The company’s DT journey began early in 2018, following the appointment of a new executive vice president who had DT experience. At the time, the company was experiencing important issues, such as the use of outdated waterfall-style processes, reduced employee motivation, and communication deficiencies between consultants and developers, particularly in requirement elicitation. The
main motivation for adopting DT was to improve the requirement-elicitation processes and help consultants and analysts identify the most relevant functionalities. Doing so would increase product desirability for the client and improve the company’s long-term business viability (better value-proposition models).

Another important motivation was to improve the relationship between clients and Proaction’s account managers and consultants. Since external stakeholders are closely involved in each stage of DT, they are likely to endorse a partnership role and increase their trust in the company’s products. Finally, since Proaction planned to expand (one-year growth from five to 30 developers), the executive management anticipated challenges regarding efficient collaboration and knowledge transfer and the potential for demotivation. DT was expected to foster team engagement, promote employees’ creativity, encourage empathic listening and strong collaboration, and lead to a “fail fast, learn fast” philosophy.

Approach to the DT Journey
Figures 2 and 3 capture the DT activities carried out by Proaction during a period of 14 months as well as the deliverables that each activity generated. At least one stage of the DT process of Fig. 2 is covered in each migration activity of Fig. 3. The initial cycles enabled Proaction to zoom in on a road map, value proposition, and business model for UTrakk. With relevant customers, subsequent cycles dug more deeply into each major functionality.

Insights From the Trenches
To understand the DT transformation at Proaction, its context and its outcomes, four employees and four clients were interviewed. The average length of each interview was 60 min. The interviews were coded in an open fashion to find empirical evidence of whether DT did what it was intended to do, identify the major impacts of adopting DT, and specify the prerequisites and other elements for successful adoption in traditional software factories. While she was not involved in the interviewing and coding, the last author’s input was informative, since she was the executive vice president who initiated the transition.

Our findings show that while DT lives up to its promises, its implementation requires discernment and caution. Table 1 summarizes the items that a traditional software factory should consider when migrating toward DT. For each item, we provide techniques and guidelines that proved useful at Proaction to avoid pitfalls and increase positive outcomes. Next, we present prerequisites for success, elements to sustain momentum and manage the change process, emerging challenges, and (un)expected impacts of the change.

P1: Obtain Management Buy-In
As for any change, strong management backing for DT is essential (clients 1 and 3), especially because the initial steps appear lengthy or ineffective and, later, because of fluctuations in the morale of the teams involved. This applies to upper management, which needs to convince...
its employees to trust the new way of working, and the client’s DT champions. Depending on the latter’s personality and DT experience, meetings can range from superinteractive to rather stiff, and their frequency can vary. A steering committee can help to increase awareness about this, potentially soliciting other people to volunteer as champions.

P2: Change Client and Development Team Mind-Set
Success requires a certain change in the clients’ mind-set (clients 1–3; employees 2 and 4). One client said, “So, I don’t buy a software product, I buy a way of working.” Instead of deciding all by themselves, clients must be open to validating ideas with other clients (do we really need/want this?) and the development team (is this feasible?). Success also requires a change in the development team’s mind-set (clients 3 and 4; employees 1 and 4). Foremost, team members must become aware and convinced of the need for cocreation, since value can be obtained only by understanding the needs of the client. The humility of listening to the clients and testing one’s understanding of a problem must be understood first, then practiced. Further, developers need to become convinced that, as client 3 said, “the final product will never exist as long as one continues reflecting and envisioning the future of the product,” which is a good example of DT’s stance that everything is tentative.

P3: Prepare Client and Development Team for the First DT Activities
Due to the unconventional collaborative activities of DT and the need to change mind-sets, one major challenge is the need to prepare clients ahead

---

### FIGURE 2. The nine stages of a typical DT process

1. **Understand**
   - Develop an initial contextual knowledge and an empathic understanding of users and the deeper human side of the problem to solve.\(^{12}\)

2. **Observe**
   - Watch, ask questions, and reflect on users’ behaviors and interactions as well as the places where they evolve.\(^{12}\)

3. **Point of View**
   - Look at existing solutions to see if they can be reused or combined to create a solution that will have an impact on users’ experiences.\(^{12,13}\)

4. **Ideate**
   - Brainstorm as many ideas as possible, without judgment, to create an innovative solution.\(^{12}\)

5. **Prototype**
   - Develop rapidly a rough sketch or model of the solution.\(^{12}\)

6. **Test**
   - Convey the idea of the solution to the users and get their feedback by testing the prototype with them. Based on this feedback, go back to the prototype, modify it if necessary, and retest it.\(^{12}\)

7. **Storytelling**
   - Develop a concrete, fully conceived plan of action that will motivate the intended audience to carry it forward.\(^{14}\)

8. **Pilot**
   - When the solution is almost ready for deployment, test it in a real-life context in the form of a pilot deployment.\(^{14}\)

9. **Business Model**
   - Adapt the business model and obtain approval to launch the solution.\(^{13}\)
FIGURE 3. Proaction’s approach to its DT migration: Insights from the trenches. The nine DT stages are explained in Figure 2. UI: user interface; UX: user experience; OM: ongoing management; EC: emerging challenges; P: prerequisite; I: impact.
Table 1. The recommended techniques and guidelines to manage prerequisites, meetings, and expectations.

<table>
<thead>
<tr>
<th>Techniques and guidelines</th>
<th>Prerequisites to develop and sustain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inform and communicate</td>
<td>P1: Obtain/ retain management buy-in</td>
</tr>
<tr>
<td>Involving people</td>
<td>• Provide management champion(s) with strong data on the business impacts of the method as well as content that can be used in internal communications.</td>
</tr>
<tr>
<td>Plan and organize</td>
<td>• Be transparent about upcoming events/activities, requirements for the clients' employees, expected morale variations at different steps, and probable needs for support from the champion.</td>
</tr>
<tr>
<td>Document and keep track</td>
<td>• Keep the champion(s) regularly informed, outlining their importance and role.</td>
</tr>
</tbody>
</table>

P1: Obtain/ retain management buy-in
- Provide management champion(s) with strong data on the business impacts of the method as well as content that can be used in internal communications.
- Be transparent about upcoming events/activities, requirements for the clients' employees, expected morale variations at different steps, and probable needs for support from the champion.
- Organize the champions as a steering committee, if suitable.

P2: Change/ preserve the client and team mind-set
- Be convinced of your message and show it.
- Demonstrate the value of the DT approach. Provide strong data on the scientific bases of the method (as suitable, depending on the team's profile), success stories similar to the problem at hand, and business impacts of the method.
- Adapt the mode of communication to the local culture and team profile.
- Make sure that the management champion(s) periodically reassure the clients' team about the acceptability of the process.
- Develop trust inside the teams (for example, by ensuring that they know each other personally). Enroll each person in supporting the team's attitude and morale.
- Train the internal team on the DT approach (for example, the process and frame of mind). Have the members practice with internal challenges (such as their own work environment) before exposing them to external clients.
- All through the project, train new internal and external team members. Use the training as an opportunity to reinforce the mind-set of the whole team.
- Provide easy and visible reminders of the expected attitudes, such as brochures and gadgets.
- Use positive-results tracking and outcomes to reinforce the mind-set of the teams. Point out the links between the attitudes displayed and the positive results achieved. Reflect positive changes (including the benefits to the internal functioning of the teams) back to the team members.

P3: Prepare clients and teams and keep them informed
- Be transparent about upcoming events/activities, time requirements, and expected morale variations at different steps.
- Be graphic, with pictures and videos of real DT workshops.
- Be transparent about the downsides of the DT approach, such as the effort and time required for the initial steps.
- Train the internal team on the DT approach (for example, the process and frame of mind). Have the members practice with internal challenges (for instance, their own work environment) before exposing them to external clients.
- Use DT workshops (for instance, persona building, journey maps, business-model development, and so forth) as involvement opportunities for internal and external teams.
- Build and share the complete road map for the product development to maintain vision and focus during the project execution.
- Put in place a ticketing system to facilitate the management and execution of tasks.

(Continued)
Table 1. The recommended techniques and guidelines to manage prerequisites, meetings, and expectations (cont.).

<table>
<thead>
<tr>
<th>Techniques and guidelines</th>
<th>Inform and communicate</th>
<th>Involve people</th>
<th>Plan and organize</th>
<th>Document and keep track</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Put in place a collaboration platform to facilitate communication and plan as a team.</td>
<td>• Engage the internal team in the field observation and shadowing of clients for each persona and the relevant processes.</td>
<td>• Take up a suite of groupware tools (for example, in the present use case, Aha for the road map, Jira for ticketing, Balsamiq sketching for design, Zeplin.io to share sketches and mockups, Azure DevOps for task- and source-code management, Smartsheet for program communication, UTrakk for meetings and the action plan, and so on).</td>
<td></td>
</tr>
<tr>
<td>During the course of action</td>
<td>• Add external facilitators or attendees to the meetings and workshops (especially the first ones) to formalize the DT method and model the expected behaviors.</td>
<td>• Select attendees so as to avoid hierarchy pressure. The level of pressure depends on the local business culture, and personalities and can be probed through local champions. If necessary, split meetings to reduce the number of hierarchy levels in each.</td>
<td>• Set up an archival method to keep track of all relevant information for the duration of the project and for potential future projects.</td>
<td>• Designate a note taker who will keep track of the verbal and written exchanges as well as the ambience and nonverbal interactions. The information produced during meetings is often scattered and tangled. The notes will encode ideas, insights, and decisions to make them retrievable in the future.</td>
</tr>
<tr>
<td>Emerging challenges</td>
<td>• Be clear during the activities about the end-to-end process, which usually includes further planning steps, such as prioritizing and feasibility assessments.</td>
<td>• Engage people in person during the production process (for instance, designate privileged points of contact, with periodic personal connections) to better assess changes in expectations.</td>
<td>• As the outcomes of the activities go through planning steps inside the company, communicate the new plans and road maps as soon as possible.</td>
<td>• Describe the archiving systems to reassure clients about the usefulness of all contributions, even if they are not used immediately.</td>
</tr>
</tbody>
</table>

OM: Organizing meeting and workshop logistics

EC: Managing clients’ expectations

OM: ongoing management; EC: emerging challenges.
of time for the activities planned for the first encounter, often a full-cycle workshop (clients 1–3). Sending the meeting agenda by email does not suffice, since, in the worst case, it might lead not only to indifference but also negative reactions. It is especially important to clearly explain to management what will be requested from its employees and even involve management in the organization of DT meetings through the steering committee.

The developers must be briefed adequately about the DT approach (employees 2 and 4), since they are technically very knowledgeable and, thus, even more likely to dismiss DT activities. Employee 2 recommended putting in extra effort during the DT bootstrap, perhaps involving outsiders in the initial meetings (making them more “official”). Younger team members (that is, millennials) have different notions of communication, and management must be open to them. Employee 4 said, “One just needs to reach out a hand, one has to pull them toward us, such that they adapt themselves.”

Sustain Positive Momentum Throughout the Process
It is important to sustain the momentum (clients 1 and 3) created at the outset of the change. While the client has invested in a process rather than a finished product, the end users and other client stakeholders use the product in progress to solve their own tasks. At some point, they might be happy with the product, have too limited a view to satisfy certain expectations. “The easiest things to put in place have been put in place … but we do not have the big elements that we have seen during the meeting,” while client 3 stated, “One has to address the expectations, or not. One just has to be clear. The upcoming release in the next months will satisfy certain expectations.” The time needed for meetings, follow-up, and changing the stakeholders’ mind-set further slows down velocity.

The interviewed clients and employees agreed that DT improved requirements elicitation in the annual road map and lower-level software modules (clients 1, 3, and 4). In particular, client 1 mentioned that, for the first time, there was a feeling of being heard and invested in the product’s evolution. Client 3 confirmed that “just that conversation, in which [Proaction] asked [us] what [we] actually needed, showing that [they] cared, that a new release was coming—that has created a nice vibe.” Client 4 said that not only did the product better fulfill expectations, it was delivered faster and more intuitive to use.

The key DT ingredient for greater client satisfaction is the focus on feedback-based decision making, from the client to the development team and within the development team (clients 1–3; employee 4). Clients installed their own mechanisms to solicit and manage feedback from their local user base in the form of prototypes (even just slideware), videoconferences, and a custom knowledge base of opportunities for product improvement. That amassed feedback could be shared with Proaction. Internally, DT sparked more feedback between employees, pushing them to identify inhibitors and opportunities for improvement in the product and development process.
### I2: Better Client Proximity

The clients and employees confirmed that DT led to a more intense interaction (clients 1 and 3; employees 2 and 3), as indicated by the feedback-based decision making. In contrast to traditional, agenda-driven meetings, DT conferences stress interaction between people and induce one to take a step back and think about one’s actual needs for a product. Especially powerful were the workshops in which multiple clients participated, since they enabled clients to put their own needs into perspective and even provide advice to each other. While successful client interactions require a change in mind-set, the threshold to participate is low, typically just an initial ice-breaking activity. Throughout the many events involving clients, a personal connection is built that benefits later activities, such as product support.

### ABOUT THE AUTHORS

**Nolwen Mahé** is a cofounder of Design Thinking Montréal, a collective of training and implementation professionals. Mahé received a B.S. in engineering from the Institut national des Sciences appliquées de Rennes, France, in 1984 and an M.B.A. from McGill University, Montreal, Canada, in 1998, as well as a Project Management Professional certification from the Project Management Institute. Contact her at nmahe@hotmail.com.

**Mathieu Templier** is an associate professor of information systems at Université Laval, Quebec City, Canada. Templier received a Ph.D. in administration, specialization information technology from HEC Montréal in 2015. He is a member of Centre de recherche en technologies de l’information et affaires and collaborator in the Software Ecosystem Health project. Contact him at mathieu.templier@sio.ulaval.ca.

**Bram Adams** is an associate professor of software engineering at Polytechnique Montréal, where he heads the Lab on Maintenance, Construction, and Intelligence of Software and is a principal investigator in the Software Ecosystem Health project. Adams received a Ph.D. in computer science engineering from Ghent University, Belgium, in 2008. Contact him at bram.adams@polymtl.ca.

**Sylvie Bissonnette** is the executive vice president of information technologies and chief information officer at Proaction Technologies, Montréal, and a cofounder of Design Thinking Montréal. Bissonnette received a B.Sc.A. in informatics from Université du Québec en Outaouais, Gatineau, Canada, in 1988. Contact her at SBissonnette@proactioninternational.com.

**Josianne Marsan** is a professor of information systems at Université Laval, Quebec City, Canada, where she heads the Centre de recherche en technologies de l’information et affaires (CeRTIA), and is a co-investigator in the Software Ecosystem Health project. Marsan received a Ph.D. in administration, specialization information technology from HEC Montréal in 2010. Contact her at josianne.marsan@sio.ulaval.ca.
I3: Happier Development Team
DT also led to a tighter-knit development team (client 3; employees 1–4). People talk more to each other, have fun together (even outside of work), and are proactive and motivated. The team works together in synergy without forgetting the personal growth of each member. As such, each member feels significant, can observe where the team is heading, and supports Proaction’ focus on improving product quality.

I4: Unexpected Benefits
The change in the client and employee mind-set has brought a number of unexpected benefits. First, a feeling of ownership/entitlement about the improved quality of the product. Employee 4 said, “It’s because I was there, I talked about my needs,” which, in turn, leads to bigger investments of a client company into the product, with larger roll-outs. Second, the DT way of collaboration inspires clients to spice up their other projects with activities such as mini- and role-playing games to engage meeting participants and deepen the understanding of others’ needs. Even the operators of the deployed products started to adopt DT ideas to streamline their work (client 1). Third, the members of the development team have a stronger belief in what they are doing, and, for example, are more willing to volunteer for tasks involving clients. Fourth, the acquired DT skills will likely become second nature to the members of the development team, readily applicable in other contexts; for example, focusing on value and refining questions to better understand the problem.

This article provided an experience report from Proaction Technologies’ migration toward DT, pointing out positive results, such as substantially increased client and development team satisfaction. During the process, a number of practical insights were identified that should be considered by other companies contemplating a migration to DT, especially the early tackling of changes in the mind-set of the clients and development team. Future work should shed light on the interaction between DT and other modern software-engineering practices, such as Agile and DevOps. Furthermore, a deeper analysis is needed of the correlation between DT and objective measures of software quality and cost effectiveness.

References
Three Phases of Transforming a Project-Based IT Company Into a Lean and Design-Led Digital Service Provider

Seppo Kuula, Enfo Oyj and University of Oulu

Harri Haapasalo, University of Oulu

Juha-Matti Kosonen, Silli Solutions Oyj

// Digital transformation requires a continuous review of value creation, value capture, and resourcing. In this article, we define a systematical service design concept to enable all stakeholders to achieve better outcomes in cocreation activities. //

THE BUSINESS LANDSCAPE in the digital era is service oriented, genuinely global, and in constant change. Companies are reimagining their business digitally, trying to radically improve their competitiveness while exploring the underlying changes in customer needs. Genuinely new, service-dominant business models are being created. While every company is going through a digital transformation, a prevailing misconception is that it relates only to technology. Digital transformation actually is more about how technology changes customer expectations and business processes, thus constantly alters the business environment.¹

Value creation provides legitimacy for the company’s existence and the basis for its business.² The focus of the commercialization of the offering should be on the ability to understand and support the customers’ value-creation process. The service deliverables (value creation) and customer needs (value capture) can be aligned by utilizing design thinking (DT) in a value-cocreational service-dominant logic (SDL) framework. This logic offers a sound foundational framework for understanding value cocreation and dynamic resource integration, underlining the collaborative nature of value creation.³⁻⁷ DT is a widely accepted human-centric management practice that advances design tools in business development.⁸ A combination of these two practices provides the systematic framework for continuous, systematic, and co-creative service design, accounting for technology, business, and human behavior.

The main aim of this article is to describe a systematic service-design approach for cross-functional knowledge-intensive business service (KIBS)
development in a constantly changing business environment wherein digitalization is driving companies toward service-dominant business models. This research falls under Design Science Research Methodology (DSRM). Our development work has followed the logic of constructive research, adapting the role of the researcher from Action Design Research (ADR), wherein the created framework is tested and further developed in a real KIBS environment over several years. Our case company provides information and communication technology (ICT) services for customers across all industries.

**SDL and DT**

The most significant difference between industrialization-driven goods-dominant logic and globalization-driven SDL can be seen in the definition of value ( cocreation) and the exchange (integration) of resources. In SDL, all actors, including the customer, use their available resources to cocreate value as integrators. This value is perceived by the customer on the basis of value-in-context. SDL implies that value is cocreated with the customer rather than embedded in output. The objective of the supplier in a cocreational relationship is to customize the offering by engaging the customer in the value-creation processes.

Competitive advantages are based on core competencies like knowledge, skills, and processes. In SDL, the value is iteratively cocreated with the customer and a competitive edge is created progressively through constantly improving the service experience.

Through multiple perspectives, service design should synthesize and creatively transform the collective knowledge through new service or product concepts. This approach, generally called DT, combines a deep end-user experience, systems thinking, iterative rapid prototyping, and multistakeholder feedback. DT focuses on gaining an understanding of an area of human experience and transforming this integrated knowledge into new solutions by accounting for the angles of value, creation, and capture.

DT is a human-centric management practice that takes advantage of the service-design practices in business development. DT is not a scientific theory but more of an applied mind-set and framework: be curious, try things, reframe problems, embrace the process, and collaborate. DT encourages learning by doing so in desired solution creation. The difference between conventional service design and DT is that DT not only explores the value-creation space but also explores value capture in the business model. Research ends in insight, creation ends in ideas, and delivery ends in reality. Compared with DT, agile development and lean start-up have some strong similarities, like user centrivity, iterative learning and extensive team communication.

The double-diamond model is another way to describe the DT process; it focuses more on the problems space in its description. The diamond shape aims to visualize the thinking modes, divergent and convergent thinking, along the design process (empathy, definition, ideation, prototyping, and testing). Divergent DT is used first for the first and third phases before reaching a convergent outcome. This approach prevents one of the most common mistakes: solving the wrong problem. Practical design methods like user diaries, journey mapping, and character profiles are used through all phases.

**Methodology**

DSRM requires creating an innovative, purposeful construct for a specific problem domain, which must be evaluated to ensure its utility for the specified problem in this research service design and value-cocreation framework. Accordingly, the DSRM results of the research must be presented effectively, to both technology-and management-oriented audiences.

Our development follows the logic of constructive research, adapting the role of the researcher from ADR. Building knowledge with the assistance of case studies is a research strategy that involves using one or more cases to create constructs, propositions, or midrange theory from case-based, empirical evidence. The constructive stepwise process...
has been adapted from Kasanen et al.\textsuperscript{20} The problem originated in the business-development process, when the company was transformed from resource provider into an agile digital solution provider. In this process, the offering was iteratively developed to respond to the increasing cloud-based shadow IT need in which the supplier creates a full stack (from data management through application to usability) solution to the customer’s specific business need. The service-design solution we studied was developed using the iterative ADR process over years and the validation was performed through real-life case studies.

The construction process had three main iterations to find the solid state of the cocreational service-design process. The first approach was based on the strategic-planning process (in 2014) as an answer to creative offering development alongside an iterative business development. The solution was called the Digihub, reflecting the nature of close collaboration between the different actors and disciplines during solution creation. The second iteration was seeking more formalized processes for managing the creativity and a repeatable solution for demonstrating the outcome design through prototyping. The third and final iteration (in 2016) combined business and service design and value creation together under a DT framework and cocreational delivery management. This solution is defined in detail as an outcome of the framework development, assuming general applicability in KIBS business-development processes. Finally, we have validated and evaluated the service-design and value-cocreation framework. The demonstration of the solution framework is done in three real-life cases, applying the ADR\textsuperscript{19} (Table 1): 1) problem formulation; empathize, define; 2) brainstorm, test, and prototype; and 3) formalization of the learning as typical for DSRM.

### Constructing the Solution

Our case organization, Siili, was founded in 2005 and provides data analytics and design and technology services to companies across industries. Originally, the company was a coding resource provider, but a pull-driven offering development process expanded its competence portfolio, first toward data management and then toward design services.

Siili’s service design usually led to digital service creation, in which the company had hundreds of UX-designers, coders, and data management specialists supporting solution development. The ratio of service designers to developers (including UX and data) was about 1:30. A technologically agnostic approach is important for a cocreative solution-development partnership, which addresses design-related challenges such as whether a digital service creator can avoid leading a customer toward digital solutions. In our study, two out of three cases found a solution in which digital development was not the solution.

### The First Cycle: The Digihub

This research started in the beginning of 2014 as a response to a strategic need to focus on ensuring delivered solutions truly created value for customers, simultaneously transitioning

---

Table 1. The demonstration of the solution framework is performed using three typical real-life business cases.

<table>
<thead>
<tr>
<th>Case</th>
<th>Empathize</th>
<th>Define</th>
<th>Brainstorm</th>
<th>Test</th>
<th>Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Service innovation for a defending market (pharmaceutical)</td>
<td>Deep contextual interviews (qualitative)</td>
<td>Better use of the existing information</td>
<td>Service blueprints</td>
<td>Conceptual solutions for doctors</td>
<td>New concept and business model, piloted in the hospital</td>
</tr>
<tr>
<td>2. Validating the new business wholesale model (insurance)</td>
<td>Contextual interviews (quantitative)</td>
<td>New structure for the existing product</td>
<td>Service value proposition canvas</td>
<td>Public pilot with real customers</td>
<td>Structured telephone interview</td>
</tr>
<tr>
<td>3. A service vision for a manufacturing company</td>
<td>Creating different customer profiles and personas; customer journey</td>
<td>Compelling digital services against new competitors</td>
<td>Brainwriting and crazy eights; methods</td>
<td>Service walk-through sessions</td>
<td>Semistructured follow-up interviews</td>
</tr>
</tbody>
</table>
the company’s service development from opportunity management (push) to account management (pull). Siili had identified the need to strengthen its design capabilities, especially expanding competences from usability to business and service designs. Business-design expertise was required to understand customers’ business processes, opportunities, and value capture while service-design expertise was required to facilitate the continuous problem-and-solution definition in the value proposition. The tasks of the named strategy project were to: 1) (continuously) identify the business problems and business processes with the customer; 2) eliminate waste and focus on value creative actions; and 3) control the process and validate value cocreation by measuring the key parameters.

Siili’s business-development organization assessed these tasks from two directions, business architecture (processes and information flow) and service design, and found its capabilities to understand customers’ business and processes very limited. In this case, it was possible to expand the enterprise architecture and business-process expertise over existing information-system architecture and information-management expertise, but there was no service-design competency in the organization. The company began to explore possible acquisition targets with modern service-design competency and formed an internal project for generating a model to develop the cocreation service-design process. The results of this iteration (executed between 2014 and summer 2015) and the end result of the first development round were called the Digihub (Figure 1). The following managerial decisions were based on this iteration.

1. The company acquired another company with a strong service-design background and a moderate business-design background at the end of 2014, due to lacking service-design competence.
2. Siili’s service-development organization analyzed and started to apply the lean start-up approach to defining an iterative and cocreational collaboration process.
3. In 2015, the company redefined its offerings to include service innovation as a bootstrap among the system architecture and it created a new process for customer collaboration and service management.

The Second Cycle: Designing Through Prototyping

The Digihub setup did not always meet the desired outcome because it was highly dependent on development resources. In particular, the customer’s business organization was not always committed to neither the cocreational approach nor the prioritization of continuous development. The second iteration originated in the H1/2015 strategy review and focused on defining the iterative problem and demonstrating the outcomes through prototyping. The idea was to either prove the value of the outcome in an early phase or to pivot the development before incurring significant costs. The firm’s technology expertise, together with user-experience design and prototyping practices, was strong but its business-design skills were limited. The lean start-up process-related rapid prototyping was seen as a solution to facilitate business development.

This approach was challenged again at the end of 2015, mainly because of an insecure business-design

---

**FIGURE 1.** The results of the first development round for a systematic service design approach for cross-functional KIBS. Digihub’s 2015 model of the cocreative service design process is shown.
The first iteration originated in 2011 and was executed between 2011 and summer 2012. The focus was to leverage a commercntual service-development process, exploring the problem space and the lean start-up circle, build–measure–learn, to define and pattern the solution. Prototypes were used for collecting real-world feedback and learning about the solution definition with continuous improvement. When the solution is proven to meet the expectations, the final solution is developed with agile methods through the minimum viable product (MVP). In the MVP phase, the solution and the development process must pass final acceptance from both value creational and value-capture angles. Within this process, our goal is to start from the customer’s strategy and service vision, aligning value-creation activities with early-stage experiences, evaluating alternative problem spaces, creating prototypes for iterative testing, evaluating ideas and concepts for solutions, and redefining the goal as a continuous process.

As described previously, building organizational competence and interventions were performed in a real business environment. The design process and research results were evaluated in confirmatory workshops in spring 2017. Based on this development, we constructed the generic cocreational- and DT-inspired framework for continuous and iterative service design needs shown in Figure 3.

The synthesis of managerial decisions after this cycle was as follows.

1. To systematize our offering, we needed to combine DT with
digital service cocreation, considering all actors.

2. We had to create structure for MVP to make our offer more effective.

3. To understand what real customer needs are, rather than the requirements, we must also understand a customer’s business model and service vision.

Evaluating the Solution
To display proof of the validity of the DT-inspired cocreational service design, the case company applied the processes for continuous and cocreative service design in several service deliveries. We have documented three real-life cases for the validation.

In the sense of a weak market test, all three cases proved to be valuable while customers implemented the prototype of service delivery (Table 1). At the end of the first case, the project team devised a new service strategy and concepts regarding how to pursue a competitive advantage by serving doctors in a more meaningful manner and they developed a new business concept that was later piloted, also with a private hospital, to further develop the concept. For the second case, we can say that the DT-driven concepting process was proven to be valid as the insurance product had a high sales rate from the launch and high customer engagement over an extended period. In the third case, the storyboard served as a tool that communicated the customer-driven vision of the company’s future services and was used as guidance for the service-development activities. The vision was also coupled with a mockup prototype of the digital service envisioned. The mockup prototype made the story more tangible and helped the stakeholders to get a better grasp of the vision.

Our research demystifies the knowledge-intensive business-service delivery by systematizing the delivery processes. Through the systematization, we strive to contribute to the search for business efficiency. Searching for customer satisfaction—without systematization, with any means possible—easily ends up in waste and variation in quality. We have created and tested, at least on a rough level, the cocreational service-design model described. (Naturally, we have a more detailed model for company purposes.) Features following all three original iterations, however noted in ex-post analysis, were quite easy to summarize as

- cocreation with the client in a design sprint format (agile cocreation and rapid prototyping)
- the supplier was able to use the learned customers’ business insight to expand its offering to other (digital) services (continuum) within all these cases
- making sure the correct items were developed, the business risk was better controlled (although doing things correctly is not enough to avoid reclamations)
- ensuring value in delivery involved positioning the supplier

**FIGURE 3.** The results of the third development round: service design and value cocreation. The case company’s processes for continuous and cocreative service design in 2017 are shown.
better, from relieving services to enabling services.

During validation, we discovered that one of the most important pinchpoints, in an organizational sense, is the learning itself. The double diamond certainly is a process platform in DT, which is the back end for a continuous, deliverable definition process.

The case company’s solution-delivery capability iteratively increased over the time we studied the phenomenon. Transitioning the offering transformed the entire company, from customer collaboration to delivery processes. Siili found success in its business through the transformation, which brought additional pressure to development, together with continuous growth. The transformation had to be rooted in culture, so it required a long time period. Three major iterations were required before the approach reached a repeatable form. DT and cocreation were seamlessly integrated into the case company’s service-delivery process, providing more efficiency and effectiveness, as well as the expected continuous deliverable definition. In the nature of DT, the offering and delivery development work has continued at Siili, after these three rounds described in this research were completed.

In service design, it covers the iterative process through several phases, from inspiration, through ideation, reflection, prototyping, and testing, and finally to implementation. However, business management should aim for the repetition and systematization of services; even customers require customization. A recent study on portfolio management for service and product offerings[21,22] led us on an avenue of developing KIBS deliveries to clarify the value creation and increase the cost efficiency, and thus the profitability. With this point of origin, we have created a systematic service-design approach for cross-functional service development. We have utilized three iteration cycles with a constructive approach to develop the model. As part of the constructive process, we also tested the solution in real-life business cases to validate our approach.

Based on our study, this DT-inspired cocreational solution-development approach can serve as a foundation to develop services. The prototype offers an understandable framework for the stakeholders involved, providing clarity and common understanding and language through the process. With the model’s help, the stakeholders can operate a smart problem-solving process and thus achieve better outcomes in their cocreation activities. The model also reduces the resource risks that are included in development projects, such as building a solution that has not been validated with the end users.

The double diamond prototype roots the activities at a level that can be utilized as a managerial process model for communicating service development and as a customer cocreation process to define the commercial deliverables. In our study, the model worked in the case of KIBSs; however, more lessons from different types of services are required. Of course, the detailed level descriptions differ with various types of services. Therefore, the organization-specific learning process of service design offers significant learning and development opportunities, which is a research avenue for our future studies.

Acknowledgment
We would like to thank Siili Solutions Oyj for acting as a case company in our study and for providing all of the necessary materials. The corresponding author of this article is Prof. Harri Haapasalo.

References
9. K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A design science research methodology for information systems research,”
FOCUS: DESIGN THINKING

SEppo Kuula is the chief executive officer at Enfo Group and a researcher with the Department of Industrial Engineering and Management, University of Oulu, Finland. Kuula received an M.Sc. in industrial engineering and management and an eMBA in service dominant logic in professional services from Oulu University. He was nominated as the Finnish Software Entrepreneur of the Year in 2014 for entrepreneurial leadership. Contact him at seppo.kuula@enfogroup.com.

Harri Haapasalo is a professor with the Department of Industrial Engineering and Management, University of Oulu, Finland. His research interests include business management, product management, and management of production processes. Haapasalo received a doctorate in technology management from the University of Oulu. He has more than 300 international scientific publications. He has been a chair and committee member for numerous international conferences as well as a reviewer and on the editorial board for several international scientific journals. Contact him at harri.haapasalo@oulu.fi.

Juha-Matti Kosonen is the director of design with Siili Solutions Oyj. His research interests include design theory and academia. Kosonen received an M.S. from Aalto University’s International Design Business Management program. Contact him at juha-matti.kosonen@siili.com.

Design Thinking in Industry

Talita Rozante de Paula, Tatiane Santana Amancio, and Jose Adan Nonato Flores

**From the Editor**
To shed light on business experiences in design thinking (DT), we have asked IBM for observations from real projects, as IBM has been practicing DT for many years in various projects, regions, and domains. Obviously, some projects are more successful than others. Talita Rozante de Paula and Tatiane Santana Amancio bring their experiences from a large, successful DT application. Counterpoint author Jose Adan Nonato Flores highlights that there are also risks, as there are with any method. Obviously, this is not an IBM corporate view but rather snapshots of project experiences. —Christof Ebert

---

**POINT**

Talita Rozante de Paula and Tatiane Santana Amancio

**CREATING THE RIGHT product** is a consistent topic of debate for competent product management. The issue is not always about creating something new as per the set-out specifications but rather whether the result is exactly aligned with customers’ needs. A product can only be useful and considered a success if it is a necessity or exceeds expectations. The missing element is the customer, or rather what the customer exactly wants.

In product engineering, multiple dedicated teams are usually engaged in the various development phases, beginning with eliciting and gathering requirements, defining the problem and solution, designing the result, implementing the design, and then testing. Normally, these teams interact only among their circle and seldom with customers. This is how traditional project and product management works and it has been able to hold its ground for some time. As the market evolves in terms of speed and volatility, the needs of the customer are equally progressive in nature. The market demands product turnover at enormous speed with scope for scalability.

This is where empathy, one of the foundations of design thinking (DT), comes into play. In the simplest terms, *empathy* is the ability to relate to others. DT is a new fad that has gained momentum. Fortunately, it is well aligned to help with understanding the customers’ needs and creating innovative products as an outcome. It is a mind-set rather than a process at its core, with the intention to offset the transitional customer interaction model. With DT, customer involvement is enforced as mandatory, as opposed to restrictive. It is in this area that we see the chances of the product failing to make its mark with the target customer.

It is necessary to talk and listen to customers to collect their requirements, but often that is all one does: ask the required number of questions, collect the appropriate details, collate the data in a predefined template, and proceed to the next step of creation. This is where the problem begins and is how empathy fits into the whole picture step by step.

First, we need not only a predefined questionnaire of what is perceived as required but also incorporate why it is required as well as
its intended purpose. In other words, the customers’ pain point is the key to identifying what consumers need. Next, we must understand that this could be just the customer’s perspective; we need to drill further down the rabbit hole, asking sharper questions to discover more options to try to get to the exact root. Knowing the actual problem enables the team to identify appropriate options and be able to implement the most suitable ones. Throughout this entire process, clients must be beside the team every step of the way, so they are able to see the idea come to fruition while they relate to it. This is what guarantees that the solution follows the customers’ requirements to the end. With this DT mentality, empathy is incorporated into software product engineering and we would see better user engagement as well as improved and more meaningful products come to market.

Recently, an IBM executive challenged us to conduct DT sessions “optimally” without impacting project operation, involving key stakeholders and executives. Challenge accepted. Let’s do a DT light.

We designed an optimized journey. The prework involves a high-level supplies, inputs, process, outputs, and customers (SIPOC) process created in a 2-h remote session, five 4-h sessions involving 30 people distributed across five days for four weeks, an additional 8-h session with 30 stakeholders, and playback. The key to making the sessions so short with no quality loss is choosing the right exercises—sailboat, SIPOC review, as-is, fishbone, ideation and prioritization grid—and involving the key stakeholders, who bring different points of view.

Yes, it was very busy, especially in the first sessions. That’s when we realized that we could further optimize time and quality by performing a tailored version of as-is, fishbone, and ideation together to enhance the materialization of the rich discussions that emerged in the as-is review and turn them into input for the fishbone method and ideation.

Empathy was practiced in all the sessions. Representatives from the technical and management teams were involved in all exercises, from problem analysis to the ideation and prioritization grid. This optimization was fundamental for meeting the efficiency and effectiveness required of the challenge. What does it mean? There were 100 working hours, more than 60 people involved, more than 1,000 Post It notes applied and transformed in 30 automation ideas, more than 20 operational changes, and many other actions in infrastructure, management, and skills. Additionally, integrating the team with the session was greatly improved by discovering or better understanding each group’s challenges and impacts, as well as how groups could support one another.

Waiting was not required for the action plan to be defined in a playback session. The team was so engaged that they started some ideas even before the playback and used them not only to share the ideas that came up with other teams and executives, but also to execute the results. The work is currently in backlog prioritization with executives, which will be executed using the Lean2Agile approach.

The satisfaction survey that was executed amazingly showed 99% of promoters—the only complaint was about the available physical space. The experience was replicated with other projects and clients, with the same effectiveness and good outcomes.

Invest money to get a good return: monetarily, satisfaction, or any other parameter of an investor’s interest! DT can generate great value, but how much does it cost when you factor in preparation, facilitators, material, information digitalization, and consolidation into an action plan, as well as stakeholders not working in the project operation? We must ensure the improvement actions will come out of the flipcharts and become benefits.

DT can certainly add value to the product, project, people, and organizations, but undoubtedly requires a realistic look at its costs for full optimization. Let’s have the experience of a DT light.

Response of Jose Adan Nonato Flores

Flexibility and multiple techniques are part of the DT methodology. Doing prework well, investigating the stakeholders thoroughly, taking the time to already know their story, and choosing appropriate innovation techniques even when you have a very high promoter score (i.e., good feedback from surveys) are the actual challenges here. Real success will be achieved when the ideas materialize, and the benefits can be measured.

Counterpoint

Jose Adan Nonato Flores

DT IS A methodology, but it will not work for all circumstances. Let me show some risks that we observed in projects with DT.

My first experience with DT was during a training workshop. One of the first sentences that I can remember related to focusing on the user experience. This is great to keep in mind when working on a solution for a particular client. Knowing how the end
The user will feel using the proposed solution definitively brings better results. We have applied this DT method and it really works, but we have also faced challenges in applying it. DT workshops often have two problems. The first is that even when the methodology encourages creating new ideas, the participants usually use the outdated ideas to solve the identified issue, so the session results more in expressing old known ideas instead of creating new ones. This is not necessarily bad, but the heart of the methodology is promoting creativity and trying to find solutions that they didn’t consider in the past and are quick wins for everyone involved. This usually doesn’t happen systematically in industrial DT application. So creating new ideas will not automatically happen.

Another risk that we often perceive is that, after the DT workshop is completed, far too many prioritized ideas from the different activities in the workshops result. This long list is then presented to the client and some of the ideas are put into a plan. Many ideas do not pass this filter since the persons involved in the decision making are not willing to make changes due to budget or time constraints or technical difficulties. Although many ideas do not pass this second filter, some usually survive this stage and are executed, benefitting the people involved and mainly the client, who is the most important actor.

Commitment is necessary. I think that by proceeding step by step and involving more stakeholders in an iterative way, the team needs to be in touch with DT, identifying the possible benefits and weaknesses according to the scenario and adapting it to get the best results. One good experience, even with a short scope, will facilitate future implementations.

Our real-world experience with DT is that while it promotes creativity, the challenge is how to integrate it when implementing the fresh ideas together with the known ones. To resolve this break between idea creation and regular product lifecycle and project work, the phases after the workshop are indispensable to successfully implement the ideas. Otherwise, the DT effort can result in a waste of time and the best ideas could be discarded and discourage creativity.

**Response of Talita Rozante de Paula and Tatiane Santana Amancio**

DT as a method is flexible and should be adapted to software product and organization needs. However, the time available to execute DT workshops is typically rather restricted, so the DT flow cannot be done as would be necessary. This dynamic largely contributes to a turbulent startup. Without prior analysis of what outcome is expected, false expectations or frustrations with the outcome often occur. Our experience has shown us that it is essential to have a prework analysis and to define a clear problem statement or business challenge, which techniques can best support the product being developed, which stakeholders need to be involved, and most importantly to get the commitment of the team involved.

**TALITA ROZANTE de PAULA** is the continuous improvement leader for the IBM Latin America Client Services Excellence Team. Contact her at talitarp@br.ibm.com.

**TATIANE SANTANA AMANCIO** is a computer scientist and transformation and innovation consultant with IBM Latin America in Brazil. Contact her at tsantana@br.ibm.com.

**JOSE ADAN NONATO FLORES** is the enterprise custom automation leader for IBM Latin America. Contact him at Jose.Adan.Nonato@ibm.com.

**ABOUT THE AUTHORS**

Talita Rozante de Paula is the continuous improvement leader for the IBM Latin America Client Services Excellence Team. Contact her at talitarp@br.ibm.com.

Tatiane Santana Amancio is a computer scientist and transformation and innovation consultant with IBM Latin America in Brazil. Contact her at tsantana@br.ibm.com.

Jose Adan Nonato Flores is the enterprise custom automation leader for IBM Latin America. Contact him at Jose.Adan.Nonato@ibm.com.
Emerging Perspectives of Application Programming Interface Strategy

A Framework to Respond to Business Concerns

Juho Lindman, University of Gothenburg
Jennifer Horkoff, Chalmers University of Technology and University of Gothenburg
Imed Hammouda, Mediterranean Institute of Technology, South Mediterranean University
Eric Knauss, Chalmers University of Technology and University of Gothenburg

// Software specialists increasingly find themselves in situations where their application programming interface (API)–related decisions have implications on software business. We present a strategic API framework to aid in consideration of business concerns when designing, updating, or maintaining APIs. //</p>
APIs are considered more of an artifact that must be continuously enhanced, while further research has considered API usability.

Several reports from the industry offer useful practical design considerations for APIs, including advice on collecting usage data, monetization strategies, and at what point to open an API to external parties. Although this advice can be useful, the focus is still often on the API but without considering the role the API plays in the wider organization or how it fits into an organizational strategy. Our industrial experiences show that more information is needed about the challenges and best practices of API design and management in an organizational context. Furthermore, existing practical considerations do not make use of the wider knowledge of established software business analysis frameworks.

Our research work was carried out within an industry–academia collaboration of the Software Center (see “The Software Center”). We show how conceptual frameworks can be used to drive API strategy development, drawing on our research work in organizations with several companies in practice (see “Methodology and Case Companies”). The approaches are seen as complementary and can be applied separately depending on company needs.

API and Digital Innovation

We draw on earlier knowledge of digital innovation and approach APIs as digital innovation objects, specifically as one layer within a larger context, allowing us to account for the role of APIs in the broader business strategy. Figure 1 presents APIs in relation to other relevant layers. We also note the boundary objects between the layers that will impact an organization’s strategy for its APIs.

Background: APIs as Digital Innovation Objects

Digital objects (such as APIs) are interactive and editable due to their implicitly multilayered architecture. To benefit from this layered approach when reasoning about API strategies, four relevant layers, similar to the layers from Yoo et al., have emerged from our empirical

---

**THE SOFTWARE CENTER**

The Software Center (http://www.software-center.se) is an industry–academia collaboration aimed at increasing the competitiveness of Nordic Software Engineering hosted by the Department of Computer Science and Engineering, Chalmers University of Technology and the University of Gothenburg, Sweden. Several other universities are also part of the center, which works primarily with a selected set of partner companies. The work in the Software Center is organized into dedicated projects where work is carried out with the industrial partners in half-year sprints. The results of the projects address the immediate concerns of these companies. The Strategic Application Programming Interface (API) project focuses on APIs as key enablers for a business strategy, generating new value for businesses and customers. Keeping up with the ever-changing market requires a well-defined API strategy in terms of identified processes, methods, and instruments to manage the API value chain.
FEATURE: API MANAGEMENT

METHODOLOGY AND CASE COMPANIES

The research employed a methodology including multiple case studies. The case data were collected from workshops, interviews, and thematic discussions with experts from four companies in the embedded systems industry. We worked with the companies to develop frameworks for their immediate needs regarding the management of application programming interfaces (APIs) and then learned through observing their application in practice. We first collected perceptions from the key stakeholders and then combined these insights with knowledge gained via literature reviews, providing and receiving feedback to and from our partners concerning their specific challenges and experience with various versions of the framework over several workshops. More detail on our methodology can be found elsewhere.12,13

Company one is a global firm operating in the area of network video cameras, currently providing network video products that are installed in public spaces (e.g., train stations and universities) and business areas (e.g., casinos and retail stores). The company creates value by providing a cloud API to enable third-party developers to create their own apps using camera data.

Company two is an international company developing various mechatronic devices for its global customers around the world. The company is in the process of implementing a new API workflow, which is expected to solve the bottlenecks in its current business process. The company wants to be able to analyze the effects these changes have on its API ecosystem.

Company three offers services, software, and infrastructure for telecommunication and networking equipment. The study was conducted in cooperation with a unit that was working on developing strategies for specific software frameworks related to state, fault, and alarm handling.

Company four provides a range of processing and packaging technologies. It supplies both services and products to support manufacturing tasks. We are focusing on accessing machine control software via an API.

FIGURE 1. The APIs as digital innovation objects with relevant layers (rows) and boundary objects (dots between rows). BAI: business asset interface.

METHODOLOGY AND CASE COMPANIES

Relevant Layers for API Strategies

We explain each layer, illustrated by our running example, and provide typical analysis questions associated with each layer, as found from our practical experiences. We illustrate the layers with a running example from company one focusing on access to camera data through cloud applications.

Domain Layer
The domain layer motivates the high-level need for the API ecosystem. It captures the roles that own the use cases motivating API usage software, the API itself, and the use of business assets. In our camera example, these are the end customers who want to use camera data for analytical purposes (e.g., crowd control or face recognition). Typical questions with respect to this layer include the following: Who are the end users? What business models/cases exist? What domain events can be anticipated (e.g., an area at maximum capacity)?

API Usage Layer
Situated on top of an API, software applications use the API to realize user-visible features. An example is an application that uses camera data to count individuals. Because the app software interacts directly with one or several APIs, this layer raises concerns that must be addressed in
an API strategy. Typical questions with respect to this layer include: Is the app software developed by internal or external developers? Is the app software usually slim or thick (e.g., is the app based on simple calls of a use case-centric API, which it exposes to end users, or does it have significant logic)? Among others, these aspects impact how users of the API react to updates.

**API Layer**

An API allows access to one or more business assets. In the running example, a cloud API provides raw camera data to usage apps. The main concerns on this layer are decisions with respect to API design. Typical questions include: Should the API respond synchronously or asynchronously to requests? To what extent should the API provide protection from unauthorized access, use, disclosure, disruption, modification, or destruction?

**Business Asset Layer**

The business asset layer encompasses the business assets of a company, such as properties of a product, core algorithms, and data. In our example, this is a network-attached security camera and the associated video data it may produce, including the metadata that can be extracted from the videos (e.g., crowd activity). With respect to API strategy concerns, the business asset layer raises the following typical questions: Are new business assets emerging? Can existing ones be exposed better? Are assets nearing the end of their lifecycle?

A layered view of our example is summarized in the second column of Figure 2. Examples for each layer for the other three companies, with details anonymized, are provided for further illustration.

**Boundary Objects Between Layers**

We have found it useful to further identify boundary objects between the API layers. If a boundary object changes, both adjacent layers are affected. With respect to API strategies and challenges, we identified the general characteristics of such boundary objects. API-related boundary objects are visible in the organizations when making decisions because an API strategy can be based only on boundary objects that are known. Many problems in managing APIs can be avoided if one party does not unilaterally change a boundary object without prior coordination with other parties. Here we provide examples of three common boundary objects relevant for API strategies (Figure 1).

**Use Cases**

A use case describes how actors can reach their (domain) goals based on available features (e.g., to identify individuals and count them across several cameras distributed over a building). An API strategy concerns raising questions, such as the following: Which use cases exist? How could use cases change (triggered by domain or app software)? What is the impact of the change?

<table>
<thead>
<tr>
<th>Layer</th>
<th>Company One</th>
<th>Company Two</th>
<th>Company Three</th>
<th>Company Four</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>End User, Anyone Using Enhanced Camera Data</td>
<td>Device End Users</td>
<td>Telecommunication Providers</td>
<td>Infrastructure Service Staff</td>
</tr>
<tr>
<td>Use Case</td>
<td>Cloud Services Apps</td>
<td>Internal and External Product Developers, Mobile App</td>
<td>New and Revised Product Features</td>
<td>Infrastructure Service Tools and Applications</td>
</tr>
<tr>
<td>API Usage</td>
<td>Product Profile</td>
<td>Development Framework</td>
<td>Machine API</td>
<td></td>
</tr>
<tr>
<td>API Specification</td>
<td>Cloud API</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BAI Model</td>
<td>Camera, Camera Data, and Metadata</td>
<td>Product Implementation, Code</td>
<td>Design Rules, Fault Handling</td>
<td>Machine Functions</td>
</tr>
</tbody>
</table>

**FIGURE 2.** A multilayer API view applied to our partner company examples.
API Specification
API specification is a contract between the app developer and API provider. In our example, this might be an established way to access raw data for data analysis. With respect to an API strategy, an important question concerns the expected impact of or resistance to change.

BAI Model
Contrary to the API specification, which tells what the API does, the BAI model describes relevant aspects of business assets to be exposed. In our camera example, these aspects include the ability to tilt, zoom, and access a raw data stream, but it probably does not influence the color of the camera’s casing. Questions that relate to API strategy concerns could consist of the following: Are all important aspects included? Is the model effective (i.e., does it allow for developing app software that addresses important use cases)? Is the model efficient (i.e., does it allow for addressing relevant use cases in a way that uses reasonable amounts of resources)?

Viewing APIs Through BAPO Concerns
All of our company respondents have emphasized that API development should not be an accidental or ad hoc activity but rather a systematic process supported by an integrated environment of policies, methods, tools, resources, and so on. America et al. argue that for a development method to achieve the best possible fit, four interdependent software development concerns should be considered: business, architecture, process, and organization (often referred to as the BAPO model). In our experience, this model helps to consider a wide range of strategic concerns in API strategy design, acting like a checklist and identifying areas where issues occur or a strategy may be missing. We find that BAPO is complementary to the four-layered architecture presented earlier.

Background: APIs From a BAPO Perspective
From the lens of BAPO, one could identify the following kinds of challenges related to API design and development. We illustrate the discussion with example questions raised by our industrial partners.

Business
This perspective addresses the concern of making APIs a business capability and a business development model. It bears similarity to the domain layer in the “Relevant Layers for API Strategies” section but focuses more on the business strategy of the organization, as opposed to users and use cases. Typical API-related business perspective questions include: How does API design drive business-level decisions, such as vendor selection, and how does one generate business value out of API usage? For our camera example, company one must determine the business value of enriched camera data, including which third-party developers to include in its API ecosystem.

Architecture
The main concern of this perspective is to investigate the technical issues associated with API design and development, similar to the API layer presented, but from a broader perspective (architecture of the interacting systems as opposed to API design). Questions to ask include: How does one manage API versioning (e.g., side-by-side deployment of different versions)? How does one design APIs for extension? How does one check the backward compatibility of APIs between different versions? In the camera example, one can consider whether different third-party developers will have access to differing API versions and how to encourage the adoption of new updates. The latter is an issue echoed by company three.

Process
This perspective raises concerns related to identifying roles, responsibilities, and relationships to respond and act more effectively within the API team. Typical questions are as follows: How does one update an API? How does one inform about and release new versions? How does one retire an API? For our camera example, company one must determine the process for updating the cloud API, including internal steps,
potential bottlenecks, and communication processes with third-party developers.

**Organization**

This perspective covers concerns related to the organizational aspects of the API strategy (i.e., mapping the identified roles and responsibilities to existing organizational structures). Usual questions include the following: How does one achieve better alignment between interacting organizational architecture units that may use/create/update the API? How does one build an effective API team? Who controls and governs change? In the camera case, the organization must determine who within the organization is responsible for API development (or which components of API development), including what roles will be required to govern changes.

**APIs and Governance**

Governance issues, including access control, openness, and resource management, are increasingly important in the API strategy, as echoed by our company partners. Analyzing APIs in terms of strategic layers raises several questions related to such governance issues. Different issues arise per layer (business assets, APIs, app usage, and domain). Companies make decisions about which kinds of governance arrangements support their business goals at each layer.

**Background: Common Pool Resources**

The work of Ostrom et al.\(^{15}\) on common pool resources provides a rich source of theories related to the issues of collective action framed in terms of governance, openness, ownership, and economic subtractability (rivalry) of the different social situations where governance is needed.

We have found that this theory provides a useful framework to discuss and classify API-related governance issues, particularly as our API-protected business asset can be considered as such a resource (a good), although not necessarily common, unless by design.

In general, according to Ostrom, there are two main criteria of collective action problems: one related to exclusion and one related to subtractability.\(^9\) Exclusion is determined by how easy it is to prevent access to the goods (e.g., it is difficult to block a sunset but easy to block entry to a company PC). Subtractability means how detrimental the uncooperative actions of the participants are to other participants (e.g., it is difficult to run out of sunsets, but one may run out of company PCs).

Along those two dimensions, Ostrom identifies four types of goods: private, club, common pool, and public goods. Private goods, such as doughnuts and PCs are both excludable (other individuals are excluded from consuming) and rivalrous (whatever is consumed, no one else can consume). Club goods, such as daycare centers and country clubs, are excludable (goods are accessed by club members only) but nonrivalrous (goods are available to all club members).

Common-pool resources are characterized by both difficulty of exclusion and difficulty of subtraction. They are also threatened by overuse and depletion. Finally, public goods, such as sunsets and common knowledge, are characterized by the difficulty of exclusion (accessible to all) and low rivalry (consumption of a good does not limit the consumption by others).

**API Layers as Strategic Resources**

We can use Ostrom’s theories to systematically consider governance and openness issues from the perspective of the various API layers. The independence of the layers means that a providing company does not have to select one access model for all layers (e.g., one layer can be private, while others are more open). We use our running company one camera example for illustration. Figure 3

<table>
<thead>
<tr>
<th>Level</th>
<th>Layer</th>
<th>Private</th>
<th>Club</th>
<th>Common Pool</th>
<th>Public</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Product, System, Services Embedded in Domain</td>
<td>Enhanced Data</td>
<td>Enhanced Data</td>
<td>Bandwidth Consumption</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>API Usage</td>
<td>Enhanced Data Applications</td>
<td>Bandwidth Consumption</td>
<td>Open Demonstration Apps</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>API</td>
<td>Screened Third-Party Developers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Business Asset</td>
<td>Camera Data and Metadata</td>
<td>Camera Data and Metadata</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 3.** The governance and access concerns per layer for company one.
FEATURE: API MANAGEMENT

provides a visual summary, mapping to our API layers.

In terms of the business asset layer, the company wants to allow camera data and metadata to be kept private. In individual camera system installations, owners will decide if their data will be available to cloud apps via the cloud API and possibly to which specific applications.

Access to selected parts of the asset is exclusively granted through API calls.

At the API layer, company one is interested in a club development community, where only approved third-party developers (or themselves) would have access to their cloud API.

At the API usage layer, third-party developers will make their functionality available for a price (club) and may face common pool concerns regarding bandwidth consumption for camera apps, particularly if the apps use live streaming. In addition, company one may create and release its own open source cloud apps to train and attract additional third-party developers.

In terms of the domain layer, application users may also face issues related to bandwidth consumption. End users of the enhanced camera data are unlikely to provide public resources but may either use the data for their own internal purposes (private) or may sell them in an exclusive way (club).

In this article, we present a multidimensional conceptual framework that has emerged from collaborative research on API strategies with several large companies. We have found that this framework helps managers, designers, and developers to discuss API management.

1) By understanding APIs as objects of digital innovation, our framework allows us to collect basic information about the relevant layers and boundary objects.

2) This information can be further processed in the BAPO part of our framework, where
business, architectural, process, and organizational concerns are elaborated. To manage any changes to boundary objects, the layers offer insights to relevant stakeholders.

3) To define development aspects, information about intended governance of the APIs is crucial. Such considerations are provided by the governance dimension of our framework.

In practice, a company might need to apply the conceptual framework differently depending on forces, such as whether the company has one or more APIs or whether part of the API is strategic while other parts are more opportunistic and temporary. Regardless, our framework filled an important gap within our case companies by providing them with a vocabulary to express strategic concerns as well as with a set of important perspectives to consider their strategy with respect to, for example, architectural or development aspects.

We continue to apply and improve the framework with our partner companies in the Software Center, furthering our understanding of which existing and planned API elements fall within each layer and revealing the gaps in practice. More detail on specific dimensions is described in further work.12,13 We continue to work on the complementary integration of the various framework components and their empirical interplay, including guidelines on when to apply a particular perspective. Further, we argue that the perspectives we have discussed in our model are applicable to other kinds of software development activities (such as quality assurance) that may require extensive analysis and design in an ecosystem context. The idea of applying our perspectives to other software development contexts is another direction for future work.

References
The Online Controlled Experiment Lifecycle

Aleksander Fabijan, Malmö University
Pavel Dmitriev, Outreach
Helena Holmström Olsson, Malmö University
Jan Bosch, Chalmers University of Technology

// Unlike other techniques for learning from customers, online controlled experiments (OCEs) establish an accurate and causal relationship between a change and the impact observed. We show that OCEs help optimize infrastructure needs and aid in project planning and measuring team efforts. We conclude that product development should fully integrate the experiment lifecycle to benefit from the OCEs. //

Although techniques for gathering customer feedback have traditionally focused on the collection of what the customers say they do (for example, interviews and focus groups), the software industry is evolving toward complementing these methods by accumulating data that reveal what customers actually do. This enables software companies to be more accurate in evaluating their ideas and to move away from assumptions and toward trustworthy, data-driven decision making. The Internet connectivity of software products provides an unprecedented opportunity to evaluate hypotheses in near real time. Consequently, the number of hypotheses that product management generates and aims to evaluate can be tremendous. For example, there are billions of possibilities for merely styling a product (for example, the “41 Shades of Blue” online controlled experiment (OCE) at Google). In fact, style and content management is only one area for experimentation. Evaluating improvements to ranking algorithms and recommender systems are popular applications of online experimentation, among others. Additionally, although there are several ways to evaluate hypotheses (for example, pattern detection, classification, and so forth), none of them show such a direct causal and accurate impact of an idea on customer value as OCEs. Our objective is to provide an understanding of the key phases of OCEs to help more companies benefit from this methodology.

Background

In an OCE, users of a software product are randomly distributed between several variants (for example, different designs of a product interface) in a persistent manner. The existing variant without the change is
labeled as the control and the other variants as treatments. The goal of this setup is to accurately evaluate ideas about software products (such as whether changing the checkout button color from gray to blue increases the number of purchases). While the user interacts with the product, activity signals (including clicks, dwell times, and so forth) are instrumented and key metrics (engagement, user satisfaction, and so on) are computed. \(^3,^9\) If the difference between the metric values for treatment and control is statistically significant, the introduced change very likely caused the observed effect on the metrics. With this, OCEs provide an accurate understanding of how much customer value the change in the products actually delivers.

**The Impact of Experimentation**

Incremental product improvements are not the only scenario in which OCEs impact product development. A number of benefits can be experienced as a result of OCEs. \(^9,^11\) First, OCEs have an impact on how work effort is \((I_3)\) planned and \((I_3)\) measured for feature teams. Predicting impactful changes to the product is very difficult without OCEs. By identifying the changes that showed the most impact on the product-wide metrics, teams learn from each other and reprioritize their work. \(^8\) At the same time, teams’ contributions are measured not only by the number of changes they deployed but also with the exact amount of the metrics’ improvements. With OCEs, performance goals for teams take on a new meaning.

Second, OCEs impact how product teams ensure \((I_4)\) the incremental improvement of the product and \((I_4)\) product quality. By deploying only the changes that increase customer value, and because many quality issues can be detected only while the products are used by the customers, software products are improving in every sense. Also, OCEs enable companies to \((I_5)\) optimize infrastructure needs. By observing the effect of a treatment on a subset of users while in the design and execution phase, infrastructure capacity can be estimated for the full deployment.

Finally, OCEs enable an accurate \((I_6)\) discovery and validation of customer value. Because they control for all other factors, they enable companies to accurately identify what the customers value and which changes contribute to that. Knowing this is not only important for the teams developing the features but also for other product-wide teams that are planning investments [for example, see \((I_7)\)].

> We are often surprised by the impact of a change which we thought would be inconsequential, like changing the background color or adding a line on a webpage, on the overall revenue.
> —Data Scientist at Microsoft

**Method**

The objective of this article is to provide an understanding of the key phases of OCEs. To derive this, we conducted a longitudinal in-depth case study at Microsoft’s Analysis and Experimentation group following guidelines described by Runeson and Höst. \(^12\) We analyzed more than 400 experiments conducted at Microsoft between January 2015 and June 2018 while working at the case company. Specifically, we examined experiment notes and experiment scorecards, observed and recorded 22 practitioners individually analyzing various experiments for 30 min each, and analyzed other available documentation originating from eight product teams: Bing, Skype, Office, Windows, Cortana, Xbox, MSN, and Office Mobile. In addition, we used the results of data analysis conducted in our prior studies. \(^10\)

We annotated the collected qualitative data through thematic coding \(^13\) and compared it with quantitative scorecard data revealing the complete experiment outcomes. For example, when several interviewees mentioned a “feature for gradual increase of the treatment allocation,” we recognized this as one of the essential features for the experimentation platform and confirmed its usage by analyzing the experimentation platform logs. To mitigate reliability threats, we employed member checking; for example, multiple researchers and practitioners reviewed our work at multiple points in our study. We mitigated construct validity threats by collaborating only with practitioners that were highly familiar with the process of conducting OCEs. With respect to external validity, we believe that our results are applicable for companies that have software engineering and data-science capabilities similar to our case company. For instance, they are able to release new software multiple times a day and collect metadata from the products while they are used by the actual consumers. Because this study is based on experiences that are derived from a single large-scale software company, further research is needed with case companies that differ in size, organizational aspects, and technical capabilities to improve the generalizability of our results.

**The Experiment Lifecycle**

In many teams, OCEs at Microsoft are an integral step of the feature
release process handled by experiment owners (EOs) that execute trials to evaluate ideas and are responsible for them throughout their lifecycle. An experiment lifecycle consists of three main phases: ideation, design and execution, and analysis and learning. The focus of the ideation phase is twofold: generating a minimal viable hypothesis (MVH) and developing a minimum viable product (MVP). The design and execution phase is concerned with the creation and execution of the experiments. Finally, in the analysis and learning phase, experiments are evaluated, and the learnings from them are institutionalized. A thorough understanding of the phases and their integration in the product development process is critical to fully benefiting from OCEs. We illustrate this partition in Figure 1, and we discuss each of the phases in greater detail in Table 1. To demonstrate the efficacy of OCEs in relation to the presented impacts ($I_1$–$I_6$), we select four example experiments and present them in Table 2. We visualize the ones with the user interface in Figure 2.

**Takeaways**

Our key learning while deriving the experiment lifecycle is that intuitively placing experimentation in the middle or at the end of the traditional “plan-build-test-deploy” process does not suffice. Consider the following recommendations when organizing your experimentation process.

**Plan Experiment Executions**

To facilitate the experiment lifecycle, we recommend that you establish the roles of experiment coordinator and EO in product teams. Experiment coordinators should have an overview of the planned, finished, and ongoing experiments conducted by the EOs. A study at Bing identified that the process from the start of the experiment to the completion of the last iteration takes, on average, 42 days. To facilitate this, several teams at Microsoft hold a weekly “experiment prioritization meeting” where multiple EOs debrief experiment coordinators and provide their input on which MVHs should be tested next, along with the empirical evidence explaining why. To obtain such evidence, they can perform mockup studies within the company and examine past experiments that explored similar ideas.

**Institutionalize Experiment Metadata**

The union of experiment learnings is the validated definition of customer value. Provide practitioners an ability and incentive to capture experiment metadata such as MVHs and MVPs in the ideation phase of the experiment lifecycle. If an MVP is visual (such as a layout change), screenshots should be captured along with any other metadata describing the functionality of the variation. In the analysis and learning phase, experiment scorecard and debrief notes should be recorded. Future experimenters will use the accumulated knowledge to debug and understand exactly what the MVP in another experiment changed. Experiment coordinators will use this information to prioritize future experiments. Similarly, experts designing metrics will use this knowledge to update their definitions to make them more accurately capture customer value.

**Prioritize Trustworthiness**

Experimentation is very sensitive to errors and interferences. Provide practitioners with statistical foundations so they are able to accurately set up experiments. Specifically, provide tools for power analysis and experiment examination to determine to what extent the results of their experiments can be trusted. For example, if your product does not have many active users, changes in variations will likely need to be larger for statistic tests to be valid. To disseminate the importance of rigorous

![FIGURE 1. The experiment lifecycle consists of (a) ideation, (b) design and execution, and (c) analysis and learning.](image-url)
testing practices, we suggest that you give tutorials and provide workshops on a regular basis. At Microsoft, this is done in several different ways. One example is a company-wide experimentation conference titled OneAnalyst. At this conference, held twice a year, hundreds of engineers, data scientists, and program managers are updated on the recent experiments and their outcomes, pitfalls, and lessons learned to guarantee the trustworthiness of the method.

**Protect the Business**

When many ideas are tested, some of them may introduce degradations.

---

### Table 1. The experiment lifecycle phases in detail.

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Challenges and mitigations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MVP, MVH</td>
<td>An MVH should be placed into a queue with other hypotheses to get prioritized based on its expected impact and weighted against its estimated cost of development. Experiment coordinators use this information to prioritize experiment execution. Previous work addresses this by testing an MVP on a smaller sample and gradually increasing the treatment group (such as from minor to major markets) to get more sensitive results.</td>
</tr>
<tr>
<td>2</td>
<td>MVP</td>
<td>The MVP should be tested for stability and common issues that could cause a bad user experience. It can be challenging to know when an MVP is ready for experimentation. Related work discusses scheduling and isolation groups to overcome this challenge.</td>
</tr>
<tr>
<td>3</td>
<td>Design</td>
<td>Concurrent experiments might interfere with each other (for example, experimenting with text and background colors may result in devastating outcomes), and such situations should be automatically detected and prevented. Related work discusses scheduling and isolation groups to overcome this challenge.</td>
</tr>
<tr>
<td>4</td>
<td>Execution</td>
<td>Eventually, EOs need a platform that helps them initiate, iterate (for example, rerun the experiment with more users), and analyze experiments. Open source or commercially available platforms, such as Wasabi or Optimizely, can be utilized. Over time, an in-house platform with advanced features may be needed.</td>
</tr>
<tr>
<td>5</td>
<td>Analysis</td>
<td>When the statistical power of the MVH is fulfilled, the experiment is analyzed. The metrics are computed and analyzed for statistically significant changes. Specifically, success criteria of the MVH are compared to the actual experiment outcome, and a decision is made whether the experiment was successful and which variant won.</td>
</tr>
<tr>
<td>6</td>
<td>Institutionalization</td>
<td>Experimentation at scale should enable EOs to step back and question the underlying assumptions about, and relations between, the lower-level metrics that their experiments are impacting and the higher-level product-wide metrics that are difficult to move in a short-term experiment. This leads to better metrics. Although previous attempts have been made to automate the institutionalization of experiment learnings, more work is needed.</td>
</tr>
</tbody>
</table>
### Illustrative OCEs and their (un)surprising impact.

<table>
<thead>
<tr>
<th>Microsoft Office Word app OCE</th>
<th>Birthday-greeting OCE</th>
<th>Bot-detection OCE</th>
<th>MSN video OCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Office mobile apps product team conducted an OCE with its Word mobile app.</td>
<td>The Bing product team evaluated the impact of a feature that displays a birthday greeting.</td>
<td>Finding search results is a computational operation that consumes resources.</td>
<td>MSN evaluated the value of showing an immersive video experience.</td>
</tr>
<tr>
<td>MVP: The Office team developed and introduced the contextual command bar [Figure 2(a)]. MVH: Team members hypothesized that app users would do more editing on their phones because the new command bar would improve editing efficiency, resulting in increased commonality and frequency of edits and increased user retention.</td>
<td>MVP: Registered users would see an animated greeting on their birthday [Figure 2(b)]. The existing variant did not display this message. MVH: The Bing team hypothesized that this feature would increase the site’s usage without degrading experience.</td>
<td>MVP: Bing product team members developed a more pervasive bot-detection algorithm. MVH: The Bing product team hypothesized that an improved algorithm would reduce the resources necessary for the computation of results without harming user experience.</td>
<td>MVP: The MSN team created a version of their site’s video player that covered a larger area of the page [Figure 2(c)]. MVH: The team hypothesized that the larger video player would enable a more distraction-free experience.</td>
</tr>
<tr>
<td>An OCE was designed with one treatment (a new bar) and one control and sufficient power to detect 1% changes in metrics. The OCE was executed once for two weeks.</td>
<td>The OCE was designed with a control that did not have the birthday feature and a treatment variant with the birthday feature. Due to initial findings (see the “Offline data, however,…” sentence in the “Analysis and learning column”), the experiment was iterated.</td>
<td>An OCE was designed with one treatment and one control. The OCE was iterated several times to tune the bot-detection algorithm.</td>
<td>An OCE was designed with one treatment and one control. The experiment was iterated two times.</td>
</tr>
<tr>
<td>The results of the OCE indicated a substantial increase in engagement (counts of file edits) but no change in user retention. The learnings were shared with other teams. The experiment was considered successful. In addition, a need was identified to conduct more experiments of this type to understand the relationship between the editing activity and user retention.</td>
<td>Several percent of the users that saw the promotion engaged with it. Offline data, however, showed an abnormally high number of birthdays on 1 January, most of which were probably not valid. A follow-up iteration increased the prominence of the “update your birthday” link for 1 January birthdays. Approximately one-fifth of the users who received this birthday wish initiated the update.</td>
<td>Initially, the experiment’s results showed degradations in key metrics, which indicated that user experience was degraded. After several iterations, no degradation to user experience was detected, while a much larger number of bots was identified.</td>
<td>The first iteration of the experiment caused an alert indicating a difference between the observed and the expected split of users between the variants. The analysis of the experiment data revealed a bug in the product’s configuration files.</td>
</tr>
<tr>
<td><strong>Impact</strong></td>
<td><strong>Analysis and execution</strong></td>
<td><strong>Design and execution</strong></td>
<td><strong>MVP: Product quality</strong></td>
</tr>
<tr>
<td>(1) Work effort measurement</td>
<td>(1) Work effort measurement</td>
<td>(1) Work effort measurement</td>
<td>The team identified the exact impact on key metrics of introducing this type of design change to one of its products.</td>
</tr>
<tr>
<td>The team deployed the feature confident that it improves their product.</td>
<td>The team deployed the feature confident that it improves their product.</td>
<td>The team deployed the feature confident that it improves their product.</td>
<td>The team identified the exact impact on key metrics of introducing this type of design change to one of its products.</td>
</tr>
<tr>
<td>(1) Incremental improvement</td>
<td>(1) Discovery of customer value</td>
<td>(1) Discovery of customer value</td>
<td>Users entering their correct birthday is a clear signal of stronger commitment to the product, something that hadn’t been part of Bing’s key product-wide metrics before.</td>
</tr>
<tr>
<td>The team deployed the feature confident that it improves their product.</td>
<td>The team identified the exact impact on key metrics of introducing this type of design change to one of its products.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1) Optimizing infrastructure needs</td>
<td>(1) Optimizing infrastructure needs</td>
<td>(1) Optimizing infrastructure needs</td>
<td>The results of the OCE indicated that by excluding any other factor beside the change in the prediction algorithm, a roughly 10% saving on infrastructure resources is achieved.</td>
</tr>
<tr>
<td>(1) Product quality</td>
<td>(1) Product quality</td>
<td>(1) Product quality</td>
<td>The MSN team learned through an experiment that a previously unknown bug existed in their product. By removing the bug, team members improved the quality of their product.</td>
</tr>
</tbody>
</table>
Consider, for instance, the bot-detection OCE presented in Table 2 that classified real people as bots and served them less valuable content. While it is impossible to completely remove the impact of bad MVHs, you can minimize the degradation to user experience by asking EOs to incrementally ramp up variations and stop experiments based only on large degradations (without considering statistical significance). Or, if an MVP should have low visibility to avoid attention from a competitor, ask EOs to plant their experiments over longer periods of time. In this way, sufficient amounts of data can typically still be collected to satisfy power analysis without placing the new feature in the spotlight.

Integrate an Experimentation Platform
Adopting an experimentation platform that accommodates the institutional learning as well as enabling everyone to easily evaluate their ideas is vital. In fact, most of the critical aspects of the experiment lifecycle can be facilitated through a sophisticated and well-integrated experimentation platform, which we described in a related study. Investment in talent and infrastructure to conduct OCEs is needed as well, although, for companies without internal expertise, commercial and open source platforms that implement parts of the lifecycle are available.

The fact that OCEs enable software companies to accurately and continuously evaluate ideas with customers in product development activities is not novel. What is surprising, however, is that the impact of experimentation extends beyond the traditional use in deciding whether or not to deploy a change. Once integrated into the development process, OCEs provide an unprecedented advantage that is difficult to sidestep and quickly become business critical. The gain of being able to continuously and accurately understand which ideas actually increase customer and, consequently, business value provides a handsome return on the investment needed for the lifecycle to be integrated. Despite many challenges with integrating the experiment lifecycle [Table 1 (see the column “Challenges and Mitigations”)], mitigations have been made available for many of them. For the big rocks that still stand in our way, such as the automation of OCE analysis, more research is needed.

References
3. K. Rodden, H. Hutchinson, and X. Fu, “Measuring the user experience
FEATURE: DATA-DRIVEN DEVELOPMENT

ALEKSANDER FABIJAN is a researcher at Malmö University, Sweden. He conducts his research at Software Center, Gothenburg, Sweden, a European collaboration of companies and universities striving to accelerate the adoption of novel approaches to software engineering. Fabijan received a Ph.D. from Malmö University in 2018. Contact him at aleksander.fabijan@mau.se.

PAVEL DMITRIEV is the vice president of data science at Outreach, Seattle, Washington, where he works on enabling data-driven decision making in sales through experimentation and machine learning. His research was presented at a number of international conferences, such as the Association for Computing Machinery Conference on Knowledge Discovery and Data Mining, the International Conference on Software Engineering, The Web Conference, the Conference on Information and Knowledge Management, BigData, and Software Engineering and Advanced Applications. Dmitriev received a Ph.D. in computer science from Cornell University, Ithaca, New York, in 2008. Contact him at pavel.dmitriev@outreach.com.

HELENA HOLMSTRÖM OLSSON is an associate professor in the Department of Computer Science and Media Technology at Malmö University, Sweden. Her research interests include agile transformation, data-driven development, software ecosystems, and organizational models focusing on empowerment and autonomy. Holmström Olsson received a Ph.D. in informatics from the University of Gothenburg, Sweden, in 2004. She is a frequent reviewer for software engineering and information systems journals and conferences, and she is also a recurrent member of program committees for conferences in these areas. In 2017, she was the program chair for the Software Engineering Advanced Applications conference and the International Conference on Software Business, and she was part of the organizing team for the Rapid Continuous Software Engineering workshop collocated with International Conference on Software Engineering. Contact her at helena.holmstrom.olsson@mau.se.

JAN BOSCH is a professor of software engineering at the Chalmers University of Technology, Gothenburg, Sweden. His research interests include evidence-based development, software architecture, and innovation experiment systems. Bosch received a Ph.D. from Lund University, Sweden. He is the editor of Journal of Systems and Software as well as Science of Computer Programming. He has been the general or program chair of several conferences, served on numerous program committees, and organized countless workshops. Contact him at jan.bosch@chalmers.se.


Closing the Gap Between Software Engineering Education and Industrial Needs

Vahid Garousi, Queen’s University Belfast
Görkem Giray, Independent Researcher
Eray Tüzün, Bilkent University
Cagatay Catal, Wageningen University & Research and Bahcesehir University
Michael Felderer, Blekinge Institute of Technology and University of Innsbruck

Many recent software engineering graduates often face difficulties when beginning their professional careers, due to misalignment of the skills learned in their university education with what is needed in industry. In this article, we report a literature review of the studies that have been done to make improvements on this issue.
engineers. By summarizing what we as a community know in this area, this article aims to benefit the readers (both educators and hiring managers) by providing the overall state of the community with respect to aligning SE education with industrial needs and documenting the body of knowledge in this area.

The Review Procedure
In our review and mapping, we followed the established process for performing SLR studies in SE \(^4\) and used our experience from conducting SLRs in the past. \(^3\) All of the authors conducted each of the steps as a team. We searched the Google Scholar database. Our search string was as follows: (educational needs OR knowledge needs OR desired skills OR essential competencies OR knowledge requirements OR skill requirements) AND (software engineers OR software developers). We address the following review questions:

- What skills are most important in the software industry? Given the rapidly changing nature of SE, we wanted to know if the most important have changed in the last five years.
- Is there evidence of knowledge deficiencies in graduating SE students? What are the topics with highest knowledge deficiencies?
- To what extent are soft skills important, in addition to hard (technical) skills?

We only included papers that focused on aligning SE education with industrial needs and were based on empirical data, such as survey results or interview data. We included the latter criteria to exclude papers based purely on personal opinions. After compiling an initial pool of 94 papers, we voted systematically using the aforementioned criteria. Our final pool included 33 papers. We used meta-analysis, \(^6\) a research method that is a form of synthesis, which combines the quantitative data from primary studies (the pool of 33 papers in this article) to aggregate the results of primary studies to provide a consolidated overview on a given topic.

We provide a more detailed description of our SLR process and discuss how we identified and addressed the potential threats to validity to our review in an online web extras section \(^7\) that shows the 33 papers in our final pool. All of the data that we have extracted from the papers can be found in an online repository formulated as a Google spreadsheet. \(^8\) In this article, we use the “[Pi]” format to refer to the papers in the pool. The data shows that attention for this topic has risen in recent years (Table 1).

More Than 4,000 Data Points From 12 Countries
Most of the papers in the pool had extracted data from one country only, e.g., [P2] had data from the United Kingdom and [P8 ... S11] had data from the United States. The advantage of our metasynthesis (meta-analysis) is that the combined data set has data from 12 countries, which provides stronger evidence on the subject than the single-country studies. The top countries from which data were gathered were the United States (15 papers), Canada (four papers), South Africa (four papers), New Zealand (two papers), and Spain (two papers). The United Kingdom, Norway, Philippines, Jordan, Australia, Finland, and Samoa were each represented in one paper. Two papers had data from both the United States and Canada, and one paper surveyed worldwide data.

The number of data points (survey respondents) varied, with studies that had between eight [P28] and 600 respondents [P21]. Since the studies were primarily conducted in different countries, there is slim chance that a single software engineer could have participated in more than one study in the pool. Thus, when we add up the number of respondents from all 33 studies, we can say that the data and evidence are from up to 4,132 respondents. By combining data and evidence from all previous studies and by including such a large combined data set, our study aims to provide a comprehensive overview.

The Most Important Skills in the Industry
The questionnaires designed for and used by the studies had differences with respect to the concrete SE topics used in them. In other words, when asking respondents to rate (rank) the importance of SE topics, different papers used different sets of SE topics. Six studies used the SE topics as proposed in different versions of the SE Body of Knowledge (SWEBOK) \(^9\) (version 1.0 developed in 1999, version 2.0 in 2004, and version 3.0 in 2014) [P1, S3, S4, S6, S14, S32]. Two studies [P3, S26] used a similar guideline from the IEEE, called the SE Education Knowledge, which was developed in 2004. [P4] used the Association for Computing Machinery (ACM) SE 2004 curriculum guideline. [P26] used the ACM Body of Knowledge of Computing Curriculum for Computer Science. Three other studies used the ACM IT curriculum and three used the ACM Information Systems (IS) curriculum. The remaining 20 studies did not use a single curriculum model, but instead synthesized the list of SE topics either from the literature or by an initial interview with practitioners.
Table 1. A list of the studies reviewed in this meta-analysis.

<table>
<thead>
<tr>
<th>ID</th>
<th>Paper reference</th>
</tr>
</thead>
</table>

(Continued)
With such diverse SE topics used in the studies, we selected the most relevant model, SWEBOK version 3.0. We mapped the SE topics discussed in the papers to the 15 SWEBOK knowledge areas (KAs), which are as follows:

- requirements
- design (and architecture)
- development (programming)
- testing
- maintenance
- configuration management
- project management
- SE process
- SE models and methods
- quality
- SE professional practice
- SE economics
- computing foundations
- engineering foundations
- mathematical foundations.

The next step was consolidating the quantitative data of skill (topic) importance from all of the papers; almost all of them had presented ranking of the most important skills. To be able to cross-compare and synthesize data in a consolidated way, we harmonized the importance ranking data as follows. We normalized the topic rankings in each paper to the range of [0, 1] for each SWEBOK KA. For example, for [P1], three of the 14 ranked topics related to the design KA, including general architecture (ranked 1), object-oriented design (ranked 9), and user-interface design (ranked 12). We calculated the average of (1, 9, 12), which equals 7.33, and divided it by 14, the number of all SE topics in that paper. The normalized rank metric was

<table>
<thead>
<tr>
<th>ID</th>
<th>Paper reference</th>
</tr>
</thead>
</table>
done by student teams, either in class or even together with companies. Mathematical and engineering foundations, as well as SE economics, rank low in both charts. This may highlight the establishment of SE (and its education) as a separate engineering discipline that relies on other sciences, such as computer science, mathematics, and economics. Adopting ideas from these subjects offers new approaches to solving problems in engineering software.

In line with this finding, it is interesting to observe that requirements, testing, and design are considered more important than actual development. However, in our experience, this is not always reflected in SE education, especially if it is embedded into computer science curricula.

Knowledge Gaps: Highlighting the Topics That We Should Teach More

In quantitative terms, eight of the 33 studies also measured the knowledge gap (deficiency) from their survey participant responses, which was usually done by subtracting the importance-in-job measure of a given SE topic from the measure of how much the participant had learned during his or her university education. We extracted the quantitative knowledge-gap values and calculated their normalized average. In Figure 2, we show a scatter plot to visualize the average knowledge-gap values versus their importance. The x axis shows the average importance and the y axis shows the average knowledge gap. In all eight papers, the two factors were shown to be quite correlated and, with increasing reported importance, more of a knowledge gap has generally been reported. The greatest reported knowledge gaps are in the areas of configuration management, SE models and methods, SE process, design (and architecture), and testing. Thus, in general, university programs and companies that are training newly hired staff will focus on these topics. We have also divided the scatter plot of Figure 2 into four quadrants to clearly see the SE topics with low or high importance and a low or high knowledge gap.

Topics in Q1 (high importance, high gap) are those that require the most attention with respect to the need for improvements in SE education in university programs. They have high importance but also have a high knowledge gap. Topics in Q2 (low importance, high gap) should be the focus next with respect to SE education (after those in Q1). They have relatively low importance, but high knowledge gaps in those topics remain and thus need attention for more education and training on those topics.

For topics in Q3 (high importance, low gap), university programs are generally doing a good job, since knowledge gaps in these topics are relatively low, while they are quite important with respect to technical needs in the industry. Only the software development topic falls slightly in Q3 of one of the scatter plots.

Topics in Q4 have low importance and a low knowledge gap, so they are least in need of improvements and the attention of SE education in university programs. The mathematical foundations KA falls into Q4 in both scatter plots.

Hard Skills Alone Are Not Enough: Do You Have Soft Skills?

It is widely discussed in the community that hard (technical) skills alone do not make a great software
FIGURE 1. The most important skills: (a) data from all the papers versus (b) papers published in the last five years only.
and that soft skills are equally important (if not more).

Hard skills are composed of domain knowledge and technical skills, while soft skills are composed of team and interpersonal skills. “Soft skills contribute significantly to individual learning, team performance, client relations and awareness of the business context” [P16].

In 24 of the 33 studies, the importance of soft skills was recognized. We categorized soft skills as teamwork and communication (discussed in 19 studies), leadership (13 studies), critical thinking (11 studies), and others (17 studies). Other important soft skills such as cultural fit, understanding of business drivers, aptitude, attitude, coping with ambiguity, learning and curiosity, and passion/drive to innovate were also mentioned.

One of the studies, [P16], specifically focused on industry expectations of soft skills in IT graduates. The data came from a regional survey conducted in New Zealand in 2016. Key findings from the study that are of interest to educators are as follows. While in-house technical training is widely used to advance graduate skills and teach new technologies, most employers consider these soft skills to be untrainable in the workplace, making them the critical hurdle for employment. Furthermore, studies show that short-term pressure on employers for technical skills can result in overlooking soft skills. One interesting quote from the study is, “The public sector especially needs engineers with a sophisticated understanding of the social environment within which their activity takes place, a systems understanding, and an ability to communicate with stakeholders.” Another is, “Today’s working environment is all about relationships, both internal and external. We need people who can step-up and be accountable without always needing a coach/mentor standing by. People working in isolation contribute more errors than teams.”

Some studies even reported quite bold findings, e.g., survey data of an American study [P9] showed that “soft skills are significantly more important than hard skills for entry-level positions.” A study performed in New Zealand [P5] reported that, “Soft skills are critical skills in SE and makeup seven of the top eight most important skills [in that study].” While, “soft
skills and business skills must be included in curricula,” study [P22] recommends.

These statements are in agreement with our finding that the knowledge area of SE professional practice is of high importance (see Figure 1), which is comprised of topics such as professionalism, group dynamics, and communication skills.

### Other Interesting Findings

We observed many other interesting findings when reviewing the papers. For example, there were suggestions for decreasing an emphasis on certain topics in SE university education (i.e., what we should teach less). [P1] expressed that as, “Participants felt that their university education gave them a much better grounding in mathematics than in software topics” and thus recommended that, “emphasis on certain mathematics topics should be changed [decreased].” The empirical data also showed that “much mathematics is being forgotten, whereas much new software knowledge is being acquired on-the-job.” [P3]
also reported that there is “overemphasis on mathematical topics and underemphasis on business topics” in SE education. [P3] called for less of an educational focus on parsing and compiler design, formal specification methods, digital electronics, and digital logic in SE programs.

Going further, some studies discussed how determining the amount of coverage each SE topic should have is not enough and that educators should teach using “real-world” software system examples. For example, [P27] reported that, “Real-life and practical experience must be included in students’ education.” [P26] also highlighted the need for “more exposure to real life, exercises, team assignments or industry projects.” Some of the authors have had experience in such ideas.10

Other interesting suggestions were made in [P28], as follows. “Instead of a greenfield project, a more valuable experience would provide students a large preexisting codebase to which they must fix bugs (injected or real) and write additional features. Also valuable would be a management component, in which students must interact with more experienced colleagues (students who have taken the class previously, who can act as mentors) or project managers (teaching assistants) who teach them about the codebase, challenge them to solve bugs several times until the “right” fix is found, or who give them sometimes capricious and cryptic weekly commandments on requirements or testing that they must puzzle out and solve together as a team.” The authors of this article often heard similar comments when talking to experienced SE practitioners.

Implications and the Road Ahead

The findings presented in this article show the importance of an SE professional practice and soft skills in general. These include the importance of certain SE activities and skills in SE education (especially requirements for engineering, design, and testing), knowledge gaps in specific areas of SE (especially configuration management, SE models and methods, and SE process), and the importance of real-world examples in SE courses.

The authors have already started to benefit from the findings of the presented review and meta-analysis study in their SE education activities. This review has helped us to identify the most important SE topics, based on the largest synthesized body of evidence in the literature. Also, we found that the greatest knowledge gaps are in configuration management, SE models and methods, SE process, design (and architecture), and testing. Furthermore, in our ongoing university SE courses, we have started to align our teaching materials with the important topics and areas that have the greatest knowledge gaps. Also, in the context of a large software company in Turkey with which one of the authors was affiliated, an industrial training program for potential new hires was recently conducted13 based on the insights provided by this review study. We are certain that the results and findings presented in this article will also benefit other educators and hiring managers by helping them adapt their education/hiring efforts to best prepare the SE workforce.

Finally, the findings also show that mathematical and engineering foundations are often overemphasized in SE programs. This information highlights the need to further establish SE as a separate engineering discipline using knowledge from computer science and other basic sciences, such as mathematics, economics, or even psychology, and to separate computer science from SE university programs.14

References

7. V. Garousi, G. Giray, E. Tüzün, C. Catal, and M. Felderer, “The web extras section of the article ‘closing the gap between software engineering education and industrial needs’.”


Digital Object Identifier 10.1109/MS.2020.2967194
The Sound of Software Development

Music Listening Among Software Engineers

Laura Barton and Gülİpek Candan, University of British Columbia
Thomas Fritz, University of Zürich
Thomas Zimmermann, Microsoft
Gail C. Murphy, University of British Columbia

Listening to music is a common phenomenon among software developers in today's work environments. Music can reduce stress, improve happiness, and even increase performance. We conducted two surveys with 2,242 professional software developers and found that between 63 and 88.2% of participants listen to music at work at least some of the time often when writing code or doing repetitive tasks, and that these listeners tend to be more extroverted.

Music CAN HAVE a positive impact on a human's performance and perception of certain tasks. Studies have shown that listening to music can, for instance, improve performance, enhance mood, and reduce stress during physical exercise,1 work tasks,2 and software development efforts.3,4 At the same time, several works have found possible detrimental repercussions of music in specific areas, such as reading comprehension,5 and that the effects of music on a person may vary based on personality type.5

Today, software developers often work in open-floorplan offices. In these environments, the use of headphones is a common phenomenon, providing an opportunity for individuals to choose to listen to music while working. Given the potential for music listening to be inspirational for productivity, creativity, and concentration,6 we were interested in how professional software developers engage with music during work.

With two large-scale surveys, we investigated the music-listening habits of professional software developers while at work and the effects of this activity. These inquiries provide insights about music listening during software development work that go beyond existing laboratory-based3 and limited-scope in-depth field studies.4 The first survey, which involved 1,445 software developers, delved into when and why music listening occurs, gathering responses about developers' frequency of and reasons for listening to music and the situations and activities in which they do so. In this study, we also investigated what kinds of music they choose, as an exploratory step toward understanding whether there is sufficient variation to enable follow-up explorations into subtleties in music listening6 and the effects of music type on mood and arousal.7

The second survey, which involved 797 software developers, incorporated questions about music listening and personality traits into a broader inquiry on developer personality and development practices. This second survey avoided participation bias and allowed us to determine if
trends from the first study were seen in a larger, unbiased group.

The analysis of the responses shows that most software developers regularly or continuously listen to music during work (63% in one survey and 88.2% in the other), and that developers often do so to cut down on background noise in the office (70.9%) and reduce distractions more generally, something commonly found especially in open-office work spaces. The results also show that music listening helps developers to regulate their emotions by lifting their mood (61.4%), relaxing them (54.6%), or making them feel more energetic (48.3%).

The top reason for not listening to music while working was that it disturbs the developer’s focus (65.8%). This apparent contradiction is partly explained by our examination of the activities correlated with listening to music: while less mentally taxing tasks, such as writing or testing code, are very likely to be accompanied by music, cognitively demanding work, such as code reviews and learning new programs, was rated as being less appropriate for music listening.

Methodology
To examine software developers’ music-listening behavior, we conducted two studies—survey 1 and survey 2—with a total of 2,242 professional software developers.

The focus of survey 1 was to investigate the why and how of software developers’ music-listening behavior. In this study, we asked 20 questions that addressed the reasons for listening or not listening to music, the work activities during which developers are more or less likely to listen to music, headphone usage, and the favorite music during work (see the questionnaire and supplemental material that accompanies this article in IEEE Xplore). We additionally asked about demographics, such as gender, age, educational background, and office size. We received a total of 1,445 responses to survey 1, which we distributed through two channels.

- Using personalized email, the questionnaire was sent to 1,200 developers in a large software company; 486 responded (a response rate of 40.5%), most of them from the United States (95.3%).
- Using social media, we advertised for the survey more globally among professional software developers, and we received 959 responses, most of whom were from Germany (76.2%), Switzerland (10.4%), and Austria (6.4%).

By survey 2 was on how many software developers listen to music and their personality traits. To prevent a self-selection bias of the survey population toward music listeners, we integrated our question in an independent and general survey on developer personality and development practices. In particular, we added one question to the survey asking about whether participants listen to music during work. The survey further contained questions on other development practices and on personality traits. The questions about personality traits were taken from the International Personality Item Pool, a repository of 50 survey questions used to measure personality in the Five Factor or so-called Big Five model: openness, conscientiousness, extroversion, agreeableness, and neuroticism. Participants were recruited through personal email invitations within the same large software company used for survey 1; we received 797 responses to survey 2 (out of 3,000 invitations, a response rate of 26.6%).

Both surveys were anonymous and extensively piloted before we opened them for participation. Participants could enter a drawing of Amazon.com gift cards as appreciation for their time.

Prevalence of Listening to Music at Work
Between 63% (survey 2) and 88.2% (survey 1) of the participants reported listening to music at work at least some of the time. The difference between these two rates could be explained by a possible self-selection bias of survey 1, which was an invitation to developers to participate in a survey regarding listening to music at work. Although the invitation asked to “take the survey regardless of whether you listen to music or not,” it was more likely to attract those who do listen to music. Still, it is safe to say that listening to music at work is common, as a significant majority of developers in both
surveys reported practicing it. In survey 1, only 171 people out of 1,445 (11.8%) indicated that they never listen to music while they work; 141 of those 171 had tried listening to music while working but had stopped.

In survey 1, there was a higher likelihood of listening to music among younger developers, especially those 32 years of age or under (91.7% versus 83.8%). Those with a private office (83.8%) were less likely to listen to music than those who shared an office space (89.4%).

In survey 2, we compared the music-listening behavior of developers to their scores on the Big Five personality traits. We followed a common practice of normalizing the mean to 25 and the standard deviation of the personality scores to five. This facilitates comparison across different populations and groups. Higher scores for a dimension mean that a person exhibits that personality trait more. For the five personality traits, we found three differences that were statistically significant at $p < 0.05$ (a false discovery rate less than 5%): software developers who listened to music while working scored higher on extroversion (25.34 versus 24.43) and openness (25.31 versus 24.47) and lower on conscientiousness (24.78 versus 25.38). Neuroticism and agreeableness had no statistically significant correlation with music habits. In a logistic regression model, the personality traits on their own were not reliable predictors of whether someone listened to music while working.

### Reasons for Listening or Not Listening to Music

In survey 1, we asked participants to choose one or more reasons why they listen to music on the job (if self-identified as a music listener) or why they do not (if not self-identified as a music listener). We identified the possible options from related studies and had a text box for participants to provide additional reasons. See the supplemental material in IEEE Xplore for a list of related studies and additional reasons provided by participants.

For listeners of music at work, the reasons stated for listening, in their order of popularity, are as follows:

1. “cuts down background noise” (70.9%)
2. “lifts my mood” (61.4%)
3. “increases my focus” (56.2%)
4. “reduces distractions” (55.2%)
5. “relaxes me” (54.6%)
6. “makes me more energetic” (48.3%)
7. “motivates me” (48.1%)
8. “improves my productivity” (44.6%)
9. “signals coworkers to not disturb me” (22.6%)
10. “increases the quality of my work” (18.8%)
11. “gives me a sense of control over my aural environment” (16.2%).

The popularity of “cuts down on background noise” (number 1) and “reduces distractions” (number 4) shows that developers seek ways to control their personal environment and manage distractions, especially with the prevalence of shared offices (69.7% of participants shared the office with others). While “gives me a sense of control over my aural environment” was not a popular affirmative response (number 11), it is reasonable to extrapolate from the former two high-ranking responses that this is exactly what developers are doing—by blocking out background noise and other nearby distractions, software personnel are trying to regulate their auditory inputs so that their focus is not disrupted.

The reasons stated for not listening to music at work, in their order of popularity, are:

1. “can’t concentrate/distracts me” (65.8%)
2. “decreases my productivity” (39.8%)
3. “reduces the quality of my work” (26.5%)
4. “I have too many interruptions” (26.5%)
5. “too much overhead” (19.6%)
6. “I’m in too many meetings” (13.9%)
7. “it is not allowed at my work” (12.0%)
8. “indicates to coworkers that I’m available” (7.5%)
9. “lowers my mood” (1.9%).

Interestingly, while 65.8% of those who do not listen to music at work said they do not do so because it is distracting, 56.2% and 55.2%, respectively, of those who do listen at work stated that music “increases their focus” and “reduces distractions.” This apparent contradiction is likely a reflection of the fact that not all activities are a good match for musical accompaniment, such as learning something new, and that developers are not constantly listening to music throughout the day and choose to turn it off for specific reasons, as examined in the following section. It could also be that not all music causes equal reactions in the listener; software professionals in our study specified volume, character, tempo, and the absence of lyrics (at least in a language they understand) as factors that made music a support for their work rather than a hindrance.

Additionally, music provides a way to self-regulate affect (i.e., mood and emotion), encouraging the developer to stay motivated or aiding relaxation when work becomes stressful. “Lifts my mood” was the second-most-popular reason for listening to music (61.4%), with “relaxes me” (number 5) and “makes me more...
energetic” (number 6) both ranking high as well. For sample responses to the questionnaire, see “Selected Quotes From Respondents.”

Activities During Which Developers Listen to Music
Focusing on only those coders in survey 1 who listen to music at work, we examined which activities were most often accompanied by music. Participants chose from a list of activities regarding whether they were less or more likely to listen to music—or if there was no difference. The activities, in order of the most likely to be accompanied by music, are:

- writing code (85.5%)
- repetitive tasks (76.8%)
- building code (64.8%)
- testing code (60.2%)
- writing documentation (48.7%)
- debugging (45.4%)
- email (32.3%)
- management tasks (29.5%)
- code reviews (25.7%)
- program understanding (25.6%)
- multitasking (23.9%)
- learning (19.9%)
- memorization (14.4%).

Not surprisingly, tasks that are largely independent (writing code) or not especially mentally taxing (repetitive tasks) are the most likely matches for music. There is a drop in the frequency of music listening when the tasks become verbally expressive (writing documentation, code reviews, management tasks, emails), or when they require creative problem solving and open-ended thinking (debugging, program understanding, learning), i.e., more cognitively demanding activities that involve different modes of processing.

Many developers switch the type of music based on the task they are

SELECTED QUOTES FROM RESPONDENTS

WHY DO YOU LISTEN TO MUSIC?
- “Sometimes, a task is so dull, music makes it easier to bear.”
- “The holy place to be while developing is right in the flow—music helps entering it.”
- “To reduce boredom. Sometimes I need to write a ton of code that isn’t really that complex. It requires minimal thinking, but it’s still a ton of work. By listening to music there are only two things happening in my world: writing code and humming along to the music. I don’t worry about people walking past my office, squirrels running in trees, or birds flying around. You can argue that I listen to music to escape from this wonderful work environment.”
- “Typing rhythm is faster when listening to music.”
- “Cutting down background noise really is one of the most important reasons. When other colleagues around me in the same office are on the phone or discussing topics unrelated to my work, it’s hugely distracting.”
- “If a task is repetitive and does not require a significant amount of concentration, music distracts me from the repetitiveness of it and allows me to complete more before getting worn out.”

WHY DO YOU NOT LISTEN TO MUSIC?
- “I must use all my resources to deliver high-quality code. Music distracts me—it changes my mind while I concentrate on many things. Music attracts my attention, it generates thoughts and feelings. Coding while listening to music results in more bugs than average—especially conceptional failures are increasing, and those are the most productivity-killing ones.”
- “Osmotic Communication. Sometimes you overhear relevant things only by accident.”
- “I only listen while doing rote-type tasks. I find that if I have a task that requires significant thinking, music reduces my ability to think clearly.”
- “While about everyone has headphones in the office, it is frowned upon to use them continuously throughout the day. One of the fundamental ideas behind team rooms is that information flows more easily because everyone can absorb discussions that they are not involved in. Headphones detract from this goal.”
- “If I’m trying to focus on a new problem or learn a new technology, I tend to turn off the music because it can be distracting. However, all of the noise in the open space is also distracting, so it’s kind of a lose-lose situation.”
working on (39%). For example, one developer wrote, “When writing code, I like music which helps me to concentrate, such as Trance or any other electronic genre. Otherwise, when I’m stressed or angry about something, it goes more in the metal/rock direction, something that helps me to cool down.”

Music-Listening Preferences
We also asked software developers about the music they listen to while working and whether it is the same as what they listen to for pleasure. This information provides initial information for others to build on to explore personal preferences in music listening and their effect on work, such as considering whether different musical styles might lead to different arousal levels. We contrast the music genres that are popular for work and pleasure in the word cloud in Figure 1(a). Developers are more likely to listen to classical, electronic, instrumental, and no-lyrics music while at work. When listening to music for pleasure, they are more likely to listen to rock, metal, and pop than at work. These data may correspond to research that suggests that cognitive tasks benefit from moderate levels of arousal.

More than 50% of developers preferred music genres that typically do not include prominent vocals, e.g., electronic dance music, classical, and film or video game soundtracks. Sixty-eight survey participants explicitly stated that they preferred listening to music without lyrics (or lyrics in a language they do not understand). “Songs with lyrics distract me when I need to focus on writing something,” one developer wrote. Because even rote coding requires linguistic processing, we suggest that developers recognize the distraction caused by the competing information of lyrics in music and therefore try to limit this interference.

We also linked the music genre to the motivation for listening to music; see the word cloud in Figure 1(b). Developers who want to signal coworkers to not disturb them are more likely to listen to music without lyrics. Those who listen to music to increase the quality of their work favor instrumental and classical music. Software professionals who want to be more energetic gravitate toward rock, pop, and electronic music.

Related Work
A multitude of studies has examined the effects of music listening on a variety of activities, showing, for instance, that self-selected and motivational music can lead to a performance increase during physical exercise, and that slower-tempo background music can affect shopping behavior and lead to an increase in gross sales in supermarkets.

When it comes to work, studies on the effects of music have yielded mixed results, with an overall positive trend suggesting that music is beneficial to work performance, helps to reduce stress, and improves individuals’ happiness and satisfaction. For instance, Lesiuk found that listening to music lowered

![FIGURE 1. Comparison clouds showing the music-listening preferences of software developers. (a) The word cloud shows which music genres developers are more likely to listen to at work (red) versus music they listen to for pleasure (blue). (b) The word cloud links the music genre to the motivation for music listening; the music genre is prefixed with a number representing the motivation (for example, 6 represents the motivation “Signaling coworkers to not disturb me”). Not all motivations could be linked to music genres (“Lifts my mood,” “Cuts down background noise,” and “increases my focus”).]
developers’ stress and anxiety without detracting from the logic and syntax of their code. Surgeons had reduced autonomic reactivity and performed arithmetic calculations faster and more accurately while listening to their own preferred soundtrack.\(^12\)

Most studies on the effects of music at work have focused on a specific task, often in a laboratory setting. The study by Lesiuk\(^4\) monitored developers engaged in their typical jobs over several weeks: a baseline control week, two weeks in which they were encouraged to listen to music, a week when they were not allowed to listen, and a return to listening privileges at the end of the study. The self-reported quality of work was highest during the music-listening weeks, and developers spent longer than they intended to complete their tasks during the week without music.

The effect of music listening has also been shown to be affected by a variety of factors, such as a person’s music preference, the habituation to working with music, personal traits, and task type.\(^12\)–\(^14\) For instance, music and noise were inimical to performance on a reading comprehension test,\(^14\) both for introverts and extroverts, although introverts suffered a greater impact. In another study, introverts were negatively impacted by background music during reading comprehension and memorization, while extroverts showed no significant difference in task performance.\(^5\) Daoussis and McKelvie\(^15\) came to a similar result for reading recall tasks and pointed to Eysenck’s theory of cortical arousal as an explanation: introverts have a lower optimum arousal threshold, whereas extroverts require higher stimulation to reach their optimum arousal level for working. Because music provides stimulation, extroverts can find it energizing, whereas introverts are in danger of becoming overloaded by the extra stimulation.

The experimental tasks of previous studies are highly varied in both type of cognitive demand and complexity (for example mental arithmetic,\(^12\) reading comprehension,\(^5\),\(^15\) or memorization\(^6\)) and are not always representative of real technology-workplace scenarios. Only a few studies focus on actual coding and typical developer tasks.\(^3\),\(^4\) Software development can run the gamut from repetitive tasks to extremely demanding and innovative problem solving, and it is likely that music is not a good match for every situation. Most studies use silence as the control, whereas a typical software office will, in fact, be open plan; the choice for a real-world developer is not between silence and music but between background noise and music.

**Discussion**

The participants’ two main motivations for listening to music were to shut out the distractions of the office around them and to regulate their mood when they need to stay energized or manage stress. These habits appear to confirm the previous research on workplace music listening: in short, music can help regulate focus and mood, but not everyone is assisted by the extra stimulus, and, furthermore, music can be distracting when coupled with cognitively demanding tasks.

Open-plan offices may not be avoidable in continuously expanding and evolving technology companies. As we have reported in this article, some developers use music as a means of coping with these office situations. Anecdotal reports on the web suggest that the use of music is not universally accepted across all companies (for example, see “Headphone use has been banned, how should I challenge this?” on Stackexchange.com). An interesting question is whether music is necessary or whether white or pink noise, which blocks out other sounds without itself being engaging, might achieve similar goals. In fact, five of our survey participants indicated that they listen to white noise to mask office sounds without being distracting in and of itself, and 68 coders explicitly wrote that they prefer music without lyrics (or lyrics in languages they do not speak). Several more stated that familiar music is best; for example, two of their comments were “The better I know a song, the better I can focus on my task” and “Led Zeppelin is pretty good for me, because I know every song by heart, so I don’t need to pay much attention to it.” Software professionals seem to have an inherent understanding that music should not be so stimulating as to dominate their central cognitive focus.

Music listening may be especially important for those who have an introverted personality and are easily overstimulated, helping them to find a way to mitigate the effects of office noise.\(^14\),\(^15\) While we found that introversion has some correlation with choosing not to listen to music, using white noise or other noise-masking sound may remove the distractions of background chatter without irritating introverted developers. More study is needed to understand how personal choices and personality traits impact appropriate choices for individuals and companies alike to provide productive workplaces.

Apps such as MyNoise.net, Brain.fm, and Focus@Will promise improved concentration through so-called scientifically designed music that is tailored to different tasks and sometimes even generated by artificial intelligence. There may also be merit in developing
new noise-management technologies, such as headphones that adjust to the listener’s cognitive state using electroencephalogram sensors, similar to Mindset headphones (www.thinkmindset.com), a recent project on Kickstarter. Thirty-one of our participants mentioned that they select their music based on their mood or the task at hand. By adjusting the music to the developers’ cognitive load, the overhead of selecting and managing music could be removed and the music type could be adjusted to avoid any task distraction, especially for cognitively demanding work.

Given that we saw a very wide range of musical preferences in our
study—from black metal to techno to Vivaldi concerts—music selection is personal and should be left to the individual. Not only do developers match their music to their mood and task, but very strong preferences can come into play; one developer could love dance music, whereas another would find it especially annoying. It is inconclusive whether listening to music directly improves coders’ productivity, although the two are generally correlated. However, it does seem clear that developers are happier when they are able to listen to music; this positive affect is more likely to be what improves performance. Therefore, to see benefits, software professionals need to be allowed to choose their own music to listen to over headphones (versus, say, an officewide stereo).

**Limitations**

A limitation of any empirical research is the generalizability of the results, that is, different contexts may lead to different results. Our findings might also have been skewed by the outsize presence of the large company in our population. However, the participants from the large company responded differently than other participants on only a few questions, so we do not believe that it dominated the results. To support replication, we provide the questionnaire (see the questionnaire that accompanies this article as supplemental material in IEEE Xplore).

Listening to music while working is a common practice, but it is done thoughtfully by software developers based on their current mood and task focus. Many of their natural choices are supported by research, suggesting that music can improve performance and productivity under some but not all conditions. Additionally, music may not always be inherently desirable, but is a solution to the problem of ambient open-office noise.

Software developers adjust their workplace listening—the genre, volume, or whether they engage in it at all—to match their current activity, mood, and personality type. Adaptive technologies could therefore learn from what developers are already intuitively doing by tailoring music or other auditory stimulation in the environment to benefit the user.

**Acknowledgment**

We thank everyone who responded to our surveys.

**References**

THIS ISSUE’S “PRACTITIONERS’ Digest” department reports on papers from the 2019 International Conference on Requirements Engineering. Feedback and suggestions are welcome. In addition, if you try or adopt any of the practices included in this article, please send me and the authors of the paper(s) a note about your experiences.

**Design Thinking for Requirements Elicitation**

“Design Thinking in a Nutshell for Eliciting Requirements of a Business Process: A Case Study of a Design-Thinking Workshop” by Levy and Huli describes a design-thinking workshop for facilitating requirements elicitation for business processes. The paper describes simple guidelines to facilitate such workshops:

- identify the problem
- exercise empathy (persona and empathy-map development)
- define (map out the context)
- ideate (discuss solution options following divergent-convergent steps)
- prototype and test (low fidelity, time-boxed).

When these workshops are conducted in a business context, with appropriate time constraints, participants can identify perceptions and uncover business-process problems at a higher level, such as across group or division boundaries. The addition of emotional language to the analytical analysis of the business process creates an opportunity to change the culture among the practitioners involved in the process. The use of design thinking for business-process analysis is a new way of applying these concepts, and it indicates a promising new perspective that could become a useful best practice in IT organizations. The main limitations of the process are convincing people to devote the effort required and creating a proper environment (i.e., noncritical) in which creative ideas can surface. Finally, it is important for an organization to follow up and ensure that workshop outcomes are actually implemented.

To test this approach, the authors conducted an exploratory case study in a large IT development organization. The results showed that applying design-thinking tools increased discussion among stakeholders. The inclusion of input from more stakeholders can lead to identification of new requirements. The authors also implemented the new approach in a health provider organization, to enhance the innovation processes, and a cable television organization, to understand how to motivate end users to use a self-guided problem-solving service. In both cases, the new approach enabled fruitful discussions.

Although they were skeptical at the beginning, participants reported that the Design Thinking Workshop was useful, creative, and engaging, leading to new ideas. The expected benefit for companies that use this approach is an easy, well-defined process that includes people across organizational boundaries and can identify innovative and creative solutions to business challenges, including technological and emotional aspects. This paper can be accessed at http://bit.ly/PD-2020-March-01.

**Gender Differences**

“Analyzing Gender Differences in Building Social Goal Models: A
Quasi-experiment” by Gralha et al.² explores how problem-solving facets related to gender inclusiveness impact the creation and modification of social-goal models. The specific aspects examined are

- motivation for using the software
- information-processing style
- computer self-efficacy
- attitudes toward risk
- ways of learning new technology.

These facets tend to affect women and men differently. The results of a study in which 100 participants (including both university students and practitioners) either created or modified an iStar 2.0 model (a particular requirements modeling language) showed that participants with a comprehensive information-processing style and a more conservative attitude toward risk (characteristics more frequently seen in women) completed the tasks more slowly but more accurately. In addition, the results showed differences between men and women in visual effort, mental effort, and stress.

The steps in this approach, which can all be performed online, are

- preliminary training
- first interview
- mistake-based training
- self-assessment
- peer review
- second interview
- self-reflection.

The evaluation of this approach with students, a proxy for novice requirements engineers, showed a significant reduction in the number of mistakes made during the elicitation process compared with a more standard approach. The participants found the approach useful and easy to understand.

While experienced practitioners might not make the same types of mistakes as study participants, the self-assessment questionnaire could still help them improve their performance. Although the approach has not yet been validated in industry, if one considers students as a proxy for new hires, this approach could be useful for training and improving their
initial performance. In fact, the industrial advisory board for the academic institution of the researchers had a very positive reaction to the work and thought that SaPeer could help students improve their interviewing abilities and build soft skills. The complete training is freely available online at https://zenodo.org/record/2625706#.XZJqDpNKh24, and the paper can be accessed at http://bit.ly/PD-2020-March-04.

References
THE MOST USED knowledge graph today is Google Search. What makes Google Search powerful is its ability to derive knowledge by finding a correlation between the concepts in a search query as opposed to a correlation between words (also known as from strings to things). In a Google search, Google is able to increase the relevance of search results by making connections in its knowledge graph that exceed the possibilities of textual search.

As an example, ask yourself, “What concepts do you associate with the words Steve Jobs”—in this exact combination. You might think of Apple, Pixar, Steve Wozniak, or Buddhism. But someone else might have thought about a completely different person—the plumber down his or her street who is also called Steve Jobs. If you store information about both the entrepreneur Steve Jobs and the plumber Steve Jobs, the challenge is to map the right Steve to the context of the query.

The open source smart graph Weaviate (written in Go and available through GitHub and the Docker network) aims to democratize similar semantic systems to business and community users so they can create their own scalable knowledge graphs. The biggest differentiator between Weaviate and solutions like Google Search and IBM Watson is that most of the data that organizations need for their crucial decision making are not publicly available or cannot be stored outside their own data centers for legislative, privacy, and ethical reasons.

With Weaviate, organizations can set up their own knowledge graph, which means that sensitive and valuable data remain within the premises of the organization. Medical, financial, and government data are just a few examples that illustrate this point.

The enterprise version of Weaviate diverges from the open source version in that it comes with a different license, support, and service-level agreements, but the bits and bytes are the same. This structure allows small-time users like start-up entrepreneurs or academics to create high-end smart graphs similar to those created at large enterprises (Figure 1). Development started in 2016 and has had 12 committers, and almost 8 million lines of code have changed over time. Table 1 shows that currently, the product consists of more than 90,000 lines of code. It is assumed that the overlap between the machine-learning model, data store, and software will have a bias toward the machine-learning model.

This column is comparable to some previous articles that described...
a product with an open source and a commercial license version (Bayesian networks, WordPress, and Eye). The most comparable of the three is Eye, both in application domain (reasoning about qualitative information) and its installed base. The product may be relatively small in terms of size (lines of code); this can be attributed to the growing efficiency of the core programming language (Go) (Figure 2) and the continuous refactoring of the codebase (from its inception until today, almost 8 million lines of code have come and gone to reach the most optimized version).

In the 2016 article “Towards a Definition of Knowledge Graphs,” the authors give a comprehensive definition of a knowledge graph: “A knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge.” When creating knowledge graphs with traditional graph databases, there is much focus on the graph (for example, through linked data and Resource Description Framework [RDF] structures), but our aim was to bring tools to developers that would help with the reasoning part by offering natural language processing through an intuitive RESTful and GraphQL application programming interface (API) while maintaining speed and scalability.

The Contexionary
The core design principles of our smart graph are focused around a user experience-friendly API and scalability with one core focus in mind: enabling anybody to create his or her own high-end semantic knowledge graphs.

The first inspiration came from the article “From Machine Learning to Machine Reasoning,” in which Leon Bottou argues (as explained by Yann LeCun on the Artificial Intelligence Podcast) that a learning system should be able to manipulate (data) objects that are stored in a space and restore them when a new meaning can be attached to these data. Weaviate uses an enhanced and improved semantic vector-space storage model based on word-embedding techniques developed at Stanford University called Global Vectors for Word Representation. A supermarket can be used as a metaphor to understand the structure and organization of the semantic vector space. If you go to a supermarket with a shopping list containing three items (an apple, a banana, and laundry detergent), when you locate the apple, you know that in the supermarket space, the distance from the apple to the banana will be less than the distance from the apple to the laundry detergent. This example is similar to how a vector space storage model based on word embedding works. Concepts that are correlated are stored close together. Weaviate doesn’t store data in a traditional table structure or graph structure but in a spatial representation where data objects are placed together based on their semantic meaning.

Table 1. The size of the product.

<table>
<thead>
<tr>
<th>Language</th>
<th>Files</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go</td>
<td>698</td>
<td>71.725</td>
</tr>
<tr>
<td>JSON</td>
<td>37</td>
<td>15.124</td>
</tr>
<tr>
<td>JavaScript</td>
<td>5</td>
<td>3.112</td>
</tr>
<tr>
<td>YAML</td>
<td>14</td>
<td>795</td>
</tr>
<tr>
<td>Markdown</td>
<td>7</td>
<td>528</td>
</tr>
<tr>
<td>Bash</td>
<td>25</td>
<td>429</td>
</tr>
<tr>
<td>Groovy</td>
<td>2</td>
<td>28</td>
</tr>
<tr>
<td>Plain Text</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>Total</td>
<td>789</td>
<td>91.753</td>
</tr>
</tbody>
</table>

JSON: JavaScript Object Notation; YAML: YAML Ain’t Markup Language; Bash: The Bourne-Again Shell.
This model is called the contextionary. Every data object added to the knowledge graph is placed inside the vector space based on its semantic meaning. This means a user can query for data based on what the data are about (i.e., the context in which they are placed) rather than fixed keywords (Figure 3).

**Development**

For use in an enterprise production environment, the contextionary has to be fast, scalable, and user friendly to be successful. Retraining of machine-learning models should not interfere with the user experience of the developer. Therefore, one of the key features is the ability to add a new data object to the contextionary without the need for retraining the model because knowledge graphs need to be fast. Retraining a model after the addition of new data objects would be unusable in real-life cases.

In the contextionary, this issue is solved by calculating new vector positions for data objects based on the semantic meaning of the group of concepts that describe it. When one adds a new data object through the RESTful API—for example, a company with the name Apple—the contextionary generates a new vector position for this object algorithmically without the loss of semantic meaning. This is done by calculating a weighted centroid based on the known concepts and occurrences of the concepts in the original training corpus.

The vector position of a new object is determined in real time by taking the vector position of the data object and calculating the centroid based on the number of occurrences of the word in the original training corpus. The fewer times a word occurs, the more weight it is given.

The weight factor is added to prevent the centroid from gravitating toward the center of the contextionary. The center is something we want to avoid because, through experimentation, we learned that this is the place where ambiguity exists.

A second feature is the ability to deal with ambiguous words. Because the model works with individual words as tokens, a term with multiple meanings becomes ambiguous. For example, the token “apple” refers to a vector position related to both fruit and the iPhone. By moving the centroid of the search term toward other words (e.g., between “apple” and “company”), semantic placement of the data objects can be determined with varying accuracy. This same algorithm is used to search through the vector space while using multiple keywords.

Although the latest cutting-edge research in natural language processing (e.g., BERT, ELMo, and so on) uses bidirectional vector representations (i.e., a word is given a vector position based on the context rather than as a single representation of that term), Weaviate uses the

```go
func (cv *Vectorizer) occurrencesToWeight(occs []uint64) []float32 {
    factor := cv.config.OccurrenceWeightLinearFactor
    max, min := maxMin(occs)
    weights := make([]float32, len(occs), len(occs))
    for i, occ := range occs {
        weights[i] = 1 - ((float32(occ) - float32(min)) / float32(max-min) * factor)
    }
    return weights
}
```

*FIGURE 2. An example of calculating the occurrences in Go.*

```json
{
    "Company": {
        "name": "Apple Inc.",
        "foundedIn": 1976,
        "foundedBy": {
            "Person": {
                "name": "Steve Jobs"
            }
        },
        "hasCeo": {
            "Person": {
                "name": "Tim Cook"
            }
        }
    }
}
```

*FIGURE 3. A JSON representation of Weaviate’s class and property structure.*
unique representation of a word because it works as a map of meaning in which to place data objects.

The training of concepts is also incorporated. For example, the words “Steve” and “Jobs” often occur in relation to the company Apple. When adding data objects, we give commonly occurring combinations of words more weight when determining the vector position. This further increases the semantic accuracy of the data object. Figure 4 illustrates how these objects are represented; note that all known words have individual 600-dimensional vector representations, like “company,” “name,” “apple,” “in,” “founded,” and so on.

A third important feature is the ability for organizations to add concepts to the contextionary that are specific to the environment in which they operate. The contextionary is trained on many open data sources such as Wikipedia, so it knows about common knowledge. However, a word combination that might not be known (e.g., “The Wonderful Wizard of Woz,” a nickname for Steve Wozniak) is a concept on which the contextionary was not trained. To enable organizations to add domain-specific knowledge, we use the same methodology, similar to the way humans learn new concepts and nomenclature—by defining it in language that is already known. A vector position for the combination “The Wonderful Wizard of Woz” is calculated based on the words and concepts the knowledge graph already knows. Thus, when new data objects are added, the knowledge graph can store the new information based on the semantic meaning it now has for the word combination “The Wonderful Wizard of Woz.”

**Focus on the API**

**User Experience**

What most new technologies have in common is the ease of use. The two most important criteria for choosing a query language are adoptability (is there something we can leverage with a large community?) and simplicity (how can we avoid complex graph query languages?).

We considered SPARQL, the RDF query language developed and defined by the World Wide Web Consortium and available since 2008. Although SPARQL is loved in the academic community, the engineering community had a different opinion: that it is complicated and not very developer friendly. Therefore, we decided to adopt the new graph query language GraphQL, an open source data query and manipulation language that was started at Facebook but is now part of the Linux Foundation. The use of GraphQL comes with a tradeoff; it is less expressive but much easier to use and adopt. There are dozens of libraries available in many languages, and it is easy exposable through the RESTful API (Figure 5).

Exploration syntax is based on GraphQL, which makes it extremely simple to request data and explore the graph. An example of a query is based on a demo data set that is supposed to return names of companies listed on the New York Stock Exchange. Exploring only “windows” would most likely return “Microsoft Corporation,” but by moving away from the concept of “technology” and toward

```json
{
    " synonym": "The Wonderful Wizard of Woz",
    " concept": "Nickname of Stephen Gary Wozniak, co-founder of Apple Inc."
}
```

**FIGURE 4.** A JSON example of adding new custom concepts. All known words have individual 600-dimensional vector representations, like “wonderful,” “wizard,” and so on.

```json
{
    Get{
        Things{
            Company{
                explore: {
                    concepts: ["windows"],
                    moveAwayFrom: {
                        concepts: ["technology"],
                        force: 0.9
                    },
                    moveTo: {
                        concepts: ["glass"],
                        force: 0.85
                    }
                }
            }
            name
        }
    }
}
```

**FIGURE 5.** Weaviate’s GraphQL representation.
“glass,” it is more likely that producers of glass windows will be given.

**Use Case: Automatic Classification and Semantic Search Through Cybersecurity Threats**

Knowledge graphs can be employed for many use cases where semantic search, automatic classification, or knowledge representation plays a role. A good example to illustrate all three functionalities is a cybersecurity use case based on the Mitre ATT&CK Framework, a vast array of different types of cybersecurity threats with corresponding mitigation recommendations. However, mapping messages (e.g., classifying that an email containing information about somebody asking for a password is a phishing attack) is a task that is often still done manually or based on fixed keywords. Based on a description of observations and system inputs, Weaviate enables the automatic identification and classification of the most likely type of attack and recommends corresponding countermeasures.

The semantic search functionality allows searching through vast amounts of security tickets for context rather than keywords alone. For example, phishing-type tickets can be found by querying for “fraudulent attempt to obtain sensitive information” without that relation ever being explicitly made. The words in the query can be found in a position nearby the word “phishing.”

Cybersecurity is just one of the many domains where knowledge graphs can be applied. Semantic search, automatic classification, and knowledge representation are new frontiers within data management because the problems they solve are currently handled manually by creating complex taxonomies or labeling data sets.

**What the Future May Bring: Knowledge Networks**

The next step is the development of a knowledge network peer-to-peer protocol enabling the semantic exploration of concepts across organizations. The protocol we aim to develop determines how peers can securely explore and combine information from their own knowledge network with information stored in Weaviate instances that belong to other peers. A peer can be any entity, such as a university, business, or individual.

Because a query can be agnostic of the language used, a network of Weaviate instances, hosted over the World Wide Web, can be queried without the questioner knowing what keywords to use in advance. Networks can be public or private and hosted within an organization’s own digital walled garden or over multiple datacenters.

We see many opportunities to use the power of knowledge graphs without having to surrender your knowledge to software giants that may exploit that knowledge in unknown ways.

**References**


**ERRATA**

In the “SE for AI” column in the January/February 2020 issue of *IEEE Software* [1], the captions for Figures 1 and 2 were swapped due to an error during production. The correct caption for Figure 1 is “A sample of the industrial applications of AI to SE.” The correct caption for Figure 2 is “A January 2019 Twitter post by Andrew Ng (https://www.andrewng.org). It was retweeted more than 1,000 times and garnered 3,400 likes.”

We apologize for the errors and any confusion they may have caused.

**Reference**

Anaya’s Journey: A Vision for a Future Software Academy

Austen Rainer

Anaya joins the university’s Software Academy as an international student from Lakshadweep in southern India. She is aware of how unusual it is for an international student to study an undergraduate degree at the university and also how fortunate she is to have secured one of the university’s prestigious international scholarships. She chooses to study for the bachelor with honors degree in engineering the software future. She is attracted to the program’s forward-looking perspective, employability prospects, engagement with both practice and research, and breadth and flexibility of content and delivery.

In the beginning, Anaya is anxious about, and struggles with, the cultural and academic differences she encounters living away from home and studying at the academy. She benefits from the coordinated support of the university’s International Relationships Office; the academy itself; and the Women in Tech student society, which introduces her to local meetups for professional and social connections. She joins the Globe International Café, Women Who Code, and the LGBTQ+ Social Group.

Anaya writes to her parents about how she “really likes” the intent and design of the degree program. She describes how it is structured around a coherent and progressive spine of team-based, open-ended project courses at each level of study that incorporate the Academy’s free access to industry-leading online content, such as O’Reilly’s Learning Paths and Manning Publication’s liveBook.

The team-based project courses develop Anaya’s capabilities: to apply science, engineering, commerce and the humanities to difficult problems and social messes; evaluate the appropriateness of software-based solutions; use technologies suited to the problems being tackled; engage the full spectrum of stakeholders; contribute real value; constructively evaluate oneself and others; and welcome such evaluations; and be prepared for careers, problems, and technologies not yet foreseen. All of the assessments for these courses are team based to emphasize the value of others’ contributions. The projects in the program are carefully designed to encourage boldness, allow her to fail safely, and, most importantly, help her become stronger from those experiences.

Senior students act as mentors and appraisers (through their more advanced courses, for which they receive credit), constructively evaluating the quality of Anaya’s work, the outputs from that work, and the meaningful impact of those outputs. They invite her to help organize the academy’s annual international tech conference, “diff[future],” which brings together educators, researchers, practitioners, and policy makers. The theme this
year is “The Software Academy as the Global Difference Engine.”

Instead of lectures, there are open-space discussions and unconferences facilitated by staff, practitioners, and students. Academics and practitioners curate online content—blog articles, tutorials, GitHub sites, gists, and research articles—to provide flexible learning opportunities for students.

Anaya takes a summer internship with a tech startup, AWL Solutions, and a virtual placement with Mozilla, working on the Rust programming language. Both earn her extra credit. She successfully secures a place in Google’s Women Techmakers Scholars Program. For her final-year project, she rewrites the Pattern Python2 library into Python3. Her project becomes the catalyst for a new research collaboration for the university’s Knowledge and Data Science Research Center. “Dad,” she writes home, “I no longer feel like an imposter in this discipline.”

Throughout her time at the academy, Anaya’s work and achievements are publicly celebrated. There are project showcases open to industry and students; short videos on the academy’s YouTube channel; projects on the academy’s GitHub site; and articles from academics, students, and practitioners on the academy’s blog. In a research-based article for the blog, she proposes an updated version of the Turing test: “When Duplex Spoofed Alexa: The CaricaTuring Test.”

Anaya extends her studies by a year to complete a diploma in values-based software engineering, which enables her to complete a one-year placement at the Chikankata Mission Hospital in Zambia, investigating the application of data-intensive, mobile systems to improve rural health in developing countries.

Upon graduating from the Software Academy, Anaya hears the call to leadership so that her growing knowledge and expertise in computing can have a wider humanitarian value. She develops ideas for a new kind of tech startup. She uncaps her pen and writes the first line of a manifesto: “All software projects are societal projects.”

References

Acknowledgments
I thank Prof. William Scanlon for creating the opportunity that motivated me to write an earlier version of this creative story and Dr. Marie Angela Ferrario for prompting me, perhaps without realizing it, to consider publishing it. My gratitude also goes to the editors of IEEE Software for their openness to considering this story for publication. I dedicate this short story to the memory of Atta Mohammed Elayyan (عثمة علیان), who was killed in the Al Noor Mosque shootings, in Christchurch, New Zealand, on 15 March 2019.
IEEE INTERNATIONAL SYMPOSIUM ON HARDWARE-ORIENTED SECURITY AND TRUST

4–7 May 2020 • San Jose, CA, USA • DoubleTree by Hilton

Join dedicated professionals at the IEEE International Symposium on Hardware Oriented Security and Trust (HOST) for an in-depth look into hardware-based security research and development.

Key Topics:
- Semiconductor design, test and failure analysis
- Computer architecture
- Systems security
- Cryptography and cryptanalysis
- Imaging and microscopy

Discover innovations from outside your sphere of influence at HOST. Learn about new research that is critical to your future projects. Meet face-to-face with researchers and experts for inspiration, solutions, and practical ideas you can put to use immediately.

REGISTER NOW: www.hostsymposium.org
Boris Cherny on TypeScript

Nate Black

From the Editor

In Episode 384 of “Software Engineering Radio,” Boris Cherny, author of Programming TypeScript, speaks with Nate Black, explaining how TypeScript can scale JavaScript projects to larger teams, larger code bases, and across devices. TypeScript is a gradually typed language, allowing you to add compile-time verification to a JavaScript project bit by bit. TypeScript aims to be practical by catching common mistakes but without adding too much burden on the programmer. Other topics include: structural typing, type refinement and programmer intuition, when to use escape hatches and how to ban them, interoperability with JavaScript, and using TypeScript with frameworks such as Angular, React, and React Native. We provide summary excerpts below; to hear the full interview, visit http://www.se-radio.net or access our archives via RSS at http://feeds.feedburner.com/se-radio.—Robert Blumen

Nate Black: What is TypeScript?

Boris Cherny: TypeScript is a programming language that compiles to JavaScript. It mostly came out of Microsoft, with contributions from Google. Its type system is structural, static, strong, mostly inferred, and gradually typed. TypeScript is the most successful among gradually typed languages and the most popular language that compiles to JavaScript.

Why are there so many languages that compile to JavaScript?

JavaScript can run on every computer and every phone. Because it’s so ubiquitous, Facebook, Google, and others have built big applications (apps) on it. But it lacks fundamental features, such as static types that are necessary to scale a program across more engineers and more devices. Compile-to-JavaScript languages are intended to overcome limits to scalability.

How similar or dissimilar is TypeScript from JavaScript?

TypeScript compiles to and interoperates well with JavaScript. Every valid JavaScript program is also a TypeScript program. It might not type check, but it will compile. If you have a JavaScript file that ends with a .js and rename it to a .ts extension, that’s a TypeScript program.

What does it mean to scale?

It means more lines of code, more engineers, and more devices. Big tech companies have millions of lines of code. Code will start breaking when you make changes to it if the consequences of changes are not totally clear. Static
Type has helped solve that problem; when you modify a line of code or the app programming interface to some function or some kind of module, you know exactly what else it will break for certain classes of errors.

TypeScript scales across more engineers, serving as documentation when type notations are added to functions (for example, “Function F accepts a number and returns a string”). It scales across more devices; once you start scaling to services and writing multithreaded JavaScript, TypeScript enforces well-defined protocols by letting you type both sides of the communication, thereby increasing confidence. For example, you can write TypeScript that runs on both the browser and server.

How does the compilation process work when JavaScript is run through the TypeScript compiler? What does the TypeScript compiler do?

JavaScript is an interpreted language. When you put JavaScript code into a text file, feed it into your browser, and then run it, it must be compiled to byte code or machine code before the user actually executes the program. It’s the same with TypeScript, a language that targets JavaScript so it can run on any platform that supports JavaScript. TypeScript takes code and then outputs JavaScript code, which you can then run as you would have before.

You wrote that changing type definitions of the TypeScript program won’t change the compiled JavaScript output. How is that possible and what does that mean?

TypeScript is JavaScript plus some types. The types will never affect the output of the program, so you can do what you want with the types. You can make it safer or less safe by using different kinds of types. But the generated JavaScript output will look exactly the same. The types affect type checking done by the TypeScript compiler before it compiles. But it can still compile your code even if it doesn’t type check or if the types don’t totally work. If you want to opt into more safety, TypeScript has various safety flags, one of which controls this behavior and will not admit code unless the program type checks.

How can it compile the code even if it doesn’t type check?

TypeScript looks at the value level of your program as opposed to the type level—the JavaScript part of it, not the TypeScript part—to compile it. The types are used just for type checking, before it compiles.

If the type checking conveys information that it didn’t type check correctly, how would I respond?

When you write TypeScript in an editor that supports it, you will get errors in your text editor warning you that you made a mistake. You can configure your project such that unless there are no errors, the code doesn’t compile. Type errors alert you to likely mistakes in your program that you should probably fix, but if you want, you can ignore them.

What is the paradigm behind TypeScript?

The types are there when you need them but not when you don’t. TypeScript supports various paradigms of programming: functional, object
oriented, and imperative. TypeScript can infer all the types for you, or you can specify them explicitly. It works well with all these styles of programming. The idea is that types are fun and they’re useful, so it shouldn’t be tedious to fix them.

What more can you say about interoperability between TypeScript and JavaScript?

Modern JavaScript programs use a lot of code. JavaScript is a very modular language, so you often have first-party JavaScript code along with stuff from third-party packages that might be written in TypeScript. Your code might be a mix of JavaScript and TypeScript. This is a use case that TypeScript had to support when it was designed. It’s important in JavaScript to be able to use whatever packages you want and for it to work correctly. If these are written in TypeScript, you also get the benefit of type checking and autocompletion in your text editor for free.

Can you isolate those parts of the code that have less strictness or where you don’t have the type information available?

I’ve read that most runtime errors in JavaScript are type errors. Was the goal for TypeScript to eliminate that class of errors?

Yes. Some mistakes are harder for a programmer to identify than others. Mistakes might stem from architectural issues that will take a design review to walk through and understand. TypeScript makes it easy to prevent the really dumb mistakes that could have been caught but weren’t because you weren’t using types. 😞
PURPOSE: The IEEE Computer Society is the world’s largest association of computing professionals and is the leading provider of technical information in the field.

MEMBERSHIP: Members receive the monthly magazine Computer, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

COMPUTER SOCIETY WEBSITE: www.computer.org

OMBUDSMAN: Direct unresolved complaints to ombudsman@computer.org.

CHAPTERS: Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

AVAILABLE INFORMATION: To check membership status, report an address change, or obtain more information on any of the following, email Customer Service at help@computer.org or call +1 714 821 8380 (international) or our toll-free number, +1 800 272 6657 (US):
- Membership applications
- Publications catalog
- Draft standards and order forms
- Technical committee list
- Technical committee application
- Chapter start-up procedures
- Student scholarship information
- Volunteer leaders/staff directory
- IEEE senior member grade application (requires 10 years practice and significant performance in five of those 10)

PUBLICATIONS AND ACTIVITIES

Computer: The flagship publication of the IEEE Computer Society, Computer publishes peer-reviewed technical content that covers all aspects of computer science, computer engineering, technology, and applications.

Periodicals: The society publishes 12 magazines and 18 journals. Refer to membership application or request information as noted above.

Conference Proceedings & Books: Conference Publishing Services publishes more than 275 titles every year.

Standards Working Groups: More than 150 groups produce IEEE standards used throughout the world.

Technical Committees: TCs provide professional interaction in more than 30 technical areas and directly influence computer engineering conferences and publications.

Conferences/Education: The society holds about 200 conferences each year and sponsors many educational activities, including computing science accreditation.

Certifications: The society offers three software developer credentials. For more information, visit www.computer.org/certification.

BOARD OF GOVERNORS MEETING

28 – 29 May: McLean, Virginia

EXECUTIVE COMMITTEE

President: Leila De Floriani
President-Elect: Forrest Shull
Past President: Cecilia Metra
First VP: Riccardo Mariani; Second VP: Sy-Yen Kuo
Secretary: Dimitrios Serpanos; Treasurer: David Lomet
VP, Membership & Geographic Activities: Yervant Zorian
VP, Professional & Educational Activities: Sy-Yen Kuo
VP, Publications: Fabrizio Lombardi
VP, Standards Activities: Riccardo Mariani
VP, Technical & Conference Activities: William D. Gropp

2019–2020 IEEE Division VIII Director: Elizabeth L. Burd
2020-2021 IEEE Division V Director: Thomas M. Conte
2020 IEEE Division VIII Director-Elect: Christina M. Schober

BOARD OF GOVERNORS


Term Expiring 2021: M. Brian Blake, Fred Douglis, Carlos E. Jimenez-Gomez, Ramalatha Marimuthu, Erik Jan Marinissen, Kunio Uchiyama

Term Expiring 2022: Nils Aschenbruck, Ernesto Cuadros-Vargas, David S. Ebert, William Gropp, Grace Lewis, Stefano Zanero

EXECUTIVE STAFF

Executive Director: Melissa A. Russell
Director, Governance & Associate Executive Director: Anne Marie Kelly
Director, Finance & Accounting: Sunny Hwang
Director, Information Technology & Services: Sumit Kacker
Director, Marketing & Sales: Michelle Tubb
Director, Membership Development: Eric Berkowitz

COMPUTER SOCIETY OFFICES

Washington, D.C.: 2001 L St., Ste. 700, Washington, D.C. 20036-4928; Phone: +1 202 371 0101; Fax: +1 202 728 9614; Email: help@computer.org
Los Alamitos: 10662 Los Vaqueros Cir., Los Alamitos, CA 90720; Phone: +1 714 821 8380; Email: help@computer.org

MEMBERSHIP & PUBLICATION ORDERS
Phone: +1 800 678 4333; Fax: +1 714 821 4641; Email: help@computer.org

IEEE BOARD OF DIRECTORS

President: Toshio Fukuda
President-Elect: Susan K. “Kathy” Land
Past President: José M.F. Moura
Secretary: Kathleen A. Kramer
Treasurer: Joseph V. Lillie
Director & President, IEEE-USA: Jim Conrad
Director & President, Standards Association: Robert S. Fish
Director & VP, Educational Activities: Stephen Phillips
Director & VP, Membership & Geographic Activities: Kukjin Chun
Director & VP, Publication Services & Products: Tapan Sarkar
Director & VP, Technical Activities: Kazuhiro Kosuga

Digital Object Identifier 10.1109/MS.2019.2957682

revised 14 January 2020
Keep Your Career Options Open

Upload Your Resume Today!

Whether you enjoy your current position or you are ready for change, the IEEE Computer Society Jobs Board is a valuable resource tool.

Take advantage of these special resources for job seekers:

- JOB ALERTS
- CAREER ADVICE
- WEBINARS
- TEMPLATES
- RESUMES VIEWED BY TOP EMPLOYERS

No matter your career level, the IEEE Computer Society Jobs Board keeps you connected to workplace trends and exciting new career prospects.

www.computer.org/jobs