



Agile Systems Engineering

Christof Ebert, Vector Consulting Services
Frank Kirschke-Biller, Volkswagen

From the Editor

Systems thinking facilitates mastering the growing complexity of products and uncertainties in volatile business climates. Dependencies must be evaluated, architectures modeled and conveyed, interfaces agreed, and services specified—across a variety of interacting components. From IT with its distributed services to Internet of Things, medical, and mobility systems, there is no software without systems engineering. This article provides a hands-on overview on agile systems engineering from a technology perspective. Stay tuned and let me know your feedback.—*Christof Ebert*

THERE IS NO software without systems engineering. Software is always part of a bigger system. Even a simple application interacting with a human user consists of software building blocks, a service delivery, various hardware systems, user experience, energy management, connectivity and many more. Systems engineering or “systems thinking,” as it is often called, ensures that products deliver more value than the capabilities of their individual parts. Such value beyond the individual components is what we call a system, or even a system of systems. A system

is a set of interacting elements that together achieve a purpose.

Often, we face numerous software engineers working on their respective deliverables yet failing to achieve a comprehensive vision and understanding the entire system with its dependencies. Test and integration will first show the deficiencies, and later enhancements typically demand a huge extra effort, often called “technical debt” or “insufficient maintainability.” A recent interaction with a software team lead underlines this challenge. She was asking for more people due to otherwise not being able to deliver the product in time. Being asked about an architecture analysis and potential refactoring she strongly

insisted that there is “no time for high-level analysis” when deadlines are critical and software must be delivered. Along this misunderstood agile thinking, they did not only neglect traceability and testability, but even did not maintain their own modeling, which they initially saw as so important. With her overly ambitious agile delivery focus, she was unaware that straw fires might look nice in her weekly burn-down charts but not really mean sustainable progress.

The Magic Triangle of Cost, Quality, and Innovation

Participants in our annual surveys in this column observe three significant

challenges.¹ Innovation, cost, and quality have emerged as the single most relevant short-term challenges across industry domains, indicating the need to succeed in a fast-changing world with unclear business drivers. Obviously, the threats of product liability and global visibility of insufficient quality have reached technology companies. The time is gone when software could mature with the early adopters at a bleeding edge. What is delivered must

With the (post)pandemic new normal many companies have reached a turning point—across industries. Global growth has slowed down, but at the same time successful companies are growing with a shift toward autonomy, convergence, ecology, and services. These four key drivers represent a huge transformation, along the entire supply chain. This does not only affect product IT, that is, what used to be called “embedded

Yet complexity is growing due to this fast spread of large software-driven systems of systems.⁴ Unmastered dependencies, lack of portability, and the huge lifecycle cost of such a bottom-up software perspective is driving the current emergence of systems engineering as a mandatory activity for all software projects. Figure 1 exhibits the fast-growing complexity and its challenges. The horizontal axis shows the evolution from traditional software teams to global ecosystems. The vertical axis shows the move from software products toward multidiscipline systems of systems.

One obvious answer to master software growth and the uncertainties in innovative business is agile development. Yet it often reaches limits when products get more complex, and dependencies increase.^{1,2,5} Globally distributed teams, large projects, continuous updating “over the air,” safety-critical systems as well as complex hardware require a targeted adaptation of agile methods utilizing systems engineering practices.

Often, we face projects that fail on essentials such as requirements engineering or insufficient traceability of design decisions to various components. When asked about the reasons, we hear that it is agile, meaning that deliveries matter more than process. Many software products and companies suffer from this misinterpretation of agile development and not seeing the forest for the trees. The implementation of challenging functions to be delivered with periodic builds occupies all energy, leading to a lack of consideration (or understanding) of the system in its entirety. Agile software teams are so focused in finishing their continuous deliveries that they do not realize that the system will become increasingly complex.

Systems engineering or “systems thinking,” as it is often called, ensures that products deliver more value than the capabilities of their individual parts.

be mature and compliant. Mastering this magic triangle top-down demands an agile system engineering approach, that is, “time boxing” and “expense boxing” with a restrictive portfolio management to avoid that scarce resources being wasted in firefighting.^{2,3}

software,” but also remote functionality such as services within enterprise IT, and thus demands a holistic approach. Software matters both as a key value driver and as the glue to dynamically deliver services for previously not specified needs.³

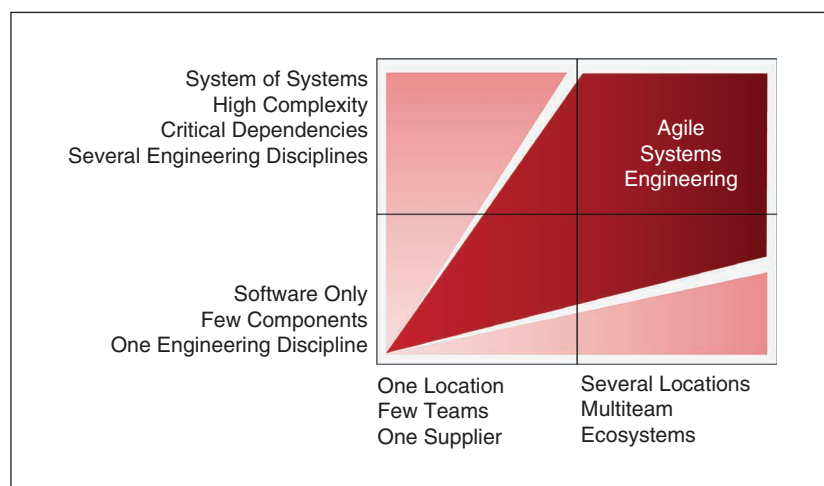


FIGURE 1. The need for agile systems engineering.

Obviously, it needs more than just scaling agile to large software products. It also needs a method to master systems-driven complexity. The power of a systems engineering methodology focusing on architecture and developing a model that comprehends all of the dimensions, behaviors, and components of a system is increasingly understood. It is interesting to note that it is the more seasoned practitioners and experienced senior managers that demand systems engineering, rather than young professionals who had been educated with agile methods and software modeling but never really learned about the major failure points in any project, namely requirements, dependencies, and quality requirements such as compliance, robustness, cybersecurity, and performance.³

Agile Systems Engineering

IT and cyberphysical systems are under enormous market pressure. While they must be uncompromisingly innovative in terms of technology and customer function, the markets demand ever shorter cycle times and continuous efficiency increases while maintaining high safety standards. Flexibility and costs must be constantly improved in global competition.

Traditional development processes that achieve innovation and quality through a complex development and validation process are only partially effective for highly innovative products. Our customer projects and studies show that rework costs can be reduced by 20–50% over the entire product life-cycle by improving requirements and systems engineering.^{1–3,5}

Agile techniques are the essential lever for flexibility and efficiency. But how can agility be mapped to systems engineering? Textbook methods do not fit to industrial practice. Performance,

functional safety, cybersecurity, and a growing global variety of variants require specifically adapted development processes—starting with systems engineering. Governance and traceability must be balanced in the specific

with traceability for end-to-end change management; key performance indicators in the development process, for example, function points, benchmarks, and trend-opportunity indicators.

There is no software without systems engineering. Software is always part of a bigger system.

context with lean procedures. Heavy frameworks such as software platform embedded systems (SPES) and scaled agile framework (SAFe) are suitable for large projects but at the same time are complex and associated with high implementation risks. They limit agility with their built-in overheads due to heavyweight processes.⁵ In addition, they do not address the specific challenges of critical systems such as functional safety and the agile collaboration of several development partners.

Agile systems engineering must consider four dimensions along the entire lifecycle (Figure 2), as described in the following paragraphs:^{6–9}

- *Business:* Clear visibility of the business value of requirements; quality-first focus to avoid expensive and time-consuming rework; product catalogues with feature implementation plan for consistent development roadmaps and accelerated design decisions; technical debt, and its impact on efficiency and future cost; systematic regression techniques for the agile partial deliveries; planned black-box reuse of hardware and software elements,
- *People:* Collaboration models and agile cooperation of teams, which are distributed worldwide; governance of ecosystems and supply chains based on contractable system-level requirements, design, development, validation, and integration; efficient and tool-based distributed knowledge management; mapping of central roles such as an embedded safety officer in all Scrum teams.
- *Process:* Adapted agile methodology with hierarchical scrum for system engineering and orchestration of cross design teams; test-driven requirements engineering for consistency between development and validation activities; test-driven development with continuous build pipeline; flexible synchronization of early software and hardware deliveries; separation of concerns and introduction of frequent synchronization points to allow for continuous integration and deliveries.
- *Technology:* Modeling and simulation as well as model-based system technology and hardware simulation; architectural analysis;

early experience of functions and thus early verification and validation on basis of models; architecture and interface simulation; performance modeling and for bigger systems a digital twin to simulate changes before deploying to the real system. Powerful toolchains offering comprehensive

to achieve consistency and coherence across work products, while facilitating agile engineering deliveries

- ensure business attitude in connecting business needs and market value with engineering management and day-to-day operational decision making

- enforce a rigid quality-first focus with testability and consistency across quality requirements
- manage complexity and project risk starting with requirements engineering and connecting to project management, configuration management, supplier monitoring, and related activities.

One obvious answer to master software growth and the uncertainties in innovative business is agile development.

views on different system aspects and a seamless interface to software development tools play a major role, too.

Agile systems engineering is preserving these dimensions top-down

- facilitate an efficient process with systematic and repeatable results
- enhance teamwork with collaboration and synchronization of activities and deliverables across organizations, locations, disciplines, and lifecycle stages

Agile systems engineering needs careful balancing of different constraints to achieve a useful method across projects and engineering teams. As a case in point, look at the four small screenshots embedded in Figure 2. The lower right picture, for instance, shows what we call the satellite model of systems engineering. Understanding that competent resources are scarce, we have derived knowledge management models that combine push and pull to inform in a top-down manner about governance schemes, while at the same time capturing bottom-up any open question to ensure coherent and systematic

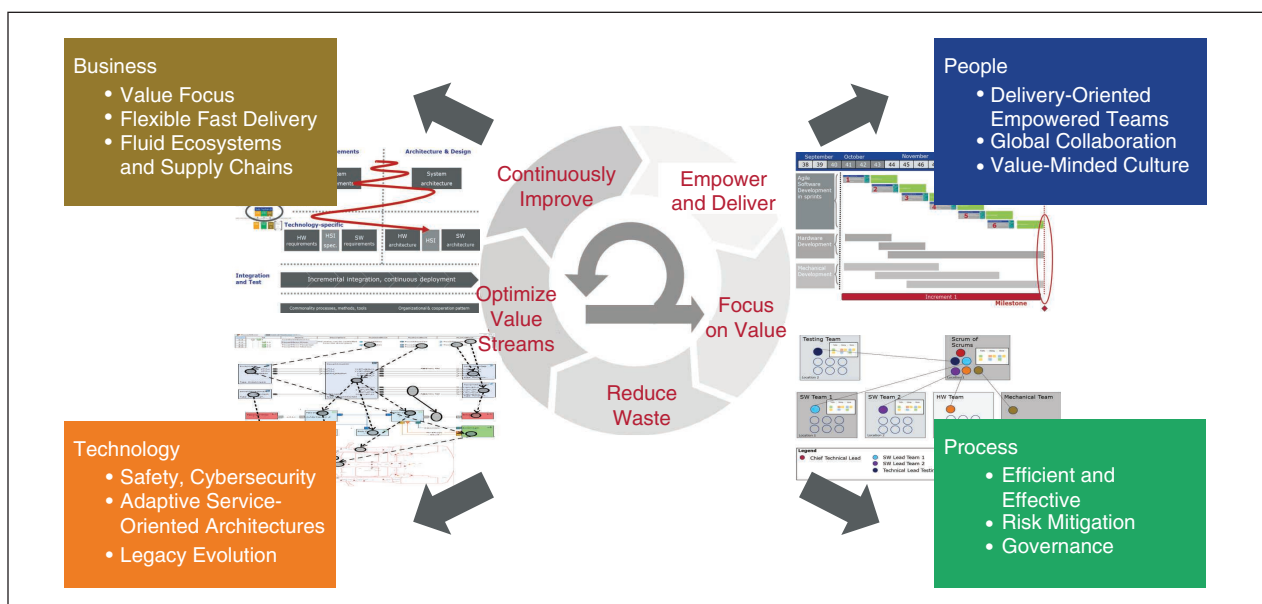


FIGURE 2. The four perspectives of agile systems engineering.

handling. Coaching has become the key enabler of systems engineering, and we typically install such coaching satellites along all three dimensions of an agile matrix organization in functions, locations, and projects.

Systems engineering demands suitable models and pictures to abstract and communicate across heterogeneous user communities. Systems Modeling Language (SysML) which is a UML profile extension, has evolved as graphical notation. Unfortunately, there is no single standard, but many variants which lead to a restricted interoperability of SysML models among different companies and modeling tools. Like UML, SysML models lack precise unambiguous semantics and thus prohibit methods of formal verification and model checking. For instance, traceability models and data consistency checking for model equivalence is hardly possible. Twenty-five years ago, David Parnas coined some simple yet effective guidance for creating software jewels that apply to systems engineering, at least as much as they do to software engineering:⁶

- Design before implementing
- Document your design.
- Review and analyze the documented design.
- Review implementation for consistency with the design.

Although systems engineering demands model exchange and visual information sharing across team and even company boundaries, collaboration with different modeling tools still is extremely cumbersome. Even the Object Management Group's (OMG) XML Metadata Interchange (XMI) which is the standard format for storing SysML metadata information is highly tool and vendor specific. So, we still face many companies using simple drawing tools such as MagicDraw

and Visio with SysML plug-ins. Given the complexity of system models and the need to sharing, analyzing, and maintaining these models, we recommend professional tools, such as Capella, Enterprise Architect, Papyrus, PREEvision and Rational Rhapsody. Papyrus and Visual Paradigm are available as free editions and thus allow a simplified start with a professional tool. Note that with all these

industry projects. Yet, more recent efforts to introduce agile governance frameworks have often been fruitless due to mere overkill.⁵ The same evolution track is visible in systems engineering standards and frameworks that, in their ambition to become a jack of all trades, are currently becoming increasingly complex, often sacrificing usefulness and usability for complexity and overheads.⁷⁻¹⁰

Flexibility and costs must be constantly improved in global competition.

tools, exchange of graphical information across tools is close to impossible. This means a lock-in with any of your modeling tool, and thus careful evaluation up front according to your specific needs.

One observation after almost two decades of agile transformation projects is that each organization needs its own adapted process model that fits its boundary conditions of market, technology, and culture—and is systematically followed.^{5,7} Although often promised as a silver bullet, a standard for everyone does not fit. We have seen that with the Capability Maturity Model Integration (CMMI), which initially was an excellent model to grow process maturity and later became extremely complex when it tried to cover every potential application domain from development to supply chain, from safety to cybersecurity, and so on. The same happened to agile methods like Scrum and Kanban, which are highly beneficial in practically all

Experiences With Agile Systems Engineering

Let us look to a transformation project where we introduced agile systems engineering in a large multinational organization. Figure 3 shows the outline with business needs on the left side, specific changes, both agile and system engineering in the middle, and results on the right side. The biggest challenge in most volatile, uncertain, complex, and ambiguous projects is the continuous evolution of business goals and their steady flux into almost ad hoc design decisions. Development is therefore not primarily driven by a rigid process but rather by an evolving product.

A cornerstone in agile systems engineering is a feature-based process that is based on solid requirements engineering and feature dictionary with subordinate logical functions and software components deployed to variable physical components. Such company-wide information model forms the basis for global cooperation

and black-box reusability. Project-specific feature implementation planning serves as backlog input for system development in an orchestrated approach and enables agile and coordinated functional growth. The prioritization

these perspectives heavily interact, they still must be orchestrated.

Systems engineering by its nature is not only software and code but covers all disciplines. For instance, supply chains with complex technologies

thus viable for future extensions, we need separate layers with a service perspective. Vertical integration is a clear no-go—yet hard to learn, as we see day by day.

Another example is the interdependency of functions that are deployed on different cores of the CPU. Designers must ensure the functions that are similar or communicate a lot with each other should be placed on the same core to reduce communication costs and delays. Cybersecurity, for instance, can waste some +30% throughput performance due to secured communication beyond chip boundaries. Also, real-time scheduling matters. Often, we face engineering teams who when asked how they derive their respective real-time requirements would answer something like, “It is 200 ms because we always use 200 ms.” Consider timing requirements such as a reliable multi-core scheduling algorithm to make sure there is no latency incurred.

Agile systems engineering needs careful balancing of different constraints to achieve a useful method across projects and engineering teams.

of functions and definition of a minimally viable product as well as periodic integration phases for the solution are essential elements of the planning.

Systems engineering must model and assess critical dependencies top-down. We typically distinguish the perspectives listed in Table 1. While

require full transparency across components. Often engineering teams insist that test is only possible with software allocated to hardware entities. Wrong. This would only delay test and decrease portability and flexibility considerations. To allow flexible systems that are maintainable and

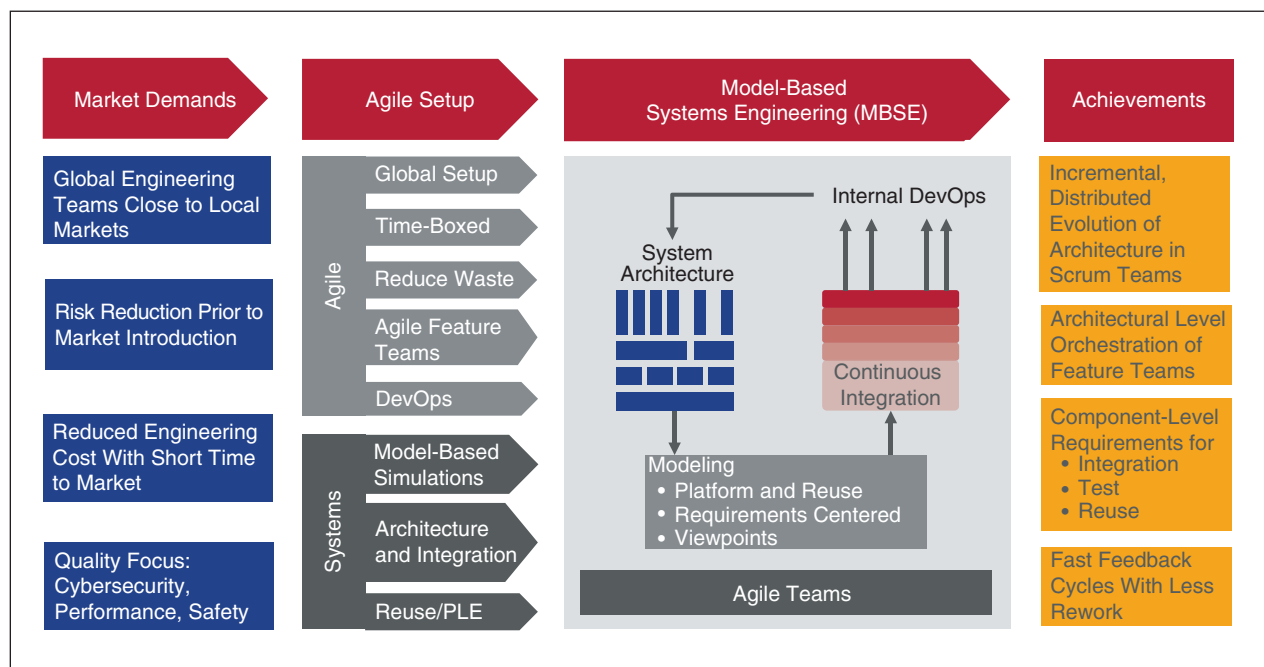


FIGURE 3. Industry case study with agile setup and lean MBSE.

Table 1. Systems engineering perspectives, typical traps, and mitigation guidance.

SE perspective	Relevance	Traps and risks	Mitigation guidance
Value	The perceived value which translates the product to a viable business model	Endless feature lists, without traceability to business value	Ensure a traceability of requirements to a business value, an owner, and preferably a client. Use prioritization
Behavior	Primary functions and their results, such as algorithms and their results, end-to-end	Functions are specified with only sunny-day scenarios	Capture real worst-case feature correlations. Use negative requirements, such as misuse and abuse cases
Structure	Logic building blocks and their interfaces	Systems organically grow by adding content here and there	Apply periodic refactoring and architecture analysis. Use measurements such as coupling, overriding, and so on
Information	System information entities and their relationship, validity, and data protection	Inconsistent data and unclear validity of information	Check data entities at runtime with plausibility checks
Services	Access points to trigger behaviors at runtime including their definition, discovery, and orchestration	Software functions are labeled a service without the necessary orchestration	Build a service-oriented architecture with orchestration, registry and flexible usage
Performance	Runtime interaction, including the management to ensure robustness and availability	Runtime behavior is not considered thus slowing down entire systems	Analyze scheduleability, real-time reaction, and timing constraints based on actual requirements
Reusability	Defined variation points in the system structure	Unmaintainable stand-alone variants	Define variation points for black-box reuse to deploy systems in specific market deliveries. Maintain a master product catalogue and roadmap
Portability	Usage of a system or product in different context	Product components are connected to each other or to a specific hardware	Maintain components and products that can be dynamically allocated
Deployment	Mapping of services to software and software to hardware building blocks	Overly specific constraints—or not considering hardware requirements at all	Specify the underlying HW constraints and possible deployment scenarios already during requirements analysis

Agile systems engineering encompasses the entire product lifecycle and organization, not just the development activities. This includes periodic software maturity reports with a focus on delivery dates, feature-based implementation, and test status reports, which include earned value management, full transparency in open questions, tool integration with defined interfaces, and the coordination and traceability of distributed features to close the circuit. Therefore, systems engineering needs comprehensive lifecycle management method and tool

support for requirements, modeling, traceability, verification, and continuous build and deployment.⁴ The underlying information model must therefore take different perspectives and layers of abstraction to move from product features and system specifications to component requirements and software specifications.

Artificial intelligence (AI) and machine learning (ML) are increasingly deployed for mastering the development process and artifacts.¹ Whether it be debugging, error flagging, or testing, AI is helping teams in speeding

up their checks and processes. One way that ML helps is with coding-based assistants, which helps to identify critical patterns inside data sets. Delivering critical insights to drive new feature development in coming iterations. Giving developers access to a broader range of insights, which can be tested across multiple teams around the world. ML also enhances agile practices through automatic identification of minimum viable regression test suites and updating traceability in test-driven requirements engineering. AI and ML in systems engineering

ABOUT THE AUTHORS



CHRISTOF EBERT is the managing director of Vector Consulting Services, Stuttgart, 70499, Germany. He serves on the editorial board of *IEEE Software* and is a Senior Member of IEEE. Further information about him can be found at <https://twitter.com/christofebert>. Contact him at christof.ebert@vector.com.



FRANK KIRSCHKE-BILLER is a senior manager at Volkswagen in Wolfsburg, 38436, Germany, where he is leading systems engineering topics. Previously, he was with Ford in global management positions. Contact him at frank.kirschke-biller@volkswagen.de.

Yet, more recent efforts to introduce agile governance frameworks have often been fruitless due to mere overkill.

need methodologies for multiparadigm modeling rather than classic UML/SysML perspectives and for mastering AI-based model evolution.

In deploying agile systems engineering we achieved the following results:

- **Reliability:** Self-organization considerably reduces the amount of cooperation in comparison to the classic project management approach.
- **Transparency:** The transparency of the project status and thus the project management is significantly improved.
- **Efficiency:** Speed and quality are improved. Management and teams recognize and appreciate improvements and agile team spirit. In particular, the early definition of requirements and the acceptance of changes during development as part of an agile process are essential for efficiency.
- **Lifecycle management:** Product complexity is better managed and controlled.
- **Quality:** Reuse and embedded quality responsibility create better quality.

The transformation to agile systems engineering requires professional change management. We recommend an agile transformation with evolutionary


deployment, power users for coaching, meaningful piloting, continuous training, and suitable tool support. Such change is not for free, and we strongly suggest integrating related cost, both people and change, directly into the annual budgets, together with performance indicators to track progress and benefits.

Making Agile Systems Engineering Happen

Agile systems engineering supports the continuous and incremental development of requirements and the validation up to the launch and, above all, the ongoing development in the lifecycle accordingly. Due to the increased degree of abstraction with a focus on the customer function at the beginning of the requirements development and analysis, problem descriptions are much clearer, easier, and less redundant. This not only increases the development speed but also ensures clearly understandable domain concepts within the project. Models help with the consistency of system requirements with software requirements and further on to design and validation. They facilitate consistency, transparency, and traceability when changes are implemented at a later lifecycle stage.

Agile systems engineering should not dogmatically use a complex process model but, rather, chose selected individual agile methods as suitably as possible and then continuously optimizing them based on practical experience. This is the only way that the agile culture is perceived as useful and actively used by engineers and their management. The most difficult challenge is connecting short-term software deliveries with strategic systems evolution. It is not either bottom-up agile software deliveries or top-down comprehensive systems

understanding but the combination of both—in each engineering mind. Such transformation means continuous learning and sharing of experiences.

Fred Brooks, the famous computer pioneer and chief engineer at IBM, already understood the relevance of systems engineering four decades ago and underlined its relevance: “The hardest single part of building a software system is deciding precisely what to build. No other part of the work cripples the system if done wrong. No other part is more difficult to rectify later.” 

References

1. C. Ebert and B. Tavernier, “Technology trends: Strategies for the new normal,” *IEEE Softw.*, vol. 38, no. 2, pp. 7–14, Mar. 2021. doi: 10.1109/MS.2020.3043407.
2. International Council on Systems Engineering (INCOSE): Vision for Systems Engineering 2025. <https://www.incose.org/products-and-publications/se-vision-2025> (accessed Apr. 4, 2021)
3. C. Ebert and A. Dubey, “Convergence of enterprise IT and embedded systems,” *IEEE Softw.*, vol. 36, no. 3, pp. 92–97, May 2019. doi: 10.1109/MS.2019.2896508.
4. A. Bucchiaron et al., “What is the future of modeling?,” *IEEE Softw.*, vol. 38, no. 2, pp. 119–127, 2021.
5. C. Ebert and M. Paasivaara, “Scaling agile,” *IEEE Softw.*, vol. 34, no. 6, pp. 98–103, Nov 2017. doi: 10.1109/MS.2017.4121226.
6. D. L. Parnas, “Why software jewels are rare,” *Computer*, vol. 29, no. 2, pp. 57–60, Feb. 1996.
7. Object Management Group: Model Driven Architecture. <https://www.omg.org/mda/> (accessed: Apr. 4, 2021)
8. *Systems and Software Engineering. System Life Cycle Processes*, International Standardization Organization (ISO), ISO/IEC/IEEE CD 15288, Geneva, Switzerland. <https://www.iso.org/standard/81702.html> (accessed Apr. 4, 2021)
9. W. Böhm, M. Broy, B. Rumpe, S. Schröck, K. Pohl, and C. Klein, *Model-Based Engineering of Collaborative Embedded Systems. Extensions of the SPES Methodology*. Heidelberg: Springer, 2021.
10. “TOGAF standard, Version 9.2.” The Open Group. <https://pubs.opengroup.org/architecture/togaf9-doc/arch/> (accessed: Apr. 4, 2021)



IEEE COMPUTER SOCIETY
Call for Papers

Write for the IEEE Computer Society's authoritative computing publications and conferences.

GET PUBLISHED
www.computer.org/cfp

75 YEARS
IEEE COMPUTER SOCIETY

IEEE

Digital Object Identifier 10.1109/MS.2021.3082585