Alignment of Requirements and Testing in Agile: a Collaborative Academia-industry Experience

Alessio Bucaioni*, Antonio Cicchetti*, Federico Ciccozzi*, Manvisha Kodali†, and Mikael Sjödin*
* Mälardalen University, Västerås, Sweden
† Westermo, Västerås, Sweden
*{alessio.bucaioni, antonio.cicchetti, federico.ciccozzi, mikael.sjodin}@mdh.se
†{manvisha.kodali}@westermo.se

***Abstract*—Agile development aims at switching the focus from processes to individuals interactions, from heavy to minimalistic documentation, from contract negotiation and detailed plans to customer collaboration and prompt reaction to changes. With these premises, requirements traceability may appear to be an overly exigent activity, with little or no return-of-investment. However, since testing remains crucial even when going agile, the developers need to identify at a glance what to test and how to test it. That is why, even though requirements traceability has historically faced a firm resistance from the agile community, it can provide several benefits when promoting precise alignment of requirements with testing. This paper reports on our experience in promoting traceability of requirements and testing in the data communications for mission-critical systems in an academia-industry collaborative project. We define a semi-automated requirements tracing mechanism which coordinates four traceability techniques. We comment on the application of the proposed solution to an industrial project aiming at enhancing the existing Virtual Router Redundancy Protocol by adding Simple Network Management Protocol support.**

***Keywords*—testing; agile development; requirements traceability; academia-industry collaboration**

I. Introduction

Software is paramount in modern society as our lives are affected by, and in many cases rely on, it. Traditional and documentation-driven software development processes start with the elicitation and specification of functional and non-functional requirements. Thereafter, a high-level design is defined in terms of the architectural description of the software to be developed [1]. While the implementation is inspired by documented requirements and high-level design, the testing is driven by requirements. It is unquestionable that, within traditional development processes, requirements affect most of the development phases. Therefore, the ability to navigate back and forth from the requirements to any other development artefact is considered pivotal [2]. Requirements Traceability (RT) commonly refers to the ability to follow the life of a requirement, forwards and backwards, within the whole software life-cycle, that is to say from requirement to test case, across design and implementation artefacts [2]. From the early 90s, elicitation and documentation of requirements and development artefacts started to incarnate strife and frustration for practitioners. On the one hand, technology and industry changed at an extraordinary pace making hard for the stakeholders to definitely identify requirements. On the other hand, stakeholders' expectations on the final software product increased. This triggered the need for new development processes which could be less bounded to heavy documentation and bootstrap phases: it was the dawn of Agile development [3]. For "agility", in this context, it is meant the ability to switch the focus from processes, comprehensive (but heavy) documentation, contract negotiation and detailed plans, to individuals and dense interaction among them, customer collaboration and prompt reaction to changes [4]. In this landscape, requirements engineering and especially RT may seem to be superfluous and burdensome, and to return little value to the development. However, when going agile, testing cannot be disregarded, but rather occupies a crucial position. In order to nimbly test a software system during its agile development, it is crucial to identify at a glance i) what to test (e.g., requirements, code and development artefacts) and ii) how to test it (i.e., test cases). Although agile methods focus on face-to-face communication and continuous delivery, RT can come in handy for achieving agile testing thanks to precise mechanisms for relating requirements to test cases. The alignment between requirements and testing in terms of traceability can indeed provide several benefits among which progress checking, customer focus, resource savings, knowledge sharing and improved software quality [5] [6] [7]. However, there is evidence showing that RT has historically faced a firm resistance from the agile community since it is regarded as an activity that may introduce excessive overhead, hence standing in the way of the agile creed [8].

This paper reports on an academia-industry collaborative experience in promoting traceability of requirements and testing in data communications for mission-critical systems. We provide a semi-automated requirements tracing mechanism which coordinates four traceability techniques and apply it to an industrial project aiming at enhancing the existing Virtual Router Redundancy Protocol (VRRP[1]) by adding Simple Network Management Protocol (SNMP[2]) support. The mechanism can be considered as a necessary trade-off between a manual and an automated solution. In fact, on the one hand the overhead of a manual mechanism would not be compatible with the "agility" of the project. On the other hand, a fully automated mechanism would require a remarkably long bootstrap phase for being implemented in addition to an extensive and precise knowledge of requirements. The scientific contributions of this paper are:

- the definition of a lightweight approach for requirements

[1]https://en.wikipedia.org/wiki/Virtual_Router_Redundancy_Protocol.
[2]https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol.

tracing in agile software projects, and
- the reported collaborative academia-industry experience .

The remainder of the paper is structured as follows. Section II presents a comparison between existing related approaches documented in the literature and our solution. Section III describes the proposed solution in all its constituents. Section IV shows the application of the proposed solution in an academia-industry collaborative Scrum project. Section V and Section VI discuss the benefits and limitations of our solution and conclude the paper, respectively.

II. Related Work

This work deals with the problem of keeping track of requirements coverage and satisfaction through test cases in agile development. Generally, such a problem is known as alignment of requirements and verification & validation, and RT is considered as a possible solution [8], [9]. In general, RT is distinguished between horizontal, when tracing the evolution of requirements during the development process, and vertical, when considering the life of a requirement in terms of related artefacts, notably design decisions, source code, tests, and documentation [10]. In this respect, our solution supports vertical RT.

In traditional software development processes, requirements engineering encompasses a set of well defined preliminary phases dealing with analysis, planning, and documentation. Requirements are supposed to be largely known, and hence RT can be tackled as pertaining to requirements management stages. This allows, for instance, to configure and exploit traceability tools, or to adopt model-based/formal techniques for gaining some form of automation [8]. The work in [8] gives a broad overview of both solutions and challenges in realising the alignment between requirements and verification & validation in general, and by exploiting RT techniques in particular. As stated before in this paper, the basic principles of agile make traditional alignment solutions not suitable: requirements are very often only partially known, they evolve rapidly during the development process, and there is no space for preliminary analysis nor bootstrap activities. It is worth noting that, even if requirements are not subject to remarkable evolutionary pressure in traditional processes, RT is very often perceived as a time-consuming activity with poor ROI if not adequately supported [11], [12]. Notably, traces can be affected by “decay”, that is the progressive loss of consistency between requirements and linked artefacts due to maintenance activities not reflected in the current traceability information [12]. This issue is exacerbated in agile processes, such that the lack of traceability is conceived as an intrinsic problem for this kind of development [13], [14]. We propose to alleviate the concerns related to requirements evolution by means of a semi-automated approach, which represents a trade-off between reliability of manual tracing and reduction of tedious and error-prone tasks through automation.

The recent work in [15] provides an extensive illustration of traceability in agile processes, coming from both academia and industry. Interestingly, several empirical observations they make and conclusions they draw are consistent with our industrial settings and suitability of the adopted solution. More specifically, when considering the types of relevant traceability, most companies chose “rationale” and “contribute”, i.e. two vertical traces. In this respect, in our work requirements are the rationale to the creation of test cases. When it comes to traceability mechanisms, most companies exploit product/sprint backlogs and in some cases spreadsheets for implementing RT. It is not surprising that observations reveal a decrease of satisfaction proportional to the size of the company [15]. In other words, traceability-matrix-like approaches, being largely manual, add too much overhead to be compatible with the agile vision. This aspect is also confirmed by checking the most relevant challenges expressed by interviewees: difficulty in identifying proper traceability links, difficulty in motivating the personnel to keep traces, and low ROI. In our scenario, tracing activity is simplified by the fact that it only addresses links between requirements and test cases. The solution is amalgamated with current used tools (both for requirements definition and test case specification) thus avoiding the need of additional knowledge. Moreover, the automated inspection can be a useful tool even for verifying the progress of the sprints. This last aspect is again confirmed by the conclusions made in [15] when discussing the potential benefits of RT mechanisms. Some approaches dealing with RT in agile development processes exist; notably, in [13] the authors define a set of requirements for RT support in agile and introduce a methodology based on Test-Driven Development and customisable roles (to provide multiple trace links semantics). The main difference with our proposal is that we were seeking a lightweight solution, requiring a negligible bootstrap effort in terms of implementation and adoption, due to the reasons mentioned above. A methodology forcing the adoption of test-driven development, or the preliminary definition of roles and links semantics project-by-project would have been considered as too much overhead and hence not acceptable. A heterogeneous solution for RT is suggested in [11]. In particular, the authors propose to exploit requirements layering with respect to different development process stages, project epics, and roles involved in their management. Then, they assign a different RT approach for each layer. Our approach shares with [11] the vision of adopting heterogenous strategies for supporting RT. However, we do not exploit requirements layering, mainly due to the volatility of project scenarios and the bootstrap costs of configuring RT support for each new project.

III. Proposed Solution

In this section we describe a mechanism introducing RT in the agile system development life cycle (SDLC) [16] for enhancing the alignment between requirements (REQs) and testing in terms of test cases (TCs). The intent is to show how the software development can be improved in terms of, e.g., resource savings, progress checking and improved software quality, while preserving its “agility”. The proposed solution leverages four existing traceability techniques:

- Tagging. It is a technique which assigns a keyword to a piece of information for tracing the information throughout the development life-cycle [17]. The proposed solution uses tagging for assigning unique identifiers (IDs) to REQs and TCs and for tagging the TCs implementation with the IDs of REQs and TCs.
- Information retrieval. It is a technique which establishes traceability links among artefacts based on the similarity

between their contained information [18]. The proposed solution uses information retrieval for retrieving all the elicited REQs and the TCs along with their relationships.
- Integrating documents. It is a technique which merges different information in a single document [19]. the proposed solution uses integrating documents for creating explicitly correspondences between REQs and TCs.
- Requirements traceability matrix (RTM). It is a technique which uses a two-dimensional grid for mapping REQs to other development artefacts [20]. The proposed solution uses RTM for collecting and displaying the traces information among REQs and TCs. In this respect, we extend the base RTM semantics with values for describing the relationships holding among REQs and TCs.

To the best of our knowledge, the proposed solution represents the first attempt in combining a set of generic RT techniques with the aim of providing an extensive RT mechanisms within the agile SDLC. Figure 1 shows the proposed solution in relation to the main phases of the agile SDLC as defined in [16]. The proposed solution comprises six tasks:

1) Assign IDs to REQs;
2) Assign IDs to TCs;
3) Match REQs and TCs;
4) Tag TCs implementation with IDs of REQs and TCs;
5) Build the RTM;
6) Populate the RTM.

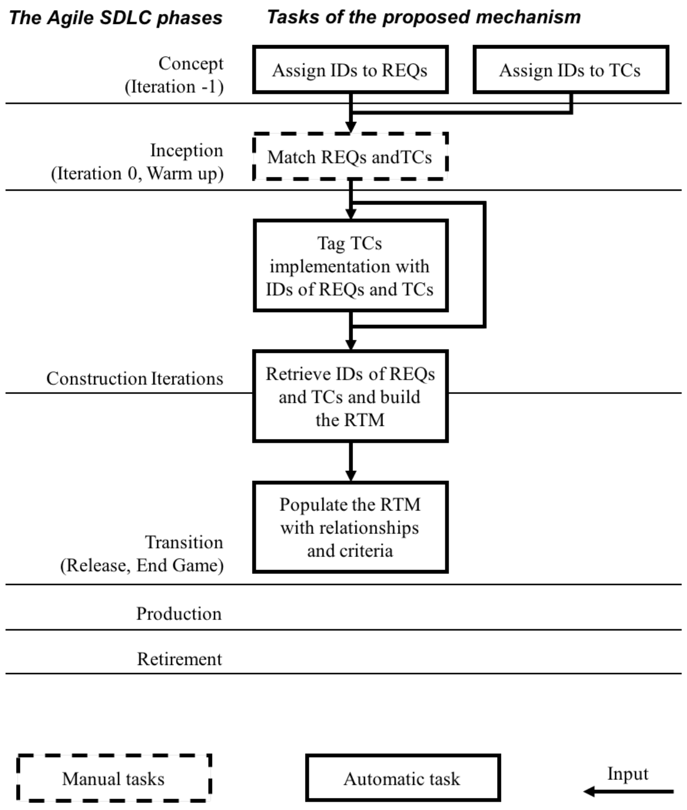
The Agile SDLC phases
Tasks of the proposed mechanism
Concept (Iteration -1)
Assign IDs to REQs
Assign IDs to TCs
Inception (Iteration 0, Warm up)
Match REQs andTCs
Tag TCs implementation with IDs of REQs and TCs
Construction Iterations
Retrieve IDs of REQs and TCs and build the RTM
Populate the RTM with relationships and criteria
Transition (Release, End Game)
Production
Retirement
Manual tasks
Automatic task
Input

Fig. 1: proposed solution

1. Assign IDs to REQs. The goal of this task is to associate IDs to the identified REQs. The input of this task is the list of the identified REQs. The output is a list L_{REQ} = $\{r_1, ..., r_n\}$ of records r_i. r_i is represented by the pair $<id, desc>$, where id is the REQ ID and $desc$ is the REQ description. This task should be performed during the inception phase (see phases in Figure 1) and it can be automated by means of scripts.

2. Assign IDs to TCs. The goal of this task is to associate IDs to the planned TCs. The input of this task is the list of the planned TCs. The output is a list L_{TC} = $\{t_1, ...,t_m\}$ of records t_i. t_i is represented by the pair $<id, desc>$ where id is the TC ID and $desc$ is the TC description. This task should be performed during the inception phase and it can be automated by means of scripts.

3. Match REQs and TCs. This task should be performed prior the development occurring in each construction iteration phase and aims at formalising the relationships between REQs and TCs, which are typically identified during the concept phase (as TCs originate from the user stories derived from the REQs). The input of this tasks are the L_{REQ} and L_{TC} lists produced by the previous tasks. The output is a list $L_{REQ/TC}$ = $\{r/t_1, ..., r/t_n\}$ of records r/t_i, where r/t_i is represented by the pair $<id, L_{TC_{ID}}>$ where id is the REQ ID and $L_{TC_{ID}}$ is the list of the IDs of the TCs verifying it. .

4. Tag the TCs implementation with IDs of REQs and TCs. In the previous task, trace links between REQs and TCs are established. The goal of this task is to extend this links to the TCs implementation. To this end, each TCs implementation is tagged with the corresponding TC IDs and with the IDs of the REQs it verifies. The input of this task is the $L_{REQ/TC}$ list produced in the previous task. The output of this task is the TC implementation tagged with IDs of REQs and TCs. This task should be performed during the TCs design occurring in each construction iteration phase and it can be fully automated by scripts. It is important to note that in some development iterations TCs might not be defined prior the construction iteration or transition phases. When this happens, the task 2 and 3 of the proposed mechanism can not be performed. In this scenario, this task provides the engineer with a further possibility for defining trace links between REQs and TCs, thus it enables more agility in the development process.

5. Retrieve IDs of REQs and TCs and build the RTM. The goal of this task is two-fold. On the one hand, it builds the RTM. On the other hand, it verifies that all the trace links between REQs and TCs have been considered when tagging the TCs implementation. To this end, the IDs of REQs and TCs should be extracted from the TCs implementation and saved into a temporary list. Such a list is compared to the $L_{REQs/TCs}$ list produced in the task three with the aim of identifying and marking possible differences. Eventually, the two lists are merged and the RTM should be constructed accordingly. This task should be performed before the testing occurring in each construction iteration and transition phases and it can be fully automated by scripts.

6. Populate the RTM with relations and criteria. The goal of this task is to populate the RTM with the test results. The input of this task is the RTM produced in the previous task. The output of this task is the populated RTM.

More specifically, each entry e_i of the RTM is represented by a pair $<relation, status>$ where $relation$ describes the relationship between REQs and TCs while $status$ describes the TC result. $relation$ can have a value in the set $\{planned$, $added$, $deleted\}$, obtained from the comparison done in the previous task. $planned$ indicates that the REQ is verified by the TC as planned during the tasks 1 and 2. $added$ indicates that the decision to verify the REQ with the TC has been taken during the task 4. $deleted$ indicates that the REQ has been deleted during the development process. $Status$ can have two base values, $passed$ and $failed$. However, depending on the adopted pass/fail criteria it can be extended with further values such as, e.g, blocked and not-run. This task should be performed after the testing occurring in the transition phase and it can be fully automated by scripts.

IV. Applying the Solution to an Academia-industry collaborative Scrum Project

In this section we describe the application of our mechanism to the VRRP-MIB project. The VRRP-MIB project is an industrial project of Westermo[3] aiming at enhancing the existing VRRP protocol by adding SNMP support.Full It is defined as a SCRUM project consisting of 34 REQs and 3 TCs grouping 22 tests. The project was run for six months by a team composed of 5 engineers from Westermo for a total of 253 man-hours. The elicitation phase required 10 man-hours. The development phase required 160 man-hours. The testing phase required 80 man-hours. Eventually, 2 man-hours were spent for checking that all the elicited REQs were tested. In the following, due to its verbosity and complexity, we will discuss a simplified version of the VRRP-MIB project corresponding to the first Scrum sprint and consisting of 10 REQs and 2 TCs.

REQs		
1	*Name*	vrrpOperVrId
	Description	Full OID description. The system should provide VRID as index into table
	Constraints	None
	Qualification	The test should verify that the OID within table entries match the VRID of a created instance (created via CLI)
2	*Name*	vrrpOperVirtualMacAdd
	Description	Full OID description (Our VRRP implementation VMAC meets VRRP standard)
	Constraints	None
	Qualification	Test should verify that VMAC matches VRRPv2 specication (i.e., VRRP/IETF-prex + VRID)

TABLE I: Example of REQs for the VRRP-MIB project

TCs		
1	*Name*	verifyOperState
	Description	Verify that state changes the all supported operstates are correct
2	*Name*	verifyVrrpAdvancedVrrpSetup
	Description	Verify that the values updated from VRRP are correct while having two VRRP instances and merging with a third instance

TABLE II: Example of TCs for the VRRP-MIB project

Table I reports an example of two REQs considered in the first sprint and how they were described within Westermo. In particular, each REQ was described by means of a *name*, a *description*, a *qualification* and some *constraints*. The qualification gives basic information for testing while the constraints specify possible relationships with other REQs. Similarly, Table II shows an example of two TCs involved in the first sprint and how they were described within Westermo. Each TC was described by means of a *name* and a *description*. Within Westermo, REQs and TCs are represented and saved in separate files and organised in folders which are stored using the version control mechanism GIT[4].

According to the first two tasks of the proposed mechanism, we assigned IDs to REQs and TCs. These assignments were automated by means of Python scripts. Table III and IV show the resulting descriptions containing IDs for REQs and TCs, respectively.

REQs		
1	*Name*	vrrpOperVrId
	ID	r.weos.snmpmib.vrrpv2.ss1
	Description	Full OID description.The system should provide VRID as index into table
	Constraints	None
	Qualification	The test should verify that the OID within table entries match the VRID of a created instance (created via CLI)
2	*Name*	vrrpOperVirtualMacAdd
	ID	r.weos.snmpmib.vrrpv2.ss2
	Description	Full OID description (Our VRRP implementation VMAC meets VRRP standard)
	Constraints	None
	Qualification	Test should verify that VMAC matches VRRPv2 specication (i.e., VRRP/IETF-prex + VRID)

TABLE III: Example of REQs description with IDs

TCs		
1	*Name*	verifyOperState
	ID	t.weos.snmpmib.vrrpv2.ostate
	Description	Verify that state changes the all supported operstates are correct
2	*Name*	verifyVrrpAdvancedVrrpSetup
	ID	t.weos.snmpmib.vrrpv2.adv-test
	Description	Verify that the values updated from VRRP are correct while having two VRRP instances and merging with a third instance

TABLE IV: Example of TCs description with IDs

Table V shows the tabular representation of the formalised relationships between REQs and TCs as the result of the application of the third task of our proposed mechanism.

REQs	TCs
r.weos.snmpmib.vrrpv2.ss1	t.weos.snmpmib.vrrpv2.adv-test
r.weos.snmpmib.vrrpv2.ss2	t.weos.snmpmib.vrrpv2.adv-test
r.weos.snmpmib.vrrpv2.ss3	t.weos.snmpmib.vrrpv2.adv-test
r.weos.snmpmib.vrrpv2.ss4	t.weos.snmpmib.vrrpv2.operState
r.weos.snmpmib.vrrpv2.ss5	t.weos.snmpmib.vrrpv2.operState
r.weos.snmpmib.vrrpv2.ss6	t.weos.snmpmib.vrrpv2.operState
r.weos.snmpmib.vrrpv2.ss7	t.weos.snmpmib.vrrpv2.adv-test
r.weos.snmpmib.vrrpv2.ss8	t.weos.snmpmib.vrrpv2.adv-test
r.weos.snmpmib.vrrpv2.ss9	t.weos.snmpmib.vrrpv2.adv-test
r.weos.snmpmib.vrrpv2.ss10	t.weos.snmpmib.vrrpv2.adv-test

TABLE V: Correspondence between REQs and TCs by IDs

In particular, the TC *t.weos.snmpmib.vrrpv2.adv-test* tests the following REQs:

- *r.weos.snmpmib.vrrpv2.ss1*
- *r.weos.snmpmib.vrrpv2.ss2*
- *r.weos.snmpmib.vrrpv2.ss3*
- *r.weos.snmpmib.vrrpv2.ss7*

[3]http://www.westermo.com

[4]https://git-scm.com

- *r.weos.snmpmib.vrrpv2.ss8*
- *r.weos.snmpmib.vrrpv2.dss9*
- *r.weos.snmpmib.vrrpv2.ss10*

While the TC *t.weos.snmpmib.vrrpv2.operState* tests the following REQs:

- *r.weos.snmpmib.vrrpv2.ss4*
- *r.weos.snmpmib.vrrpv2.ss5*
- *r.weos.snmpmib.vrrpv2.ss6*

As aforementioned, the formalised relationships between REQs and TCs are stored in a textual file which is, in turn, and stored in the corresponding GIT folder. According to the fourth task of our proposed alignment mechanism, the TCs implementation must be tagged with the IDs of TCs and REQs. Within Westermo, TCs are implemented by means of Python scripts[5]. Those, were extended with two variables *testID* and *references*. *testID* is a string containing the ID of the TC whereas *references* is an array of strings containing the IDs of the corresponding REQs. Such an extension was automatically performed by means of Python scripts. Moreover, we used Python scripts for automating the creation and the filling of the RTM, too. For the selected sprint, we performed unit and integration tests as well as impact analysis. Unit tests were run manually during the day, while integration tests were automatically run for nightly builds. Impact analysis was performed at the end of the test activities by counting the passed/failed tests. After the testing activities, a Python script collected the test results together with the information regarding the relationships among REQs and TCs.

Table VI shows the final RTM filled with REQs, TCs, results and relationships. Within the VRRP-MIB project, Therefore, it was not necessary to extend the value of the variable status, as discussed in Section III. During the testing activities, the scrum team decided to exercise the *r.weos.snmpmib.vrrpv2.ss1* REQ with the *t.weos.snmpmib.vrrpv2.operState* TC too, despite during the sprint planning event the *r.weos.snmpmib.vrrpv2.ss1* REQ was associated with the *t.weos.snmpmib.vrrpv2.adv-test* TC only. This change was captured by the Python script which marked the relationship between *r.weos.snmpmib.vrrpv2.ss1* REQ and *t.weos.snmpmib.vrrpv2.operState* TC with the *added* value.

V. Discussion

Alignment between requirements and testing brings multiple benefits such as progress checking. This is especially true when such an alignment is achieved by means of semi-automatic mechanisms as the one presented in this paper. The proposed solution leverages the RTM and its newly extended semantics as the core artefacts for visualising traces between requirements and test cases as well as for checking the development progress. In fact, the information about correspondences between requirements, test cases and testing results allows the engineer to grasp the progress of the project at a glance without the use of additional tools as shown in the VRRP-MIB project. With respect to resource saving, in the VRRP-MIB project, we were able to achieve a 10% reduction of the required man-hours. In fact, the execution of a similar project required 253 man-hours, whereas within the VRRP-MIB project we were able to shorten the testing activities by 23 man-hours. Moreover, we did not need to spend additional 2 hours for checking that the REQs were tested. Agile processes focus on the importance of customer involvement during the whole development process. Within the VRRP-MIB project, we observed an improved customer experience as the customer's decisions, and their fulfilment, could be easily recorded and trace throughout the development process. Moreover, we observed that after the first sprint, the engineers involved in the project started to use the RTM as an effective progress tracking tool. Within the VRRP-MIB project, we did not observe any variation in the defects rate of the software. In fact, the proposed mechanism does not aim at improving the quality of testing. However, it improves the quality of the software development by supporting a more rational and traceable process. This, in turn, affects the quality of the software product during the production and retirement phases. Agile processes tend to cut down detailed documentation in order to minimise the overhead introduced by activities not related to the implementation. In this context, the artefacts produced within the proposed solution could serve as a base for the automatic generation of documentation. Doing so, documentation activities would not represent an unbearable burden and could seamlessly be included in the development without jeopardising agility. Concerning possible limitations of the mechanism, it can be argued that assigning identifiers to test cases, as a first step, could be unfeasible as test cases might be defined after the implementation phase. Although this could be a valid concern, it should be noted that the proposed mechanism allows the developer to enter the identifiers at later stages, when test cases are implemented, and the RTM is still generated, as described in Section IV. Our mechanism does not provide any support for assessing testing effectiveness and it at its best when applied to medium-small software projects. Although the work done in the collaborative project shows that the mechanism does not introduce any significant development and managerial overhead, it might require an initial effort in terms of scripting activities to provide automation, as discusses in Section IV. While this could be a valid concern, it should be noted that this is a one-time effort affecting only the first application of the mechanism as the scripts could be reused for the following projects, unless technology shifts.

The work reported in this paper was carried out in a tight collaborative fashion between academics and industrial practitioners. The project ran for six months with monthly meetings. Already from the beginning, both sides were actively involved in the definition of project goals, milestones and timeframe. Tight collaboration fostered new ideas and challenged our initial hypotheses making us to achieve a solution which was able to decrease the effort required for testing activities in a real world scenario. In this respect, the agile development process has to be considered as a key enabler of the collaboration, as it disclosed the opportunity of enacting quick develop-and-check iterations of the proposed mechanisms.

VI. Conclusion

The Agile vision is to avoid monolithic development steps in order to promote flexibility. In this respect, traceability

[5]Due to confidentiality we can not show the Python scripts implementing the TCs.

		TCs	
		t.weos.snmpmib.vrrpv2.adv-test	*t.weos.snmpmib.vrrpv2.operState*
REQs	*r.weos.snmpmib.vrrpv2.ss1*	(planned,passed)	(added,passed)
	r.weos.snmpmib.vrrpv2.ss2	(planned,passed)	
	r.weos.snmpmib.vrrpv2.ss3	(planned,passed)	
	r.weos.snmpmib.vrrpv2.ss4		(planned,passed)
	r.weos.snmpmib.vrrpv2.ss5		(planned,passed)
	r.weos.snmpmib.vrrpv2.ss6		(planned,passed)
	r.weos.snmpmib.vrrpv2.ss7	(planned,passed)	
	r.weos.snmpmib.vrrpv2.ss8	(planned,passed)	
	r.weos.snmpmib.vrrpv2.ss9	(planned,passed)	
	r.weos.snmpmib.vrrpv2.ss10	(planned,passed)	

TABLE VI: RTM with REQs and TCs relations

between requirements and test cases has been traditionally perceived as an accessory and time-consuming activity with low return-of-investment. In this paper we illustrated the activities aimed at introducing a traceability mechanism in an industrial agile development context. The proposed solution is necessarily pragmatic, as resulting from trading off fully manual and fully automated mechanisms. In fact, both of them would introduce excessive overhead and hence violate the fundaments of agile. The observations we conducted on a concrete project show encouraging results in terms of effectiveness. It is worth noting that the proposed technique does not assess how good are the tests, rather it keeps track of how requirements have been tested. As discussed throughout the work, these traces can provide useful feedbacks on how the verification & validation has been operated and its progress status in the sprints.

As future work, we plan to investigate additional automation to provide basic code documentation as derivable from the traceability between requirements and test cases. Moreover, we intend to study the application of our tracing technique in other suitable agile development scenarios, in order to better validate the proposed approach.

Acknowledgment

The authors would like to thank Peter Johansson, Per Erik Strandberg, Jonas Nylander and Jon-Olov Vatn from Westermo Research and Development for their support through the VRRP-MIB project.

References

[1] Ivar Jacobson, Grady Booch, James Rumbaugh, James Rumbaugh, and Grady Booch. *The unified software development process*, volume 1. Addison-wesley Reading, 1999.

[2] Orlena CZ Gotel and Anthony CW Finkelstein. An analysis of the requirements traceability problem. In *Requirements Engineering, 1994., Proceedings of the First International Conference on*, pages 94–101. IEEE, 1994.

[3] Andrea De Lucia and Abdallah Qusef. Requirements engineering in agile software development. *Journal of Emerging Technologies in Web Intelligence*, 2(3):212–220, 2010.

[4] Martin Fowler and Jim Highsmith. The agile manifesto. *Software Development*, 9(8):28–35, 2001.

[5] Christopher Lee, Luigi Guadagno, and Xiaoping Jia. An agile approach to capturing requirements and traceability. In *Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE 2003). Citeseer*, 2003.

[6] Veerapaneni Esther Jyothi and K Nageswara Rao. Effective implementation of agile practices. *IJACSA) International Journal of Advanced Computer Science and Applications*, 2(3), 2011.

[7] Jane Huffman Hayes, Alex Dekhtyar, and James Osborne. Improving requirements tracing via information retrieval. In *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*, pages 138–147. IEEE, 2003.

[8] Elizabeth Bjarnason, Per Runeson, Markus Borg, Michael Unterkalmsteiner, Emelie Engström, Björn Regnell, Giedre Sabaliauskaite, Annabella Loconsole, Tony Gorschek, and Robert Feldt. Challenges and practices in aligning requirements with verification and validation: a case study of six companies. *Empirical Software Engineering*, 19(6):1809–1855, 2014.

[9] E.J. Uusitalo, M. Komssi, M. Kauppinen, and A.M. Davis. Linking requirements and testing in practice. In *International Requirements Engineering, 2008. RE '08. 16th IEEE*, pages 265–270, Sept 2008.

[10] Shari Lawrence Pfleeger and S.A. Bohner. A framework for software maintenance metrics. In *Software Maintenance, 1990, Proceedings., Conference on*, pages 320–327, Nov 1990.

[11] J. Cleland-Huang, G. Zemont, and W. Lukasik. A heterogeneous solution for improving the return on investment of requirements traceability. In *Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International*, pages 230–239, Sept 2004.

[12] G. Regan, F. McCaffery, K. McDaid, and D. Flood. The barriers to traceability and their potential solutions: Towards a reference framework. In *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*, pages 319–322, Sept 2012.

[13] Angelina Espinoza and Juan Garbajosa. A study to support agile methods more effectively through traceability. *Innovations in Systems and Software Engineering*, 7(1):53–69, 2011.

[14] Irum Inayat, Lauriane Moraes, Maya Daneva, and Siti Salwah Salim. A reflection on agile requirements engineering: Solutions brought and challenges posed. In *Scientific Workshop Proceedings of the XP2015*, XP '15 workshops, pages 6:1–6:7, New York, NY, USA, 2015. ACM.

[15] Vuong Hoang Duc. Traceability in agile software projects, 2013. Master's thesis at the University of Gothenburg, http://hdl.handle.net/2077/38990.

[16] Scott W Ambler. The agile system development life cycle (sdlc). *Ambysoft Inc.,[Online]. Available: http://www. ambysoft. com/essays/agileLifecycle. html.[Accessed 14 Maio 2014]*, 2009.

[17] Marcus Jakobsson. *Implementing traceability in agile software development*. Department of Computer Science, Lund University, 2009.

[18] Jane Huffman Hayes, Alex Dekhtyar, and James Osborne. Improving requirements tracing via information retrieval. In *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*, pages 138–147. IEEE, 2003.

[19] Madhuri Kolla and Mounika Banka. Merging functional requirements with test cases. 2015.

[20] Gunavathi Duraisamy and Rodziah Atan. Requirement traceability matrix through documentation for scrum methodology. *Journal of Theoretical & Applied Information Technology*, 52(2):154–159, 2013.