Chapter 12
Improved Agile: A Customized Scrum Process for Project Management in Defense and Security

Luigi Benedicenti, Paolo Ciancarini, Franco Cotugno, Angelo Messina, Alberto Sillitti, and Giancarlo Succi

12.1 Introduction and Motivation

Developing software has become an increasingly more complex endeavor, especially in well-established domains with rich semantics and multiple, often conflicting needs coming from multiple customers. In the armed forces domain, these factors have traditionally been addressed via rigid development processes. Such processes seemed to conform well to the highly hierarchical structure believed to be integral part of a military organization. But today this is no longer the case.

Asymmetric opponents, multiple conflicting needs, and the imperative for a high-quality, flexible, and agile response have changed military doctrine profoundly. Although the effect of multiple, rapidly changing requirements coming from different actors had been identified for a relatively long time – Clausewitz was arguably the first to systematize the study of this effect and called it "friction" [1] – only recently has it come to bear on the creation of complex support tools like command and control systems or logistic management systems.

L. Benedicenti
University of Regina, Regina, Canada
e-mail: luigi.benedicenti@uregina.ca

P. Ciancarini
University of Bologna, Bologna, Italy
e-mail: paolo.ciancarini@unibo.it

F. Cotugno
Italian Army General Staff, Rome, Italy
e-mail: franco.cotugno@esercito.difesa.it

A. Messina • A. Sillitti (✉) • G. Succi
Innopolis University, Innopolis, Russian Federation
e-mail: a.messina@innopolis.ru; a.sillitti@innopolis.ru; g.succi@innopolis.ru

Z. Mahmood (ed.), *Software Project Management for Distributed Computing*,
Computer Communications and Networks, DOI 10.1007/978-3-319-54325-3_12

In addition to these considerations, new restraints in the defense budgets have effected a deep cultural change in the armed forces. For example, the US Department of Defense has changed its procurement policies to allow for common off-the-shelf equipment to be sourced instead of specialized equipment. Software production, however, is changing less rapidly, due in part to the absence of reliable methods to deal with complexity and rapid change in an already well-structured environment.

On the other hand, software cost per executable line of code (ELOC) or function point has decreased significantly in the commercial world in the last decade due to effectiveness of the latest generation of programming languages and the availability of open-source computer-aided software engineering (CASE) tools. Moreover, the adoption of agile development methods appears to have contributed to increased software development effectiveness and efficiency in those contexts.

At the General Staff of the Italian Army, these factors have led to the internalization of its software development, which requires contractors to work with a mix of civilian and military development teams. In turn, this has led to the creation of a new software development process, based on agile principles, and more specifically on scrum, called iAgile (for improved Agile).

Project management is an integral part of this process, because even in an agile context, it is still necessary to respond to the needs of the armed forces, chief among which are security and customer satisfaction. Drawing on the principles of agile development and intersecting them with the Italian Army procedures resulted in the incorporation of the following aspects in the management strategy:

- Frequent releases.
- Change management.
- Effective cooperation in a complex environment between requirement owners (goal owners), acquisition owners (gold owners), and contractors, who are numerous in the defense sector and do not traditionally interact easily with one another.
- Focus on small and empowered teams, to be able to react fast to evolving customer needs and emergency situations

The infrastructure of the iAgile process consists of four main “pillars”: agile training, innovative CASE tools, structured user community governance, and custom agile development doctrine. Refer to Fig. 12.1. All these pillars are part of the management strategy that enables iAgile to be effective.

Agile training is fundamental for change management, as it builds the knowledge needed to operate within the iAgile framework. But it is not enough, because change requires both motivation and reinforcement to be actualized. Motivation is provided through standard organizational means at the beginning, but after the first sprint, it is fueled by the visible progress demonstrated by the functionality of the partial product. Reinforcement is provided via the structured user community governance that enables feedback to be immediate and in bringing users close to the development greatly increases user satisfaction and consequently the team’s level of confidence.

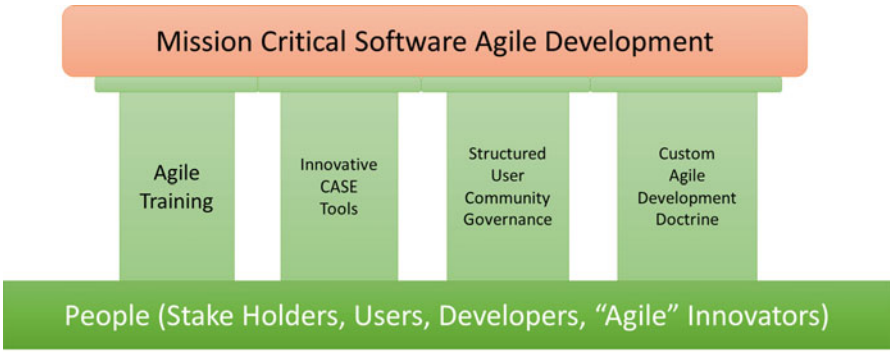

Fig. 12.1 Infrastructure Supporting Agile Software Development

The innovative CASE tools pillar is needed because without supporting tools tailored for iAgile, the software development would proceed at a slower pace due to lack of automation and a progress monitoring system. In particular, it was essential to develop a suite of noninvasive monitoring tools that measure the software artifacts being created without requiring developer intervention, which would slow down developers. The monitoring tools could then generate progress reports that concretely demonstrate the effectiveness of the development, acting as a reinforcement mechanism and, when needed, providing data for causal analysis and problem resolution.

The structured user community governance is needed to empower selected experts to act as delegated product owners with the same decision power as their department heads. Achieving this step required the full support of the top-level authorities in the Army. This generated the required level of autonomy in development teams so that development could proceed without the constant bottleneck of top-down authorizations. It was possible to realize this structure because the unit of development in our case is a single user story, which can be altered very rapidly and thus is perceived to have little impact on the project; yet, the freedom to develop a large number of user stories concurrently results in a marked productivity increase.

Last but not least, a doctrine had to be developed to ensure that any replica of the process throughout the military environment be coherent with the iAgile process methodology.

Naturally, it is important to build a management team suitable for such process. As the development effort grew, we found it necessary to create more teams and link the teams together using a variant of the "Scrum of Scrums" method. The development began with a single seven-people team and ended up with seven teams similarly construed. This led us to the lengthening of our cycles to 5 weeks from an initial duration of 3 weeks, to accommodate the interaction among the groups. It also called for the redefinition of the roles of product owner and scrum master. The product owner role had the added responsibility to keep track of the stakeholders. This "wayfinding" role was not explicit in our process initially, but given the number of stakeholders involved in the project, it was a necessity. The scrum

master role was redefined in terms of hierarchical authority. This was needed to make it possible for the scrum master to have the authority to resolve the issues brought forth by team members. Additionally, the scrum master also actively elicited feedback from all team members, rather than waiting for feedback passively. As scrum masters gained authority, it also became necessary to assign them the responsibility to keep the feedback loop running. To help enact the Scrum of Scrums, we then created the role of global product owner. This role became responsible for the synchronization of each individual scrum. Finally, as the skill sets grew, it became necessary to create a new role that reflects the awareness of the skill sets and their location in the teams to plan for skills allocation in the next sprint. We called it the scrum coach.

These management structures and role changes deviate from the scrum method and differentiate iAgile from standard scrum, justifying also its continued success since its very first project.

To test this project management strategy, we developed an entire command and control system for the Italian Army. This development was the first application of iAgile, and as a result it is fair to say that iAgile evolved concurrently with the development of our first case study. In this chapter, we will show how this development project evolved and the lessons we learned by running this project. The results will show that iAgile has been able to achieve high levels of productivity, quality, and customer satisfaction, but this has come at the cost of a profound cultural change in the development teams.

This chapter is organized as follows: The next section contains the state of the art on agile management. The next three sections provide a description of the method we developed together with the tools and the anticipated benefits of adopting our method. Section 12.6 presents a case study for the application of our method to build a command and control system. Section 12.7 contains the conclusions and future work.

12.2 State of the Art

12.2.1 Agile Project Management

Conventional project management is a storied discipline. Its roots can be traced back to the early 1900s through the contribution of pioneers such as Henry Gantt and Henri Fayol. Modern project management began in the late 1960s, and today there are numerous certifications for project managers, perhaps the most important of which are supported by the International Project Management Association and the Project Management Institute. In the late 1980s, the manufacturing industry began to change project management practices to include ways to increase efficiency and reduce semiworked product stock through the practice of just-in-time

manufacturing which, in turn, led to innovative project management methods like kanban [2] and lean [3].

The principles and practices of project management, usually, rely on the relative predictability of the scope of the project and on a set of preset requirements. Often, requirements are somehow formalized, and if there is a business agreement among partners, requirements are usually included in it. However, this method creates gaps. These gaps are of three kinds: The first kind is the gap between desired outcomes and actual outcomes. This originates from our inability to act precisely on a rapidly changing external environment, and it is compounded from the lack of perfect information on this environment. The second kind of gap is the gap between plans and actions in organizations. This originates from the inherent difficulty of defining a course of action for every single possible situation that might occur, which generates inability to achieve the planned outcome. The third kind of gap is the gap between actions and outcomes. This originates from the inability to foresee how actions will generate outcomes given a changing environment [1].

To close these gaps, a new type of project management is needed: one that is more flexible and is able to be proactive toward rapid requirement changes. This kind of project management is inspired by the Agile Manifesto, which was originally conceived for software production, but is now more widespread in its applicability [4]. Thus, the first books on agile project management (e.g., [5]) try to summarize some of the concepts in agile management, but because there is not much experience in the field, the suggestions and analyses emphasize the differences between conventional and agile project management, which carries the danger of overlooking important lessons from the traditional project management discipline. For example, the role of upper management in agile projects is to manage "the linkages between strategy and tactical project execution," while the traditional role is to approve projects and commit resources [5]. In reality, a more integrated approach in which both roles are adopted is better suited to a rapidly changing environment.

Most of the progress in agile project management was the result of experiential learning. For example, Jeff Sutherland et al. [6] documented the use of distributed scrum: a version of agile project management and a tool to increase productivity and team effectiveness.

In 2008, grounded theory proposed to investigate agile project management, which signals the continued interest in characterizing agile project management from actual practice [7]. In particular, areas of investigations involve the role of the project manager in an agile project, the process and problems of transitioning into an agile framework, and management of offshored or outsourced agile projects.

More recently, a systematic literature review on agile project management has been developed in 2014 by Chagas et al. [8], which contains information on 34 primary studies to create an ontology of terms and principles in the context of maturity models, to facilitate the transition between the two methods and consequent mentalities.

A better definition of agile project management has also been developed in [9]. According to this definition, agile project management deals with the

complexity and uncertainty of a software project, which creates unplanned change. To do that, agile project managers rely on some principles directly related to the ones expressed in the Agile Manifesto. Some of the practices in agile development methods, for example, team autonomy and continuous learning, are naturally extended to agile project management as well.

These principles have been expanded to more elaborate methods such as the Crowder and Friess agile project management [10]. Still, these ideas need additional validation and tailoring to specific situations. This is why case studies are of fundamental importance to understand how the principles formulated in a document can be instilled into an actual project and what lessons can be learned from the project's outcomes.

Currently, agile project management is entering the mainstream. For example, de Kort just published a book [11] on development and operations (DevOps) on the Microsoft stack that contains guidelines on how to use Microsoft tools in an agile context. This is but one of the many publications devoted to improving the experience of mainstream developers wishing to adopt a fully agile approach.

Yet, there are still some issues emerging from agile project management practices. One of them appears to be particularly important: communications. This issue occurs in all kinds of project management, not only the agile kind, but given that some of the standard tools for communication like project documentation are somewhat reduced in an agile approach, communication becomes especially relevant in an agile project. At the time of writing, there is no general solution for balancing communication and productivity, and this issue remains the subject of active research.

12.2.2 Project Management in Defense Domains

The most used project management methodology in the military environment is Projects in Controlled Environments, version 2 (PRINCE2® [13]), and its derivatives, e.g., North Atlantic Treaty Organization (NATO) [12]. This type of management activity requires heavily trained professionals and a massive quantity of formal documentation both in the planning and execution phases. For rapidly changing scenarios and volatile requirements, these articulated and time-consuming methods may be difficult to apply. PRINCE2® is a project management process-based approach method. Eight basic processes intended to be applied by a project manager in managing a project are modeled. The modeling is made in terms of steps in a logical sequence. PRINCE2® is adaptable to various types of projects in a different range of complexity. In the manual (Managing Successful Projects with PRINCE2®, 2009), a number of "components" are listed as guidance for a project manager in applying the process model. PRINCE2®, differently from other methodologies based on the availability of information about proven practices, provides a more prescriptive set of steps to be followed by project managers and teams. A possible advantage of PRINCE2® comes from the fact that it is

prescriptive in a large measure, and it may induce a degree of standardization in medium to large organizations. Tailoring of the methodology to specific projects is possible provided that the same basic steps are kept and the same terminology is used. Relevant benefits in corporate program management can be observed especially in the areas of staff training and project performance tracking. The possible constraints to creativity are a disadvantage especially in the area of software development where the human factors are becoming more and more relevant to the design solutions.

12.3 The iAgile Development Process

iAgile is a software development process based on distributed scrum [6]. Just like scrum, iAgile is an iterative development process in which a product owner, a scrum master, a manager, and a development team collaborate to deliver concrete functionality in a sequence of short development cycles (the sprints). However, the unique challenge of working within an armed forces context and having to include civilian consultants no longer as isolated subcontractors but as integral part of the development team led us to some changes that differentiate iAgile from scrum substantially. Furthermore, iAgile's domain is mission-critical software for the defense environment, which adds specific nonfunctional requirements like security and graceful system degradation. These requirements too contribute to further differentiate iAgile from standard scrum processes adopted in the enterprise.

In fact, iAgile has a number of formalized procedures that are assigned to personnel in various iAgile roles. Many of these roles are derived from the scrum method, but often there is a significant reshaping of the responsibilities associated with the roles. The basic principles of iAgile can be summarized as follows:

1. Development teams are the most relevant assets in the production process, but their activity has to be accompanied by a set of objective, real-time, noninvasive measures of effectiveness and performance agreed upon by team members and fully understandable by the customer. The set of measures must cover the most relevant aspects of the product as indicated by the user/stakeholder priorities (safety, security, quality, etc.). The first users of the measures are the team members themselves. The measures are collected and displayed in a software development virtual control room. Reports are automatically generated and sent to all the relevant community members.
2. The traditional role of product owner (PO) is shared by the global product owner board instead of being covered by a single individual. The GPO always includes a customer stakeholder representative, the most relevant application domain experts, and the teams POs. The team POs are responsible for team synchronization and feedback, which is an important feature to ensure the teams are always aligned in their production objectives and schedules.

3. The scrum master (SM) role in iAgile has been reinforced and has acquired a program management task. The SM has no decision power over the team member (in fact, scrum masters are themselves team members), but the reports the SM produces are the only means to assess the development status and the team performance. Every report is shared with the team members before reaching the GPOs and the stakeholders.
4. The skills of the technical team members are accurately scrutinized during the preliminary team building activities. The capability of working in parallel with subject matter experts not necessarily having a software engineering background is a key factor for selection of the team members. The developers are supposed to apply extended "pair programming" with asymmetric roles where the second programmer in the pair can be a security or quality expert. The technical growth of the team members is implemented throughout the whole production process and obtained by the insertion of "knowledge acquisition user stories" in the product backlog. Asymmetric pair programming in particular is not commonly used in scrum.
5. A new role was added to the process: scrum coach. This role is responsible for keeping track of the skills matrix in the development team and informing the asymmetric pair programming process. In standard scrum development, the team has no skill differentiation. In our environment, however, there are a variety of skills that coalesce. They come from the unique mix of consultants and armed forces personnel needed to be able to tackle the complexity of user requirements and the context within which the armed forces operate today. Not keeping track of the team's skill set would be dangerous because it could result in the misunderstanding of the tacit assumptions that often are an integral part of user stories.
6. A network of all relevant stakeholders (decision makers, top users, and application domain experts) is established at the project start and managed by the GPO as a professional social network. The network is used to share all the project-relevant information and is consulted when preparing all major project decisions.
7. All the scrum-derived "rituals" (stand-up meetings, sprint planning, reviews, and deliveries) are documented in electronic form or taped.
8. The reconciliation of the roles played in iAgile and the roles played as members of the armed forces needs to be addressed strongly and decisively through change management practices. These include, for example, a top-level commitment, in our case coming directly from the Army Chief of Staff; a clear sense of urgency, which in the Army is easier to instill; and a clear identification of the final outcome, which generates the motivation to achieve such outcome and also provides a clear, measurable target whose achievement can be detected and measured unequivocally [14]. This last point merits some further elaboration. Often, project managers define the final outcome in terms of percentage of implementation of the product's requirements and deviation from the planned budget. In our case, however, the outcome is defined in terms of costs, customer satisfaction, and quality. These project objectives are independent of the method

and tools used to manage the project and can be defined unequivocally while allowing uncertainty about requirements and the development process, which is the staple of agile development methods.

The principles articulated above contribute to create a very different process from the standard enterprise scrum. This creates a number of challenges, the most relevant of which are training and user community governance. Training is crucial for team members so that they can become productive team members immediately. Because iAgile is new and evolving, training evolves as well. It was thus natural to involve academia not only in the design of the process but also in the development of appropriate training programs for the various roles.

User community governance is a challenge because the concept itself, at the beginning, was nebulous for the armed forces. The challenge arises because of the large number of implicit assumptions when describing user stories in a highly specialized environment. Normally, this would not be a problem because the development team is familiar with that environment. However, it turns out that the assumptions are rarely checked for consistency; and when consultants are added to the environment, it becomes apparent that leaving these assumptions in the implicit state does not lead to successful projects. The following sections briefly describe how we approached both challenges.

12.3.1 Training

This is probably one of the areas containing the most significant differences when compared to traditional agile. iAgile is continuously changing to incorporate the awareness of potential vulnerabilities concerning mission-critical applications. Up-to-date knowledge on software engineering at the theoretical level is required to fully understand the true nature of the software product and the development of the production techniques. Depending on the specific role in the process, more focused training is needed, emphasizing realistic exercises and role playing.

All the components of the development teams (technical, management, and specific domain expertise) have to be familiar with the key concepts of software engineering and its evolution to fully understand the nature of the software product and be aware of the differences between the software development process and any other industrial product. This awareness has to be shared in the teams, and for this reason a common theoretical introductory part is included in every iAgile training course. A specific practical part depending on the role to be covered in the production process follows the first segment. Software engineers are trained on specific iAgile techniques such as requirement gathering and understanding and the sprint execution, whereas process managers are trained on noninvasive program management techniques.

12.3.2 User Community Governance

One of the key features of iAgile is the full involvement of the user community in the software development process. The area of mission-critical software in the Defense & Security area is characterized by substantial complexity of the user requirements often specified through a synthetic "mission need" document where most of the real needs are implicit or embedded. The user/stakeholder has relevant domain-specific expertise, which is very difficult to elicit but is pivotal for the correct implementation of the application. As reported in the case study, it is quite normal when the user begins to fully understand his or her own requirement only after several product deliveries. Besides the direct involvement of selected user experts in the development teams, in this environment, the realization of a network where a broader community of users/stakeholders can be reached is vital for the success of the iAgile project.

12.4 iAgile Support Tools

In iAgile, a number of noninvasive tools are used to monitor the rapid production cycles and report in real time on all necessary effectiveness indicators. Typical measures are code complexity (McCabe, Halstead, etc.), number of produced lines/programmer/day, number of bugs and fixes (code defect rate), number of user stories elaborated, user story quality estimation, US tracking, etc. Most of the measures are taken in real time and shared with the development teams using a simple presentation software product (software development control room).

Similar to all agile approaches, iAgile is based on the values of the Agile Manifesto and in particular on "Individuals and interactions over processes and tools" [15]. However, this does not mean that tools are not important in iAgile (or in any other agile approach); it means that tools should not create artificial constraints to how people work and collaborate, as they are only a support to perform activities in a more efficient way. In particular, tools should be easy to use and flexible enough to adapt to the preferred way of working adopted by the development team. Therefore, the tools originally developed to support traditional, plan-based development approaches can be hardly adapted to support the agile ones. There are a number of software vendors that propose their tools that have been designed to support traditional approaches with new versions and/or extensions to include also the agile approaches. However, such one-size-fits-all approach to support any development methodology adopted inside a company may create several problems to agile teams, in particular:

1. *Too many functionalities*: supporting a number of different development approaches forces tool vendors to include many features that can make the tools difficult to use and require a relevant amount of training. In particular, agile approaches usually require a limited amount of functionalities compared to

a traditional, plan-based approach; therefore, developers are easily lost in such environments and demand simpler tools that focus on their needs.
2. *Difficulty to configure*: due to the support of a wide range of processes, such tools require a complex customization activity before the tools can be actually used. Moreover, such configuration needs to be synchronized among all the installations that are usually local.
3. *Heavy tools*: the amount of functionalities offered is also connected to the systems requirements for the machines running such tools that may interfere with the development activities slowing down the developers' machines.
4. *Use of old paradigms*: most of such tools are based on old-style client-server architectures, require specific operating systems, and are not available through browsers or on mobile devices. This is a direct consequence of the evolution of the tools designed to support traditional, plan-based approaches that have been designed a long time ago, and vendors find their adaptation to the new paradigms difficult to implement.

Such limitations affect agile teams in a negative way, reducing their effectiveness in particular in the early stages of the introduction of an agile approach in a company. This is an important step: the team members need to develop in a very different way compared to the past, and the results of such early adoptions are often under strict scrutiny by the management. Such results may affect deeply how agile is perceived in the organization and whether the new methodology will be widely adopted in the company or dismissed as ineffective.

Even with all these limitations, this one-size-fits-all approach is considered the safest one in many organizations that are used to traditional development approaches for several reasons:

1. *Well-known tools*: the agile support has been added to tools already used in the organization providing at the beginning a certain level of confidence to both the management and the developers. However, in many cases, developers realize the limitations of such tools quite soon, dramatically reducing their use since they do not fit perfectly the process they want to adopt.
2. *Well-known vendors*: many providers of agile specific solutions are quite new to the market, and many large companies have never acquired software from them before. This is particularly critical when vendors are start-ups and small and medium enterprises (SMEs) that could disappear from the market at any time creating relevant problems to the users of their technologies. For this reasons, many companies prefer to adopt tools from large and well-known players with whom they have long-term relationships and they think they can trust. This is a relevant risk that is taken into consideration by many companies, especially the large ones. However, the increased use of open data formats can mitigate it and reduce the real risk of adopting tools from such start-ups and SMEs.
3. *Tools homogeneity*: introducing new tools in a company (especially in the large ones) requires a relevant amount of effort (e.g., installation of server and client components, system administration, training, etc.). For this reason, the tools used by the different team are often standardized, and the organizations prefer to

adopt for agile development the tools that are already well known. However, this may affect the agile teams as discussed before.

4. *Integration with already existing systems*: many project management and development support tools are deeply integrated with the information systems of a company, especially in the large ones. This integration provides several advantages in the day-to-day operations but creates a huge resistance to change. Moreover, in many cases, the processes are adapted to the existing infrastructure and not vice versa. This creates a number of problems for the flexibility of an organization and prevents them to experiment new approaches even if they can provide relevant benefits. Such deep integrations are not implemented anymore in modern architectures since the flexibility of the processes has acquired more relevance.
5. *Communication with the management*: monitoring the status of a project is of paramount importance for the management. This activity is often performed through standard reporting systems that are deeply integrated with the information systems of the company. Agile is completely different from this point of view, and standard reporting approaches are usually difficult to implement and are hardly adaptable to such approaches. Therefore, agile teams are often perceived as black boxes where it is difficult to expose the status of a project to other parts of the organization. For this reason, specific tools that fit the agile environment but are able to communicate with the management outside the team are needed to guarantee the success of agile approaches in many organizations.

12.4.1 Evolution of Tools

iAgile is designed to bring agile approaches inside mission-critical projects that are traditionally implemented through plan-based methodologies. Usually, such projects are carried out inside organizations with a quite rigid structure that consider agile approaches difficult to be implemented in their context. In this kind of context, regular agile development techniques cannot be adopted for several reasons, including the lack of specific tools designed to support the agile development in conjunction with the overall organization structure. Mission-critical projects often have strict requirements about providing elements that are needed by certification authorities. Even if certification standards do not prescribe the usage of a specific development approach, almost only the plan-based ones have been used in such contexts, and only in such areas support tools are available. However, the market needs (e.g., reduction of costs, improvement of customer satisfaction, volatility of requirements, etc.) are pushing companies to experiment and adopt novel development approaches. Therefore, the development of specific tools able to support the agile development considering the certification aspects is needed.

For a successful implementation of the iAgile methodology (and for many other agile methods), many support tools are required to speed up activities, share information easily, and assess continuously the status of a project [16]. However,

according to the values and the principles of the Agile Manifesto, the development team has to focus on delivering working software removing whatever is not valuable for the customer [17, 18]. Moreover, the same set of values and principles state that the development team should periodically reflect on how to become more effective and tune their behavior accordingly. This implies that a deep analysis of the effectiveness of the development team is performed analyzing different aspects of the work such as the effort spent and the quality of the source code, as such aspects are connected with the principles of the Agile Manifesto.

The implementation of a monitoring activity that inspects continuously the effort spent and the quality of the source code has been implemented in very different ways in the past following the guidelines identified by the personal software process (PSP) and the team software process (TSP) in conjunction with the requirements of the levels of the Capability Maturity Model (CMM). However, such approaches have several limitations including:

- They have been developed in an era when all development approaches were plan based (mainly waterfall, V-shape, and iterative).
- The data collection and analysis was only partially automated requiring a large amount of context switching between the productive activities and the monitoring ones. Such switches are known to create a number of problems interfering with the productive activities.

Therefore, the monitoring activities need to be implemented in a completely new way to be effective for an agile development team.

12.4.2 Noninvasive Measurement Tools

Noninvasive measurement techniques are the only approaches able to satisfy both the continuous monitoring needs and the strong focus on valuable activities for the customers. For this reason, a noninvasive measurement infrastructure is needed to support effectively the iAgile development process.

In particular, during the early phases of the definition of the iAgile methodology, several research prototypes have been adapted and tested in the team to verify the suitability and effectiveness of the approaches. Such prototypes were able to collect both process and product metrics. Process and product metrics are discussed below:

- *Process metrics*: such metrics describe characteristics of the development process investigating the amount of effort spent in specific activities and the role of each team member inside the development, where the knowledge is kept and how it is shared among the team members; how defects are collected, managed, and fixed; etc. Basically, process metrics provide an analysis (at different levels of detail) of the behavior of the team as a whole, of the behavior of each team member, and of the collaboration among team members [19–23]. Such analyses are useful to perform several activities including:

- *Providing a high-level overview of the status of the project*: data collected from each team member and from repositories (e.g., issue tracking systems, version control systems, etc.) can be integrated to provide a big picture of the overall status of the project and the development team identifying where the effort of the team is spent, how much of it is dedicated to the development of new features and how much to the maintenance of the already existing ones, etc. Moreover, this kind of analysis can provide information about how fast is the team in implementing user stories, the level of completion of the project, the level of volatility of the requirements, etc.
- *Providing feedback to each team member about its contribution*: a data collection and analysis program is effective only if the people involved get benefits from it. In particular, the implemented program is designed to provide detailed reports about the activities performed by each team member describing the activities performed at a fine-grain level to help them to improve their work habits and improve their overall effectiveness.
- *Accessing the level of collaboration inside the team*: the data collected is able to highlight how much collaboration is performed inside the team looking at the artifacts that are shared and are modified. This allows an indirect assessment of how people collaborate and how the knowledge about specific parts of the source code is shared across the team identifying areas that need improvements. In particular, it is possible to identify areas that are not covered enough, and there is a need of ad hoc training activities and/or knowledge sharing among team members.

Process metrics are extracted by different tools that are intended to collect different kinds of data. In particular, there are different kinds of process data that can be collected. We can classify them as metrics collectable online and metrics collectable off-line:

- *Online metrics collection*: the data collection system required the installation of plug-ins for the development environment to be able to monitor constantly the opened files and how much time each developer spent in each file, class, and method of the system under development. Moreover, another tool working in background was able to trace all the active applications and keep track of the time spent in each one. This tool was useful to trace the applications used other than the development environment. Such data need to be collected during the development since they are not collected by any other system. Therefore, a continuous data collection is important to avoid missing data.
- *Off-line metrics collection:* the data collected in this case are collected in an indirect way from a number of different kinds of artifacts such as bug reports and source code commits. In particular, data about time required to close issues and the related fix, who deal with them, etc. Such data can be collected at any time since it is stored as a side effect of storing other information (e.g., bug reports), and it can be collected periodically without losing quality.

- *Product metrics*: such metrics describe characteristics of the source code investigating a number of quality aspects such as its maintainability, its architecture, its complexity, etc. Basically, product metrics provide an analysis (at different levels of detail) of the structure of the source code and its evolution over time highlighting areas that are crucial for the overall system and the ones that need to be improved [24–27]. Such analyses are useful to perform several activities including:
  - *Assess the quality of the system*: data are collected at different levels of granularity providing information about different quality aspects at different levels (e.g., method, class, file, package, etc.). This kind of information can be compared to the levels that are accepted by the literature and by the specific requirements of the system developed. This allows the identification of areas of the source code that are likely to generate problems in the future and prevent them. Moreover, such analysis can be used to train newcomers to become confident to the different parts of the code and help them in becoming productive as soon as possible.
  - *Provide an overview of the evolution of the system*: the data collection can be performed at different stages of the development. Ideally, it can be performed every day to provide a continuous analysis of the evolution of the system and detect early possible problems that may arise. The analysis of the evolution of the system helps in the identification of architectural problems and helps in the definition of possible solutions.

Product metrics are extracted by different tools that are intended to collect different kinds of data. In any case, all the data are collected off-line connecting to the version control systems that store the source code. In many cases, the extraction of the data is performed through a static analysis of the source code. It is a good practice to store code that actually compiles and link properly to have all the data extracted since some tools require at least code able to compile correctly. However, many tools are able to extract data even from code that does not compile.

Besides the data collection, another important aspect is the integration of the collected information and the generation of visualizations that the different stakeholders find useful [28]. Different development teams and different companies have very different needs that require an extremely flexible reporting approach that may include the visualization through a web page, the generation of a custom report, the generation of a periodic email, etc. The flexibility of the reporting is important to allow the team to respond quickly to internal ever changing needs and to the inquiries coming from the management that prefers the traditional way of reporting about the status of a software project.

12.5 iAgile Benefits

12.5.1 Costs

Not all the cost reduction factors have been fully analyzed since many of them are related to human and social factors such as the growing capability of the production teams to deal with the specific product complexity. Relevant savings have been identified due to the implementation of a more effective production structure (resources and time) and a better understanding of the user needs/requirements [29].

12.5.2 Customer Satisfaction

The improved agile methodology described in this chapter is heavily based on a strong involvement of the user in the production cycle. The involvement is not limited to the initial phase where the user needs are defined and the user stories negotiated, but is continuously applied through the life cycle. Experts from the user communities are trained on the basic feature of iAgile and fully included in the production teams. During the sprint reviews, the "customer" top-level representative who is asked to accept the product delivery may receive the final presentation briefing by a component of his own community who is acting as team product owner.

12.5.3 Quality

The quality benefits introduced by iAgile are subtler to characterize than initially expected. In most agile methods, there is an initial assumption that quality will be higher because new code will be developed only when needed, immediately integrated, and eventually known by the majority of the development team. This assumption is not necessarily valid in iAgile, and in fact we argue that when multiple teams work on different code bases, the beneficial effects of such practices like pair programming and continuous integration are reduced because the physical separation of the teams creates a knowledge separation as well, whereas the code is inherently connected and often linked by unforeseen side effects that creep beyond the separation by protocols.

Thus, the error rate in our development is not expected to be lower than the one we measured in previous developments. However, we expect a marked improvement in the quality as perceived by the customer, because of the following fundamental reasons:

- The customer is able to witness development as it happens and is able to influence the development of his or her requirements in a direct and productive manner.
- The code base being developed using iAgile is smaller than the corresponding code base developed using a traditional method because only the requirements needed by the customer survive in the code base. Thus, the scope of the code base is more precise and responsive to the user needs.
- The faster feedback mechanism allows the user to signal errors early on in the development of a functionality, which increases the probability of passing verification and validation earlier than with other development methods.

12.6 Applying iAgile: A Case Study

To put the method described above to the test, we have decided to apply our method to a real defense project, thus creating a case study. This case study contains our experience in creating a command and control system for the 4th Logistic Division of the Italian Army's General Staff. This project was the first agile pilot the Army approved and was instrumental to investigate development cost reductions. Additionally, the constantly changing conditions in the theater of operations demanded an approach that could deal with rapid changes in user requirements, often mandated by previously unknown operational conditions. This experience has been continuing for about two and a half years, and although a first operational release has been issued, it is still being improved and expanded as the users gain more confidence and understanding of the possibilities of this means of software development.

12.6.1 Introduction to LC2EVO

In 2014, the Italian Army General Staff Logistic Department started the development of the Land Command and Control Evolution (LC2EVO) system. This evolutionary military command and control (C2) software system was based on the "Mission Thread"-based approach adopted by NATO.

This effort was generated by the urgent need to support the evolution of the land component of C2 achieving high "customer" satisfaction in a situation of extreme volatility of the user requirement. At the same time, it was necessary to substantially reduce the budget necessary for this software development and maintenance.

The Army software developers started the development using a commercial agile software development methodology available at the time: scrum. This methodology was, and still is, very successful in commercial environments, where it is often indicated as the method of choice for the majority of the Android- and Linux-based software application producers.

When we initially started, we had only one experimental scrum team building LC2EVO,, and we adopted production cycles of 3 weeks. The major key actors with programmers were subject matter experts, security specialists, a scrum master, and a product owner. The team components were taken from industry and military and were located at the army staff main facility. The production was extremely successful, and even the very first sprint, which was supposed to be only a start-up trial one, actually delivered the planned product.

Subsequently, though, the increasing product complexity and the growth of stakeholders' expectations made it necessary to create an ad hoc methodology. It had become very clear that the commercial scrum methodology was not capable of handling the peculiarity of a high-reliability software production with such an articulated and extended user community as the one in charge of the land C2 operational requirement.

A community of interests stemming from the Army but including experts from the universities and the defense industry conceived a customized agile software development process, iAgile, and applied it to the LC2EVO production. This methodology is currently shared with an even broader community including defense industries, universities, and the software engineers taking part in the Defense & Security Software Engineers Association (DSSEA).

The structure of the LC2EVO software is characterized by core services and functional area services (FAS). The core services are web based, and the FAS are derived by the Mission Threads defined in the International Security Assistance Force's Concept of Operations (ISAF CONOPs) and currently adopted by NATO. Core services and the individual FAS software components can be separately changed to accommodate the particular mission need defined by the military user. At the same time, all the FAS can share the data and the artifacts developed individually maximizing code reuse.

The first practical test for LC2EVO was in a NATO exercise at the Coalition Warrior Interoperability eXploration (CWIX 2015[1]), with very positive results, particularly in the area of software security.

In LC2EVO, the core service software segment provides integrated management of all operationally relevant events using a common cartography layer. Interoperability with all the necessary NATO-listed software packages is guaranteed. Collaborative generation of all the documents necessary for the command post activity is obtained largely by making use of common off-the-shelf (COTS) product. Garrison (static situation) infrastructural data flow for common daily homeland-based operations is fully integrated. Both the georeferenced military cartography and the web-based commercial ones are developed and integrate all the available information (weather, traffic, etc.).

The FAS are under continuous development and change for every release. The principal ones are as follows:

[1] www.act.nato.int/cwix

- Battle Space Management: designed to provide the C2 support for homeland security operations, as an embedded capability of tracking friendly secure mobile units using multiple types of cartography. Voice and messaging capability are implemented as well as an integrated NATO JOCWatch feature.

  The LC2EVO infrastructure FAS have been realized with large reuse of the code and functions realized before. The core capability of FAS is to provide an extensive and detailed set of functionalities needed to manage the army real estate.
- Joint Intelligence, Surveillance, and Reconnaissance (ISR): provides management and analysis functions for the intelligence preparation of the battlefield.
- Joint Fire and Targeting: supports all the coordination and planning activities related to the fire power support including all available effectors.
- Military Engineer and Countering of Improvised Explosive Device (IED): initially designed to support the collection and management of data about unexploded ordnance (UnExO) from WW2 found on the national territory, this was soon expanded to provide support to the counter-IED operations (attack the net and force protection).

This LC2EVO software segment has more than 1000 registered users from more than 600 military facilities in the homeland and abroad. Reuse is going to happen as the realization of the strategic tool Infologistica FAS, to be used to give to top-level military commander the real-time situation of all army materials, is completed.

12.6.2 Implementing iAgile

Although the consolidation of all procedures, tools, and methodological peculiarities is still ongoing, some elements have been clearly identified. Some of these elements will be now briefly discussed.

Although the project was initially started using a custom version of scrum, a number of relevant deviations from this methodology were almost immediately applied. One of the most significant ones concerns the user story collection process. It appeared that a simple set of interviews with selected users was not giving to the software engineers the expected quantity of information to decide which software functionalities were more relevant to the users. At the time, the identified issues were as follows: the very relevant expertise of users in their domains, the limitation of the natural language, the link with standard operating procedures, the incomplete understanding of the user of his own requirements, and of course the complexity of the command and control domain.

The high level of expertise in the user, although desirable from the point of view of requirement correctness, has two major shortcomings: (1) the user gives most of the domain knowledge for granted and omits many of the detailed descriptions (e.g., the user says “operation” to indicate the event occurring after the planning, but the specific kind of operation may vary from a convoy movement to an airborne

deployment of a special unit) and (2) the user knowledge is sub-domain specific and mostly experience based, which makes every user different from all the others in the same domain (e.g., a user with relevant experience in out-of-area operations will give more emphasis to the solution of the problems he or she encountered in the last mission, which may be outdated or too "theater specific").

The use of natural language is a point of strength in the agile methodology, but for the description of the user needs in complex domains, it may result in poor or meaningless user stories or in complicated epics with many links to external doctrinal documents which are difficult to understand for the software engineers. One of the methods implemented to correct the above situation was the definition of an iAgile empirical method for the product backlog definition. This method has not been completely formalized yet, but most of its steps emerged during the setup of the LC2EVO production structure and correlated procedures. The first component of the empirical methods is the user story quality evaluation [30]. The second component is the evaluation of the user characteristics: position in the organization (pure user-user/stakeholder-stakeholder). The empirical method works as follows, where a typical multiuser-structured domain is assumed where multiple story tellers contribute to the same product backlog (PB):

Given St_n the story teller n,
WSt_n the weight factor due to the hierarchical position in the User Organization
(a top hierarchical position generates higher priority for the specific User Story),
ISt_n the communication effectiveness of the Story Teller based on its capability to transfer the story teller knowledge to the team (measured in software tasks generated from the specific user story),
UsV User Story Value.

For a specific user story an empirical value can be estimated as:
$UsV_k = WSt_n * ISt_n$

For non-functional requirements (Nfr), a weight factor is then generated by the technical members to recognize the priority of each user story:
Nfr_k {statement}$_k$, where $0.1 < Nfr_k < 1$ (1 corresponding to obligation to execute Nfr associated tasks)

The initial product backlog PB resulting by the first round of interviews is then assembled to generate a list of N scrum-like statements, and will be a weighted list:
PBin= [UsV_1 {statement}$_1$... Nfr_k {statement}$_k$.... UsV_k {statement}$_k$, UsV_N {statement}$_N$]

This initial ranking will be used to generate the first prioritized PB, which shall be negotiated between the development team and the users/stakeholders' project board. This negotiation will take into account the initial values, but quite often the priority will be changed. To trace the evolution of the user stories due to nonlinear elaboration of the stories, a user story risk factor is generated (Sr_k) associated to the de-ranking of the story. So the first workable PB takes the following form:

$$PB1 = [UsV_1\ \{statement,\ Sr_1\}_1 \ldots Nfr_k\ \{statement\}_k \ldots UsV_k\ \{statement,\ Sr_k\}_k, \ldots UsV_N\ \{statement\ Sr_N\}_N]$$

At every production cycle (Sprint) the list is re-elaborated and the risk re-assessed.

12.6.3 Results and Discussion

On all accounts, the implementation of LC2EVO using iAgile was a success. This was ascertained both quantitatively and qualitatively; and although for comparison's sake the quantitative data are the most usable, sometimes the decision to continue with a method or practice is based preponderantly on qualitative decisions.

12.6.3.1 Costs

The foremost parameter that characterizes the need to adopt iAgile in the Italian Army is cost. The development of a command and control system in defense domains is evaluated at around US$ 85 per ELOC [31]. This figure is based on systems developed in third-generation languages, using conventional plan-based development processes. In the Italian Armed Forces, this figure is accepted, and in fact some of the previous projects turned out to be even more expensive. In particular, the Italian Air Force's command and control telecommunication system (CATRIN) cost US$ 65 per ELOC in 1992, which is about US$ 111 per ELOC when adjusted for inflation. A more recent project, the Italian Automated Information System for Command & Control, was completed in 2012 and cost a similar amount: US$ 100 per ELOC. In contrast, the development cost for LC2EVO was US$ 10 per ELOC.

It should be noted that LC2EVO was implemented in a modern language, whereas previous systems had been implemented in an imperative structured language, which should account for part of the lower development costs. However, we also have some additional hypotheses.

One of the reasons for this cost reduction may be attributed to the more effective production structure adopted by iAgile. Traditionally, a large number of activities in the production of a software system are not coding activities. For example,

conflict resolution, resource allocation, documentation production, customer interaction, and demonstrations often become so much part of the development process to take up to 40% of the total personnel time. For LC2EVO, this number is less than 10%. This means that more of the project cost is directly tied to the value perceived by the customer, thus making it possible to reduce activities that may seem important at first blush but reveal themselves to be less relevant in postmortem analyses.

Another reason for the cost difference may be due to the increased interaction with the user, which creates a bond through which user needs are better understood by the team and indeed by the user himself or herself, as the evolution of the product and its continuous testing by the user establishes a virtuous feedback circle. This translates in a very high percentage of requirements implemented eliciting customer approval: approximately 90% for LC2EVO, as compared to about 40% for conventional development methods [31].

Finally, it is possible that the short development cycles created a risk-limiting effect: requirement errors are caught early and corrected when they still have little impact on the project, because no requirement chaining has occurred yet, and thus the amount of rework needed to rewrite the requirement is limited to that requirement only instead of all the subsequent dependent requirements (Fig. 12.2). In fact, the amount of software produced in LC2EVO while a software requirement discrepancy exists is about one-fourth the amount of legacy methods. It is important to note that this particular effect is not a prerogative of iAgile, but is present in all agile and many iterative methods.

12.6.3.2 Customer Satisfaction/Quality

As hypothesized, the quality in terms of defect density or error rate did not differ significantly from that of the previous systems. What changed nearly radically was the acceptance and satisfaction by the customer. This change is to be expected when adopting a user-centric view and an iterative development model, as is the case in agile software development. We do not believe that iAgile differs from other agile methods in this respect.

However, the results prove that iAgile has in fact been able to extend the results of common agile methods to a domain where these methods were not used before. LC2EVO is a mission-critical software product with strong security constraints, and to our knowledge agile methods are not adopted for this kind of software. We believe that iAgile was successful in developing a successful mission-critical command and control product because of two different validations. Firstly, the requirements were validated with the user on a constant basis, resulting in a burndown chart that is similar to that of many successful agile product developments (Fig. 12.3).

Secondly, as previously mentioned, LC2EVO has been tested in the field by means of inter-force simulations (CWIX 2015 NATO exercise) and has behaved

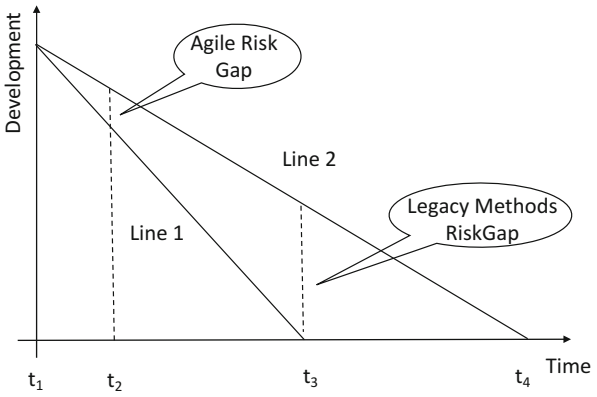

Fig. 12.2 Risk gap between iAgile and traditional methods

exactly as specified both in terms of functional requirements and of nonfunctional requirements, reliability in particular.

12.6.3.3 Learning Curve

Because it was the first time, we applied LC2EVO iAgile to a software development project, we expected to see some sort of learning curve. Figure 12.4 shows what we found while monitoring one of the seven development groups. The two control lines come from literature. It appears that there is a learning curve as it is possible to discern an overall climbing trend in the curve, but it is also apparent that software development in LC2EVO occurred at different speeds, displaying some efficiency peaks of over six ELOCs per developer per hour. This is definitely something worth checking in the future as it might be a function of the difference requirement complexity between sprints, but it could also be a dynamic specifically linked to iAgile.

12.7 Conclusions

This chapter describes iAgile, a software development and project management method that employs agile principles to make it possible to build highly secure, mission-critical software at lower costs and with higher customer satisfaction. This method is a unique combination of agile principles, innovative tools, structured user community, and a custom agile development doctrine that provides a change management path to adopt the process in a traditionally conservative environment while maximizing returns for the users and the engagement for developers.

iAgile is based on a proven enterprise development method, scrum; but the requirements of working in a mission-critical, highly specialized environment with a combination of consultants and army personnel required substantial

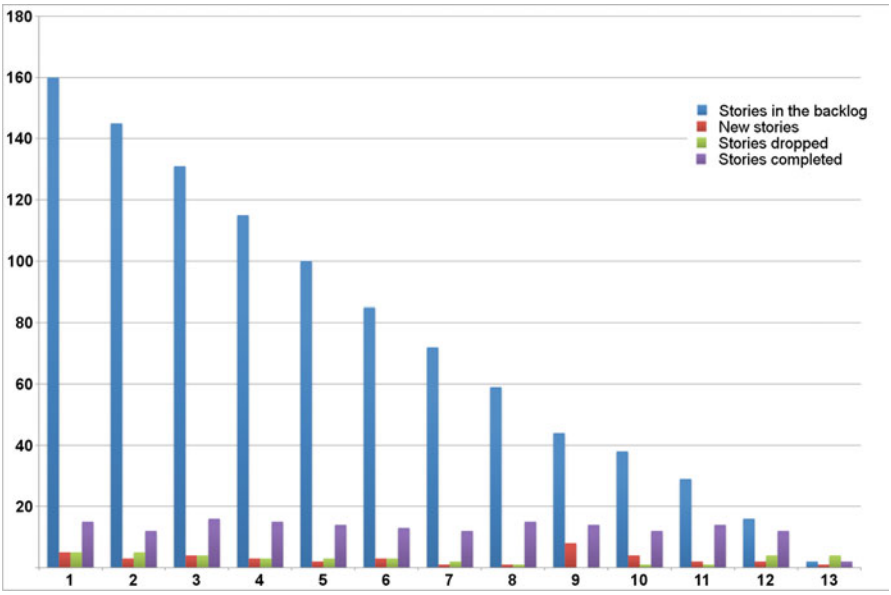

Fig. 12.3 Burndown chart for LC2EVO

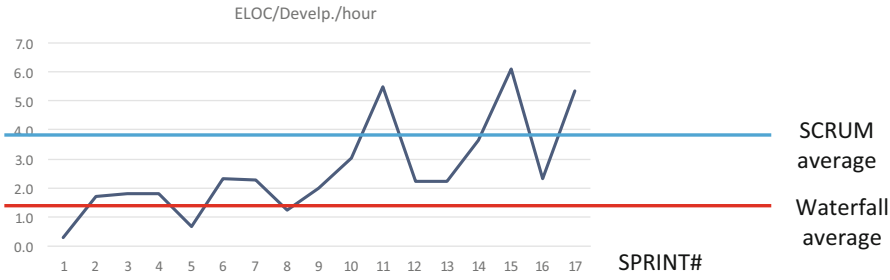

Fig. 12.4 Learning curve in LC2EVO development

changes. These include the introduction of a noninvasive measurement system to create an objective representation of the project and its progress, the specialization of the product owner into a product owner board and team product owners, the enhancement of the scrum master role, a screening method for the composition of each team, the creation of a social network of stakeholders to provide more varied opinions in complex decisions, the recording of scrum rituals, the adoption of change management practices to facilitate the introduction of iAgile in an armed forces context, specialized training, and the creation of a governance structure based on the user community.

The differences between scrum and iAgile necessitated the creation of a unique support infrastructure. The first pillar of this support infrastructure is the specialized training developed in collaboration with academia. The second pillar is the creation

of a series of innovative CASE tools, an effort which is still ongoing, and so far has produced a noninvasive measurement system and a requirement prioritization system. The third pillar is the support for a structured user community governance model, which generated a social network of professionals for reviewing the system and its progress from different points of view such as security, quality, etc. The fourth pillar is a custom agile development doctrine that enables the transferability of iAgile to other armed forces domains and/or groups.

To test iAgile we have adopted it to build LC2EVO, a command and control system for the Italian Army that is able to work in an inter-force context and is fully integrated with NATO communication protocols.

The results show that iAgile has worked as expected in this case, with an overall cost reduction of up to ten times compared with previous efforts in the same domain. However, these results have been demonstrated in one project only, albeit a 2.5-year-long one. Future work will be needed to extend the generality of these results. In particular, it will be necessary to implement iAgile in a completely different project from LC2EVO, to ensure the transferability of the methodology regardless of the team involved. Therefore, it is expected that in the Italian Army, future versions of LC2EVO will continue to be developed with iAgile and that the method will be adopted by more departments to respond to the needs of modern operations in the armed forces.

References

1. Bungay S (2011) The art of action. Nicholas Brealy Publishing, Boston
2. Waldner J (1992) Principles of computer-integrated manufacturing. Wiley, Chichester. ISBN 0–471-93450-X
3. Bicheno J, Holweg M (2009) The lean toolbox. PICSIE, Buckingham. ISBN 978–0–9541244-5-8
4. Agile Manifesto (2001) at http://www.agilemanifesto.org. Retrieved 10 July 2016
5. Chin G (2004) Agile project management. AMACOM, New York
6. Sutherland J, Viktorov A, Blount J, Puntikov N (2007) Distributed scrum: Agile project management with outsourced development teams. In: Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS '07). IEEE Computer Society, Washington, DC, USA, p 274a-. DOI=http://dx.doi.org/10.1109/HICSS.2007.180
7. Hoda R, Noble J, Marshall S (2008) Agile Project Management. In: Proceedings of the New Zealand Computer Science Research Student Conference, Christchurch, New Zealand
8. Chagas LF, de Carvalho DD, Lima AM, Reis CAL (2014) Systematic literature review on the characteristics of agile project management in the context of maturity models. In: Mitasiunas A et al SPICE 2014, CCIS 477, Springer, Cham, pp 177–189
9. Dyba T, Dingsøyr T, Brede MN (2014) Agile project management. In: Ruhe G, Wohlin C (eds) Software project management in a changing world. Springer, Berlin. doi:10.1007/978-3-642-55035-5_11
10. Crowder JA, Friess S (2015) Agile project management: managing for success. Springer, Cham. doi:10.1007/978-3-319-09018-4
11. de Kort W (2016) DevOps on the Microsoft Stack. Apress. doi:10.1007/978-1-4842-1446-6_5
12. AXELOS (2009) Managing successful projects with PRINCE2

13. Tuley F (2015) The PRINCE2® Foundation PDF training manual. Management Plaza, Tremelo
14. Kotter J, Rathgeber H (2013) Our Iceberg is melting: changing and succeeding under any conditions, Pan MacMillan. ISBN 1447257464
15. Benedicenti L, Ciancarini P, Cotugno F, Messina A, Pedrycz W, Sillitti A, Succi G (2016) Applying scrum to the army – a case study. In: 38th International Conference on Software Engineering (ICSE 2016), Austin, 14–22 May 2016
16. Sillitti A, Janes A, Succi G, Vernazza T (2003) Collecting, integrating and analyzing software metrics and personal software process data. In: EUROMICRO 2003, Belek-Antalya, Turkey, 1–6 September 2003
17. Moser R, Pedrycz W, Sillitti A, Succi G (2008) A model to identify refactoring effort during maintenance by mining source code repositories. In: 9th International Conference on Product Focused Software Process Improvement (PROFES 2008), Frascati (Rome), Italy, 23–25 June 2008
18. Sillitti A, Succi G, Vlasenko J (2012) Understanding the impact of pair programming on developers attention: a case study on a large industrial experimentation. In: 34th International Conference on Software Engineering (ICSE 2012), Zurich, 2–9 June 2012
19. Astromskis S, Janes A, Sillitti A, Succi G (2014) Continuous CMMI assessment using non-invasive measurement and process mining. Int J Softw Eng Knowl Eng World Sci 24 (9):1255–1272
20. Coman I, Sillitti A (2007) An empirical exploratory study on inferring developers' activities from low-level data. In: 19th international conference on Software Engineering and Knowledge Engineering (SEKE 2007), Boston, 9–11 July 2007
21. Di Bella E, Fronza I, Phaphoom N, Sillitti A, Succi G, Vlasenko J (2013) Pair programming and software defects – a large, industrial case study. Trans Softw Eng IEEE 39(7):930–953
22. Fronza I, Sillitti A, Succi G (2009) An interpretation of the results of the analysis of pair programming during novices integration in a team. In: 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM 2009), Lake Buena Vista, 15–-16 October 2009
23. Sillitti A, Succi G, Vlasenko J (2011) Toward a better understanding of tool usage. In: 33th international conference on Software Engineering (ICSE 2011), Honolulu, 21–28 May 2011
24. Scotto M, Sillitti A, Succi G, Vernazza T (2004) A relational approach to software metrics. In: 19th ACM Symposium on Applied Computing (SAC 2004), Nicosia, Cyprus, 14–17 March 2004
25. Scotto M, Sillitti A, Succi G, Vernazza T (2006) A non-invasive approach to product metrics collection. J Syst Arch 52(11):668–675
26. Moser R, Sillitti A, Abrahamsson P, Succi G (2006) Does refactoring improve reusability? In: 9th International Conference on Software Reuse (ICSR-9), Turin, 11–15 June 2006
27. Moser R, Abrahamsson P, Pedrycz W, Sillitti A, Succi G (2007) A case study on the impact of refactoring on quality and productivity in an agile team. In: 2nd IFIP Central and East European Conference on Software Engineering Techniques (CEE-SET 2007), Poznań, Poland, 10–12 October 2007
28. Janes A, Sillitti A, Succi G (2013) Effective dashboard design. Cutter IT J Cutter Consortium 26(1):17–24
29. Messina A, Fiore F (2016) The Italian Army C2 Evolution from the current SIACCON2 Land Command & Control system to the LC2EVO using "agile" software development methodology, 2016 International Conference on Military Communications and Information Systems (ICMCIS), Bruxelles, 23–24 May 2016. doi:10.1109/ICMCIS.2016.7496585
30. Messina A, Marsura R, Gazzerro S, Rizzo S (2015) Capturing user needs for agile software development. In: Proceedings of 4th international conference in Software Engineering for Defence Applications, Rome, pp 307–319, doi:10.1007/978-3-319-27896-4_26
31. Reifer D (2004) Industry software cost, quality and productivity benchmarks. Reifer Consultants