Agile Knowledge Engineering for Mission Critical Software Requirements

Paolo Ciancarini, Angelo Messina, Francesco Poggi and Daniel Russo

Abstract This chapter explains how a mission critical Knowledge-Based System (KBS) has been designed and implemented for a real case study of a governamental organization. The KBS is based on an ontology used to merge the different mental models of users and developers. Moreover, the ontology of the system is useful for interoperability and knowledge representation. Both the ontology and the main mission critical functionalities have been developed in agile iterations. The KBS has been used for three development activities: (i) requirement disambiguation, (ii) interoperability with some legacy systems, and (iii) information retrieval and display of multiple informative sources. Moreover, the KBS has been developed using a specific agile software development methodology inspired by Scrum but tailored for Command and Control systems. Due to fast changing operational scenarios and volatile requirements, traditional procedural development methodologies perform poorly. Thus, a Scrum-like methodology, called *iAgile*, has been exploited.

Paolo Ciancarini
Institute of Cognitive Sciences and Technologies, Italian National Research Council (CNR).
Department of Computer Science & Engineering, University of Bologna.
Mura Anteo Zamboni, 7, 40126 Bologna, Italy. e-mail: paolo.ciancarini@unibo.it

Angelo Messina
Innopolis University, Russian Federation.
Defense & Security Software Engineers Association.
Via A. Bertoloni, 1/E – Pal.B – 00197 Roma, Italy. e-mail: segreteria@dssea.eu

Francesco Poggi
Department of Computer Science & Engineering, University of Bologna.
Mura Anteo Zamboni, 7, 40126 Bologna, Italy. e-mail: francesco.poggi5@unibo.it

Daniel Russo
Consorzio Interuniversitario Nazionale per l'Informatica (CINI).
Institute of Cognitive Sciences and Technologies, Italian National Research Council (CNR).
Department of Computer Science & Engineering, University of Bologna.
Mura Anteo Zamboni, 7, 40126 Bologna, Italy. e-mail: daniel.russo@unibo.it
Corresponding author

1 Introduction

The most critical phase in system design is the one related to the full analysis and understanding of "User Requirements". Difficulties arise especially where the ambiguity on the functions to implement is a continuous challenge. Due to the volatility of the user needs, changing of scenarios, and the intrinsic complexity of software products, requirement engineering benefits from an Agile approach in terms of (i) attainment with user's contingent needs, (ii) velocity, and (iii) cost reduction.

Formal methods for requirement engineering have primarily been conceived to drive efficiently the link between customers and developers [14]. They focus on reducing the management risk connected with the initial software production phase. The results achieved by these strategies are controversial and not always cost effective [24]. The diffusion of the use of Agile practices in the software production process is putting the human factor as the key asset to capture and understand the user needs [3].

We experienced an extensive use of methodologies to identify the "unexpressed dimension" of the user requirements and to surface the "implicit" knowledge of users within a real case study of an Italian governmental Agency.

The underlying principle is the methodological formalization of the non-linear human thinking into requirements in the form of agile "User Stories". Such an approach was successfully implemented within a mission critical organization to develop critical software applications. User stories are sentences written in natural language and have a very simple structure. The vocabulary used to write a user story depends on which user describes her need, thus is some sense it depends on the mental model the user has of her needs [42]. Capturing the essence of the users mental models [30] and overcoming the intrinsic ambiguity of the natural language are the two main goals of our study. Multiple dimensions to build a dynamic representation of requirements are the core innovative aspect of this work.

We give a problem definition of how to structure the description of user stories following Agile principles. The lessons we learned and some considerations about the importance of ontology based solutions for Knowledge Based Systems (KBS) in this context are discussed. The proposed approach is useful not only for requirement engineering but also to structure a highly interoperable knowledge representation architecture which enables a fast and flexible use in mission critical contexts.

This chapter is organized as follows. In Section 2 we review the most critical aspects of the use of KBS in mission critical systems; we also recall the basics of the *iAgile* process. Section 3 shows how we manage requirements using an ontology. The use of KBS technologies by the sponsoring organization is explained in Section 4. Finally, we draw our conclusions, summing up our findings in Section 5.

2 Complex software systems specification

In mission critical domains, the velocity of release delivery is often considered as one of the most valuable assets. A release will usually be a partial version of the final product, but the important issue is that it already works usefully for its users. An on-field command view of a military operation (i.e., user view of a Command & Control system) typically is: "*I want the right information at the right time, disseminated and displayed in the right way, so that Commanders can do the right things at the right time in the right way*" [4].

Important functionalities may be developed or refined in the first few sprints, due to the continuous interaction between users and developers. The primary objective of this constant dialogue within the development team is the rise of the implicit and unexpressed knowledge, which will be translated by developers into software artifacts.

One typical example in mission critical contexts is the "situational awareness". It may be described as: "*The processes that concern the knowledge and understanding of the environment that are critical to those who need to make decisions within the complex mission space*" [4].

Such a sentence contains a huge quantity of implicit knowledge. For example, the interpretation of "*those who need to make decisions*" has to be clarified. More generally, in a typical agile user story words like "situational awareness" would be written as "*as the one who needs to make decisions, I want to achieve the knowledge and understanding of the environment that are critical to accomplish my mission*". This statement is, of course, still overloaded with implicit knowledge.

In our case study, this issue was overcome through a careful composition of the team including domain experts. Continuos face to face and on-line interactions allowed to minimize information asymmetry [1] and align the different mental models [30]. The main shared target was to deliver effective software to end users in a fast way.

To understand better the main use cases, consider that a military C2 Information System (IS) for mission critical purposes is essentially built on the exercise of authority and direction by a properly designated commander over assigned forces in the accomplishment of the mission [41].

In order to deliver this capability several integrations have to be taken into account, i.e., hardware, software, personnel, facilities, and procedures/routines. Moreover, such a system is supposed to coordinate and implement processes, like information collection, personal and forces management, intelligence, logistics, communication, etc. These functions need to be displayed properly, in order to effectively support command and control actions [28].

The IS we are reporting on has been based upon the development of mission specific services, called Functional Area Services (FAS), which represent sequences of end-to-end activities and events, to execute System of Systems (SoS) capabilities. These mission oriented services are set up as a framework for developing users' needs for new systems. Furthermore, mission services are characterized by geographical or climate variables, as cultural, social and operative variables, which

represent functional areas or special organization issues. Mission services of the C2 system have been developed according to the NATO–ISAF CONOPS (Concept Of Operations), as required by management of the governmental agency we cooperate with:

- Battle Space Management (BSM)
- Joint Intelligence, Surveillance, Reconnaissance (JISR)
- Targeting Joint Fires (TJF)
- Military Engineering - Counter Improvised Explosive Devices (ME-CIED)
- Medical Evacuation (MEDEVAC)
- Freedom of Movement (FM)
- Force protection (FP)
- Service Management (SM)

Thus a C2 system is made of a set of functional areas which in turn respond to a number of user stories.

2.1 Evolution of a Mission Critical Information System through Agile

The mission critical information system we have studied is a Command & Control system which was capable to support on-field missions according to the NATO–ISAF's framework. The initial idea was to develop a Network Centric Warfare system (NCW) [2]. This system supports many of the operational functions defined in the contest of the NCW, according to the requirement documentation. The system has been employed in many exercises and operations and went through several tests. Today the system is serving mission critical purposes in NATO–ISAF operations e.g., the Afghanistan Mission Network.

However, several difficulties and limitations arose. The acquisitions were done according to Waterfall procedures, started in the early 2000s and went on until recently. The obsolescence of the components and related functionalities, along with the maintenance and follow-up costs connected to the Waterfall software life cycle are a big issue. Several problems are related to the impossibility to develop quickly new functionalities required by on-field personnel in a fast-changing mission critical scenario e.g., a modern asymmetric warfare. This led the use of agile software development paradigms which are supposed to overcome this crucial constraints.

Therefore, since 2014 a new "Main Command and Control System" (Main C2) to support the former system (Tactical Command and Control System or Tactical C2) has been developed. It was urgent to support the evolution of the Command and Control system, assuring a higher customer satisfaction in a volatile requirement situation. Moreover, due to budget cuts, the new system had to perform better with less resources. Costs related to both development and maintenance had to shrink rapidly.

Functional Area Services (FAS) are web-based services with a client–server architecture. Any FAS software component can be separately modified to respond to specific mission needs, as defined by users. The Main C2 has been validated in NATO exercise for the first time at CWIX 2015[1], with positive results. Core services are build to maximize interoperability with all the relevant NATO software packages available and COTS product. Therefore, Main C2 is both flexible to implement rapidly user needs, with high interoperability of already existing systems, like the Tactical C2.

To develop it, a new methodology was implemented, applying the principles of the "Agile Manifesto" [3] to both increase the customer satisfaction and reduce software cost. After the Agency's top management decided to go Agile, there was some discussion about the method to use. There was the need to exploit Agile's values and capability but within a mission-critical environment.

Scrum was found as the most suited, since it allows a high degree of accountability [39]. This methodology is very successful in commercial environments and the most widespread Agile methodology [44]. Moreover, it was the methodology which was the best known within the Agency. Therefore, other methodologies were not really taken into consideration, even though they might have given similar results.

The teams are mixed: they include developers from the defense industry and governmental officials, based at the Agency's Headquarter in Rome. The initial production phase was extremely successful and even the start up "Sprint" (production cycle of five weeks) was able to deliver valuable software [12].

What happened was that the expectation of the Agency's stakeholder grew rapidly. From 2014 to 2016, the methodology was refined, to respond to mission & security critical needs of the operations domain. Thus, an ad hoc Scrum-like methodology was developed with the name of *iAgile*, and tested for the development of the main C2 system [26].

This methodology, depicted in Fig. 1, has been developed for critical applications, where requirements change already during the first specification and after delivering the first release. The adaptation of Scrum for the special needs of C2 software systems has also been proposed in [17].

A well known approach to analyzing ephemeral requirements consists of formalizing and prototyping the requirement specification using a suitable language, like for instance Prolog [40]. The Humphrey's Requirements Uncertainty Principle remins us that, for a new software system, the requirement (the user story) will not be completely known until after the users have used it [41]. Thus, within iAgile, Ziv's Uncertainty Principle in software engineering is applied, considering that uncertainty is inherent and inevitable in software development processes and product.

The incremental development approach enables easily any change of requirements even in the later development iterations. In our case study, due to the close interaction between the "requirement chain" i.e., from the customer to the development team, FAS were delivered with a high degree of the customer satisfaction.

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

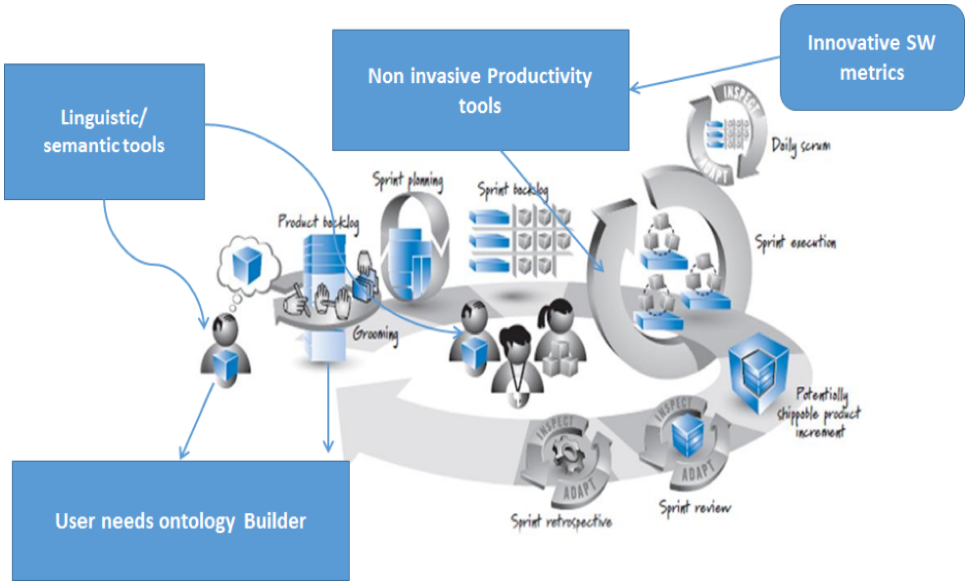

Fig. 1 Sprint representation, inspired by [33]

The Scrum methodology developed within the Agency fully supports the change of requirements according to contingent mission needs. The traditional command chain was adapted to the development needs. Both structured and horizontal characteristics of Scrum are particular effective in a critical environment. These two characteristics are embedded in the model.

Mission critical organizations need to comply with a vertical organizational chain, to empower different stakeholders to their duties. In the field where we had this experience, a hierarchy enforces clear responsibility and accountability within the command chain. So, the customer becomes the accountable official for the mission needed requirement. However, to develop different mission critical requirements, it is crucial to have a straightforward and direct communication and collaboration with final users, according to the Agile Manifesto. Therefore, in the methodology, some user representative becomes part of the development team, allowing a better understanding of the needs and a faster development of the feature.

One of the key strengths of the methodology is its flexibility. The process is defined only at a high level, to be adapted in any theater of operation. It defines values, principles, and practices focused on close collaboration, knowledge sharing, fast feedback, and tasks automation.

The main stakehlder is the Product Owner (PO), who gives to the developing team the first input which is a product's vision. It is a high level definition to address the task that will be refined during the development cycle through the Backlog Grooming.

The Backlog Grooming is a key activity which lasts over the whole development process. It focuses the problem definition, refining redundant and obsolete requirements. Moreover, it prioritizes requirements according to contingent mission needs. The acceptance criteria and the scenario definition are set by the PO in the user stories.

The developing team used by the Agency is composed as follows (such team composition is an adaptation of standard Scrum roles within a mission critical context).

- The *Product Owner* is the governmental official in charge of a specific mission critical function which has to be developed. He provides the general vision of the final functionalities to the whole team i.e., what the system has to do. It may be that PO delegates its role to another official of his team. In this case, the PO becomes a team of people that has to decided about the systems functionalities and discuss them within the development team. Ideally, the PO team has to be representative of the final user, thus it should be made also of real users.
  This crucial role is pivotal for the positive outcome of the sprint. *De facto*, the shortening of the "requirement chain" through the involvement of end users and the constant feedbacks of the PO during the sprint is a key success factor.
  In our case study the stakeholders were initially barely aware about the development process. Due to a constant involvement within the iterations, the stakeholders became aware of the development methodology and aligned their expectations increasing their satisfaction. Through this involvement, there is an alignment of both interests and expectations that raises the quality of the final artifact. So, the final product may not be fancy but down to earth with a high degree of immediate usability by a final user. Therefore, the degree of user involvement is of highest importance since it has a direct impact on the development itself and a ground for building a sense of ownership of the final product which is essential for the acceptation of the final product.
- The *Scrum Master* (SM) is a domain expert and is supposed to lead the development team and the Product Backlog management. The SM shapes the process according to mission's needs, leading continuous improvement like in any Agile team. He has to shield the development team from any external interferences as also to remove any hinder which may occur. What typically happens in mission critical organization is that information is shared only through very structured processes. So, there could be a loss of productivity, due the waste of time to obtain relevant information for the development process. The SM knows how to gain such information and is in charge of sharing it when needed, with no waste of time from the development side.
  According to the critical domain, he is accountable for the team's output. So, he is a facilitator but he takes the control of the team, considering also the different backgrounds of the members. Both PO and SM collaborate closely to refine requirements and get early feedbacks. Furthermore, his role is to build and sustain effective communications with customer's key staff involved in the development. Finally, he is in charge of the overall progress and take responsibility

for the methodology used within the development cycles. So, he may do some corrections within the team to deliver the expected output.

- The *Development Team* composed by both military and civil contractors is in charge of the effective development. The team members are collectively responsible for the development of the whole product. Within the team there are no specialized figures (e.g., architects, developers, testers), and it is the team that organizes itself internally and takes responsibility over the entire product.
  The self organization empowers the team for the execution of the Sprint Backlog, i.e., the developed Product Backlog within the sprint, based on the prioritization by the PO. The team members are lead by the SM who is mainly a problem solver and interfaces with the organization which needed the mission critical product.
  The number of team members is between three and five highly skilled software developers. The absence of a specialization is due the fact that any member is supposed to have a good knowledge about the system developed with a clear vision on the final artifact. Finally, they are also involved in the testing phase, which is carried out by an independent audit commission.
- The *Coach* is an employee of the civilian main contractor and is in charge of the management of contractual issues. Since the typical contractual form for developing contractors is body rental, the Coach facilitates organizational issues which may occur during the development cycles. Her role is to smoothen problem which may rise, to get the team oriented to the development of the artifact.

After each sprint a deployable release of the system is delivered. In order to assure security standards of mission critical applications extra testing is pursued. This activity is carried out before the deployment within the mission critical network. So, before deployment three steps are carried out as follows:

1. The development team runs a first test in the development environment and then in a specific testing environment (stage), having the same characteristics of the deployment environment.
2. Afterwards, testing activities are performed by Agency's personnel involved in test bed based activities, in limited environments to validate the actual effectiveness of the developed systems in training and real operating scenarios (Integrated Test Bed Environment).
3. Finally, testing activities on the field performed to verify the compliance of the developed systems to the national and international standards and gather operational feedback to improve the system's performance and usability.

Only after the positive check of these three steps the functionality is deployed. At the end and beginning of a new Sprint, all interested stakeholders discuss about positive and negative aspects, to improve the next iteration. Therefore, it is an incremental process, which changes with the operational scenario. It is not a frozen methodology, but it evolves along with Agency's needs.

Finally, a quite important outcome of this approach is the cost reduction in all the system's lifecycle. A first assessment of the product cost per "line of code equivalent" with respect to other comparable internally-produced software showed a cost

reduction by 50%. To consider those costs we computed a comparable software by dimension (LOC) and functional area (command and control). We considered all relative cost of personnel, documentation and maintenance costs and fix cost for office's utilities. The assessment after two years showed more significant cost reduction.

Generally speaking, we know from past experiences that, on average, cost per ELOC in similar C2 domains is about 145 dollars; with regard to ground operation the cost is about 90 dollars [32]. This study, in particular, was carried out for Waterfall in a procedural context. Based on Reifer's study, we carried out our evaluation regarding iAgile cost. It was quite surprising to realize that the software we measured had an average cost of 10 dollars per ELOC.

This was possible cutting maintenance and documentation costs, which represent the most relevant part of software development costs [31]. The cost reduction came mainly from the minor rework due to requirement misunderstanding (project risk reduction) connected to the short delivery cycle and to the integration of subject matter experts into the agile teams (asymmetric pair programming typical of iAgile). Moreover, the reduction of non-developing personnel played also an important role.

Since project management responsibilities were in charge of the Agency, the use of internal personnel reduced the cost of hiring industry's senior figures. Also the increase of teams' effectiveness from sprint to sprint led to cost cuts. Due to the incremental domain knowledge acquisition gained through domain experts and user's feedbacks developers were able to produce artifacts which were attained to customer's expectation, decreasing sensibly rework.

3 Requirements engineering, management & tracking

Agile software methodologies like Scrum put the development team at the center of the development process removing the major part of the procedural steps of the legacy methods and the connected "milestone evidence" mainly consisting of documents and CASE artifacts [5]. Agility is supposed to increase the production effectiveness and, at the same time, to improve the quality of the product.

However, in order to go Agile, a Waterfall-like static requirement documentation can not be replaced simply with a product backlog. The old-fashioned Waterfall frozen requirement document is no longer effective to capture the user needs in quickly changing mission critical environments. Replacing structured and consolidated text with volatile lists of simple sentences may result, in the case of complex systems, in a sensible loss of knowledge. Traceability of how the solutions are found and both the user and the developer growth may become "implicit and unexpressed knowledge" which are key elements within a high quality software development process.

Several studies suggest to overcome requirements misunderstanding as soon as possible, in order to improve the project results and to decrease development and maintenance costs within its life cycle [6]. This is one of the reason why the

Agency started to develop some mission critical software in an Agile way, in order to "shorten the requirement chain", fostering software quality and cost reduction.

The ambiguity concerning the functions to implement is an everyday challenge. Due to the volatility of the user needs, changing of scenarios, and the intrinsic complexity of software products, a dynamic requirement engineering worked very well in an Agile environment [15]. However, the most challenging task is to identify the "tacit dimension" of the user requirements and to surface the "implicit" knowledge of users [27].

In most agile approaches requirements are given in the form of "User Stories", which are short sentences in natural language usually describing some value to be computed in some scenario in favor of some typical class of users. Such formalization drives non-linear human thinking in a standardized form where users have to explain how they imagine the system. This approach has been implemented for mission critical applications. Capturing users requirements and overcome the intrinsic ambiguity of the natural language are two of the main goals of this effort. Fully refined requirement specification documents are no longer meaningful; instead they should incorporate some guidelines to help the developers to effectively measure the quality of the features so that these can be improved. The result is a novel proposal based on an evolution of the "Scrum type" Product Backlog, here represented:

- *User Story*. A structured sentence which summarizes the functionality. Example:
  `As <role>`
  `I want to <functionality description>`
  `in order to <goal to pursue>.`
- *Business Value*. Describes the business value of the desired functionality.
- *User Story Elaboration*. It is an extended user story and it details how the functionality has to be implemented.
- *Acceptance Criteria*. Non functional requirements are given, necessary to accept the functionality (e.g., security, compliance to standards, interoperability issues). Moreover, also functional requirements have to be verified, to accept the developed software. Tests are typically focused on these functionalities.
- *Definition of Done*. It is when the story can be considered fully implemented. The Definition of Done includes the Acceptance Critera and anything that the PO believes is necessary that the team does on the code before it can be released.
- *Expected Output*. It is a list of expected outputs from the functionality, once implemented.

Software development methodologies should be inspired by their organization's needs and not by programming concepts. Well aware of Conway's principle [11] it is the mission need that shapes the information system. Not the structure of the organization, which in our case is highly hierarchic and in its communication flows reflects the Waterfall paradigm. Due to the constant iteration between the users' community, through the Product Owner, and the development team, required applications attains users' expectations. Our experience has shown the effectiveness to overcome the limitations of existing alternatives of a Waterfall like requirement

engineering, which is ineffective for complex user requirements, especially in the mission critical domain.

If continuous interaction, typical of Agile, is crucial to overcome structural information asymmetry, which is present in any human interaction [1], experience showed that it is not enough. Any software project, especially Agile, involves different people, with different backgrounds and experiences. In other terms, we all have our "mental models" [20], which are the source of this information asymmetry. Mental models are psychological representations of real, hypothetical, or imaginary situations, identified by Kenneth Craik [13]. They are mental constructs of the world around us. A mental model is a representation of the world around us and shapes our behavior and approach to problem solving. Like a pattern, once we experimented that the solution works, we tend to replicate it. It helps us to not restart from zero any time we have to face a problem. Thus, it is a simplification. So, it is a mind construct of "small-scale models" of the reality, to anticipate events, to reason, and to underlie explanation [13].

To give an example of the difference between the semantic meaning of a nominal identical concept (i.e., difference in mental models) let us consider the notion of "battle-space geometry". Starting from a user story, a PO may write: "as a commander I want to be able to represent the forward line of my sector on a map to see the deployed materials".

The user has in mind a "line" whose geometrical elements (waypoints, start, finish and type) are decided by a superior command post that is given to him as part of a formatted order packet which he expects to appear on the map by a single click of his mouse and to be updated as the tactical situation changes. The developer's first comprehension will be "drawing" tools able to produce a line by simply calling some graphic libraries. The focus is on how to implement it writing the least possible quantity of new code. This is just an example but it qualifies the differences between the two worlds very well.

For the user the image on the video is just a representation of the real world, for instance a real portion of land where the operation is taking place. Instead, for the developer the same object is the result of a series of interactions showing a map on a video where he has to provide a design tool. As trivial as they may seem these differences are at the root of the software requirement specification problem that in the past has been tackled by freezing the requirement text and operating a certain number of translations into formal representations without reconciliation of the two different mental models.

Some concepts developed in conceptual semantics explain how the representation of the world expressed in natural language is the result of a mediation between the speaker's perception of the world and the current setup of his own mind (i.e., mental models). This poses the question on what we really do communicate about requirements when we use natural language. In [19] this problem is studied, and a solution based on feature maps is proposed.

In our case what emerged is a common ontology used by both users and developers. We found out that working on the ontology in the initial production process (i.e., Product Backlog) improved the effectiveness of the Agile approach. In fact, the

development of a Command and Control ontology, useful as knowledge representation tool as described in the next section, is also effective to merge different mental models and to support requirements traceability [43].

4 Use of KBS and OBS within iAgile

The use of Ontology-Based Systems (OBS) for managing requirements and user stories when applying Agile methods has been explored many times, but it is still an unresolved issue [23, 25]. Some literature suggests that ontology driven development should be the norm, both in general and specifically in the Agile arena [21].

The use of OBS is of paramount importance in a mission critical context. We have experienced it in peace-keeping operations, where rapid information flows coming from different actors (e.g. military, NGOs, citizens, press.) have to be processed. The different needs, contexts, and objectives of these actors are often reflected into a wide range of viewpoints and assumptions, producing different, overlapping and/or mismatched concepts and structures that essentially concern the same subject matter.

Different "organizational routines" [27] lead to different communication standards, along with "tacit knowledge" [29]. Thus, there is the need to organize the different "mental models" [30] around the development process. Ontologies are a powerful tool to overcome this lack of a shared understanding, providing an unifying framework for the different viewpoints and mental models that coexist in vast and heterogeneous contexts.

As described in [43], this shared understanding provides many practical benefits, such as serving as the basis for human communication (reducing conceptual and terminological confusion), improving the interoperability among systems with different modeling methods, paradigms, languages and software tools, and supporting the main system engineering principles (such as reusability, reliability, requirements identification, system specification, etc.). The adoption of ontologies as a core components of software architectures [8] in conjunction with Agile methodology development principles has proven its effectiveness in changing and variable contexts.

An overall idea of the main elements of the development process is depicted in Figure 2. Both KBS and OBS are build on users' mental models. This means that requirements and the ontology represent user's view and needs. So, the user stories collected are the core elements. Following Agile methodologies principles, such artifacts are fundamentals to distill both information about the system to develop, and knowledge on the domain in which such system is expected to operate. User stories have been used to extract the requirements of the C2 system, and to develop an ontology for representing the main concepts of the mission critical domain.

So, the backlog grooming (i.e., the refinement of the user stories) becomes the instant where users stories split. Requirements are defined and the ontology is developed. This split is not straightforward. Considering that ontology's entities definition is very helpful to define better user's expectations, requirements documents are de-

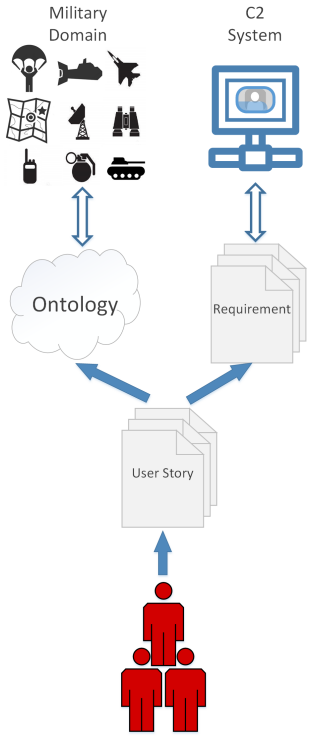

Fig. 2 The ontology of the application domain and the system requirements are derived from user stories

veloped separately from the ontology. While requirements are developed manually by the development team, the ontology is developed by Protégé[2] [22]. As shown in Figure 3, developers already exposed to semantic technologies use this standard tool to develop the domain specific ontology.

4.1 An Ontology-based Architecture for C2 Systems

One of the main challenges in the mission critical domain is the ability of managing in a precise and accurate way the complexity, variability and heterogeneity of information. In particular, the ability of integrating different sources of information, extracting the most relevant elements and putting them into the context is of paramount importance for supporting the tasks of control and decision making. In our approach, ontologies and related technologies are the main tools for facing both the methodological and technological aspects of such context.

[2] http://protege.stanford.edu

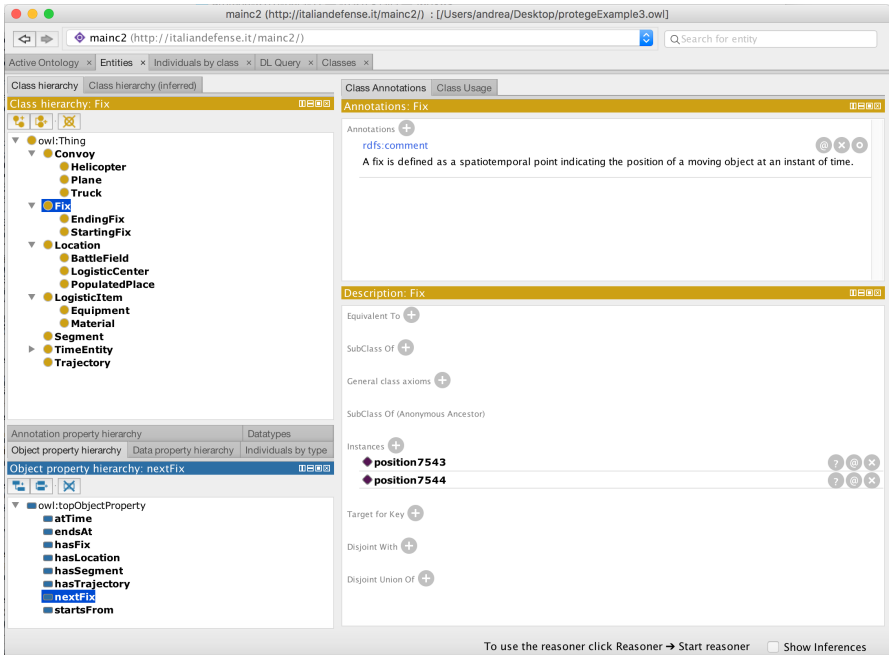

Fig. 3 A snapshot of the C2 ontology during its development with Protege. The class and property hierarchies are shown on the left, while other contextual information (e.g. annotations, instances and relevant properties) are shown on the right.

Similarly to other scenarios, also in the the mission critical domain people, organizations, and information systems must communicate effectively. However, the various needs, contexts and objectives are often reflected into a wide range of viewpoints and assumptions, producing different, overlapping and/or mismatched concepts and structures that essentially concerns the same subject matter. Ontologies are often used to overcome this lack of a shared understanding, providing an unifying framework for the different viewpoints that coexist in vast and complex C2 systems.

As described in [43], this shared understanding provides many practical benefits, such as serving as the basis for human communication (reducing conceptual and terminological confusion), improving the interoperability among systems with different modeling methods, paradigms, languages and software tools, and supporting the main system engineering principles (such as reusability, reliability, requirements identification, system specification).

The central role played by ontologies is summarized by Fig.4, which depicts the overall architecture of the C2 system. The ontology we have developed describes the data model contained in a Knowledge Base (KB), which contains all the information needed by the C2 system. The KB is populated by ad hoc software components (i.e. adapters), that extract information from all the different sources (e.g. legacy

tactile systems, news or ONG CMS, etc.), and convert it in semantic statements that conforms to the ontology. The frequency of the KB populating processes varies from sources to sources. Where possible, triggering mechanisms have been used to identify modifications in the sources and activating the data extraction. Otherwise, adapters performs the data extraction at fixed (and configurable) intervals. Of course, high data quality is a major issue here [7].

The KB contains provenance information defining:

- the origin of the data;
- the agent (usually a software component) responsible for the data extraction;
- other useful metadata (e.g. time information, type - such as insertion, deletion, update - of the KB modification, etc.).

The main advantages of having such information are the ability to discern among different authority levels, the capability of performing comparisons and advanced filters, just to cite a few. Moreover, from a technical point of view, provenance information are of paramount importance to have full control over the KB state during the whole system lifecycle (from the inception and development phases to the final operating period).

The Main C2 system is depicted at the upper vertex of the star architecture. It implements all the functionalities required by the users and described in the collected user stories. All the data required to expose such functionalities to the users are retrieved from the KB by means of standard semantic queries. The same mechanism is used for adding new information in the KB (e.g. for keeping track of the output of the system users' analysis and decision processes).

4.2 Developing domain ontologies from user stories with iAgile

Ontologies play a crucial role in the development of the framework. The primary objective is to develop an ontology that is capable of modeling the complexity of mission critical domains. The starting point and primary source of information for this task is the set of 600 user stories collected from the system final users and domain experts. User stories have been grouped in small buckets (having between 5 and 10 user stories each). The process started by considering a first bucket, that has been used to develop an initial model. Finally, the ontology has been developed iteratively, by adding a new user story bucket to the set of already considered ones at each cycle.

At the end of each cycle, a small dataset (*test dataset*) is created according to the current ontology and considering the user stories under examination. Such a dataset is used to perform a *quality check* on the current ontology. In practice, tests are a series of queries that are derived by analyzing the functionalities described in the user stories. A query test must be executed on the test dataset to check the ontology validity after the modifications performed in the last cycle. In case of a positive result, a new user story bucket is considered and another development cycle

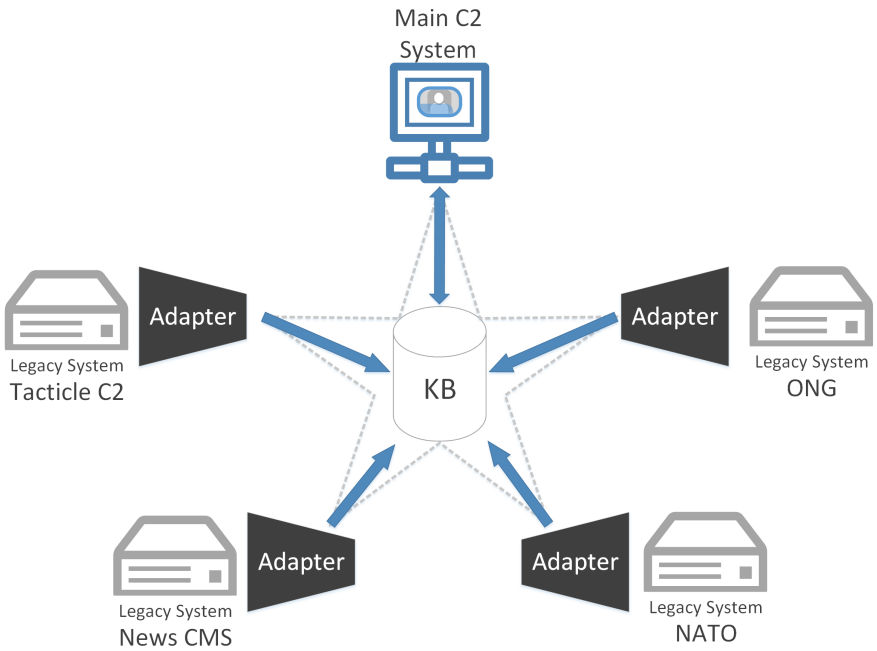

Fig. 4 The C2 system is modeled around the star architecture pattern. The domain ontology is the center of such architecture, and is used to integrate different resources and systems.

is performed. Otherwise, the ontology is refactored and modified until the quality check is satisfied.

In order to clarify the ontology development process, we present in Figure 5 a bucket composed by eight real user stories collected for the C2 system.

Each user story is identified by a unique identifier. User stories have a common fixed structure, where users' roles, objectives and user story specifications are easily identifiable. During a further step of analysis, the concepts that are the main candidates for becoming classes in the domain ontology have been underlined.

An excerpt of the ontology developed starting from the eight user stories is depicted in Figure 6. Three general concepts are modeled: logistic item, location and convoy. Well known ontology engineering principles and best practices described in [16] have been used throughout the whole process. As shown in the ontology fragment, for example, the trajectory pattern [18] has been used to model objects positions and movements. This is an elegant solution for attaching trajectories composed of segment with a geographical or physical extents to any object of the domain.

The process of ontologically modeling the domain has many practical benefits. For example, we successfully converted requirements and constraints to the data model in semantic assertions. Such restrictions can be automatically checked by using popular semantic tools (e.g. reasoners). For instance, we can imagine to add an assertions that states the segments of a trajectory should not be in overlap with

US 2825 (2656)
As <system user>I want <to figure out if the track relative to an object (equipment) represented on the map has been updated, or if the position has just been changed.><Such a status would be highlighted by a green border around the icon on the object. This function should be enabled or disabled by the user.>

US 2828(2659)
As <commander of the logistics>I want <to view a summary, in both tabular and graphical format (eg. histogram), of the efficiency logistic items. ><The total of logistics items and the level of efficiency should be displayed in percentage of the total.>

US 2829(2660)
As <commander of the logistics>I want <to be able to view on a geographical map the geographical distribution of identified logistical items (e.g. equipment, materials, etc.).><The map shall display the concentrations of materials at the logistic centers, represented by a specific colored icon associated with the type of material. The color should reflect the overall efficiency of the logistic item. For each logistic center, it should be possible to view the amount items, with an histogram graph showing both total items along with the percentage of efficiency.

US 2822(2653)
As <system user>I want <that the system can receive and display information about convoy (e.g. trucks, helicopters, planes, etc.) - for example Moving Convoy, Halted Convoy, etc.>

US 2821(2652)
As <system user>I want <that the system can store information about convoy (e.g. Moving Convoy, Halted Convoy, etc ...).>

US 2820(2651)
As <FAS web user>I want <to be able to view on a geographical map the geographical distribution of identified logistical items.><The map shall display the concentrations of materials at the logistic centers, represented by a specific colored icon associated with the type of material. The color should reflect the overall efficiency of the logistic item. For each logistic center, it should be possible to view the amount items, with an histogram graph showing both total items along with the percentage of efficiency.>

US 2819(2645)
As <FAS web user>I want <to be able to view on a geographical map the geographical distribution of identified logistical items.><The map shall display the concentrations of materials at the logistic centers, represented by a specific colored icon associated with the type of material. The color should reflect the overall efficiency of the logistic item. For each logistic center, it should be possible to view the amount items, with an histogram graph showing both total items along with the percentage of efficiency.>

US 2711(2296)
As <system user>I want <to display on the map the trajectory and the last positions of a specific object. This functionality should be enabled or disabled by the user.>

Fig. 5 User stories collected with iAgile are used to develop the system domain ontology

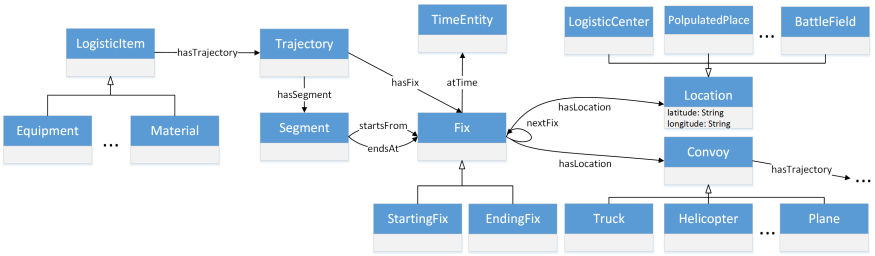

Fig. 6 A fragment of the developed domain ontology. Three general concepts are represented (i.e. logistic item, location and convoy), and the trajectory pattern has been used to model positional information

respect of both the time and space dimensions. In another words, we can impose that an object should not have two different positions at the same time. All the cases that do not respect such limitations in the data are automatically identified as inconsistencies stated by the semantic reasoners.

5 Conclusions

Knowledge Based Systems and Ontology Based Systems are key elements for the development of software-intensive mission critical systems. We reported about a real case study concerning a mission critical system developed for an Italian governmental Agency. Volatile requirements and fast changing operational scenarios led to the choice of a new development process model, transitioning from Waterfall to Agile. However, Agile is not a *panacea per se* but needs to be adapted for complex mission critical purposes. We customized Scrum into *iAgile*, developing a flexible but structured paradigm. Ad hoc steps were designed to comply with both velocity and security. Moreover, we found that such methodology led to an important saving of development and maintenance costs.

The role of the KBS we have used is double: to disambiguate requirements and to build an ontology for interoperability and knowledge representation. The ontology was designed using semantic tools during the requirement specification. Moreover, at the same time, the user story elaboration is carried out for the functionality development. This process allowed to align the different mental models of users and developers. Therefore, after the first formalization of entities, it supported the next Sprint backlogs with a high relevance to users' expectations.

The big advantage of the use of an ontology derives from the interoperability with other legacy systems. In a real-world operational scenario, the mission critical information system is fed by information of different provenance. As shown, also non-governmental actors may deliver useful elements for an up to date situational awareness. So, from different sources it is possible gather data in a flexible and incremental way improving the information completeness, necessary to take mission

critical decisions. Moreover, through the relationship between ontologies of other governmental or intergovernmental agencies both interoperability or replacement of such system is highly simplified. Since the Agency is supposed to offer security services in multilateral and multinational operations, interoperability is of strategic importance. Therefore, the presented approach is an important driver for a smooth and effective system deployment in a mission critical environment.

Future research will go in several directions.

A comprehensive approach to OBS, based on the acquired experience, will be implemented within the Agency with the aid of knowledge-based tools. Moreover, a Machine Learning approach in which requirements are automatically processed to assist the continuous development with Scrum [38] has also to be developed. Also, the use of concurrent development methodologies to support velocity and reliability has to be improved [34], along with a flexible system's architecture. Especially, the reliability in terms of systems "antifragility" of mission critical applications needs further investigation [35], [36]. Although we are aware of the relationship between the software quality dimension and its architecture [37], still efforts need to be pursue to figure out how Agency's business goals (e.g., velocity, cost reduction) impact on the system. Finally, also issues related to software reuse have to be explored [10, 9], since the cloning practices, also in critical systems, is quite common.

Acknowledgements The Authors wish to thank the Consorzio Interuniversitario Nazionale per l'Informatica (CINI) for the partial support in the context of the AMINSEP project. We also thank CNR for supporting the authors Ciancarini and Russo.

References

1. G. Akerlof. The market for lemons: Quality uncertainty and the market mechanism. In *Essential Readings in Economics*, pages 175–188. Springer, 1995.
2. D. S. Alberts, J. J. Garstka, and F. P. Stein. Network Centric Warfare: Developing and Leveraging Information Superiority. Technical report, DTIC Document, 2000.
3. A. Alliance. Agile manifesto. *Online at http://www. agilemanifesto. org*, 6(1), 2001.
4. J. B. Bearden. Command and control enabling the expeditionary aerospace force. Technical report, DTIC Document, 2000.
5. L. Benedicenti, F. Cotugno, P. Ciancarini, A. Messina, W. Pedrycz, A. Sillitti, and G. Succi. Applying scrum to the army: a case study. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 725–727. ACM, 2016.
6. B. Boehm and V. R. Basili. Software Defect Reduction Top 10 List. *Computer*, 34(1):135–137, 2001.
7. P. Ciancarini, F. Poggi, and D. Russo. Big Data Quality: a Roadmap for Open Data. In *2nd IEEE International Conference on Big Data Service (BigDataService)*, pages 210–215. IEEE, 2016.
8. P. Ciancarini and V. Presutti. Towards ontology driven software design. In *Radical Innovations of Software and Systems Engineering in the Future*, pages 122–136. Springer, 2004.
9. P. Ciancarini, D. Russo, A. Sillitti, and G. Succi. A guided tour of the legal implications of software cloning. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 563–572. ACM, 2016.

10. P. Ciancarini, D. Russo, A. Sillitti, and G. Succi. Reverse engineering: a european ipr perspective. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 1498–1503. ACM, 2016.
11. M. Conway. How do committees invent. *Datamation*, 14(4):28–31, 1968.
12. F. R. Cotugno and A. Messina. Adapting Scrum to the Italian Army: Methods and (Open) Tools. In *IFIP International Conference on Open Source Systems*, pages 61–69. Springer, 2014.
13. K. Craik. The nature of exploration, 1943.
14. S. Easterbrook, R. Lutz, R. Covington, J. Kelly, Y. Ampo, and D. Hamilton. Experiences using lightweight formal methods for requirements modeling. *IEEE Transactions on Software Engineering*, 24(1):4–14, 1998.
15. S. Gazzerro, R. Marsura, A. Messina, and S. Rizzo. Capturing User Needs for Agile Software Development. In *Proceedings of 4th International Conference in Software Engineering for Defence Applications*, pages 307–319. Springer, 2016.
16. A. Gomez-Perez, M. Fernández-López, and O. Corcho. *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer Science & Business Media, 2006.
17. D. Harvie and A. Agah. Targeted scrum: Applying mission command to agile software development. *IEEE Transactions on Software Engineering*, 42(5):476–489, 2016.
18. Y. Hu, K. Janowicz, D. Carral, S. Scheider, W. Kuhn, G. Berg-Cross, P. Hitzler, M. Dean, and D. Kolas. A geo-ontology design pattern for semantic trajectories. In *International Conference on Spatial Information Theory*, pages 438–456. Springer, 2013.
19. N. Itzik, I. Reinhartz-Berger, and Y. Wand. Variability analysis of requirements: Considering behavioral differences and reflecting stakeholders. *IEEE Transactions on Software Engineering*, 42(7):687–706, 2016.
20. P. N. Johnson-Laird. *Mental models: Towards a cognitive science of language, inference, and consciousness*. Number 6. Harvard University Press, 1983.
21. H. Knublauch. Ramblings on Agile methodologies and ontology-driven software development. In *Workshop on Semantic Web Enabled Software Engineering (SWESE), Galway, Ireland*, 2005.
22. H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen. The Protégé OWL plugin: An open development environment for semantic web applications. In *International Semantic Web Conference*, pages 229–243. Springer, 2004.
23. M. Kumar, N. Ajmeri, and S. Ghaisas. Towards Knowledge Assisted Agile Requirements Evolution. In *Proc.e 2Nd Int. Workshop on Recommendation Systems for Software Engineering*, RSSE '10, pages 16–20, New York, NY, USA, 2010. ACM.
24. G. Lucassen, F. Dalpiaz, J. van der Werf, and S. Brinkkemper. Improving agile requirements: the quality user story framework and tool. *Requirements Engineering*, 21(3):383–403, 2016.
25. J. Machado, S. Isotani, A. Barbosa, J. Bandeira, W. Alcantara, I. Bittencourt, and E. Barbosa. Ontosoft process: Towards an agile process for ontology-based software. In *49th Hawaii International Conference on System Sciences (HICSS)*, pages 5813–5822. IEEE, 2016.
26. A. Messina, F. Fiore, M. Ruggiero, P. Ciancarini, and D. Russo. A new agile paradigm for mission-critical software development. *The Journal of Defense Software Engineering (CrossTalk)*, (6):25–30, 2016.
27. R. R. Nelson and S. G. Winter. *An evolutionary theory of economic change*. Harvard University Press, 2009.
28. C. of the Joint Chiefs of Staff. Interoperability and supportability of information technology and national security systems. Technical Report CJCSI 6212.01E, Department of Defence (United States of America), dec 2008.
29. M. Polanyi. The tacit dimension, 1966.
30. J. F. Porac and H. Thomas. Taxonomic mental models in competitor definition. *The Academy of Management Review*, 15(2):224–240, 1990.
31. R. S. Pressman. *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.
32. D. Reifer. Industry software cost, quality and productivity benchmarks. *The DoD SoftwareTech News*, 7(2):3–19, 2004.

33. K. S. Rubin. *Essential Scrum: a practical guide to the most popular agile process*. Addison-Wesley, 2012.
34. D. Russo. Benefits of open source software in defense environments. In *Proceedings of 4th International Conference in Software Engineering for Defence Applications*, volume 422, pages 123–131. Springer, Advances in Intelligent Systems and Computing, 2016.
35. D. Russo and P. Ciancarini. A Proposal for an Antifragile Software Manifesto. *Procedia Computer Science*, 83:982–987, 2016. The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016).
36. D. Russo and P. Ciancarini. Towards Antifragile Architectures. *Procedia Computer Science*, 109:929–934, 2017. The 8th International Conference on Ambient Systems, Networks and Technologies (ANT 2017).
37. D. Russo, P. Ciancarini, T. Falasconi, and M. Tomasi. Software quality concerns in the italian bank sector: the emergence of a meta-quality dimension. In *Proc. 39th Int. Conf. on Software Engineering*, ICSE '17, pages 63–72. IEEE, 2017.
38. D. Russo, V. Lomonaco, and P. Ciancarini. A machine learning approach for continuous development. In *Proceedings of 5th International Conference in Software Engineering for Defence Applications*. Springer, Advances in Intelligent Systems and Computing, 2017.
39. K. Schwaber. *Agile Project Management with Scrum*. Microsoft Press, 2004.
40. L. Sterling, P. Ciancarini, and T. Turnidge. On the animation of not executable specifications by prolog. *International Journal of Software Engineering and Knowledge Engineering*, 6(1):63–87, 1996.
41. J. Sutherland. Agile can scale: Inventing and reinventing Scrum in five companies. *Cutter IT Journal*, 14(12):5–11, 2001.
42. C. Thamrongchote and W. Vatanawood. Business process ontology for defining user story. In *IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, pages 1–4, Japan, 2016.
43. M. Uschold and M. Gruninger. Ontologies: Principles, methods and applications. *The Knowledge Engineering Review*, 11(02):93–136, 1996.
44. VersionOne. 11th annual state of agile survey, 2016.