Reliability in Agile Software Engineering: A Dilemma

Arbi Ghazarian
Email: Arbi.Ghazarian@asu.edu

While agile software engineering methods have been successfully applied to many software application areas, their adoption in the domain of safety-critical and high-reliability systems has been hampered by a lack of appropriate reliability techniques. In contrast, traditional development approaches have relied on traceability-based techniques for the verification and certification of reliability of software systems. Traceability plays a vital role in the production, verification, and certification of highly dependable software-intensive systems that serve as the computational backbone for key human activities ranging from science and engineering to business and everyday services that support our social and civil infrastructures. Traceability is demanded or recommended by numerous standards including ISO 15504 [1], ISO 9001:2000 [2], Software Engineering Institute's Capability Maturity Model Integration (SEI-CMMI) [3], DoD std 2167A [4], IEEE std 830-1998 [5], IEEE std 1233 [6], IEEE std 1219-1988 [7], and IEEE std 982.2 [8]. In particular, IEEE Standard 982.1 [9] recognizes traceability as a measure of software reliability.

Existing traceability technology mostly relies on pair-wise post-mortem analysis of software artifacts such as bi-directional tracing of detailed requirements statements, especially safety requirements, in requirements specifications to their corresponding design components, design components to their corresponding source code components, and requirements statements to test cases that verify the correct implementation of the stated requirements. As a result of this traceability analysis, missing traceability links are identified and treated as potential threats to the reliability of the software systems. For instance, a detected missing link from a safety requirement statement in the system's specification to the system's design or source code indicates that the safety requirement has not been designed or implemented in the system and as such is a threat to the reliability of the system. Similarly, a detected missing trace link from a source code component to a software requirement statement indicates that there is extra and unnecessary code in the system that does not correspond to any stated requirements of the system and as such can be potentially harmful to the reliability of the system.

Unfortunately, existing traceability approaches suffer from a technical bias that impedes their effectiveness in modern agile-based software projects. The majority of these approaches were developed during 1980s and 1990s where heavyweight plan-based software development models and methodologies were the mainstream. The wide industry adoption of agile software development approaches in the recent years has posed a significant challenge to traditional traceability approaches.

The problem with traditional traceability approaches is that many of the assumptions that they rely on do not hold true for agile projects. First, most existing approaches to traceability are document-centered in the sense that traceability in these approaches is achieved outside the design of a system through creating supplementary system documentation such as traceability matrices or allocation tables. This means that the emphasis in traditional traceability approaches is not on designing traceable software systems, but rather detecting and documenting the traceability links. For instance, it is a common technique in many existing traceability

approaches to construct a requirements traceability matrix that establishes traceability links between software requirements and software components that satisfy these requirements. The creation of these additional traceability documentations is in sharp contrast to agile software engineering, which places a high emphasis on the production of source code rather than documentation.

Second, most existing traceability approaches rely on the existence of a formal requirements process, where uniquely-identifiable detailed software requirements are documented and stored in a static medium such a software or system requirements specification document (SRS) or a requirements database that will contain the requirements throughout the life of the project. In contrast, in agile software engineering, requirements are captured and communicated through informal channels such as user stories, feature backlogs, and informal discussions with on-site customers. The informality of the requirements in agile environments renders traditional traceability approaches ineffective. The provision of effective techniques for achieving software traceability in agile environments necessarily requires rethinking the traditional assumptions about software development processes.

This current lack of agile traceability approaches has created a dilemma for development organizations that produce software-intensive systems for safety critical and high-reliability applications. On the one hand, agile methodologies are believed to be capable of delivering improved performance and development productivity, which are important concerns for the survival and sustainability of these development organizations. On the other hand, the non-suitability of existing traceability approaches for agile environments makes agile development approaches a non-option for these environments as without effective traceability techniques they cannot meet the stringent software verification and certification requirements mandated by safety critical and high-reliability systems.

To address the abovementioned reliability problem in agile software engineering, I am perusing a line of research that aims to investigate novel traceability techniques that are especially useful for agile development environments (see [10][11][12][13] for some previous work). In essence, we need to answer two research questions:

1. *How can we achieve software traceability under the agility assumptions, namely:*
    a) *the informality of software requirements and*
    b) *the centrality of source code (and its design properties) as the main artifact of the agile development process?*

2. *How can we achieve system-designed traceability? That is,* ***how can we construct software systems that are traceable by their structure?***

The purpose of this article is to express my position with regards to the approach that should be taken to answer these questions, foster discussion among the software reliability research community, and solicit feedback. My position is that we should regard ***traceability as an intrinsic structural property of software systems***. This view of traceability is in contrast to traditional traceability approaches where traceability is achieved extrinsically through creating supplementary documentation such as the Requirements Traceability Matrices (RTM). I propose a change of research focus from existing analytical techniques for post-mortem detection of trace

links between software artifacts to exploring synthetic techniques that allow the proactive construction of traceable software systems. Here, the emphasis is on proactive elimination of traceability problems by engineering software systems with system-designed traceability (i.e., traceability as a design property) rather than reacting to the traceability problem through various analysis methods to discover traceability relations after the system is built. An advantage of a constructive approach to traceability is that it is a more natural fit for agile software development as it focuses on the structure of the source code (i.e., the design properties of the source code), which is the main artifact in agile development methodologies.

References:

[1] International Organization for Standardization, Software Process Improvement and Capability Determination, ISO 15504.

[2] International Organization for Standardization, Quality Management Systems, ISO 9001:2000, 2000.

[3] SEI-CMM, http://www.sei.cmu.edu/cmmi/

[4] Department of Defense, Defense Systems Software Development, DoD std 2167A, 1988.

[5] IEEE Computer Society, IEEE Recommended Practice for Software Requirements Specifications, IEEE std 830-1998, 1998.

[6] IEEE Computer Society, IEEE Guide for Developing System Requirements Specifications, IEEE std 1233-1998, 1998.

[7] IEEE Computer Society, IEEE standard for Software Maintenance, IEEE std 1219-1998, 1998.

[8] IEEE Computer Society, IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, IEEE std 982.2-1998, 1998.

[9] IEEE Computer Society, IEEE Std. 982.1 – 1988, IEEE Standard Dictionary of Measures to Produce Reliable Software, 1989.

[10] Ghazarian, A., "Traceability Patterns: An Approach to Requirement-Component Traceability in Agile Software Development", Proceedings of the 8th WSEAS International Conference on Applied Computer Science (ACS 2008), Venice, Italy, November 2008.

[11] Ghazarian, A., "Coordinated Software Development: A Framework for Reasoning about Trace Links in Software Systems", Proceedings of the IEEE's 13th International Conference on Intelligent Engineering Systems (INES 2009), pp 236-241, IEEE Computer Society, Barbados, April 2009.

[12] Ghazarian, A., "A Matrix-Less Model for Tracing Software Requirements to Source Code", International Journal of Computers, NAUN, ISSN: 1998-4308, Issue 3, Volume 2, pp. 301-309, December 2008.

[13] Ghazarian, A., "A Research Agenda for Software Reliability", IEEE Transactions on Reliability, IEEE Reliability Society Technical Operations Annual Technical Report for 2010, Number 3, Volume 59, pp 449-482, September 2010.