Traceability of test artifacts as a QA planning and test selection tool

- Venkat Moncompu[‡] and Sreeram N Gopalakrishnan[±]

Abstract: Rapid prototyping and development techniques combined with Agile development methodologies are pushing the envelope on the best practice of testing early and testing often. Keeping pace with the quick development turn-around and shorter time to market and being adaptive to late changes in requirements requires effective management of quality process. The use of traceability of test artifacts – test cases, test defects, test fixtures – mapped to the requirements – needs, features, use cases and supplementary requirements – as a QA scheduling and planning tool though mentioned in passing and claimed to have been practiced, has been largely overlooked by the industry. This paper looks into such a possibility through the use of a study for software that involves iterative application development practices and tries to bring this aspect of using traceability as a QA management tool into focus.

Introduction:
In the recent times, many software methodologies have come to be classified under the hood of "Agile Methodology." These methods came about in response to the need for *adaptive* design and development techniques as opposed to *predictive* techniques to meet the changing or evolving user requirements and needs. Software development is not a defined process, at the very least because the main inputs to the process activities are people. Agile methods are *people* oriented rather than *process* oriented. Agile methods are *iterative*. Iterative development techniques adapt to changing requirements by focusing on the product development with "good enough" requirements. That said; there is still an element of planning involved per iteration where a subset of the required features are broken down into tasks, estimated in detail, and allocated to programmers.

Use case modeling is a very popular and effective requirements management technique. Use cases capture most of the functional requirements of a software system. They describe the user goals and the sequence of interactive steps to achieve the goal. Use cases are widely adopted in iterative software development methodology such as the unified process and other agile techniques which are iterative or evolving in nature. Verification techniques to derive test cases from use cases are well established.[5,6] So planning testing cycles entails effective traceability of test artifacts to requirements *planned* for the iteration. Though the emphasis in agile development is on people than on process and on working software over comprehensive documentation and responding to change than following a plan, a QA management process needs to remain nimble to the changing and evolving needs and requirements.[7] This is precisely where the traceability matrix can be leveraged to perform optimal QA activities that give the most value.

Agile Testing:
Agile QA Testing involves closer and tighter feedback within each cycle of iteration, defining levels and types of testing in each cycle of iteration. So how can planning of requirements testing work with iterations? User needs in an agile process are defined by a *story* (sometimes captured as use-cases and features) planned to be implemented iteratively. Work break down for development (in iterations) of these use-cases and features is defined in terms of *tasks*. As a logical extension, the QA effort can also be *tasked* for planning and scheduling purposes.

The scope of testing in iteration, usually, is a set of unit and (build) acceptance tests to verify the requirements and features planned for the iteration. The need for constant and continuous regression testing is warranted as the software construction evolves and bugs get fixed, just as it scopes the features and use-cases that go into the current iteration or development cycle. Iterations, being time-boxed, do not wait for the exit or entry criteria to be met nor are they predefined.

Agile testing leaves a lot of room for exploratory and ad-hoc testing that isn't necessarily captured in the use-cases and/or features (remember "just enough documentation to develop software"). In agile methodology, the emphasis is on software construction rather than documentation unlike the traditional waterfall model of software development. The two main premises of being agile are:

1. Ability to welcome and adapt to requirement changes later in the development life cycle.
2. Testing often and testing early (in iterative cycles).

Apart from these two basic tenets, the other difference from a waterfall model is that the requirements are never really "frozen" in development such that it becomes an entry criterion for software construction phase. Prototyping is the key aspect of agile development technique that helps in getting user feedback early and continuously in the development life cycle. This reduces the 'dreaded integration phase' late in the software development phase minimizing the risk of falling short of user needs or ending up with unfulfilled requirements. User acceptance tests serve as exit (or acceptance of the build) iteration criteria and to measure progress (or burn rate) of the project. So, in techniques such as feature driven development and test driven development, the mapping of the features and use cases to test cases – traceability – serves as a valuable tool to effectively plan and schedule testing just as features and cards are used to plan development in iterative cycles. And just as use cases provide a user perspective for developers and designers; testers have the onus of ensuring the software meets the user requirements *adequately*. This can be effectively achieved by mapping test artifacts to requirements that are modeled as use cases and testing the intended functionality independently.

Scheduling and Iteration planning:
The agile techniques for software development uses tasks in place of work break-down structures referred in traditional project planning tools. To effectively understand the use of tasks and planning of effort from a QA perspective, it is useful to break-down the QA work product into iterations based on the features and functional specifications that are planned for the iteration. Traceability matrices provide a very convenient way to ensuring the intended features are tested and verified. This further provides valuable feedback to the project team (including the end-user stakeholder) about the software construction progress. To be effective, therefore, it is important that the traceability is mapped thoroughly making the features provided transparent to all stakeholders. And for the QA manager, it provides a good substitute from "traditional" selection criteria for regression and acceptance tests. It plays an important role in providing a basis for statistical information such as burn-rates and velocity for the team management.

Multi-dimensionality of Traceability:
For the sake of clarity, a case study in the form of a traceability to map test cases relating to use cases and features of a student registration system is discussed here. Consider a student-course registration system. It should have the following features:

1. Users (Students, Registrar and Professors) should be able to register into the system.
2. Users should be able to create, update or delete their profiles and preferences.
3. Users (Students) should be able to register for classes and pay for courses enrolled in securely.
4. Users (Students, Registrar and Professors) should be able to view the student transcripts based on access restrictions.
5. Users (Registrars and Professors) should be able to create course offerings and the system should provide a catalog of courses.

Now, as with any system of moderate complexity, the set of requirements can never be really termed "complete." And the process should be adaptive to the changing user needs. For the sake of this example, these set of requirements would suffice, though. And a possible set of use cases that can be identified for the system are:

No.	UC ID #	Title	Brief Description

1	**ST001**	Student registers for a course	Student searches the system to enroll for a course. The system lists all course offerings for which seats are available for the student and accepts enrollment subject to payment of fees.
2	**ST002**	Student drops out of a course	Student is able to de-register from a course that is he signed-up for. If the drop-off date is passed, the student loses a certain amount of the fee paid, otherwise, the entire fee (minus a minimal operating non-refundable fee) is returned/credited back to the student.
3	**RG001**	Registrar adds a course to the system	The system lets the registrar offer a course to students. Registrar searches for the professor offering the course and updates the information if the date is prior to registration start date.
4	**RG002**	Registrar drops course offering	The system lets the registrar remove a course from the system prior to the drop-off date.
5	**PR001**	Professor enters the course details offered to the system(registrar)	Professor specifies course timings, maximum number of students and chooses classroom provided by the system. The system tracks available rooms against requested time by the professor for the course. The system creates dependencies to the course against the maximum number of students and against the student's course records to ensure pre-requisites are met while registering for the course.
6	**ST003**	Student pays for the registered courses	Student uses the system to pay for the courses registered for (checked-out) using a secure payment transaction.
7	**ST004**	Student reviews course history	The system shall allow the student to review the transcripts and course grades for the courses registered and completed. System shall also show a summary of student-course interactions during the last 4 years.

Table 1: Use Cases

The use cases descriptions define the main success scenarios of the system. However, not every use case scenario ends in a success for the user. While elaborating the use-cases using the descriptive text to capture these alternate paths, new aspects of the systems come to light when exceptions are encountered (non-happy path behavior of the system is being captured).[1] Spence and Probasco[8] refer to them as overloading the term requirements, a common point of confusion with Requirements Management. These may not be clear from the user needs and system features captured, but they are a very vital and essential aspect of the system behavior. To ensure that the system meets these requirements and for coverage to be effective, these have to be elicited clearly and traced completely. Alternate paths may also be captured using a usability (scenario) matrix as seen in table 2. While the use cases are mapped against features (or cards) which are planned for the iteration, so can the use-cases, the use-case scenarios that stem from these and so on, cascading to the test cases (and test artifacts).

Flow ID	Main Success Scenario	Alt Flow 1	Alt Flow 2	Alt Flow 3	Alt Flow 4	Alt Flow 5	Alt Flow 6

Flow 001	Course search	Search Fails	Misspelt/Mistyped - system suggests alternates				
Flow 002	Professor search	Search Fails	Misspelt/Mistyped - system suggests alternates				
Flow 003	Retrieve transcript history	No history available	Transcripts requested are no longer available				
Flow 004	Login	Wrong credentials	Three strikes and session times out requiring manual re-verification by the user	Password reset request - Hint Question	Unable to set cookies - login failure		
Flow 005	Course registration	Payment gateway failure	Insufficient credit	Payment gateway authentication failure	Course fee cost split across multiple modes of payment - Debit, Credit, Loans	Courses selected locked-in for payment over phone or on-campus visit	Seats unavailable

Table 2: Usability Matrix

Note that the usage of the application flow, even though captured, could end-up varying the application flow based on the data – for e.g. a student logging into the system would be provided with a different set of features and screen flows compared to a professor who uses the system or a registrar. Supplementary requirements corresponding to the architectural requirements for the system cannot be mapped unless captured separately. These remain outside the functional requirements modeled by the use cases as seen in the table 3 below.

Req. ID	Performance
Perf001	System shall be scalable to about 5000 users of the system at any given time
Perf002	System shall complete the external payment gateway transaction in 60 sec. Otherwise the transaction should time out
	Security
Sec001	System shall allow users of the system to login securely
Sec002	System shall follow the thrice-a-strike-out rule for login credentials
Sec003	System shall request re-authentication at the time of beginning a payment transaction

Sec004	System shall use a secure gateway to some supported third-party clients for credit authorization on behalf of the student's credit application

Table 3: Supplementary Requirements

A sample list of business rules that have to be followed could be summarized as:

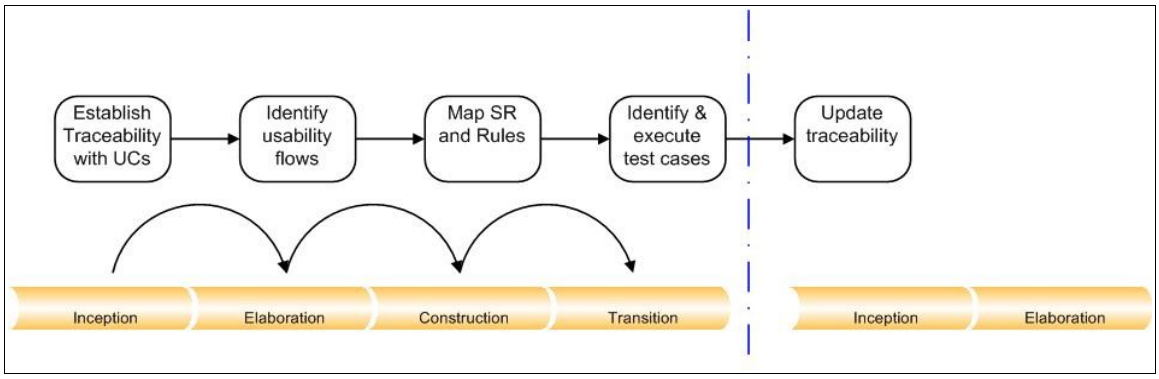
BR1: Students without pre-requisites defined for the course seeking to enroll in should be prevented from trying to register i.e checkout the course.
BR2: Students checking out courses have to register within 2 working days from the time of initiation checkout otherwise the seats shall not be guaranteed and released to the general pool.
BR3: If the courses are outside of the student's planned Major department, then such courses will should require an advisor override and cannot exceed two(2) courses outside the major program of study.

In the above case, when the mapping of the test case flows across functionality is carried, it becomes evident that the granularity of detail falls short when mapping the coverage of the test flows against the business rules as can be seen in the tabular representation below:

Use case flows	Flow 1	Flow 2	Flow 3	Flow 4	Flow 5	Flow 6	Flow 8	Flow 9	Flow 10	Flow 11	Flow 12	Flow 13	Flow 14	Flow 15
Search for a course				X	X	X				X	X	X	X	X
Keyword Search	√						√							
Instructor-based search		√						√						
Course Code Search			√						√					
Navigate to the course														
College->Dept.->Program	X	X	X	√	√	√	X	X	X	√	√	√	√	√
Check Availability	√	√	√	√	√	√	X	X	X	√	√	√	√	X
Register for a course														
Sign in	√	√	√	√	√	√				X	√	√	√	
Add to shopping cart	√	√	√	√	√	√					√	√	√	
Pay using														
Credit Card	√			X									√	
Debit Card		√			X							√		
Send Me Bill			√			X					√			
Logout	√	√	√	√	√	√					√	√	√	
Business Rule Mapping														
BR1	√	√	√	√	√	√					√	√	√	
BR2														
BR3				√	√	√					√	√	√	

Based on the feature set as set out it is possible that any one of the flows used to ensure coverage of business requirement 1 could as well serve for business requirement 2. However, on closer scrutiny the test case flow that tests non-happy path scenario of business requirement rule 2 requires a further elaboration of the test flows against feature set. Such gaps and inadequacies will come to light in a traceability matrix that is not granular and consequently, the test coverage falls short.

Tracing every non functional, business and non-business requirement to test cases and scenarios should increase the confidence and coverage of testing and QA activities that can be performed. The usability flows and concrete test cases that cover the requirements and needs can be formulated and with each iteration, targeted test cases could be identified to be run or executed to address within the specific build. Traceability is really multi-dimensional and to be effective QA artifacts, they have to transcend the various phases of the development process – initiation, elaboration, construction and transition. Further, it has to be a “living” artifact, one that is updated with each iteration.



Within iterations, a set of acceptance and regression tests have to be scheduled and performed to meet the exit criteria. Features and stories (in the form of cards) are planned in iterations in an agile methodology. With traceability matrix and mapping of the test cases to features, use cases and defects, optimum test planning assuring the software quality within each build/release becomes effortless and convenient.

By establishing effective traceability matrices, the tool helps to answer some of the following questions apart from achieving the traceability of requirements to design and construction of the software:

1. What test cases should be selected to run for the current build – verify fixed defects, regression suite for the current fixes, apart from the base code smoke and build-acceptance tests?
2. What impact does change in a specific set of non-functional and functional requirements have on the QA testing process in arriving at test estimates?
3. How can defects identified be mapped to the requirements that the iteration was scoped to achieve? And what surround testing and re-testing have to be carried out to validate before the defects can be closed out or new but related ones identified?
4. What change requests were brought about by the most recent build or iteration and what impact on quality does this new change entail?

Conclusion:
Establishing and maintaining traceability provides a hidden but valuable benefit – one of serving as a tool for planning the testing tasks in the iteration during iterative development. Traceability also establishes tracking back to the exact requirements being implemented in the iteration improving coverage and confience in the quality process. This is of greater significance in agile projects where requirements documentation isn't complete as requirements continue to evolve with each build or iteration. Agility ensures the process (and the product) *is* adaptive to changing requirements and using traceability for QA activities ensures that verification keeps up with these changes.

References:

1. Kurt Bittner, Ian Spence, *Use Case Modeling*, Addison Wesley, 2003.
2. Alistair Cockburn, *Writing Effective Use Cases*, Addison Wesley, 2006.
3. Dean Leffingwell, *Applying Use Case-Driven Testing in Agile Development*, StarEast 2005.
4. Dean Leffingwell, Don Widrig, *Managing Software Requirements – A Use Case Approach*, Addison Wesley, 2003.
5. Jim Heumann, *Generating Test Cases from Use Cases*, The Rational Edge, E-zine for the rational community, http://www.therationaledge.com/content/jun_01/m_cases_jh.html, June 2001.

6. Peter Zielczynski, *Traceability from Use Cases to Test Cases*, http://www.ibm.com/developerworks/rational/library/04/r-3217/, 10 Feb 2006.
7. Brett Pettichord, *Agile Testing What is it? Can it work?*, www.pettichord.com, 2002.
8. Ian Spence and Leslee Probasco, *Traceability Strategies for Managing Requirements with Use Cases*, Rational Software Corp. white paper, 1998.

About the authors:

Venkataraman "Venkat" Moncompu‡ works as a Project Manager for Intellisys Technology, LLC. Having a Master's degree in Engineering, he has over 12 years of IT industry experience having worked in various capacities as a Developer, Solution Designer, Business Analyst, Project Testing Coordinator and Project Manager.

Sreeram Gopalakrishnan± works as a Project Manager for India Intellisys Technology (P) Ltd. Having a Master's degree in Business Administration, he has over 12 years of IT industry experience and is a certified PMP. Has experience working as a QA analyst, Business Analyst, Practice lead and Project Manager.