The Need for Tamed Agility

1

Pragmatic, value-focused support for the design and implementation of complex IT projects appears more necessary than ever before, especially in times of ubiquitous digitalization, as "software is eating the world" (Andreessen 2011): In increasingly digital companies, the number of projects that is not heavily dependent on IT is constantly falling. The implementation of organization projects, projects for implementing regulatory requirements and merger and acquisition projects is also practically impossible without the involvement of IT—"every budget is becoming an IT budget" (Gartner 2012).

1.1 A New School of IT

IT has always involved automation, and IT has also always had a disruptive influence. Business models have always changed as a result of IT. Some disappeared, some only became possible in the first place. So is everything the same as it always was? Not entirely, because a number of factors are currently combining: The world is becoming more digital, data and applications are becoming mobile, and IT projects have to deliver quick results. Even during development, it must be possible to adapt their focus. Long project durations are undesirable, because the world has often changed so dramatically after a long project that it is difficult to know whether the originally promised benefits are actually generated. This leads to a change that is more radical than the slow progress of automation. Concepts that appeared promising yesterday are now a hindrance. It seems that enterprise IT has a new role and that it requires new or at least additional skills and capabilities.

Faced with technological disorder in the context of mobile technologies, broad digital transformation and elastic, cloud-based infrastructures, IT is no longer just a central means of production. Rather, enterprise IT is becoming an essential co-designer and co-creator of future solutions. In order to fulfill this role, it must

M. Book et al., *Tamed Agility*, DOI 10.1007/978-3-319-41478-2_1

assess the opportunities and risks of new technologies, talk to users and business departments, and know the challenges faced by the respective industry.

As a result, enterprise IT is changing from a pure service provider to an enabler and co-designer of business changes. Instead of just implementing an operating department's ideas, and instead of just providing defined services to an agreed quality, enterprise IT is taking on a consulting role. Based on its knowledge of technology costs and benefits, and of business challenges and opportunities, enterprise IT now works together with the operating departments to design solutions that can be implemented efficiently, that have innovation potentials, and that provide competitive advantages.

In other words: Enterprise IT is on the move. From the basement to the boardroom. It now has a say and takes responsibility. And it can only do this if it understands both technology and business.

Companies are currently facing huge strategic changes triggered by three key IT trends: mobility of clients and employees, agility in software development, and elasticity of IT infrastructure. These are the foundations that are increasingly defining the requirements of enterprise IT. And because an enterprise IT that satisfies these requirements has a different structure and different competencies than traditional enterprise IT, we call it the New School of IT. This is admittedly bold, but clearly states that the upcoming changes will go far beyond a normal level of change.

1.1.1 Mobility

Mobility is increasing across all industry sectors: Central business processes have mobile components, or at least components that can be mobilized. Clients and suppliers can be integrated using web-based applications or native apps and take over important parts of the business process. Mobile solutions need to be developed and delivered quickly. The aim is to rapidly launch new products or services on the market, often using a range of different sales channels.

Whether the mobility of data and applications demanded by users is always required, and whether it is socially and economically beneficial that the availability of humans is increasing, and that parts of the business process can be outsourced, is irrelevant for the question of whether enterprise IT must be able to develop and operate mobile applications. The trend toward mobility is a social trend, and the experiences gained in the private context are creating expectations in companies.

Consequently, enterprise IT must come to terms with the topic of mobility. This is exacerbated by the fact that mobility is often also an important driver for innovative applications, simply because the mobilization of data and applications can lead to structurally different applications and entirely new use cases, which makes the topic of mobility even more essential for enterprise IT. After all, the mastering of technologies that have the potential to trigger the next batch of changes in application landscapes cannot be outsourced and remains part of the enterprise IT's core business.

1.1.2 Agility

Innovative IT solutions can rarely be completely planned in advance and then "just" be implemented. Rather, they are based on the idea of permanent adaptation to new or clearer boundary conditions outlined by Ries (2011). And because the basic concepts of agile software development—fast and frequent delivery of software, concentration on source code as the central artifact of development, continuous communication with clients and users, and respect for application knowledge—benefit more than just mobile and other innovative applications, software is increasingly being developed with agile software process elements. Virtually, no software development of a relevant scale is either purely agile or entirely without agile elements (Boehm and Turner 2003). Common sense suggests that projects can vary significantly on the spectrum between strictly agile and strictly waterfall-oriented. Mary Poppendieck summarized this in her keynote speech at the 35th International Conference on Software Engineering (ICSE 2013) with the statement "agility without discipline cannot scale, and discipline without agility cannot compete."

Given that discipline can have different connotations, and given that a large number of people with a range of independent perceptions are involved in software projects of a relevant scale, certain standards are required to respond to different perceptions of the necessary discipline and restrict these perceptions to compatible ideas of discipline. A lack of compatible perceptions of discipline and their specific manifestation often results in misunderstandings. These can be countered by explicit rules and agreements, which then however represent the explicit discipline addressed by Poppendieck, i.e., an alternative to agile, personal discipline that is only based on a small number of principles. Overall, we are still faced with the problem that agile development approaches have to be supplemented by elements of requirements transparency in order to apply them to major projects in large organizations.

Probably the most popular approach to placing a square peg in a round hole and reconciling agility with the need for planning certainty is based on the Scaled Agile Framework (SAFe) approach introduced by Leffingwell (2011). However, significant doubts remain as to whether any of the original allure of agility remains in light of the extensive expansion, and also whether the implementation of SAFe in companies does not lead to completely erratic results, simply because SAFe is vague and non-specific.

1.1.3 Elasticity

Elasticity is the extension of agility from application development into application management, from the world of application software to the world of system software, infrastructure and hardware.

Infrastructures need to be elastic so that mobile applications, applications that are frequently extended with new functionality, and applications for end users can scale seamlessly—i.e., that they can deal with widely fluctuating (and also sharply

increasing) user numbers without changing their behavior so drastically that the user is disturbed. Elasticity means that infrastructures can be scaled up as well as down.

Elastic infrastructures are also necessary to ensure that the benefits of agile software development do not dissipate: If agile development delivers new software every few weeks (or even days!), it must be released into productive use (or at least tested for its suitability to be released) just as often. If this does not happen and a new release is deployed only every few months, the development team's willingness to deliver features at short notice will run out quite fast. Continuous integration of software (Cusumano 1992) and the continuous release of new features—even in heterogeneous infrastructures (Humble and Farley 2010; Duvall et al. 2007)—are therefore required to ensure that agility will not remain restricted to the development side only.

There are many ways to ensure elasticity. Cloud solutions of many types and suppliers promise scalability. Security concerns about remotely hosted, externally managed data are numerous and often quite justified. Private clouds try to reconcile both—unlimited sovereignty over the data, and scalability as in a public cloud.

However, as is usual when trying to reconcile contrary positions, compromises cannot be avoided: A private cloud is not as scalable as a public cloud—but possibly sufficient for the application in question. And the complete sovereignty over all data comes at the price of a very high vertical IT integration—but maybe not all data's security is equally critical. The design of suitable private clouds (or comparable structures) therefore requires a sense of proportion, the critical consideration of killer arguments, the rational evaluation of risks and requirements and—in the solution domain—the automation of IT infrastructure provisioning mechanisms. Automation is particularly important here because it is the only way to avoid the susceptibility to errors and dependency on individual people that traditionally plagues provisioning processes.

1.1.4 Resulting Challenges

Mobility, agility, and elasticity influence each other; they entail, overlap, and reinforce each other: Mobile applications are subject to shorter release cycles and therefore require more agile process elements. Agile development depends on an elastic and easily provided infrastructure to ensure that the benefits of frequent releases reach users immediately. This interplay fundamentally changes the way IT works, and how it is understood.

However, this change is not just technical in nature. The New School of IT also means that the significance of IT in companies is changing. Seeing correlations, establishing new business models, reaching new target groups—the foundations for this are laid ever more often in IT departments. Enterprises are increasingly "digitizing" themselves, and in the process, enterprise IT increasingly emancipates itself from its role as the operating departments' assistant. Enterprise IT is driving the new developments instead of being driven by them.

The New School of IT also means that enterprise IT cannot focus exclusively on classical software systems anymore. Moore (2011) calls these systems the "systems of records." Systems of records are characterized by high transaction volume, clear persistence design, and a high degree of consistence. Besides these, we increasingly find "systems of engagement" that spill from the consumer world into the enterprise world. These are systems that consist more of mash-up architectures than traditional enterprise application landscapes, that are configured by users, that are easily adapted and frequently released, that focus on the user experience, and that are subject to a high degree of uncertainty regarding the next features that will be requested by users.

Development and operation of systems of engagement require other skills and approaches than systems of records. Therefore, start-ups follow other (more agile) software processes than large digital enterprises with stable business models. Things get difficult though when systems of records merge with systems of engagement, when flexibility and stability need to be reconciled, when stable, consistent, and scalable systems must be equipped with mobile interfaces. Neither an agile nor a classical development paradigm is quite suitable for this—rather, a mix is required: an enterprise IT from the New School of IT. This is an enterprise IT that has mastered both paces, that is founded on stable base processes, that can work with established technologies just as with new ones, and that can implement safe, robust operations processes just as well as short release cycles and continuous integration—and that has the expertise to decide which development paradigm is best suited to which problem.

The New School of IT requires companies to rethink not just their enterprise IT, but also their operating departments, business development, and management. The most extensive changes, as described in the previous chapter, are of a strategic nature. Dealing with them and taking advantage of the resulting opportunities is the top management's responsibility.

The New School of IT also exposes every IT project manager to uncomfortable challenges: How are IT projects affected when the operating department is not just sending down specifications from three levels up, but discussing with the engineers at eye level? What does it mean if system boundaries become blurred, if clients and suppliers become partners, if software development and business development go hand in hand? Where are these requirements reflected in the software development methodology?

1.2 Agile or Plan-Driven?

Traditional plan-driven approaches seem too rigid for these challenges. The attempt to provide excessively detailed, precise, and long-term preliminary planning seems less promising where the boundaries between strategy development and software development become blurred, where software development has to respond quickly to changing competitive situations and user expectations, where new technologies

turn established service and operating models on their head. Rather, continuous alignment with user and management expectations, a lean product without superfluous features, and the acceptance of continuous change is desired. This is typically the incentive to pursue an agile development approach.

Agile approaches describe a world in which higher priority is placed on producing working software than any other artifacts, in which communication between stakeholders is regarded as more important than the use of tools and modeling languages, in which the spoken word is assigned a higher value than written text, and in which an joint understanding of discipline and common sense ensure that all stakeholders cooperate effectively with each other (Beck et al. 2001). This departure from the illusion of strict planning certainty may appear threatening to number-driven managers (maybe also to seasoned IT managers), as it seems to involve an almost complete loss of control, perhaps even careless blind confidence in the team's overall ability to work things out. Is this desirable?

The agile literature promises huge increases in productivity, but only for those who unquestioningly subscribe to the agile "faith," it seems. Virtually no evidence-based studies are available. If an agile project works out, it is due to the agile method, but if it does not work out, it is due to insufficient faith, the narrow-mindedness of management, the rigidity of stakeholders, and other factors that cannot be measured (Meyer 2014). There is a lack of clear, scientifically founded studies on the usefulness of agile methods, especially studies that provide evidence of the wonderful descriptions of perceived increases in productivity such as the 90 % improvement touted by Schwaber and Sutherland (2012), to name just one example. By contrast, experience from major projects tends to show that while agile practices are useful, they also require a certain amount of planning certainty and functional restriction of the features to be developed (Ambler 2001; Cohn 2010).

Agile approaches do not guarantee success. The IT landscape in which the projects of the New School of IT operate is too complex. Excessive freedom is just as pointless for these kinds of projects as the attempt to define every detail in advance.

In particular, the rejection of advance detail planning, requirements elicitation and design work that is propagated by agile methods quickly reaches its limits in major projects in established IT landscapes: The integration requirements that are posed by a heterogeneous system landscape, and the attention to detail that is required for the correct implementation of established business processes, cannot be captured in a stream of high-level user stories. In particular, it is virtually impossible to arrive at correct solutions in an efficient manner, using only incremental cycles of client feedback. Rather, developers and domain experts require a joint overall understanding of the business processes and IT components in order to make appropriate architecture, design and technology decisions.

From a management perspective, agile practices, such as self-organizing teams and a lack of commitment to time and budget requirements, are problematic, especially in IT projects that are developed in a client–contractor relationship and

not in-house: Employees in a start-up generally have sufficient intrinsic motivation to focus on a specific goal; and in internal projects, which are not overly critical to business, a detour here or there is forgivable (and may even promote innovation or at least instruction) as long as it does not exceed the budget framework. However, in complex projects, and especially in contractor relationships, a concrete idea of the target, direction, and expected effort of the project is essential in order to limit the economic risk for all stakeholders and ensure the smooth functioning of ongoing business operations.

A purely agile doctrine therefore does not quite seem to fit into the world of large companies: Giving up on detailed specifications altogether because it seems impossible to determine precisely which features can be delivered at which price is not acceptable for most clients. Careful advance consideration is always helpful, even if the results are known to be preliminary. The agile belief that talking is fundamentally better than writing may also be met with resistance in large companies, especially when dealing with complex software systems that are created by many stakeholders and supposed to be used for a long period of time. In such circumstances, the durability of the written word has its advantages. After all, despite a basic acceptance of the benefits of agile approaches, most clients still want to know roughly how expensive their software will be, which features can be delivered at what cost, and how long the development will take. As charming and unique as agile approaches may be in theory, in commercial practice they are quickly faced with reasonable expectations of planning certainty, coordination, and reliability.

Many of the aforementioned problems are due to an excessively dogmatic application of the agile principles, which does not take the reality of complex IT projects into account. However, this dogmatic approach can be relaxed without having to reject the key advantages of agility—responsiveness to changes and leanness of processes and products. Ultimately, in practice, strict adherence to the waterfall model is just as rare as the blind application of agile practices. Many approaches from the agile world can be logically applied in almost all projects, even in large and dispersed teams, and also in a manner that respects well-defined processes and synchronization points (Leffingwell 2011).

Upon closer inspection, many of the seemingly "radical" ideas in the agile literature are dampened by disclaimers not to overdo it, to communicate extensively and to apply common sense, but without specifying what a healthy balance of agility and planning might look like. There is certainly no panacea in this respect, as agile approaches differ depending on the project, stakeholders, and boundary conditions. Appealing for common sense is an obvious measure, but is unsatisfactory from a methodological perspective. It is certainly required, but is not an adequate condition for successful projects.

Boehm and Turner (2003) discuss dimensions of software development projects that may provide guidance for the decision of agile versus plan-driven methods for specific projects. These include purely local factors, such as project scale and criticality, as well as factors that relate to the corporate environment. Specifically:

- **Scale**: In agile projects, the focus is on the spoken word. Documents and models beyond the source code are regarded as deviations from the strict agile doctrine. But the spoken word has limited reach—only among small teams will the spoken word be sufficient to create joint understanding. Large projects with many stakeholders generally require written specifications in order to ensure that all stakeholders know what is required when. This is more plan-driven than agile. As a result, a general guideline is that small projects are more likely to consistently apply agile practices.
- **Criticality**: Does the system deal with money, human life, or even many human lives? If this is the case, a higher level of planning certainty, verification of software features, and proven comprehensive testing is advisable. Proponents of strict agility might argue that nothing can better lead to higher software quality than agile techniques. Let us assume that this is correct for a moment. Let us even assume that this applies not just to small, but also to large teams. Even then, the highest probability of correct software is not sufficient when dealing with safety-critical software. Sometimes, the correctness of the software has to be demonstrated. To do so, it must be specified. Yet, there is no place for this in pure agility doctrine. As a result, the following general guideline applies: The more critical a project, the more plan-driven elements and the more "big up-front" activities (Meyer 2014, Chap. 3) are required.
- **Dynamism**: The more dynamic the project context and the application environment of the software to be created, the greater the benefits provided by agile techniques. The strengths of agile techniques are particularly pronounced when a high level of dynamism is required. Dynamism may have completely different triggers: It may be caused exogenously, because a company's market, in which the software is to be used, is moving and it must be assumed that this movement will have an impact on the software (during its development or subsequent use). It may be organizational, because the company is currently being reorganized. Reasons for dynamism may also lie in the project, because certain requirements are fiercely contested, conflicts are foreseeable, or simply because an inadequate amount of domain knowledge exists. The latter form of dynamism does not necessarily have to affect the entire software equally. Perhaps some parts are well understood and easy to coordinate and others are not. As a result, a general guideline is that the more dynamic the context, the more a project tends toward agile techniques.
- **Personnel qualification**: While it would be desirable, not every team is fit for agile development. Agile development requires the involvement of clients, users, and the application domain. If the team does not have the relevant skills or know-how, the transfer of knowledge between users and developers generally has to be managed in a non-agile manner (i.e., via extensive specification documents), and often fails. A lack of domain knowledge by developers puts the project in jeopardy from the very beginning. If one still wants (or has) to take that risk, neither a purely agile or purely plan-driven approach is likely to work, and a situational mix of both approaches is required. The following general guideline applies: The greater the language difficulties between the development

team and users, the greater the dependence on an appropriate mix of agile and plan-driven instruments in order to compensate for this deficit.

- **Culture:** Companies with the same business purpose, same size, similar products, and the same market may differ culturally despite their commonalities. Cultural differences are often manifested in how errors and requirements for change are handled. On the one hand, some companies require the minutes of meetings to be signed by all stakeholders and, in some cases, the length of the change histories exceed the useful part of documents. On the other hand, some companies focus on recording just the key results. They accept the fact that some decisions cannot be transparent for all stakeholders, that back-and-forth discussion is required, and that decisions can simply be interpreted differently. Depending on an individual's perspective, these contradictions can either be referred to as "control-focused versus pragmatic" or as "careful versus casual." Both are just as partisan as the contradiction between "plan-driven versus agile." In fact, a company's culture often either propagates the use of agile techniques ("agility is genuinely necessary") or their limitation ("that level of agility is really not acceptable here"). A general guideline is that control-focused/careful company cultures generally tend toward plan-driven approaches and could benefit from agile injections, while the opposite is true for pragmatic/casual corporate cultures.

1.3 A Pragmatic Middle Ground

As we can see, the challenges of the New School of IT call for an approach that occupies a pragmatic middle ground between traditional and agile software development processes, i.e., an approach that does not attempt to guarantee planning certainty, trust, and value orientation based on comprehensive specifications, but that also does not expect these qualities to emerge automatically through the free interaction of forces.

Rather, large, digital companies require an approach of *tamed agility* in order to combine the necessary flexibility with essential rough planning (budget planning, portfolio planning, and IT controlling): Tamed agility is a middle ground for IT projects that can benefit from the flexibility of agile approaches, but must satisfy expectations with regard to business complexity, environment conditions, contractual requirements, etc., which make stricter preliminary planning essential.

Tamed agility combines techniques from agile approaches with planning and management methods. However, its primary aim is to ensure that all stakeholders develop a common understanding of what the essential requirements are at the start of a project, namely the requirements whose appropriate implementation determines the acceptance of the software (McMenamin and Palmer 1984). But how can these essential requirements be determined? How can they be separated from the many other, possibly also relevant, but non-essential requirements? And how can a vision

of the future system be formed based on the knowledge of the essential requirements? This is impossible without abstraction, without temporary omission of irrelevant details, and a focus on the essentials—and it is especially impossible without a readiness for compromise and respect for application knowledge.

Before we look at how this can be achieved in software development, let us first take a step back and consider a situation that has nothing to do with software: Imagine a CEO who would like to understand what his new company building will look like and how it will function. He does not want to know exactly how the heating system will work, how thick the thermal insulation is, or how much air is exchanged by the ventilation system every hour. But he would like to know what the building looks like, where his office is, and what the view from his office is like. Probably, he is not aware of any of this and simply asks the project manager about the status of the building planning. She dutifully sends him 15 PDF files that provide information about everything: the view, the office layout, the building services, the access concept, and much more. The manager now realizes that he did not want this level of detail. After some back-and-forth discussion, it may turn out that a wood model stands in the project office and that the most important building plans have been attached to the office walls. Much better than 15 PDF files—not for every purpose, but certainly for the purpose of giving an idea and an anchoring point from which a range of further questions can be asked and answered.

This example shows that different communication situations require different models. A manager requires an overview model. This does not need to be formally precise, nor does it need to be overly detailed. Rather, it must support intuitive understanding. The authority processing the building application requires a model of the building to be constructed with precise dimensions and specifications. An approximate model is not sufficient in order to evaluate things like the maximum eaves height and compliance with clearance requirements. The building authority is less interested in other details though, such as the technical design of the installations, but those are relevant for the heating engineer. And even other models are obviously required for the interior design.

Software development requires models that are at least as diverse. This may be because the final artifact, the delivered software, is itself only a model of a section of the world. Models from which software is to be generated require a different level of detail and precision than models that only need to clarify the purpose, the core aim of the project, and the look and feel of the software to be created. Such models are especially required in the early phases of software construction. And this takes us back to the CEO who wants to understand his building: Just as 15 PDF files cannot help him, a manager who just wants to get an idea of a software project's core aim and state will not learn much from a 500-page specification.

As a result, we can conclude that vague, incomplete, perhaps even inconsistent models can be useful in the early phases of software development. In some cases, they may even be just the right communication tool. Completeness is not the aim in these early project phases. Instead, the aim is to find out what does and what does not belong into the software to be created. The boundary between the actual system and its context must be defined. And, in particular, the most important requirements

must be identified, independently of their solutions. This is not just because these essential requirements must not be overlooked, but primarily because they clarify the key requirements for stakeholders with no knowledge of the application domain. Abstract models, which can be understood by all stakeholders, are particularly helpful for the initial requirements scoping of a software project.

Such an approach is most successful if the models are jointly prepared. If a model is *really* prepared *jointly* (rather than just one person preparing everything independently, and the others just approving the result), verbal communication and the joint struggle to find the best solution are unbeatable in terms of efficiency. Rough resource estimates are made based on the jointly prepared (and thus jointly understood) models (keeping in mind that this kind of estimate can only be rough and provisional).

Development then takes place using the necessary amount of agility, as late requirements are inevitable and priorities may change during development. Late requirements are exchanged for early requirements to ensure that the software being created does not become increasingly bloated. This not only means that new requirements are added, but that a continuous cleanup also takes place. Perhaps a bit less software may be enough after all, and the resource estimate is adjusted with every step toward a more solid structure and design of the software. In the design itself, the commercial risks are fairly distributed between the client and the contractor so that all sides are motivated to create the leanest possible software. This kind of tamed agility then no longer seems threatening, not even to the IT manager.

1.4 Tamed Agility in Practice

Tamed agility is not just a buzzword for another agile philosophy. It is manifested in specific instruments and procedures for the scoping, designing, development, and billing of complex agile projects, which are described in the following chapters:

The **Interaction Room** (Part II) helps teams obtain an overall picture of the business and technology, effort and risks, and the environment and dependencies without getting lost in extensive specification documents. The Interaction Room is not just a name, it is also a real, physical room. It is the central information and communication point in the project, where the focus is on the interaction between all stakeholders. Stakeholders outline models of the business processes to be supported and the data to be managed as well as the relevant application landscape on the walls of the Interaction Room. This occurs using free syntax, without specific notations, in a way all stakeholders understand. Particularly critical elements, i.e., special value, effort, or risk drivers, are highlighted with annotation symbols. These annotations allow the stakeholders to point out what is important to them and why. This occurs through personal interaction with one another, not through long-winded specifications or asynchronous communication. The live interaction results in a more direct development of a joint understanding of the scope of the project and the expected complexity of individual features, without the need for extensive documentation.

During the course of the project, the **adVANTAGE contract model** (Part III) then ensures that the agility that all stakeholders desire does not get trapped in rigid contracts, acceptance, and billing modalities, but that it is actually applied in practice as part of a fair cooperation between the contractor and the client. Sprints are planned in the Interaction Room, new and old requirements are weighed against each other, effort estimates are refined with a view toward the "big picture," and actual progress is compared to plans. The adVANTAGE model controls the sprint-based project billing and, in contrast to fixed-price or time and materials projects, ensures that the price risks are fairly distributed between the client and the contractor. All stakeholders are united in the goal of developing lean software, because additional effort is split between both sides.

References

Ambler SW (2001) Agile modeling and the Rational unified process (RUP). http://www.agilemodeling.com/essays/agileModelingRUP.htm. Accessed 23 Feb 2016

Andreessen M (2011) Why software is eating the world. Wall Street Journal, 20 Aug 2011. http://www.wsj.com/articles/SB10001424053111903480904576512250915629460. Accessed 23 Feb 2016

Beck K et al (2001) Manifesto for agile software development. http://www.agilemanifesto.org. Accessed 23 Feb 2016

Boehm B, Turner R (2003) Balancing agility and discipline: A guide for the perplexed. Addison-Wesley

Cohn M (2010) Succeeding with agile. Addison-Wesley, pp 166-171

Cusumano MA (1992) Shifting economies: From craft production to flexible systems and software factories. Research Policy 21(5):453–480. doi:10.1016/0048-7333(92)90005-O

Duvall PM, Matyas S, Glover A (2007) Continuous integration: Improving software quality and reducing risk. Addison-Wesley

Gartner, Inc. (2012) Gartner says every budget is becoming an IT budget. http://www.gartner.com/newsroom/id/2208015. Accessed 23 Feb 2016

Humble J, Farley D (2010) Continuous delivery: Reliable software releases through build, test, and deployment automation. Addison-Wesley

Leffingwell D (2011) Agile software requirements: Lean requirements practices for teams, programs, and the enterprise. Addison-Wesley

McMenamin SM, Palmer JF (1984) Essential systems analysis. Yourdon

Meyer B (2014) Agile! The good, the hype and the ugly. Springer

Moore G (2011) Systems of engagement and the future of enterprise IT: A sea change in enterprise IT. http://www.aiim.org/futurehistory. Accessed 23 Feb 2016

Ries E (2011) The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses. Crown Business

Schwaber K, Sutherland J (2012) Software in 30 days: How agile managers beat the odds, delight their customers, and leave competitors in the dust. Wiley, p 6

http://www.springer.com/978-3-319-41476-8

Tamed Agility
Pragmatic Contracting and Collaboration in Agile Software Projects
Book, M.; Gruhn, V.; Striemer, R.
2016, XVI, 334 p. 66 illus., 20 illus. in color., Hardcover
ISBN: 978-3-319-41476-8