Zeitschrift: Ferrum : Nachrichten aus der Eisenbibliothek, Stiftung der Georg Fischer AG

Herausgeber: Eisenbibliothek

Band: 93 (2024)

Artikel: The silver bullet, or how to kill the quality "beast"

Autor: Leimbach, Timo

DOI: https://doi.org/10.5169/seals-1061986

The silver bullet, or how to kill the quality "beast"

Timo Leimbach

Software projects are often examples of projects that fail to meet quality, time, and cost constraints. Despite substantial efforts to enhance the development process, there is no simple solution. Improving the process has remained a pivotal question and sparked heated debates. It covers a broad set of problems and solutions ranging from the unique attributes of software and the formal correctness of code to different approaches in project management. Originally, emphasis was placed on detailed specification and rigorous upfront planning, known as the waterfall model, while in recent years alternatives focused on incremental/iterative concepts, now called agile methodologies. For both, quality and its different conceptualizations have played an important role.

Information technology (IT) projects, particularly those within software development, have gained a notorious reputation for consistently failing to meet quality, time, and cost constraints. Even during the early days of computing, numerous examples existed where software failures had serious consequences, impacting not only quality but also safety.

Drawing on a variety of metaphors, F. Brooks from IBM, who headed the development of the operating system for the famous S/360, brought this issue to a wider audience. In explaining the delays and cost overruns of software projects he compared them to "tar pits". Caught in these mires with no progress in sight, people search in vain for a simple, straightforward solution that solves all the problems, the mythical "silver bullet" capable of slaying the monster with one shot.[1]

From hard to soft – the evolution of a problem

Despite substantial efforts to enhance the development process, the underlying issues have persisted and, in some cases, even exacerbated over time. In the 1990s the infamous CHAOS Report by Standish Group highlighted the fact that more than two-thirds of all IT projects failed to meet at least one of the three critical dimensions – cost, quality, and time.[2] Although there has been some improvement in recent years, these challenges have not disappeared entirely. The question of how to enhance the software development process remains pivotal. Various approaches have been proposed, leading to heated debates within the field. But as organizations are still in need

of better outcomes, finding effective solutions continues to be a priority.

One of the central challenges when discussing how to enhance IT projects lies in the dynamic nature of the problem itself. During the nascent years of computing, attention primarily focused on the hardware limitations of computer systems, for example machines like the IBM 650 and their counterparts. These constraints – such as limited memory and computing capacity – necessitated a pragmatic and concise approach by programmers in order to devise workable solutions. However, an important shift occurred during the 1960s with the emergence of a more capable generation of computers, which started to remove the hardware limitations. Conversely, the increasing capabilities created a rising demand for novel functionalities, including multi-user environments and diverse types of applications. This transition marked a departure from the classical, mathematically rooted problem-solving paradigms – such as sorting and optimization – to more complex, continuous operations. Examples include online transactions and management information systems.[3]

As software systems grew in complexity, their manual verification became increasingly challenging.

This shift from batch-oriented processing to real-time, online systems brought about significant changes in software development practices. Firstly with regard to testing and improving: unlike earlier times, programs could no longer be tested and adjusted in the same way. Batch programs, commonly executed one after another on computers, would halt in case of failure. Programmers then faced the task of identifying and resolving issues before rerunning the program until successful. As multi-user systems with concurrent applications gained importance, this approach faced limitations. The second change related to the size of the programs: previously, codebases were small enough to allow manual verification of logic and correctness of the code itself. However, as software systems grew in complexity, this manual task became increasingly challenging.

Given that, the advent of online/real-time systems shifted the very essence of software development. As pointed out already, software development had been associated with two distinct paradigms. The test-based approach opened up in the direction of a more engineering-oriented approach, which involved rigorous testing and refinement. The other one was the artisanal approach, often referred to as the "art of programming", which emphasized craftsmanship. Developers meticulously crafted code, akin to artisans and craftspeople shaping individual pieces. Both models faced different challenges. While the latter was ill-suited to the growing need for programs, the first required more rigorous methods to ensure the results would live up to the desired outcome. Therefore, the emergence of transactional systems necessitated a departure from these existing paradigms.[4]

As computer technology advanced, the limitations imposed by hardware gradually diminished.

This was underlined by the growing number of projects that faced significant challenges. These projects included IBM's well-known and already mentioned OS/360, which experienced delays spanning several years and incurred costs much higher than budgeted for. While this example is widely recognized, numerous other private companies also struggled in their attempts to establish management information systems or similar solutions. Altogether, these experiences raised awareness of a critical issue around software. Nowadays the term "software crisis", which is likely a retrospective label applied to it, is used to describe this period. While from an academic perspective the debate over whether this wording was already used in contemporary discussion remains open, it is obvious that the period was marked by changes.[5] As computer technology advanced, the limitations imposed by hardware gradually diminished. Simultaneously, the challenges associated with software development gained prominence. The roots of these challenges are multifaceted. They range from the inherent intangibility of software, which defied production standards established in other areas, to a shift in focus towards human interaction with the new forms of applications. This rise of the "soft" problems was exemplified by the widely circulated tree swing cartoon during the 1970s.

In summary, the landscape of computer systems continually shifted, requiring new strategies to address the systems' emerging complexities. The resulting problems in particular within software development were significant, prompting among other things the need for new approaches in software development. Acknowledging these historical shifts is crucial to the understanding of further dynamics.

In search of an answer

Not surprisingly, the quest for a solution to the problem emerged in parallel with this shift in computing. Notably, the SAGE (Semi-Automated Ground Environment) radar system marked an important point. Its evolution in the 1950s was a catalyst for the development of new approaches to larger and complex software, primarily referred to as system development since hard- and software

development were closely intertwined. Simultaneously, other ideas gained traction, and the first scientific workshops took up programming-related issues. Altogether, this gave rise to a diverse set of approaches aimed at addressing the same fundamental problem. However, not all of these approaches harmonized with each other. Discrepancies in understandings of the problem and subsequent solutions became evident,[6] for example in the course of the famous NATO conferences on software engineering, held in Garmisch-Partenkirchen in 1968 and Rome in 1969. The idea behind the conferences was to bring together scientists and practitioners from a variety of fields "to shed further light on the many current problems in software engineering, and also to discuss possible techniques, methods and developments which might lead to their solution".[7]

The primary outcome of the conferences was not a solution to the problem as such, but rather an amplified awareness of the challenges at hand.

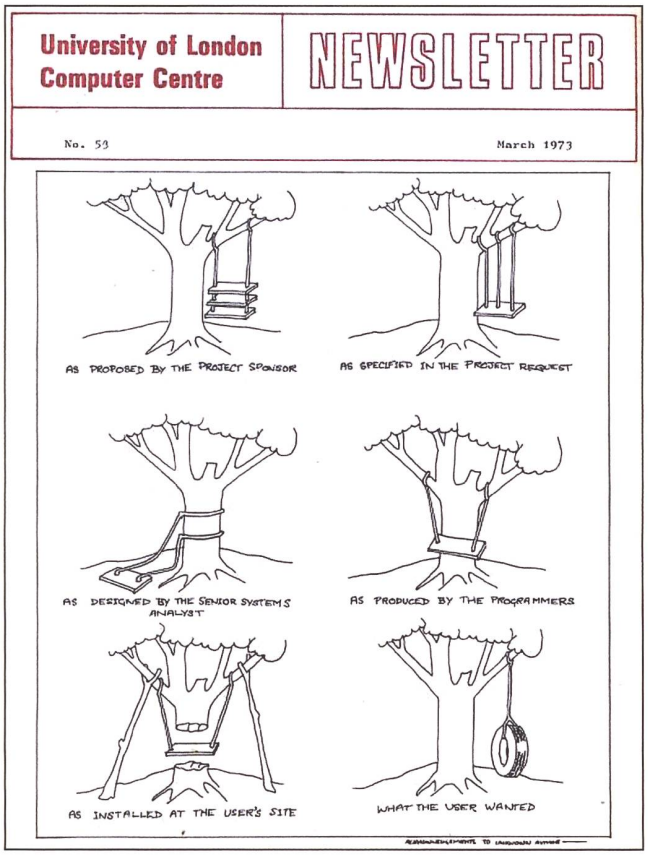
The divergence in approaches and ideas may have been exacerbated by the term "software engineering", which was introduced to mark a difference to terms like programming. Some interpreted this term through the lens of craftsmanship, drawing parallels to the art of programming as promoted by Donald Knuth.[8] From this perspective, practice and talent played pivotal roles. Conversely, a different group, predominantly composed of applied mathematicians, favoured a mathematically grounded approach to formal software verification. This approach demanded rigorous techniques for formal specification, analysis, and development – techniques deeply rooted in theoretical computer science. Another group of practitioners and scientists, including Grace Hopper, engaged in more pragmatic discussions about possible process models, addressing (among other problems) safety and quality.[9] Given the substantial disparities in understanding and potential solutions, the primary outcome of the conferences was not a solution to the problem as such, but rather an amplified awareness of the challenges at hand. This heightened awareness clearly influenced the establishment and evolution of academic disciplines, particularly in Europe. While the field of computer science was already well established within American universities and scientific circles, European researchers still struggled to establish the disciplines in their respective countries. As a consequence, the increased recognition of software development challenges started to play a pivotal role in shaping disciplines such as "Informatik" at German universities. Intriguingly, the aforementioned applied mathematicians played a crucial role in steering the academic field toward a more formal and theoretical trajectory, in contrast for example to the field of computer sciences in the US sparking debates on its direction in the following years.[10]

In the discourse surrounding software development, the concepts of quality and safety have been central points of discussion. However, their precise meanings and implications have often remained ambiguous. The conceptualization of safety was often closely tied to reliability and malfunction prevention. Initially, the focus was primarily on ensuring reliable operations of software systems. However, as software found its way into safety-critical domains (such as aircraft control systems), safety considerations expanded – in particular regarding prevention. While this was only in a limited number of cases at that point, software today pervades many other critical areas, including automotive safety features and healthcare instruments. Consequently, the definition of safety has evolved beyond mere reliability.

Quality, on the other hand, received significant attention in these discussions. Yet, like safety, it remained conceptually hazy. Often, quality was discussed in terms of quality assurance and quality control, which mirrored and reflected the discussion patterns from other disciplines. In this context quality assurance primarily referred to the development process itself. This involves practices that ensure adherence to standards, efficient workflows, and defect prevention. However, the boundaries between quality assurance and quality control can be indistinct, especially in early literature. Quality control on the other hand focuses on assessing the final product. Rigorous testing and validation, like in other engineering disciplines, should ensure product quality for the individual software. Interestingly, software engineering diverges from other engineering fields in its treatment of maintenance. While maintenance is a critical aspect of system longevity, it has received less attention and has become problematic. As software systems evolve, neglecting maintenance can lead to unforeseen issues, compromising both safety and quality.

The emergence of structured IT project management

The evolution of software development practices has been influenced by the analogy to other engineering disciplines, particularly manufacturing. Drawing inspiration from established engineering fields, the practice of software development moved its focus toward more structured approaches, with engineering management playing a pivotal role. In this context, project management has emerged as a critical discipline, addressing the challenges posed by large-scale technology development endeavours. It emerged as a distinct field during the 1950s as a result of the challenges experienced during ambitious, large-scale technology projects in the course of the Cold War arms and technology race. Initially, project management aimed to equip practitioners with the necessary skills and tools to navigate such projects. Consequently, the knowledge base of it had a strong focus on planning and control, especially schedul-

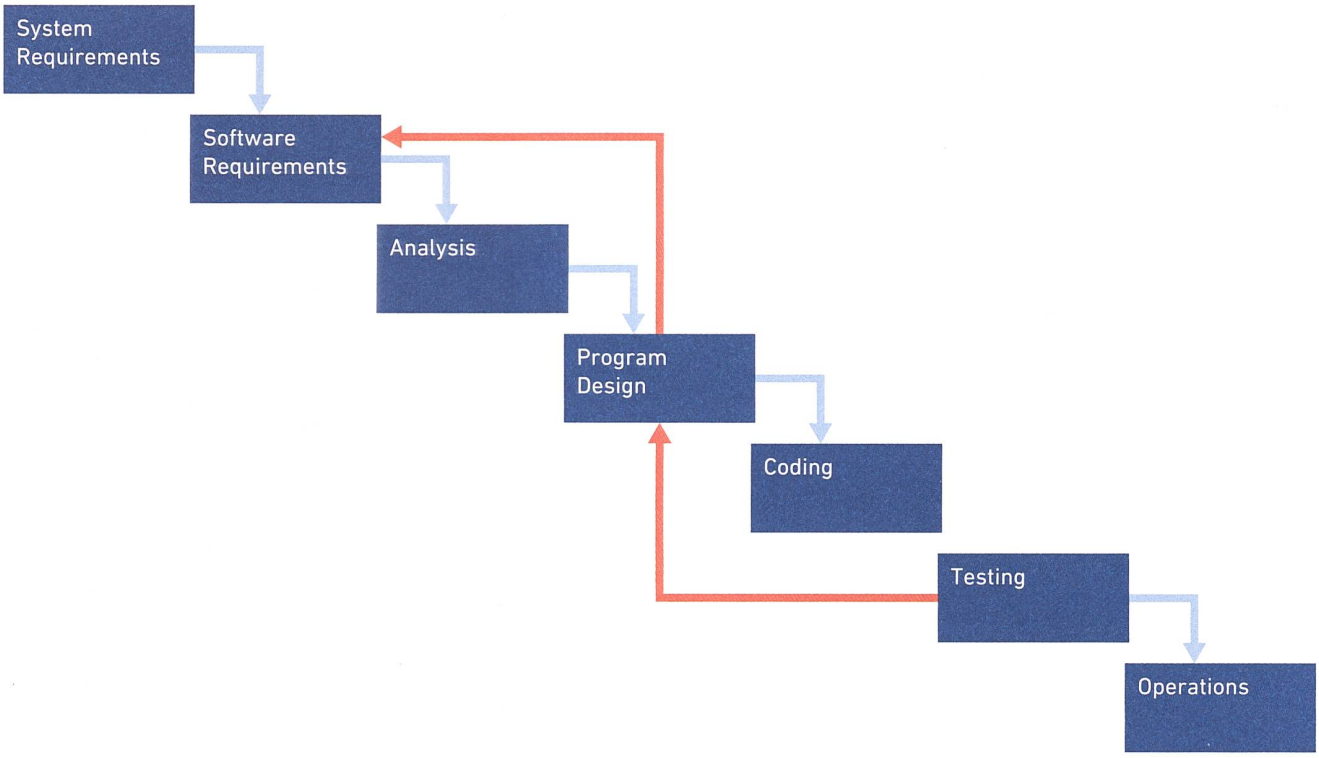


1 Oldest dated version of the tree swing cartoon from the University of London Computing Centre, 1973.

ing, optimization of resource utilization, budgeting and timing, as well as controlling the implementation. Typical tools developed and refined included Gantt charts as help with visual planning, borrowed from engineering or critical path analysis, which facilitates critical task identification and optimized resource planning. The theoretical concepts were often narrow and its underlying assumptions can be traced back to the ideas of Taylorism and scientific management, which evolved in the 1920s and 1930s.[11]

Within software development the idea was not totally new, and structured approaches have a long history. As early as the 1950s, the first examples of so-called "structured programming methods" appeared. Among these, one stands out: the approach that originated within the context of the SAGE (Semi-Automatic Ground Environment) project and was first presented in 1956. This approach places significant emphasis on rigorous upfront planning and detailed specification of requirements before any coding begins – aimed at creating a solid foundation for subsequent implementation. Quality assurance, in this context, revolves around rigorous specifications. These specifications are subject to validation through testing – a practice borrowed from other engineering disciplines. Essentially, quality is perceived as a control feature, ensuring adherence to predefined standards. Notably, the approach described dealt with a program comprising approximately 40 500 instructions. Given this manageable size, rigorous processes could be applied to ensure the software's proper functioning. In a similar vein was the idea of formal methods of verification, which were favoured by parts of the scientific community. The idea here was also to specify requirements rigorously, but instead of checks made by hand it aimed at mathematical checks to ensure correctness. However, these approaches have limitations.[12] While they work well for smaller programs with a few thousand instructions, they become impractical for larger software systems. The proof process becomes prohibitively time-consuming compared to the actual programming effort. Consequently, these methods remained limited to specific domains and were not widely adopted for general-purpose software like modern operating systems, with their millions of lines of code.

The expansion of program sizes presented a persistent challenge for structured approaches as well. Over time, a methodological framework emerged, now widely recognized as the waterfall model. This model prioritized comprehensive specification, thorough requirement gathering and meticulous initial planning. Its formalization can be traced back to a seminal 1970 article by William Royce, often credited as its inventor. Ironically, Royce's intention was to critique and enhance the model rather than advocate its adoption as-is. He highlighted a notable challenge:



2 Waterfall model as described by Royce, red arrows marking the improvements suggested.

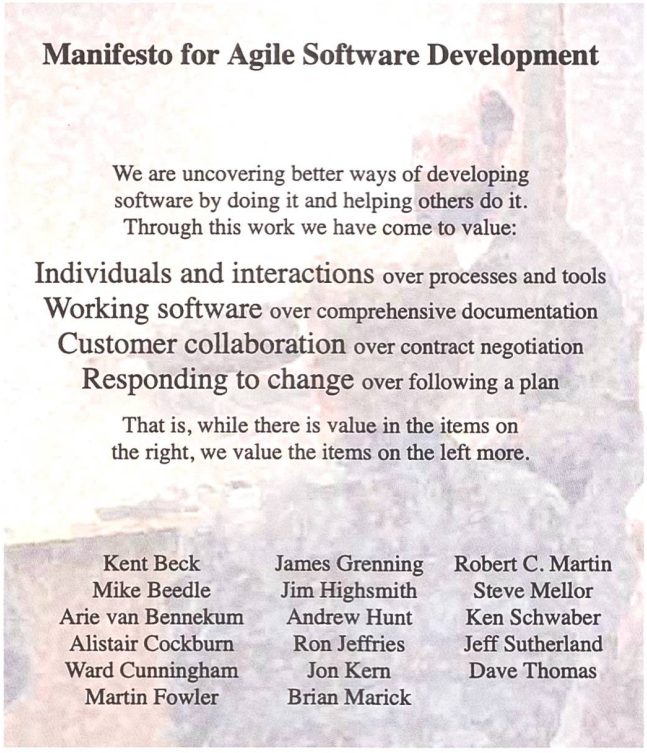
the lack of feedback mechanisms to accommodate alterations and refinements based on insights gained from subsequent stages. Particularly, he advocated for a simplified iterative approach, wherein core functionalities, after an initial development, will be refined based on initial testing. This underscored a pressing challenge in computer system development that could not be adequately addressed through increasingly stringent specifications and upfront planning alone.[13]

The challenge stemmed primarily from the continuous proliferation of computer systems into new domains of application. It became increasingly apparent that the significance of software extended beyond mere functionality – the paramount concern became whether the software fulfilled its intended purpose. Consequently, quality control began to pivot towards a dual emphasis on verification and validation. This conceptualization found prominent expression in the V-model, where the left side signifies verification and the right side validation. This evolution coincided with a burgeoning discourse on the nature of software quality throughout the 1970s.

Many conceptual models, such as Boehm's utility model, predominantly emphasized technical aspects of quality, such as reliability and testability. Aspects like aesthetics and usability were often subsumed under the rubric of human experience. This trajectory mirrors the ongoing deliberations on quality within contemporary engineering, where the imperative of quality steadily escalated, particularly as a means of differentiation in increasingly competitive markets. Nonetheless, the predominant focus remained entrenched in technical dimensions. A gradual shift in this paradigm commenced with the emergence of quality management as a distinct discipline. This transformation coincided with the rise of influential consultants such as Deming, Crosby, and Juran, alongside scholarly investigations led by Garvin and others. These developments led to an increasing emphasis on quality, also in the realm of software production.

> For many within the software industry, the intensified focus on procedures and its attendant requirements for documentation appeared misguided.

Primarily, the rise of quality management systems catalysed efforts to enhance existing methodologies. This led to divergent avenues for integrating process quality into software development. One such avenue was the aforementioned V-model of development, designed to ensure not only verification and validation but also a coherent process model. Another approach involved the introduction of maturity models aimed at assessing the quality of development processes. Additionally, entirely new process-oriented project management frameworks,



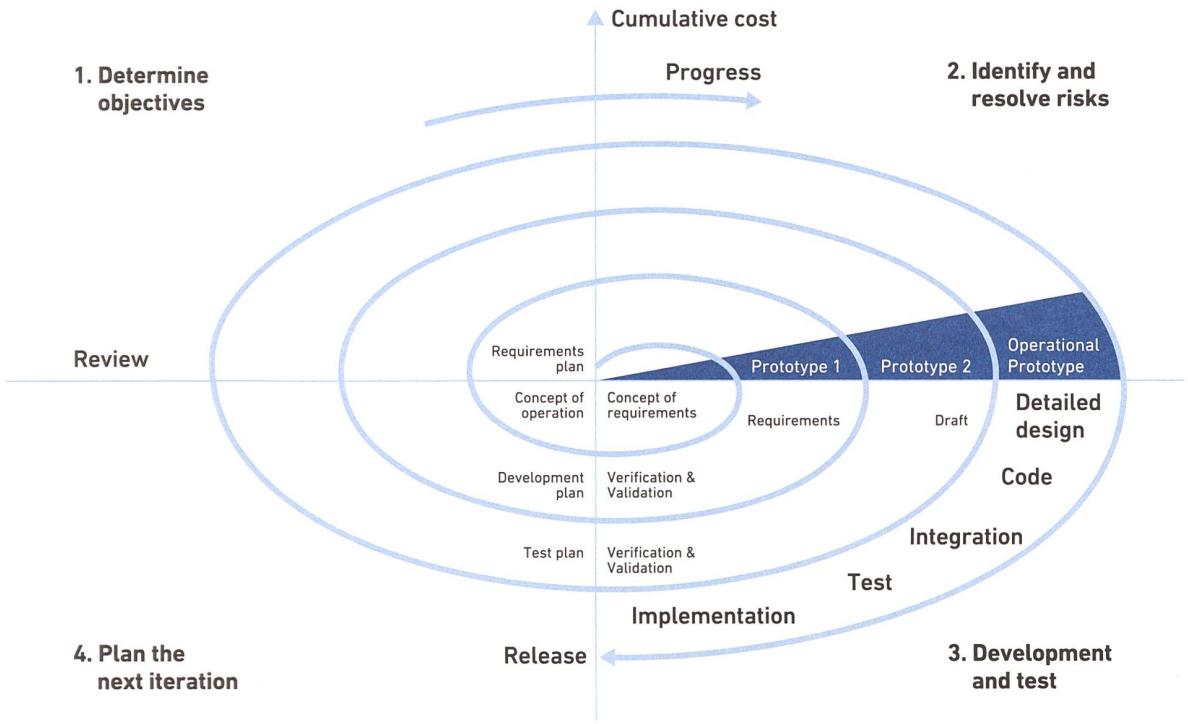
3 The title page of the Agile Manifesto set over an intentionally blurred image of the 2001 meeting, where the blurring is intended to underscore that all worked together.

such as Projects in Controlled Environments (PRINCE), were formulated and adopted as industry standards. Despite these concerted efforts towards improvement, the persisting challenges remained evident. As underscored by the aforementioned CHAOS report, the number of failed IT projects remained high throughout the 1990s.[14]

The rise of agile – the role of quality

The increased emphasis on quality did not inherently pave a clear path forward. For many within the software industry, the intensified focus on procedures and its attendant requirements for documentation appeared misguided. Rather than tackling issues such as the challenges stemming from the intangible nature of software and its ever-changing requirements, the emerging frameworks often seemed burdensome and inflexible. Consequently, there arose a desire for alternative solutions to the complexities of software development. During the 1990s, a variety of approaches began to surface, including methodologies now recognized as Scrum, Extreme Programming, and Crystal Clear, among others, collectively characterized as lightweight methods. A gathering of proponents of these methodologies convened at a ski resort in Utah in 2001, originally dubbed the "Lightweight Methods Conference". This gathering resulted in what is now commonly referred to as the Agile Manifesto, encapsulating a shared set of principles.[15]

Although the various methodologies differ from each other, they share a common enemy: the inflexible "waterfall" model. While it has been demonstrated earlier that the waterfall model never truly existed in a singular form and numerous variations were extant, it served as a representation of a process and a unifying foil, or even a straw man. Building upon this commonality, it became possible to delineate four values and twelve principles that, to some extent, encapsulated the diverse methodologies, based on the observation that they all drew upon similar conceptual frameworks and theoretical underpinnings. One such foundational concept was the notion of incremental and iterative processes, which had been recognized since the 1950s and, as evidenced by the aforementioned ideas of Royce, were fairly well known. Numerous scientific publications in the 1970s and 1980s explored this concept, resulting in various methodologies such as rapid prototyping or evolutionary development (EVO), which served as precursors for some of the methodologies discussed. Meanwhile, proponents of other methodologies focused on other roots for iterative processes. Scrum, for example, is based on the principles expressed in the "New New Product Development Game", inspired by product development processes in Japanese consumer electronics companies.[16]



4 Spiral model of iterative development after Barry Boehm.

The central idea was to achieve heightened flexibility, facilitating adaptive responses to evolving demands.

The central idea was to achieve heightened flexibility, facilitating adaptive responses to evolving demands. This emphasis on flexibility as a fundamental value required alignment with other key aspects. Among these, one of the most prominent is the empowerment of teams, affording them the autonomy to devise and modify plans according to changing needs. This principle resonates with similar ideologies advocated by figures like Kelly Johnson, the head behind the Skunk Works at Lockheed, whose presentations and writings favoured comparable ideals. Concurrently, there was a concerted effort to actively engage users and customers, with a particular emphasis on delivering value to them. This emphasis echoes the significance of user involvement in participatory design practices, which, for instance, contributed to the development of methodologies such as EVO and Rapid Prototyping. These methodologies had their roots in various communities.[17]

The emphasis on customer centricity, particularly on delivering customer value, also played a pivotal role in the second area of influence for shaping agile methodologies: leanness. Lean methodology is closely intertwined with the concept of lightweight processes, as one of its fundamental principles concerns waste reduction. However, it is important to note that lean is a rather broad and adaptable framework. It draws inspiration from traditional Japanese production methods as well as American management principles, representing a synthesis of ideas that emerged post-World War II.

The American influence is strongly connected to the work of Deming, who moved to Japan in the 1950s. Deming introduced the PDCA (Plan-Do-Check-Act) cycle, initially developed by Shewhart, to Japan. This cross-fertilized with notions of waste reduction, stemming from the imperative to optimize scarce resources before, during, and after the war. Additionally, it incorporated an emphasis on regularity to facilitate a continuous flow of work and improvements. This approach aims to achieve optimal workflow balance and prevent distortions through proactive problem recognition and resolution. Central to lean methodology is the notion of the team as the fundamental unit of collaboration, aligning well with Japanese cultural values.

These ideas found expression in various tools, such as Kaizen (continuous improvement), and methodologies like fishbone diagrams for problem identification. Often, they are amalgamated into comprehensive frameworks such as the Toyota Production System (TPS), which later inspired the concept of Total Quality Management (TQM). Across these approaches, quality is conceptualized differently compared to Western business. It transcends being merely a technical attribute of the product;

instead, it is viewed as a process aimed at delivering a product that aligns with customer preferences and is perceived as valuable. This entails avoiding unnecessary features and concentrating on essential needs, a strategy that propelled Japanese car manufacturers ahead of their American and European counterparts.[18]

In software development, these principles manifested as customer-centric development practices involving regular engagement with customer representatives. This approach entails a focus on the core functionalities of software through the prioritization of backlogs, a consistent workflow (measured by velocity), and a commitment to continuous learning and improvement, with retrospectives serving as an integral component. These elements are most prominently evident in methodologies such as Scrum and Extreme Programming (XP), which are among the most widely recognized.[19]

Principles manifested as customer-centric development practices involving regular engagement with customer representatives.

Moreover, the naming of these methodologies as "agile" reflects the influence of the Japanese understanding of quality. The term "agile" originally emerged in the American manufacturing industry in the early 1990s in response to the competitive success of Japanese companies. Faced with this challenge, the American industry sought new ideas to shape its future trajectory. In a report sponsored by the American military among others, and conducted by the Iacocca Institute at Lehigh University (named after Chrysler's longstanding CEO Lee Iacocca), agility was identified as the key response to these challenges. Published in 1991, the report defined agility as the integration of "flexible technologies of production with the skill base of knowledgeable workforce, and with flexible management structures that stimulate cooperative work".[20]

This initiative led to the establishment of the Agile Manufacturing Enterprise Forum, later known as the Agility Forum, which included companies such as Boeing, TRW, Chrysler, and GM, promoting agility as a concept. These companies collaborated on initiatives aimed at developing new processes. During this period, Kent Beck and his colleagues, for instance, experimented with XP during the development of a new payroll system called C3 at Chrysler. Given this context, it is not surprising that, in discussions regarding a suitable name for the common value set, "agile" emerged as a fitting term, while alternatives such as "lightweight" were considered inappropriate for various reasons.[21] In the years following "agile" ultimately became the overarching umbrella term for alternative methodologies in software development.

Conclusions: Towards new paradigms

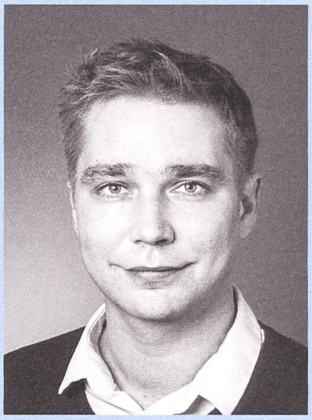
The evolution of software development methodologies has always struggled with the question of quality. It is noteworthy that the shifting perspectives on quality and its definition often mirror contemporary management trends and techniques. This underscores that the values and mindsets associated with agile, frequently discussed in the context of software development, are not unique to this field; rather, they have their origins in practices and concepts from various industries. Instead of solely exploring how these principles can be moved from IT to other sectors, it may be beneficial to delve into their origins and analyse how they emerged, subsequently developing strategies to adapt them to different industries.

Concerning the software development process itself, it becomes evident that the complexity of the challenges often originates from human-made factors. Paradoxically, the solution often lies in embracing simplicity and adopting incremental approaches to problem-solving, despite the fact that software is frequently used to build complex solutions. Furthermore, this challenges conventional metrics for success, such as time and cost, as well as the elusive concept of quality.

Notably, respected practitioners like Tom DeMarco, in his reflection on the 50-year development since the first conference on software engineering in 1968, advocate for a shift in focus. Instead of solely evaluating the development process, there is a need to assess the transformations that software brings to society and businesses – the actual value it delivers. This reorientation emphasizes the importance of looking beyond process-oriented metrics to realize the true impact of software.

About the author

Timo Leimbach, Prof. Dr.



Timo Leimbach is associate professor at the department for Digital Design and Information Studies, Aarhus University, where he researches project management and digital innovation and their interrelations with business and society. Before that he worked and researched at, among others, Fraunhofer ISI, the Research Institute for the History of Technology and Science of the Deutsches Museum, the Institute for Information Sciences and New Media at LMU Munich and the Department for Management, Politics and Philosophy at the Copenhagen Business School. He received a master's degree in Economic and Modern History as well as Business Administration from the University of Mannheim, Germany (2003) and obtained his PhD from the LMU Munich for his thesis on the development of the German software industry (2009).

Aarhus University, Denmark
timo.leimbach@cc.au.dk

Related article in the Ferrum archive:
"Die Entwicklung der logischen Basis der Computerwissenschaften"
by Heinz Zemanek in Ferrum 58/1987

Annotations

1 Frederick P. Brooks, The Mythical Man-month: Essays on Software Engineering, 25th Anniversary Edition, Boston 1995. Also contains the later article on the silver bullet.

2 Johan Eveleens and Chris Verhoef, The Rise and Fall of the Chaos Report Figures, in: IEEE software 27(1) (2009), p. 30–36.

3 Thomas Haigh and Paul Ceruzzi, A New History of Modern Computing, Boston 2021, p. 59–138.

4 Mike Mahoney, Finding a History for Software Engineering, in: IEEE Annals of the History of Computing 26(1) (2004), p. 8–19.

5 Tom Haigh, Crisis, What Crisis? Reconsidering the Software Crisis of the 1960s and the Origins of Software Engineering. Paper presented at Tensions of Europe Conference, Sofia, Bulgaria, 2009, available at: https://www.tomandmaria.com/Tom/Writing/SoftwareCrisis_SofiaDRAFT.pdf.

6 Mahoney (see n. 4); Sandy Payette, Hopper and Dijkstra: Crisis, Revolution, and the Future of Programming, in: IEEE Annals of the History of Computing 36(4) (2014), p. 64–73.

7 Peter Naur and Brian Randell, B. (Ed.), Software Engineering: Report of a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7–11 October 1968, Brussels 1969, p. I.

8 Donald Knuth, Art of Programming, Volume 1: Fundamental algorithms, Boston 1997.

9 Mahoney (see n. 4); Naur/Randell (see n. 7); Payette/Hopper/Dijkstra (see n. 6).

10 Christine Pieper, Hochschulinformatik in der Bundesrepublik und der DDR bis 1989/1990, Stuttgart 2009, p. 159–164.

11 Lauri Koskela and Gregory Howell, The Underlying Theory of Project Management Is Obsolete, in: IEEE Engineering Management Review 2(36) (2008), p. 22–34.

12 Herbert Benington, Production of Large Computer Programs, in: Annals of the History of Computing 5(4) (1983), p. 350–361.

13 William W. Royce, Managing the Development of Large Software Systems, in: Proceedings of IEEE WESCON, 1970, p. 328–388.

14 Peter Morris, Reconstructing Project Management, London 2013, p. 52–98.

15 Robert Martin, Clean Agile: Back to Basics, Boston 2019, p. 3–13.

16 Darrel Rigby, Jeff Sutherland and Hirohito Takeuchi, The Secret History of Agile Innovation, in: Harvard Business Review, 2016, accessible at: https://hbr.org/2016/04/the-secret-history-of-agile-innovation; Craig Larman and Viktor Basili, Iterative and Incremental Developments: a Brief History, in: Computer 36(6) (2003), p. 47–56.

17 Larman/Basili (see n. 16).

18 Kieran Conboy, Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development, in: Information Systems Research 20(3) (2009), p. 329–354.

19 Martin (see n. 15), p. 14–26.

20 Roger Nagel and Rick Dove, 21st Century Manufacturing Enterprise Strategy: an Industry-led View, Part 1, Bethlehem 1991, p. 1.

21 Martin (see n. 15), p. 10–13.

22 Roger Atkinson, Project Management: Cost, Time and Quality, Two Best Guesses and a Phenomenon, its Time to Accept Other Success Criteria, in: International Journal of Project Management 17(6) (1999), p. 337–342.

23 Tom DeMarco, Software Engineering: An Idea whose Time Has Come and Gone?, in: IEEE Software 26 (2008), p. 96.

Image Credits

1 © University of London.

2 © Timo Leimbach.

3 © Alistair Cockburn.

4 After: Barry Boehm, Spiral Development: Experience, Principles, and Refinements, Special Report of the Software Engineering Institute, Carnegie Mellon University, July 2000.