A Brief History of the Waterfall Model: Past, Present, and Future

Antonios Saravanos
saravanos@nyu.edu
New York University
New York, NY, USA

Abstract

The waterfall model, one of the earliest software development methodologies, has played a foundational role in shaping contemporary software engineering practices. This paper provides a historical and critical overview of the model, tracing its conceptual origins in software engineering, its formalization by Royce, and its evolution through decades of industry adoption and critique. Although often criticized for its rigidity, shortcomings, and high failure rates, the waterfall model persists in specific domains. Its principles continue to influence contemporary hybrid development frameworks that combine traditional and agile methods. Drawing on a range of scholarly sources, this study synthesizes key developments in the perception and application of the waterfall model. The analysis highlights how the model has shifted from a standalone framework to a component within modern hybrid methodologies. By revisiting its origins, assessing its present utility, and examining its role in contemporary development practices, this paper argues that the waterfall model remains relevant, not as a relic of the past but as part of context-aware development strategies. The paper contends that the model's enduring relevance lies in its adaptability. By recognizing both its limitations and its strengths, and by understanding its integration within hybrid approaches, practitioners can make more informed decisions about methodology selection and process design in diverse development environments.

CCS Concepts

• **Software and its engineering → Software creation and management**; **Software development process management**; **Software development methods**; **Waterfall model**;

Keywords

Waterfall model, Systems development life cycle, Software development life cycle, SDLC, Diverse development environments

1 Introduction and Origins

This paper offers a contemporary overview of the popular waterfall model. A structured formula for developing systems is often credited to Royce [1] (available in the ACM Digital Library), though some argue that it builds on earlier work by Benington [2]. Emerging during a period when the field was still in its formative stages, the model offered a much-needed structured approach to managing the growing complexity of software systems. It formalized a phase-driven development cycle that mirrored the logical progression of engineering projects, beginning with requirements gathering and ending in deployment and maintenance. As such, it became the blueprint for how software systems were conceptualized, designed, and built in the latter half of the twentieth century. The model gets its name from its visual appearance: a sequence of steps flowing downward from one phase to the next, resembling water flowing down a waterfall [3]. It is often categorized under the broader systems development life cycle (SDLC) umbrella, sometimes used interchangeably with the term software development life cycle [3]. Strictly speaking, one is part of the other [4]. As Ruparelia [4] explains, these distinctions have largely blurred in modern practice, where integrated systems development increasingly treats software as the central component. Consequently, the terms SDLC and waterfall model are frequently used synonymously in both academic and professional discourse [4]. According to Ruparelia [4], "a lifecycle covers all the stages of software from its inception with requirements definition through to fielding and maintenance".

Royce's 1970 formalization [1], reprinted in 1987, comprises seven phases: system requirements, software requirements, analysis, program design, coding, testing, and operations, arranged so that each depends on the deliverables of the preceding one. His paper presents a sequence of refinements. The second model (a linear flow without feedback) and the third (with feedback to the prior phase) are the versions most commonly labeled as "waterfall". Notably, Royce recommended executing the development cycle at least twice, as illustrated in Figure 1(b). The first pass, in his words, "provides an early simulation of the final product" [1], while the second produces a more robust solution. His paper was explicitly critical of a rigid, one-pass approach for large systems. Nevertheless, this guidance was often overlooked in early industry adoption, which favored the most linear interpretation and in turn invited later critiques of inflexibility and project failure. Contrary to popular belief, Royce did not advocate a strictly linear process; he prescribed repetition to accommodate learning and refinement. In retrospect, this emphasis on iteration and feedback foreshadows the incremental practices later emphasized by Agile methods.

Interestingly, the term "waterfall" does not appear in Royce's [1] original paper. It was later popularized by Bell and Thayer [5] in 1976. Over the following decades, the waterfall model became widely institutionalized in both government and private-sector software development. It was often codified in official project management standards, particularly in regulated industries such as defense, aerospace, and healthcare. Despite its historical importance, the waterfall model has faced substantial criticism, especially since the 1990s. As the software industry encountered growing challenges with project overruns, shifting customer requirements, and rapid technological change, the model's rigidity and limited adaptability came under scrutiny. Influential studies, including the Standish Group's CHAOS Report [6], reinforced these concerns by linking traditional approaches such as waterfall to high rates of project failure. These critiques, combined with the rise of iterative and

incremental methods such as Agile, led many to question whether waterfall had outlived its usefulness.

Nonetheless, this paper argues that the waterfall model should not be dismissed as a relic of the past. Although it is no longer dominant as a standalone methodology, it continues to influence contemporary software engineering in important ways. First, it remains well suited to projects with stable requirements, clearly defined scopes, and strong demands for traceability and documentation. Second, its structure and discipline have gained renewed relevance in hybrid development methodologies that combine traditional and Agile practices. In these approaches, the sequential logic of waterfall is often retained at the macro level, such as in planning or compliance phases, while Agile methods are applied at the team or sprint level to enhance responsiveness and flexibility.

Building on this historical foundation, the remainder of this paper is organized as follows. Section 2 surveys the treatment of the waterfall model in mainstream software engineering literature, emphasizing how it has been interpreted, adapted, and critiqued. Section 3 examines its contemporary relevance, particularly its continued use both as a standalone method and as a component within hybrid methodologies. Section 4 presents a critical discussion and conclusion, reflecting on the enduring legacy and evolving applications of the waterfall model in modern software engineering.

2 Perceptions of Waterfall

This section examines the evolving structure and perception of the waterfall model. First, we review how the model has been described and adapted in software engineering literature. We then examine its historical association with project failure.

2.1 Composition of Waterfall

As discussed in Section 1, the original waterfall model is often presented as a rigid, linear sequence of development phases. The literature, however, reveals a range of adaptations and interpretations. This subsection presents three representative snapshots from academic sources that illustrate how the model has been described over time. Notably, each of these sources depicts the waterfall model as consisting of five phases, rather than the seven originally described by Royce [1].

The first example is Petersen et al. [7], whose highly cited 2009 paper *The Waterfall Model in Large-Scale Development* defines five phases. The first is requirements engineering, in which "the needs of the customers are identified and documented on a high abstraction level" and "the requirements are refined so that they can be used as input to the design and implementation phase" [7]. The second is design and implementation, subdivided into two parts: the design, where "the architecture of the system is created and documented", and the implementation, where "the actual development of the system takes place" [7]. The third phase is testing, in which "the system integration is tested regarding quality and functional aspects" [7]. The fourth is release, defined as the point where "the product is brought into a shippable state" [7]. The final phase is maintenance, where "after the product has been released to the customer it has to be maintained" and "if customers discover problems in the product they report them to the company and get support in solving them".

The second example is Sommerville's [8] ninth edition of *Software Engineering* (2011), which also presents five stages of the waterfall model. The first is requirements analysis and definition, in which "the system's services, constraints, and goals are established by consultation with system users" and then "defined in detail and serve as a system specification" [8]. The second is system and software design, where "the systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture", and "software design involves identifying and describing the fundamental software system abstractions and their relationships" [8]. The third stage is implementation and unit testing, during which "the software design is realized as a set of programs or program units", each verified to ensure "that each unit meets its specification" [8]. The fourth stage is integration and system testing, in which "the individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met", after which "the software system is delivered to the customer" [8]. Finally, operation and maintenance occurs when "the system is installed and put into practical use" and "maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units, and enhancing the system's services as new requirements are discovered" [8]. The same model appears in the tenth edition of the book, published in 2016 [9].

The third example is Andrei et al. [10], who in 2019 describe the model with reference to Davis [11] (2012) and van Casteren [12] (2017). Their five stages are: requirements, defined as "analyzing business needs and extensive documentation of all features" [10]; design, described as "choosing all required technology and planning the full software infrastructure and interaction" [10]; coding, defined as "solving all problems, optimizing solutions and implementing each component described in the requirements phase, using the diagrams and blueprints from the design phase" [10]; testing, described as "extensive testing of all implemented features and components and solving any occurring issues" [10]; and finally operations, defined as "deployment to a production environment" [10]. Table 1 summarizes how these three sources present the phases of the waterfall model.

We can see these phases connected in various ways across the literature, as illustrated in Figure 1. The first waterfall lifecycle (a) is linear, with a single pass from start to finish. This representation is common in popular literature and is often associated with project failure, particularly in large projects. It reflects the second of the models presented by Royce [1]. The second model (b) illustrates a two-pass waterfall, where the requirements phase (Phase 1 in this example) is excluded from the second pass. This aligns with Andrei et al. [10], who note that "the Waterfall model assumes that once the initial requirements are set and every goal has been cleared of any ambiguities, there is an unobstructed road which the development team will follow towards finishing the project". It should be emphasized that Royce [1] proposed several variations in conjunction with the two-pass idea, but since these are not commonly associated with the waterfall model, they fall outside the scope of this paper. The third model (c) represents a more contemporary interpretation of waterfall, where failure in one phase allows for feedback and repetition of the immediately preceding phase to correct mistakes.

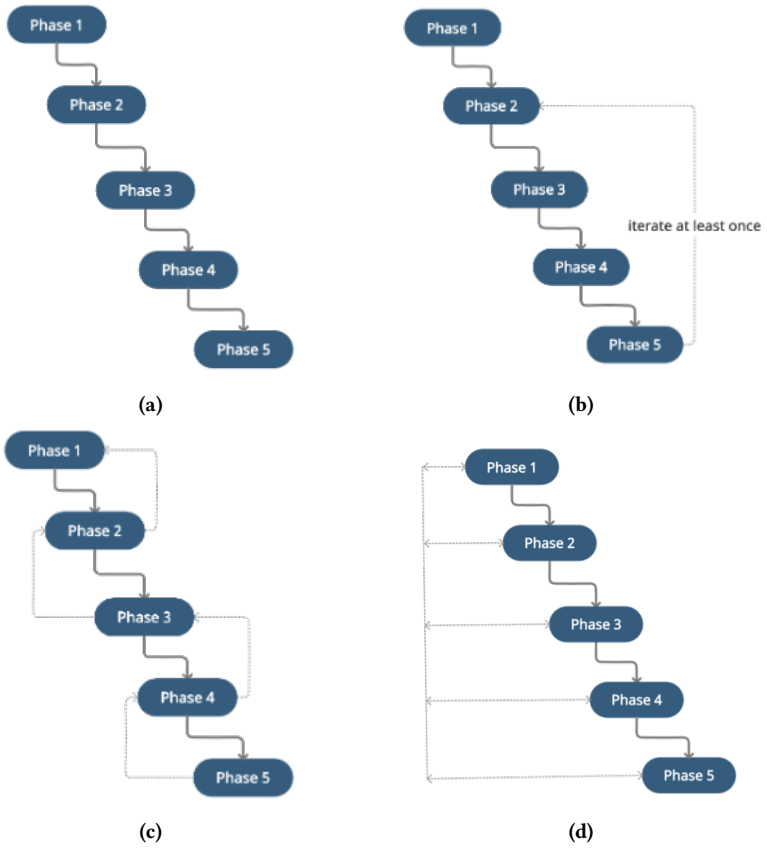

Figure 1: Four Structural Views of the Waterfall Model: (a) Linear, (b) Two-Pass, (c) Feedback Loops to Previous Phase, (d) Feedback Loops to Any Previous Phase.

Table 1: Phases of the waterfall model as presented by different sources.

Phase	Petersen et al. [7] (2009)	Sommerville [8] (2011); [9] (2016)	Andrei et al. [10] (2019)
1	Requirements engineering	Requirements analysis and definition	Requirements
2	Design and implementation	System and software design	Design
3	Testing	Implementation and unit testing	Coding
4	Release	Integration and system testing	Testing
5	Maintenance	Operation and maintenance	Operations

This corresponds to one of the variations described by Royce [1]. Finally, the fourth model (d) permits feedback to any earlier phase. This allows a team to revisit and repeat any previous stage in order to make corrections before moving forward.

2.2 Reputation and Critique in Literature

The waterfall model is closely associated with the notion of failure, a reputation it shares with the broader practice of software development [13, 14]. Before the emergence of rapid application development (RAD) in 1991 and Agile in 2001, the software development life cycle and the waterfall model were largely synonymous. The idea of failure in software development was recognized early in the field's history. Ebert [15] recalls that the 1968 conference, later recognized as the first on what became known as software engineering, identified "the so-called software crisis" as a central problem. Saravanos and Curinga [16] survey literature that highlights failure in the craft of software development. For example, Charette [17] observes that "few IT projects, in other words, truly succeed". Similarly, Bloch et al. [18] report that "on average, large IT projects run 45 percent over budget and 7 percent over time, while delivering 56 percent less value than predicted", based on a review of more than 5,400 IT projects each costing over $15 million. Lauesen [19] notes that "we have known for decades that IT projects often fail". Charette [17] further states that "from 5 to 15 percent" of projects are "abandoned before or shortly after delivery". Gilb [20] critiques

the model directly, calling it "unrealistic, and dangerous to the primary objectives of any software project". The most influential body of research linking software development with failure is the Standish Group's CHAOS Report. Eveleens and Verhoef [21] cite the 1994 report, which stated that "Standish reported a shocking 16% project success rate, another 53% of the projects had overruns of costs or time or less functionality and 31% of the projects failed outright". They emphasize the report's impact on perception, writing that "many authors have quoted the Standish figures to illustrate information technology is in a troublesome state", that "over the years their figures have had tremendous attention", and that "the figures indicate large problems with software engineering projects, and as such have had an enormous impact on information technology". However, Cerpa and Verner [13] point out that Jørgensen and Moløkken-Østvold [22] questioned the methodology of the Standish Group, suggesting that "there are serious problems with the way the Standish Group conducted their research and that the findings were biased toward reports of failure because a random sample of top IT executives was asked to share failure stories when mailed confidential surveys". Consequently, it is possible that the reputation of failure was exaggerated, or at least not as severe as the reports suggested.

3 Is There a Present and Future for the Waterfall Model?

With the proliferation of Agile methodologies [23], it is reasonable to ask whether the waterfall model still has a place in modern software engineering practice. Evidence shows that it continues to be used [7, 24]. A 2019 study by Andrei et al. [10] reported that software developers applied waterfall 28.1% of the time compared to Agile methods. Similarly, a 2020 survey conducted by the Project Management Institute (PMI) [25], reported in its annual *Pulse of the Profession* appendix [26], found that slightly more than half of organizations (56%) continued to use traditional approaches, including waterfall and similar structured methods (e.g., parallel or V-model).

Although Agile methods are gaining ground [27], the persistence of waterfall suggests that it will remain relevant. Petersen et al. [7] note that "the model is still widely used in software industry", citing Raccoon [28] and adding that "some researchers are even convinced that it will be around for a much longer period of time". We agree and argue that waterfall remains relevant for two reasons: first, in projects where it aligns with project characteristics, and second, as a foundational component within emerging hybrid methodologies that integrate waterfall with other approaches (e.g., Agile, Scrum, iterative, and incremental).

Some projects continue to be well matched to the waterfall model. Mishra and Alzoubi [27] observe that "many firms are still using the waterfall methodology since it simply works and has a successful track record". In other words, project suitability is critical. Dennis et al. [29] also caution that "choosing a methodology is not simple, because no single methodology is always best". Wallis [30] identifies three strengths of waterfall. First is its clear structure, as "the model provides a well-defined and structured approach to software development", making it "suitable for projects with stable and clearly defined requirements, where a sequential and linear development process is appropriate" [30]. Second is its focus on comprehensive documentation, an advantage in contexts requiring "regulatory compliance, knowledge transfer, and future maintenance or enhancements" [30]. Third is its affinity for project planning, as it demands "detailed project planning upfront", which "can be beneficial for managing resources, setting clear milestones, and estimating project timelines and costs" [30]. A common explanation for waterfall's high failure rate is its misapplication to projects poorly suited for it. For example, some teams may use it simply because it is the only methodology they know. Wallis [30] warns that "businesses should carefully consider whether the waterfall model aligns with their project requirements and organizational context", noting that "factors such as the stability of requirements, the need for flexibility, stakeholder involvement, and the dynamic nature of the industry should be evaluated". For projects with these needs, Wallis [30] recommends Agile or iterative methodologies. Sommerville [8] similarly cautions that "the waterfall model should only be used when the requirements are well understood and unlikely to change radically during system development".

The second explanation for waterfall's persistence is its incorporation into hybrid approaches. Kuhrmann et al. [31] define a hybrid approach as "any combination of agile and traditional (plan-driven or rich) approaches that an organizational unit adopts and customizes to its own context needs". Prenner et al. [32] similarly write that "to benefit from the strengths of both approaches, software companies often use a combination of agile and plan-based methods, known as hybrid development approaches". Tell et al. [33] provide a comparable definition: "any combination of agile and traditional approaches that an organizational unit adopts and customizes to its own context needs". The origins of hybrid practice can, according to Kirpitsas and Pachidis [34], be traced to the work of Glass [35], whose 2003 paper in *IEEE Software* is often cited as an early reference. Küpper et al. [36], drawing on the work of West et al. [37], argue that "hybrid software and systems development has become standard in practice". The benefits of hybrid approaches are highlighted by Kuhrmann et al. [38], who note that "hybrid development provides a practical balance, combining the structure and predictability of traditional methods with the flexibility and responsiveness of agile approaches". They further explain that "these combinations are often not the result of deliberate planning but instead evolve organically based on practical experience, project needs, client demands, and regulatory requirements" [38]. This observation is echoed by Küpper et al. [36], who reference the HELENA study [31] in stating that "hybrid development approaches are barely planned or defined in advance". Klünder et al. [39] similarly report that hybrid practices tend to emerge from a bottom-up rather than a top-down approach.

We can obtain an overview of the hybrid landscape as it relates to the waterfall model by examining a few key studies. Within the scope of this paper, we focus only on hybrid models that explicitly incorporate waterfall. The first is a systematic review conducted in 2020 by Prenner et al. [32], who investigated how companies organize software development processes to combine Agile and plan-driven methods. Reviewing 24 papers, the authors concluded that all hybrid approaches fundamentally rely on the waterfall model, stating that "all hybrid approaches are using in some way the phases described in Royce's waterfall model" [32]. Prenner et

al. [32] identified three organizational patterns: the waterfall–Agile approach (WAA), also called Agilefall, in which Agile methods are integrated into a waterfall structure; the waterfall–iterative approach (WIA), also called Waterative, where smaller waterfall cycles occur within iterations; and the pipeline approach (PA). Among these, WAA is the most widely used, followed by WIA and then PA. Combinations of approaches were also observed, such as WAA in conjunction with WIA.

A later study in 2022, a systematic literature review by Reiff and Schlegel [40], provides "a structured overview of the current state of research regarding the topic". They identify two definitions of hybrid: first, "a combination/mix of agile and traditional project management methodologies", and second, "the integration of an agile approach into existing traditional project management methodologies" [40]. The authors highlight four main hybrid models, two of which incorporate waterfall (water-scrum-fall and waterfall–Agile). They argue that hybrid approaches "maximize project success" and stress their value in allowing companies to "use certain agile practices, even if there are constraints that impede the adoption of a pure agile approach" [40]. Reiff and Schlegel [40] conclude that "hybrid systems that enable iteration and continuous evolution represent the future" and call for further research to establish structured frameworks and more robust evaluations of hybrid project management methodologies. This reinforces the view that the waterfall model will persist, not as a standalone methodology, but as a component within hybrid approaches.

4 Discussion and Conclusion

As this paper has shown, the waterfall model holds a significant place in the evolution of software and systems development. From its conceptual roots in Benington's [2] early process work to Royce's [1] formalization, often misunderstood and oversimplified, the waterfall model shaped how developers and organizations approached complex projects. Although widely criticized for its rigidity and limited ability to accommodate changing requirements [41], the sequential structure of the model continues to offer value in specific contexts, particularly where requirements are stable and well defined [8, 29, 42]. While the model has been closely linked to project problems and failures [43], many of these outcomes can be attributed to misapplication. In particular, difficulties arise when waterfall is used in situations where requirements are unknown at the outset or subject to rapid change. This supports the pragmatic view expressed by Sommerville [8, 44], who advocates for context-sensitive methodology selection rather than adherence to a single universal model.

Despite the emergence [45], rise [46, 47], and dominance of Agile (iterative and incremental) methodologies [23, 48], the waterfall model maintains a foothold in industry [7, 10, 23, 25]. Recent shifts in software engineering show the model finding renewed purpose in hybrid approaches that blend waterfall with Agile, combining the strengths of both traditional practices (structure and rigor) and Agile practices (flexibility) [33]. Notable examples include Water-Scrum-Fall [37, 49] and Scrumbanfall [50], which demonstrate how waterfall principles have been selectively retained and integrated into modern development workflows. These developments suggest that the story of the waterfall model is not one of obsolescence but of evolution.

This historical and critical reflection underscores that the value of the waterfall model is not confined to the past. Its adaptability, whether through selective application or hybridization, points to an enduring relevance. The model continues to coexist alongside modern methodologies, with its core principles offering value in appropriate contexts. Future research should further examine contemporary uses of the waterfall model across projects of varying scales to extract lessons learned; refine simulation techniques to support evidence-based decisions around its use (see, for example, Bassil [51] and Saravanos and Curinga [16]); and contribute to the development of structured hybrid frameworks.

References

[1] W. W. Royce. 1987. Managing the development of large software systems: Concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering*, Monterey, California, USA, 328–338. IEEE Computer Society Press.

[2] H. D. Benington. 1987. Production of Large Computer Programs. In *Proceedings of the 9th International Conference on Software Engineering (ICSE '87)*. IEEE Computer Society Press, Washington, DC, USA, 299–310.

[3] Iqbal H. Sarker, Faisal Faruque, Ujjal Hossen, and Atikur Rahman. 2015. A survey of software development process models in software engineering. *International Journal of Software Engineering and Its Applications* 9, 11 (2015), 55–70.

[4] Nayan B. Ruparelia. 2010. Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes* 35, 3 (2010), 8–13.

[5] Thomas E. Bell and Thomas A. Thayer. 1976. Software requirements: Are they really a problem? In *Proceedings of the 2nd International Conference on Software Engineering*, 61–68.

[6] Standish Group International, Inc. 1995. *The CHAOS Report*. Standish Group International, Inc.

[7] Kai Petersen, Claes Wohlin, and Dejan Baca. 2009. The waterfall model in large-scale development. In *Product-Focused Software Process Improvement*, 386–400. Springer Berlin Heidelberg, Berlin, Heidelberg.

[8] Ian Sommerville. 2011. *Software Engineering* (9th ed.). Addison-Wesley, Boston, MA, USA.

[9] Ian Sommerville. 2016. *Software Engineering* (10th ed.). Pearson Education Limited, Harlow, United Kingdom.

[10] Bogdan-Alexandru Andrei, Andrei-Cosmin Casu-Pop, Sorin-Catalin Gheorghe, and Costin-Anton Boiangiu. 2019. A study on using waterfall and agile methods in software project management. *Journal of Information Systems & Operations Management* (2019), 125–135.

[11] Barbee Davis. 2012. *Agile Practices for Waterfall Projects: Shifting Processes for Competitive Advantage*. J. Ross Publishing.

[12] Wilfred van Casteren. 2017. The waterfall model and the agile methodologies: A comparison by project characteristics. ResearchGate preprint. Retrieved October 4, 2025 from https://www.researchgate.net/publication/317225452

[13] Narciso Cerpa and June M. Verner. 2009. Why did your project fail? *Communications of the ACM* 52, 12 (December 2009), 130–134. https://doi.org/10.1145/1610252.1610286

[14] Mohammad Yasir Kotowaroo and Roopesh Kevin Sungkur. 2022. Success and Failure factors affecting software development projects from IT professionals' perspective. In *Soft Computing for Security Applications: Proceedings of ICSCS 2022*. Springer, 757–772.

[15] Christof Ebert. 2018. 50 years of software engineering: Progress and perils. *IEEE Software* 35, 5 (2018), 94–101.

[16] Antonios Saravanos and Matthew X. Curinga. 2023. Simulating the software development lifecycle: The waterfall model. *Applied System Innovation* 6, 6 (2023). https://doi.org/10.3390/asi6060108

[17] R. N. Charette. 2005. Why software fails [software failure]. *IEEE Spectrum* 42, 9 (September 2005), 42–49. https://doi.org/10.1109/MSPEC.2005.1502528

[18] Michael Bloch, Sven Blumberg, and Jürgen Laartz. 2012. Delivering large-scale IT projects on time, on budget, and on value. *McKinsey on Business Technology* 27 (2012), 2–7.

[19] Soren Lauesen. 2020. IT project failures, causes and cures. *IEEE Access* 8 (2020), 72059–72067.

[20] Tom Gilb. 1985. Evolutionary delivery versus the "waterfall model". *SIGSOFT Software Engineering Notes* 10, 3 (July 1985), 49–61. https://doi.org/10.1145/1012483.1012490

[21] J. Eveleens and C. Verhoef. 2010. The rise and fall of the Chaos report figures. *IEEE Software* 27, 1 (February 2010), 30–36. https://doi.org/10.1109/MS.2009.154

[22] Magne Jørgensen and Kjetil Moløkken-Østvold. 2006. How large are software cost overruns? A review of the 1994 CHAOS report. *Information and Software Technology* 48, 4 (April 2006), 297–301. https://doi.org/10.1016/j.infsof.2005.07.002

[23] C. Fagarasan, O. Popa, A. Pisla, and C. Cristea. 2021. Agile, waterfall and iterative approach in information technology projects. *IOP Conf. Ser.: Mater. Sci. Eng.* 1169, 1 (2021), 012025. https://doi.org/10.1088/1757-899X/1169/1/012025

[24] Watts S. Humphrey and Marc I. Kellner. 1989. Software process modeling: Principles of entity process models. In *Proceedings of the 11th International Conference on Software Engineering*, 331–342.

[25] PMI. 2020. *Ahead of the Curve: Forging a Future-Focused Culture.* Pulse of the Profession. Retrieved April 23, 2025 from https://www.pmi.org/learning/library/forging-future-focused-culture-11908

[26] PMI. 2020. Appendix. *Pulse of the Profession.* Retrieved April 23, 2025 from https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/pmi-pulse-2020-appendix.pdf

[27] Alok Mishra and Yehia Ibrahim Alzoubi. 2023. Structured software development versus agile software development: A comparative analysis. *International Journal of System Assurance Engineering and Management* 14, 4 (August 2023), 1504–1522. https://doi.org/10.1007/s13198-023-01958-5

[28] L. B. S. Raccoon. 1997. Fifty years of progress in software engineering. *ACM SIGSOFT Software Engineering Notes* 22, 1 (1997), 88–104.

[29] Alan Dennis, Barbara Haley Wixom, and Roberta M. Roth. 2012. *Systems Analysis and Design* (5th ed.). John Wiley & Sons.

[30] Julian Wallis. 2023. What is the waterfall model in software development? Retrieved April 27, 2025 from https://intuji.com/what-is-the-waterfall-model-in-development/

[31] Marco Kuhrmann, Philipp Diebold, Jürgen Münch, Paolo Tell, Vahid Garousi, Michael Felderer, Kitija Trektere, Fergal McCaffery, Oliver Linssen, and Eckhart Hanser. 2017. Hybrid software and system development in practice: Waterfall, scrum, and beyond. In *Proceedings of the 2017 International Conference on Software and System Processes (ICSSP '17).* Association for Computing Machinery, New York, NY, USA, 30–39.

[32] Nils Prenner, Carolin Unger-Windeler, and Kurt Schneider. 2020. How are hybrid development approaches organized? A systematic literature review. In *Proceedings of the International Conference on Software and System Processes (ICSSP '20).* Association for Computing Machinery, New York, NY, USA, 145–154.

[33] Paolo Tell, Jil Klünder, Steffen Küpper, David Raffo, Stephen G. MacDonell, Jürgen Münch, Dietmar Pfahl, Oliver Linssen, and Marco Kuhrmann. 2019. What are hybrid development methods made of? An evidence-based characterization. In *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, 105–114. IEEE.

[34] Ioannis K. Kirpitsas and Theodore P. Pachidis. 2022. Evolution towards hybrid software development methods and information systems audit challenges. *Software* 1, 3 (2022), 316–363. https://doi.org/10.3390/software1030015

[35] Robert L. Glass. 2003. The state of the practice of software engineering. *IEEE Software* 20, 6 (2003), 20–21.

[36] Steffen Küpper, Andreas Rausch, and Urs Andelfinger. 2018. Towards the systematic development of hybrid software development processes. In *Proceedings of the 2018 International Conference on Software and System Processes (ICSSP '18).* Association for Computing Machinery, New York, NY, USA, 157–161. https://doi.org/10.1145/3202710.3203158

[37] Dave West, Mike Gilpin, Tom Grant, and Alissa Anderson. 2011. Water-Scrum-Fall is the reality of agile for most organizations today. Forrester Research. July 26, 2011. Retrieved October 4, 2025 from https://www.forrester.com/report/water-scrum-fall/RES58861

[38] Marco Kuhrmann, Philipp Diebold, Jürgen Münch, Paolo Tell, Kitija Trektere, Fergal McCaffery, Vahid Garousi, Michael Felderer, Oliver Linssen, and Eckhart Hanser. 2018. Hybrid software development approaches in practice: A European perspective. *IEEE Software* 36, 4 (2018), 20–31.

[39] Jil Klünder, Philipp Hohl, Masud Fazal-Baqaie, Stephan Krusche, Steffen Küpper, Oliver Linssen, and Christian R. Prause. 2017. HELENA study: Reasons for combining agile and traditional software development approaches in German companies. In *Product-Focused Software Process Improvement: 18th International Conference (PROFES 2017)*, Innsbruck, Austria, November 29–December 1, 2017. Springer, 428–434.

[40] Janine Reiff and Dennis Schlegel. 2022. Hybrid project management – A systematic literature review. *International Journal of Information Systems and Project Management* 10, 2 (2022), 45–63.

[41] Conrad Weisert. 2003. Waterfall methodology: There's no such thing! Retrieved December 17, 2021 from https://www.idinews.com/waterfall.html

[42] Alan Dennis, Barbara Wixom, and David Tegarden. 2015. *Systems Analysis and Design: An Object-Oriented Approach with UML* (5th ed.). John Wiley & Sons.

[43] G. R. Gladden. 1982. Stop the life-cycle, I want to get off. *ACM SIGSOFT Software Engineering Notes* 7, 2 (1982), 35–39.

[44] Ian Sommerville. 1996. Software process models. *ACM Computing Surveys (CSUR)* 28, 1 (1996), 269–271.

[45] Subhas Misra, Vinod Kumar, Uma Kumar, Kamel Fantazy, and Mahmud Akhter. 2012. Agile software development practices: Evolution, principles, and criticisms. *International Journal of Quality & Reliability Management* 29, 9 (January 2012), 972–980. https://doi.org/10.1108/02656711211272863

[46] Rashina Hoda, Norsaremah Salleh, and John Grundy. 2018. The rise and evolution of agile software development. *IEEE Software* 35, 5 (2018), 58–63.

[47] Andrew Whiteley, Julien Pollack, and Petr Matous. 2021. The origins of agile and iterative methods. *The Journal of Modern Project Management* 8, 3 (2021).

[48] Colin Bryar and Bill Carr. 2021. Have we taken agile too far? *Harvard Business Review.* April 9, 2021. Retrieved October 4, 2025 from https://hbr.org/2021/04/have-we-taken-agile-too-far.

[49] Georgios Theocharis, Marco Kuhrmann, Jürgen Münch, and Philipp Diebold. 2015. Is water-scrum-fall reality? On the use of agile and traditional development practices. In *Product-Focused Software Process Improvement*, 149–166. Springer International Publishing, Cham.

[50] Krunal Bhavsar, Vrutik Shah, and Samir Gopalan. 2020. Scrumbanfall: An agile integration of scrum and kanban with waterfall in software engineering. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)* 9, 4 (2020), 2075–2084.

[51] Youssef Bassil. 2012. A simulation model for the waterfall software development life cycle. *arXiv preprint* arXiv:1205.6904 (2012).