Trace++: A Traceability Approach to Support Transitioning to Agile Software Engineering

Felipe Furtado[1, 2], Andrea Zisman[1]
[1]*Computing Department, The Open University, Milton Keynes, UK*
[2]*Educational Department, CESAR (Recife Center for Advanced Studies and Systems), PE, Brazil*
furtado.fs@gmail.com, andrea.zisman@open.ac.uk

***Abstract*—Agile methodologies have been introduced as an alternative to traditional software engineering methodologies. However, despite the advantages of using agile methodologies, the transition between traditional and agile methodologies is not an easy task. There are several problems associated with the use of agile methodologies. Examples of these problems are related to (i) lack of metrics to measure the amount of rework that occurs per sprint, (ii) interruption of a project after several iterations, (iii) changes in the requirements, (iv) lack of documentation, and (v) lack of management control. In this paper we present *Trace++*, a traceability technique that extends traditional traceability relationships with extra information in order to support the transition between traditional and agile software development. The use of *Trace++* has been evaluated in two real projects of different software development companies to measure the benefits of using *Trace++* to support agile software development.**

***Index Terms*—Traceability, agile methods, hybrid process.**

I. Introduction

In the last twenty years, several agile methodologies have been proposed to support software development [3][8][12] [15][20][28][32][36]. Agile methodologies bring several advantages to the software development life-cycle including, but not limited to, lightweight development processes, small number of documents, frequent deliverables, customer satisfaction, and close communication among stakeholders.

However, despite the advantages of agile methodologies, the transition between 'traditional' to 'agile' methodologies in software development organisations is not an easy task. In his paper, we use the term 'traditional software engineering' to refer to methodologies that place more emphasis on processes, tools, contracts, and plans. We consider two traditional paradigms: Unified Process (UP) [27] and Project Management Body of Knowledge (PMBOK) [33].

Several surveys have been presented in order to analyse the advantages and challenges with agile methodologies[1][2][14] [29][39][40]. As outlined in [14], 64% of the 200 industrial participants in this survey found the transition to agile methodologies confusing, hard, and slow. In this survey, the participants stated that understanding the necessary amount of rework to be executed is essential for the success and overall cost of using agile methodologies. Other identified challenges in agile adoption were concerned with the lack of team alignment, documentation, and focus; constant changes in the development cycle; and cultural acceptance. In [1][14][40], the participants pointed out issues of the agile methodology related to communication problems, loss of management control, ability to scale agile, and regulatory compliance.

Another problem that has been flagged is related to the adoption of agile management practices [1]. For example, as outlined in [4],"*in the development of large systems, the 'just enough' documentation goes beyond the traditional set recommended by the agile methods, due to the diversity of elements to be considered, as for instance geographic distribution of the teams, necessity to comply with industry regulations, strict IT governance programs, integration of the system being developed with others, or even the presence of not-so-agile people in the teams*". According to [6], agile methodologies are well known for early and frequently releases. However, in some cases agile practicioners are not aware of how changes in functional requirement may affect non-functional requirements, and could cause breaches of security and performance in a system.

In this paper we present *Trace++*, a traceability approach to assist with the transition from traditional to agile methodologies. Traceability of software systems has been recognized as an important activity in software system development [10][11][24][38]. Traceability relations can support several software development activities such as evolution, reuse, validation, rationale, understanding, and change impact analysis. Traceability relations can improve the quality of the products being developed and reduce time and cost of development.

The use of traceability techniques in agile projects has been advocated in [5][9][16][41]. In agile projects traceability can help with change impact analysis, product conformance, process compliance, project accountability, baseline reproducibility, and organisational learning [9]. In some cases, traceability is seen as a heavy process by agile developers [4][5].

The work presented in this paper complements the work on traceability for agile projects and proposes an extension of traceability relations to represent extra information in order to assist with the transition from traditional to agile methodologies. More specifically, we concentrate on four main problems related to the adoption of agile methodologies and show how *Trace++* could assist with these problems. The work has been developed based on observations and analysis of some real world agile projects. We propose the necessary information to be represented in traceability relations. The work has been evaluated in other two real world agile projects.

The remaining of this paper is structured as follows. In Section II, we describe the *Trace++* approach including the types

2332-6441/16 $31.00 © 2016 IEEE
DOI 10.1109/RE.2016.47

RE 2016, Beijing, China
Research Paper

of artifacts and traceability relations used in the work, and the different problems tackled by the approach. In Section III, we present an evaluation of the approach in two different agile projects. In Section IV, we discuss related works. Finally, in Section V, we present some conclusions and future work.

II. *TRACE++*

A. Overview of the Approach

In order to support the transition from traditional to agile methodologies, we propose to use a traceability approach called *Trace++*, between documents generated during traditional software development and agile methodologies. The traceability approach consists of extending traceability relations with extra information. The work concentrates in four different problems associated with the transition from traditional software development to agile methodologies. These problems are concerned with the (i) amount of rework that occurs per sprint, (ii) understanding of the high level scope of a project before beginning the sprints, (iii) lack of non-functional requirements documentation, and (iv) loss of management control (see Subsection II.C).

We extend the standard definition of traceability relations [10] with the notion of *information set*. The information set contains information necessary to support the four different problems of our concern. For example, in a certain type of *Trace++* relation, the information set may contain the percentage of rework in story points variations, in order to assist with the problem of absence of metrics to indicate the amount of rework that occurs in a sprint. More formally, the main elements of *Trace++* are:

- P: An agile related problem;
- Λ: Trace relations composed of a source artifact, a target artifact, a set of additional information, and a relation type;
- S = {S1, S2 ... Sj} a set of all source artifacts;
- T = {T1, T2, ..., Ti} a a set of all target artifacts;
- I = {I1, I2, ..., Im} a set of all additional information;
- Y: Relations type.

A Trace (λ) relation for a Problem P is given as:

$$\lambda(P) = \left\{ s_k, t_{k'}, \{i_{k''}\}, y_{k'''} \mid \{i_{k'''}\} \subseteq I \right\} \quad (1)$$

where:

$$s_k \in \mathbf{S} = \left\{ s_1, s_2, \ldots, s_j \right\}$$
$$t_{k'} \in \mathbf{T} = \left\{ t_1, t_2, \ldots, t_i \right\}$$
$$i_{k''} \in \mathbf{I} = \left\{ i_1, i_2, \ldots, i_m \right\} \quad (2)$$

and

$$\left\{ i_{k''} \right\} \subseteq \mathbf{I} \quad (3)$$

with the maximum number of necessary elements to provide information to each traceability relation. In some cases, a complex problem may require various traceability relations. This can be represented as the union between various traceability relations, as shown below:

$$\Lambda(P) = \lambda'(P) \cup \lambda''(P) \cup \ldots \cup \lambda^{(m)}(P) \quad (4)$$

In order to use the proposed *Trace++* approach, for each problem (P), the development team should collect the elements represented in equation (1), and use these elements to analyse the problem. Examples of the types of elements to be represented in equation (1) are shown in Subsection II.C. Guidelines for the analysis of the elements are described in Section 0.

B. Artifact Types

The aim of the approach is to provide traceability relations between artifacts generated in both traditional and agile methodologies. We analysed various types of artifacts that are generated when using different types of agile methods, and artifacts that are generated when using traditional software engineering methods, in order to identify the artifacts that are relevant to the transition between both methods.

TABLE I. AGILE ARTIFACTS

Method	Name	E	M	Author
APM	Product Vision	X	X	[21]
	Product Roadmap	X	X	
	Release Plan	X	X	
	Performance Card	X		
	Project Datasheet		X	
	Project Charter		X	
Scrum	Product Backlog	X	X	[36]
	Sprint Backlog	X	X	
	Task Board		X	
	Impediment List		X	
	Retrospective Timeline		X	
	Release Burndown/up Chart		X	
	Sprint Burndown/up Chart		X	
	Product Burndown/up Chart		X	
FDD	Feature Cards	X		[28]
	Domain Model (UML-color)	X		
	Parking Lot Chart		X	
	Feature Breakdown Structure		X	
	Development Plan		X	
DSDM	Functional Prototype	X		[15]
XP	Theme	X		[8]
	Epic	X		
	User Stories	X		
	Spyke/Research Stories	X		
	Accpetance Criteria	X		
	Theme Screening Matrix	X		
	High Level Design	X		
	Spyke Architectural	X		
	Code Refactored	X		
	Unit Tests	X		
	Acceptance Tests	X		
Lean/ Kanban	Kanban System Board		X	[32][3]
	Visual Card		X	
	Cumulative Flow Diagram		X	
Common	Kano Matrix	X		[19]
	Persona	X		[30]
	Wireframe/Mockups	X		[22]
	User Story Mapping	X		[31]
	Risk Burndown/up Chart		X	[13]
	Risk Radar Chart		X	
	Risk Backlog		X	

Table I shows the agile methods and the respective artifacts that were used in this analysis. We have classified the artifacts into two groups, based on the definition proposed in [37], namely (i) engineering-related artifacts (E) such as requirements, design, coding, and testing specifications; and (ii) management-related artifacts (M) such as project management, measurement and analysis, and management processes. These groups are important to support traceability relations in the various stages of the software development lifecycle.

The agile artifacts listed in Table I are mapped to artifacts generated during traditional software engineering development, in order to identify possible target artifacts. For this mapping, we have considered artifacts based on several criteria: (a) artifacts that belong to the Unified Process (UP) [27] or the PMBOK [33] paradigms; (b) artifacts that are related to engineering and management groups, as per the definition in [37]; and (c) artifacts that appear in all the different stages of the software development lifecycle.

C. Agile Problems

We have conducted a study involving work reported in 23 papers [6][9][16][23][41][1] and six industrial reports [1][2][14][29][39][40]. Based on this study we identified several problems and challenges that undermine adoption of agile methods. Examples of these problems and challenges are:

- Absence of the use of metrics to indicate the amount of rework that occurs in each sprint [14];
- Abandonment of the project after several iterations due to the misunderstanding on the high-level scope of the project before beginning sprints [14];
- Constant changes in requirements [14][40];
- Large projects with distributed teams [1][40];
- Lack of sufficient documentation [29][40];
- Communication failure between the various stakeholders of the project on the evolution of requirements [14][39][40];
- Loss of management control [29][39][40];
- Low quality of software maintainability in formal projects in certain industries, e.g. financial services, healthcare, telecom and government [40].

The work in this paper tackles four of these problems. We have selected problems that are often cited in the literature [6][16][26][41], and that involve tracking information between different types of artifacts generated during the software development life-cycle. In the following we describe the four problems of our concern in terms of their context, associated source and target artifacts, additional information set, and proposal to establish traceability relations. Please note that these problems are not only related to agile methodologies, but that they appear during transition from traditional software development.

Problem 1 (P1): *Absense of metrics to indicate the amount of rework that occurs in each sprint.*

Context: One of the premises of agile methods is concerned with the fact that the cost and schedule of a project are fixed, but the scope varies. This is important to provide Product Owners (PO) the flexibility to prioritize backlog items in the way that best meets their business needs, without exceeding time and cost of development [36][32]. This approach is important to produce items of big business value. For example, sometimes the PO has prioritized in the beginning of a project a set of requirements that may change along the sprint, due to market needs, but as these changes do not interfere with the schedule and cost of the project, the manager may substitute with another backlog some items that do not have contract changes. However, when there are no metrics that indicate the amount of rework caused in each sprint, multiple items of the backlog are moved to the bottom of the list as having lowest priority, and may not be implemented until the end of the project, causing customer dissatisfaction.

Source artifact: User story (US).

Target artifact: Software requirements specification, persona, wireframe, class diagram.

Additional information: Percentage of rework (story points variation), rework (business value variation), rework (new personas, wireframes and classes involved in user story);

Proposal: At the end of the execution of each sprint or at the next sprint planning meeting, the team and the Scrum Master should calculate the percentage change of story points (and/or business value) between what has been planned at the beginning of the sprint and what actually was delivered at the end of the sprint. In addition, the team members should calculate how much was spent on rework activities due to changes requested along the sprint. The same can be done with the number of new personas, wireframes, and classes identified and created along the sprint to meet business goals.

Problem 2 (P2): *Lack of understanding about the high level scope of a project before starting a sprint.*

Context: In some cases a user story is developed during a sprint, but its prioritization changes or it becomes an epic due to its size. In this case, traceability relations could help in providing more details about the user story and the scope of activities to be implemented. Moreover, these traceability relations will help with impact analysis of changes in user stories, which can cause impacts on architecture or test scenarios of the system. The lack of understanding of the scope can cause abandonment of the project after a few iterations.

Surce artifact: User story.

Target artifact: Class diagram, sequence diagram, use case diagram.

Additional information: Identification of the preceding sprints, which the user story has been implemented.

Proposal: During the stage of sprint planning, the development team will have access to all information about user stories from previous sprints, as well as traceability relations between diagrams (eg.: class, sequence and use cases).

Problem 3 (P3): *Lack of documentation about non-functional requirements (NFR).*

Context: The lack of documentation and incomplete information on non-functional requirements before starting a sprint

[1] Due to space limitations, we reference here some of the main papers and industrial reports. A complete list of these papers can be found at http://bit.ly/1RK7T4f.

may cause delay of important architectural decisions and / or absence of identification of important development tasks.
Source artifact: User story, Story acceptance test.
Target artifact: Architecture design diagrams, test design;
Additional information: Story acceptance criteria (e.g.; performance, security, usability).
Proposal: During the stage of sprint planning, the development team will have access to traceability relations between user stories, architecture documents, test scenarios, and the list of acceptance criteria related to performance, security, and usability, among others. Access to this information during the sprint planning will give the team members the opportunity to make architectural changes as soon as possible, to better define the user story acceptance criteria, to validate test scenarios, and to identify technical tasks that would only be identified in development life cycle. This will avoid delays in completing the sprint or avoid increasing costs to a project.

Problem 4 (P4): *Loss of management control when the project size set in the contract is measured with function point or use cases point.*
Context: Normally, in projects involving public companies or more traditional institutions such as financial and telecom organisations, software contracts take into account standard measures to define the size of a project. Typically, this is done based on function point analysis [17] or use cases points [25]. In these projects, during the transition from traditional to agile methods, there may exist conflicts when accounting for these measures (eg.: story points, ideal days).
Source artifact: User story, epic, theme.
Target artifact: Software requirements specification, use case specifications.
Additional information: Story points implemented, function points or use case points implemented per sprint.
Proposal: During the stage of sprint planning, the development team and the Scrum Master shall estimate the scope of the sprint in story points and function points (or use case points). At the end of the sprint, the same team must conduct a recount in order to understand the variation of the size of the project that has been implemented in relation to the planned project. For every new sprint, these values should be presented to the product owner and a comparison with the original value of the contract is made in order to renegotiate previously agreed conditions. This comparison will be based on the requirements document traced with user stories, epics, and themes.

D. Traceability Relations

Based on the artifact mappings described in Subsection II.B and the agile problems described in Subsection II.C, a large number of traceability relations can be generated combining the various artifacts. However, not every traceability relation can support the transition from traditional to agile software engineering processes. For example, a traceability relation between project data sheet with project charter artifact, or a traceability relation between risk burndown chart and risk management data sheet, cannot assist with the transition process.

Figure 1 shows the artifacts and their relations that are relevant to the work in this paper. In the approach, we propose

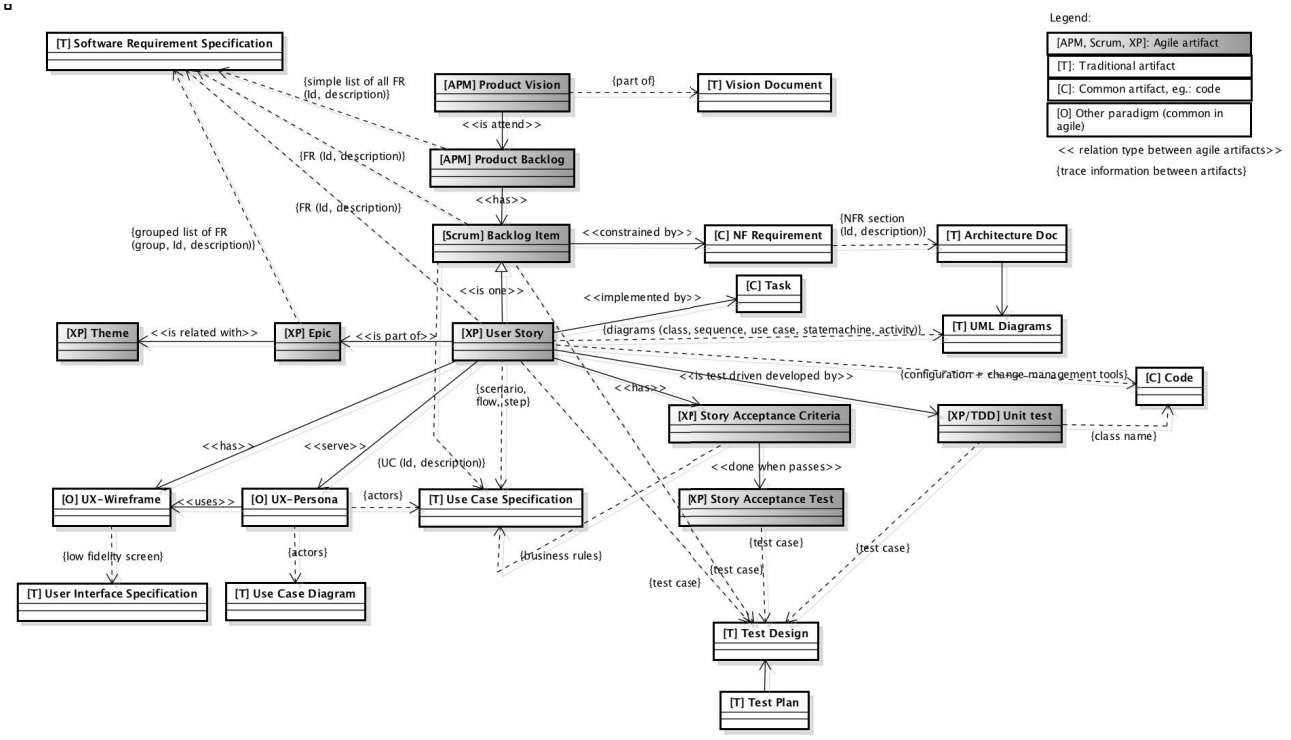

Fig. 1. Traceability Relations

TABLE II. *TRACE++* TRACEABILITY RELATIONS

Problem	Link (Trace)	Source	Target	Additional information
P1 - Rework	P1(S1, T1, I1)	S1 - User Story (Id)	T1 - Software Requirements Specification (Id, description)	I1 - % Rework (story points variation)
	P1(S1, T1, I2)	S1 - User Story (Id)	T1 - Software Requirements Specification (Id, description)	I2 - % Rework (business value variation)
	P1(S1, T9, I8)	S1 - User Story (Id)	T9 - Persona (Id, name)	I8 - % Rework (new personas involved in user story)
	P1(S1, T10, I9)	S1 - User Story (Id)	T10 - Wireframe (Id, name)	I9 - % Rework (new wireframes involved in user story)
	P1(S1, T2, I10)	S1 - User Story (Id)	T2 - Class Diagram (class name)	I10 - % Rework (new classes involved in user story)
	P1(S1, T1, I1) U P1(S1, T1, I2) U P1(S1, T9, I8) U P1(S1, T10, I9) U P1(S1, T2, I10)	S1 - User Story (Id)	T1, T9, T10, T2	I1, I2, I8, I9, I10
P2 - High-level scope	P2(S1, T2, I3)	S1 - User Story (Id)	T2 - Class Diagram (class name, atributes, operations)	I3 - Previous sprint numbers that this US had been developing
	P2(S1, T3, I3)	S1 - User Story (Id)	T3 - Sequence Diagram (class name, operation)	I3 - Previous sprint numbers that this US had been developing
	P2(S1, T4, I3)	S1 - User Story (Id)	T4 - Use Case Diagram (use case Id, actors)	I3 - Previous sprint numbers that this US had been developing
	P2(S1, T2, I3) U P2(S1, T3, I3) U P2(S1, T4, I3)	S1 - User Story (Id)	T2, T3, T4	I3 - Previous sprint numbers that this US had been developing
P3 – Lack of documenation (RNF)	P3(S1, T5, I4)	S1 - User Story (Id)	T5 - Architecture Document (NFR section -> Id, description)	I4 - Story Acceptance Criteria related to <performance, security, usability, etc.)
	P3(S2, T8, I4) U P3(S1, T5, I4)	S2 - Story Acceptance Test (Id)	T8 - Test Design (Test case Id), T5	I4 - Story Acceptance Criteria related to <performance, security, usability, etc.)
P4 – Loss of management control	P4(S1, T6, I5, I6)	S1 - User Story (Id)	T6 - Software Requirements Specification (Id, description)	I5 - Story Points implemented, I6 - Function Points implemented per sprint
	P4(S1, T6, I5, I7)	S1 - User Story (Id)	T6 - Software Requirements Specification (Id, description)	I5 - Story Points implemented, I7 - Use Case Points implemented per sprint
	P4(S1, T6, I5, I6) U P4(S1, T7, I5, I6)	S1 - User Story (Id)	T6, T7	I5, I6
	P4(S1, T7, I5, I6)	S1 - User Story (Id)	T7 - Use Case Specification (Id, name, scenario, flow, step)	I5 - Story Points implemented, I6 - Function Points implemented per sprint
	P4(S1, T7, I5, I7)	S1 - User Story (Id)	T7 - Use Case Specification (Id, name, scenario, flow, step)	I5 - Story Points implemented, I7 - Use Case Points implemented per sprint
	P4(S1, T6, I5, I7) U P4(S1, T7, I5, I7)	S1 - User Story (Id)	T6, T7	I5, I7
	P4(S3, T6, I5, I6)	S3 - Epic (Id)	T6	I5, I6
	P4(S4, T6, I5, I6)	S4 - Theme (Id)	T6	I5, I6

different types of traceability relations and different types of information sets, as follows:

- *Type of traceability relation between artifacts: <is atend>, <has>, <is one>, <is part of>, <is related with>, <done when pass>, <is test by>, <uses>, <serve>, <constrained by>, <implemented by>;*
- *Types of information sets: {actors}, {business rules}, {test case}, {functional requirement Id}, {class name}, {scenario}, {flow}, among others.*

Table II summarises the various types of traceability relations in *Trace++* with respect to the agile problems (P1 to P4) relevant to this paper. The traceability relations were created based on the elements of equation (1).

III. EVALUATION

Trace++ has been evaluated in two real projects (projects A and B) in one telecom and in one banking organisation in Brazil, located in Porto Digital[2] in Recife. The main goal of the evaluation was to analyse how *Trace++* contributes to alleviate the four agile transition problems described in this paper. More specifically, the evaluation analysed if *Trace++* can:

(a) improve decision-making on how to prioritize backlog items and better visualize possible items that may not be implemented (*Problem P1*);

(b) improve understanding of the project scope before the beginning of the sprint, in order to prevent abandonment of the project after a few iterations (*Problem P2*);

[2] http://www.portodigital.org

(c) improve understanding of the non-functional requirements before the beginning of a sprint in order to avoid delays in important architectural decision and / or in the absence of identification of important development tasks (*Problem P3*); and (d) provide better control of project scope and minimize conflicts that arise when traditional software development measures are used instead of agile measures (*Problem P4*).

Problem P1: *Absense of metrics to indicate the amount of rework that occurs in each sprint.*

Goal: improve decision-making on how to prioritize the backlog items and better visualize what are the risk items;

Question: how much rework occurred along the sprint because of requirement change requests?

Metrics (as per Table II)**:**

- I1 - % Rework (story points variation);
- I2 - % Rework (business value variation);
- I8 - % Rework (new personas involved in user story);
- I9 - % Rework (new wireframes involved in user story);
- I10 - % Rework (new classes involved in user story).

Receiver: Product Owner (PO);

Supplier: Scrum Master / Team;

Periodicity: every sprint (weekly, fortnightly or monthly);

Collection time: at the end of each sprint;

Where it will be stored: XML format;

How will the metrics be collected: the Scrum Master will have a form to fill in the data that should be collected in 4 phases: sprint planning 1 and 2, daily meeting, and at the end of the sprint:

Sprint Planning 1: for each user story selected by the PO, the team calculates the amount of story points;
Sprint Planning 2: the team breaks down each user story into smaller tasks, preferably at most 16 hours;
Daily meeting: the Scrum Master register in the bug track ing tool the changes in each story user (*source*) and the number of classes, personas and wireframes (*targets*) that have been added, as well as the amount of additional effort;
At the end of sprint: the team calculates the amount of sto ry points, the effort for each user story and the percentage increase.

How will the metrics be analyzed: during the planning meeting, at which the backlog is prioritized, the PO will know how much of the backlog has already been consumed. So he will have more elements to support his decision-making and identify the backlog risk items.

Evaluation criteria: Given the rework percentage presented at every new sprint, the PO will be asked if the approach provides more visibility of the increase of the project scope and, therefore, if it is helpful to replan the backlog from previous sprints.

Fig. 2. Evaluation Guideline for Problem 1

Trace++ was evaluated based on the guidelines proposed in [35], following four main steps: (i) planning, (ii) data collection, (iii) analysis of collected data, and (iv) recording of the results. The planning step was executed based on GQM (Goal-Question-Metric) paradigm [7]. For each agile problem we describe a conceptual level (goal), an operating level (question), and a quantitative level (metric). As an example, consider the guideline for *Problem P1* described in Figure 2. The guidelines for the other problems are available at http://bit.ly/1RK7T4f.

Table III summarises information about the companies and projects (A and B) used in the evaluation, as well as the methods used in these projects, the types of contracts, and the respective agile problems associated with each project. For each project, Table IV shows information about the number of sprints, user stories, tasks, and function points that have been collected during the evaluation phase. As shown in Table III, project A was developed by a team of nine members, using hybrid methodologies, and is related to problems P1, P2, and P3; while project B was developed by three parallel teams, with a total of nine people, and is related to problem P4. Project A started in 2010 and it is still under development and maintenance; Project B started recently and it is also under development.

TABLE III. EVALUATED PROJECTS

Project	Area	Team	Methods	Contract type	Problem evaluated
A	Telecom	9 people in the project team (project manager, team leader, software engineers, designers and test engineers)	Hybrid (UP, Scrum and XP)	Fixed price and schedule; Flexible scope.	P01, P02, P03
B	Bank	3 parallel teams with 6 software engeniners, 2 testers and 1 designer	Hybrid (UP and Scrum)	Fixed price; 1100 functions points.	P04

Project "A" is run by a company that develops solutions for a telecom company in Brazil. The project is about the development and maintenance of a billing system. The software development process used by the client is based on the Unified Process (UP), while the team's process uses a hybrid approach of Scrum, XP and UP. The types of artifacts generated by the project are: requirements document, use cases, user stories (documented in the JIRA tool[3] and team task board), acceptance criteria, class diagrams, sequence diagrams, test plan, test design, and unit test.

Project "B" is run by another company that develops solutions for public sectors in banking. The project is about the development of a new system to a public bank with a contract based on funcion points. The software development process used by the three parallel teams is a hybrid approach of Scrum and UP. This project has a public bidding contract with 1100 function points, which required documentation based on traditional processes. The project consists of three subsystems developed by three parallel teams of developers (six developers, two testers, one designer). The user stories were prioritized so that each subsystem would not last more than 30 days. Each subsystem was divided into three sprints of 10 days each.

[3] https://www.atlassian.com/software/jira

Given the data and documents available for projects A and B, 69 *Trace++* traceability relations were manually created for the two projects used in the evaluation. Examples of these traceability relations for each type of problem are shown in Figure 3. In the figure, US is an identifier used by the project to represent a user story, NFR stands for non-functional requirements, and FR stands for functional requirements. A complete list of all traceability relations can be found at http://bit.ly/1RK7T4f.

TABLE IV. PROJECT INFORMATION

Project	Sprints	User Stories	Tasks	Funcion Points
A	3 sprints, 15 days each	9 user stories totalizing 39, 29 and 32 story points per sprint, respectively	264 hours of developme nt effort	Not applicable
B	3 sprints, 10 days each	24 user stories divided into three subsystems totaling 54, 51 and 50 story points per subsystem	Not collected	124, 132 and 115 function points per subsytem for a total of 1100 FP

Problem P1

P1(US001, FR007, 160%) U P1(US001, (Class1, Class2), 250%)

This example shows the highest effort variation (250%) during one sprint caused by user story US001, function requirement FR007, and additional classes "Class1, Class2".

Problem P2

P2(US001, (Class-domain-model, Class1, Class2), (30, 31, 37)) U P2(US001, (Class3, Class4), (30, 31, 37))

This example shows previous sprints (30, 31, 37) involving user story US001 and related UML diagrams.

Problem P3

P3(US001, (NFR2-1, NFR4-2-3, NFR4-2-4), (AC1, AC2, AC3, AC4, AC5))

This example shows a traceability relation between user story US001 and a non functional requirement (NFR2-1), that requires automation scenarios related to SOAP and HTTP APIs, with an acceptance criteria (AC1) that also needs to maintain automation scenarios.

Problem P4

P4(US001A, (FR001, FR002), 5, 41) U P4(US002A, (FR001, FR002, FR006, FR008), 5, 41) U P4(US003A, FR013, 8, 41)

This example shows three user stories (US001A, US002A, US003A) related to functional requirements FR001, FR002, FR006, FR008, FR013, representing 18 story points (5+5+8), with the whole sprint concluded with 41 functional points.

Fig. 3. Example of Traceability Relations

In the following we present the results of the analysis of the collected data for each problem.

Problem 1 (P1): *Absense of metrics to indicate the amount of rework that occurs in each sprint.*

Figure 4 shows the story point variation in terms of its size per user story, and Figure 5 shows the task effort variation in terms of hours per user story, for project A. The graph in Figure 4 shows a decrease in the range of changes between the first and the other user stories (from 160% to 60%), when using the traceability relations provided by *Trace++*. As shown in the figure, some user stories did not have variations on the size. This is attributed to the fact that the scope of these user stories has already been well defined in the sprint planning. Although 60% is still a high value for changes, the approach provides a better view of the amount of rework required and mechanisms to reduce the rework over the next sprints in order to minimize the backlog of product risk items.

A similar situation occurs in Figure 5, in which the effort variation in the first user story was reduced from 250% to 46%. This reduction was possible due to the amount of rework specified in the traceability relations, which was not known before. In this particular case, the remaining variation peaks (100% and 133%, respectively) were related to low complexity in the user stories (eight and three story points, respectively), which in absolute numbers represented eight hours of additional work.

Analysis: The use of traceability relations from *Trace++* demonstrated that in every sprint planning meeting, at which the backlog was prioritized, the product owner (PO) knew about how much of the total backlog was consumed. These gave the PO more information to support decision-making and, therefore, identify the risk items that could be left out of the project. In addition, the PO confirmed that the approach provided more visibility about increase in the scope of the project assisting with the replanning of the backlog in relation to previous sprints. It was also confirmed that it is not necessary to wait until the end of the project to complete the analysis phase.

Another advantage of the approach was concerned with the analysis of the consolidated data for all the sprints. In this case, the PO noticed that there was a decrease in the size of the rework variation after the first sprint was evaluated (from 70% to 38%). The same occurred in relation to the task effort, in which the variation of effort rework decreased after the first sprint was evaluated (from 45% to 35%).

Other benefits highlighetd by team: The team that participated in the evaluation and in the development of project A, highlighted the following benefits of the *Trace++* approach:
(a) "*The traces are created iteratively, at the end of each sprint. Thus, it avoids an additional effort of creating a traceability matrix around the legacy system*";
(b)"The percentages are presented and this has helped in making the decision of what will be prioritized between the choice of new items and changes in current items";
(c)"The graphics with the percentage variation (story points and effort) help at the time to replan the backlog and to identify elements are most affected by the changes. For example, if a particular class is being so affected by the changes, it may be appropriate to hold a refactoring activity to optimize it, or establish a pair-programming rotation so that more people know of its contents, or to convince the team to perform TDD (Test Driven Development) for creating more classes of tests in order to automate the regression tests".

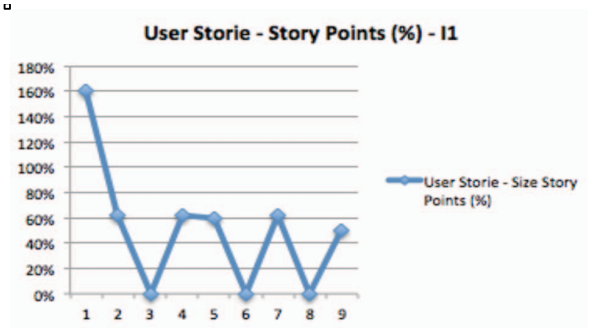

Fig. 4. P1 – Story points (size) variation per User Stories

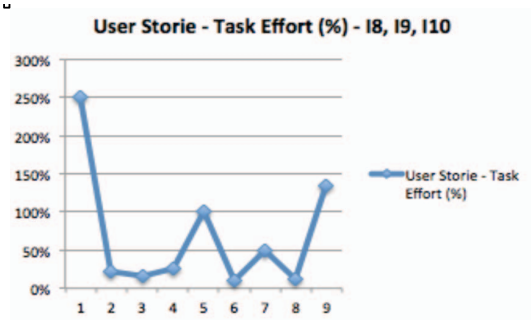

Fig. 5. P1 – Task effort (hours) variation per User Stories

Problem 2 (P2): *Lack of understanding about the high level scope before starting the sprint.*

Analysis: In the case of problem P2, the team also agreed that the *Trace++* approach provides more visibility about the scope of the sprint. The team affirmed that in some cases the approach could influence problem P1, since the improvement on understanding the scope of the project may reduce the amount of rework variation. This behavior was observed between two consecutive sprints, where the effort variation reduced from 45% to 35%. In addition, during the second sprint planning meeting, in which the tasks in the sprint backlog are detailed, the team was able to have a better idea of the traceability information involving user stories and class and sequence diagrams, helping with the details of the tasks that were part of the sprint backlog.

Other benefits highlighetd by team: The participants highlighted the following: "*Considering the specific context of the project, where the requirements have been evolved over the last five years, some more experienced members of the team have not made much use of information related to the class and sequence diagrams. However, due to high staff turnover such information can be essential to improve understanding of the scope of each sprint backlog, as was the case of two developers who recently joined the team*".

Problem 3 (P3): *Lack of documentation about non-functional requirements (NFR).*

Analysis: In the case of problem P3, the use of *Trace++* provided the team with information about user stories and non-functional requirements previously defined in the project architecture document. Access to this information during sprint planning stage gave the team the opportunity to review whether there is a need for architectural changes in the system, and better define acceptance criteria of user stories. These avoid delays in completing the sprint and increase on project costs.

The team also agreed that this approach supports the alignment between the constraints and quality attributes defined in the project architecture document and the acceptance criteria of user stories.

Other benefits highlighetd by team: The participants highlighted the following: "*In the specific case of the selected user stories, although they were identified few architectural impacts, the approach was also helpful for new members of the team who could question alternative ways to meet certain non-functional requirements, for example, the need to automate some scenarios using the JUnit Framework*[4]".

Problem 4 (P4): *Loss of management control when the project size set in the contract is measured with function point or use case points.*

Analysis: In the case of problem P4, in every new sprint of project B, the percentage of deviation of the number of function points was presented to the PO in order to compare with the original value specified in the contract. This was important to allow the PO to plan each new sprint considering the consumption of accumulated function points. In this exercise, the original requirements document was used as a basis, considering the requirements and user stories indicated on each traceability relation. The traceability relations helped the team to see that the requirements were enforcing more changes along the sprints. The use of the relations assisted the team to have more control over changes in the project scope.

The PO also commented that the above has helped him to renegotiate the scope of the project with its senior manager, since in every 10 days (sprint size), he was shown 1100 function points and, therefore, was able to avoid surprises at the end of the project. Figure 6 shows the variation of function points per sprint. As shown in the figure, at the end of the third sprint the PO could be notified that an extra 69 function points were developed. This will be used in the planning of the next subsytem, which would initially have 805 function points available, but will now be reduced to 736.

Other benefits highlighetd by team: The participants highlighted the following: "*This approach is allowing the team to continue using normally story points to track the progress of the sprint backlog through the burndown chart and only at the end of the sprints we count the function points. This helps the PO and Scrum Master with the visibility of the project scope variation*".

[4] http://junit.org

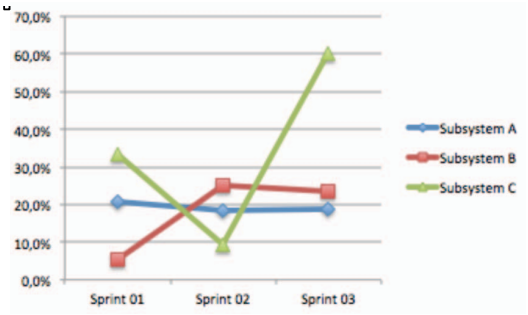

Fig. 6. P4 – Function points variation per Sprint

IV. Related Work

Several approaches and techniques have been proposed to support software traceability [10][11][18][24][38]. However, the majority of existing approaches mainly discuss the use of traceability techniques in traditional software development processes and not in agile projects. More recently, some approaches have been proposed to use traceability in agile projects [4][6][9][18][23][26][34][41].

The work in [9] provides general guidelines for using traceability in different types of agile projects, depending on the size, longevity, complexity, and criticality of the project.

According to [18], the application of traceability concepts to agile projects is still novice. In their work, the authors proposed a roadmap to provide guidelines to simplify traceability tools and provide traceability relations relevant to agile projects. The work focuses on Scrum [36] and XP [8] methodologies and proposes different types of traceability relations between: Stakeholder-User story, User story-User story, User story-Acceptance criteria, User story-Test cases, and Test cases-Refactoring. The different traceability relations were not seen with the same level of importance by the various agile developers, and some stakeholdres created more relations than others.

In [23], the author introduces the concept of traceability types identified through interviews conducted with developers, testers, configuration managers, product owners and Scrum Masters from multiple agile projects. The author concluded the importance of the following traceability types: Stakeholder-Requirement; Requirement-Version; Requirement-Requirement; Requirement-Code; Requirement-Test cases; User evaluation-Version.

The work in [4] describes a traceability management tool to ensure traceability among user stories, traditional requirements documents, test specifications, architecture design, and source code. However, to guarantee a non-invasive traceability, the authors understand that "*traceability techniques should be minimally intrusive, in the sense that people should be able to keep using the tools they are used to for creating the artifacts and still be able to maintain traceability among the artifacts produced*".

In [6], the authors present a Traceability Process Model (TPM), which is compatible to agile development processes such as Scrum and FDD, to support traceability of non-functional requirements. In [34] the authors propose an auditing model for ISO 9001 traceability requirements that is applicable in agile (XP) environments. The work in [41] analyses the benefits against the challenges of using traceability in agile software projects. The authors advocate the use of traceability to help software companies with more focused customers. The work in [26] integrates traceability within Scrum development process.

Despite some advances in the topic, the works involving traceability and agile processes are still imature. Our *Trace++* approach contributes to the area by providing traceability relations between artifacts generated during traditional and agile software development, and by assisting the problem of transitioning from traditional to agile projects, since some organisations use hybrid processes.

V. Conclusions and Future Work

In this paper we present *Trace++*, a traceability technique that extends traditional traceability relationships in order to support the transition from traditional to agile software development. We concentrate on four real problems that exist in agile projects, namely (i) lack of metrics to measure the amount of rework that occurs per sprint, (ii) lack of understanding about the high level scope of a project before starting the sprint, (iii) lack of documentation about non-functional requirements, and (iv) lack of management control. The work has been evaluated in two agile projects involving two organisations. The results of the evaluation are discussed in the paper.

Currently, we are extending the work to support the generation of *Trace++* relations in an automatic way and evaluate the cost of generating these relations. We also plan to evaluate the use of *Trace++* with respect to different characteristics of a project such as size, complexity, and clarity, as proposed in [9]. Another area is concerned with the evaluation of when in the life-cycle of agile projects traceability relations should be created. The work is also being extended to support other types of problems relevant to industry. Moreover, the approach should allow for an extensible approach in which the definition of a new problem does not require the identification of new traceability relation types, but instead, it should allow for reuse of existing *Trace++* information.

Acknowledgment

This research work was supported by the Brazilian National Research Council (CNPq) of the Ministry of Science, Technology and Innovation of Brazil, process #206556/2014-4. The international cooperation with the Open University was part of the Science without Borders[5] program.

References

[1] S. Ambler, "Agile adoption survey", 2014a, http://bit.ly/1MSNg5s.

[2] S. Ambler, "Test driven development survey", 2014b, http://bit.ly/21s1a47.

[3] D. Anderson, Kanban – Successful evolutionary change for your technology business. Blue Hole Press, Washington, 2010.

[5] http://www.cienciasemfronteiras.gov.br/web/csf

[4] P. O. Antonino, T. Keuler, N. Germann and B. Cronauer, “A non-invasive approach to trace architecture design, requirements specification and agile artifacts”, 23rd Australasian Software Engineering Conference, 978-1-4799-3149-1/14 IEEE. DOI 10.1109/ASWEC.2014.30, 2014, pp. 220-229.

[5] B. Appleton et al, “Lean traceability: a smarttering of strategies and solutions”, Configuration Management Journal, 2007.

[6] A. F. B. Arbain, I. Ghani and W. M. N. W. Kadir, “Agile non functional requiremnents (NFR) traceability metamodel”, 8th Malaysian Software Engineering Conference (MySEC). 978-1-4799-5439-1/14. IEEE, 2014, pp.228-233.

[7] V. R. Basili, G. Caldiera and H. D. Rombach, "The Goal Question Metric Approach"m 2nd ed., Wiley-Interscience, Encyclopedia of Software Engineering, 2001. p. 528-532.

[8] K. Beck, Extreme programming explained: embrace change. 2a edição, Boston, MA, Addison-Wesley Professional, 2005.

[9] J. Cleland-Huang, “Traceability in agile projects”, DOI 10.1007/978-1-4471-2239-5_12, Springer-Verlag London, 2012.

[10] J. Cleland-Huang, O. Gotel and A. Zisman, Software and systems traceability. ISBN: 978-1-4471-2238-8 (Print) 978-1-4471-2239-5 (Online), 2012.

[11] J. Cleland-Huang et al. “Software traceability: trends and future directions”, The 37th International Conference on Software Engineering, ICSE, Hyderabad, India, 2014.

[12] A. Cockburn, Crystal clear: a human-powered methodology for small teams. 1a Edição, Addison-Wesley Professional, 2005.

[13] M. Cohn, Succeeding with agile: software development using Scrum. Addison Wesley, 1st edition, Boston, ISBN-13: 978-0321579362, 2010.

[14] L. Dronzek and T. Lanowitz, Market snapshot report: agile realities. Voke research, 2012. http://www.vokeinc.com.

[15] Dynamic systems development method (DSDM) consortium. DSDM atern handbook, 2013, http://www.dsdm.org/dig-deeper/book/dsdm-atern-handbook.

[16] A. Espinoza and J. Garbajosa, “A study to support agile methods more effectively through traceability”, Innovations in Systems and Software Engineering, v. 7, n. 1, 2011, pp. 53-69.

[17] D. Garmus, Function point analysis: measurement practices for successful software projects, Addison-Wesley, 1st Edition, ISBN-13: 978-0201699449, 2000.

[18] A. Ghada and S. Zeinab, “A multi-faceted roadmap of requirements traceability types adoption in Scrum: an empirical study, The 9th International Conference on INFOrmatics and Systems (INFOS), 15-17 December, Software Engineering - Challenges of Openness Track, 2014.

[19] M. Griffiths, PMI-ACP Exam Prep. RMC Publications, Inc. EUA, 2012.

[20] J. Highsmith, Agile software development ecosystems, Boston, EUA, Addison-Wesley, 2002.

[21] J. Highsmith, Agile project management – creating innovative products, EUA, Addison-Wesley, 2004.

[22] IDF – Interaction Design Foundation, https://www.interaction-design.org.

[23] M. Jacobsson, “Implementing traceability in agile software development”, Department of Computer Science Lund University, Faculty of Engineering, LTHSE-221 00 Lund, Sweden www.lth.se.

[24] W. Jirapanthong and A. Zisman, “XTraQue: traceability for product line systems”, Softw Syst Model (2009) 8:117–144, DOI 10.1007/s10270-007-0066-8, Springer-Verlag, 2007.

[25] G, Karner, Metrics for Objectory, University of Linköping, Sweden. No. LiTH-IDA-Ex-9344:21, 1993.

[26] M. Kodali, “Traceability of requeriments in Scrum software development process”, Malardalen University School of Innovation Design and Engineering Vasteras, 2015.

[27] P. Kruchten, The Rational Unified Process: an introduction (3rd edition), ISBN-13: 078-5342197709, Addison Wesley Signature Series, December, 2003.

[28] J. D. Luca, Feature driven development, Nebulon Pty Ltd., 2002, http://www.featuredrivendevelopment.com.

[29] C. O. Melo, V. A. Santos, H. Corbucci, E. Katayama, A. Goldman and F. Kon, Métodos ágeis no Brasil: estado da prática em times e organizações. Technical Report RT- MAC-2012-03. Science Computing Department, IME-USP, May, 2012. Portuguese Version Only.

[30] L. Nielsen, Engaging personas and narrative scenarios. PhD-Series, 2004.

[31] J. Patton, User story mapping: discover the whole story, build the right product, O'Reilly Media, 2014.

[32] M. Poppendieck and T. Poppendieck, Lean software development: an agile toolkit, EUA: Addison-Wesley, 2003.

[33] Project management institute (PMI), PMBOK - A guide to the project management body of knowledge, 5th edition, Newtown Square, PA, 2013.

[34] M. Qasaimeh and A. Abran, “An audit model for ISO 9001 traceability requirements in agile-XP environments”, Journal of Software, Vol. 8, No. 7, July 2013, pp. 1556-1567.

[35] P. Runeson, M. Höst, “Guidelines for conducting and reporting case study research in software engineering”, Empir Software Eng, 14:131–164, DOI 10.1007/s10664-008-9102-8, Springer, 2009.

[36] K. Schwaber, Agile project management with Scrum. EUA: Microsoft, 2004.

[37] Software engineering institute (SEI), CMMI for development, version 1.3, staged representation, 2010. Pittsburgh, PA, CMU/SEI-2010-TR-033.

[38] G. Spanoudakis and A. Zisman, "Software traceability: a roadmap," in S. K. Chang, ed., Handbook of Software Engineering and Knowledge Engineering, August, 2005.

[39] Versionone, The 8th state of agile development survey, 2014, http://bit.ly/250N4eJ.

[40] Versionone, The 9th state of agile development survey, 2015, http://bit.ly/1owKxb6

[41] D. H. Vuong, “Traceability in agile software projects”, University of Gothenburg Chalmers University of Technology Department of Computer Science and Engineering SE-412 96 Göteborg, 2013.