TNO innovation for life

Vision and Outlook for System Evolution and Diversity

TNO Public › TNO 2024 R10716
7 February 2025

TNO-ESI
www.tno.nl
+31 88 866 50 00
info@tno.nl

TNO 2024 R10716 – 7 February 2025

Vision and Outlook for System Evolution and Diversity

Author(s)	Wytse Oortwijn, Jozef Hooman, Debjyoti Bera, Rosilde Corvino, Dennis Dams, Jos Hegge, Dennis Hendriks, Piërre van de Laar, Jade Minten, and Nan Yang
Classification report	TNO Public
Number of pages	19 (excl. front and back cover)
Number of appendices	0

Summary

High-tech cyber-physical systems (CPS) are becoming increasingly diverse: they may have many variations and configurations, might be part of product families, and may be highly customizable. Additionally, CPSs tend to continuously evolve—have variation in time—for example due to technology updates, changing demands, or changing requirements. However, CPSs are typically not designed in a way to easily deal with all aspects of diversity and evolvability. The growing diversity of CPSs, and the need to continously evolve all system variations and configurations, make it more and more challenging to develop CPSs, especially since the demands on system quality are also increasing.

This document describes a research agenda that addresses the evolution and diversity of systems, aimed to improve the efficiency and effectiveness of system engineers by means of *models* and *digital assistants* that operate on these models[1]. We see models as representations of common languages, formalisms, and ways of working that enable interdisciplinary teams to work together fluently.

This document highlights three research areas that together form the approach for managing evolution and diversity:

1. To efficiently and effectively make changes to a diverse system without risk, it is crucial to first understand the system in its current state. Such an understanding can be created by consolidating available system knowledge in *models*, which together form a consistent single source of truth. Not all knowledge of a system may be readily available. For example, there might be important knowledge hidden in (legacy) code implementations, test cases, design documents, archived repositories, etc. Any such 'lost knowledge' should be regained by means of automated tooling, and the models should be enriched with that. Design and implementation artifacts can then be generated from these models, like documentation and tests, which are up-to-date and consistent by construction. Moreover, models and development artifacts should be queryable, for example, using AI/LLM-based assistants, to get quick insights into systems and their changes.

2. To manage system evolution it is important for engineers to get quick feedback on the quality and correctness of systems and any (envisioned) changes to them, to find potential problems and regressions early in a cost-effective way. This is done by enabling early V&V (verification and validation) on the models (i.e., consolidated system knowledge), to be able to reason about the quality of envisioned changes to the system before their implementation, leading to short feedback cycles and significantly reduced cost.

3. Engineers must be able to change the system with confidence. To effectively change the (legacy) implementations of existing software systems, (semi-)automatic massive code transformations should be used. Furthermore, change impact analysis techniques should be used to further reduce the risk of regression. Finally, automatic design/model synthesis should be employed to automatically (re)compute the parts of the design related to (changes to) requirements, leading to correct-by-construction designs and later implementations.

[1] This aim is in line with the ambition of the recent TKI HTSM program plan [1].

Contents

1 Introduction

The goal of this document is to deepen and strengthen the research agenda of TNO-ESI within the program line of "System Evolution & Diversity".

With *system evolution* we mean the process of changing a system in response to changing business needs, opportunities, operational environment, and system requirements. Then *system evolvability* is the degree to which a system can be changed in this respect before it degrades [2]. System diversity, on the other hand, strongly links to variability. *System variability* is commonly understood as the ability to adapt a system to befit a specific context or a particular customer requirement [3, 4]. A *system variant* is then an instance of such an adaptation. With *system diversity* we mean the collection of variants for a system [5].

The importance of dealing with system evolution and diversity is stressed frequently. These have been on the roadmap of ESI since the beginning of the institute, leading to, e.g., the following publications [6, 7]. System evolution and diversity has been part of the industry trends and drivers identified in the recent Systems Architecting and Systems Engineering vision document [8], under 'Industry need 1: Innovation for product families and industry', and 'Industry need 2: System of systems (Sos) and solution ecosystems'.

Furthermore, their importance is also stressed for instance by INCOSE (The International Council on Systems Engineering):

> "The pace of technology advancements continues to accelerate, and impacts the nature of systems solutions along with their positive and adverse effects on society" [9]

as well as in a Software Engineering Research and Development agenda by CMU (Carnegie Mellon University):

> "When we consider the software-reliant systems of today, we see that they are not static (or even infrequently updated) engineering artifacts. Instead, they are fluid—meaning that they are expected to undergo almost continuous updates and improvements and be shown to still work" [10]

and by ITEA 4 (the Eureka[2] Research, Development & Innovation cluster on software innovation):

> "The ability to develop configurable software modules will be a key element for the development of digital loops. Developing an ad-hoc software for each configuration of complex systems will be out of reach. We need to master this configurability feature to successfully and productively control and optimize complex systems. Consequently, we also need an efficient management of the variants for the different configurations" [11]

There are a number of possible reasons for the evolution of a system. For example:

- To obtain new or improved functionality, including non-functional properties, such as cost price reduction or improvements of performance, reliability, safety, or sustainability. Drivers

[2] `https://www.eurekanetwork.org`; Accessed 2024-03-21.

for such changes may come from user requests, marketing, competition, industry roadmaps or legislation.

- Changes because of organizational aspects, such as new or retired staff, internal reorganization, and outsourcing.
- Technology updates:
  - Domain specific updates, e.g., using the latest camera, a new type of generator, or AI techniques. Typically, these will have an impact on the user experience, such as improved images, faster response, etc.
  - Generic updates, e.g., a CPU update, a new version of the operating system, changing obsolete middleware, solving security issues, bugfixes, or removing technical debt. Usually these changes (should) have no impact on the user experience. These changes are mainly driven by the availability of new technology, updates, or end of service by 3rd party suppliers, obsolete hardware, and interoperability requests.

Possible reasons for system diversity include:

- Support for product families, i.e., clusters of systems that share a large subset of functionality and differ on a few specific variation points.
- System customization, e.g., allowing them to be flexibly used for a variety of different use cases. Examples are flexible manufacturing systems that can produce various different types of products [12], like production printers.
- System integration, for example to reuse system (sub)components in different contexts. This is common practice in, e.g., software, which is often continuously adapted to drive several different machines, possibly over multiple product generations.
- System maintenance, e.g., allowing a system to (still) be used while the environment changes. For example, software should continue to run reliably under a new version of the operating system while still also supporting the older operating system.

This document describes the TNO-ESI roadmap and research agenda that addresses system evolution and diversity. The remainder of this document is organized as follows. Chapter 2 elaborates on the challenges (Section 2.1) of system evolution and diversity and the vision with respect to these fields (Section 2.2). Chapter 3 discusses the approach for meeting the envisioned situation, which leads to a number of concrete research questions (Section 3.1). Chapter 4 gives related work, and Chapter 5 concludes.

2 Problem and Vision

Typically, a system is not designed to easily support all aspects of evolvability and diversity. In general, it is impossible to predict all future changes and application areas of a system. Moreover, system evolution and diversity are hampered by several cross-cutting factors:

- There is often a limited set of tests.
- Documentation gets outdated easily. In general, information is scattered at many places, inconsistent, and out-of-sync.
- The knowledge and understanding of a system is often implicit, undocumented or no longer available, for example since the domain expert has left or retired. The lack of system understanding has a major impact on development efficiency.
- A consequence of limited system understanding is that it requires significant time and effort for new team members to get productive. Instead of being able to consult up-to-date documentation, they often need to be mentored by experts and senior team members, impacting their productivity as well.

Furthermore, there are various challenges that hamper system evolvability, notably:

- Often a product evolves from a first prototype and when successful more and more features are added incrementally to meet customer demands. Moreover, in the high-tech domain, equipment stays in the field for decades, while many aspects change frequently such as requirements, technology, and its environment. Developing, maintaining, and supporting all features and variations of a product that may arise over time, in such a way that they all meet the customers' expectations until the product is end-of-life, is challenging.
- Software typically evolves over many product generations. Consequently, maintenance costs are growing at the cost of developing innovative features for customers [13]. Software development is slowing down and becomes more unpredictable, and this trend is bound to get worse with the increasing complexity of code bases. Maintenance is estimated at 75% to 90% of software development life cycle cost [14, 15]. Reducing the software maintenance effort, and making effective use of software legacy (rather than considering it an ever-growing burden), are major challenges.
- Systems consists of various design and implementation artifacts (documentation, requirements, tests, etc.) that quickly get outdated and out-of-sync as the system evolves. Keeping all artifacts consistent and up-to-date is challenging, especially under pressure of deadlines.
- Bugs in system design and (software) implementations are often found too late in the development process, leading to costly field issues that are difficult and time-consuming to diagnose and resolve.
- High-tech systems are developed by decomposing the system into (often many) components, that each have their own responsibility and are worked on by separate teams. These components interact with each other through interfaces, in such a way that together they are responsible for making the overall system function according to its requirements. As a result, engineers and architects typically work within their own relatively small scope, and may lack an understanding of how their component interacts with the rest of the system. When engineers lack such an understanding, the impact of making changes to the component is unclear, which increases the risk of introducing regressions. A lack

of understanding can thus lead to integration problems, e.g., when some worked-on component needs to communicate with other components in the system. Getting a (quick) understanding of the impact of local changes to the overall system is a challenge, which holds especially for new team members, or when dealing with (legacy) components whose knowledge has been lost.

Additional challenges with respect to system diversity are:

- Different needs from customers and market segments usually lead to many product variations. This makes it difficult to maintain, test, and evolve all these variations. The system complexity may increase significantly, possibly even exponentially, in the number of system/component/product variants.
- Different system variants may quickly go out-of-sync, for example because some system variant may be hard to test since it requires a specialized environment. An overview of variants and their commonalities and differences is often lacking, which further increases the complexity of software evolution and maintenance.

These challenges are in-line with the highlighted, more general trends and challenges reported in [8], in particular 'Increasing dependence on complex systems' (e.g., how to deal with evolving or unknown system contexts?), 'Growing system diversity' (e.g., how to deal with systems becoming more and more diverse?), and 'Growing scarcity of engineering experts' (e.g., how to reduce the learning curve for working on diverse systems?).

2.1 Problem statement

System evolution and diversity management is challenging since systems become more and more complex, require faster and more changes, while at the same time the demands on system quality increase.

Systems typically need to operate correctly in a variety of contexts, leading to variability, and therewith increased system complexity. Nevertheless, despite this high complexity, systems have to rapidly change and evolve to remain competitive, while also being actively maintained to remain up to date. Unexpected propagation of the effects of system changes and undesired emergent behavior often lead to a long and unpredictable test and integration phase. Maintenance can be a significant effort hampering innovation, causing development to slow down and becoming too unpredictable. Despite diversity and continuous change, systems are expected to behave as intended without disruption, while also being/staying safe and reliable.

Not only systems, but also their development teams evolve. Human resources are scarce, and people come and go. Experts often spend significant time in helping new team members get up-to-speed due to a lack of consolidated knowledge to be consulted instead. A lack of consolidated knowledge hampers efficient system development and increases the risk of introducing errors and regressions. This conflicts with the need for more and faster innovations.

2.2 Vision

Our ambition is to improve the efficiency and effectiveness of system engineers by means of *models* and *digital assistants* that operate on these models[3]. This ambition is consistent with the SERC (Systems Engineering Research Center) research roadmaps [16], for example by their

[3]This ambition is in line with the recent TKI HTSM program plan [1], which expresses the ambition to minimally double the efficiency and effectiveness of system engineers in 2035 by means of digital engineering assistants.

envisioning of “Human-Machine co-learning” where humans and machines strengthen and complement each other for more efficient and effective system engineering. The ambition is also shared by CMU [10] which envisions that engineers work closely together with automated and AI systems, by automatically determining the best possible implementations based on the expressed intent of the engineer, in a fluid iterative process. Also INCOSE [9] envisions that system engineers will be assisted by (AI) algorithms to be more efficient and effective in delivering solutions.

Our general vision is that system evolution and diversity is managed by engineers that are optimally supported by digital agents, which operate based on consistent information that is captured in models, to allow engineers to deal with the increasing system complexity. The two main components in this vision are:

A Complete and consistent *models* that capture essential system information, like for instance requirements, architecture, guidelines, interfaces, code patterns, behavior, variation points, configurations, evidence of system correctness, and digital twins (i.e., virtual system representations that are updated with operational data). This thus links to the approach of Model-Based System Engineering (MBSE) [17].

B Digital assistants for computer-aided model-driven (re-)engineering, to

- Capture unconsolidated system knowledge in expressive and unambiguous models that together form a (evolving yet shared) single source of truth, thereby making this knowledge insightful for system engineers;
- Extract knowledge, facts and insights from existing design and implementation artifacts, like documents, code, and execution logs, into such models;
- Remove technical debt (semi-)automatically, including computer-aided discovery of code patterns, static analysis, and massive code transformations;
- Generate and synthesize suitable artefacts from models such as documentation, visualization, monitoring, code (e.g., supervisory control software), and tests;
- Check model quality and correctness early in the development process (i.e., ‘shift-left’), enabling cost-effective quality engineering;
- Check conformance to architectural principles and code guidelines;
- Perform design space exploration, to decide on the most valuable changes or the most optimal design choice; and
- Reason about the impact of changes to ensure risk-free system and software evolution.

Point A listed above will be domain specific yet based on general formalisms and tools, using standards as much as possible; key is to avoid duplication to obtain a single source of truth.

The digital assistants described in Point B will be domain independent and generic, by working with the general formalisms and tools that Point A will be based on. These digital assistants can be used to obtain on-the-fly suggestions and feedback, to validate and verify the information of Point A, and to make decisions on changes. Moreover, they can be used to ensure correctness of the information after changes and update the evidence about system correctness.

In addition, there should be processes in place to ensure consistency of the information and prevent technical debt (also on the modeling level). The models of Point A should represent common languages, formalisms, and ways of working, that enable interdisciplinary teams to work together fluently and consistently. This thus links with the approach of MBSE, and is consistent with the recent Systems Architecting and Systems Engineering vision document [8] (e.g., see Sections 5.1 and 10 of that document).

3 Approach

The systematic approach for getting to the envisioned model-driven computer-aided software evolution and diversity management is threefold:

1 Consolidate system knowledge and (semi-)automatically regain any lost knowledge in models that form a single source of truth.
   a. Consolidate system knowledge as multidisciplinary models forming a single source of truth.
   b. Gradually extend the scope and expressiveness of these models.
   c. Extract existing knowledge from design/development artifacts to get to more and more complete models.
   d. Obtain information about the system by querying the models.
   e. Generate consistent design/development/documentation artifacts from the models.

2 Cost-effectively determine the quality and correctness of system designs, implementations, and any changes to them, to find potential problems and regressions early.
   a. Smart system maintenance, e.g., by help of (generative) AI, or augmented reality assistance during maintenance.
   b. Early verification and validation (V&V) on models.
   c. Gradually improve the V&V capabilities to handle increasingly rich models and deal with system diversity.
   d. Efficient mix of techniques for V&V and knowledge extraction.

3 Change the system in an efficient, correct, and risk-free manner.
   a. (Semi-)automatic massive code transformations.
   b. Comprehensive relational analysis for comparing models and development artifacts.
   c. Change impact analysis, for assessing the impact of local changes to the global system.
   d. Computer-aided evolution, e.g., giving advice on how to evolve the system.
   e. Automated synthesis of correct-by-construction system designs.

In the first phases of working out this approach, the focus will not be on completeness, but on consistency, avoiding duplication of information, and preventing the creation of (more) technical debt. The first design assistants may be lightweight, reducing manual work to provide fast benefits. Then models can be learned and updated using operational data, making them gradually more complete and allowing them to remove technical debt.

Point 1. To efficiently make changes to (diverse) systems and their software without risk, it is crucial to understand the system in its current state. However, the required knowledge for such understanding may be unavailable or undocumented, or only be implicitly available, for example by being hardcoded in software or test scenarios, and therewith hard to access.

Point 1 in the approach is to consolidate system knowledge in unambiguous models that form a single source of truth (implying also that these models and the information they capture are consistent). Over time, the scope and expressiveness of these models are gradually extended, to be able to capture increasingly richer aspects (like variability and product families, time

and resources, more disciplines like mechanics and electronics, and architectural information from systems and their software), allowing the single source of truth to become more and more complete. Any existing knowledge that resides in, e.g., code bases, execution logs and documents, should be (semi-)automatically extracted by means of digital assistants, based on behavioral learning, static analysis, and (generative) AI. Digital assistants are needed in this consolidation process to avoid models from going out of date, since development artifacts (source code, documentation, execution logs, etc.) may be continuously evolving.

Once system knowledge is consolidated in models, these models can be queried/consulted, e.g., by means of visualization, or prompting in the context of generative AI, and documentation artifacts can be generated from them. The queried information and generated artifacts are consistent by construction, as well as up to date: artifacts can be instantly regenerated after having changed the system, which increases productivity.

Point 2. Having queryable models and consistent artifacts helps (new) team members to get a better understanding of the system in less time and to maintain the system in a smart way, e.g., by help of generative AI, leading to cost-effective quality improvements.

Additionally, models enable early V&V to find defects and software bugs early, leading to significantly reduced cost. Early V&V leads to short feedback cycles and helps to effectively manage system complexity, e.g., in case many system variants exist. Over time, when the models become more and more complete, the required V&V capabilities to handle these increasingly richer models grow naturally along. To make V&V as well as knowledge extraction efficient and effective, the various techniques for them (e.g., simulation, model checking, theorem proving, model learning, static analysis, ...) are combined to make use of their complementary strengths.

Point 3. Only when there is sufficient understanding of a system, it can be changed with confidence. To effectively change the (legacy) implementations of existing systems, (semi-)automatic massive code transformations are used. These code transformations are specified using suitably expressive specification formalisms that make it intuitively clear what is being changed as well as how, and make it easy to express the transformation. The formalisms for specifying code transformations should be compatible with the V&V capabilities as explained for Point 2, to be able to prove the correctness of the transformations up-front, and therewith reduce the testing effort.

Furthermore, techniques for comprehensive relational analysis are used to be able to compare various design and implementation artifacts with each other, to validate their consistency. For example, execution logs obtained from different software versions could be compared with each other to see if there are any changes between them, and if so, to assess whether these changes are expected or not. This leads to techniques for change impact analysis to reduce the risk of introducing regression while changing the system, manually or automatically. Change impact analysis provides computer assistance to indicate the impact of local changes to the global system behavior.

To further improve efficiency and reduce the risk, implementation artifacts are generated from the models (when they have become sufficiently complete). These artifacts can, like documentation artifacts, instantly be re-generated after having changed the models.

Finally, computer-aided evolution will help developers maintain and evolve their models, by having digital assistance for determining where particular changes are needed to account for new or changing requirements, without breaking unchanged parts of the system. This helps

to apply changes made to one system/product variant, e.g., to fix a security issue, to also be applied to all other system variants. Automatic design/model synthesis is used to (re)compute the parts of the design based on (changes to) requirements, leading to correct-by-construction designs, and later implementations, so that verification may only be needed for parts/aspects not covered by the specified requirements for synthesis.

3.1 Research questions

The challenge is to improve this situation, leading to various research questions, for example:

- With respect to system knowledge consolidation:
  - How can model-driven development be gradually introduced in the ways of working of the high-tech industry, in a way to consolidate both new and existing knowledge as a single source of truth?
  - What are suitable modeling formalisms for consolidating multidisciplinary knowledge of diverse systems?
  - How can knowledge efficiently be extracted from existing development artifacts into models that are consistent with one another?
  - How should consolidated knowledge be organized, presented, and interfaced to keep it accessible and understandable by engineers?
  - How does model-driven development compare with respect to traditional document-based ways of working, in terms of gained efficiency and saved effort?
  - What (combinations of) modeling formalisms should be used to capture all technical diversity of a product, so that the complexity of these models stays manageable, i.e., scales well with the amount of diversity that is included in the models? And how can these models help to reduce the diversity?
- With respect to cost-effective quality engineering:
  - How should engineers work together with digital assistants to reduce the length of feedback cycles on designs and enable fluid and rapid development?
  - How effective is early V&V in finding design problems early, and what is the return on investment?
  - Which V&V techniques should be applied in which situations, cost-effectively?
  - How can different V&V/analysis/learning techniques be composed/combined to complement each other, to be able to scale to large industrial systems?
  - How to efficiently and effectively test changes to systems when those systems are manufactured and installed in a large number of variants (both due to system diversity and system evolution)?
- With respect to efficient risk-free system evolution:
  - How can massive industrial code bases automatically be transformed in a way that is trusted by engineers?
  - How can different design, implementation, and execution artifacts efficiently be compared with each other to determine their consistency?
  - How can actionable insights be determined and presented to engineers, based on observed/detected differences between different versions of a system?
  - When a change to one system component breaks another system component, how can that component be automatically adapted/repaired/diagnosed with respect to that change?

  - How does synthesis-based engineering compare to document-based and model-based ways of working, with respect to additional gains in efficiency and quality?

These research questions are also covered by the research topics described in the recent Systems Architecting and Systems Engineering vision document [8]. For example:

- With respect to system knowledge consolidation, the research topic “Methodologies that interrelate and integrate Software Engineering into (Model-Based) Systems Engineering” is included.
- With respect to cost-effective quality engineering, the research topic “Novel methodologies, models, and tool support, for effective V&V for a wide variety of product variants and specific customer configurations” is included.
- With respect to risk-free system evolution, the research topic “Methodologies for lifecycle management of SW-defined systems to frequently update and upgrade systems while ensuring minimal disruptions to ongoing operations” is included.

4 Related Work

We briefly summarize the vision and roadmaps of CMU, SERC, and INCOSE, and relate them to our vision and approach.

4.1 CMU

The agenda for software engineering research & development of the CMU describes a vision "where humans and AI are trustworthy collaborators that rapidly evolve systems based on programmer intent" [10]. Figure 4.1 shows the research lines leading to this vision. A few research focus areas that are related to the ESI's research agenda on System Evolution and Diversity are:

- *Assuring continuously evolving software systems* which focuses on how to create an evolvable assurance argument, which should only require incremental changes as incremental changes are made to a system. This includes, for instance, combining different types of evidence, assured composition, reassuring evolving systems, and bounding propagation.
- *AI-augmented software development* which aims at using maturing AI techniques to augment human decision making in software engineering and to enable learning from the vast amount of software engineering data. This includes automated code repair, eliminating the design/code conformance gap, multi-artifact system analysis, and automated evolution and refactoring.
- *Software construction through compositional correctness* recognizing that the only viable way of developing and evolving systems will be through technologies that enable compositional development. Among the mentioned topics for research to address platform complexity is compositional correctness via Model-Driven Engineering (MDE) tools, covering modeling languages, transformation, artifact generation, analysis, and synthesis.
- *Engineering AI-enabled software systems* focusing on the challenge of handling the uncertainty that AI components bring to a system. This includes, for instance, techniques to analyze and manage change, reliability in AI-enabled systems, and monitoring and self-adaptation.

These research areas are consistent with our vision and approach. In particular, the research area *Assuring continuously evolving software systems* is essentially about how to ensure that systems remains correct while they evolve (and diversify), which is covered by Point 3 in our approach. The research area *AI-augmented software development* addresses the use of AI to improve the efficiency and effectiveness of software engineers, which is consistent with Points 1 and 3 in our approach. With respect to the research area *Software construction through compositional correctness*; in our approach the construction of systems and their software is heavily guided by models forming a single source of truth. Such a single source of truth may consist of various models, possibly made by various different disciplines, that together form a consistent whole and hence must be composable. Finally, the research area *Engineering AI-enabled software systems* addresses the uncertainty that AI can bring to a system, in a broad sense. Point 2 in our approach addresses the quality and correctness of designs and implementations of evolving systems, and is therewith related.

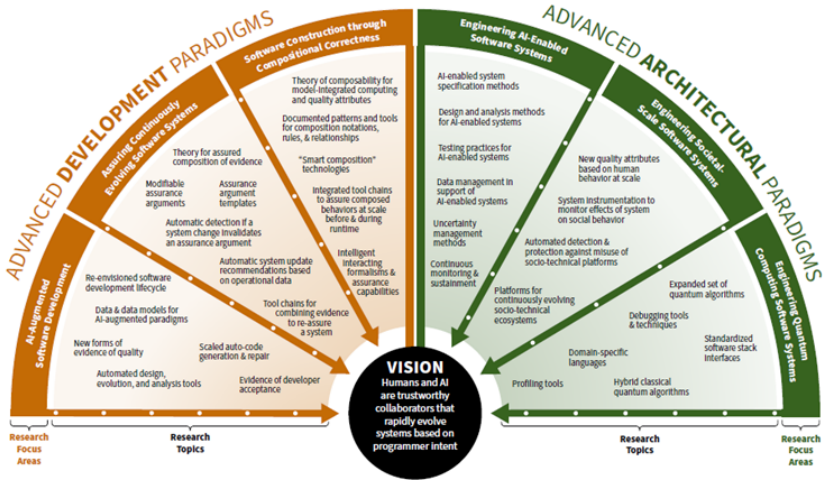

Figure 4.1: A picture of the CMU vision [10].

4.2 SERC

The SERC roadmaps [16] mention three mission areas:

- *Velocity*: support emergent and evolving mission objectives. This includes agile development, commercial DevOps, composability, continuous V&V, instrument development, simulation-based development, and value-based acquisition and development.
- *Security*: demonstrable ability to safeguard critical technologies and mission capabilities. This includes (system-aware) modeling formalisms, and tools to assure that system behavior is trustworthy, e.g., model-based simulation and testing.
- *Artificial Intelligence and Autonomy*: understand, exploit and accelerate the use of AI and autonomy in critical capabilities. The AI/Autonomy mission area aims at human-machine co-learning (with, for instance, augmented engineering) and includes the development of robust and evolvable AI and machine learning algorithms and techniques whose outcomes are explainable, constructing/learning suitable models from various data sources, and using human AI and machine learning teaming to improve various engineering tasks.

The missions are enabled by Digital Engineering: the transformation of the Systems Engineering discipline from document based methods and artifacts to linked digital data and models (see also [18]). The Digital Engineering research roadmap aligns with the five goals of the DoD sponsor's strategy:

1. Formalize the development, integration, and use of models to inform enterprise and program decision-making; includes taxonomies, ontologies, automated decision framework, high fidelity models, and semantic rules.
2. The Authoritative Source of Truth, which includes a collaboration framework, data integration/interoperability, and digital twin automation.
3. Technological Innovation, which includes semantic web technology data exchange, AI and machine learning, life cycle reasoning, and data/information that is seamlessly updated/exchanged continuously in "real-time" cutting across the entire enterprise.
4. Collaborative Environments, which includes development of dashboards to visualize multi-parametric information, change management and model management, authoritative data identification, and ubiquitous computing.

5. Workforce and Cultural evolution, including tool training, process & methods, and digital assistants.

The SERC vision of digital engineering—going from document-based methods to model-based methods and linked data—is consistent with our vision of consolidating knowledge in models forming a single source of truth.

4.3 INCOSE

The INCOSE Systems Engineering Vision 2035 [9] identifies a number of global megatrends that shape the systems of the future. For instance, the importance of environmental sustainability, the interconnected world, digital transformation, and smart systems. By 2035, a wide variety of new and emerging practices will be adapted, such as:

- The future of systems engineering being model based. Systems are specified, designed, analyzed, and verified using the next generation of modeling, simulation, and visualization environments. With these next-generation model-based capabilities, system engineers can explore alternative designs faster by an order of magnitude compared to the effort to analyze a single design today.
- Artificial Intelligence to assist system engineers in making significant changes and delivering solutions more efficiently and effectively.
- The use of data science as an integrated analytic tool to help system engineers to comprehend large-scale data sets needed to assess complex systems.
- Human-systems integration for managing the increasing levels of complexity in designing smart and autonomous systems that interact with humans.

The practice of using models to develop and maintain evolving systems, and the use of digital assistants to manage the models in the context of the whole system lifecycle, is consistent with our vision and approach.

5 Conclusion

This document describes the TNO-ESI roadmap and research agenda for managing the evolution and diversity of increasingly complex systems. This research agenda is in line with [1] and consistent with the visions and roadmaps of CMU, SERC and INCOSE, where the importance of dealing with system evolution and diversity is also stressed. With this document, we expect to be able to formulate a stronger vision for (1) the development of the research program of TNO-ESI on methodologies for managing the complexity of diverse, evolving systems, and (2) for applying these methodologies in industry.

Acknowledgments: The research is carried out under the responsibility of TNO-ESI. The research is supported by the Netherlands Organisation for Applied Scientific Research TNO.

References

[1] TNO-ESI. *Systems Engineering voor Hightech Systems, Strategisch Programmaplan 2024-2027, TKI HTSM*. TNO-ESI document nummer 2023-10109. 2023. URL: https://hollandhightech.nl/programma-s-en-projecten/strategische-programmas/systems-engineering-voor-hightech-systems (visited on 12/06/2024).

[2] Pallab Saha. *Handbook of Enterprise Systems Architecture in Practice*. 2007. ISBN: 9781599041919. DOI: 10.4018/978-1-59904-189-6.

[3] Jilles van Gurp, Jan Bosch, and Mikael Svahnberg. "On the Notion of Variability in Software Product Lines". In: *Proceedings Working IEEE/IFIP Conference on Software Architecture*. 2001, pp. 45–54. DOI: 10.1109/WICSA.2001.948406.

[4] Matthias Galster et al. "Variability in Software Systems—A Systematic Literature Review". In: *IEEE Transactions on Software Engineering* 40.3 (2014), pp. 282–306. DOI: 10.1109/TSE.2013.56.

[5] Ina Schaefer et al. "Software diversity: State of the art and perspectives". In: *International Journal on Software Tools for Technology Transfer* 14 (2012), pp. 477–495. DOI: 10.1007/s10009-012-0253-y.

[6] Piërre van de Laar and Teade Punter. *Views on Evolvability of Embedded Systems*. Springer, 2011. DOI: https://doi.org/10.1007/978-90-481-9849-8.

[7] Piërre van de Laar and Teun Hendriks. "A retrospective analysis of Teletext: An interoperability standard evolving already over 30 years". In: *Advanced Engineering Informatics* 26.3 (2012), pp. 516–528. ISSN: 1474-0346. DOI: https://doi.org/10.1016/j.aei.2012.04.007.

[8] Teun Hendriks and Sezen Acur. *Vision and Outlook for Systems Architecting and Systems Engineering in the High-Tech Equipment Industry*. TNO report number 2024 R10542. 2024. URL: https://resolver.tno.nl/uuid:91ca077f-403c-4589-9d04-da0695f463e6 (visited on 12/17/2024).

[9] INCOSE. *Systems Engineering Vision 2035*. 2021. URL: https://www.incose.org/about-systems-engineering/se-vision-2035 (visited on 04/09/2024).

[10] Anita Carleton et al. *Architecting the Future of Software Engineering: A National Agenda for Software Engineering Research & Development*. 2021. URL: https://insights.sei.cmu.edu/library/architecting-the-future-of-software-engineering-a-national-agenda-for-software-engineering-research-development (visited on 04/09/2024).

[11] ITEA 4. *ITEA 4 Strategic vision and technology roadmap*. 2021. URL: https://itea4.org/publication/download/itea-4-strategic-vision-and-technology-roadmap.pdf (visited on 04/09/2024).

[12] Ulrich A. W. Tetzlaff. "Flexible manufacturing systems". In: *Optimal Design of Flexible Manufacturing Systems*. 1990, pp. 5–11. DOI: 10.1007/978-3-642-50317-7_2.

[13] Gartner Research. *Application Modernization Should Be Business-Centric, Continuous and Multiplatform*. 2019. URL: https://www.gartner.com/en/documents/3956295 (visited on 04/09/2024).

[14] Galorath. *Software Maintenance Cost*. URL: https://galorath.com/software-maintenance-costs (visited on 04/09/2024).

[15] Jussi Koskinen. *Software Maintenance Costs*. 2010. URL: https://web.archive.org/web/20120313070806/http://users.jyu.fi/~koskinen/smcosts.htm (visited on 04/09/2024).

[16] SERC. *Research Roadmaps 2019–2020*. 2019. URL: https://sercuarc.org/wp-content/uploads/2021/08/ROADMAPS_3.5.pdf (visited on 04/09/2024).

[17] Jacco Wesselius et al. *MBSE in the High-Tech Equipment Industry*. TNO report number 2022 R11504. 2022. URL: https://resolver.tno.nl/uuid:aad3da36-9766-4e64-9214-bdf2e014dc2f (visited on 12/10/2024).

[18] Tom McDermott et al. *Task Order WRT-1001: Digital Engineering Metrics*. Technical Report. SERC-2020-TR-002. Systems Engineering Research Center (SERC), 2020. URL: https://sercuarc.org/wp-content/uploads/2020/06/SERC-TR-2020-002-DE-Metrics-6-8-2020.pdf (visited on 04/09/2024).

TNO-ESI

High Tech Campus 25
5656 AE Eindhoven
www.tno.nl