# Development and Evolution of Agile
## Changes in a World of Change

**Thomas J Marlowe**
**Department of Mathematics & Computer Science**
**Seton Hall University**
thomas.marlowe@shu.edu

**Vassilka Kirova**
**Bell Labs Consulting**
vassilka.kirova@bell-labs-consulting.com

**Garett Chang**
**Highstep Technologies**
garett@highstep.com

**Omer Hashmi**
**Vice-President, Agile Brains Consulting**
ohashmi@agilebrainsconsulting.com

**Stephen P. Masticola**
**SPM Consulting**
steve.masticola@verizon.net

**Abstract**[1,2]

Agile software development is an approach first codified in the Agile Manifesto in 2001. This was a statement of core values that became associated with a set of principles and practices. Key ideas include early and constant customer involvement, self-organizing teams that embrace change, rapid delivery of value, short timeboxed iterations coordinated by a shared list of items—a product backlog and driven by user stories and use cases, clean code, test-driven development, and continuous integration. The values, principles, and practices have permeated the technical and business world, translated and modified to fit many domains, affecting both production and management. But as with any good idea, agility can be misinterpreted, or used when inappropriate. Even a proper implementation must be tempered with good understanding of the domain, overall context, and appropriateness of selected agile practices, and modified to fit the enterprise, the domain, and the problem. In this paper, we briefly trace the evolution of agile methods, placing them within a wider organizational framework, and offer guidelines for their use.

**Keywords:** Agile methods, agile software development, software engineering, Kanban, Lean, Scrumban, scaled agile, agile in context, automated regression testing, DevOps, business agility

### 1. Introduction

We present an overview and perspective on the current state of "Agile", both in and beyond the world of software engineering. The Agile "revolution" began with a product-driven, team and customer-centered view of software development, enunciated as a set of priorities and values in the 2001 Agile Manifesto [4]. This became associated with a set of principles and practices. Key ideas such as early and constant customer involvement, embracing change, self-organizing teams, short timeboxed iterations coordinated by a Product Backlog and driven by user stories and use cases, emergent requirements, and frequent delivery of usable minimal viable products (MVPs), have rapidly permeated software development. Other important ideas included clean code [77], test-driven development, refactoring [34], continuous integration, and transparency. Associated with agile methods is the notion of a situation-dependent but generally relaxed view of formality, when not needed. This has often been misinterpreted as lack of discipline or need of documentation, and has led to many pitfalls and disappointments in the implementation of agile [82].

A primary motivation for preferring agile processes to waterfall, spiral, and other software engineering approaches [91] is that developing software cannot be treated as a theoretically well-defined process in practice. Schwaber [100] differentiates between "theoretical" processes (i.e., those which can be specified in a white-box manner from well-understood *a priori* parameters) and "empirical" processes (i.e., those which are not predictable from theory.) Empirical processes cannot be successfully managed through up-front planning alone. Instead, they must be treated as black boxes and constantly monitored. Schwaber correctly notes that, software development processes are empirical because

1. Applicable first principles are not present
2. The process is only beginning to be understood
3. The process is complex
4. The process is changing

This argues strongly against a process relying on upfront planning in detail, and equally strongly suggests moving toward an approach controlling the process of development as it occurs, tolerant of change, and without a detailed advance plan. Table 1 presents a more complete description of the differences between theoretical and empirical domains.

| | Theoretical Modeling | Empirical Modeling |
|---|---|---|
| 1. | Typically needs fewer measurements; experimentation only for estimation of unknown model parameters | Requires extensive measurements as it relies entirely on experimentation for the model development |
| 2. | Provides information about the internal state of the process | Provides information only about that portion of the process that can be influenced by control actions |
| 3. | Promotes fundamental understanding of the internal workings of the process | Treats the process as a "black box" |
| 4. | Requires accurate and complete process knowledge | Requires no detailed process knowledge—only that output data obtainable in response to input changes |
| 5. | Not particularly useful for poorly understood and/or complex processes | Often the only alternative for modeling the behavior of poorly understood and/or complex processes |
| 6. | Naturally produces both linear and nonlinear process models | Requires special methods to produce nonlinear models |
| 7. | Naturally and effectively detects anomalies, leading to correction or model revision | Requires effort and interpretation in detecting errors and anomalies, and in subsequent correction/revision |

Table 1. Theoretical & Empirical Modeling (cf. Schwaber [100])

"Agile" thinking has not remained fixed but has evolved as a worldview. It matured into a number of established development methodologies such as XP [10], Scrum [101] and Crystal [103], affected the earlier, related Lean and Kanban methods [89], and scaled up into approaches for entire organizations, e.g., SAFe [98, 99] and LeSS [69], enabling adoption of agile practices and

[1] This paper itself evolved through team interaction from a synthesis of [75] and [76].

[2] The authors wish to thank Fr. Joseph Laracy of Seton Hall University for technical editing of this document.

development of agile competencies across teams, programs, product lines and organizations, not only for development processes but corporate management and supporting activities as well. Further, via DevOps [25, 111], agile impacts operations, marketing, vendor management, and customer support, as well as development and operations activities of IT organizations, with continuous and concurrent development, delivery, and deployment.

Agility as a worldview and process discipline has spread widely, with the appropriate changes, into many areas, including management and business processes. Together, Lean and Agile have also established themselves as leading methodologies in financial and insurance sectors, healthcare services, manufacturing, and other domains (see [18]).

But as with any good idea, agility can be taken to extremes. It is sometimes viewed like an infallible religion, and at times used, contrary to its core message, as a mere buzzword to put off difficult questions and pressing decisions. Agility is not without costs and tradeoffs and may require or benefit from modification or combination with other approaches, even within the software engineering domain. It is not a silver bullet, but a powerful approach that where warranted can be modified and implemented in a way that properly reflects the target domain's context.

The agile community distinguishes capital "A" Agile development methods from lower-case "a" agility as a trait, e.g., organizational agility—a results-driven embedded attribute of an organization, bringing resilience, speed, flexibility, attunement, and preparedness to deal with market changes and challenges, as a core organizational competency and source of competitive advantage. Here we focus on "Agile" methods, but use "agile" in both senses.

The rest of this paper is organized as follows. In Section 2, we consider the context that gave birth to the Agile Manifesto. In Section 3, we explore the evolution of Agile methods and examine challenges in their adoption. Section 4 reviews the modern agile development practices and the spread of agile thinking to other facets in technology organizations. In Section 5 we examine situations calling for modifications to agile frameworks and discuss enablers of agile adoption. Finally, in Section 6 we present our conclusions and future directions.

## 2. The context for the birth of agile

The birth of agile methods must be understood in the context of the history of software engineering. In the early years, without high-powered computers, user interfaces, graphics, or the Internet, most programs either encoded mathematical algorithms (often for experimentation or scientific/engineering applications) or managed simple data processing, delivered by the IT department to experts or specialists to use offline. There was no established software engineering discipline [81]—programming was largely *ad hoc*.

With a larger set of users, and a wider set of applications including process control and complex data processing supported by databases, the discipline of software engineering emerged. The principal software development model was the Waterfall [91], a feature-oriented, document-driven model with sequential phases (requirements gathering, specification, design, implementation, testing, deployment, and maintenance). This was a carryover from computer hardware design processes, influenced by algorithm design, where it had generally worked well. While the flow of information between phases was fairly clear, structures, activities, and notations differed between phases, were often *ad hoc*, and initially lacked the discipline of providing feedback to earlier phases.

Even enhanced, but still mainly sequential models (V-shaped, Incremental, Spiral, Concurrent Development [91]) with multiple development passes, feedback, and incremental development converging on a final work product, proved inadequate for many applications, especially with the advent of personal computers and the Internet, a greatly increased user community, and computer applications with graphical user interfaces, configurability, user-driven actions, and more, resulting in continuing difficulties. Projects became caught in the Project Triangle of Scope, Time, and Cost, to which one should add (desirable) Quality—with the scope largely predefined by contract and development along a broad frontier rather than narrowly focused, constraints on time and cost become onerous. As Sutherland [5] observed, attempting to control an empirical process such as software development with a predefined plan is an exercise in futility. Such a process has to be watched continuously and adjusted frequently to reflect the agile forces, see Figure 1.


Figure 1. The iron project triangle and agile forces at work

It is also understood that many software errors resulted from or remained undiscovered when using classical development models, especially with late testing. (For a nuanced discussion, see [8].) On the other hand, these classical methods have been extended and enhanced over the years, e.g., with extensive simulation and prototyping, and often with test-focused approaches, and still remain viable options for development of certain classes of applications in specific domain and project contexts.

This perceived "software crisis" was partially mitigated by an increased use of object-oriented (OO) languages [28] and modeling, and software engineering (OOSE) approaches, focusing on application entities and their responsibilities. These were facilitated by improvements in compilers, computer architecture, power and storage [43], development tools such as version control systems [38, 39], testing tools [97], software architectures [105], and integrated development environments (IDEs) [94], made even more significant by distributed computing and the Cloud.

This allowed for a more comprehensive and flexible approach to development, with the introduction of the (Rational) Unified Process (UP) and the Unified Modeling Language (UML) [51, 52], and slightly earlier, design patterns—a catalog of useful idioms and guidelines [35]. The Unified Process replaces classical "phases" with concurrent workflows and adds a dynamic view with four repeatable phases—inception, elaboration, construction and transition. It also offers a third dimension of best practices such as UML models and component-based architecture. UML and the UP provide a series

of consistent models with fairly straightforward transitions and low representational gap—although not without requiring some effort and judgment. Requirements are driven by use cases and scenarios, tracing user interactions with the system, including problems and exception flows, with documentation of extra-functional concerns in a supplementary specification [67]. The process is iterative and naturally incremental, facilitated by bounded scope use cases and (largely) separate development, extension, and modification of objects. OO development also increased reuse (often with modification) of code and decreased code duplication, as did the use of design patterns [35, 67, 77].

For all of that, in many development shops, the Unified Process, as used initially, required formal development and documentation of sets of artifacts corresponding to groups of UML models for each workflow. Moreover, while development would be iterative-incremental, the work products of early iterations were not necessarily "deliverables"—that is, did not always provide working software with clear value to the customer, leading to some of the same schedule and budget problems as had been encountered in the Waterfall era.

## 3. The evolution of agile software engineering

### 3.1 The Agile Manifesto and agile methods

In 2001, seventeen expert software practitioners and thought leaders in the OOSE community set forth the Agile Manifesto [4], a statement of values followed by a list of design principles. They stated, "Through [our] work we have come to value:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan."

The (rephrased) principles and guidelines included:

1. Continuous customer interaction, with early and continuous delivery of valuable software, on a timescale of a couple of months to, by preference, a couple of weeks.
2. Preparing for and adapting to changing requirements, even late in development.
3. The need for management, other business roles, and developers to work together daily throughout the project.
4. Valuing and trusting motivated individuals and self-organized teams as key to projects, giving them the environment and support they need, and trusting them to get the job done.
5. Face-to-face is the most efficient and effective method of conveying information to and within a development team.
6. Software development should prioritize simplicity, technical excellence, and good design. Working software is the primary measure of progress.
7. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
8. At regular intervals, the team should reflect on how to become more effective, then tune and adjust its behavior accordingly.

These principles and beliefs were soon codified into "agile methods" and frameworks such as Extreme Programming [12] and Scrum [101]. Following these principles, agile projects are structured into short iterations (now typically 2 weeks), during which all relevant activities/workflows, including definition of emergent requirements, requirements analysis, architecture, design coding, testing, and integration take place continuously and as much in parallel as practicable. The process structure is iterative-incremental: each iteration is expected to deliver working code, continuously converging on the product envisioned by the customer and the developers (see Figure 2).

These approaches emphasize dynamic discovery of requirements through ongoing customer interaction, definition of user stories, and fast and continuous delivery of value to the customer. (In some cases, the customer may be in-house, and, for software developed "on spec," one may then have to rely on experts and prospective users.) Formalism is deemphasized except where clearly is needed, or required for reasons such as compliance or security, but this doesn't imply lack of coding or organizational discipline. A key point, often not recognized or appreciated but central to agile methods, is the effective use of backlogs for re-focusing project effort and agreeing with the customer on scope adjustments/reduction ("Trimming the Tail") when needed. (Less frequently, this can actually improve or extend the project.)

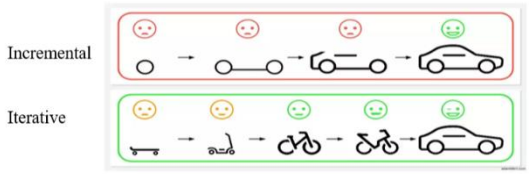


Figure 2. Incremental vs. iterative development (Hashmi [41])

In summary, agile methods appear best-suited for a capable, cross-functional team responsible for a feature, or a component such as a microservice, of a project, for fairly well-understood problems, in domains with which the agile teams have a certain degree of comfort.

### 3.2 Issues with agile process methods

Agile approaches spread quickly, but not without problems. Some groups, typically well-rooted in past practice, have misinterpreted parts of the guidelines and failed to align on the vision [45]. Others have followed the guidelines too rigidly, never realizing the expected benefits and stopping transformation activities too soon. And some have interpreted the scope of applicability quite narrowly, focusing only on development teams, leaving all other functions such as marketing or IT to work in their prior mode. The introduction of DevOps [25, 104], which initially enabled IT organizations to speed up their processes, spread to development organizations and helped create a uniform approach for rapid delivery of high-quality software features and solutions to the enterprise, markets and customers. Integration of agile and DevOps has further enabled external collaboration, partnerships, and the creation of value networks.

Two early failings, common to both the "misinterpreters" and the true believers, were to consider the entire application development process as covered by iterations, and all requirements as being equally suitable for continuous discovery. Notably, difficulties arose when, as discussed below, extra-functional/non-functional (NFR) requirements, such as latency, scalability, reliability, security, safety, or usability [91], were

treated as entirely evolutionary. Even a bigger problem arises when one assumes that all architectural decisions can be made during the iterations, and not allowing time to set a clear architectural direction at the start of the project. As agile matured, these issues have been recognized and addressed by such practices as the "architectural spike" [10] or "architectural runway" [69, 97]. Still another problem, particularly among management, is to assume that good people and good tools will automatically result in good development, without considering the need for training and team development. Finally, organizations often bypass reflection (at least, beyond team retrospectives) when things are going well, with some finding it difficult to fit reflection into the organizational culture [78].

In addition, there are always those who are taken by buzzwords, and whose understanding of agility in some cases is almost directly opposite to the original intent—that one could begin development without negotiation with the customer, just casual interaction, and without much effort in exploring or documenting requirements. Thankfully, most such enterprises either mature or disappear [119].

More importantly, using an agile process does not mean one can dispense with any of the following:

- A business case analysis for the development organization and for the client/potential customers, to consider the dimensions of the project, partner capabilities, the desirability of the work, the levels of interaction, and the expected Return on Investment (ROI) and resource estimation, as far as can be determined *a priori*. The determination of technical feasibility, especially with a high degree of innovation or novelty, or deployment on a new platform, is also critically important. Often these questions are investigated in a preliminary Exploration phase (called "Iteration Zero") before the first "development" iteration.
- Identifying, understanding, and accommodating essential requirements and risks, particularly external risks, and binding extra-functional/nonfunctional requirements such as accessibility constraints, security, privacy, safety, and timeliness in real-time applications, from the start, as largely inflexible constraints, including institution of standards for secure code (see [48, 117]) and safety [42]. This does not mean, however, that MVPs and initial releases need to address all of these in full generality.
- Serious and ongoing quality assessment beyond the "working software" metric, careful reviews and demos, and honest retrospectives to recognize the success and evaluate needed changes in project, process, and team execution, but never to assign blame to individuals (although this may not always be taken as seriously or objectively as it should be), with simulation, additional metrics, and other approaches for software quality assurance (SQA) [50]. This is especially important for features that cannot easily be demoed as functionality, either because they do not address increased user functionality, as in improved code refactoring feature, or because they relate to handling of physical emergencies (e.g., cardiac arrest) that cannot or should not be created for a demonstration.
- Last but not least, team preparation: one should not expect the team, or even more so a group of individuals who have yet to become a team, to learn new domains, new types of applications, or new language features, tools and practices on the fly while developing the application. There should be a certain degree of expertise at the start and if necessary, training, coaching, and consulting should also be made available to the teams and organizational management.

## 4. The use of agile in software engineering

In this section, we further consider how the practice of agile, object-oriented software engineering has itself evolved since the Manifesto in 2001. We look first at the technical aspects, and then at the impact of agility on technical and corporate management.

### 4.1 Modern practice

Some practices and approaches have been part of agile methods from the start. Agile methods preserve, and if anything, reinforce the (focused) use of systems of diagrams and notation such as UML, although with reduced formality. These methods also employ, even more than older OOSE approaches, patterns for design, requirements, testing, and other activities, together with refactoring [34, 61] guided by design patterns at the one end and driven by "code smells" [34, 77] and technical debt [23, 71] at the other. Further, they rely on tools including source control systems, test automation frameworks, especially for unit and interface testing, lightweight static code analyzers, and continuous integration environments.

Beyond this core, there have been some substantial developments in design and coding practices. While none of these practices violate the initial agile vision, each has evolved/enhanced the approach that an agile team might take in developing software, and even more so, the appearance of the resulting design and code, and often, the "form" of the resulting product.

- Component-, service-, or microservice-based software architectures, including specification and testing of component interfaces, possibly with mock objects and the use of design patterns such as Adapter and Façade [67, 77].
- A greater emphasis on full-stack development, largely as a result of mobile, web, Internet of Things [IoT], and data science applications. A full stack might include components such as: user interface, services and API, data and third-party services, application code, and business logic.
- Incorporation of Aspects [3, 13, 56] to varying degrees, or using dedicated microservices to handle cross-cutting concerns at the software architecture level, such as logging or security and access control services. Aspects were originally considered "non-OO," impractical, or in some cases dangerous, but are now generally accepted, and are integrated into languages and frameworks including Java8 and Spring, and recommended for implementation of larger or more complex systems [77].
  Aspects still need to be used with care, and agile teams need to adhere to the coding conventions that make aspects work (such as prefacing set-method names with the particle *set*). In addition, static code checking will be needed to ensure that AOP conventions are adhered to, or at least to flag potential trouble spots.
- Functional language features are now part of the agile toolkit, with languages such as Kotlin or Scala, and features such as lambda expressions in Java [59].

- A trend, noted in [77] as a significant change, toward shorter, more cohesive classes and methods, with dependencies in modes with low coupling [91] and reduced visible state, plus idiomatic situations such as collections and their members, especially in heavily used, critical, high-risk, or frequently modified code segments ("points of protected variation" [67]).
- Incorporation of robust automated testing, at all test levels: unit, component, subsystem, system, and deployment levels with functionality and performance aspects tested to customer requirements. See Section 5 for details.
- More robust change management at both technical and management levels [9], better automated traceability analysis [30, 72], and in general, more powerful, efficient, and effective static and dynamic analyses [66].
- An increased focus on internationalization [112]. These issues, however, are not unique to agile, and are not dealt with further in this paper.
- A greater emphasis on explicit consideration of run-time concurrency, with implications for design, programming, and testing [77].
- Approaches to deal with exploration and discovery and with continuous deployment.

Alone or in combination, these and other software technology changes and enhancements may introduce changes in the software architecture and new or different dependencies between the product backlog user stories and related tasks, affecting the content, ordering, and grouping of design and coding activities, and as such the content of the intermediate products. For example, designing and implementing a security or access control aspect can impact the order of design tasks and the technical content of iterations; even the implementation of a straightforward logging aspect will simplify (but lightly constrain) the design and coding of subsequent components and impact the following iterations.

### 4.2 Agile, Kanban, and Lean

Agile has both influenced and been influenced by Lean development [90] and kanban practices, and various enterprises have explored blending disciplined agile approaches such as Scrum with kanban and lean practices.

In the early development of agile, kanban was seen as a different and competing process management methodology, and this persists in some quarters [15]. Later developments integrated kanban into agile processes, taking advantage of its ability to control work in progress and focus an agile team's effort on the most important work ready to proceed.

The word "Kanban" literally means "work card" in Japanese. Kanban originated in the Toyota Production System as a method to precisely match effort with demand via a "pull" process of resource allocation. Kanban cards for parts production contained the type of part needed, how many were needed, and the stage of completion. Workers free to accept work would take a kanban card from a board of parts demands, attach it to a cart, and update the card as the required parts were placed on the cart and produced. A key concept of kanban is to limit the work in progress (WIP) to prevent inefficient context switching by workers; therefore, only a bounded number of kanban cards can be in progress at any given time, with that bound determined by the team's capacity to execute concurrent work items [46].

In more recent times, kanban is seen to fit well into agile at the team level and above [99, 114]. In the SAFe 5.0 process [96], kanban is an optional element used at the team level to control the completion of features within a sprint. Figure 3 shows how a team can use a kanban board to limit and control work in progress. A kanban board also fits well into agile processes as a "big visible information radiator" to make everyone concerned aware of the team's status and progress [2].

Highstep [44], initially a small company with limited resources, invested in development, maintenance, and support, has been slowly moving towards incorporating kanban principles in its software development process. Today, Highstep is a mature Scrumban organization, using an approach blending kanban with the discipline of Scrum [31, 37], but relaxing its strict time-boxed approach, adding more flexibility to deal with unpredictability, particularly in the service mode, without overly stressing personnel or resources.

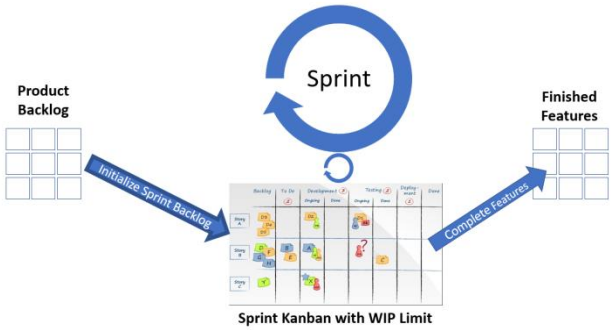


Figure 3. The kanban board as a process element (after [102])

### 4.3 Testing, test automation, and quality assurance in agile

Agile processes almost universally use *continuous integration* [27, 79] to continuously update a codebase shared by the entire agile team as features are created and modified. This requires a paradigm of testing different from that of the waterfall model. Even in the V-model extension of Waterfall [80], where a test strategy and partial test suite are developed early, there is only a single build-and test at the end of the project. In contrast, agile must execute the same V-model every sprint (or even test within a sprint), since any sprint can induce *regressions*, i.e., backward steps in development or changes that break formerly stable functionality. To prevent regressions from accumulating in the solution under agile development, the product must be regression-tested periodically, and ideally every sprint. We examine the support needed for this level of regression testing.

A mature software product can have thousands of features. Each must be tested when it is possible that development might break it (*change-impact testing*), and also periodically, even where breaking changes are not predicted (*general regression testing*[3]) [55]. Tools such as automated change management [9], program analyzers [66], and traceability bases [30, 72] can often predict the features or support elements that might be impacted by a change, but it is not possible to predict every change without being overwhelmed with false positives. Therefore, general

[3] Regression testing is not a test level and not limited to product or system testing; rather, it is an approach to be applied at every test level: unit, component, subsystem integration, product, and system.

regression testing, in addition to change-impact testing, is needed at the unit, component, subsystem, and system test levels. But manual regression testing at any test level is too slow and too labor-intensive (and too often imprecise) to be practical for agile development. Therefore, to produce acceptable-quality products, automated regression testing is essential.

An additional set of testing concerns arises with the use of aspects [10, 63], especially if test tools are not aspect-aware. Changes in features implemented with aspects, or the effects of changes in the code served by aspects (pointcuts), may not be visible in the implementation. Various problems can thus be introduced, including but not limited to type errors, non-deterministic semantics, and incorrect exception handling. In practice, many of these problems can be guarded against by using an automated unit test tool and test suite enriched to check for aspect-related problems. Regression testing must likewise be extended to deal with these considerations.

Test-driven development (TDD) [3] is often mistakenly identified as the single mechanism for developing test automation. TDD, while extremely useful, must be broadened from unit testing. Automated component, subsystem, and system tests can, and should, be created or updated while the architecture and interfaces of the system are being worked on.

Finally, to keep the product stable and the test base healthy, it is important in agile development to quickly identify and correct the cause of any failed automated test case. A failed automated regression test case may be due to any of the following:

- The feature has genuinely been broken due to a change in the code.
- The feature still works properly from a customer standpoint, but the automated test case has found inconsequential changes in behavior.
- There is a defect in the automated testbed or test case.

To avoid continually breaking builds, agile teams should have testbeds at their disposal at all times, and should run regression tests on any code before it is checked in to the build. Automated testing is typically used in agile environments as part of the continuous integration process, as well as at the developer level before new code is integrated.

### 4.4 Continuous deployment and configurability

There have been significant advancements in deployment strategies, particularly for *continuous deployment* [105], in which updates are continuously rolled out for customer feedback. The transition from successful testing into an actual production environment has often been a cause of concern. The intent is to cut over as quickly and as smoothly as possible in order to minimize downtime. One such modern strategy is the Blue Green deployment [32], with two near-identical production environments—a Blue Environment and a Green Environment— one active, serving the Production traffic, and the other as a staging environment for the final testing of the next release. Once a change is ready, it is released and the traffic redirected from the current Production Environment to the other, switching their roles. Not only does this allow for speedier deployment, it also supports rapid roll-back in case something goes wrong.

A related technique, used for continuous delivery, is hypothesis-driven development (HDD) [17, 89], resembling the cycles of science's hypothesis-and-test or of mathematical modeling, where experiments consist of tweaking the product, including its user interfaces and graphics elements, and gathering and analyzing user interactions, data, and feedback.

Another advanced technique is Feature Toggles, or Feature Flags [33]. Feature Toggling allows safe deployment of feature sets to a controlled user base by providing alternative code paths in the same deployment unit. Two major factors to be considered in the implementation of toggles are longevity (the expected feature toggle lifetime) and dynamism (its velocity of change). Release Toggles support Continuous Delivery by allowing in-progress latent, possibly incomplete or even untested, code to be shipped in a disabled flag state, and toggling it active once released.

Feature Activation, an analog of Feature Toggles, although related more to configuration management and access control, offers additional options. It can be used by developers, by software providers, by enterprises, and by individual customers or users. Developers can use it to support beta-testing [55, 109]—evaluation of a new or improved feature or interface by a small group of ordinary users—or as an adjunct to HDD, for comparative and concurrent study of different user interfaces or other features.

A software provider may use Feature Activation for pricing/licensing options or trial periods, or to provide variants for different user communities with different access permissions, data visibility, and views (for example, medical records software used by medical professionals, administrators, patients, and the public at large). IDE providers can offer academic versions, with limited access to advanced features but with student-oriented help and communication modules. Enterprises can use it as a first level of access control, and both enterprises and users can select a configuration when installing or personalizing software packages.

On the other hand, both Feature Toggles and Feature Activation are a good fit for some solution domains, but not all. They work in domains in which features can be turned on and off quickly and autonomously by the development organization or software provider, such as social media sites and apps with live updates, but not in domains where regulatory agencies must pre-approve new or changed features, or where there is high life-safety or economic risk.

### 4.5 Technical and corporate management

Three facets of technical management role and practices in the agile world can be distinguished: "caring" about agile teams, incorporating agility in technical management, and interfacing between agile development teams and corporate management, as well as other organizations and teams. The first generally means pulling back from detail management of team activities, allowing teams to be self-organizing and self-directing, while ensuring they have all the needed resources and expertise to deliver quality products, and further enabling individuals to develop their skills and progress toward their career goals.

Dedicated product owners (at different levels), assisted by technical managers, must maintain oversight: ensuring that product backlogs reflect the original and emerging requirements, coordinating client interaction, and keeping track of progress, backlog accuracy and prioritization, budget, and resources.

The transition at this level is not all that difficult, if sometimes bumpy, for firms that have understood and embraced agile values, but has been difficult when either the technical managers or their superiors did not understand or did not buy into agile methods, or were seriously invested in older approaches. The interface role in particular is problematic for a technical manager whose teams wish to follow agile processes, but who faces lack of understanding, support, or enthusiasm across technical and business management. This has led to enterprises specializing in shepherding companies in the transition to agile/lean philosophy and methodology (see, for example [41]).

Agile also extends beyond the product- and client-centered, incremental iterative development model for a single product, into continuous development. Agile software engineering works in concert with the DevOps model for software delivery and can effectively address facets like security: consider DevSecOps [70, 87]—a DevOps model that addresses security and privacy early and throughout the continuous exploration, development, integration, and deployment activities.

The flexibility of agile methodologies does not mean that every prior problem goes away. In particular, resource estimation still matters. Brooks' Law [14] still applies—adding new personnel late in the project may be counterproductive, as it may take more time and effort to get them up to speed and incorporate them in the team than they can possibly add to the project. Likewise, late addition of new tools or resources may not help, if it takes the team longer to become comfortable with them than the potential gains from their use. (But this of course does not argue against taking such actions to cope with exigent problems.)

Also, as can be expected, the self-organization, autonomy, and role-driven leadership of agile teams has boundaries, except perhaps in the smallest of startups, where almost everyone is on one team. In addition to a product owner, there are often additional roles, including but not limited to a Tech Lead, Line Manager, and typically a scrum-master or agile coach. The technical leader guides the technical implementation, and coordinates and ensures that role-driven leadership doesn't drift too far from the product vision, while the Line Manager assists on the administrative role of team management, resource allocations and work-life balance. The scrum-master, or team agile coach, serves as the agile evangelist, helping the team embrace agile principles, guiding the team on overall agility, and helping with team cohesion, while also resolving impediments and ensuring resource availability, and serving as a channel for management and leadership interactions. (With its mix of technical, administrative, support, and bureaucratic functionality, this position can often be hard to fill on an ongoing basis.) In larger organizations or for larger projects, there may be higher levels of governance—a product owner for an entire project or product line, a coordinator of multiple teams, and corporate management with responsibility for technical areas and/or customer contact.

Modern agile approaches extend up and out into corporate management and organizational modes of operation. While the technical guidelines are not fully applicable, agile management and business processes management emphasize agile features. Key activities include building interdisciplinary teams, strong and pervasive intra-enterprise communication, rapid decision making, frequent retrospectives, continuous customer interaction and product delivery, and adaptability to changing market demands, customer expectations, and unexpected events. Coordination, alignment, and continuous improvement are achieved through regular checkpoints to assess progress, status, and plans, realign to strategic objectives, and engage in periodic retrospection.

## 5. Applicability of agile methods

In this section, we first consider situations in which agile methods have limited applicability or need modification or extension to the nature of the domain or project. We then highlight key business enablers of agility.

### 5.1 Modifications and limitations

Critical extra-functional concerns may require some modification of agile practices. The need for an extensive (but possibly not complete) initial understanding of security, privacy, and accessibility constraints has resulted in enhancements of agile practices analogous to DevSecOps [70] and Privacy Engineering [40, 68]. Likewise, dealing explicitly with issues of timeliness, safety, and external interaction in real-time and active systems, requires changes in models and tools (compare the extensions in Real-Time UML [26]), and modifications in technical facets of agile processes, to accommodate, among other changes, greater formality and extensive system, stress, and platform testing.

Limitations on projects fall largely into five areas: product size and complexity, the development structure for large projects, the nature of the application, requirements for formality or consultation, and the need for creativity, novelty, and innovation.

First, agile may not scale well for large, monolithic components beyond a certain size, complexity, or level of distributed execution. Other software architecture approaches may be needed to establish a high-level decomposition and specify stable interfaces [73]. While agile development can readily handle transactional systems and other data-rich applications, including most data science applications, some system and data issues need to be addressed in advance. These include data modeling and representation issues, as well as protocols for interaction with smart devices (as in the IoT). Likewise, significant data collection and data cleaning where needed must precede timeboxed iterations resulting in software deliveries. However, the spread of microservices and containers, and the use of design patterns including database connections, facilitate the decomposition, development, and deployment of bounded context, cohesive services, thus somewhat mitigating the problem [58].

Second, large projects have required some adjustments [11, 24]. Agile methods for software development have been team-focused, but large projects require multiple teams, often distributed across multiple sites or in different business entities, and across time zones and countries. Business and technical processes need alignment [7, 53], particularly with respect to two specific concerns. First, as originally envisioned, agile methods required frequent, preferably face-to-face communication between the team members, and second, required teams to be in frequent communication, and each team to have continuous contact with the customer. The first problem has been largely resolved by larger-scale agile frameworks [69, 97], and by strategies including defining components with cohesive business logic and clear boundaries [74]. Inter-team

communication and collaboration is enabled by a combination of electronic means and visits [122]; customer contact can be handled by identifying a customer advocate—often the product owner or a designee. Remote teams [95] require some of the same adjustments, and others, as has become increasingly apparent in the year of the pandemic. At the other end of the spectrum, agile processes may need some modification for very small projects or teams [36].

Third, standard agile practices may not be appropriate for certain highly mathematical applications. Mathematical, statistical, and algorithmic software often has well understood specifications and requires minimal customer contact, except possibly for UX issues. Also, some process control systems, as in chemical manufacturing or power generation, or protocol-centered components as in networks or security algorithms, may not be well-suited for fixed-length short iterations and deliverables.

Fourth, some classes of applications may require a higher degree of compliance, governance, or formality. These may relate to safety, security, privacy, or intellectual property, have significant timing constraints, or involve cyber-physical systems. There may be legal, regulatory, or standards requirements for formal artifacts or even use of formal methods in domains including health, military, and national security applications. Others may require expert approvals at specific points, in addition to extensive customer contact. Another, related issue is getting signoff from regulatory bodies; these typically do not operate in an agile mode and certify only finished products rather than intermediates.

There is an increased interest in and use of hybrid methods, combining sequential and agile methods and practices for large system development [49, 62, 85, 113] and industrial applications [92] in particular. Hybrid approaches are also proposed to accommodate large or staged process deliverables [29, 85] or the need for more substantial requirements engineering [64]. IoT development also requires novel approaches to testing and dealing with non-determinism; a modified methodology for IoT application development is presented in [123] (see also [22]).

Nonetheless, agile methods with appropriate practices will almost always be the right choice for the development of individual components and user interfaces. Some guidelines remain good practice beyond their strict domain of applicability: identifying critical and core components, functionality, and exceptions, continuous client contact, regular and frequent team meetings supplemented by regular contact between teams, developing backups for key personnel, test-driven development including interface testing, prioritized backlogs, retrospectives, and more.

### 5.2 Dealing with exploration and discovery

Finally, agile development seems a natural fit for applications with a moderate degree of uncertainty about the feasibility of the solution and incremental innovation rather than fundamental discovery. Agile was not originally focused on speculative exploration or fundamental research—where major discovery, new algorithms, data structures, or models may be needed, or in data science applications that rely on pattern discovery using artificial intelligence, machine learning, or data mining. In such explorations, timeboxing may be inappropriate, backlog content may be hard to identify, and rapid or continuous delivery of partial solutions may not be possible.

But this does not mean that research cannot occur in the context of agile development. Indeed, agile teams frequently realize at some point that research is needed to implement planned features, either when they either do not know how to solve a specific problem and must invent a way, or when they must evaluate alternative approaches for solving the problem.

XP invented the concept of (*research*) *spikes* as time-boxed research efforts to solve highly specific problems. As research spikes are executed, the team gains confidence that they have workable solutions (or that a satisfactory one cannot be found, and they are at an impasse) [120]. Spikes also appear in implementing agility at organizational level, speeding up decision and learning cycles, and placing trust in individual teams to enable fast response to feedback and changes. This drives fast delivery of value, but also allows for "fail fast" cycles, fast corrections, and the initiation of discovery and innovation spikes where needed [93].

### 5.3 Business enablers of agility

The organization needs to build an agile culture as a foundation for operational agility across the entire enterprise. Managers and technical leaders must not just understand agility, but also serve as advocates, willing to argue for the benefit of what some might consider wasteful team activities: team meetings, time spent refactoring, and especially reflection. Technical and corporate management should understand agile processes, team dynamics, indicators of progress, the nature of backlog management, and metrics—not just working software delivered, but other measures of progress and quality. As mentioned above, this may entail engaging agile transformation consultants, with workshops and training for both technical and management staff—and perhaps for support staff such as Human Resources personnel as well.

As importantly, the client/customer must be open to ongoing contact and collaboration, alert to changes in environment and requirements, and accepting of deliverables at intermediate stages or continuously. The customer must also be willing to share information and provide feedback, early and continuously, and be comfortable with a contract that accepts late decisions. The customer should ordinarily not interfere with design or implementation details, except if they affect interfaces or agreements on emergent requirements.

Some specific practical issues with using agile in deliverable products need to be addressed in practice. We have already discussed issues with customer contact, but larger enterprises have additional challenges. Such businesses often have, not a single customer, but thousands to millions of users. Involving significant numbers of end customers in the day-to-day operations is almost always impractical, and businesses must modify agile to address this reality. For low-risk applications like social media platforms, it may be possible to continuously roll out updates and collect customer feedback [105]. Aspects of this approach have been codified as hypothesis-driven development (HDD, see above). Table 2 presents a summary.

| Enterprise | Customers | Handling |
|---|---|---|
| Small | Few | Direct team contact |
| Medium | Tens | Product owner or designee |
| Large | Hundreds | Management responsibility |
| Global | Thousands | HDD |

Table 2. Customer contact modalities

For high-risk applications, such as life-safety applications, continuous deployment is impractical because the risk of breaking the customer's business (as well as legal and financial risk to the enterprise) is excessive. For such applications, beta testing in customer-controlled sandbox environments is usually effective, especially if the beta-test customers have good tester mentalities.

Such barriers are comparable to those experienced in data science: resistance by senior management and technical staff; poor understanding or implementation; failure to adapt compatible business practices; failure to commit adequate resources, including training and team formation/structure; and failure to adhere to process. Each has one distinguishing problem: for data science, siloing of expertise and efforts within the enterprise; for agile processes, poor relationships with collaborators and clients, although siloing and functional vs. cross-functional teams can also be a problem for agile and DevOps [20].

Finally, management and the development team must agree that agile methods are appropriate for the project at hand [107]. Typically, corporate policy or development group structure dictates the process for all teams, but some companies such as Google now allow teams to select their own development method, as long as boundaries, infrastructure and integration rules for large or complex, multi-team, multi-site or cross-enterprise projects are observed.

### 5.4 Agility beyond software development

Agile as a concept is not limited to software development [60]. Agile methods for business functions and agile guidelines for customer contact, rapid and continuous development, team structure, and (implicitly) tool support can be used everywhere [19, 21, 47, 57]. With modification, they may be appropriate for development in knowledge-centric projects such as team authorship, or for just-in-time custom manufacture [116].

Their use for production is more problematic, but agile has been considered for the development of computer hardware [65, 86, 108], in construction [6, 16, 83], in manufacturing [1, 54, 88, 118, 121], and elsewhere [80, 110, 115]. These studies offer some reservations, but suggest that agility can offer some useful process and practice guidelines for particular industries.

## 6. Conclusions and future work

We have surveyed the birth and evolution of agile methods and their spread from a software engineering approach to a methodology for the entire software development enterprise, also looking briefly at its relation to DevOps and other approaches, and the interaction between agile and kanban. We considered the limitations of agile methodologies and the modifications they require, and described the enabling role organizational culture, business processes, and policies play.

In sum, the overall focus of agile has evolved, with its core principles now perhaps closer to the informal "modern" guidelines enunciated by Kerievsky [84].

- Make People Awesome
- Make Safety a Prerequisite
- Experiment & Learn Rapidly
- Deliver Value Continuously

where "safety" should be understood to embrace not only physical safety but also binding extra-functional constraints and critical risk factors including security, privacy, intellectual property protection, timeliness, or economic risk.

In the future, we plan to look more extensively at the use of agile in other domains, including 4.0 industries, and to examine further the relationship and interaction of agile with various Lean approaches and projects.


### Acknowledgments

We would like to thank Nagib Callaos for his support and encouragement, and Fr. Joseph Laracy, Margit Scholl, Helly Shah, and Robert Cherinka and Joseph Prezzama, for their comments and suggestions.


### References


1. A. Adomavicus (2018). 5 steps to an agile transformation in manufacturing. DevBridge, 06/27/2018. https://www.devbridge.com/articles/5-steps-agile-manufacturing/
2. Agile Alliance (n.d.). "Information radiator." https://www.agilealliance.org/glossary/information-radiators/
3. Agile Alliance (n.d.). "What is test-driven development?" https://www.agilealliance.org/glossary/tdd
4. The Agile Manifesto (2001). https://agilemanifesto.org/
5. Agility.im. How does an empirical process work? https://agility.im/frequent-agile-question/how-does-an-empirical-process-work/
6. M.A. Ajam (2017). Can we use Agile "Methods" in Construction Project? Applied Project Management, 17 October 2017.
7. S. Ambler (2009). Scaling agile software development through lean governance. SDG '09: Proceedings of the 2009 ICSE Workshop on Software Development Governance. May 2009, pp 1–2. https://doi.org/10.1109/SDG.2009.5071328
8. S. Ambler (2014). The Non-Existent Software Crisis: Debunking the Chaos Report. drdobbs.com. https://www.alejandrobarros.com/wp-content/uploads/old_old/Dr_Dobbs.pdf
9. Atlassian.com (n.d.) Jira--Issue and Project Tracking Software. https://www.atlassian.com/software/jira
10. A. Bajaj, O.Sangwan (2017). Aspect oriented software testing. 809-814. 10.1109/CONFLUENCE.2017.7943261
11. M. Balbes (2019). Can Agile Truly Scale to the Enterprise. The Agile Architect, 06 18 19. https://adtmag.com/articles/2019/06/18/balbes-return-agile-enterprise.aspx
12. K. Beck (2005). Extreme Programming Explained: Embrace Change, 2nd Edition (The XP Series). Pearson.
13. M.L. Bernardi, G.A. DiLucca (2005). Improving Design Patterns Modularity Using Aspect Orientation. IEEE International Workshop on Software Technology and Engineering Practice (STEP 2005), 209-210.
14. F.P. Brooks, Jr. (1995). The Mythical Man-Month. 1995 [1975]. Addison-Wesley.
15. T. Brown (2019). Scrum vs. kanban: Which agile framework is better? OpenSource.com, 08 08 19. https://opensource.com/article/19/8/scrum-vs-kanban
16. R. Burger (2019). Agile Construction Management. The Balance/Small Business, January 21, 2019.

https://www.thebalancesmb.com/what-is-agile-construction-management-845374
17. A. Cho (n.d.). Hypothesis-driven Development. IBM Garage Methodology. https://www.ibm.com/garage/method/practices/learn/practice_hypothesis_driven_development/
18. CollabNet Versionone (2019). 13th Annual State of AgileTM Survey.
19. K. Conboy, B. Fitzgerald (2004), Toward a conceptual framework of agile methods: a study of agility in different disciplines. WISER '04: Proceedings of the 2004 ACM Workshop on Interdisciplinary software engineering research, November 2004.
20. R. Conn (2020). How to Effectively Bridge The DevOps Skills Gap. OverOps, December 1, 2020. https://www.overops.com/blog/devops-skills-gap/
21. J. Conroy (n.d.). Can Agile PM be Applied Outside of IT Environments? https://www.projecttimes.com/articles/can-agile-pm-be-applied-outside-of-it-environments.html
22. C. Consel, M. Kabac (2014). Internet of Things: A Challenge for Software Engineering, ERCIM News 98, July 2014.
23. T. Coq et al. (n.d.). Introduction to the Technical Debt Concept. https://www.agilealliance.org/introduction-to-the-technical-debt-concept/
24. P. Diebold, A. Schmitt, S. Theobald (2018). Scaling agile: how to select the most appropriate framework. XP '18: Proceedings of the 19th International Conference on Agile Software Development: Companion, May 2018, Article No.: 7, 1–4. https://doi.org/10.1145/3234152.3234177
25. DORA—DevOps Research & Assessment (2019). 2019 Accelerate State of DevOps Report. https://cloud.google.com/devops/state-of-devops/
26. B.P. Douglass (2004). Real Time UML: Advances in the UML for Real-Time Systems (3rd Ed.). Pearson Education.
27. Duvall, P., Matyas, S., Glover (2007). A. Continuous Integration: Improving Software Quality and Reducing Risk. Pearson Education, Inc., 2007.
28. E. Elliott (2018). The Forgotten History of OOP. October 31, 2018. https://medium.com/javascript-scene/the-forgotten-history-of-oop-88d71b9b2d9f
29. U. Eriksson (2016). How to Make Agile and Waterfall Methodologies Work Together. White Paper, ReQTest, 6 December 2016.
30. A. Espinosa, J. Garbajosa (2011), A study to support agile methods more effectively through traceability. Innovations Syst Softw Eng 7:53–69, DOI 10.1007/s11334-011-0144-5
31. A. Fictner (n.d.). Kanban is the New Scrum. https://hackerchick.com/kanban-is-the-new-scrum/
32. M. Fowler (2010). Blue-Green Deployment. https://martinfowler.com/bliki/BlueGreenDeployment.html
33. M. Fowler (2017). Feature-toggles (AKA Feature flags). https://martinfowler.com/articles/feature-toggles.html
34. M. Fowler (n.d.) Refactoring. https://refactoring.com/
35. E. Gamma, R. Helm, R. Johnson, J. Vlissides (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, 1994. ISBN 978-0-201-63361-0.
36. A. Gancarczyk, L. Griffin (2019). The Small Scale Agile Manifesto: These six values enhance agile methodologies to help smaller teams work more efficiently. 30 Jan 2019. https://opensource.com/article/19/1/small-scale-agile-manifesto
37. I. Germanov (2019). Kanban vs Scrum vs Scrumban (2019): What Are The Differences? 08 12 2019. https://ora.pm/blog/scrum-vs-kanban-vs-scrumban
38. Git (n.d.). Git—Distributed even if your workflow isn't. https://git-scm.com/
39. GitHub Guides (2016). Hello World—GitHub Guides. https://guides.github.com/activities/hello-world/
40. I. Hadar, T. Hasson, et al (2018). Privacy by designers: software developers' privacy mindset. Empirical Software Engineering 23 (1), 259-289, February 2018.
41. O. Hashmi, O. (2019). Agile Product Delivery [PowerPoint slides]. Agile Brains Consulting. https://www.agilebrainsconsulting.net/
42. R.D. Hawkins, T.P. Kelly (2009). Software Safety Assurance – What Is Sufficient? 4th IET International Conference on Systems Safety, incorporating the SaRS Annual Conference. https://www-users.cs.york.ac.uk/~rhawkins/papers/HawkinsKelly%20IET%2009.pdf
43. J.L. Hennessy, D.A. Patterson (2019). A New Golden Age for Computer Architecture. Communications of the ACM, February 2019, 62 (2), 48-60. https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext
44. Highstep (n.d.). A software design, development, and delivery company. https://highstep.com/
45. E. Hochmüller, R.T. Mittermeir (2008). Agile Process Myths. APOS '08: Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral. May 2008, pp 5–8. https://doi.org/10.1145/1370143.1370145
46. C. Hollingsworth (2011). What Kanban can do. PM Network, 25(3), 66–67. https://www.pmi.org/learning/library/kanban-template-software-task-management-4367
47. D.X. Houston (2014). Agility beyond software development. ICSSP 2014: Proceedings of the 2014 International Conference on Software and System Process, May 2014.
48. M. Howard, D. LeBlanc, J. Viega (2010). 24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them (1st Edition). McGraw-Hill, 2010.
49. Intland Software (2019). Agile-Waterfall Hybrid: Smart Approach or Terrible Solution? White Paper, 2019. https://content.intland.com/blog/agile/agile-waterfall-hybrid-smart-approach-or-terrible-solution
50. ISO/IEC 25010:2011 (2011). Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models.
51. I. Jacobson, M. Christerson; P. Jonsson; G. Overgaard (1992). Object Oriented Software Engineering. Addison-Wesley ACM Press. ISBN 0-201-54435-0.
52. I. Jacobson, G. Booch, C. Rumbaugh (1998). The Unified Software Development Process, Addison Wesley. ISBN 0-201-57169-2.
53. N. Jastroch, V. Kirova, C. Ku, T.J. Marlowe, M. Mohtashami (2011), Adapting Business and Technical Processes for Collaborative Software Development. 17th International Conference on Concurrent Enterprising, June 2011.
54. X. Jun (2009). Agile Manufacturing Model of Small and Medium-Sized Manufacturing Enterprises. 2009 Second International Conference on Future Information Technology and Management Engineering.

55. C. Kaner, J. Falk, H.Q. Nguyen (1999). Testing computer software. John Wiley and Sons, Inc.
56. J. Kanjililal (2016). My two cents on aspect-oriented programming. InfoWorld. https://www.infoworld.com/article/3040557/my-two-cents-on-aspect-oriented-programming.html
57. J. Kanjililal (nd). 4 agile best practices every enterprise architect should follow. TechBeacon. https://techbeacon.com/app-dev-testing/4-agile-best-practices-every-enterprise-architect-should-follow?
58. J. Karlsson (2019). Principles of good large-scale Agile. What works—and what doesn't—when it comes to Agile at scale. https://www.javacodegeeks.com/2019/01/selecting-software-architecture.html
59. S. Kaushal (n.d.). Lambda Expressions in Java 8. Geeks for Geeks. https://www.geeksforgeeks.org/lambda-expressions-java-8/
60. A. Kelly (2015). Does Agile Work outside Software? Agile Connection, January 7, 2015. https://www.agileconnection.com/article/does-agile-work-outside-software.
61. J. Kerievsky (2005). Refactoring to Patterns, Addison-Wesley, ISBN 0-321-21325-1.
62. M. Kuhrmann, P. Diebold, J. Münch, C. Prause, et al (2018). Hybrid Software Development Approaches in Practice: A European Perspective. IEEE Software, January 2018.
63. M. Kumar, A. Sharma, S. Garg (2009). A study of aspect oriented testing techniques. 2009 IEEE Symposium on Industrial Electronics & Applications, Kuala Lumpur, 2009, pp. 996-1001, doi: 10.1109/ISIEA.2009.5356308.
64. M. Kumar, M. Shukla, S. Agarwal (2013), A Hybrid Approach of Requirement Engineering in Agile Software Development. 2013 International Conference on Machine Intelligence and Research Advancement, December 2013.
65. M. Laanti (2016). Piloting Lean-Agile Hardware Development, XP '16 Workshops: Proceedings of the Scientific Workshop Proceedings of XP2016. May 2016, Article No.: 3, pp 1–6. https://doi.org/10.1145/2962695.2962698
66. K. Lalithraj (2020). Static vs Dynamic Code Analysis: How to Choose Between Them. OverOps, September 18, 2020. https://www.overops.com/blog/static-vs-dynamic-code-analysis-how-to-choose-between-them/
67. C. Larman (2005), Applying UML And Design Patterns, 3rd edition, Prentice Hall. 0-13-148906-2.
68. R. Lemos (n.d.). Why you need to get your team up to speed on privacy-aware development. TechBeacon. https://techbeacon.com/security/why-you-need-get-your-team-speed-privacy-aware-development
69. The LeSS Company (2020). LeSS Framework. https://less.works/less/framework/index.html
70. S. Lietz (2015). What is DevSecOps? DevSecOps Blog, June 1, 2015. https://www.devsecops.org/blog/2015/2/15/what-is-devsecops
71. C. Lilienthal (2019). Sustainable software architecture: Getting rid of technical debt. JaxEnter, 07 25 19. https://jaxenter.com/sustainable-software-architecture-technical-debt-160372.html
72. T.J. Marlowe, V. Kirova (2008). Addressing Change in Collaborative Software Development through Agility and Automated Traceability. 12th World Multiconference on Systemics, Cybernetics and Informatics (WMSCI 2008), 209–215, Orlando, USA, June-July 2008.
73. T.J. Marlowe, V. Kirova (2009). High-level Component Interfaces for Collaborative Development: A Proposal. 2nd International Multi-Conference on Engineering and Technological Innovation (IMETI 2009), July 2009.
74. T.J. Marlowe, N. Jastroch, S. Nousala, V. Kirova (2012). Collaboration, Knowledge and Interoperability — Implications for Software Engineering. Workshop on Collaborative Enterprise (CENT) 2012, 16th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI 2012), Orlando FL, July 2012.
75. T.J. Marlowe (2020). Keynote, "The Development and Evolution of Agile", IIIS Plenary Presentation (Asynchronous), September 2020.
76. T.J. Marlowe, V. Kirova, G. Chang, "The State of Agile: Changes in the World of Change". WMSCI 2020, Orlando FL, September 2020.
77. R.C. Martin (2009). Clean Code, Prentice Hall, 2009. ISBN 0-13-235088-2.
78. C. Matthies (2019). Agile process improvement in retrospectives. ICSE '19: Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings. May 2019, 150–152. https://doi.org/10.1109/ICSE-Companion.2019.00063
79. R. McBean (2019). The art and craft of test-driven development. Increment.com, issue 10, August 2019. https://increment.com/testing/the-art-and-craft-of-tdd/
80. F. McCaffery, P. Donnelly, A. Dorling, F.G. Wilkie (2004) A Software Process Development, Assessment and Improvement Framework for the Medical Device Industry. In: Fourth Int. SPICE Conf, 28-29 April 2004, Lisbon, Portugal. https://eprints.dkit.ie/144/
81. R.M. McClure (2001). The NATO Software Engineering Conferences. http://homepages.cs.ncl.ac.uk/brian.randell/NATO/Introduction.html
82. B. Meyer (2014). Agile!: The Good, the Hype and the Ugly, Springer. ISBN-13: 978-3319051543.
83. P. Milind, S. Gopinath, A. Kumar (n.d.). Using agile in construction projects: It's more than a methodology.
84. Modern Agile (2020). https://modernagile.org/
85. E. Moster (2013). Using Hybrid SCRUM to Meet Waterfall Process Deliverables, Ph.D, Thesis, East Carolina University, 2013.
86. A.S. Mutschler (2016). Using Agile Methods for Hardware. Semiconductor Engineering, January 28th, 2016. https://semiengineering.com/using-agile-methods-for-hardware/
87. New Context (n.d.). The "What" "How" and "Why" of DevSecOps: What is DevSecOps?. https://newcontext.com/what-is-devsecops/
88. OptiProERP (2020). What is Agile Manufacturing and how can it help you succeed? https://www.optiproerp.com/in/blog/what-is-agile-manufacturing-and-how-can-it-help-you-succeed/
89. B. O'Reilly. How to Implement Hypothesis-Driven Development. https://barryoreilly.com/how-to-implement-hypothesis-driven-development/
90. M. Poppendieck, M.A. Cussomano (2012). Lean Software Development. IEEE Computer, September-October 2012, 26-32.
91. R.S. Pressman, B. Maxim (2015), Software Engineering: A Practitioner's Approach, 8th edition. McGraw Hill, 2015, ISBN 0-07-301933-X.
92. V. Rahimian, R. Ramsin (2008), Designing an agile methodology for mobile software development: A hybrid

method engineering approach. 2008 Second International Conference on Research Challenges in Information Science, June 2008.
93. D.K. Rigby, S.Berez, G Caimi, A. Nobel: Agile innovation, Bain & Company Brief, April 19, 2016.
94. M. Rouse (2018). Integrated development environment (IDE). Tech Target.
95. Isaac Sacolick (2020). 7 best practices for remote agile teams. Infoworld, 04 13 20. https://www.infoworld.com/article/3532286/7-best-practices-for-remote-agile-teams.html
96. SAFe WhitePaper 5.0, Final Draft (2019). https://www.scaledagile.com/resources/safe-whitepaper/
97. P. Satasiya (2019). Top Ten Automated Testing Tools. DZone.com. https://dzone.com/articles/top-10-automated-software-testing-tools
98. Scaled Agile Framework (2019). DevOps. Last update December 27, 2019. https://www.scaledagileframework.com/devops/
99. Scaled Agile Framework (n.d.). Team kanban. https://www.scaledagileframework.com/team-kanban/
100. K. Schwaber (n.d.). SCRUM development process. http://www.jeffsutherland.org/oopsla/schwapub.pdf
101. Scrum Alliance (2017). The Scrum Guide™. November 2017. https://www.scrumalliance.org/learn-about-scrum/the-scrum-guide
102. Scrum Professional (2018). To Scrum, to Kanban or to Scrumban, that is the question. ProWareNess. https://www.scrum.nl/blog/to-scrum-to-kanban-or-to-scrumban-that-is-the-question/
103. SCRUMStudy (2020). What is Crystal? January 21, 2020. http://blog.scrumstudy.com/what-is-crystal/
104. H. Shah (2020). Agile vs DevOps: What's the difference? JavaCodeGeeks, 11 10 20, https://www.javacodegeeks.com/2020/11/agile-vs-devops-whats-the-difference.html
105. M. Shahin, M. Ali Babar, L. Zhu (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. IEEE Access, vol. 5, pp. 3909-3943, 2017, doi: 10.1109/ACCESS.2017.2685629[44]
106. M. Shaw, D. Garlan (1996). Software Architecture: Perspectives on Emerging Discipline, Prentice Hall. ISBN-13: 978-0131829572.
107. R. Shaydulin, J. Sybrandt (2017). To Agile or Not to Agile: A Comparison of Software Development Methodologies, April 2017.
108. S. Sigel (2017). Pros and cons for agile hardware product development. GrabCad, May 14, 2014. https://blog.grabcad.com/blog/2014/05/14/pros-cons-agile-hardware-product-development/ [and links therein]
109. Software Testing Help (2020). What is Beta Testing: A complete guide. https://www.softwaretestinghelp.com/beta-testing/
110. Stack Exchange (2017). What could be a valid definition of DevOps to introduce it to a novice? March 2017. https://devops.stackexchange.com/questions/788/what-could-be-a-valid-definition-of-devops-to-introduce-it-to-a-novice#806
111. G. Straçusser (2015). Agile project management concepts applied to construction and other non-IT fields. PMI® Global Congress 2015—North America, Orlando, FL. Newtown Square, PA: Project Management Institute.
112. M. Sweezey, S. Vizuri (2014). Internationalization best practices for agile teams. AgileConnection, January 7, 2014. https://www.agileconnection.com/article/internationalization-best-practices-agile-teams
113. M. Tanveer (2016). Agile for Large-Scale Projects—A Hybrid Approach. 2015 National Software Engineering Conference (NSEC), December 2015.
114. R. Tarne (2011). Taking off the agile training wheels, advance agile project management using Kanban. Paper presented at PMI® Global Congress 2011—North America, Dallas, TX. Newtown Square, PA: Project Management Institute.
115. Ö. Uludağ, M. Kleehaus, N. Dreymann, C. Kabelin, F. Matthes (2019). Investigating the adoption and application of large-scale scrum at a German automobile manufacturer. ICGSE '19: Proceedings of the 14th International Conference on Global Software Engineering. May 2019, 22–29. https://doi.org/10.1109/ICGSE.2019.00-11
116. L. van Moergestel, E. Puik, D. Telgen, J.-J. Meyer (2012). Production Scheduling in an Agile Agent-Based Production Grid, WI-IAT '12: The 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology - Vol. 02, 293-298, December 2012. https://doi.org/10.1109/WI-IAT.2012.139
117. J. Viega, J.G. McGraw (2001). Building Secure Software: How to Avoid Security Problems the Right Way. Addison-Wesley Professional. p. 528. ISBN 978-0201721522.
118. VivifyScrum (2019). Agile Project Management in Manufacturing – Coming the Full Circle. 04 19 2019 https://www.vivifyscrum.com/insights/agile-project-management-in-manufacturing
119. E. Warren (2019)., Agile might be dead but agility isn't. Forbes, 07 16 19. https://www.forbes.com/sites/forbestechcouncil/2019/07/16/agile-might-be-dead-but-agility-isnt
120. D. Wells (1999). Spike solution. http://www.extremeprogramming.org/rules/spike.html
121. LVK Withanagamage, RMVS Ratnayake, EJ Wattegama, A Conceptual Framework to assess the Applicability of Agile Manufacturing Techniques. 2018 International Conference on Production and Operations Management Society (POMS), 2018.
122. A. Yagüe, J. Garbajosa, J. Díaz, E. González (2016). An exploratory study in communication in Agile Global Software Development. Computer Standards & Interfaces, 48, 184-197, November 2016. https://doi.org/10.1016/j.csi.2016.06.002
123. F. Zambonelli (2016). Towards a General Software Engineering Methodology for the Internet of Things. arXiv, 21 January 2016. (See also references therein.)