Editor: **Markus Borg**
CodeScene
markus.borg@codescene.com

The Magazine at 40: Viewing Requirements Engineering Through a Ruby Lens

Kristian Sandahl, Björn Regnell, and Markus Borg

From the Editor

In this final issue of the year, we mark a milestone: IEEE Software's 40th anniversary—a ruby jubilee! While this column still awaits the silver achievement badge (25 years)—Suzanne Robertsson founded it in the early millennium—I'm pleased to present a discussion with two seasoned professors of requirements engineering. They will share their personal reflections on the discipline over the past four decades. I'm particularly pleased to feature a retrospective by the new retiree, Prof. Sandahl. Naturally, we conclude with an outlook on what is to come. Long live the new quadragenarian!—*Markus Borg*

FORTY YEARS IS a fascinating time span. For me, Markus (MB), it's roughly the time since my birth. As a racket sports player, turning 40 means I can now compete in the senior category. Were I a car, I would qualify for historic vehicle registration in several countries. As a human, however, I start receiving invitations to regular health screenings.

Traditionally, 40 years marks the typical length of a professional career. The period is lengthy enough to encompass many significant events. Yet, it remains comprehensible enough for reflection—making it an ideal moment to do so!

Digital Object Identifier 10.1109/MS.2024.3429774
Date of current version: 10 October 2024

The Formative Years

MB: Kristian, you've been involved in the requirements engineering (RE) discipline since its inception. Please tell us about the very early days.

KRISTIAN SANDAHL (KS): My pleasure. During the 1970s and 1980s—when the term RE was coined—the focus was on creating and maintaining functional, written, paper-based specifications. At that time, templates for specifications were cutting-edge. A textbook featuring usable templates was at least twice as expensive as other software engineering books.

I vividly remember when the first edition of the *IEEE Guide for Software Requirements Specifications* (IEEE 830) was approved by the board in September 1983. Around the same period, large U.S. Government agencies began transitioning to electronic documentation with SGML, a precursor of XML.

DISCUSSANTS

KRISTIAN SANDAHL. Professor of software engineering and head of the Programming Environments Laboratory at Linköping University, Sweden. Just announced his retirement—an ideal time for a retrospective! Affiliated with Ericsson Research from 1995–2001, during which time he mentored a young Ph.D. degree student (Björn) during an internship.

BJÖRN REGNELL. Professor of software engineering at Lund University, Sweden. Over 30 years of experience in software engineering with a focus on empirical research in requirements engineering. Steering Committee Chair of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ) 2013–2017.

MB: The time of strong belief in structure, right?

KS: Absolutely. However, the field progressed from templates and structured text to more expressive models. The late 1980s witnessed the advent of the use case approach within Objectory, which evolved into the Unified Modeling Language (UML). The ideal of that time was a seamless design starting from use cases and then identifying classes. Finally, more detailed sequence diagrams, and perhaps state-machine diagrams for detailed behavior.

The intent was to enable iterative refinement into design and implementation models. Thus, ensuring traceability from requirements to implementation. Traceability had, up until then, been awkwardly handled by large matrices linking requirements with design elements. The introduction of use cases was often helpful, particularly because they were understandable by many stakeholders, not just IT specialists. Still, in many cases, a huge semantic gap remained between analysis models and design, especially when the design involved reusing existing software.

The RE Heydays

MB: This was still before the academic conferences?

KS: True, but then the very interesting 1990s followed. We saw the inauguration of the three conferences RE, ICRE, and REFSQ. This triggered an exciting influx of ideas from various fields.

MB: Which other fields do you think influenced RE the most in this period?

KS: I would highlight three areas. First, information systems (IS) experts had been analyzing and designing organizational business processes since the 1960s. Their job was to provide requirements and a system architecture which the programmers then implemented. The IS influence also contributed more qualitative research methods into RE. On the other hand, software engineering professionals traditionally began with the requirements and worked toward designing the engineering processes. It is interesting to note that these two professions happily met in the requirements.

Second, we experienced an increase in adopting techniques from artificial intelligence in the 1990s. In contrast to today's hype, this was caused by that decade's AI winter. Some AI researchers specializing in knowledge acquisition found themselves useful in enhancing requirements elicitation. For instance, Hudlicka discussed the common challenges in both knowledge and requirements elicitation when interviewees struggle to articulate their thoughts clearly.[1] and Natural language processing emerged as another AI application to help analyze textual requirements, improve traceability, and automatically classify requirements.

Third, formal methods became computationally tractable and gained popularity in safety-critical applications. Through logical inference, they could ensure consistency and goal fulfillment. As software inspections became widespread in the industry, these methods were increasingly used as knowledge-based methods for requirements validation.

MB: How did you share this growing body of knowledge with students?

KS: As the field matured, books appeared that I used in university teaching. *The Sommerville and Sawyer* textbook from 1997 summarized the state of practice.[2] A single book cannot cover everything, but this came close. I like how it discussed the pros and cons of various RE practices to guide the engineers. Requirement management with different techniques for prioritization and classification became prominent.

Another seminal book was *Nonfunctional Requirements (NFR) in Software Engineering* by Chung et al.[3] NFRs had really been an underappreciated area of RE but became

a central topic during the early 2000s. The book was a landmark, exploring tradeoffs among different system quality factors. It was also instrumental in integrating input from usability engineering and other quality-focused disciplines into requirements specifications. Other fields then saw RE as the vehicle to make a change—with explicit requirements, the programmers could not ignore them in subsequent steps. At the same time, requirements reuse for quality, for example security, emerged as an interesting market.

The Agile Transformation

MB: What about agile? This must have disrupted the RE community too.

KS: Iterative methods to gradually reach the project target slowly became a standard, even in large-scale systems development. This introduced challenges regarding which requirements to schedule for each release. Additionally, we understood that organizations need to balance investment in detailing requirements against the risk of them being scrapped or changed during release planning.

I believe RE practice in the first 15 years of the new millennium was quite fragmented. On one side were the enthusiastic proponents of agile methods, that defied planning in advantage of implementation. Short user stories came into fashion for handling prioritization, scheduling, and different stakeholder interests simultaneously. A list of backlog items posed the requirements specification. Unfortunately, this approach often resulted in the loss of an overview, dependencies, and nonfunctional properties of the system. Trivializing requirements sometimes worked, but sadly, our understanding of how to express and work with the desired capabilities and qualities of the system under development didn't progress.

BJÖRN REGNELL (BR): I can't help but weigh in here. Agile was a backlash for RE in many organizations, with Scrum new-speak essentially renaming all concepts. But things have luckily improved. In fact, after agile, DevOps, and cross-functional CI teams, the RE work is often more decentralized to the developers. A flat organization may not have the CTO staff to do centralized RE, and it is pushed out to the platform, service, and app teams. Now developers must do more core RE work.

MB: And we all know the challenges of scaling agile. What else can you share about RE in large enterprises?

KS: Large companies have invested in different requirements tools to manage the vast amounts of information. Defining increasingly complex systems at different abstraction levels is no walk in the park. Tools like DOORS excel at generating parts of the requirements specifications and facilitating follow-ups. But for this to work, practitioners must spend the effort to make the requirements atomic—which means sacrificing sight of their dependencies. Traceability remains an issue and will be for long. Automated methods can help find approximate traces, but precise modeling is necessary to fully understand the dependencies involved.

In parallel, modeling got a new boom with the standardization of UML 2.x and later SysML in 2006. Interestingly, SysML even recognizes a requirement as its own model element. This enabled a clearer depiction of dependencies between requirements and other artifacts like tests. Yet, visualizing more than 100 requirements on a screen remains difficult. Some have turned to graphical databases to store models and generate needed queries, though these databases can be too slow for some needs. But sometimes SysML can provide substantial value in RE—for example using it to build simulation models of cyberphysical systems to identify failed requirements.

MB: Any comments about the last 10 years?

KS: Data-driven approaches have been very intriguing over the past decade. Analyzing operational data can provide insights for new or adjusted requirements. Some DevOps advocates even push to automate this feedback loop from operation back to requirements definition.

Additionally, I must mention the recent surge in large language models. It has sparked hope in generating requirements neatly from diverse and unstructured information. An interesting challenge for the future is to make the generated requirements interpretable by human decision-makers so that they can coexist with traditionally developed requirements.

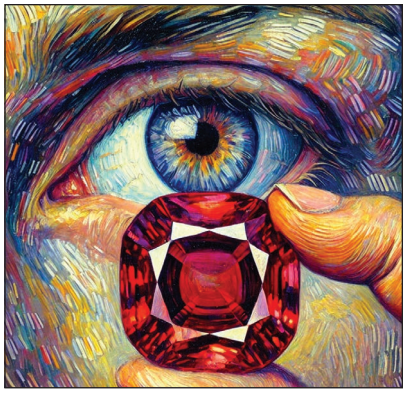

FIGURE 1. Viewing requirements engineering through a ruby lens.

The Current Landscape

MB: Björn, what are your thoughts on the current AI hype that seems to have touched every aspect of technology?

BR: I think, with the AI-boosted software engineering, we're finally on the verge of a significant shift. We're moving away from the tedious, nitty-gritty coding tasks to focus more on the *real* engineering work—eliciting intentions, focusing on creativity and imagination, and carving down relevant priorities. Also the really hard thing: quality requirement tradeoffs and interpreting user feedback.

All of these tasks are part of core RE. A software engineer in dialog with a reliable large language model can soon be much more effective in creating feasible technical descriptions that *actually* correspond to our *real* intentions. Hopefully leaving subpar IT systems behind. Ideally, we can finally concentrate on the *essential* complexity of the domain and get support in focusing on our dialectic interaction between imagination and implementation with our goals in mind.

However, this doesn't imply that deep coding knowledge is becoming obsolete. Programmers will still need to develop innovative and complex business logic and critically review the sometimes delusional AI-generated code—an opaque result of training on yesterday's code.

Honestly, we didn't see this AI capability coming when we did a similar reflection[4] and crystal ball outlook[5] when REFSQ had its 10-year anniversary. We largely missed the current AI boom.

Squinting at the Contours

MB: What surprises and key trends in RE have stood out to you?

> Short user stories came into fashion for handling prioritization, scheduling, and different stakeholder interests simultaneously.

BR: I remember being astonished by the complexity of real-world RE during my internship with Kristian. That complexity still astounds me, but I'm excited to see the new tools that are up for beating this challenge.

When I squint at the history of RE, I see three major evolution steps, or "eras" if you like:

1. the narrow focus on the monolithic specification
2. the broader perspective: empirical studies of, and solutions for, the messy real world
3. smart, open, post-agile RE

Let me unsquint my eyes and elaborate on this. My reflection is that RE has evolved through the eras as the result of seven focus shifts. We have gone from the following:

- Complete ➔ Economical

Initially, the RE field naïvely emphasized complete, upfront requirements, which often proved cost-ineffective. Spending too much of the scarce RE resources on simple, low-risk requirements simply isn't feasible. Today, good RE practice balances analysis efforts in relation to the benefits of gained domain knowledge and reduced risk of requirements misinterpretation.

- Specification ➔ Prioritization

Over the years, we've realized that focusing solely on monolithic specifications with "shall" statements is far from enough. Requirements need to include rationale and priorities while capturing the viewpoints of many stakeholders. Contemporary RE focuses on facilitating decision-making and navigating complex tradeoffs. Additionally, we have shifted from a sole functional specification focus to doing the hard job of eliciting quality requirements and finding balanced quality targets on slippery quality scales.

- Analytical ➔ Empirical

In the early 1990s, many RE papers presented solutions based on analytical arguments, often without any empirical evidence whatsoever. Later, the RE field became a center point of gravity for the empirical revolution of software engineering research methodology. Today, all major academic RE forums enforce mature criteria for reviewing empirical papers.

- Toy examples ➔ Industrial cases

In line with the shift from analytical to empirical methods, RE research has also shifted from toy examples to real-world, large-scale industrial cases. Many academic RE researchers have realized that the most interesting research questions

require close collaboration with industrial partners.

- Proactive ➔ Reactive

The agile overtake was, in many respects, a backlash to RE practice. A crude interpretation of agile is that requirements are not needed, promoting immediate coding to see what emerges. But a purely reactive mode lands in a step-wise walk out in the desert of technical debt and irrelevant products. Time and again, it has been proven that understanding the underlying intentions is absolutely key to the development of useful and valuable software. Nowadays, successful companies recognize this and use prototyping, A/B-testing, and feedback from users as inputs to continuous learning in a proactive manner.

- Closed ➔ Open

Looking back to the early days of RE, most software was bespoke, and most requirements were part of contracted closed-source development. Since then, open source software has revolutionized how we share the planet's software engineering brains and infrastructure. Now, democratic and meritocratic governance of software commons is essential for much of our critical software infrastructure.

- Manual ➔ Smart

Already in the late 1990s, we started to realize that effective RE work needs smart tooling. Natural language processing was put into work in RE tasks such as finding similar requirements or detecting ambiguities. Now, with AI-enhanced RE and large language models, we are arriving at the potential of very effective RE bots that can support us in improving RE models, understanding users' needs, and assessing the implications of quality tradeoffs.

ABOUT THE AUTHORS

KRISTIAN SANDAHL is a professor at the Department of Computer and Information Science, Linköping University, 583 30 Linköping, Sweden. Contact him at kristian.sandahl@liu.se.

BJÖRN REGNELL is a professor at the Department of Computer Science, Lund University, 223 63 Lund. Sweden. Contact him at bjorn.regnell@cs.lth.se.

MARKUS BORG is a principal researcher at CodeScene, 215 32 Malmö, Sweden, and an adjunct associate professor at the Department of Computer Science, Lund University, Sweden. Contact him at markus.borg@codescene.com.

The Road Ahead

MB: Based on the current discussion, we must surely speculate about what the RE future might hold.

KS: It has been a privilege to be part of RE's development over the years. RE, both in research and practice, is the natural meeting point for many stakeholders and will play a crucial role in developing sustainable systems for the future. I call for more interdisciplinarity going forward. RE has been driven by IT-related communities. We need way more contributions from domain experts in other disciplines, such as chemistry or social science.

BR: On that note, I believe in opening RE activities to the user communities. They are often the real experts. Many organizations now dare to communicate transparently with user communities, involving users as a valuable resource in RE.

Of course, there is also the continued *AIfication* of RE. This will enable smart and rapid RE. And, vice versa, we need more work on RE for future AI systems. And research on RE for future AI-boosted software engineering.

KS: Yes, the data-driven solutions we will build into future software products are indeed promising. The underlying data quality will become the new arena for product realization. Still, we cannot overdo data manipulation. If we start spending too much time fiddling with the data, we'll find ourselves back at programming, but in the language of data.

It surprises me that formal methods haven't been more widely adopted, despite the prevalent education in relevant mathematical and theoretical concepts among today's students.

Practically, the choice of representation should suit the recipients of the requirements, whether they are directly developing the system, overseeing its development, or automating its generation.

MB: Wow, that was quite the reflection. On behalf of the readers of the Requirements column, I say thank you very much! I'll encourage those interested to reach out to you directly if they want to continue the discussion. Let's hope the RE dialogue continues until *IEEE Software's* golden jubilee in 2034!

References

1. E. Hudlicka, "Requirements elicitation with indirect knowledge elicitation techniques: Comparison of three methods," in *Proc. 2nd Int. Conf. Requirements Eng.*, Colorado Springs, CO, USA, 1996, pp. 4–11, doi: 10.1109/ICRE.1996.491424.
2. I. Sommerville and P. Sawyer, *Requirements Engineering: A Good Practice Guide*. Hoboken, NJ, USA: Wiley, 1997.
3. L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Berlin, Germany: Springer-Verlag, 2000.
4. V. Gervasi, E. Kamsties, B. Regnell, and C. B. Achour-Salinesi, "Ten years of REFSQ: A quantitative analysis," in *Proc. 10th Int. Workshop Requirements Eng. Found. Softw. Qual.*, 2004, pp. 55–72.
5. A. L. Opdahl, E. Dubois, and K. Pohl, "Ten years of REFSQ: Outcomes and outlooks," in *Proc. 10th Int. Workshop Requirements Eng. Found. Softw. Qual.*, 2004, pp. 73–93.

Digital Object Identifier 10.1109/MS.2024.3459128