Agile Requirement Traceability Matrix

Serin Jeong, Heetae Cho, Seonah Lee
Gyeongsang National University
Jinju, Republic of Korea
{rin0608,cht3205,saleese}@gnu.ac.kr

ABSTRACT

Over the past 20 years, agile development methodology has been commonly adopted by developers. One reason is that it flexibly responds to changes. In contrast, a typical requirement traceability matrix is inflexible in incorporating changes. In this paper, we propose Agile Requirement Traceability Matrix (ARTM). ARTM manages traceability mappings between two artifacts and automatically generates the entire requirement traceability matrix in spreadsheet. The case study, we also conducted, shows that ARTM flexibly incorporates changes in the spirit of agile development.

1 INTRODUCTION

Agile development methodology has been commonly adopted by software developers. One of various reasons is that it flexibly responds to changes so as to succeed in uncertain environments [1]. In contrast, a typical requirement traceability matrix[1] is inflexible in incorporating changes. Thus, it is usually made at the end of development phases for quality assurance [3]. Moreover, the projects in practice do not comply with the traceability standards [8].

Traceability matrix tools were proposed, developed and commercialized [4]. For example, Doors is a commercial tool managing requirement traceability with a change management system [4]. Klock et al. proposed a tool that infers traceability relationships between items by using natural language processing and information retrieval techniques [6]. Garcia et al. developed a web application in which a developer clicks on a html component and selects a requirement item in a drop down menu. Based on the selection, the application creates the requirement traceability matrix [5]. However, those tools typically require a considerable effort from developers.

In this paper, we propose Agile Requirement Traceability Matrix (ARTM) to flexibly incorporate mapping changes. ARTM is a spreadsheet application that manages the mappings between two artifacts in each sheet. It automatically generates a requirement traceability matrix (RTM), whenever a developer wants to check the mapping relationships among multiple artifacts. In addition, it checks the inconsistencies in RTM and marks them so that the developer easily finds the inconsistency issues in the RTM. We also conduct a case study to verify ARTM. The case study shows that, with ARTM, a developer easily inputs the mapping changes along with the practices of agile development methodology.

[1]If you search "requirement traceability matrix template" in Google, you could find many spreadsheets, listing artifact names as columns. One of them is found in the link: https://www2a.cdc.gov/cdcup/library/templates/.

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3195089

2 TYPE OF REQUIREMENT CHANGES

The papers [2][9] suggest the types of requirement changes. Based on them, we formulate the mapping changes according to requirement changes, as shown in Table 1.

Table 1: Mapping Changes according to Requirement Changes

Requirement Changes	Mapping Changes
Create a new requirement *a'*	$a' \mapsto b'$
	$a' \mapsto b\circ$
Refine a requirement *a* with detailed ones	$(a \mapsto b) \land (a' \mapsto b\circ)$
	$(a \mapsto b) \land (a' \mapsto b')$
Decompose a requirement *a* into multiple ones	$\neg(a \mapsto b) \land (a_1' \mapsto b_1\circ) \land (a_2' \mapsto b_2\circ)$
	$\neg(a \mapsto b) \land (a_1' \mapsto b\circ) \land (a_2' \mapsto b')$
	$\neg(a \mapsto b) \land (a_1' \mapsto b') \land (a_2' \mapsto b\circ)$
	$\neg(a \mapsto b) \land (a_1' \mapsto b_1') \land (a_2' \mapsto b_2')$
Merge multiple requirements a_1 and a_2 into one	$\neg(a_1 \mapsto b_1) \land \neg(a_2 \mapsto b_2) \land (a' \mapsto b\circ)$
	$\neg(a_1 \mapsto b_1) \land \neg(a2 \mapsto b_2) \land (a' \mapsto b')$
Replace a requirement *a* with another one	$\neg(a \mapsto b) \land (a' \mapsto b\circ)$
	$\neg(a \mapsto b) \land (a' \mapsto b')$
Delete an existing requirement *a*	$\neg(a \mapsto b)$
Deactivate a requirement *a*	$a \mapsto \varnothing$

In Table 1, *a* and *b* represent artifact items. When *a* is the current item, *a'* represents a new item and $a\circ$ represents another item in the artifact of *a*. The arrow $a \mapsto b$ means that *a* is mapped to *b*. For example, when *a'* is created, it could be mapped to a new item *b'*, as shown in the "create" row of Table 1. The notation, $\neg(a \mapsto b)$ means that a mapping relationship between *a* and *b* has been removed. For example, when *a* is decomposed into a_1' and a_2', the mapping $a \mapsto b$ should be removed. Also a_1' and a_2' can be mapped to $b_1\circ$ and $b_2\circ$. The mapping $a \mapsto \varnothing$ means that *a* is inactive. The mapping changes are established even if *a* and *b* are exchanged.

3 ARTM

Kniberg's spreadsheet automatically creates user story cards from backlogs [7]. Based on his spreadsheet, we propose Agile Requirement Traceability Matrix (ARTM).

3.1 Implementing RTM in Agile Practices

In order to implement RTM in agile practices, ARTM should support incremental development by marking unimplemented user stories in backlogs. ARTM also has the capability of enabling developers to input the mapping changes between two artifact items, whenever s/he finds them. ARTM can also automatically generate the RTM that presents the mapping relationships among multiple artifacts.

3.2 Automatic Generation of RTM

To generate RTM, we developed Algorithm 1. The input of the algorithm is a queue of sheets that contain the mapping relationships between two artifacts. The output of the algorithm is a new sheet that contains the RTM among multiple artifacts.

Algorithm 1: Mapping algorithm

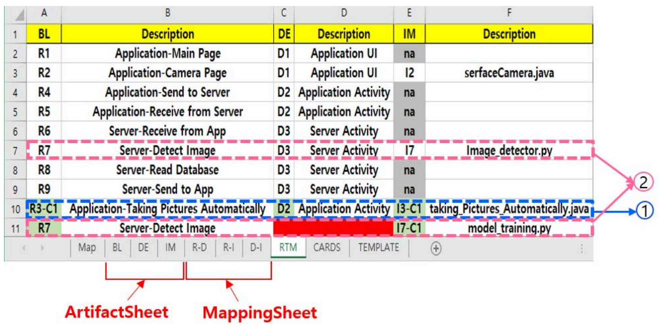
```
Input:
- MappingSheet [M]: A queue of sheets contained the mappings between two artifacts
Output:
- RTMSheet: A sheet that generates the final RTM
Data:
- currentMS: A sheet dequeued from MappingSheet[M]
- A: The first artifact contained in currentMS
- B: The second artifact contained in currentMS
Create a title
Sort MappingSheet[M] by the number of mappings
while $MappingSheet.size \neq 0$ do
    $currentMS \leftarrow MappingSheet.dequeue()$
    if $RTMSheet.isEmpty()$ then
        Copy currentMS to RTMSheet
        continue
    if $currentMS.A \in RTMSheet$ && $currentMS.B \in RTMSheet$ then
        Compare each item of currentMS with each item of
        RTMSheet, check the traceability errors and copy items
    else if $currentMS.A \in RTMSheet$ && $currentMS.B \notin RTMSheet$ then
        Compare each item of currentMS with each item of
        RTM and copy items
    else if $currentMS.B \in RTMSheet$ then
        Compare each item of currentMS with each item of
        RTM and copy items
    else
        $MappingSheet.enqueue(currentMS)$
Check the traceability errors
```

The algorithm first creates a title row that arranges *ID* and *Description* of artifacts in user-defined order. It then sorts *MappingSheet[M]* according to the number of mappings. It next repeatedly copies the mappings of *MappingSheet[M]* to *RTMSheet* until the queue is empty. In the repetition, the *currentMS* variable has the sheet dequeued from *MappingSheet[M]*. If *RTMSheet* is empty, all mapping relations of *currentMS* are copied to *RTMSheet*. The variables *A* and *B* represent the first and second artifacts of the *currentMS*, respectively. If *RTMSheet* contains both *A* and *B*, the algorithm compares items of *currentMS* with items of *RTMSheet* and checks errors in traceability. If it finds the mapping relationships that belong to *currentMS* but not *RTMSheet*, it copies them to *RTMSheet*. If *RTMSheet* exclusively contains *A* or *B*, it compares items of the contained artifact with items of *RTMSheet* and then copies the items of the other artifact to *RTMSheet* according to the mapping relationships of *currentMS*. If *RTMSheet* does not contain both *A* and *B*, it enqueues *currentMS* to the queue of the mapping sheets. Through the repetition, the mappings between artifacts are collected and used to create the final RTM. The algorithm finally checks errors in traceability and marks them.

4 CASE STUDY: MUSHROOM DETECTOR

We applied ARTM to a project that developed a mushroom detector. It consists of an android app and a server. The android app takes pictures and displays image information. The server manages a database of mushroom information and detects the mushrooms images with a Convolutional Neural Network based image classifier.

At the the second sprint plan of the project, we made two changes in the mappings of requirements and implementation items. We recorded the changes in the *R-I* mapping sheet and marked them in green (see *MappingSheet*). To check RTM, we clicked on the "create RTM" button in our spreadsheet application.



	A	B	C	D	E	F
1	BL	Description	DE	Description	IM	Description
2	R1	Application-Main Page	D1	Application UI	na	
3	R2	Application-Camera Page	D1	Application UI	I2	serfaceCamera.java
4	R4	Application-Send to Server	D2	Application Activity	na	
5	R5	Application-Receive from Server	D2	Application Activity	na	
6	R6	Server-Receive from App	D3	Server Activity	na	
7	R7	Server-Detect Image	D3	Server Activity	I7	Image_detector.py
8	R8	Server-Read Database	D3	Server Activity	na	
9	R9	Server-Send to App	D3	Server Activity	na	
10	R3-C1	Application-Taking Pictures Automatically	D2	Application Activity	I3-C1	taking_Pictures_Automatically.java
11	R7	Server-Detect Image	🟥	🟥	I7-C1	model_training.py

Figure 1: A Result Sheet of Our Case Study

Figure 1 shows the created RTM. The first change was the replacement from R3(Taking Pictures) to R3-C1(Taking Pictures Automatically), as shown in Number ①. The replacement from I3 to I3-C1 occurred accordingly. The second change was the refinement from I7(Image_ detector.py) into I7 and I7-C1, as shown in Number ②. This refinement is made to create a CNN model for detecting the mushrooms images. However, we made no changes in the mapping sheet of design components and implementation items. Therefore, the RTM marks the cell of design components in red, indicating the errors in RTM. The description is copied from *ArtifactSheet*.

We performed the project with an agile process, where we maintained backlogs which sorts the requirements according to their priorities. The requirements with low priorities were not implemented yet. In Figure 1, the na cells marked in gray indicate the unimplemented requirements.

5 CONCLUSION

To create a flexible RTM in the principle of agile development, we proposed ARTM, a spreadsheet application, that manages the mappings between two artifacts in separated sheets, that automatically generates the final RTM, and that automatically checks the inconsistencies in the mappings of several artifacts. We also showed the cases of incorporating the mapping changes according to the requirement changes in ARTM. In the future, we will improve the performance of ARTM, because we found that the loading time of the application takes around 2 minutes. We will also conduct more case studies to confirm the effectiveness of ARTM.

REFERENCES

[1] Agile Alliance. 2018. Agile 101. (2018). https://www.agilealliance.org/agile101/
[2] Chang C.K. Cleland-Huang, J. and M. Christensen. 2003. Event-based traceability for managing evolutionary change. *IEEE Trans. Softw. Eng.* 29, 9 (2003), 796–810.
[3] Gotel O.C. Huffman Hayes J. MÃďder P. Cleland-Huang, J. and A. Zisman. 2014. Software traceability: trends and future directions. In *Proc. of Future of Softw. Eng.* ACM, 55–69.
[4] NicolÃąs J. AlemÃąn J.L.F. Toval A. Ebert C. de Gea, J.M.C. and A. VizcaÃŋno. 2011. Requirements engineering tools. *IEEE software* 28, 4 (2011), 86–91.
[5] J.E. Garcia and A.C. Paiva. 2016. A Requirements-to-Implementation Mapping Tool for Requirements Traceability. *JSW* 11, 2 (2016), 193–200.
[6] Gethers M. Dit B. Klock, S. and D. Poshyvanyk. 2011. Traceclipse: an eclipse plug-in for traceability link recovery and management. In *Proc. 6th int. workshop on traceability in emerging forms of softw. eng.* ACM, 24–30.
[7] H. Kniberg. 2015. *Scrum and XP from the Trenches.* Lulu. com.
[8] MÃďder P. Kuschke T. Rempel, P. and J. Cleland-Huang. 2014. Mind the gap: assessing the conformance of software traceability to relevant guidelines. In *Proc. 36th ICSE.* ACM, 943–954.
[9] Iimura Y. Tashiro H. Massey A.K. Saito, S. and A.I. AntÃşn. 2016. Visualizing the effects of requirements evolution. In *Proc. 38th ICSE Companion.* ACM, 152–161.