Speed, Data, and Ecosystems

The Future of Software Engineering

Jan Bosch, Chalmers University of Technology

// An evaluation of recent industrial and societal trends revealed three key factors driving software engineering's future: speed, data, and ecosystems. These factors' implications have led to guidelines for companies to evolve their software engineering practices. //

AS MARC ANDREESSEN wrote in 2011, "Software is eating the world."[1] Industry investment in software R&D is increasing,[2] and software, rather than mechanics and hardware, now defines a product's value.[3,4] So, industry is under severe pressure to improve software-intensive systems' capabilities to deliver on today's software needs.

I analyzed key industrial and societal trends related to these developments. Extrapolating from those trends, I identified three factors that are central to software engineering's continued progress and thus have important implications for software engineering's future. Figure 1 summarizes these trends, factors, and implications.

I conducted this research in conjunction with the Software Center (www.software-center.se), a software engineering research collaboration among seven companies, including Ericsson, Volvo Cars, Grundfos, Saab, and Jeppesen (part of Boeing), and five Swedish universities. Thus, the information is based on significant industry experience.

Trends in Industry and Society

The future is generally difficult to predict. However, several societal and technological trends indicate that certain developments and transitions will occur, even if it's not clear when. Here, I summarize six trends influencing the evolution of software engineering practices. The trends are organized from high level and societal to more specific and technological.

The Shifting Nature of Product Innovation

In the past, especially in the embedded-systems industry, a system's or product's mechanical parts were most often targeted for innovation. By introducing new materials, presenting alternative designs that reduced weight or increased structural integrity, or adhering to fashionable designs, companies could differentiate their products. Even if the product contained electronics and software, these technologies were considered secondary and not necessarily central to the product. The software had to work but wasn't viewed as differentiating for the product.

In addition, software development was subjugated to mechanical development, even if the software could be developed significantly faster than the mechanical system.

The trend. Now, software is becoming the central differentiator for many products, whereas mechanics and hardware (electronics) are rapidly becoming commodities. In addition, the system architecture often seeks to separate the mechanics and hardware from the software to allow for two largely independent release processes. So, software can be updated frequently, both before the product leaves the factory and after it's been deployed to customers. As part of this trend, customers increasingly expect their product's software to evolve.

 0740-7459/16/$33.00 © 2016 IEEE

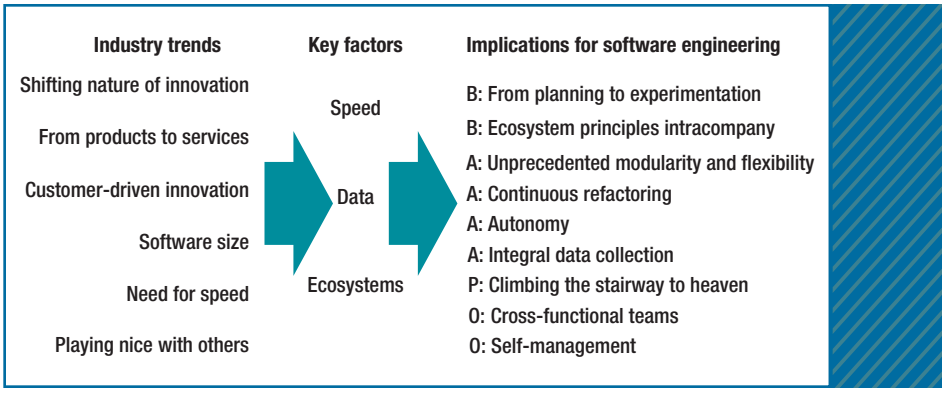

FIGURE 1. The trends and factors influencing software engineering's future, and the implications for business (B), architecture (A), process (P), and organization (O).

The evidence. At the Software Center, several companies have undergone this transformation. For instance, AB Volvo estimates that software drives 70 percent of all innovation in its trucks. Volvo Cars estimates that electronics and software drive 80 to 90 percent of its innovation. Over the last decade, telecom company Ericsson's focus has also shifted, with more than 80 percent of its R&D budget dedicated to software. A recent *Harvard Business Review* article confirmed this trend,[5] as did Valeriy Vyatkin's state-of-the-art review showing that the ratio of software in machinery has doubled from 20 to 40 percent over the last decade.[4]

From Products to Services

Businesses and consumers are increasingly aware of capital expenditures' limiting effects. Owning large, expensive items, often funded by borrowed capital, is expensive and limits a company's ability to rapidly change course when customers demand changes. So, many companies are moving from owning buildings, equipment, and other capital-intensive items to service arrangements in which they pay a fee to access the facility or item.

Consumers, especially in this age of Generation Y, are also shifting their values from owning to having access to expensive items. Developments such as the access economy exploit the fact that many people own expensive items but use them for only small amounts of time each day or week. For instance, the typical car is used less than an hour per day.

The trend. Many industries, including automotive and telecommunications, are fundamentally changing their business models and thus companies' key incentives. This move from products to services has two implications. First, the focus changes from selling as much of a product as possible to providing as many services as possible at the accepted quality level. For example, for a car company beginning to provide mobility services, the question becomes how to provide them with as few cars as possible because the product is now a cost item. Second, companies have an incentive to maximize their products' economic lives. For example, companies often deploy new software in products already in the field because that approach is the most cost effective.

The evidence. Besides academic sources,[2,6,7] two industrial examples illustrate the trend. First, Ericsson's global services unit is growing faster than its product units, in terms of revenue and staff.[8] Operators look to access their network as a service, focusing on customer acquisition and market share growth. Second, automotive companies expect that by 2020, between a third and half of their cars will be used through service agreements rather than ownership.

From Technology- to Customer-Driven Innovation

Technology forms the foundation for innovation. New technologies enable new use cases and let consumers accomplish their goals in novel ways. So, companies invest heavily in technology innovation with the expectation of being rewarded with product differentiation that drives sales and sustains margins.

However, for several industries, despite using patents and other IP protection mechanisms, new technologies tend to become available to all players at roughly the same time, as I mentioned before. So, these companies derive little benefit in terms of differentiation. As technology-driven innovation's benefits decrease, companies increasingly prioritize customer-driven innovation.[9]

The trend. Customer-driven innovation involves identifying and meeting new customer needs as well as better meeting known customer needs. This requires deep customer engagement in both qualitative and quantitative ways. Instrumenting software systems, both online and offline, to collect customer behavior data is critical for customer-driven

innovation because successful innovations are often developed before customers even express the needs the innovation addresses.[9]

The evidence. Petra Bosch-Sijtsema and I studied companies that had adopted new techniques to collect customer insight as part of product development.[9] This trend is also evident in how new industry entrants have disrupted or are disrupting incumbents. From stalwarts such as Amazon for retail and Tesla for automobiles, to Uber for taxi transportation and Airbnb for hospitality, none of these companies disrupted or won in their markets by using better technology than the incumbents. Because the incumbents better understood their customer bases and had vastly more resources, they might have used technology to address customer needs as well as or better than these new entrants. However, the new entrants better understood customers' unexpressed needs and developed innovative approaches to meet them.

> Software, rather than mechanics and hardware, now defines a product's value.

Software Size

For product innovation to move from mechanics and hardware to software, new features and functionality must be realized through software rather than other technologies. This has obvious implications for software size, relative and absolute R&D investment in software, and other product development aspects.

The trend. Depending on the industry, software's size in software-intensive systems is increasing on an order of magnitude every five to 10 years. Industry often underestimates this growth's implications. The main challenge is that a software system 10 times larger than the previous generation's requires new architectural approaches; different ways to organize development; and significant modularization of testing, release, and postdeployment upgrades. This growth also incurs the complications of running a larger R&D organization. To address these challenges, companies employ approaches such as modular architectures, IT services, and open source components.

The evidence. Several studies have documented software growth in software-intensive systems. One of the most illustrative studies is by Christof Ebert and Capers Jones, who analyzed this trend for embedded systems.[3] Vyatkin came to many of the same conclusions.[4]

The Need for Speed

User adoption of new technologies, products, and solutions is continuously accelerating. Once measured in years, user adoption has decreased to months and now weeks and days over the last decade. For example, whereas Facebook took 10 months to reach a million users, the Draw Something mobile app reached a million users in just days. With enterprise's "consumerization," corporations are also demonstrating this need for speed, driving toward faster adoption of new applications, technologies, and systems.

The trend. Companies today must respond to new customer needs and requests at unprecedented speeds, which requires a level of enterprise-wide agility that's often exceedingly difficult in traditional, hierarchical organizations. The need for speed requires companies to pursue different ways to organize, build, and architect software and software development.

Particularly in heavily regulated industries, incumbents often control their products in ways that don't support agility and speed but that slow everything down. New competitors enter these industries from the side and work in relatively unregulated areas, which lets them innovate much more quickly than their incumbent counterparts. Even when compliance is required, new entrants—lacking the existing players' legacy—tend to find more resource-efficient and faster ways to comply.

The evidence. To describe this trend, Larry Downes and Paul Nunes used the compelling phrase "big bang disruption."[10] They presented several cases from various industries in which fast-moving new entrants outcompeted incumbents on price, performance, and user experience.

Playing Nice with Others

No company operates in a vacuum, but many large organizations' internal operations receive orders of magnitude more attention than events outside the company. However, this is changing rapidly in many

industries with software-intensive systems. Companies are increasingly realizing the benefits of playing nice with others and availing themselves of the opportunities presented by using their partner ecosystem more proactively and intentionally.

The trend. The competitive battleground for companies is shifting from focusing on internal scale, efficiency, and quality and serving customers in a one-to-one relationship, to creating and contributing to an ecosystem of players that can include suppliers, complementors, customers, and, potentially, competitors. We see the ecosystem trend not only in the mobile industry's app stores but also in business-to-business markets such as those surrounding SAP and Microsoft Office. Establishing and evolving ecosystems of different partner types is the key differentiator in several industries and might ultimately decide which companies win a market and which get relegated to less dominant positions.

The evidence. Bosch-Sijtsema and I discussed several cases in which companies improved their competitiveness by effectively using their ecosystems.[11] However, one of the most illustrative cases is Apple. By creating its App Store, the company established itself as the dominant player in the mobile industry.[12] While competitors such as Nokia were focusing on device quality, Apple was creating a partner ecosystem to build new iPhone applications—and this became a key differentiator for the company.

The Key Factors

The following three factors are at the heart of the trends I just described.

Speed

Analyses show that the ability to respond quickly to events such as customer requests, changing market priorities, or new competitors is critical to continued success. Companies must respond at a constantly accelerating rate, so speed will affect the entire organization, from its business models to its organizational structures.

Data

With storage costs falling to zero and virtually every product's connectivity exploding, collecting data from products in the field, customers, and other sources is a reality that's still unfolding. However, the challenge isn't the big data but the organization's ability to make smart, timely decisions based on the data. Although many companies still rely on their managers' opinions, future organizations will increasingly use data to inform decision making. So, data collection, data analysis, and decision making based on that data will strongly affect companies' functions, architecture, and ways of working.

Ecosystems

Future organizations will have increasingly interdependent ecosystems. Because the ecosystem is central, business success requires intentional, not ad hoc, management of ecosystem partners. This is true for both large keystone players and the typically smaller complementors. As a result of increased speed and data, companies will have to frequently and aggressively change their role and position in their ecosystems. To effectively manage changing relationships—while forward integrating in the value chain by offering solutions or services and moving backward by providing components or entering adjacent markets—organizations will have to proactively manage the ecosystem. This will strongly affect software architecture, interfaces, and ways of working with partner R&D teams.

Establishing and evolving ecosystems of different partner types might ultimately decide which companies win a market.

Implications

I discuss here the trends' implications for software engineering's future, using the BAPO (business, architecture, process, organization) framework.[13]

Business

This area involves two implications: the transition from planning to experimentation and the adoption of ecosystem principles.

From planning to experimentation. Companies must transition from working with planned releases with detailed requirement specifications to continuously experimenting with customers—for example, by optimizing previously implemented features, iteratively developing new features, or building entirely new products.

This transition is critical for two

reasons. First, research has shown that more than half the features in a typical software-intensive system are never or hardly ever used.[14] Building slices of features and then measuring changes in customer or system behavior is a structured way to minimize investment in features that don't add value. Second, as I discussed earlier, customer needs and desires change rapidly. Companies that don't constantly test new ideas with customers risk being disrupted by companies that more readily identify shifts in customer preference.

> The "stairway to heaven" model describes the evolution of companies' software development processes.

Adopting ecosystem principles. Customer experimentation requires organizing in fundamentally different ways. Traditional functions and hierarchies are no longer sufficiently fast and efficient. Teams will require more autonomy to make decisions locally on the basis of qualitative and quantitative data from systems in the field.

Moreover, the sheer size of the systems being built these days makes it increasingly difficult to handle their complexities. Instead, we must view them as ecosystems with several parts and autonomous organizational units responsible for the parts.

This autonomy's principles are similar to those of software ecosystems in which the parties make decisions independently—within the underlying constraints of the ecosystem's architecture and platform—while contributing to the ecosystem's overall goal. Traditional organizations focus on power hierarchies to centralize decision making. Going forward, teams will be increasingly autonomous, and organizational leaders will need to emphasize purpose and culture, which will provide the guardrails for the teams to operate within. In effect, software ecosystem principles will be adopted inside organizations.

Architecture

This area involves four implications: unprecedented architecture modularity and flexibility, continuous refactoring, autonomy, and integral data collection.

Unprecedented architecture modularity and flexibility. Although modularity and flexibility have been important software architecture elements since their conception, they're often compromised to accomplish operational (runtime) quality attributes such as performance. However, with the increasing importance of speed, experimentation, and team autonomy, modularity and flexibility are being prioritized over other quality attributes.

The microservices architecture that Amazon, Netflix, and others employ is an example of a highly modular architecture. In this architectural style, large complex systems are modeled as collections of small, independent communicating processes. Although controlling and predicting architecture properties might seem difficult, doing so provides high flexibility and modularity and easy monitoring of system behavior. The behavior is predictable for similar system loads, which allows comparisons to earlier system executions.

Continuous refactoring. Especially in the embedded-systems industry, there's a tendency to treat software like mechanical design—that is, design once and use and extend for a long time afterward. For software, this leads to the accumulation of architecture technical debt.

Continuous refactoring of software-intensive systems' architecture will maintain the architecture's suitability for its intended purpose and minimize the cost of adding features and use cases. However, architects will have to be able to constantly identify and prioritize refactoring items to use the allocated resources optimally.

Autonomy. Driven by the transition to services, software's growing size, and human labor's high cost, software-intensive systems will be increasingly autonomous. One of the most illustrative examples is the rapid emergence of semiautonomous cars. Most industries have significant opportunities to transition from semiautonomous systems that help users accomplish their business goals to systems that autonomously accomplish those goals.

Architecturally, autonomy requires reflective functionality: the system collects data about its performance and adjusts that performance according to its goals. Because data from different parts of the system must be combined to derive relevant information for decision making, architectures will include data-fusion functionality.

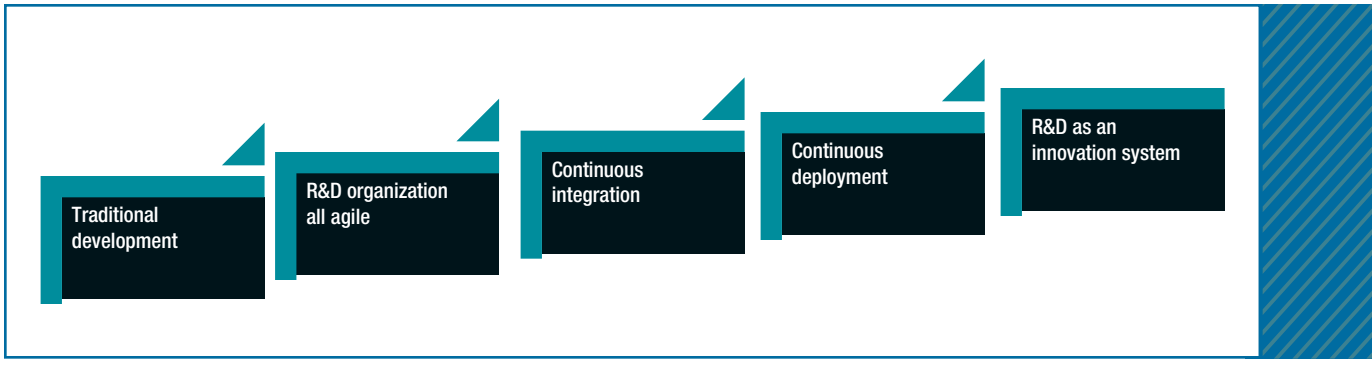

FIGURE 2. The "stairway to heaven" model describes the evolution of companies' software development processes.

Integral data collection. Increasingly, autonomous software-intensive systems need continuous data collection about their operation so that they can control and change their behavior when needed. In addition, feedback from deployed systems and their users is becoming increasingly important for experimentation. So, collecting operational, usage, and other data is rapidly becoming integral to architecture.

Future architectures must assume that data collection at all system levels is required and should be integrated by default. Not collecting data for certain system parts will require an explicit decision. As I mentioned earlier, fusing, aggregating, and abstracting data will be key requirements for architectures. Such capabilities will be required both for the system's reflective functionality and for informing the system's R&D organization.

Process

My colleagues and I developed a "stairway to heaven" model describing how companies evolve their development processes from a traditional waterfall style to agile development (see Figure 2).[15] In that model, companies adopt continuous integration as a core enabling technology. Once new functionality is constantly developed and available at production quality, owing to the continuous-integration environment, customers will want to access new functionality before the regular release process. At this point, the company moves toward continuous deployment. Once continuous deployment fully rolls out, the company can run more experiments with customers and the systems installed in the field.

Each step in the stairway has significant implications for work processes, organizational units, tooling, and general work methods. In most companies, the steps become increasingly challenging as the required changes involve larger and larger parts of the organization. For instance, besides R&D personnel, the verification-and-validation, release, customer documentation, customer support, and sales and marketing teams must be involved to manage this fundamental shift in software deployment. Aligning all these groups in a well-functioning process is much more difficult than adopting agile development because it requires changes in the R&D organization.

Organization

The two implications here are cross-functional teams and self-management.

Crossfunctional teams. Traditional organizations rely on functionally organized hierarchies that group people with similar skill sets—for example, product management, development, verification, or release. Although this allows for pooling of skills and flexible resource allocation to activities, it often leads to slow decision making and execution because of the many handovers between functions and decisions that must go up and down the hierarchy.

Going forward, cross-functional teams will be empowered to make decisions and work with limited coordination between teams. Agile R&D teams are an example. As organizations climb the stairway to heaven, these teams will replace the hierarchical functions. For instance, besides engineers, teams will include members with skills in verification and validation; release; product management; and, potentially, sales, marketing, and general business.

Self-management. A disadvantage of hierarchical management is the time required to make decisions. In fast-moving, highly complex environments, relying on a hierarchical model is a recipe for disaster. The alternative is to decentralize management to the point that individuals

ABOUT THE AUTHOR

JAN BOSCH is a professor of software engineering at Chalmers University of Technology and the director of the Software Center. His research interests include software architecture, evidence-based (or data-driven) software engineering, software ecosystems, and customer-driven and open innovation. Bosch received a PhD in computer science from Lund University. Contact him at jan@janbosch.com.

and teams manage themselves. Agile teams today often have significant autonomy. As teams become increasingly cross-functional, self-management will be required to maintain competitiveness. Management will be more concerned with growing and steering the organization's culture, resulting in individuals and teams making good decisions despite the lack of the traditional hierarchies.

Software's growing role in society is mindboggling at times, and the rate of innovation it enables is impressive. However, all this software must be built, which means that software engineering's importance is also growing. As Yogi Berra said, "It's tough to make predictions, especially about the future." So, the implications I described aren't intended to be concrete predictions but rather extrapolations based on the six trends I discussed. The future will show how accurate I've been.

References

1. M. Andreessen, "Why Software Is Eating the World," *Wall Street J.*, 20 Aug. 2011; www.wsj.com/articles/SB10001424053111903480904576512250915629460.
2. M.A. Cusumano, "The Changing Software Business: Moving from Product to Services," *Computer*, vol. 41, no. 1, 2008, pp. 20–27.
3. C. Ebert and C. Jones, "Embedded Software: Facts, Figures, and Future," *Computer*, vol. 42, no. 4, 2009, pp. 42–52.
4. V. Vyatkin, "Software Engineering in Industrial Automation: State-of-the-Art Review," *IEEE Trans. Industrial Informatics*, vol. 9, no. 3, 2013, pp. 1234–1249.
5. W.C. Shih, "Does Hardware Even Matter Anymore?," *Harvard Business Rev.*, 9 June 2105; https://hbr.org/2015/06/does-hardware-even-matter-anymore.
6. H. Gebauer and T. Friedli, "Behavioral Implications of the Transition Process from Products to Services," *J. Business and Industrial Marketing*, vol. 20, no. 2, 2005, pp. 70–78.
7. R. Oliva and R. Kallenberg, "Managing the Transition from Products to Services," *Int'l J. Service Industry Management*, vol. 14, no. 2, 2003, pp. 160–172.
8. S.W. Elfving and N. Urquhart, "Servitization Challenges within Telecommunications: From Serviceability to a Product-Service System Model," *Proc. 2013 Spring Servitization Conf.* (SSC 13), 2013, pp. 95–100; www.aston-servitization.com/publication/file/35/31_spring-servitization-conference-2013-proceedings.pdf.
9. P. Bosch-Sijtsema and J. Bosch, "User Involvement throughout the Innovation Process in High-Tech Industries," *J. Product Innovation Management*, vol. 32, no. 5, 2014, pp. 793–807.
10. L. Downes and P. Nunes, *Big Bang Disruption: Strategy in the Age of Devastating Innovation*, Penguin, 2014.
11. P.M. Bosch-Sijtsema and J. Bosch, "Plays Nice with Others? Multiple Ecosystems, Various Roles and Divergent Engagement Models," *Technology Analysis and Strategic Management*, vol. 27, no. 8, 2015, pp. 960–974.
12. D. Tilson, C. Sørensen, and K. Lyytinen, "Change and Control Paradoxes in Mobile Infrastructure Innovation: The Android and iOS Mobile Operating Systems Cases," *Proc. 45th Hawaii Int'l Conf. System Science* (HICSS 12), 2012, pp. 1324–1333.
13. F. van der Linden et al., "Software Product Family Evaluation," *Software Product-Family Engineering*, LNCS 3014, Springer, 2004, pp. 110–129.
14. E. Backlund et al., "Automated User Interaction Analysis for Workflow-Based Web Portals," *Software Business: Towards Continuous Value Delivery*, Springer, 2014, pp. 148–162.
15. H.H. Olsson, H. Alahyari, and J. Bosch, "Climbing the 'Stairway to Heaven'—a Multiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software," *Proc. 38th EUROMICRO Conf. Software Eng. and Advanced Applications* (SEAA 12), 2012, pp. 392–399.