software

MDPI

Article

Evolution towards Hybrid Software Development Methods and Information Systems Audit Challenges

Ioannis K. Kirpitsas [1] and Theodore P. Pachidis [2,*]

[1] Department of Computer Science, International Hellenic University, Agios Loukas, 65404 Kavala, Greece
[2] Human-Machines Interaction Laboratory (HUMAIN-Lab), Department of Computer Science, International Hellenic University, Agios Loukas, 65404 Kavala, Greece
* Correspondence: pated@cs.ihu.gr; Tel.: +30-2510-462281

Abstract: The key objective of this paper is to investigate the evolution of hybrid software development methods and highlight the main difficulties that arise with regard to information systems (IS) auditing. While technology firms today are under constant pressure to deliver software faster due to emerging needs worldwide, this continuous effort leads to innovative development models, apparently driven by practice. Since modern software development is neither pure linear phases progression nor agile, a challenge arises with regards to selecting the appropriate combination of approaches that serve to reach goals and assure value creation for organizations.

Keywords: software development; hybrid development methods; SDLC; agile; information systems audit

Citation: Kirpitsas, I.K.; Pachidis, T.P. Evolution towards Hybrid Software Development Methods and Information Systems Audit Challenges. *Software* **2022**, *1*, 316–363. https://doi.org/10.3390/software1030015

Academic Editor: Tommi Mikkonen

Received: 29 June 2022
Accepted: 16 August 2022
Published: 24 August 2022

1. Introduction

Technology firms have faced the challenge over recent decades to deliver software faster, due to the emerging need for cutting-edge digital technologies. The increased demand for software systems is reflected in the size of the information and communication technology (ICT) sector globally [1], which grew from USD 2.67 trillion in 2006 to USD 5.06 trillion in 2019, with a projection to reach USD 5.82 trillion in revenue in 2023 [2]. During these years, the world entered the 4th Industrial Revolution [3], and technological breakthroughs unleashed capabilities that disrupted existing economic models, creating new trends such as machine learning, artificial intelligence, Internet of Things, big data, blockchain, assistive technologies, digital identity systems, and robotic process automation [4,5], among others. In today's world, the choice of a software development methodology has major concerns and often requires combinations between development approaches and infrastructure models.

In response to the need for faster software systems delivery, conventional plan-based methodologies were gradually replaced by lighter agile methodologies [6], beginning in the mid-1990s. However, from the beginning of the 2000s, initial enthusiasm [7] was followed by concerns regarding the implementation and inconveniences inherent in agile methodologies. These concerns were restrained with the rise of hybrid software development methods that combined elements from both the waterfall and agile methodologies to increase efficiency throughout the software development lifecycle. Nevertheless, hybrid methods are highly individualized [6,8] by project teams, raising the challenge of making the appropriate combination of elements that serve to reach team goals and assure value creation to organizations when adopted.

Together with the constant need for faster software systems delivery, information technology (IT) infrastructure requires scalability, mobility, connectivity, and elasticity to accommodate software evolutions' radical changes and assure operations efficiency. Soon after the "*Manifesto for Agile Software Development*" in 2001 [9], technology also manifested a trend of migration from specialized systems to dedicated services, becoming platform-

and hardware-independent. The need for creating virtual machines acting as real computers led to the development of modern cloud computing and served to create hybrid IT environments that consist of on-premises and on-demand systems. This resulted in the conceptualization of IT governance with the enterprise governance of IT (EGIT), with its first release in 2003 [10], scoped to provide an assessment framework for auditing evolving hybrid IT environments.

Business success in the 21st century, the era of digital transformation, depends heavily on embracing change and sustained outcomes [11], aligned to broader organizational needs that connect the dots between innovation, digitization, data analytics, artificial intelligence, and faster decision-making. From the technology firms' viewpoint, seeking continuous improvement relies on the establishment of change principles that empower teams to constantly re-imagine work and re-define software development processes, beyond blueprints and on a project-by-project basis [12]. The need for embracing these technical, process, data, and organizational changes possesses substantial risks to organizations. To anticipate these risks, organizations monitor and control them by running information systems audits to verify that business and IT risks are aligned with the IT strategy.

According to the Information Systems Audit and Control Association (ISACA), software generates value by alignment and by impact [13]. Specifically, value is acknowledged from the support to ongoing business operations, while the impact is recognized from changes in the way work is carried out and in cost structures. The need for systematic investigation arises, regarding the conformance of hybrid software development methods to IS audit standards, while organizations create and adopt such dynamic and evolving models.

The structure of this paper includes a detailed classification of the main software development methodologies in Section 2, separating traditional heavyweight from lighter agile, examining the ground on which hybrid software development methods appear and their common characteristics. Section 3 follows with an investigation of the challenges arising regarding IS auditing during the evolution of software development methodologies. Section 4 includes a discussion of findings and recommendations for future research, and Section 5 summarizes the conclusions.

2. Classification of Software Development Methodologies

Surprisingly, software development methodologies can be approached and classified from various viewpoints; however, the term "methodology", according to Avison and Fitzgerald [14], is rather vague and the meaning is not commonly acknowledged. Nevertheless, there is consensus that methodologies can be classified based on tasks and processes followed during software development. Often, these methodologies are referred to as software development life cycles (SDLCs). In principle, each SDLC is a conceptual framework that includes all software development stages, from idea inception to implementation, deployment, maintenance, and finally, retirement.

While SDLCs consist of the above common stages, different ideas for effectiveness and efficiency of managing the SDLC create a plethora of different approaches that evolve over time. Moreover, every software development project is different from every other and thus has a different life cycle. However, there are certain common characteristics between all established SDLCs: the decomposition of the problem domain into sub-problems, the analysis of each sub-problem, and the synthesis of the solution to provide a holistic approach.

Figure 1 shows the evolution of software technology, which has caused rapid advancements in sciences and radical changes in the daily life of humanity, over the last fifty years. The software development methodologies (SDM) evolution timeline is indicative of the modern world's change in needs. These changing societal needs result in the necessity for the supply of more sophisticated software products and services. From the SDMs' viewpoint, the ability to deliver cutting-edge software solutions is a result of the transition from traditional heavyweight to more flexible agile methodologies and hybrid methods by combining elements from both the traditional and agile approaches.

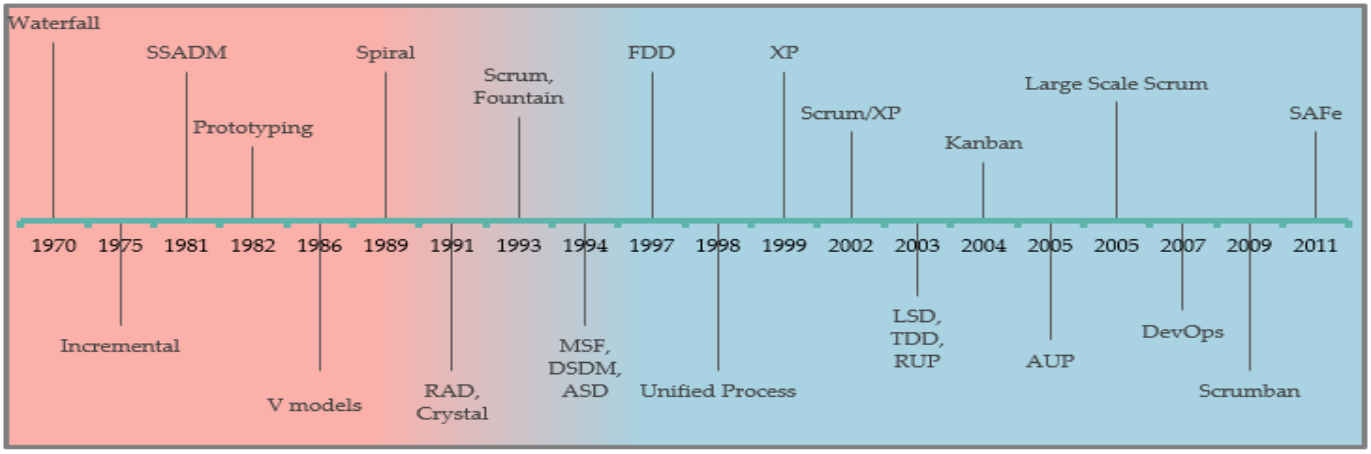

Figure 1. Software development methodologies timeline.

The evolution of SDLCs is indicative of the need for managing the constant increase of software product and service complexity. On the one hand, traditional software development methodologies are considered structured and plan-driven, following phases over a sequence with dependences. Moreover, traditional software development methodologies include the provision of detailed documentation during the software development lifecycle. The documentation includes detailed functional and non-functional requirements, design, coding, testing, and deployment plans.

On the other hand, agile software development methodologies evolved due to the need for software development without clear initial requirements, adapting to emerging needs and changes during the software development lifecycle. In contrast to traditional, plan-driven methodologies, these lighter and significantly more flexible methodologies focus on collaboration between business users and developers, eliminating the need for detailed plans, documentation, and strict processes. In this way, the focus shifts to the delivery of potentially working software frequently and iteratively, rather than at the end as a matter of linear development.

2.1. Traditional Software Development Methodologies

In the following sections, the traditional, or heavyweight, methodologies are described, with their main advantages and disadvantages.

2.1.1. Waterfall Model

The Waterfall model is the fundamental plan-driven software development methodology, proposed by Royce in 1970 [15]. Its profound success and ongoing use are due to its structure-phased approach, based on which development teams follow a sequence of predefined steps, as shown in Figure 2. The origin of the name is derived from the cascade progression from one phase to another once it is completed.

Main advantages of the Waterfall model [15–18]:

- Simple and easy to understand and follow.
- Linear advancement and completion of each level for proceeding to the next one.
- Clear determination of goals.
- Detailed plan and documentation that empowers communication between peers.

Main disadvantages of the Waterfall model [15,19]:

- Difficult to establish changes that were not incorporated in the analysis and design.
- Uncertainty at the beginning of development.
- Software is delivered at the end.
- Testing takes place at the end.
- Users are not actively involved.

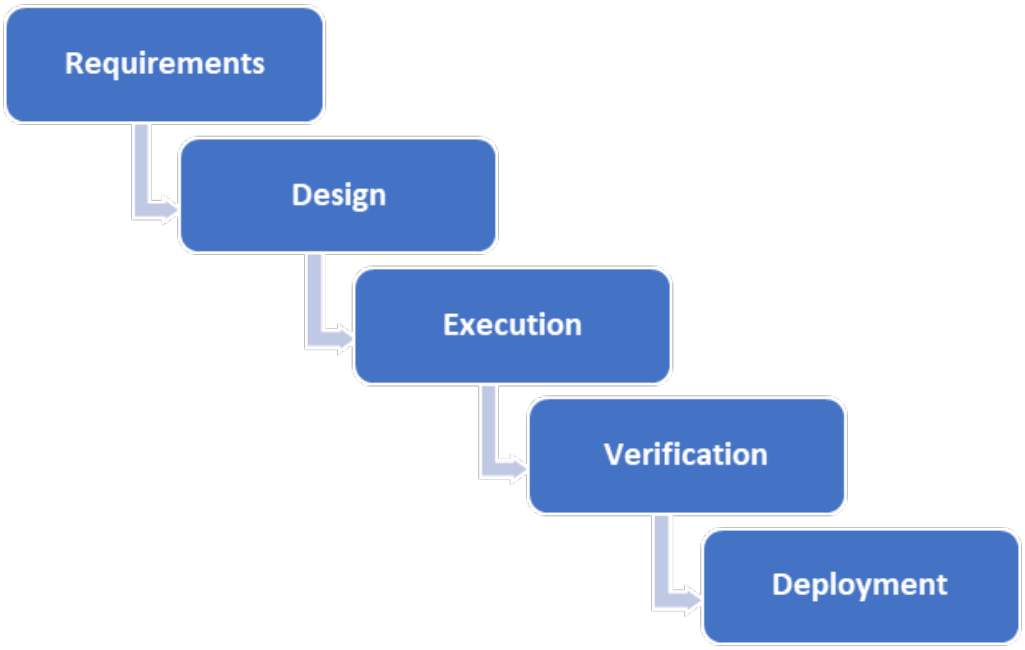

Figure 2. Waterfall model. Source: adapted from [15].

2.1.2. Incremental Model

The Incremental model is an alternative to the Waterfall model, proposed by Basili and Turner in 1975 [20,21]. The key idea of this model is that, while requirements are still collected and listed upfront, development is decomposed into smaller components of functionality, and the team proceeds with stepwise refinement, as shown in Figure 3. The team plots the increments into a delivery plan, considering prerequisites and dependencies of the increments, value to cost score ("*V to C*" [22]), purchasing plans, and requirement updates from the customer. Notably, each one of the increments goes through the levels of analysis, design, code, and test. The Incremental model was the groundwork for the Prototyping model that emerged in the following years with the use of software tools and technics.

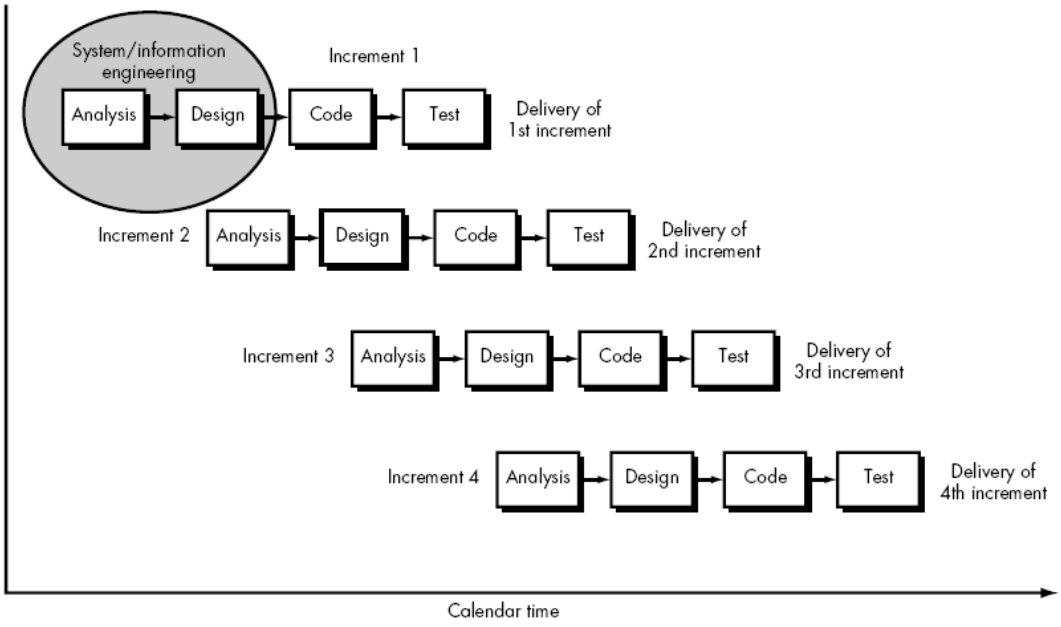

Figure 3. Incremental model. Source: adapted from [23].

Main advantages of the Incremental model [20,21]:

- Increased flexibility and adaptability.
- Enhanced quality, due to code tests occurring on each increment.
- Improved code reliability.

Main disadvantages of Incremental model [20,21,24]:

- Need for resources to be committed for extensive periods.
- Requires strong change management control processes.
- Requires that both problem and solution are well-understood.

2.1.3. Structured Systems Analysis and Design Methodology (SSADM)

The Structured Systems Analysis and Design Methodology (SSADM) was introduced in 1981 as a standardization approach by Learmonth Burchett Management Systems (LBMS) and the Central Computer Telecommunications Agency (CCTA) for managing governmental software development projects in the United Kingdom [25]. SSADM has been formally specified in British Standard BS7738. The main reason for deriving SSADM was the limited interaction capability between users and the software development team in the Waterfall model. Moreover, requirements are difficult to get modified once set and agreed, while the software development process is driven bottom-up. As such, software is delivered at the end and requires effort and resources for rebuilding, as shown in Figure 4. To anticipate these issues, SSADM introduces a concept of six stages [25]: Survey, Structural Analysis, Structured Design, Configuring Hardware Study, System Construction, and Maintenance.

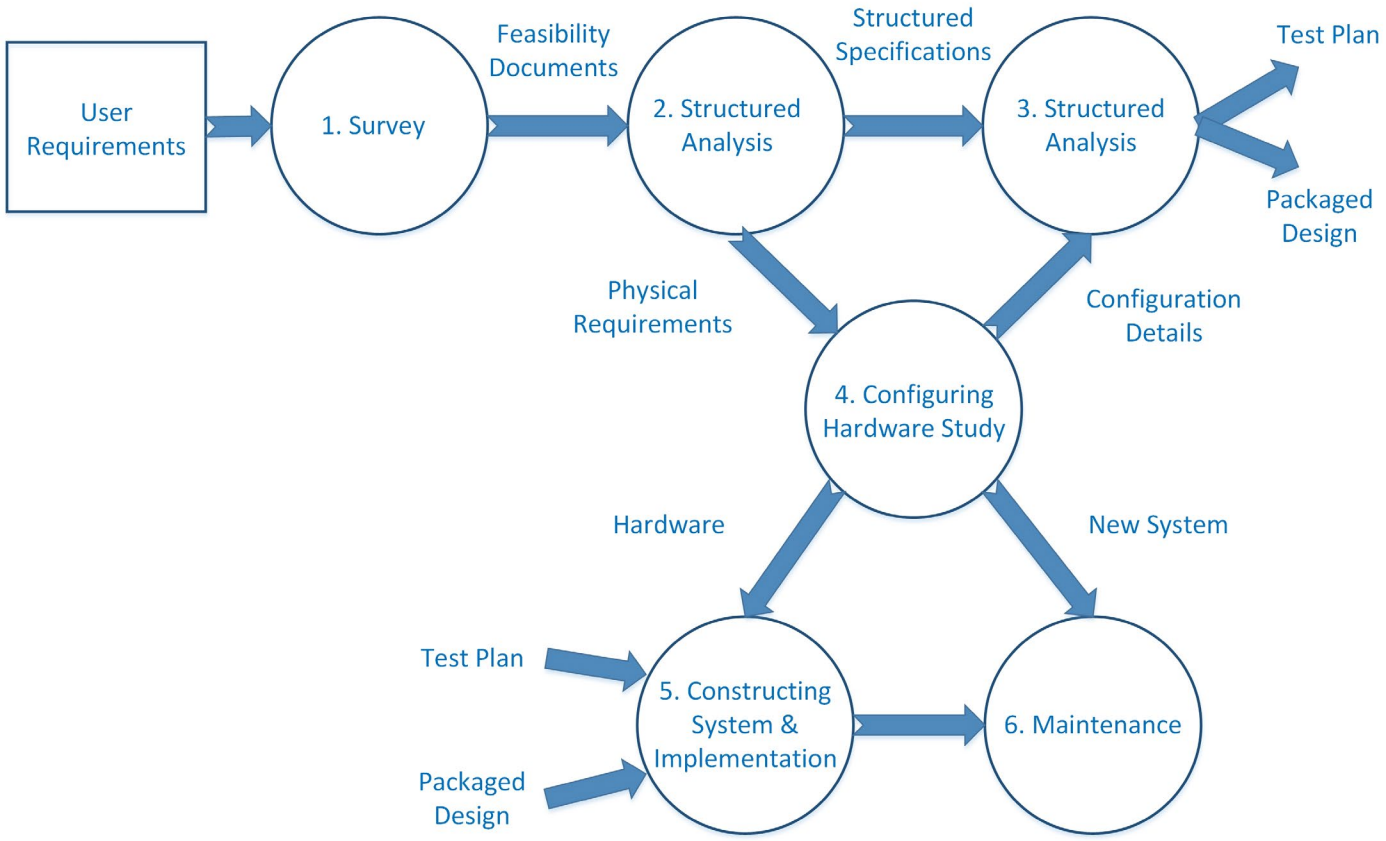

Figure 4. Structured Systems Analysis and Design Methodology (SSADM). Source: adapted from [25].

Main advantages of SSADM [25,26]:

- Structured methodology that offers room for better communication between peers and high visibility throughout the generated documentation.
- Enhanced flexibility and modularization.

- Rigid control throughout the life cycle.

Main disadvantages of SSADM [25,26]:

- Schedule flaws, due to high attention paid to the analysis.
- Requires capital to be implemented and maintained.
- Practically inefficient to be applied in smaller projects.

2.1.4. Prototyping Methodology

The concept behind the inception of the Prototyping software development methodology, first noted in 1982 by Newman and Jenkins [27], relies on the need to simultaneously derive functional requirements alongside physical design specifications [28,29]. In the Prototyping model, as shown in Figure 5, the development team produces prototypes for demonstration purposes in various forms, such as paper drawings and mock-up screens, as well as working elements with limited functionality. Once these prototypes are created, the software development life cycle follows two primary variations. It either adapts the "*throw-away*" approach, where unneeded functionality is discarded, or the "*partial keep*" approach, based on which specific elements are retained until reaching the final design concept.

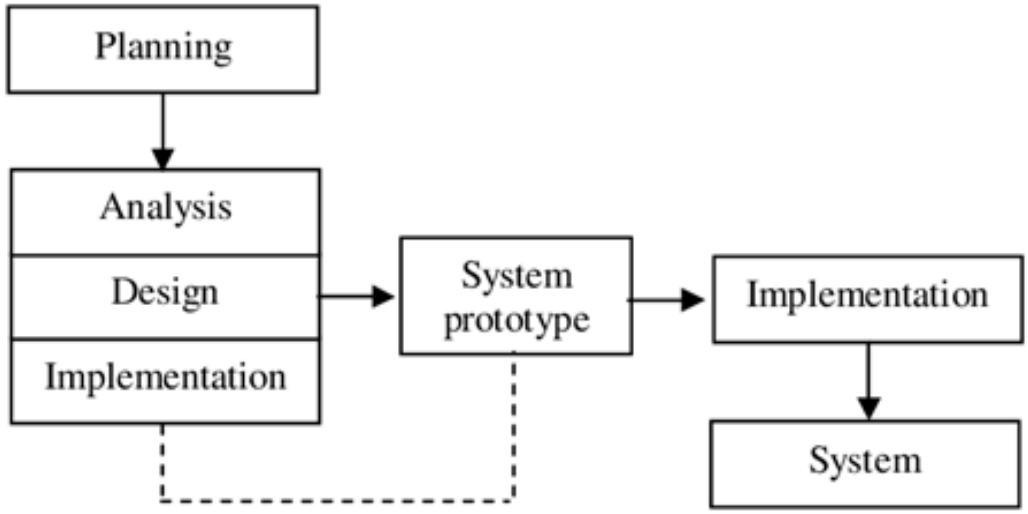

Figure 5. Prototyping model. Source: Adapted from [28].

Main advantages of the Prototyping model [28–30]:

- Active involvement by users.
- Early detection of missing functionality.
- Risk of project failure is relatively reduced.
- Overall cost efficiency, due to early detection of errors and missing functionality.

Main disadvantages of the Prototyping model [29,30]:

- Time consumption for prototype building before proceeding to final product development.
- Upfront costs for building prototypes, which are eventually leveled with saving capital from the final product development.
- Insufficient to produce detailed functional specifications documents for each prototype.

2.1.5. V-Model variations

V-models represent software development life-cycle methodologies that are considered extensions of the Waterfall model. In contrast to a horizontal linear advancement of levels, Boehm [31] published a paper in 1979 introducing the V-model (also known as "*Vee*"), based on which process execution occurs in a V-shape. During the following years, two variations of the V-model appeared: the classic-V and the double-V, as described in the following section.

V-Model

This methodology, also known as the "*Systems Engineering Vee*" [31–33], emphasizes the importance of verification and validation activities from the early stages of SDLC. As an advancement of the Waterfall model, the V-model imposes process life-cycle management on the elements of each level, as shown in Figure 6.

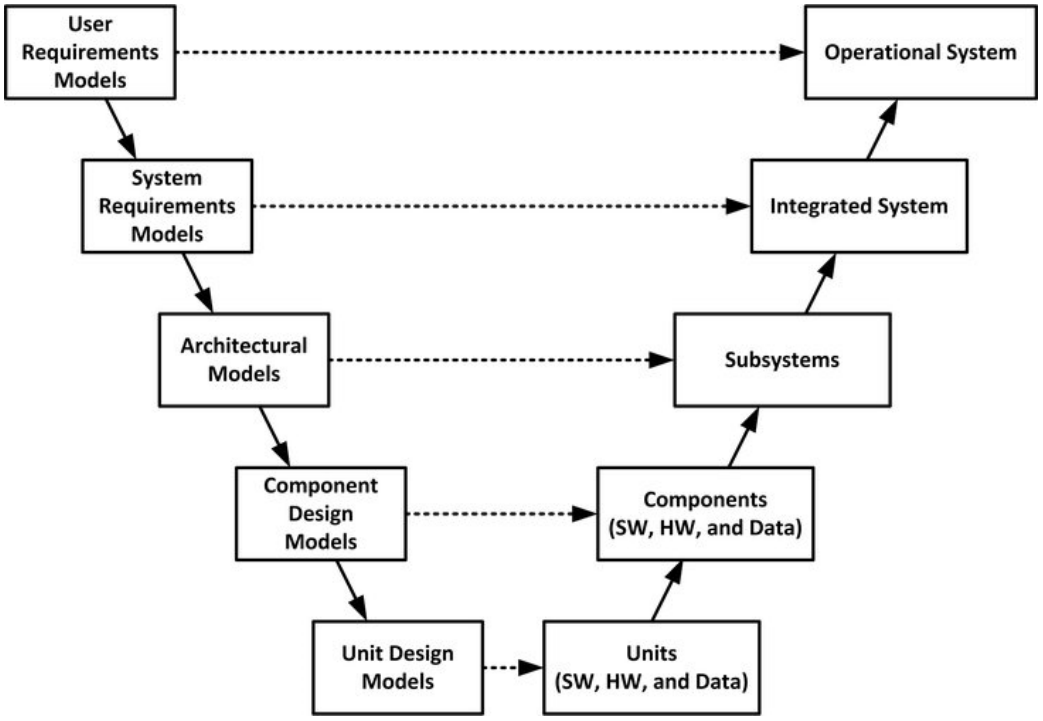

Figure 6. V-model. Source: adapted from [33].

Main advantages of the V-model [31–34]:

- Overall simplicity that enables easy adoption by teams and onboarding of new team members.
- Straightforward to follow and efficient when requirements are stable.
- Increased transparency by specific deliverables expected in each phase.

Main disadvantages of the V-model [31–34]:

- Poor flexibility to manage ongoing changes in requirements.
- Software is delivered at the end of the deployment and in the absence of prototypes or other artifacts that may eliminate user acceptance failure risks.
- Requires high technical expertise of the team.

VV-Model

While the V-model emphasizes the connection of the documentation on the left-hand side to testing on the right-hand side, there are two main issues that arose that led to the creation of the VV-model [33], or "*Double V-Model*", as shown in Figure 7. First, the direct connection of documentation with testing activities is found to be rather imperfect in practice, considering that functional specifications often lack sufficient information towards system testing. Moreover, system testing depends on the architecture and the technical designs, on top of the functional specifications. The information contained in the functional specifications document serves to verify and agree with business users on the functionality of the software that is being delivered.

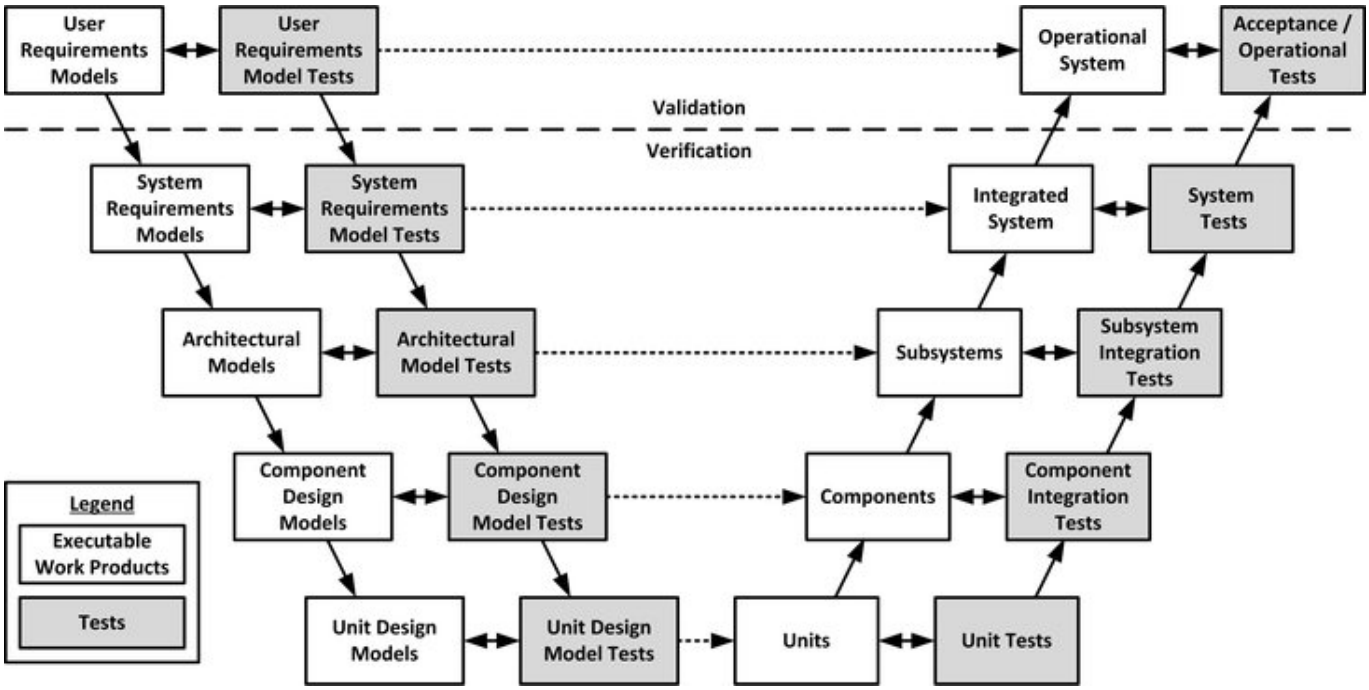

Figure 7. VV-model (or double V-model). Source: Adapted from [33].

Main advantages of VV-model [33,34]:

- Closer connection between development and testing activities occurring in each phase.
- Establishes a high level of collaboration and responsibility between development and testing teams.
- High discipline model.
- Main disadvantages of VV-model [33,34]:
- Increased complexity, which usually leads to the adoption of the V-model instead of the VV-model.
- Requires classification of critical application tests to secure efficient resource allocation.
- Demanding on budget and time.

2.1.6. Spiral Model

A decade after the introduction of the V-model, Boehm elaborated on his initial idea in 1988, suggesting an improvement with the Spiral software development methodology [35,36]. After several years of evolution of and experience with the Waterfall model and the V-model, he introduced a new model, as shown in Figure 8, dictated level advancement on a spiral, in contrast to a straight horizontal line or V-shape.

Main advantages of the Spiral model [37,38]:

- Risk-driven approach.
- Provides a viable framework for integrated software–hardware integration.
- Early estimation of cost (radius of the spiral).
- Risk management that comparatively eliminates customer satisfaction issues.

Main disadvantages of the Spiral model [37,38]:

- Rule and protocol strictness that must be effectively implemented
- Considerably high complexity of SDLC processes.
- Amount of documentation due to several intermediate stages.
- End date is difficult to calculate at the beginning.
- Not suitable for small and considerably simple projects.

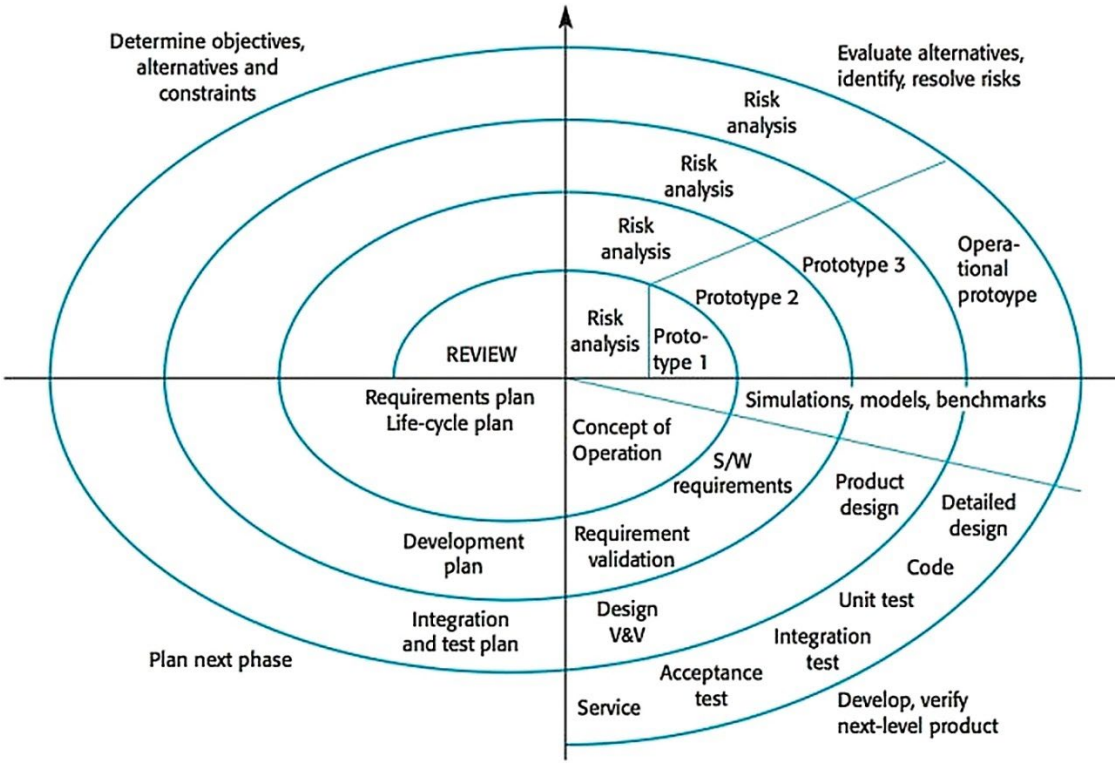

Figure 8. Spiral model. Source: Adapted from [36].

2.1.7. Fountain Model

The Fountain model was proposed by Henderson and Edwards in 1993 [39,40], as a way to represent highly iterative SDLCs. The model provides relative freedom for advancement from one phase to another, regardless of full task completion at each phase. This model uses the metaphor of the water flow at a fountain as shown in Figure 9, whereas in Fountain software development, the analysis flow is superimposed upwards to design and implementation, creating the so-called software pool. This highly repetitive method is a preliminary approach to object-oriented software development, with the objects occurring being stable elements that can be identified early in the life cycle.

Main advantages of the Fountain model [40,41]:

- Enables incremental and iterative software development.
- Allows overlapping of activities between phases.
- Enhanced flexibility in changing requirements.

Main disadvantages of the Fountain model [42,43]:

- Limited documentation production throughout the SDLC.
- Demanding in resources to facilitate overlapping development.
- Limited risk management due to object reuse generalization.

This highly repetitive method is a preliminary approach to object-oriented software development, with the objects occurring being stable elements that can be identified early in the life cycle.

2.1.8. Rapid Application Development (RAD)

The use of existing traditional sequential methods, following the horizontal-line precedents of phases, V-shape, and spiral, may have offered structured approaches for software development; however, requirements change during the project's lifecycle, especially in large projects. An impactful solution was proposed by James Martin in 1991, with his publication describing Rapid Application Development (RAD) [44,45]. The core element of RAD and main innovation, as shown in Figure 10, is that coding relies on iterative feedback

by the user, rather than on detailed requirement sheets that are elicited at the beginning of the project, where visibility is usually limited.

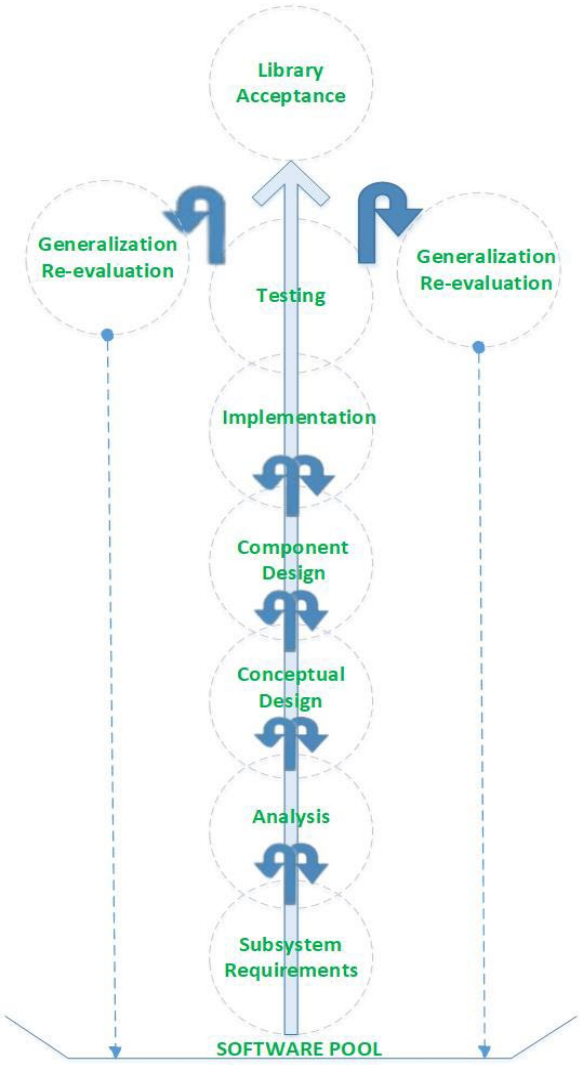

Figure 9. Fountain software development model. Source: Adapted from [41].

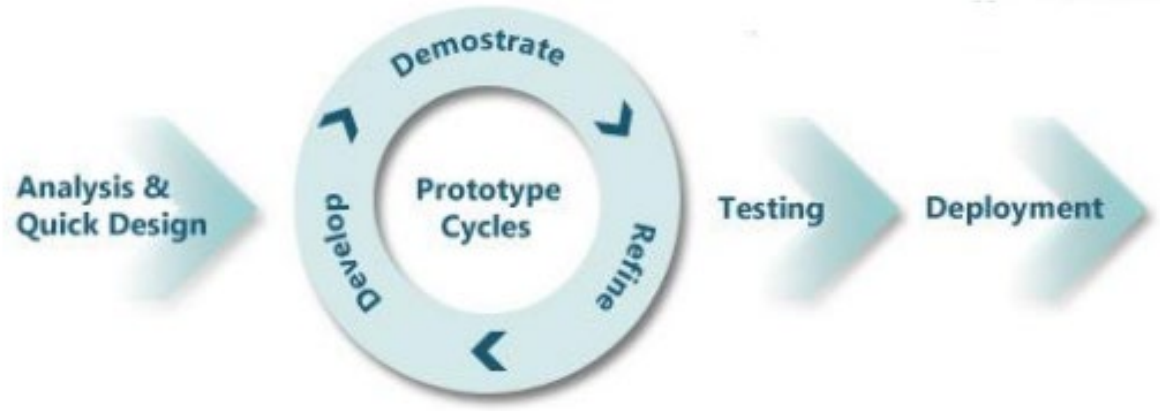

Figure 10. Phases in the James Martin approach to RAD. Source: Adapted from [45].

Main advantages of Rapid Application Development [46,47]:

- High speed of product delivery.

- Increased quality, due to users' involvement in analysis and design.
- Enhanced flexibility.
- Overall risk management.

Main disadvantages of Rapid Application Development [46,47]:

- Scalability when projects expand and require inter-team communications.
- Front-end development focus that undergoes back-end best practices.
- Success depends on highly skilled and experienced developer teams.
- Requires commitment from stakeholders, which in large enterprises is a matter of scheduling conflicts between senior managers.

2.1.9. Unified Process (UP) and Rational Unified Process (RUP)

Object-oriented methodologies for software development specifically aim to model and implement software as a collection of interacting objects. In practice, this takes place using specialized modeling languages, such as the Unified Modeling Language (UML) [48–50], activities, and techniques needed to address specific issues of the object-oriented paradigm. Notably, an object-oriented programming (OOM) methodology is considered as a programming paradigm based on objects that incorporate reusability and modularity.

The Unified Process (UP) model was proposed during the transition from the procedural programming of the 1980s to object-oriented, use-case-driven, architecture-centric approaches. It was documented in 1999 by Jacobson et al. [50] and is the basis of Rational Unified Process (RUP), a process/product created and marketed by Rational Software Corporation. UP incorporates best practices for software development teams based on previous experience gained by utilization within successful organizations, as shown in Figure 11.

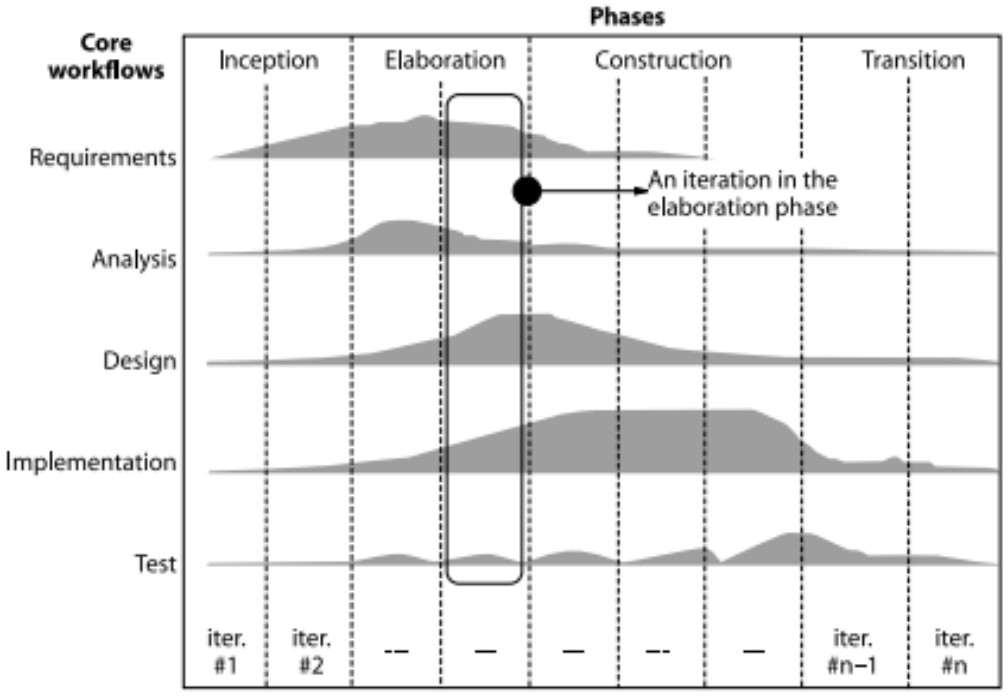

Figure 11. The Unified Process (UP) model. Source: Adapted from [50].

Main advantages of the Unified Process model [51,52]:

- Empowers team collaboration.
- Capability for incorporating changes, both from customer and development team.
- A framework for unifying software processes within organizations, with the use of UML diagrams for team collaboration.

- Suitable for both small and large projects.

Main disadvantages of the Unified Process model [51,52]:

- High complexity.
- Requires intimate familiarization.
- No working software until the end of coding, thus risks regarding meeting customer expectations.
- Integration throughout the SDLC.

2.1.10. Microsoft Solution Framework (MSF)

Microsoft Solution Framework [53–55], first introduced as version 1.0 in 1993, specifies in detail various aspects of the project, such as its phases and milestones, project team, and task assignments. In this methodology, special attention is paid to project preparation and risk management. The philosophy of MSF relies on learning from all experiences, as shown in Figure 12. MSF is considered ideal for imposing agility in projects where a higher level of structure is required.

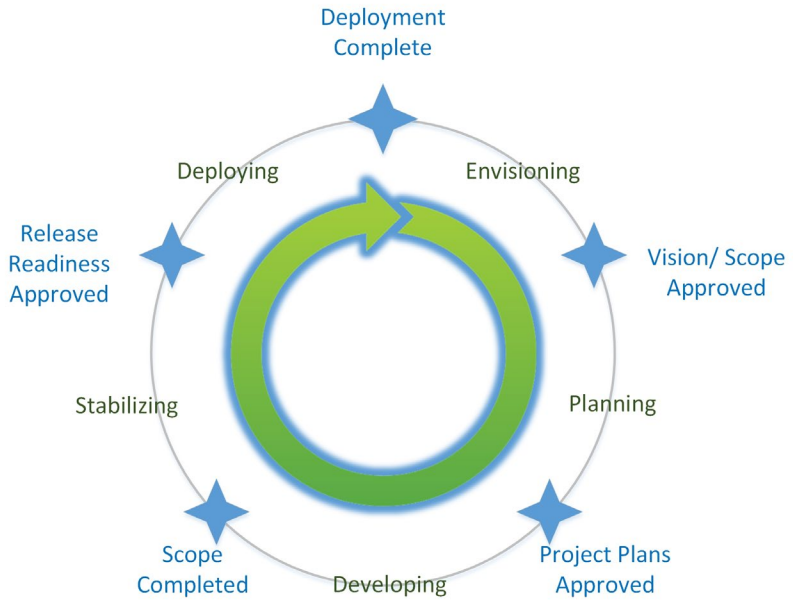

Figure 12. Microsoft Solutions Framework (MSF). Source: Adapted from [53].

Main advantages of Microsoft Solutions Framework [54,56]:

- Enhanced speed and flexibility.
- Increased risk management.
- Shared responsibility.

Main disadvantages of Microsoft Solutions Framework [57]:

- Complexity in the construction process.
- Increased administration needs.

2.1.11. Additional Heavyweight Methodologies in the Literature

Supplementary to the already visited, main traditional software development methodologies, a list follows with other methodologies found in the literature.

- *Bing-Bang* [58]: No document or process followed.
- *Code and Fix* [59]: Also called "*Cowboy coding*", a two-face model where software engineers code in 1st phase and fix on the 2nd, until satisfying the customer.
- *Sashimi model* [60]: A waterfall model with an option for overlaying development phases.
- *Sawtooth model* [61]: A variation of the V-model, incorporating prototyping.
- *Ropes model* [62]: Rapid object-oriented process for embedded systems.

- *Parallel model* [63]: Parallel concurrent development occurs when different versions of an object are developed simultaneously.
- *WINWIN model* [64]: Extends the spiral model by adding Theory W activities ("*make everyone a winner*") to the front of each cycle.
- *Component-based model* [65]: Identifying and reusing already existing components.
- *Architecture-based model* [66]: Approaching software design in terms of major design elements and their relationships among them.
- *Intelligent model* [67]: Also called the "*knowledge-based software development model*", a combination of the Waterfall model with an expert system to integrate knowledge.

2.2. Agile Software Development

Agile software development methodologies stand as a flexible lightweight approach that gained significant popularity with the release of the "*Manifesto for Agile Software Development*" in 2001 [9]. That year, Cockburn invited 16 reputable software engineering professionals and independent thinkers to a ski resort in Utah, USA, to elaborate on ideas about better ways of developing software. These ideas were expressed via the Manifesto, highlighting the agile principles as follows:

- "*Individuals and interactions over processes and tools*".
- "*Working software over comprehensive documentation*".
- "*Customer collaboration over contract negotiation*".
- "*Responding to change over following a plan*".

The Agile Manifesto not only had significant impact in the software development community, but would be applied in the following years across industries and business contexts through agile transformation, as noted by Boston Consulting Group [68]. In years since, scientists have approached a definition for agile; however, the term itself remains vague. According to IEEE Computer 2003's paper by Williams and Cockburn [69], agile is defined by feedback and change. From another viewpoint, McCauley [70] claims that agile is defined by speed and simplicity, while Schuh [71] outlines agility as a way of building software by empowering people, and Ambler [72] considers agility as an iterative and incremental approach performed by highly collaborative, self-organized teams.

The next section summarizes findings of agile software methodologies, including Crystal family, extreme programming (XP), Scrum, Scrum/XP hybrid, Large-Scale Scrum (LeSS) Lean, Kanban, Scrumban, Agile Unified Process (AUP), adaptive software development (ASD), feature-driven development (FDD), test-driven development (TSD), dynamic systems development (DSD), DevOps and scaled agile framework (SAFe).

2.2.1. Crystal Family

An early approach to modern agile software development was introduced by Cockburn in 1991 [73–75] with the Crystal family of methodologies, as shown in Figure 13. This is the result of years of study and team interviews after he noticed results that were still successful even though formal methodologies had not been followed. In his publication, Cockburn classified what teams followed that led software development projects to success, dividing the Crystal family of methodologies into colored categories.

Main advantages of the Crystal family of methodologies [76,77]:

- Elaboration through feedback directly from users.
- Improved team communications.
- Frequent deliveries of software.

Main disadvantages of the Crystal family methodologies [76,77]:

- Minimum documentation approach.
- Risk for scope creep due to lack of pre-defined plans.

Figure 13. Crystal family of methodologies. Source: adapted from [75].

2.2.2. Adaptive Software Development (ASD)

Adaptive software development (ASD) was introduced by John Highsmith and Sam Bayer in 1994 [78]. The ASD model has roots in RAD, suggesting the creation of software incrementally and iteratively with constant prototyping, as shown in Figure 14. This takes place by emphasizing results rather than specifying tasks expertise.

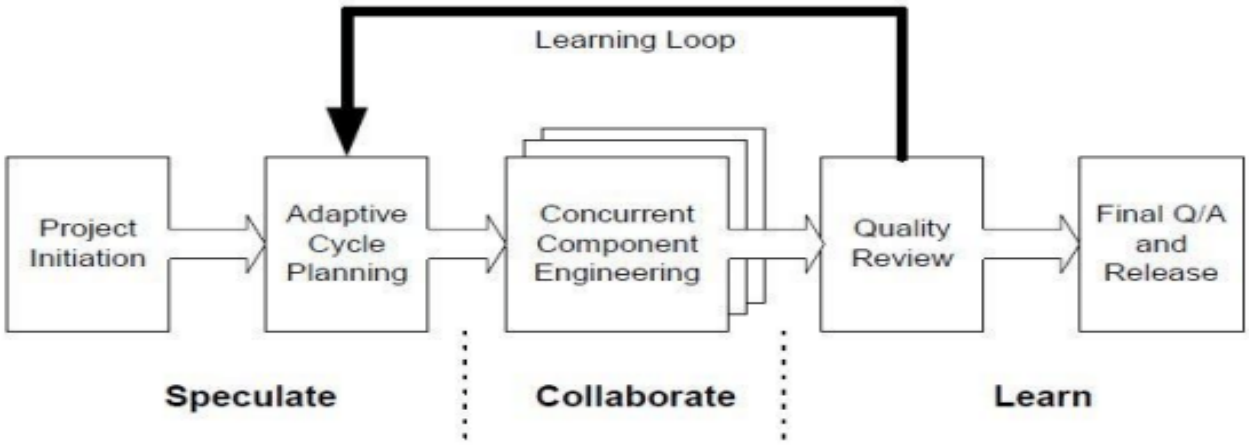

Figure 14. Adaptive Software Development. Source: adapted from [79].

Main advantages of Adaptive Software Development [80,81]:

- Strong collaboration between the developers and the customer.
- High visibility on product and progress.
- Low risk for project completion delays.

Main disadvantages of Adaptive Software Development [80,81]:

- Iterative testing imposes costs on the project's life cycle.
- Engagement of resources in a wider timely manner for iterations.
- Difficult to scale or work in parallel project streams.

2.2.3. Scrum

Scrum is considered one of the most commonly used frameworks for agile software development [82]. It was jointly introduced in 1995 by Jeff Sutherland and Ken Schwaber in the paper "*The SCRUM Development Process*" [83]. The authors were among the 17 software engineering professionals that co-signed "*The Agile Manifesto*", in 2001 [9]. The first detailed

Scrum guide was published in 2010, clarifying what Scrum is. The core philosophy of Scrum is managing software development in a volatile environment, through flexibility and adaptability, as shown in Figure 15.

Figure 15. The Scrum development process. Source: adapted from [84].

Unlike conventional methodologies, Scrum stands as a lightweight framework that provides room for the development team to become collaboratively self-organized, choosing their tools and techniques to deliver software with the highest possible value.

Main advantages of Scrum [85–88]:

- Team obtains the full idea about the product before development.
- Acceptance of changes at any time during the development.
- Quick development of software.
- Able to develop software according to the priority of the requirements.
- Schedule never changes.
- Teams are self-managed.

Main disadvantages of Scrum [87,88]:

- Requires major culture transformations within organizations to be fully adopted.
- Teams need training and accumulated experience to become efficient.
- Detailed estimates for scope, budget, and time are limited to the sprint level.
- Documentation is limited.

2.2.4. Dynamic Systems Development Method (DSDM)

The dynamic systems development method (DSDM) was introduced in 1994 by the Agile Business Consortium [89,90] and was scoped to develop and promote an independent RAD framework. The framework focuses on frequent delivery and tight collaboration with the user; however, what distinguishes DSDM from other methodologies is the strictness over cost and schedules. Interestingly, it brings the well-known 80/20 Pareto principle [91] into software development. According to this, DSDM highlights [92] that 20% of the development efforts should be reflected in 80% of the delivered software increments.

DSDM has three discrete phases, as shown in Figure 16: pre-project, project, and post-project. The initial phase of the pre-project includes verification of the business case, go/no-go decision, product objectives, and budget allocation confirmation so that project can be initiated. The project phase consists of five sub-stages, combining sequential, iterative, and incremental ways of work [89,91]. The first two sub-phases, Feasibility Study and Business Study, are considered sequential. These are followed by iterative and incremental Functional Model Design Iteration, Design and Build Iteration, and Implementation. Finally, during the post-project phase, the team focuses on enhancing fixes to maintain effectiveness and efficiency.

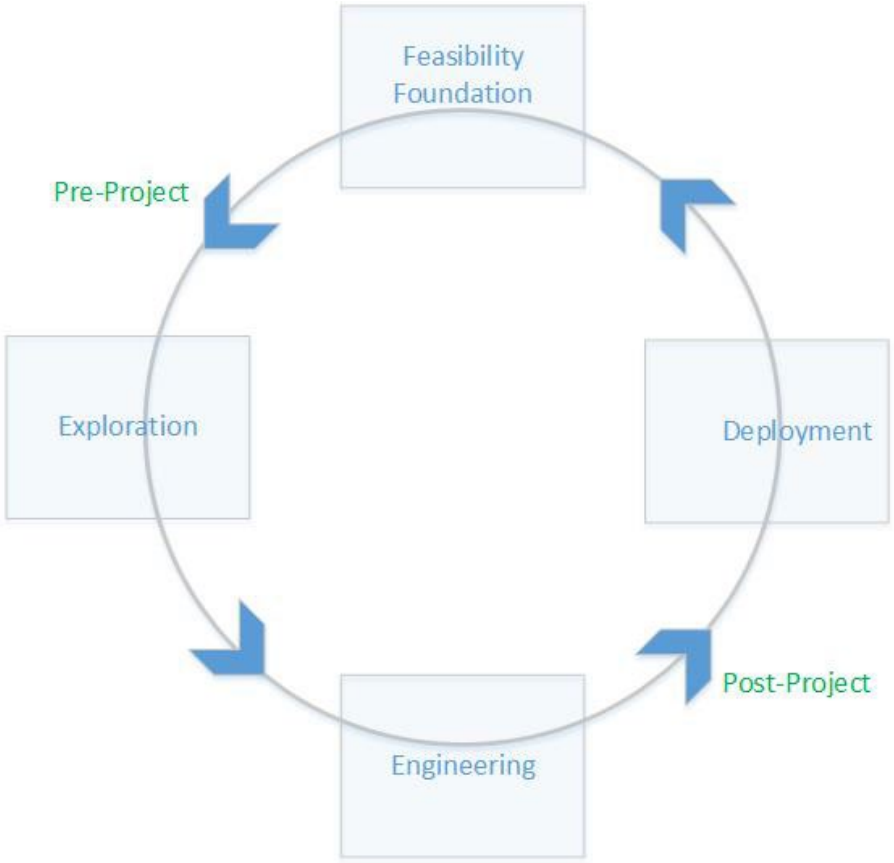

Figure 16. The dynamic system development method. Source: adapted from [92].

Main advantages of the dynamic systems development method [93].

- High user involvement.
- Increased visibility during project elaboration.
- Quick delivery (80% of software in 20% of development time).
- Tight schedule and budget control.

Main disadvantages of dynamic systems development method [93].

- Requires skilled and experienced personnel.
- Comparatively limited freedom provided to developers for creativity.
- Occurrence of management overheads.
- Difficult to be adapted for small teams.

2.2.5. Feature-Driven Development (FDD)

As the practice of agile software development methodologies increased in the early 2000s, numerous frameworks were suggested, among which feature-driven development become notable in 1997 by Jeff De Luca and Peter Coad [94], who opted for software development based on making progress around features. The initial ideas of FDD were

developed further by Stephen Palmer, who established and published his elaborated ideas in 2002, stating that, starting from an overall model, the team moves on by "designing by feature-building by feature" [95]. The five steps of FDD are described in Figure 17. The model is a combination of the upfront design in the initial, overall model that elaborates further, iteratively and incrementally. Notably, while the overall model splits into features, each feature then breaks down into smaller functionality elements that can be delivered within a maximum of two weeks.

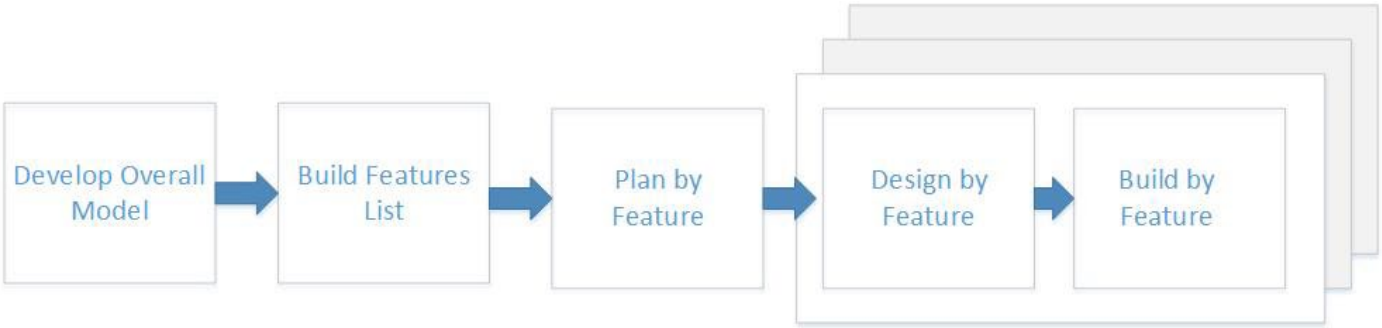

Figure 17. Feature-driven development. Source: adapted from [96].

Main advantages of feature-driven development [97,98]:

- Fast delivery, with short iterations.
- Usually favored by clients due to tangible and frequent results.
- Emphasizes quality at all steps.
- Provides high visibility of progress.

Main disadvantages of feature-driven development [98,99]:

- Strong dependency on the chief programmer, who selects the features and mentors the team.
- Limited documentation.
- Unfavorable for small projects.

2.2.6. Extreme Programming (XP)

Extreme programming [100,101] is a framework of the agile software development methodologies that focuses heavily on coding the most important features first, proceeding iteratively with little or no upfront design. XP found tremendous popularity after its publication by Beck in 1999 and was acknowledged as among the most important software development methodologies up to that date. Notably, XP relies on deep and constant interaction of the development team with the business users on each iteration, leveraging changes, testing, and continuously refactoring throughout the iterations, as shown in Figure 18.

Main advantages of extreme programming [100–103]:

- Code simplicity and maintainability.
- Enhanced flexibility.
- High interaction with users to meet customer expectations.
- Reduced project failure.
- Creates working software faster.

Main disadvantages of extreme programming [100–103]:

- Relatively costly.
- Lack of measuring code quality assurance.
- Provides better results when developers are collocated.

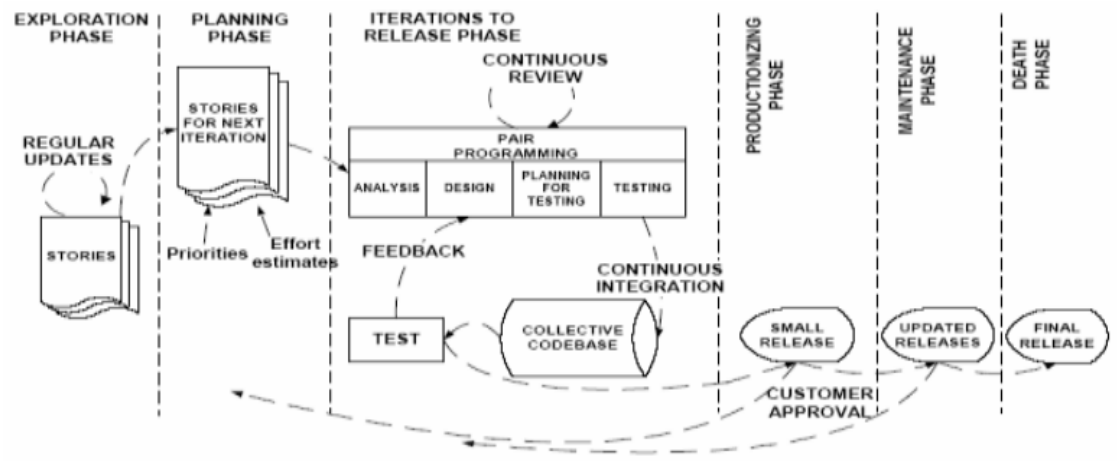

Figure 18. Extreme programming. Source: adapted from [101].

2.2.7. Test-Driven Development

The TDD model was introduced in 2003 by Kent Beck [104], proposing the concept of implementing automated tests before implementing and refactoring the code. These tests are run by tools that usually target a small part of the functionality each time so that the overall functionality is verified by the execution of the sum of the tests. Implementation of the tests requires algorithms that, upon execution, should provide the expected callbacks as positive feedback. As such, the philosophy of TDD relies on the principle of test-first and then code-refactoring for optimization.

Notably, although the model is based on testing, TDD is not considered a testing technique. This test-first philosophy considers automated tests as an essential analysis step, driving design and code implementation [105]. Moreover, executing tests early in the lifecycle makes testing as lightweight as possible, and the code is less likely to get degraded during development. In TDD, the development of a unit of code elaborates iteratively from customer's requirements to test cases, towards the creation of unit testing blocks. The code unit then, as shown in the lower part of Figure 19, goes through acceptance testing and unit testing, towards repeated revisions in the final stage of regression testing.

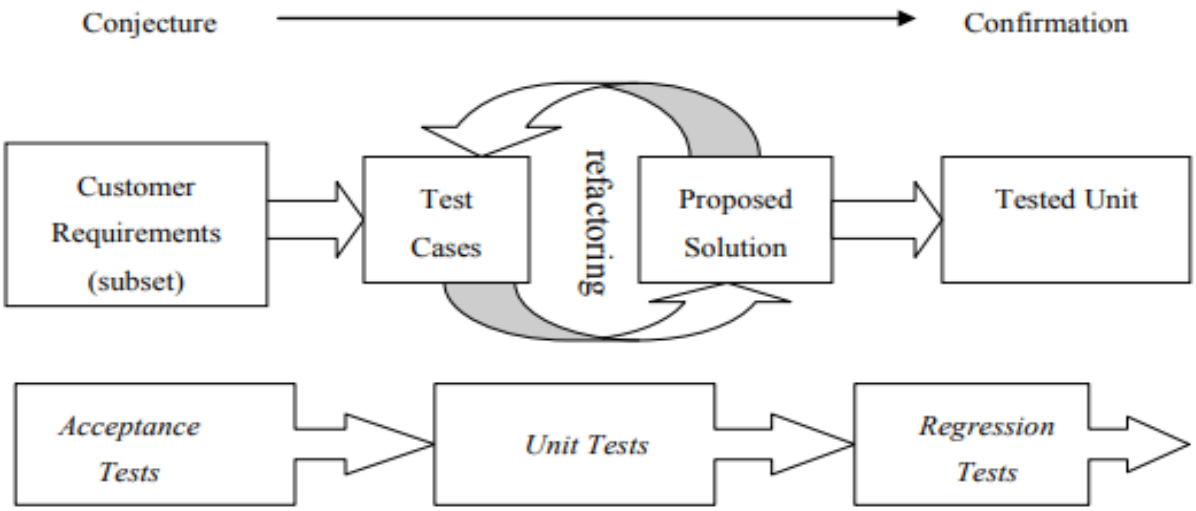

Figure 19. Test-driven development. Source: adapted from [105].

Main advantages of test-driven development [106,107]:

- High maintainability of code.
- Smooth refactoring that makes new feature additions easier to code.

- Cost-effectiveness, due to elimination of code errors that require rework.
- Main disadvantages of test-driven development [106,107]:
- Continuous testing may delay development progress.
- Requires mindset change, skills, and team training to learn and adjust to TDD.

2.2.8. Lean Software Development (LSD)

Lean software development (LSD) follows the concept of lean thinking shown in Figure 20, which was introduced in the mid-1950s by Toyota [108], the Japanese car manufacturing company. LSD model guides companies to standardize methodologies, activities, and work products by following some main principles [101,108]:

i. Eliminate waste.
ii. Amplify learning.
iii. Defer commitment.
iv. Deliver as fast as possible.
v. Empower the team.
vi. Build integrity in.
vii. See the whole.

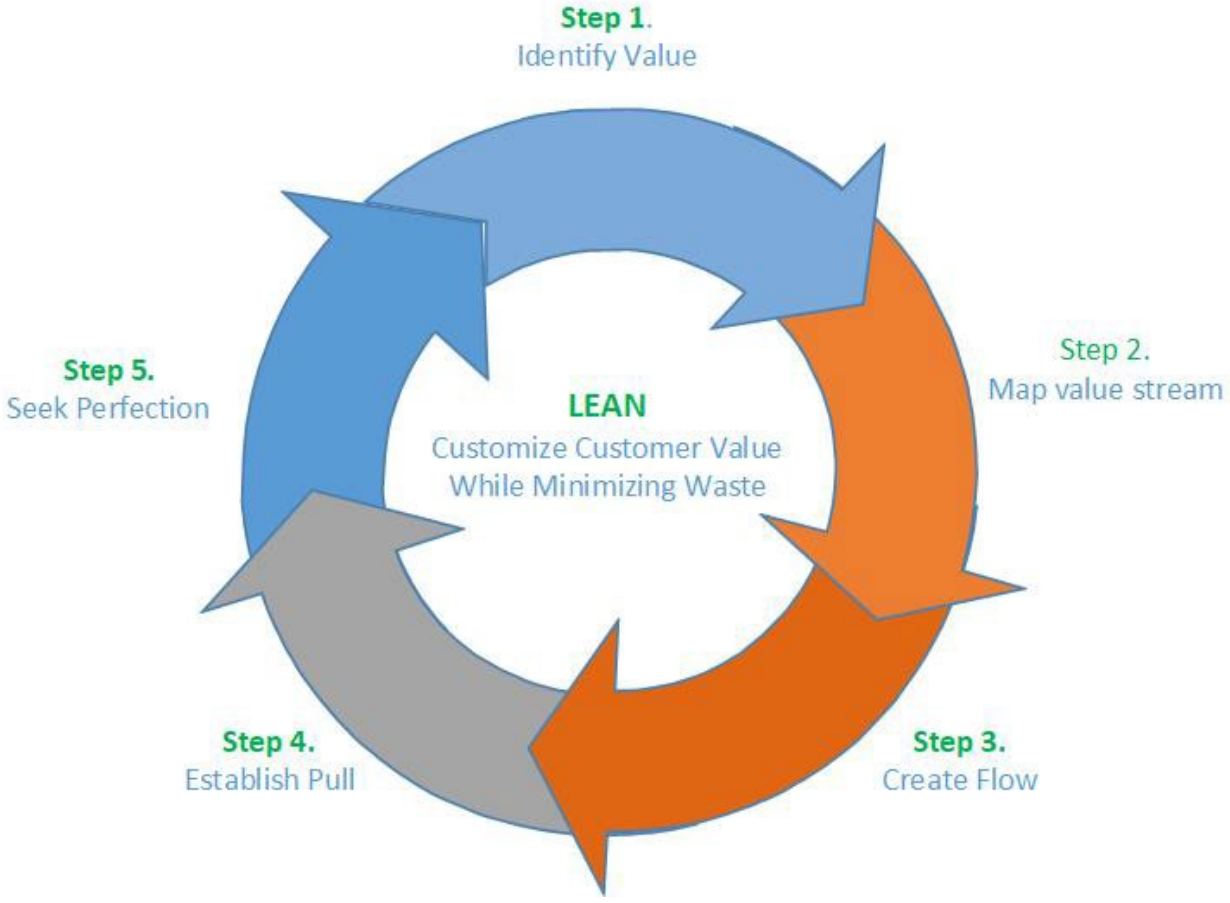

Figure 20. Lean thinking. Source: adapted from [101].

Main advantages of lean software development [101,109]:

- Increased collaboration and decision-making capability of the team.
- Streamlined approach that eliminates waste.
- Early delivery of working software.
- Economies of scale from waste elimination and resources utilization.

Main disadvantages of lean software development [101,109]:

- Requires organization-wide cultural change.

- Demands strong documentation and precise data for every step.
- Considerably difficult to scale.

2.2.9. Large-Scale Scrum (LeSS)

While Scrum focuses on a single iteration of a single team, Large-Scale Scrum (LeSS) was introduced in 2005 by Vodde and Clarman [110] to eliminate the gap in scaling Scrum in big product groups. LeSS recommends two different scaling frameworks, one for up to eight people and one for unlimited people, as shown in Figure 21. In both subsets, LeSS core principles are the following [110,111]:

- One Product Backlog for all teams.
- One Definition of Done for all teams.
- One Product Owner.
- One Sprint.
- One Potentially Releasable Product on every Sprint.
- Many cross-functional teams.

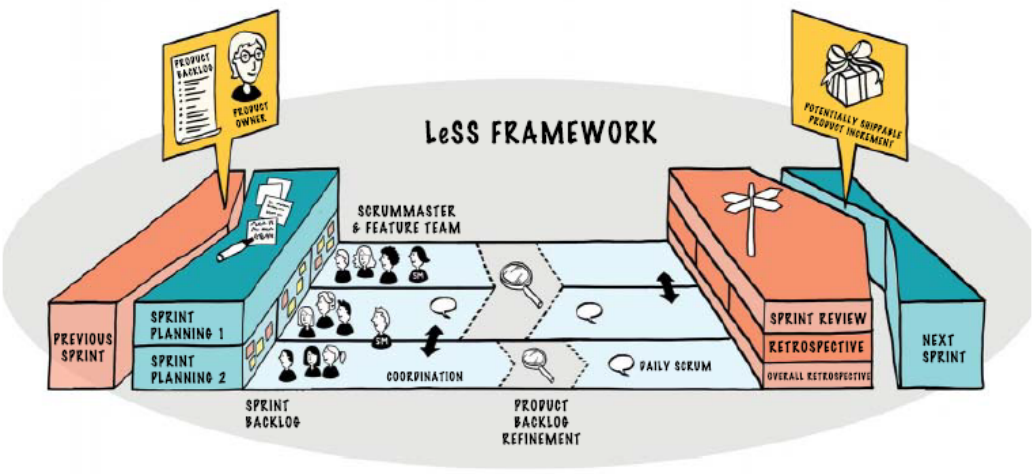

Figure 21. Large-Scale Scrum framework (LeSS). Source: adapted from [110].

Main advantages of Large-Scale Scrum [112]:

- Cost efficiencies by economies of scale.
- Operation simplification.
- Resource utilization and elimination of co-existence of multiple roles.
- High visibility across teams and the customer.

Main disadvantages of Large-Scale Scrum [113]:

- Requires organization maturity to apply.
- Demands a skillful and experienced Product Owner to cope with teams.
- Requires extensive organizational training for adaptation and maintenance.

2.2.10. Kanban

While Lean and Kanban as concepts were introduced during the 1950s, the adaptation of Kanban as a flow-control mechanism that supports the SDLC was first noted in 2004, by a small IT team within Microsoft [114]. The use of Kanban assisted the software team to visualize the flow from ideation to deployment, limiting work in progress (WIP) between stages, and highlighting bottlenecks, as shown in Figure 22. In this way, the team managed to prioritize and elaborate work with high visibility, producing constantly while developing only the items which are requested by the customer.

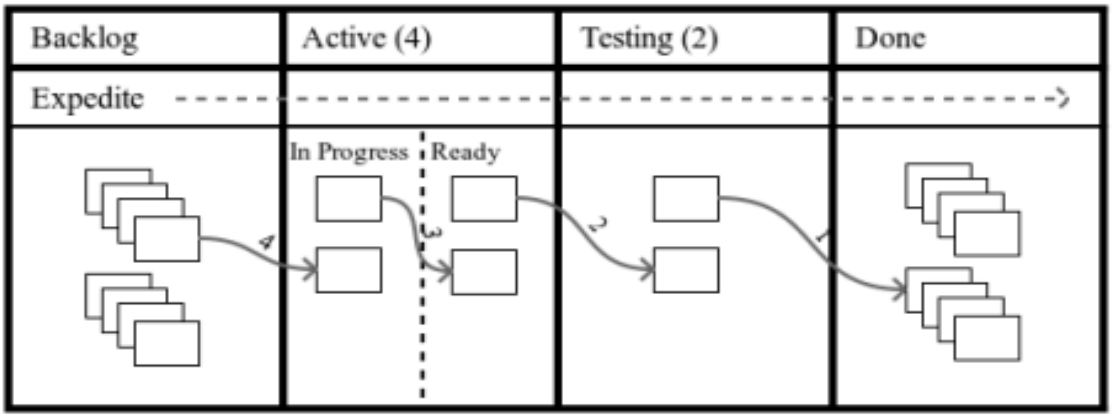

Figure 22. An example of Kanban for software development. In addition to Backlog and Done states, the team uses Active and Testing states, indicating currently 4 and 3 items respectively. Source: adapted from [115].

Main advantages of Kanban [116,117]:

- Increased progress visibility.
- Capability for focusing on priorities.
- Strongly collaborative environment.
- Reduced costs and waste.

Main disadvantages of Kanban [116,117]:

- Lack of timeframes associated with each phase.
- Cannot be used independently, it usually fits other frameworks (e.g., Scrum).
- Focus on task monitoring and transition.
- Inability to iterate results for required reworks.

2.2.11. Agile Unified Process

Agile Unified Process (AUP) was introduced by Scott Ambler in 2005 [118] as an evolution of Rational Unified Process (RUP) [49]. The difference in AUP is that it is an iterative and incremental process that consists of workflows and phases, as shown in Figure 23. In more detail, AUP differentiates from RUP by combining inception, requirement gathering, analysis, and design of the software solution into one workflow, called the Model workflow. Apart from the Model workflow, AUP consists of Implementation, Test, Deployment, Configuration, PM, and Environment workflows that progress over two weekly iterations that lead the transition of the process across its four serial phases, as shown in Figure 23.

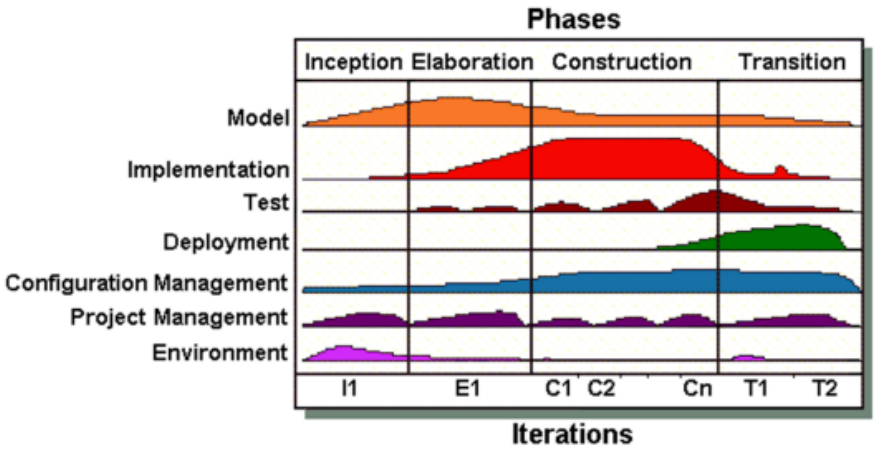

Figure 23. Agile Unified Process. Source: adapted from [119].

Main advantages of Agile Unified Process [120,121]:

- Applicable to both small and large software development projects.
- Constant flow of software delivery.
- Increased customer satisfaction, as they are involved in the development process.
- Flexibility for handling changes in requirements.

Main disadvantages of Agile Unified Process: [120,121]:

- Requires trained and skilled resources.
- Considerably heavy and streamlined in comparison to other agile methodologies.
- Lack of emphasis on document designs.
- Difficult to estimate the cost of the product at the beginning of the project.

2.2.12. DevOps

DevOps [122,123] is a modern agile framework emphasizing the ability to release software applications and services with high velocity by removing barriers between the development and operations teams. The framework found popularity after a conference speech in 2009 by John Allspaw and Paul Hammond [124]. DevOps' objective is to enable continuous business change through Continuous Delivery and Continuous Integration, as shown in Figure 24. The key to success in DevOps is for the development team to be aware of how deployment is carried out and, at the same time, for the deployment team to have basic knowledge of the architecture and coding rules applied.

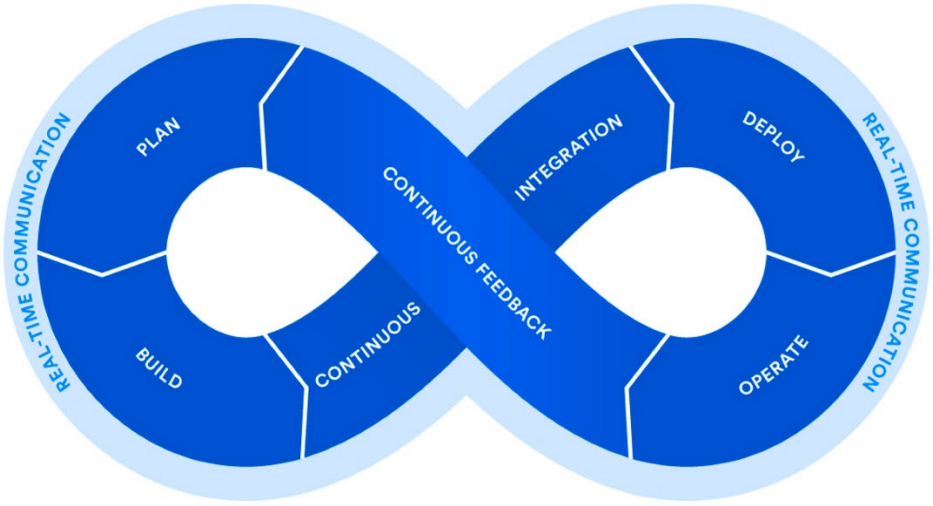

Figure 24. DevOps Loop. Source: adapted from [123].

Main advantages of DevOps [125,126]:

- Continuous release and deployment
- Increased collaboration, visibility, and trust within the team.
- Fast delivery and scalability.

Main disadvantages of DevOps: [123,127]:

- Requires DevOps expertise.
- Speed is usually prioritized over security.
- Costly to adapt and maintain.

2.2.13. Scrumban

In principle, agile teams work empirically. While Scrum is considered to be among the leading agile software development practices [82], certain limitations such as the lack of documentation policy, state tracking of each backlog work item, and sprint tracking, raised the need to eliminate these issues [128]. In this direction, pioneer methodologist Corey Ladas introduced his study [129] on the concept of Scrumban in 2009, based on

which Scrum limitations and dysfunctions could be anticipated when combining Scrum with appropriate Kanban practices, as shown in Figure 25.

Figure 25. Scrumban framework. Source: adapted from [130].

In this way, the Scrumban framework focuses on the following aspects [130]:

- Visualizing work with Kanban board.
- Work in Progress (WIP) limitations at each workflow stage.
- Pulling items, when needed, placed into freeze.
- Explicit team policies, enabling members to take quicker decisions.
- Shorter planning meetings for updating the backlog queue.
- Retention of constant Scrum events such as reviews and retrospectives.
- Metrics based on cycle time and lead time, in contrast to velocity.

Main advantages of Scrumban [116,131]:

- Enhanced flexibility, combining elements from existing lightweight software development methodologies.
- Elaboration of a team's level of Scrum effectiveness.
- Limitation of workflow delays.

Main disadvantages of Scrumban [131]:

- Effectiveness is dependent on team control over their workload.
- Demands experience in both Scrum and Kanban.

2.2.14. SAFe

The scaled agile framework (SAFe) was introduced in 2011 by Dean Leffingwell [132–134] as an open online knowledge base that provides best practices for lean-agile practitioners at the enterprise level. The SAFe framework recommends a set of workflows to be followed by enterprises for scaling lean and agile practices, as shown in Figure 26.

Main advantages of the scaled agile framework [134,135]:

- Alignment and visibility between teams on the enterprise level.
- Controlled environment for iterative development with a lean mindset.
- Waste elimination.

Main disadvantages of the scaled agile framework [135]:

- Requires high commitment from all involved enterprise levels and users.
- Extensive terminology and process overhead.
- Complexity regarding man-day cost estimations.

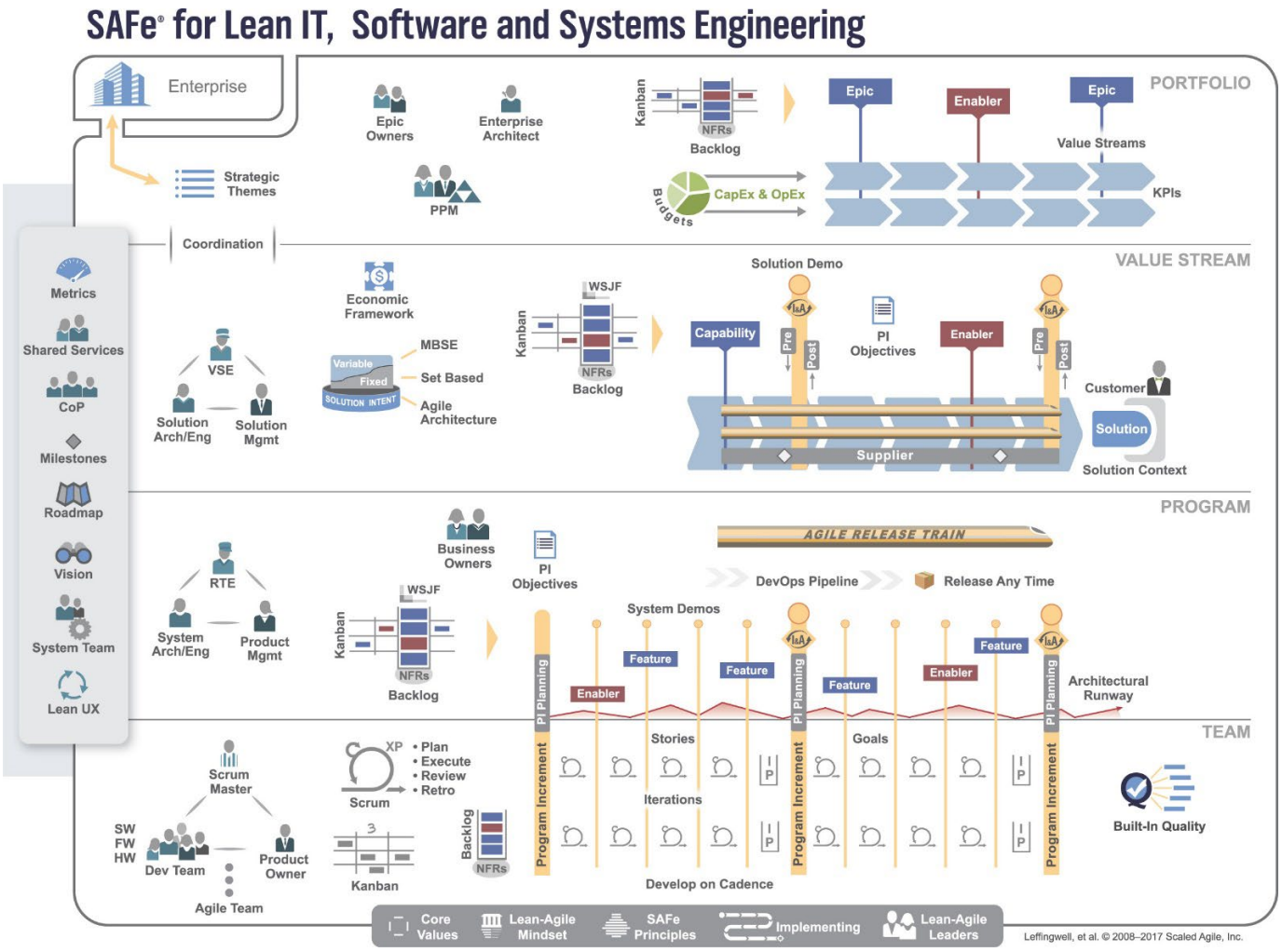

Figure 26. Scaled agile framework. Source: adapted from [132].

2.2.15. Scrum/XP Hybrid

Scrum/XP hybrid is a lightweight methodology introduced in 2002 by Mar and Schwaber [136] that today is followed by self-organized teams that adapt SAFe, as shown in Figure 27, combining practices from the Scrum and extreme programming methodologies.

The novelty of Scrum/XP relies on hybridization, using Scrum as a guide regarding agility, while technical practices are driven by XP principles, as shown in Figure 28. In this way, the model retains the standard Scrum Master role that promotes agility and educates the team in the use of Scrum, XP, and SAFe practices [137]. Similarly, the Product Owner continues to carry responsibility for clarifying to the development team what needs to be built.

Main advantages of Scrum/XP [139,140]:

- Small releases, enabling progress visibility for customers.
- Continuous testing, integration, and refactoring.
- Collective code ownership.

Main disadvantages of Scrum/XP [139,140]:

- Requires considerable experience in both Scrum and XP practices from teams.
- Scrum-wrapped XP projects need to anticipate the non-colocation of developers.

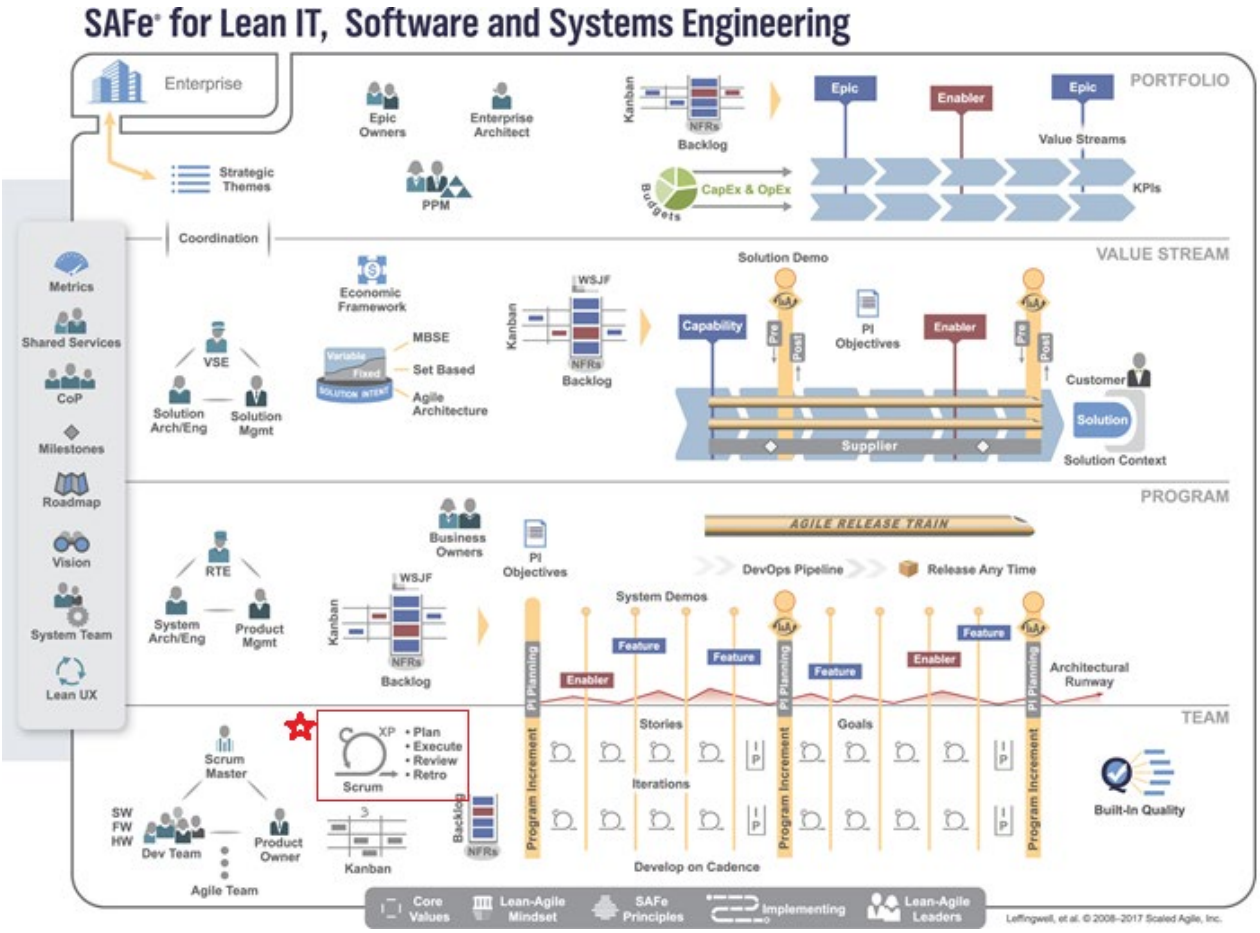

Figure 27. Scrum/XP hybrid within SAFe. Source: adapted from [133].

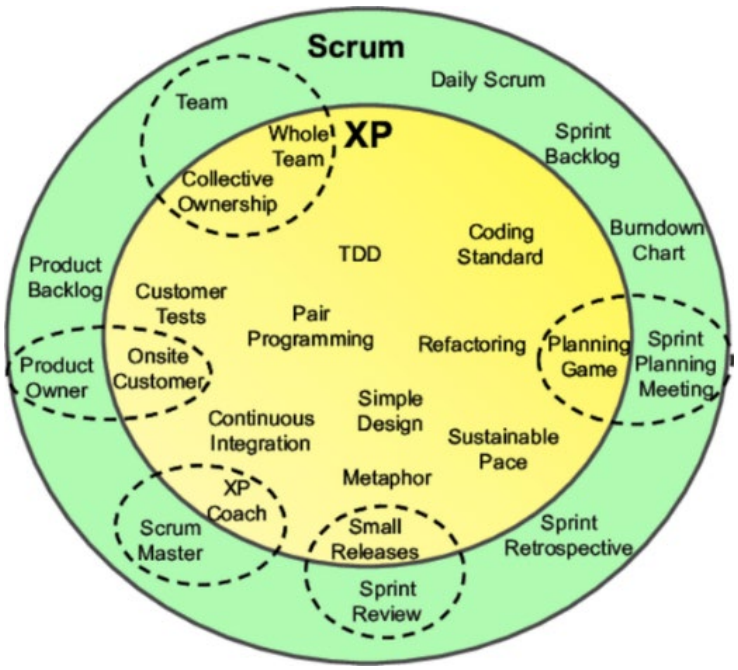

Figure 28. Scrum/XP hybrid. Source: adapted from [138].

2.3. Hybrid Software Development

Although the literature is explicit in its definitions of traditional heavyweight and agile lightweight software development methodologies, delivery practices indicate that these methodologies are rarely implemented in a fastidious manner. Inevitably, software development teams and practitioners dynamically combine elements from different methodologies, deriving hybrid methods that are suitable to their needs from project to project. A related IEEE Software publication from November 2003, just two years only after the release of the Agile Manifesto, played a key role in the systematic development of hybrid software development methods. In this IEEE publication, individual thinkers expressed their opinion on what software practices ought to be [141]. Their idea was that hybrid software development is best regarded as coherent sets of combined practices, methods, and tools that complete each other by minimizing performance trade-offs when used individually [28]. Hybrid software development methods grew significantly from that point onwards.

According to Kuchrmann et al. [12], hybrid software development is "*any combination of agile and traditional approaches that an organizational unit adopts and customizes to its own context needs*". Based on this, it is interesting to explore trends with regard to key components required to construct such methods. As noted by Küpper et al. [142], systematic development of hybrid software development processes includes three commonly appearing layers, as shown in Figure 29:

- *Practices*: The range of tasks of the development team during SDLC, and their rules of progression for process running. Practices are categorized into single practices, (i.e., code refactoring, daily stand-ups), and methods (i.e., Lean, Crystal, KanBan, etc.).
- *Frameworks*: These are one or many selected practices and methods, specifying the SDLC management routines to be followed by the software development team.
- *Context*: This is derived from project goals and is related to accumulated success factors. These factors guide the selection of practices and methods, defining the desired outcome when applying a framework.

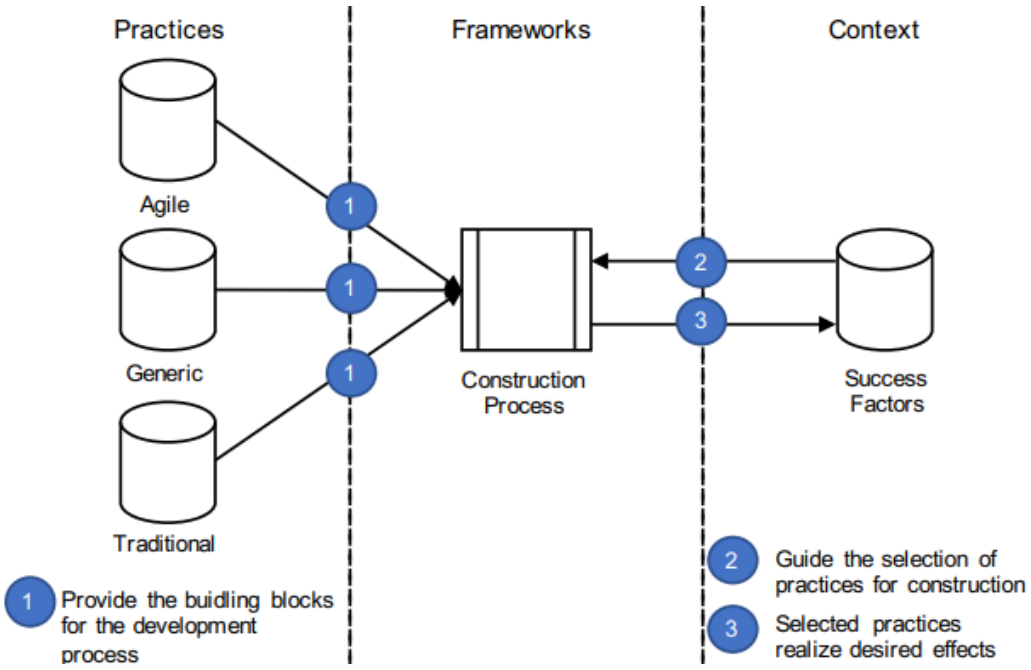

Figure 29. Components needed for the construction of hybrid software development methods. Source: adapted from [142].

Following the identification of the key components needed to construct a hybrid software development method, the formation of hybrid development methods was found to follow certain patterns. According to the systematic literature review on how hybrid

development methods are organized by Prenner et al. [143], as published at the proceedings of the International Conference on Software and System Processes (ICSSP '20), Royce's traditional Waterfall model [15] is applied in some way in all hybrid approaches, differing in the arrangement of the phases. These four patterns found are the Waterfall-Agile approach, the Waterfall-Iterative approach, the Pipeline approach, and the Combinations.

2.3.1. Waterfall-Agile Approach

The Waterfall-Agile approach has the sequential waterfall phased model as the structural basis that emphasizes control, timeframes, and finite project cycles [144]. As indicated in Figure 30, there are six discrete phases through which the software development project subsequently progresses. Starting from the requirements gathering and analysis phase, work is elaborated to the architectural design of the solution, driven by commonly appearing artifacts such as UML diagrams and specifically Activity, Class, Sequence, and Robustness diagrams [144,145]. Notably, teams undergo risk mitigation on the design artifacts that corresponds to customer needs via an optional build of prototypes. The sum of artifacts created during design is a matter of review at the end of design, elaborating on the created user interface.

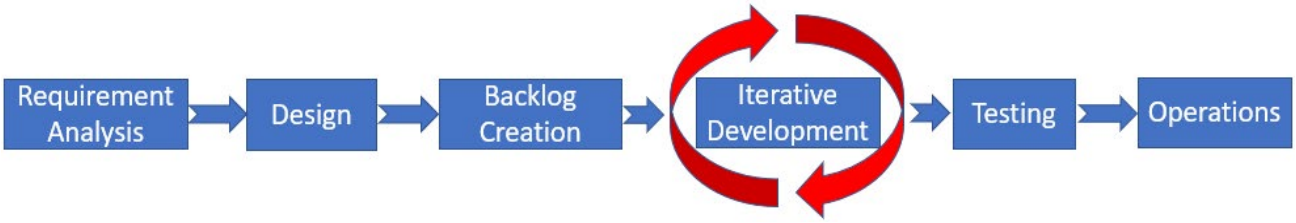

Figure 30. Waterfall-Agile Approach. Source: adapted from [143].

To facilitate agile development, the design phase is followed by the creation of a product backlog of prioritized user stories. These requirements drive iterative development by the team, most commonly using the Scrum framework's agile principles, elaborating progressively [142,144]. While unit testing is facilitated by the development team during the sprints, iterative development completion is followed by systems integration testing (SIT) [146] and user acceptance testing (UAT) [147]. Finally, the produced and verified code is deployed into production or released to the client, providing the necessary documentation, training, and maintenance support.

2.3.2. Waterfall-Iterative Approach

The Waterfall-Iterative approach, also noted as "*Waterative*" [148], is a Waterfall model with its phases being executed iteratively as the project elaborates, as shown in Figure 31. This integration of Waterfall and Iterative approaches includes a requirement analysis phase for each iteration, defining the iteration's goal.

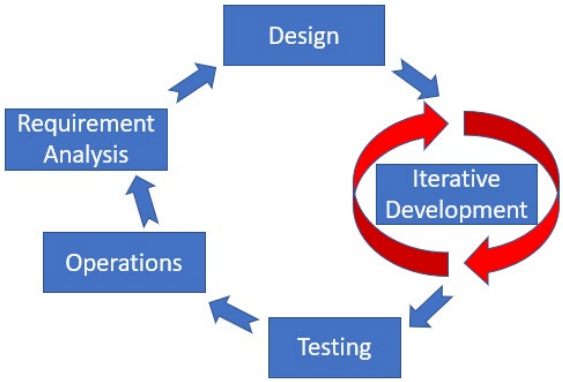

Figure 31. Waterfall-Iterative Approach. Source: adapted from [143].

Notably, elaboration of the design takes place based on the requirements selected for each iteration, adding functionality to the user interface on each cycle. Subsequently, the development phase during each iteration follows Scrum principles, similar to the Waterfall-Agile approach. Finally, systems integration testing (SIT) and user acceptance testing (UAT) take place before the deployment of each iteration's deliverable to production.

2.3.3. Pipeline Approach

Unlike the Waterfall-Agile and Waterfall-Iterative approaches that follow a subsequent elaboration of each phase, the Pipeline approach is based on the parallel execution of phases, leading to incremental feature-driven development, as shown in Figure 32. The initial phase of requirements analysis includes user stories creation, which serves to represent the to-be functionality for priority in design and delivery order. The development process run takes place in sprints during each iteration, before testing the potentially releasable increment in production.

Requirements Analysis	N	N+1	N+2	N+3	N+4
Design	N-1	N	N+1	N+2	N+3
Development	N-2	N-1	N	N+1	N+2
Operations	N-3	N-2	N-1	N	N+1
Testing	i-2	i-1	i	i+1	i+2

Figure 32. Pipeline approach. Source: adapted from [143].

2.3.4. Combinations of Approaches, Frameworks, and Practices

Interestingly, research on how hybrid development approaches are structured indicates that mixture occurs on the approaches, frameworks, and practices levels.

- *Combination of approaches*. According to Prenner [143], hybrid development occurs primarily as a matter of combination between the three profound approaches: WAA, WAI, and Pipeline. Notably, the Waterfall model finds usage in all three of these approaches.
- *Combination of frameworks and practices*. Combinations between different frameworks and practices to establish hybrid software development were studied thoroughly by the international research project named HELENA (*Hybrid DEveLopmENt Approaches in software development systems)* [149,150], publishing their results as shown in Figure 33.

By post-processing HELENA project results, the frequency of the most common methods is highlighted, which is indicative of their importance to software development teams. With reference to Figure 33, each combination is identified with a number that represents the count of combinations occurring from participating audience's preference. These software development method combinations result in the 20 most commonly appeared combinations that are included in Figure 34.

Based on this, teams primarily choose the following five software development method combinations:

i. Scrum with Iterative Development
ii. **Scrum** with **Prototyping**
iii. **Waterfall** with **Iterative Development**
iv. **Scrum** with **Kanban (or Scrumban)**
v. **Scrum** with **DevOps**

			Method																		
			Traditional					Agile											Both		
			Rational Unified Process (custom)	Rational Unified Process (standard)	Spiral Model	V-Model Derivative	Watefall/Phase model	Agile Portfolio Management (APM)	Behavior-Driven Development (BDD)	DevOps	Disciplined Agile Delivery (DAD)	Extreme Programming	Feature-Driven Development (FDD)	Kanban	Large-Scale Scrum (LeSS)	Lean Development	Scaled Agile Framework (SAFe)	Scrum	Crystal Family	Iterative Development	Prototyping
Method	Traditional	Rational Unified Process (custom)				1			1			1		1				1			1
		Rational Unified Process (standard)																			
		Spiral Model																		1	
		V-Model Derivative					7	2	1	3		2		3		2		6		8	7
		Watefall/Phase model						6	2	5		3	2	4		2	2	10		14	10
	Agile	Agile Portfolio Management (APM)							2	4		2	2	4	1	3	2	9		7	5
		Behavior-Driven Development (BDD)								5		1		4				7		5	4
		DevOps										1	1	5		1		10		10	5
		Disciplined Agile Delivery (DAD)																			
		Extreme Programming											1	3	1	2	1	4		5	4
		Feature-Driven Development (FDD)																1		3	2
		Kanban													1	5	2	13		10	7
		Large-Scale Scrum (LeSS)														1	1	1		1	1
		Lean Development															2	9		7	3
		Scaled Agile Framework (SAFe)																3		2	2
		Scrum																		21	15
	Both	Crystal Family																			
		Iterative Development																			12
		Prototyping																			

Figure 33. Focused view of the upper left quadrant of HELENA project results. Source: adapted from [6].

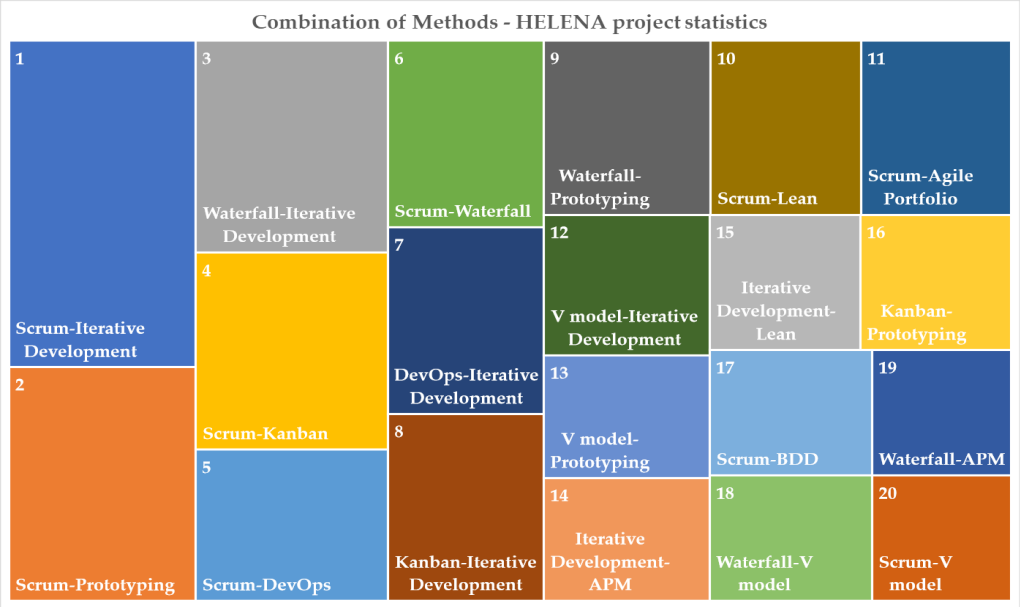

Figure 34. Own post-processing HELENA results [6] regarding software development methods combinations.

3. Information Systems Audit Challenges in the Modern ERA

The evolution of software development methodologies over the last 50 years highlights the global trend of lightweight agile methodologies gaining ascendancy over traditional. While organizations seek speed and adaptability to continuous change, agile software development frameworks evolve further into hybrid methods that consist of elements from different approaches, beyond blueprints, analytical plans, thorough documentation, and risk monitoring mechanisms. These radical changes in the way organizations set up their business models to accommodate software delivery by following lightweight methodologies are a continuous challenge for IS/IT audit teams also, since the value and impact of such methodologies require alignment and adjustment to expectations and regulations.

Information technology auditing according to S. Gantz [151] "*examines processes, IT assets, and controls at multiple levels within an organization to determine the extent to which the organization adheres to applicable standards or requirements*". This translates to verification of proper utilization of information technology and systems by organizations for strategic alignment with their mission and goals. As such, audits determine their status according to the evidence collected and if the technology has been correctly tested. In principle, audits can be classified into three categories [13]:

- Internal: run by auditors within their organization as self-assessments. These audits restrict the sharing of findings outside the organization and cannot be used for licensing.
- External: in the case of vendor–supplier relationships, a customer orders and operates an audit to verify the expected level of performance of their relationship.
- Independent: run from third-party independent auditors for licensing, certification, or product approval.

In all the above types of audits, despite the objective of the party requesting an audit, IS/IT auditors attempt to determine the truth regarding products, processes, and systems.

The full spectrum of audits that can be held by auditors according to the enterprise architecture standard ISO 42010/IEEE 1471 is shown in Figure 35. There are four different levels into which roles and functions within an organization fall:

- Executive business model: examines how the organization is governed.
- Business processes: middle-management administration processes for business operations.
- System of systems: the connection of hardware and software for day-to-day business operations.
- Technical interfaces: the lowest level; most vulnerable for breaches, failures, and faults.

3.1. Factors That Affect Information System Audits

According to 2022′s Information Systems Audit and Control Association (ISACA) publication by CISA certified auditor, A. Sayana [152], information systems audits have changed a great deal over time. Such change is highlighted as a necessity to follow information-systems-related changes occurring in the business environment, the technology landscape, sociopolitical trends, and governance.

3.1.1. Business Environment

Related to the evolution of the business environment, researchers M. Lindgren and H. Bandhold raise the following question: *"how can we successfully compete in a constantly changing business environment"*? In recent years, organizations have looked to maintain and improve their competitiveness by shifting their focus to developing technological advantages, rather than acquiring commodities and materialistic assets. A view of the organizations' ranking based on market capitalization today [153] reveals that most of today's leaders depend on technological innovation and creativity (e.g., Apple, Tesla, Google, and so on). Organizations leverage IT gains by also evolving their business models to accommodate technological innovation. Moreover, digitization of the workplace, which boomed during COVID-19, has unleashed new operating models that enable efficient remote working and collaboration.

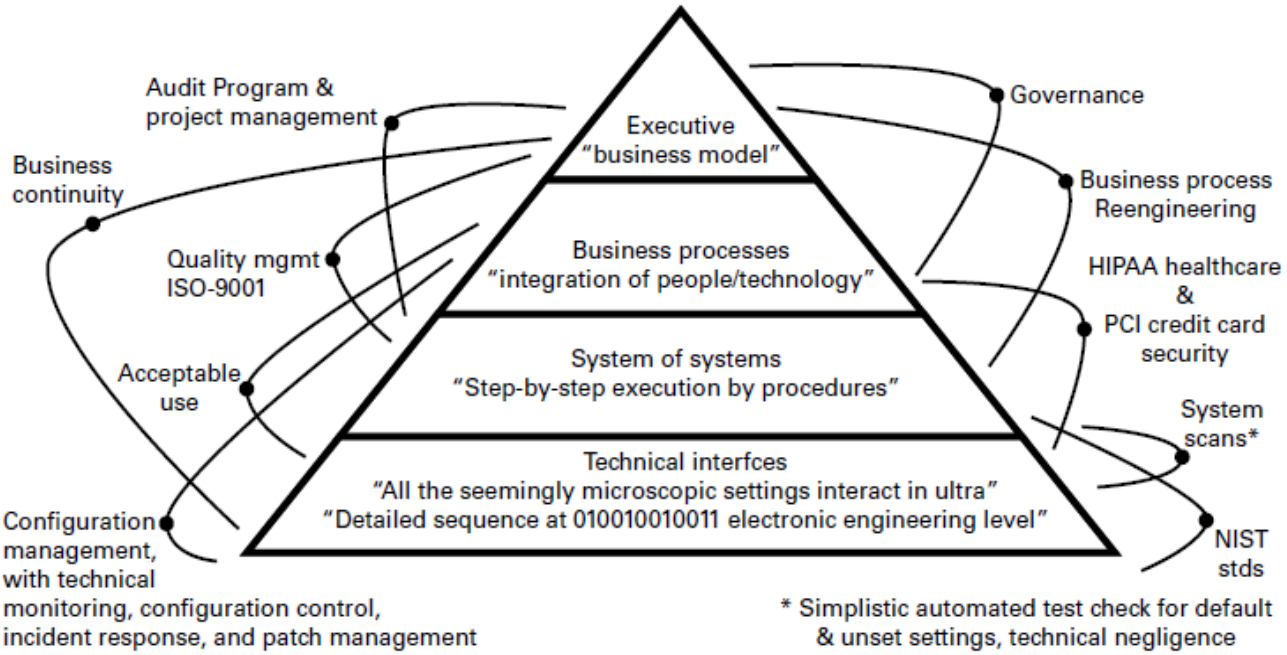

Figure 35. Audits range based on ISO 42010/IEEE 1471. Source: adapted from [13].

3.1.2. Technology Landscape

A key factor in the evolution of information audits is the rapid pace of technological advancements. Inevitably, there have been tremendous changes as a result of the increase in the use of personal computers since the mid-1980s, the evolution of mobile communications that created a click-and-swipe world, the influence of cloud technology, and the wide use of virtual machines replacing and upscaling physical computer multi-processing. According to an April 2022 Statista report [154], the global population's internet connectivity had reached a phenomenal 63%, equal to five billion internet users worldwide, compared to only 5.8% at the end of 2000. At the same time, artificial intelligence, machine learning, big data analytics, Internet of Things, blockchain and robot process automation, and cybersecurity—among others—are widely accepted as integral parts of today's technological landscape. Certainly, the technological domain extends over a broad range that requires of modern IS/IT auditors an enhanced 360° view of the current spectrum.

3.1.3. Sociopolitical Global Trends

In 2005, three-time Pulitzer award-winner Thomas Friedman published the book "The world is flat" [155], pointing out that we are becoming part of a global supply chain of technology and manufacturing. At the same time, globalization must comply locally since organizations must fulfill requirements by governments and other local or regional regulatory bodies. Moreover, while global economic inequities persist (according to a 2020 report from the United Nations), cybercrime has dramatically increased in the last twenty years. Specifically, SurfShark reports [156] that the financial losses due to cybercrime grew almost 400 times from 2001, which translated to an increase from $2000 to $788,000 in losses per hour over this period. Another sociopolitical aspect with a significant impact on audits is the obligation of organizations to protect the environment and fight against climate change. As the frame of compliance to environment and climate becomes wide and complex, modern auditing must ensure that attention is paid by organizations as required.

3.1.4. The Need for Governance

The need for governance is constantly rising, given the radical changes in the business environment, the technology landscape, and sociopolitical trends. According to Van Grembergern [157], IT governance is "*the organizational capacity exercised by the board, executive*

management and IT management to control the formulation and implementation of IT strategy and in this way ensure the fusion of business and IT". Notably, organizations leverage IT gains to the fullest when the business strategy is aligned with the IT, the IT performance is systematically monitored, and risks are managed and mitigated, as shown in Figure 36. The lack of effective IT governance puts business value at risk, which results in a loss of confidence and trust by the shareholders and, eventually, the customers. Declining to accomplish IT governance's mission and scope results in significant issues, such as lack of knowledge for decision-making, controllability of IT, comprehensibility of regulations, corruption, negligence, and fraud. As such, an effective IT government mandates comprehensive audit systems and processes that need to constantly be reviewed based on local and global standards.

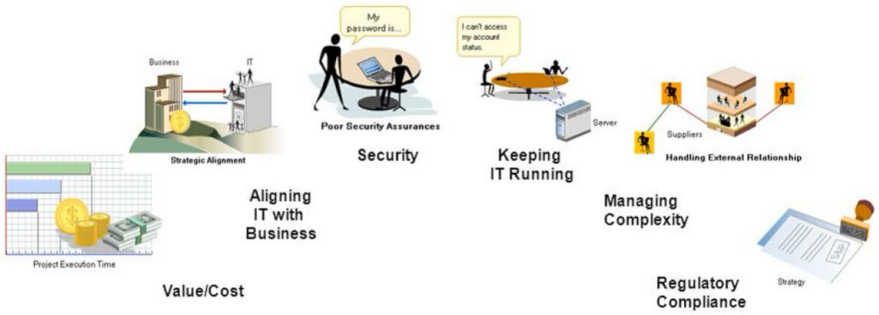

Figure 36. Organizational challenges relating to IT. Source: adapted from [158].

An organization's capacity to exert IT governance is evaluated by conducting IS/IT audits, which ensure that IT systems sustain business goals and strategy. As such, auditing requires a thorough understanding of organization IT governance frameworks and standards.

3.2. IT Governance Frameworks

Notably, the definitive IT Governance Framework, according to ISACA, is the Control Objectives for Information and Related Technology (COBIT) [159,160], as shown in Figure 37. This framework was first released in 1996 (latest version: 2019) by ISACA volunteers, containing a complete set of strategies, processes, and procedures for control. COBIT is a framework for monitoring the maturity and optimizing the utilization of the IT resources within an organization, combining IT and business goals.

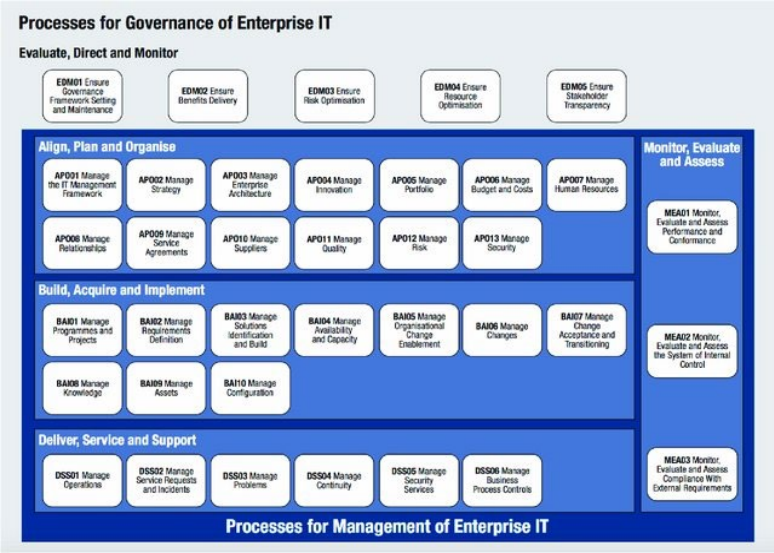

Figure 37. The COBIT 2019 framework. Source: adapted from [159].

The wider range of available frameworks available to information system auditing practitioners today is shown in Figure 38.

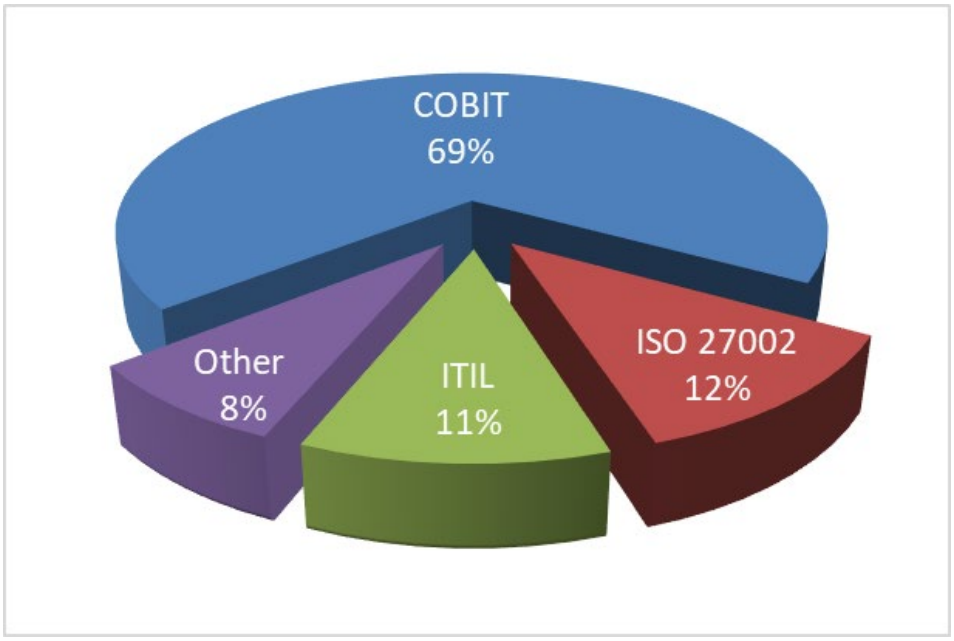

Figure 38. Standards and frameworks that are used for planning IT audit activity. Source: adapted from [161].

Organizations have a wide range of additional frameworks to choose from for assessing the impact of IT governance on accomplishing business objectives. Among them, the most well-known are:

- *IT Assurance Framework (ITAF)* [162]: an ISACA framework for designing, conducting, and reporting IT audits.
- *IT Infrastructure library (ITIL)* [163]: a set of practices for the alignment of IT with business goals, providing a baseline for planning, implementing, and measuring deliverables.
- *The Open Group Architecture Framework (TOGAF)* [164]: a framework for governing enterprise information technology architecture.
- *Committee of Sponsoring Organizations (COSO)* [165]: a voluntary organization that provides guidance to organizations with its frameworks for operational performance, internal control, risk management, and fraud deterrence.
- *VAL IT* [166]: a COBIT-based framework that enables the creation of business value from IT-enabled investments.

3.3. IT Governance Standards

In addition to frameworks that are available to organizations for conducting IS/IT audits, there are standards that must be followed in the form of sets of operational or technical measures, procedures, and practices. The following bodies raise standards during IS/IT audits:

- *International Standards Organization (ISO) IT Standards* [167]. The International Standards Organization raises specifications for products, services, and good practices, assisting organizations to become more effective and efficient. The key standards that apply to IS/IT are:
  - i. *ISO 9001 (Quality Management Systems).* Stipulates the requirements of a quality management system.
  - ii. *ISO 15489 (Records management).* Specifies the rules to create, capture, and manage records.

iii. *ISO 19011 (Guidelines for auditing management systems).* Provides guidance for the internal and external auditing of managed systems.
iv. *ISO 20000 (IT operations).* The first standard for IT services management; includes the design, transition, delivery, and improvement of service requirements, securing value creation for both the customer and the service provider.
v. *ISO 27000 (IT security).* Establishes guidelines and general principles for initiating, implementing, maintaining, and improving information security management in an organization.
vi. *ISO/IEC/IEEE 42010:2011 (Systems and software engineering—Architecture description).* Specifies the required architecture content, architecture frameworks, and architecture languages description for the creation, analysis, and sustainment of IT system architecture descriptions.
vii. *ISO 27002 (Information security controls).* Provides the appropriate range of generic information security controls and implementation guidance.
viii. *ISO 31000 (Risk).* Assists organizations in effectively managing the risks in an environment full of uncertainty.
ix. *ISO 38500 (Governance).* Provides a framework for effective IT governance for top management to understand and satisfy legal, regulatory, and ethical obligations with respect to organization use of IT.

- *Information System Audit & Control Association (ISACA) Standards* [168]. ISACA provides the minimum acceptable performance required to meet the professional responsibilities set out in the ISACA Code of Professional Ethics. These standards are collected, maintained, and published as a factsheet by ISACA known as the IT Assurance Framework (ITAF).
- *International Standards for Supreme Audit Institutions (ISSAI) Standards* [169]. Some specific ISSAIs relating to the audit of information systems are:
  i. *ISSAI 5300* Guidelines on IT audit.
  ii. *ISSAI 5310* Guidelines for Information Systems Security audit.
  iii. *ISSAI 5450* Guidelines for Public Debt Information Systems audit.

3.4. Governance in Software Development

Every organization practicing software development has in place a minimum set of structures, processes, and policies by which software development and deployment is directed and controlled to mitigate risks to business value. This set is recognized as the software development governance elements adopted by the organization. The purpose of software development governance is defined by the following pillars [170]:

- *Value management*. Securing alignment and impact of the software developed by the organization.
- *Flexibility.* Leveraging resource utilization towards the selection of the most appropriate software development methodology for the case at hand.
- *Risk management.* Enabling continuous risk management during the SDLC, adhering to internal and external needs for compliance.
- *Change management.* Establishing a change management mechanism during the SDLC that enables the embrace of changes.

3.5. IS/IT Control Audit Objectives

In principle, IS/IT audit trails IT controls that are at the heart of the IT environment of any organization. Controls are considered the sum of procedures, policies, and methodologies in place to assure protection of the organization's assets, preciseness, consistency of records, and overall operational coherence to management standards. Distinguishing between general controls and application controls is essential when conducting audits. General controls establish the environment wherein software systems are not only developed, but also operated, managed, and maintained, in terms of specific procedures. On the other

hand, application controls are conditional to each software application, including data input validation and completeness checks, access authentication and authorization, data encryption rules, error reporting, and so on. IT controls can be visualized as a concentric, onion-like layered framework [171], as shown in Figure 39.

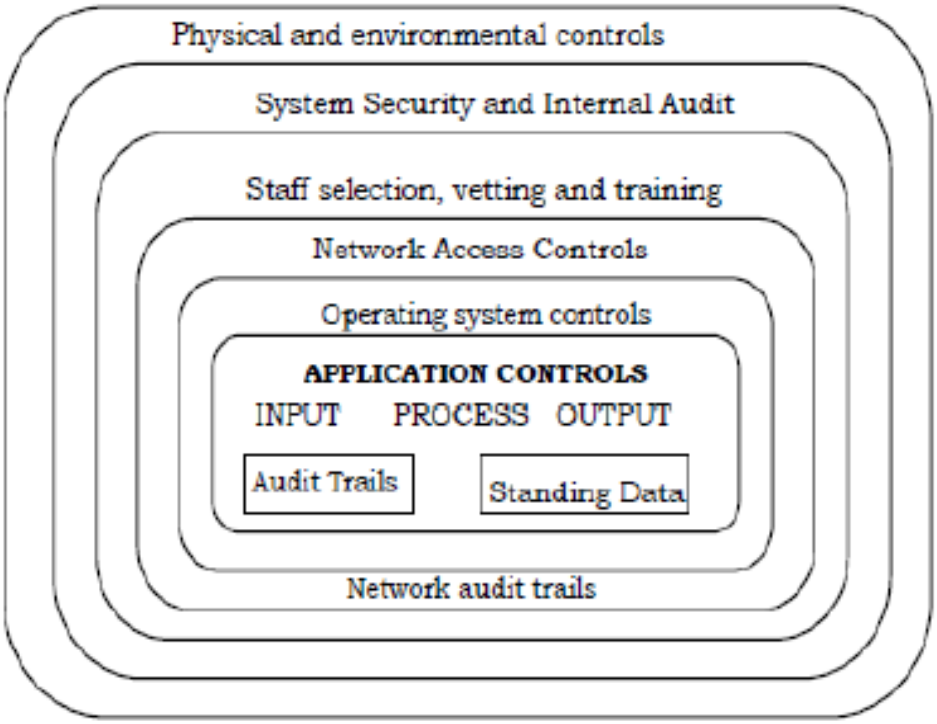

Figure 39. IT Controls framework. Source: adapted from [171].

3.6. Information Systems Audit Process

Organizations interested in obtaining a periodic certification, usually annually, for their performance against selected policies, compliance with regulations, and adherence to standards, are required to take a series of individual audits. These sets of smaller and ongoing audits ensure that organizations act to remain compliant and aligned with business goals included in audit programs. Audit programs are executed periodically and repeatedly, based on a standard set of procedures carried out. These procedures include:

- Audit planning.
- Scheduling of audits.
- Competence assurance of auditors.
- Audit team selection.
- Audit roles and responsibilities assignment.
- Conducting audits.
- Records maintenance.
- Performance tracking.
- Issues tracking.
- Reporting to management.

While size, infrastructure complexity, and business model complexity play a significant role and influence the audit program, an IS/IT audit encompasses specific steps. Auditors show consistency in structuring and following an audit process that follows certain steps when conducting audits. Audit processes consist of serial steps that define the so-called audit quality control plan. Predominantly, the best practice for establishing the audit process, according to the study guide for Certified Information System Auditors (CISA) by David Cannon et al. [13], combines a mixture of ISO audit standards with additional details from ISACA. The proposed auditing process by CISA's study guide is based on a series of

generally accepted auditing procedures, as shown in Figure 40. IS/IT audit is considered a systematic review of historical data and is conducted by sequentially following these procedures in steps.

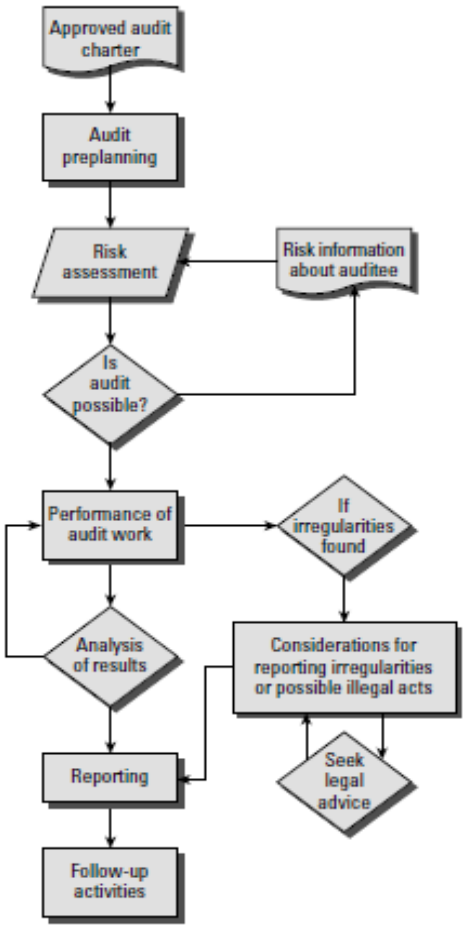

Figure 40. The CISA's study guides the audit process. Source: adapted from [13].

3.7. Auditing SDLCs

In software development, value is secured by fitting the system development life cycle processes into a project management framework that implies the disciplines for planning, organizing, controlling, and reporting [13]. Any software development methodology, to secure value addition, must have a viewpoint that serves this embodiment of SDLC processes in IT projects. Notably, there are two contrasting viewpoints on software development; evolutionary and revolutionary [13,172]. The primary scope of auditing software development is to assess whether the prescribed project is oversight and that the selected SDLC and change-management processes are followed.

3.7.1. Evolutionary Development

Evolutionary software development [13,60,172] is managed through a combination of the selected SDLC and traditional project management. In evolutionary development, software development effort relies on writing code by programmers, which follows the execution of certain levels of analysis and solution design. In this way, in projects, software development is driven following life-cycle management, and software is delivered in incremental stages, known as releases.

Traditional SDLCs Audit

In traditional software development, applications undergo a common process of functional and non-functional needs analysis, design, coding, implementation, and production use, followed by ongoing maintenance. Eventually, all programs are replaced by a newer version. The essence of conducting a software development audit is to determine whether organizational objectives have been properly identified and met throughout the lifecycle. According to ISACA [13,168], commonly used traditional models (e.g., Waterfall, Spiral, and Prototyping) receive a type of audit based on the ISACA's seven-phase SDLC model, as shown in Figure 41. If there is evidence of a missing phase, auditors raise concerns of inappropriate shortcuts, which increase the risk of major control failures.

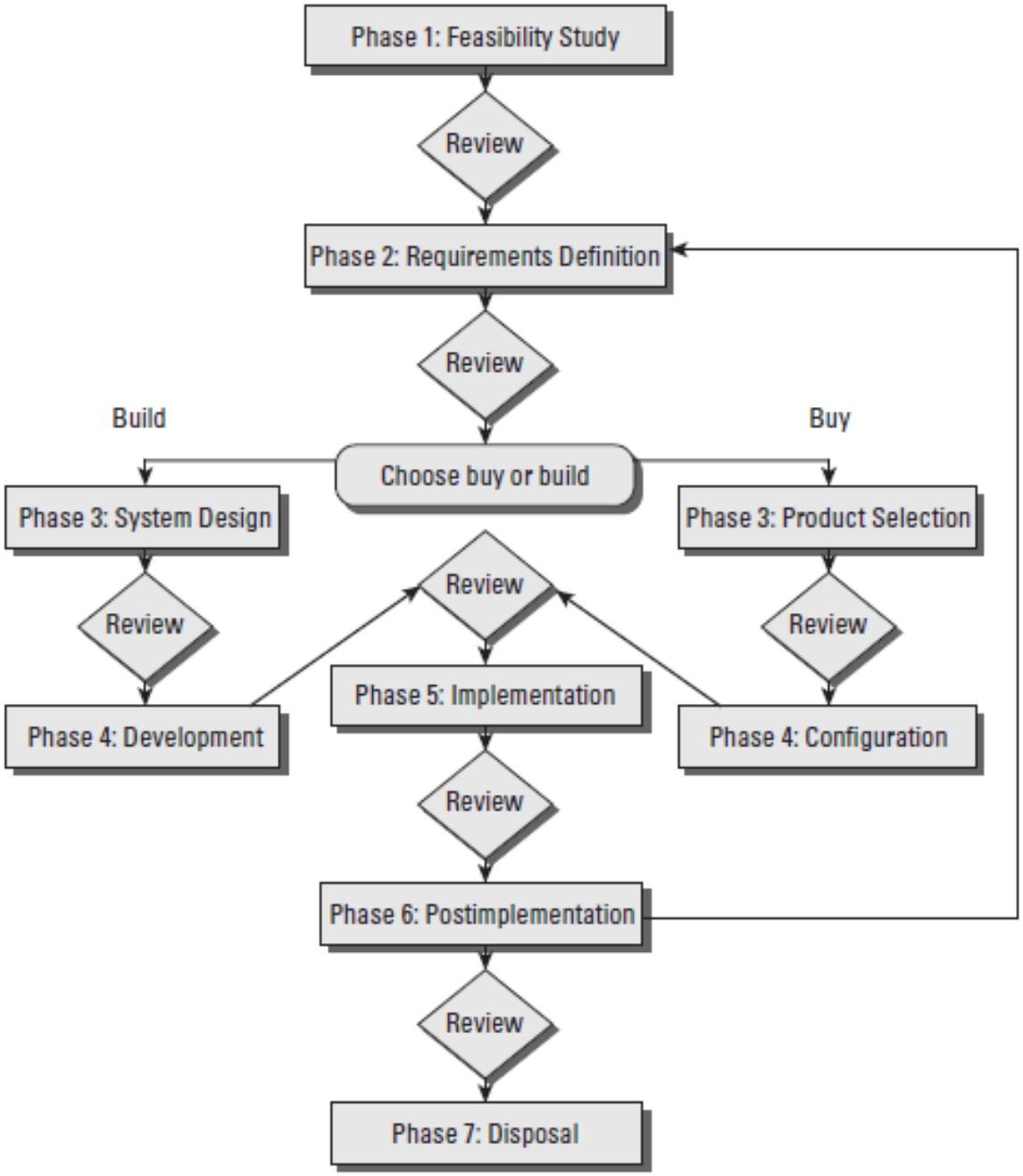

Figure 41. The ISACA seven-phased SDLC audit model. Source: adapted from [13].

Conducting a software development audit requires a formal review between each phase. As such, auditing of each phase undergoes investigation for an evidence-based review of specific focus points [13]:

- *Feasibility Study phase auditor's interest*. Includes review of the business case, needs analysis, cost justification, risk mitigation plan, and occurrence of formal management approval to proceed to the next phase.
- *Requirements Definition phase auditor's interest.* Involves obtaining complete functional/non-functional requirements, flowcharts, and conceptual entity relationship diagrams (ERDs). Requirements must be connected to success factors and acceptance test criteria comprising security concerns, while management acknowledges and formally approves plans and estimated costs.
- *System Design phase auditor's interest.* Reviewing the design baseline and design documents including flowcharts and model diagrams. Assessing integrity control of the flowing data, traceability of user transactions, quality control alignment, and evidence of management approval to proceed to the next phase.
- *Development phase auditor's interest.* Primarily, verification that quality control processes are followed during software development. This includes debugging and fixing issues throughout recorded testing, while compliance to original requirements is derived through formal user acceptance, followed by management review and agreement to proceed to the next phase.
- *Implementation phase auditor's interest*. Confirmation of software installation and operation when entering the production environment and support of end-users with documentation support. Management needs to have agreed before the new software is deployed to the production environment.
- *Post-implementation phase auditor's interest*. Verification of software development project closure and alignment of the output with the organization's objectives by contacting reviews to confirm that the new software system was developed as designed and that controls were applied throughout the SDLC.
- *Disposal phase auditor's interest.* Validation that previous version disposals followed respective processes and documents and that accounting records are in place.

Agile Development Audit

In alignment with the aspirations of modern organizations to adopt lightweight software development methodologies, IS/IT auditing needs to be cognizant of business needs for agility, innovation, and high-paced adaptation of new technologies. The integration of business management with technology management based on agile models is essential and rather challenging to assess during IS/IT audits. Not surprisingly, there is a common perception among management as a result: "*We are running on Agile, so there is nothing to audit*" [173]. Certainly, agile organizations are IS/IT auditable organizations; audits are based on established practices but with significant freedom to the auditor in constructing the auditing program. The main differences between agile and non-agile IS/IT auditing are shown in Table 1.

Table 1. Agile audits vs. traditional audits. Source: adapted from [174–176].

Attribute	Agile Audit	Traditional Audit
Focus	Defined value expectations	Audit objectives
Engagement	Sequential	Linear stages
Planning	Iterative and incremental	Master plan
Ownership	Team-based	Internal audit team
Findings	Collaborative discovery	Audit objectives
Documentation	Rationalized	Detailed

Table 1. *Cont.*

Attribute	Agile Audit	Traditional Audit
Resources management	Time-boxed	Dedicated
Status updates	Iterative and incremental	Master plan

Despite the differences between agile development methodologies, the limited literature, and lack of a standard agile audit framework, ISACA's 2017 publication by Alexiou [176] provides guidelines to audit practitioners on how to approach agile auditing [177,178]. These guidelines are summarized below.

- *Start early with data collection*. It is essential to have a working data sample to start the audit process, which can be later updated during the audit lifecycle. Early data checks are essential for identifying preliminary issues with data compilation, reading issues, and processing outcomes.
- *Preserve team's motivation.* It is recommended to avoid the introduction of work optimization frameworks to high-performing agile teams with proven capacity to deliver.
- *Adapt on the fly during sprints.* When running agile in sprints, changes are accommodated according to a team's decision, with limited justifying documentation and management approval needed. Teams need to maintain development progress with no discounts on quality and risk management.
- *Adhere to agile mindset communication*. Auditing agile organizations depends on interaction with the development team in the absence of hierarchical approvals and strict communication protocols, which may seem paleolithic to the team. Complementary agility perception and motivation by both the auditor and auditees are key to success.
- *Possess an understanding of the big picture*. This will lead to safer conclusions on what needs to be audited and how can be approached, monitored, and reported.

Continuous Auditing

The concept of information system continuous auditing is a fairly new idea and is under ongoing investigation by researchers and practitioners in the literature. Continuous auditing has its roots in financial auditing, and the first theoretical model was proposed in 2010 by O'Donnel [179]. The main problem with conventional auditing is a dependency on evidence-based reviews after a certain amount of time has passed since the actual business activities occurred. Practically, this carries the risk of auditing an organization too late. As such, there is a need for checking IT controls on a more frequent basis: nearly real-time, if possible.

According to the Global Technology Audit Guide (GTAG) [180], continuous auditing is defined as "*any method used by auditors to perform audit-related activities on a more continuous or continual basis*". The essence of continuous IS/IT auditing is to identify and assess vulnerabilities that can hurt organizations by establishing dynamic thresholds that enable dynamic and prompt responses to changes, as shown in Figure 42. As such, the results of continuous auditing, coupled with continuous monitoring enabled by technology, are substantial for the organization's sustainability, cost-effectiveness, and resource utilization optimization.

3.7.2. Revolutionary Development

The opposing view of revolutionary development [13] depends on the invention of advanced 4th-generation programming languages (4GL) that enable users to create software themselves rather than writing code from software developers; lately, 5th-generation programming languages with higher levels of automation and artificial intelligence (5GL) are included here as well. In this way, end-users are allowed to experiment with software development to satisfy specific needs. This viewpoint empowers freedom, which is in stark contrast to software development that is restricted to controlled methodologies with strict lifecycles. The revolutionary development approach is based on the premise that

business users should be allowed to experiment to generate software programs for their specific needs.

Figure 42. Continuous auditing model. Source: adapted from [181].

4. Discussion

Reviewing the literature on the evolution of software development methodologies results in no empirical evidence that a particular software development methodology is better than another. While the main advantages and disadvantages for each are pinpointed, there are plentiful influencing factors, such as the size of the organization and the team, its experience in the use of each methodology, the capability and maturity level of the organization, and the uniqueness of each software development project's needs. In principle, a methodology provides theoretical appropriateness, which is then a matter of adaption to the case of each different project.

Noticeably, traditional heavyweight methodologies require managerial strictness and process inelasticity and are often considered non-compatible with the modern, fast-moving, and constantly changing digital world. However, the transition to modern lightweight methodologies for faster and more flexible solutions delivery has been an ongoing challenge until recently. The simplicity offered by traditional methodologies, and specifically by Royce's Waterfall model, is crucial for project understanding and determination of goals by the team to eliminate concerns in day-to-day software development operations. Early evolutionary approaches were introduced by improving certain trade-offs of the widely used Waterfall model, enabling iterations, progressing incrementally, enhancing testing with V-model and VV-model, or developing prototypes rapidly.

Lightweight methodologies flourished after the publication of *The Agile Software Development Manifesto* in 2001, with an across-industries tendency for adaptation of the so-called agile mindset. Initial enthusiasm for agile methodologies was soon accompanied by skepticism concerning their practical implementation. Agile frameworks inevitably offer greater adaptation to change and the ability to work with limited visibility at the beginning of a project in comparison to traditional methodologies. Nevertheless, these benefits are often in contrast to specific project management requirements. For instance, widely accepted methodologies such as Scrum and XP lack documentation provisions, while in Kanban, there is an absence of timeframes, which can cause the team to miss important delivery milestones. The introduction of SAFe targets offers agile scaling capabilities in practice, but eliminating the self-organization of teams leads to the adoption of a top-down-like management approach regarding decision-making. In the same context, DevOps, which is among the latest agile frameworks, goes beyond existing standards regarding continuous delivery; however, it lacks the clear guidelines that are essential for teams.

Since there is no single software development methodology that fulfills the needs of all projects and environments, practitioners are under continuous experimentation in selecting combinations that serve their needs better, on a project-by-project basis. It

appears that plan-driven methodologies, predominantly the Waterfall model, are adapted by teams by adding iterative development. Similarly, agile-based methodologies are adapted with integrated traditional approach elements, such as architecture planning and baseline estimations and monitoring. These combinations of agile and traditional software development methodologies are considered hybrid methods, driven by team pragmatism to accomplish different goals of each project.

Post-processing data from previous research studies and the HELENA project in particular [6] were essential to verify team preference towards a particular methodology while establishing hybrid methods. It was determined that agility in software development is driven primarily by Scrum as the foundation of the new hybrid method. This was concluded from the fact that Scrum appears in four out of the top five selections between software development methodologies. The other methodologies that are mostly preferred for combination with Scrum are iterative development, prototyping, Kanban, and DevOps. Furthermore, it is highlighted that the Waterfall methodology remains highly favorable for establishing hybrid methods, appearing in the third position and preferably used with iterative development. Notably, further research on the selection's triggers, as well as extending the research to multiple frameworks during the establishment of hybrid methods, will promote knowledge regarding the structuring mechanism of such methods.

Parallel to the evolution of software development methodologies, significant advancements occurred in the field of software architecture description during the last decades. In particular, the IEEE 1471-2000 [182] was published for the first time in 2000 in response to the need for conceptualization and standardization of architecture description, as a proposed standard format. This standard has evolved into ISO/IEC/IEEE 42010:2011 [183], which was last updated to its current version in 2007. After ten years of evolution driven by joint efforts from academics, industry professionals, and government officials, the result was the establishment of a foundation for systems architecture thinking, recognizing a common basis for methods, frameworks, and automation tools. It is worth mentioning that the era before the ISO and IEEE architecture description of systems is acknowledged as the architectural "Tower of Babel" [184].

There have been several attempts to capture software system architecture in one diagram that satisfies the concerns of all its stakeholders, clarifying the high-level structure of the design and implementation of the software. Notably, the "*4 + 1*" view of the model of software architecture, introduced by Kruchten in 1995 [185], is acknowledged as an important generic view incorporating the logical, process, physical, and development views, illustrated with the support of selected use cases. In recent years, following the rapid evolution of internet technologies and specifically distributed blockchain solutions, the sophisticated "*1 + 5*" architectural views model was introduced by Gorski in 2021 [186]. This updated version introduces additional UML language semantics to visualize IT systems architecture with respect to business processes, which are used to construct two UML profiles: UML profile for integration flows and UML for distributed ledger deployment.

The evolution of IS/IT auditing is associated with ongoing changes in the business environment, the technology landscape, sociopolitical trends, and the need for governance. Experimentation by researchers and technology sector practitioners to embrace agility has numerous challenges in connecting mindset with processes. Specifically, the IS/IT audit process is based on the ability to standardize controls and assess based on performance evidence. Thus, conducting IS/IT audits at organizations following traditional software development methodologies has significantly less complexity and tolerance than ISACA and ISO standards. However, the tendency towards agility adaption requires the IS/IT audit process to be aligned and cognizant of current trends. This need for agility has disrupted IS/IT auditing, triggering the exploration of the continuous auditing that is used in financial auditing. Agile and continuous IS/IT auditing has raised high interest among researchers and practitioners from the largest auditing organizations worldwide.

Assuring value creation in technology firms under the scope of adapted software development methodologies' viewpoint remains highly challenging. IS/IT audits are

perceived as in alignment with strict monitoring of IT governance controls, offering great conformance to evolutionary software development, which depends on SDLCs. Extending the research in agile framework adaptation of standard ISACA audit processes is considered essential. Moreover, there is substantial room for exploring IS/IT auditing capabilities from the revolutionary software development viewpoint.

5. Conclusions

Software development methodologies have evolved rapidly over the last fifty years, from the changes that occurred from the Code and Fix era and the Waterfall model of the early 1970s, to the latest modern philosophies of agility and hybrid development methods of today. This fascinating and ongoing journey of continuous research and adaptation to modern management philosophies by organizations affects the way teams work together, disrupting the fundamentals of conducting IS/IT audits. While agility is considered a goal by organizations across industries, for audit services, firms are a necessity when auditing agile-oriented organizations, providing the required freedom to auditors to understand and assess better software systems, data, people, and processes during audits. The need to assure value creation and adherence to industry standards, guidelines, and regulations remains present in technology firms. Likewise, the continuous evolution of IS/IT auditing is expected to be maintained in the future to align with the evolution of the technology landscape, changes in business models and sociopolitical trends, and the necessity of IT governance.

Author Contributions: Conceptualization, I.K.K. and T.P.P.; methodology, I.K.K. and T.P.P.; validation, I.K.K. and T.P.P.; formal analysis, I.K.K. and T.P.P.; investigation, I.K.K. and T.P.P.; resources, T.P.P.; writing—original draft preparation, I.K.K.; writing—review and editing, I.K.K. and T.P.P.; visualization, I.K.K. and T.P.P.; supervision, T.P.P.; funding acquisition, T.P.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All results of the international study "HELENA" on the use of Hybrid dEveLopmENt Approaches in software systems development can be found at https://helenastudy.wordpress.com/helena-results/publications/ (accessed on 15 August 2022).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nuša, E.; Rojko, K.; Lesjak, D. Traditional and new ICT spending and its impact on economy. *J. Comput. Inf. Syst.* **2022**, *62*, 384–396. [CrossRef]
2. Global ICT Spending, Forecast 2020–2023. Available online: https://www.idc.com/promo/global-ict-spending/forecast (accessed on 14 June 2022).
3. The role of ICT in the 4th Industrial Revolution. Available online: https://www.acts-net.org/events/past-events/58-the-role-of-ict-in-the-4th-industrial-revolution-4ir (accessed on 18 May 2022).
4. Veldhoven, Z.V.; Vanthienen, J. Digital transformation as an interaction-driven perspective between business, society, and technology. *Electron. Mark.* **2021**, *32*, 629–644. [CrossRef] [PubMed]
5. Zimmermann, A.; Schmidt, R.; Bogner, J.; Jugel, D.; Möhring, M. Software evolution for digital transformation. In Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), Funhal, Madeira, Portugal, 23–24 March 2018; pp. 205–212. [CrossRef]
6. Kuhrmann, M.; Diebold, B.; Münch, J.; Trektere, K.; McCaffery, F.; Garousi, V.; Felderer, M.; Linssen, O.; Hanser, E.; Prause, C. Hybrid software development approaches in practice: A European perspective. *IEEE Softw.* **2019**, *36*, 20–21. [CrossRef]
7. Wysocki, W. A hybrid software process management support model. In Proceedings of the 24th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems, Procedia Computer Science, Virtual Conference, 16–18 September 2020; pp. 2312–2321.
8. Vijayasarathy, L.; Butler, C. Choice of software development methodologies: Do organizational, project, and team characteristics matter? *IEEE Softw.* **2016**, *33*, 86–94. [CrossRef]

9. Manifesto for Agile Software Development. Available online: https://www.agilealliance.org/agile101/the-agile-manifesto (accessed on 21 May 2022).
10. Mateescu, G.; Vladescu, G. Auditing hybrid IT environments. *Int. J. Adv. Comput. Sci. Appl.* **2014**, *5*, 1–10. [CrossRef]
11. The New Equation. Available online: https://www.pwc.com/jp/en/press-room/pwc-the-new-equation210701.html (accessed on 9 May 2022).
12. Tell, P.; Klünder, J.; Küpper, S.; Raffo, D.; MacDonell, S.G.; Münch, J.; Pfahl, D.; Linssen, O.; Kuhrmann, M. What are hybrid development methods made of? An evidence-based characterization. In Proceedings of the International Conference on Software and Systems Process (ICSSP), Montréal, QC, Canada, 25 May 2019; IEEE Computer Society Press: Los Alamitos, CA, USA, 2019; pp. 105–114. [CrossRef]
13. Cannon, D.; O'Hara, B.; Keele, A. *CISA: Certified Information Systems Auditor, Study Guide*, 4th ed.; John Wiley & Sons, Inc.: Indianapolis, IN, USA, 2016.
14. Avison, D.; Fitzgerald, G. Methodologies for developing information systems: A historical perspective. In *The Past and Future of Information Systems: 1976–2006 and Beyond, Proceedings of the IFIP 19th World Computer Congress, TC-8, Information System Stream, Santiago, Chile, 21–23 August 2006*, 1st ed.; Avison, D., Elliot, S., Krogstie, J., Pries-Hege, J., Eds.; Springer: Berlin, Germany, 2006; Volume 214, pp. 27–38.
15. Royce, W. Managing the development of large software systems. *Proc. IEEE WESCON* **1970**, *26*, 328–388.
16. Senarath, U.S. Waterfall methodology, prototyping and agile development. *Tech. Rep.* **2021**, 1–16. [CrossRef]
17. Thakur, D.; Deepti, S.; Chaudhary, A. A comparative study between waterfall and incremental software development life cycle model. *Int. J. Emerg. Trends Sci. Technol.* **2015**, *2*, 2202–2208. Available online: https://journals.indexcopernicus.com/api/file/viewByFileId/174825.pdf (accessed on 5 August 2022).
18. Ruël, H.J.M.; Bondarouk, T.; Smink, S. The waterfall approach and requirement uncertainty. *Int. J. Inf. Technol. Proj. Manag.* **2012**, *2*, 43–60. [CrossRef]
19. Waterfall Methodology: Working, Advantages & Disadvantages. Available online: https://www.analyticssteps.com/blogs/waterfall-methodology-working-advantages-disadvantages (accessed on 5 August 2022).
20. Basili, V.; Turner, J. Iterative enhancement: A practical technique for software development. *IEEE Trans. Softw. Eng.* **1975**, *1*, 390–396. [CrossRef]
21. Basili, V.; Larman, C. Iterative and incremental development: A brief history. *IEEE Comput. Soc.* **2003**, *36*, 76–79.
22. Incremental Delivery. Available online: https://www.gristprojectmanagement.us/software-2/incremental-delivery.html (accessed on 27 May 2022).
23. Boateng, K.O.; Nunoo-Mensah, H. eLAB: An electronic lab simulation tool. In Proceedings of the 1st Conference in Engineering, Science, Technology and Entrepreneurship, Kumasi, Ghana, 6–7 August 2015.
24. What is Incremental Model- Advantages, Disadvantages and When to Use It? Available online: http://tryqa.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/ (accessed on 5 August 2022).
25. Parthasarathy, S. Structured systems analysis and design method (SSADM). In *Systems Analysis Design & Introduction to Software Engineering*, 4th ed.; Everest Publishing House: Pune, India, 2015; pp. 41–51.
26. The use of SSADM (Structured Systems Analysis and Design Methodology) as a Standard Methodology on Information Systems Projects, Seminar Paper. 2001. Available online: https://www.grin.com/document/106034 (accessed on 5 August 2022).
27. Naumann, J.D.; Jenkins, A.M. Prototyping: The new paradigm for systems development. *MIS Q.* **1982**, *3*, 29–44. [CrossRef]
28. Helmy, M.A.W.; Hassan, N.; Mohd, Z.; Hanafi, H. Web based intelligent appointment system. In Proceedings of the Merapatkan Jurang Digital: Masyarakat Berpengetahuan, Model Malaysia, Kuala Lumpur, Malaysia, 18–19 March 2009; pp. 124–136.
29. Susanto Azhar, M. System development method with the prototype method. *Int. J. Sci. Technol. Res.* **2019**, *8*, 141–144.
30. Advantages and Disadvantages of Prototype Model. Prototyping Model in Software Engineering for Testing. Available online: https://www.aplustopper.com/advantages-and-disadvantages-of-prototype-model/ (accessed on 5 August 2022).
31. Weilkiens, T.; Lamm, J.G.; Stephan, R.; Markus, W. The V-model. In *Model-Based System Architecture*, 1st ed.; John Wiley & Sons: Hoboken, NJ, USA, 2016; pp. 343–352.
32. Fleischer, C.; Sauer, D.V.; Barreras, J.V.; Schaltz, E.; Christensen, A. Development of software and strategies for battery management system testing on HIL simulator. In Proceedings of the 11th International Conference on Ecological Vehicles and Renewable Energies (EVER 2016), Monte Carlo, Monaco, 6–8 April 2016. [CrossRef]
33. Using V Models for Testing. Available online: https://insights.sei.cmu.edu/blog/using-v-models-for-testing (accessed on 30 May 2022).
34. Durmuş, M.; Ustoglu, I.; Tsarev, R.; Börcsök, J. Enhanced V-Model. *Informatica* **2018**, *42*, 577–585. [CrossRef]
35. Boehm, B. A spiral model of software development and enhancement. *IEEE Comput.* **1988**, *21*, 61–72. [CrossRef]
36. Munassar, N.M.A.; Govardhan, A. A comparison between five models of software engineering. *IJCSI Int. J. Comput. Sci. Issues* **2010**, *7*, 94–101.
37. Bhosale, S. Spiral model: Applications in web based applications. *IPASJ Int. J. Comput. Sci.* **2014**, *2*, 1–4.
38. Doshi, D.; Jain, L.; Gala, K. Review of the spiral model and its applications. *Int. J. Eng. Appl. Sci. Technol.* **2021**, *5*, 311–316. [CrossRef]
39. Henderson-Selers, B.; Julian, M.E. The fountain model for object-oriented system development. *Object Mag. July-August* **1993**, *21*, 71–79.

40. Pillai, K. The fountain model and its impact on project schedule. *ACM SIGSOFT Softw. Eng. Notes March* **1996**, *21*, 32–38. [CrossRef]
41. Fountain Model in Software Development Life Cycle (SDLC). Available online: https://ugcnet-computerscienceguide.blogspot.com/2022/01/fountain-model-in-software-development.html (accessed on 5 August 2022).
42. Advantages and Disadvantages of Software Engineering Waterfall Model, Prototype Model, Fountain Model and V Model and Applicable Scenarios. Available online: https://blog.katastros.com/a?ID=00500-0a0f26d6-121e-4ef3-ae75-211c0c4e4f15 (accessed on 5 August 2022).
43. Fountain Model. Available online: https://blog.actorsfit.com/a?ID=01500-a412fec7-2ed1-4837-917b-8157ee32a0b4 (accessed on 8 August 2022).
44. Martin, J. *Rapid Application Development*, 3rd ed.; Macmillan Publishing Company: New York, NY, USA, 1991.
45. Abd Ghadas, Z.A.; Wan Ismail, W.N.S.; Abdul Aziz, A.; Harun, N.A.; Jusop, M.; Rahman, C.A. LAFAMS: Account management system for Malaysian small legal firms. *Pertanika J. Soc. Sci. Humanit.* **2015**, *23*, 239–250.
46. What is RAD Model? Phases, Advantages and Disadvantages. Available online: https://www.guru99.com/what-is-rad-rapid-software-development-model-advantages-disadvantages.html (accessed on 8 August 2022).
47. The Advantages and Disadvantages of RAD Software Development. Available online: http://www.my-project-management-expert.com/the-advantages-and-disadvantages-of-rad-software-development.html (accessed on 8 August 2022).
48. Kirpitsas, I. Implementation of Web-Based Application for Self Assessment of Professional Qualifications. Master's Thesis, Interdepartmental Programme of Postgraduate Studies (I.P.P.S.) in Information Systems, University of Macedonia, Thessaloniki, Greece, 15 November 2019; pp. 32–44.
49. Booch, G. The UML and the rational unified process. *IEEE Softw.* **2020**, *37*, 12.
50. Jacobson, I.; Booch, G.; Rumbaugh, J. The unified process. *IEEE Softw.* **1999**, *16*, 96–102.
51. What are the Advantages of Unified Process in Software Engineering? Available online: https://www.raiseupwa.com/miscellaneous/what-are-the-advantages-of-unified-process-in-software-engineering/ (accessed on 8 August 2022).
52. The Difference Between Agile and Unified Process Methodology. Available online: https://blog.bydrec.com/agile-vs-unified-process-methodology (accessed on 8 August 2022).
53. Microsoft Solutions Framework (MSF). Available online: http://architectureportal.org/microsoft-solutions-framework (accessed on 1 June 2022).
54. Microsoft Solutions Framework v3 Overview. Available online: https://www.researchgate.net/publication/236735939_Microsoft_Solutions_Framework_v3_Overview (accessed on 30 May 2022).
55. Giotis, T.C. How to deliver successful IT projects using MSF team model and MSF process model. In Proceedings of the PMI®Global Congress, Budapest, Hungary, 14–16 May 2007.
56. Microsoft Solutions Framework. Basic Principles. Available online: https://newline.tech/microsoft-solutions-framework-basic-principles/ (accessed on 8 August 2022).
57. The Advantages and Disadvantages of the MSF Method. Available online: https://blog.actorsfit.com/a?ID=01300-6cacab87-424d-48d0-8893-550f7a919986 (accessed on 8 August 2022).
58. Johnson, J. *Bing Bang Boom*; Technical Report; The Standish Group International Inc.: Boston, MA, USA, 2014. [CrossRef]
59. Cowboy Coding: Code & Fix Model. Available online: https://study.com/academy/lesson/cowboy-coding-code-fix-model.html (accessed on 3 June 2022).
60. Matkovic, P.; Tumbas, P. A comparative overview of the evolution of software development model. *J. Ind. Eng. Manag.* **2010**, *1*, 163–172.
61. Sherrell, L.; Chen, L. The w life cycle model and associated methodology for corporate web site development. *Commun. AIS* **2001**, *5*, 7.
62. Douglas, B.P. ROPES: Rapid object-oriented process for embedded systems. In *Doing Hard Time: Developing Real-Time Systems using UML, Objects, Frameworks, and Patterns Reading*; Addison-Wesley Professional: Boston, MA, USA, 1999; pp. 14–16.
63. Parallel Development, IBM Rational Synergy 7.2.0. Library. Available online: https://www.ibm.com/docs/en/rational-synergy/7.2.0?topic=synergy-parallel-development (accessed on 18 May 2022).
64. Boehm, B.; Port, D.; Yang, Y. WinWin spiral approach to developing COTS-based applications. In *COTS-Based Software Systems*; EDSER-5 Position Paper; Dean, J., Grave, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2003.
65. Hasselbring, W. Component-based software engineering. *Int. J. Softw. Eng. Knowl. Eng.* **2002**, *17*, 289–305. [CrossRef]
66. The Architecture Based Design Method. Available online: https://www.researchgate.net/publication/235088008_The_Architecture_Based_Design_Method (accessed on 22 May 2022).
67. Kravets, A.; Shcherbakov, M.; Kultsova, M.; Tadashi, I. Knowledge-based software engineering. In Proceedings of the 11th Joint Conference, JCKBSE 2014, Volgograd, Russia, 17–20 September 2014; pp. 156–171.
68. How to Get Agile Right, Boston Consulting Group. Available online: https://www.bcg.com/featured-insights/how-to/agile (accessed on 16 May 2022).
69. Williams, L.; Cockburn, A. Guest editors' introduction: Agile software development: It's about feedback and change. *IEEE Comput.* **2003**, *36*, 39–43. [CrossRef]
70. Mccauley, R. Agile development methods poised to upset status quo. *ACM SIGCSE Bull.* **2003**, *33*, 14–15. [CrossRef]
71. Schuh, P. *Integrating Agile Development in the Real World*; Charles River Media, Inc.: Needham, MA, USA, 2004; ISBN 1584503645.

72. Disciplined Agile Software Development: Definition, Ambysoft Inc. Available online: http://www.agilemodeling.com/essays/agileSoftwareDevelopment.htm (accessed on 28 May 2022).
73. Crystal Clear a Human-Powered Methodology for Small Teams. Including The Seven Properties of Effective Software Projects. 2004. Available online: https://www.researchgate.net/publication/234820806 (accessed on 30 May 2022).
74. Cockburn, A. *Crystal Clear: A Human-Powered Methodology for Small Teams: A Human-Powered Methodology for Small Teams*, 1st ed.; Fuller, J., Ed.; Addison-Wesley Professional: Boston, MA, USA, 2004.
75. Anwer, F.; Aftab, S.; Waheed, U.; Muhammad, S. Agile software development models TDD, FDD, DSDM, and crystal methods: A survey. *Int. J. Multidiscip. Sci. Eng.* **2017**, *8*, 1–10.
76. Crystal Method in Agile. Available online: https://www.toolsqa.com/agile/crystal-method/ (accessed on 8 August 2022).
77. Crystal Methods in Agile Development/Framework. Available online: https://www.geeksforgeeks.org/crystal-methods-in-agile-development-framework/ (accessed on 8 August 2022).
78. Bayer, S.; Highsmith, J. Radical software development. *Am. Program.* **1994**, *7*, 35–42.
79. Than, M.Z. An Analysis on Adaptive Software Development (ASD) Framework. *Tech. Rep.* **2012**. Available online: https://www.researchgate.net/publication/360083381_An_Analysis_on_Adaptive_Software_Development_ASD_Framework (accessed on 30 May 2022). [CrossRef]
80. Adaptive Software Development (ASD). Available online: https://airfocus.com/glossary/what-is-adaptive-software-development/ (accessed on 8 August 2022).
81. Characteristics of Adaptive Software Development. Available online: https://www.geeksforgeeks.org/characteristics-of-adaptive-software-development/ (accessed on 8 August 2022).
82. 15th Annual State of Agile Report. Available online: https://digital.ai/resource-center/analyst-reports/state-of-agile-report (accessed on 8 August 2022).
83. SCRUM Development Process. Available online: https://scrumorg-website-prod.s3.amazonaws.com/drupal/2016-09/Scrum%20OOPSLA%201995.pdf (accessed on 7 June 2022).
84. Mathai, M.; Venugopal, R.; Abraham, J.T. Hybrid model for software development. *Int. J. Res. Eng. Technol.* **2016**, *5*, 198–202.
85. Usmani, N.; Farooqui, S.; Ali, M.; Mahmood, W. Benefits to organizations after migrating to scrum. In Proceedings of the 29th International Business Information Management Association Conference, Vienna, Austria, 3–4 May 2017; pp. 3815–3828.
86. Hema, V.; Thota, S.; Kumar, S.; Padmaja, C.; Krishna, C.; Mahender, K. Scrum: An effective software development agile tool. In Proceedings of the IOP Conference Series: Materials Science and Engineering, Warangal, India, 9–10 October 2022; Available online: https://iopscience.iop.org/article/10.1088/1757-899X/981/2/022060/pdf (accessed on 5 August 2022).
87. Morampudi, N.; Gaurav, R. Evaluating strengths and weaknesses of agile scrum framework using knowledge management. *Int. J. Comput. Appl.* **2013**, *65*, 1–6. Available online: https://research.ijcaonline.org/volume65/number23/pxc3886058.pdf (accessed on 8 August 2022).
88. Top Scrum Master Challenges & Ways to Overcome Them. Available online: https://www.knowledgehut.com/blog/agile/5-hurdles-that-scrum-masters-commonly-face (accessed on 8 August 2022).
89. Zafar, I.; Nazir, A.; Abbas, M. The Impact of Agile Methodology (DSDM) on Software Project Management. In Proceedings of the International Conference on Engineering, Computing & Information Technology ICECIT 2017, Akdeniz University, Antalya, Turkey, 21–23 August 2017; pp. 1–6.
90. The DSDM Agile Project Framework 2014 Onwards. Available online: https://www.agilebusiness.org/page/TheDSDMAgileProjectFramework (accessed on 8 August 2022).
91. Pareto Principle. Available online: https://corporatefinanceinstitute.com/resources/knowledge/economics/pareto-principle/ (accessed on 7 August 2022).
92. DSDM—Dynamic Systems Development Method. Available online: https://mark-whitfield.com/dsdm-dynamic-systems-development-method/ (accessed on 22 August 2022).
93. Dynamic Systems Development Methodology. Available online: https://www.ukessays.com/essays/information-systems/dynamic-systems-development-methodology.php (accessed on 8 August 2022).
94. Jeff De Luca on Feature Driven Development. Available online: https://www.it-agile.de/fileadmin/docs/FDD-Interview_en_final.pdf (accessed on 6 June 2022).
95. Feature Driven Development & Empirical Modelling. Available online: https://warwick.ac.uk/fac/sci/dcs/research/em/publications/web-em/04/featurelist.pdf (accessed on 9 June 2022).
96. Palmer, S.; Felsing, J. *A Practical Guide to Feature Driven Development*, 1st ed.; Prentice Hall: Haboken, NJ, USA, 2002; p. 57.
97. What Is DSDM? Available online: https://www.codeproject.com/Articles/5097/What-Is-DSDM (accessed on 7 August 2022).
98. Feature Driven Development (FDD): An Agile Methodology. Available online: https://www.toolsqa.com/agile/feature-driven-development/ (accessed on 8 August 2022).
99. Feature Driven Development (FDD). Available online: https://code-mentor.org/feature-driven-development-fdd/ (accessed on 8 August 2022).
100. Beck, K. Extreme Programming Explained, 1st ed.Addison-Wesley Professional: Boston, MA, USA, 1999.
101. Kumar, R.; Maheshwary, P.; Malche, T. Inside agile family software development methodologies. *Int. J. Comput. Sci. Eng.* **2019**, *7*, 650–660. [CrossRef]

102. Yadav, K.; Yasvi, M.; Shubhika. Review on extreme programming-XP. *Int. J. Adv. Electron. Comput. Sci.* **2019**, *6*, 21–27. Available online: http://www.iraj.in/journal/journal_file/journal_pdf/12-574-156567485721-27.pdf (accessed on 7 June 2022).
103. Extreme Programming (XP). Available online: https://teachcomputerscience.com/extreme-programming-xp/#Disadvantages_of_Extreme_Programming (accessed on 7 June 2022).
104. Beck, K. *Test Driven Development: By Example*, 1st ed.; Addison-Wesley Professional: Boston, MA, USA, 2002.
105. Parsons, D.; Lal, R.; Lange, M. Test driven development: Advancing knowledge by conjecture and confirmation. *Future Internet* **2011**, *3*, 281–297. [CrossRef]
106. 6 Compelling Benefits of (TDD) Test Driven Development. Available online: https://www.knowledgehut.com/blog/agile/6-compelling-benefits-of-tdd-test-driven-development (accessed on 8 August 2022).
107. Advantages and Disadvantages of Test Driven Development (TDD). Available online: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-test-driven-development-tdd/ (accessed on 8 August 2022).
108. Cawley, O.; Wang, X.; Richardson, I. Lean software development—What exactly are we talking about? In Proceedings of the 4th International Conference on Lean Enterprise Software and Systems (LESS), Galway, Ireland, 1–4 December 2013.
109. Advantages of Lean Software Development. Available online: https://www.instinctools.com/blog/advantages-of-lean-software-development/ (accessed on 8 August 2022).
110. Larman, C.; Vodde, B. *Large-Scale Scrum, More with LeSS*, 1st ed.; Addison-Wesley Professional: Boston, MA, USA, 2016; pp. 10–13.
111. Introduction to LeSS. Available online: https://less.works/less/framework/introduction (accessed on 8 August 2022).
112. Large Scale Scrum (LeSS): A Short & Crisp Introduction. Available online: https://echometerapp.com/en/large-scale-scrum/ (accessed on 8 August 2022).
113. Nine Disadvantages of LeSS, From Someone Who's Doing It. Available online: https://seattlescrum.com/nine-disadvantages-of-less/ (accessed on 8 August 2022).
114. Ahmad, M.O.; Dennehy, D.; Conboy, K.; Oivo, M. Kanban in software engineering: A systematic mapping study. *J. Syst. Softw.* **2017**, *137*, 96–113. [CrossRef]
115. A Lean Approach to Efficient Workflow Management. Student Guide. *Lean, Agile & Kanban Processes for Software Projects by Evan Leybourn.* Available online: https://theagiledirector.com/images/LeanKanban.pdf (accessed on 26 May 2022).
116. Ganev, P. Advantages and Disadvantages of Using Scrum, Kanban and Scrumban for Software Development. Available online: https://www.academia.edu/36983292/Peter_Ganev_Advantages_and_disadvantages_of_using_Scrum_Kanban_and_Scrumban_for_software_development (accessed on 8 August 2022).
117. Ahmad, M.; Oivo, M.; Markkula, J. Kanban in software development: A systematic literature review. In Proceedings of the 39th Euromicro Conference Series on Software Engineering and Advanced Applications, Santander, Spain, 4–6 September 2013; pp. 9–16. [CrossRef]
118. Ambler, S.; Nalbone, J.; Vizdos, M. *Enterprise Unified Process: Extending the Rational Unified Process*, 1st ed.; Prentice Hall PTR: Hoboken, NJ, USA, 2005; ISBN 0-13-191451-0.
119. Edeki, C. Agile unified process. *Int. J. Comput. Sci. Mob. Appl.* **2013**, *1*, 13–17.
120. Agile Software Development Methods: What is the Agile Unified Process? Available online: https://blog.bydrec.com/agile-software-development-methods-what-is-the-agile-unified-process (accessed on 8 August 2022).
121. The Agile Unified Process (AUP). Available online: https://www.methodsandtools.com/archive/archive.php?id=21 (accessed on 8 August 2022).
122. Yarlagadda, R.T. DevOps and its practices. *SSRN Electron. J.* **2021**, *9*, 111–119.
123. What is DevOps? Atlassian. Available online: https://www.atlassian.com/devops (accessed on 19 August 2022).
124. Talks We Like: 10+ Deploys Per Day: Dev and Ops Cooperation at Flickr, by Hammond and Allspaw. Available online: https://www.rundeck.com/blog/twl-10-deploys-per-day-hammond-allspaw (accessed on 8 August 2022).
125. Almeida, F.; Simões, J.; Lopes, S. Exploring the benefits of combining devops and agile. *Future Internet* **2022**, *14*, 63. [CrossRef]
126. DevOps as a Service: Advantages and Disadvantages. Available online: https://logicera.net/devops-as-a-service-advantages-and-disadvantages/ (accessed on 8 August 2022).
127. Disadvantages of using DevOps. Available online: https://www.3pillarglobal.com/insights/disadvantages-of-using-devops/ (accessed on 8 August 2022).
128. Bhavsar, K.; Gopalan, S.; Shah, V. Scrumban: An agile integration of scrum and kanban in software engineering. *Int. J. Innov. Technol. Explor. Eng.* **2020**, *9*, 1626–1634. [CrossRef]
129. Ladas, C. *Scrumban-Essays on Kanban Systems for Lean Software Development*; Modus Cooperandi Press: Seattle, WA, USA, 2009.
130. Aini, Q.; Budiarto, M.; Putra, P.; Santoso, N. Gamification-based The Kampus Merdeka Learning in 4.0 era. IJCCS Indones. *J. Comput. Cybern. Syst.* **2021**, *15*, 31–42. [CrossRef]
131. What is Scrumban? The Best Parts of Scrum and Kanban. Available online: https://www.process.st/scrumban/ (accessed on 8 August 2022).
132. Leffingwell, D.; Yakyma, A.; Knaster, R.; Jemilo, D.; Oren, I. *SAFe®Reference Guide, Scaled Agile Framework®for Lean Soft-1644 Ware and Systems Engineering*, 1st ed.; Addison-Wesley Professional: Boston, MA, USA, 2016; pp. 1–7.
133. Part I: Overview SAFe®. Available online: https://www.oreilly.com/library/view/safe-40-distilled/9780134209487/part01.html (accessed on 19 August 2022).

134. Introducing the Scaled Agile Framework. Available online: https://scalingsoftwareagility.wordpress.com/2011/10/23/introducing-the-scaled-agile-framework%E2%84%A2/ (accessed on 29 May 2022).
135. Benefits of Scaled Agile Framework (SAFe). Available online: https://www.tietoevry.com/en/blog/2019/06/benefits-of-scaled-agile-framework-safe/ (accessed on 8 August 2022).
136. Mar, K.; Schwaber, K. Scrum with XP. InformIT 2002. Available online: http://www.informit.com/articles/article.aspx?p=26057 (accessed on 8 August 2022).
137. Scrum and eXtreme Programming (XP). Available online: https://www.scrum.org/resources/blog/scrum-and-extreme-programming-xp (accessed on 8 August 2022).
138. Fuior, F. Key elements for the success of the most popular Agile methods. *Rev. Română Inform. Autom.* **2019**, *29*, 7–16. [CrossRef]
139. Pros and Cons of Scaled Agile Framework. Available online: https://premieragile.com/advantages-and-disadvantages-of-scaled-agile-framework/ (accessed on 8 August 2022).
140. Scrum is Not Enough: How to Sell the Benefits of Scrum + Extreme Programming. Available online: https://techbeacon.com/app-dev-testing/scrum-not-enough-how-sell-benefits-scrum-extreme-programming (accessed on 8 August 2022).
141. Glass, R.L. The state of the practice of software engineering. *IEEE Softw.* **2003**, *20*, 20–21. [CrossRef]
142. Küpper, S.; Rausch, A.; Andelfinger, U. Towards the systematic development of hybrid software development processes. In Proceedings of the International Conference on the Software and Systems Process ICSSP '18, Gothenburg, Sweden, 26–27 May 2018; pp. 157–161. [CrossRef]
143. Prenner, N.; Unger-Windeler, C.; Schneider, K. How are hybrid development approaches organized?—A systematic literature review. In Proceedings of the International Conference on Software and Systems Process ICSSP '20, Seoul, Korea, 10–11 October 2020; pp. 145–154. [CrossRef]
144. Introduction to Unified Modeling Language (UML), 3rd INSPIRATION Training 4–5 December 2012. Available online: https://www.gfa-group.de/web-archive/inspire/www.inspiration-westernbalkans.eu/5/9/5/3/7/7/Introduction_to_the_Uni-1691fied_Modeling_Language__UML_.pdf (accessed on 8 August 2022).
145. Types of UML Diagrams. Available online: http://www.peter-lo.com/Teaching/U08182/Types%20of%20UML%20Diagrams.pdf (accessed on 8 August 2022).
146. Kandl, S.; Elshuber, M. A Formal Approach to System Integration Testing. Available online: https://arxiv.org/ftp/arxiv/papers/1404/1404.6743.pdf (accessed on 9 August 2022).
147. What Is User Acceptance Testing (UAT): A Complete Guide. Available online: https://www.softwaretestinghelp.com/what-is-user-acceptance-testing-uat/ (accessed on 9 August 2022).
148. Gharajeh, M.S. Waterative model: An integration of the waterfall and iterative software development paradigms. *Database Syst. J.* **2019**, *10*, 75–81.
149. Kuhrmann, M.; Diebold, P.; Münch, J.; Tell, P.; Garousi, V.; Felderer, M.; Trektere, K.; Mccaffery, F.; Linssen, O.; Hanser, E.; et al. Hybrid software and system development in practice: Waterfall, scrum, and beyond. In Proceedings of the International Conference on Software and Systems Process ICSSP '17, Paris, France, 5–7 July 2017; pp. 30–39. [CrossRef]
150. Kuhrmann, M.; Nakatumba-Nabende, J.; Pfeiffer, R.H.; Tell, P.; Klünder, J.; Conte, T.; MacDonell, S.G.; Hebig, R. Complementing materials for the HELENA-Edu Study. *Tech. Rep.* **2019**. [CrossRef]
151. Gantz, S.D. *The Basics of IT Audit*, 1st ed.; Syngress Publications, Elsevier Inc.: Waltham, MA, USA, 2014; p. 16.
152. Sayana, A. The evolution of information systems audit. *ISACA J.* **2022**, *1*, 1–5.
153. Largest Companies by Market Cap. Available online: https://companiesmarketcap.com (accessed on 26 May 2022).
154. Global Internet Penetration Rate as of April 2022, By Region. Available online: https://www.statista.com/statistics/269329/penetration-rate-of-the-internet-by-region (accessed on 30 May 2022).
155. Friedman, T.L. *The World Is Flat: A Brief History of the Twenty-First Century*, 1st ed.; Penguin Books Ltd.: London, UK; ISBN 13 9780141022727.
156. Cybercrime Statistics, Surfshark. Available online: https://surfshark.com/research/data-breach-impact/statistics (accessed on 21 May 2022).
157. Grembergen, V.V. From IT governance to enterprise governance of IT: A journey for creating business value out of IT. In Proceedings of the conference on e-Business, e-Services, and e-Society, I3E 2010, Buenos Aires, Argentina, 3–5 November 2010; p. 3.
158. Introduction to COBIT, Its Role in IT Governance and How to Apply It in UCIT, Excerpts from University of Calgary IT Session. 5 June 2009, pp. 6. Available online: https://slideplayer.com/slide/1652467/ (accessed on 14 June 2022).
159. Information Systems Audit and Control Association (ISACA). *COBIT 5: A Business Framework for the Governance and Management of Enterprise IT*; ISACA: Rolling Meadows, IL, USA, 2012.
160. ISACA. *COBIT®2019 Framework: Introduction & Methodology*; ISACA: Schaumburg, IL, USA, 2019; p. 19.
161. Radovanovic, D.; Radojević, T.; Lucic, D.; Šarac, M. IT audit in accordance with Cobit standard. In Proceedings of the 33rd International Convention on Information and Communication Technology, Electronics and Microelectronics: MIPRO 2010, Opatija, Croatia, 25 May 2010; pp. 1137–1141.
162. IT Assurance Framework (ITAF) Fact Sheet. Available online: https://www.isaca.org/-/media/files/isacadp/project/isaca/why-isaca/fact-sheets/itaf-fact-sheet_0318.pdf?la=en&hash=AC15112AAB593ED96DB4866B6622C5302EF87C35 (accessed on 8 June 2022).

163. ITIL®4: The Framework for the Management of IT-Enabled Services. Available online: https://www.axelos.com/certifications/itil-service-management (accessed on 10 June 2022).
164. The TOGAF®Standard, 10th Edition. Available online: https://www.opengroup.org/togaf (accessed on 10 June 2022).
165. The Committee of Sponsoring Organizations (COSO). Available online: https://www.csu.edu/internalaudit/cosoandcobit.htm (accessed on 10 June 2022).
166. Val IT Framework. Available online: https://cio-wiki.org/wiki/Val_IT_Framework (accessed on 10 June 2022).
167. International Organization for Standardization. Available online: https://www.iso.org/home.html (accessed on 10 June 2022).
168. ISACA Standards, Guidelines, Tools and Techniques. Available online: https://www.isaca.org/en/resources/isaca-journal/issues/2020/volume-1/standards-guidelines-tools-and-techniques (accessed on 11 June 2022).
169. The International Standards of Supreme Audit Institutions. Available online: https://www.intosai.org/fileadmin/downloads/documents/open_access/ISSAI_100_to_400/issai_200/issai_200_en.pdf (accessed on 11 June 2022).
170. Chulani, S.; Williams, C.; Yaeli, A. Software development governance and its concerns. In Proceedings of the 1st international Workshop on Software Development Governance, Leipzig, Germany, 12 May 2008. [CrossRef]
171. IT Audit Manual. AFROSAI-E Information Technology Audit Guideline, 1st ed. Available online: https://afrosai-e.org.za/wp-content/uploads/2019/07/IT-Audit-Manual-2017-1st-Edition.pdf (accessed on 1 June 2022).
172. Sircar, S.; Nerur, S.P.; Mahapatra, R. Revolution or evolution? A comparison of object-oriented and structured systems development methods. *MIS Q.* **2001**, *25*, 457–471. [CrossRef]
173. Chong, E. Auditing agile-a brave new world. *ISACA J.* **2016**, *2*, 1–6.
174. Joshi, P.L. A review of Agile internal auditing: Retrospective and prospective. *Int. J. Smart Bus. Technol.* **2021**, *9*, 13–32. [CrossRef]
175. Javanmard, M.; Alian, M. Comparison between agile and traditional software development methodologies. *Cumhur. Univ. Fac. Sci. Sci. J.* **2015**, *36*, 1386–1394.
176. Agile Audit Practice, ISACA Now Blog. Available online: https://www.isaca.org/resources/news-and-trends/isaca-now-blog/2017/agile-audit-practice (accessed on 6 June 2022).
177. Mkoba, E.; Marnewick, C. Conceptual framework for auditing agile projects. *IEEE Access* **2020**, *8*, 126460–126476. [CrossRef]
178. Gartner Research: Adopting Agile in Audit, Gartner, Inc. 2019. Available online: https://www.gartner.com/en/audit-risk/trends/agile-auditing (accessed on 9 August 2022).
179. O'Donnel, J. Innovations in audit technology: A model of continuous audit adoption. *J. Appl. Bus. Econ.* **2010**, *10*, 11–20.
180. Codere, D. *Global Technology Audit Guide Continuous Auditing: Implications for Assurance, Monitoring, and Risk Assessment*; The Institute of Internal Auditors: Altamonte Springs, FL, USA, 2005; p. 7.
181. Vasarhelyi, M.A.; Teeter, R.A.; Krahel, J.P. Audit education and the real-time economy. *Issues Account. Educ.* **2010**, *25*, 405–423. [CrossRef]
182. IEEE 1471-2000. IEEE Recommended Practice for Architectural Description for Software-Intensive Systems. Available online: https://standards.ieee.org/ieee/1471/2187/ (accessed on 8 August 2022).
183. ISO/IEC/IEEE 42010:2011, Systems and Software Engineering—Architecture Description. Available online: https://www.iso.org/standard/50508.html (accessed on 8 August 2022).
184. ISO and IEEE Publish New Edition of Standard for Architecture Description of Systems. Available online: http://www.iso-architecture.org/ieee-1471/pr-42010-2011-12.html (accessed on 8 August 2022).
185. Kruchten, P. Architectural blueprints. The "4 + 1" view model of software architecture. *IEEE Softw.* **1995**, *12*, 42–50. [CrossRef]
186. Górski, T. The 1 + 5 architectural views model in designing blockchain and IT system integration solutions. *Symmetry* **2021**, *13*, 2000. [CrossRef]