The Adapted V-Model: A Practical Approach to Agile Testing

Yasser Ghanim

Software Engineering Competence Center (Egypt)
E-mail: yghanim@itida.gov.eg

ABSTRACT

Agile methods largely remove the distinction between phases and roles with no predefined order of activities. Some agile approaches such as Test-Driven Development (TDD) switch the traditional order of coding and testing and largely remove the boundary between the two implying that testing is becoming more a technical responsibility of the developers! Testers are complaining of loss of identity and scope within agile teams. In parallel some modern non-agile testing approaches such as Risk-based testing also open the boundaries between testing and analysis and suggest more involvement of testers in business and system analysis and sometimes system architecture. This article devises an adapted version of the V-Model for Scrum and suggests a practical approach that aligns the modern testing approaches with agile methodolgies. Scrum will be taken as a representative for Agile. The proposed approach is developed based on the Testing Process Improvement Guide (TPIG) developed by the Software Engineering Competence Center.

Keywords: Agile, Testing, Process Improvement, SCRUM, ISTQB, V-Model, TPIG.

1- INTRODUCTION

Testing has become a well-established discipline with the emergence of many testing standards and models. The main concern of testing is assessing product quality and isolating defects. This is an agreed goal whether in traditional or Agile setups. Traditionally, testing is viewed as the safety net applied toward the end of the development lifecycle to protect the end user from potential product failures (product/quality risks). Testers were advised to plan for testing as an independent activity from product construction (design and coding) and to provide separate testing role/function for objective and independent evaluation. However the Agile methods largely remove the distinction between roles and functions inside the development team. Even when cross-functional teams are formed, team members are not assigned distinctive roles but are rather referred to as The Team (as per the Scrum methodology.)

In 2001, a group of individuals agreed on a common set of values and principles which became known as the Manifesto for Agile Software Development or the Agile Manifesto [1], [2]. The Agile Manifesto contains four statements of values:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

The Agile Manifesto argues that although the concepts on the right have value, those on the left have greater value.

Some aggressive quality-oriented agile approaches such as Test-Driven Development (TDD) in the XP methodology switch the traditional order of coding and testing and largely remove the boundary between them. Jonathan Kohl, a known blogger on software testing writes on his blog: "I welcomed Agile Development, and have championed it now that I've experienced it. I didn't feel a threat to my job as a tester, but I knew things were going to change. The only thing that bothered me was a new impression towards testers that seemed to be emerging. The attitude sounded like: we're doing testing now thank you very much, so we don't know where you will fit in Agile projects" [3].

Testing and development are tightly related, but the interaction and boundaties vary between development lifecycles. Testers must understand the differences between testing in traditional lifecycle models (e.g., sequential such as the V-model or iterative such as RUP) and Agile lifecycles (e.g. Scrum, XP, Pair Programming…etc) in order to work effectively and efficiently.

One main aim for this article is to show how tester skills fit in agile projects. The paper discusses a practical approach that aligns modern testing approaches and techniques with Agile development and devises a clear process framework for such alignment in the form of a modified V-Model for agile. The widely accepted ISTQB syllabus and its Fundamental Test Process will be taken as reference for modern testing approaches. Scrum which is the most widely spread agile approach will be taken as a representative for Agile methodologies. The article negates the hypothesis that testing as a role is demolishing or receiving smaller weight in agile teams. In contrary it assigns higher weight to skilled testers within agile development.

2- INDUSTRY ANALYSIS

This section shows how both testing and scrum certifications and trainings are sought by the software engineers in Egypt as an indicator for the industry shift toward modern testing as well as agile methodologies. The paper observes an increasing demand on proper alignment between modern testing and agile development. Testing training and certification have witnessed great growth in Egypt in the past five years. We will base our analysis on the ISTQB training and certification in Egypt as ISTQB has become the most used reference for SW testing.

Since the first ISTQB training arranged by the Software Engineering Competence Center (SECC) in 2009 to a limited number of test engineers, hundreds of testers are now certified against the different ISTQB levels. The growth reflects the need for modern and systematic testing approaches to deal with the

ever increasing business and technical complexity. In response to this, SECC founded the Egyptian Software Testing Board (ESTB) which is the local arm of the ISTQB in Egypt.

For Agile, demand is also evident. More companies and individuals are seeking agile services and training. Testers are now required to align with companies requirements to work within agile teams.

This section depicts the growth/trend in seeking both ISTQB and Scrum courses and certifications. Data is based on SECC records. Data are collected till end of August 2014 which means 2014 figures are subject to increase.

2-1 TESTERS DEMAND ON TESTING CERTIFICATION

Nearly two thousand test professionals from all levels have taken the ISTQB foundation level exam in Egypt through the ESTB (additional unknown number might have taken the exam online, but it is expected to be marginal.) Little more than half this number succeeded in obtaining the certificate.

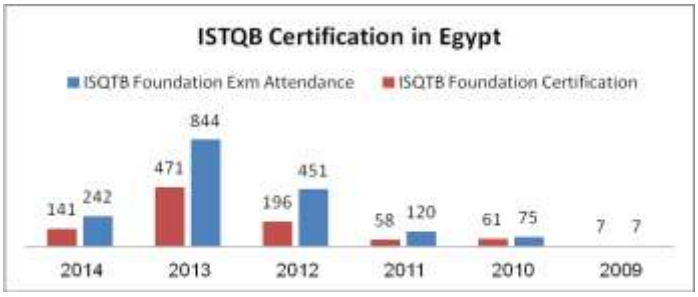

Figure 1. ISTQB Certification in Egypt

In parallel, SECC is offering training in a number of selected testing topics offered in greater depth than the ISTQB standard training. Testers demand on testing courses offered by SECC is shown in this chart:

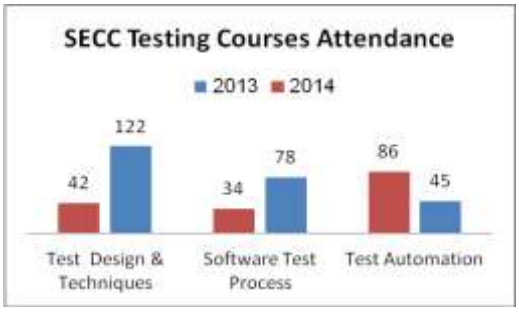

Figure 2. SECC Testing Training Courses

It is interesting to see that Test Automation is ranked as number one testing course in 2014, with the big role Test Automation plays in agile development as the paper discusses below.

2-2 DEMAND ON AGILE CERTIFICATION

In the other dimension of Agile/Scrum training and certification, agile professionals' demand on the relevant courses and certifications offered by SECC is shown below:

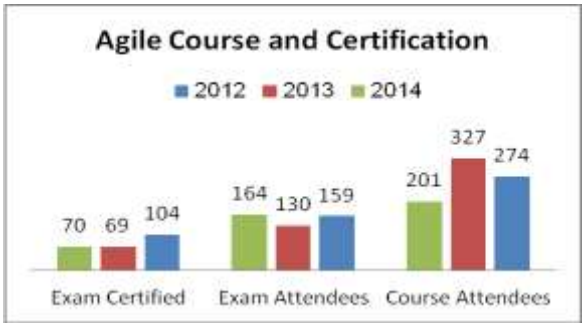

Figure 3. SECC Agile Courses and Certificates

For companies seeking agile services, SECC records shows that since mid 2012 SECC provided Agile consultation service to around 30 Egyptian software houses including some of Egypt's leading software houses with 200+ professionals.

SECC training records do not show the percentage of testers among the agile trainee and certificate seekers. However the ISTQB provides some useful data on the percentage of testers interested in Agile.

Among the software professionals interested in Agile, more testers are becoming interested in Agile Testing. As per the research conducted by the ISTQB in May 2014, 64% of the surveyed test engineers are interested in Agile Testing certification. The following question was asked to test engineers and test managers: "would you be interested in Agile Tester Certification?" The answers were as follows with 64% of test engineers and 63% of test managers answering with yes:

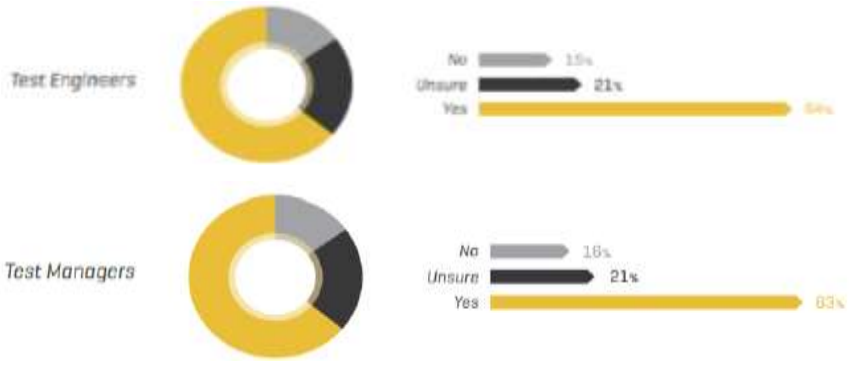

Figure 4. ISTQB Testers Survey

We can conclude that Agile development in general and Agile testing in specific are dragging more inertest and witnessing an increasing demand by software engineers including test engineers. In parallel modern testing whether within traditional or agile contexts is also a hot topic, and test engineers understand the need for more systematic and more aggressive testing approaches to deal with the quality challenges posed by the complex modern applications and complex business needs. The paper concludes high urgency and importance within the software industry in Egypt for aligning modern testing frameworks with Agile development. A reasonable assumption can lead to extending this conclusion to the software industry abroad.

3- TESTING IN TRADITIONAL DEVELOPMENT

One of the foundational software development models that emphasizes the role of testing is the V-Model which is the basis for many other SW Development and Testing models such as CMMI, TMMI and ISTQB. It provides the view of testing as a process that runs in parallel with development. It also relates the test execution phases known as test levels to product construction phases allowing for more systematic test preparation and also test participation in construction (through reviews and static verifications).

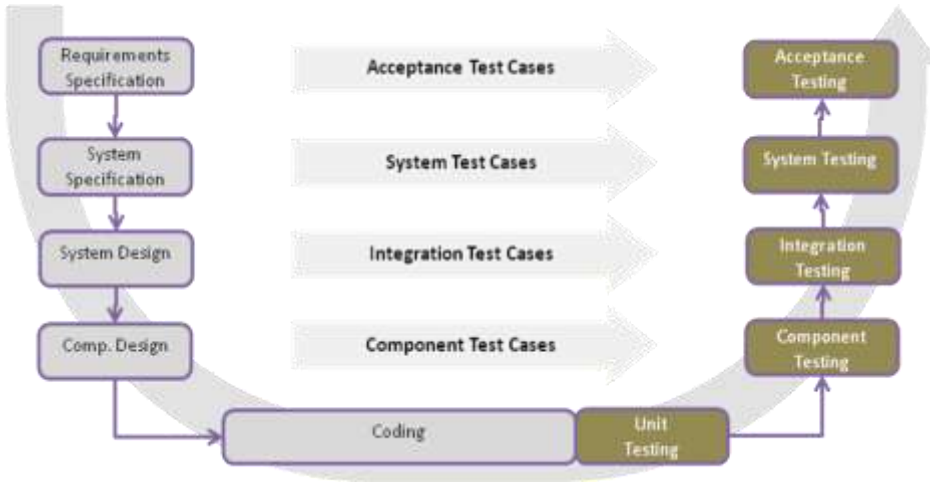

Figure 5. The V-Model

The V-Model is traditionally considered a Water-fall model not applicable to iterative development especially Agile. Part of the key testing concepts in the V-Model is test preparation as a separate phase from test execution.

The ISTQB syllabus sets a process model for testing known as the Fundamental Test Process which extends the concepts of prepration and execution and the notion of test levels to include more specialized testing approaches and techniques. Rsik-Based testing, test conditions, and Black-Box test design techniques are some examples.

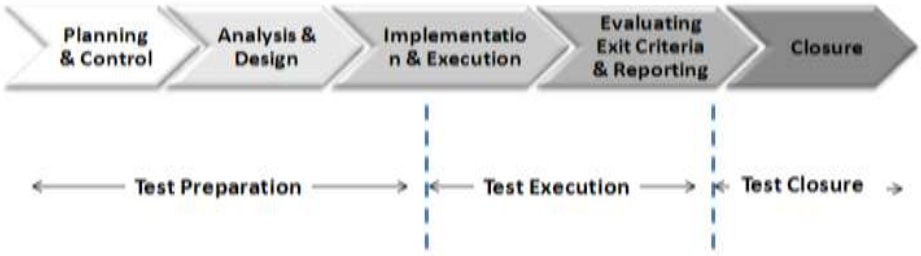

Figure 6. The ISTQB Fundamental Test Process

A variation of the V-Model is the W-Model which adds test preparation and review activities as another arm parallel to the construction arm.

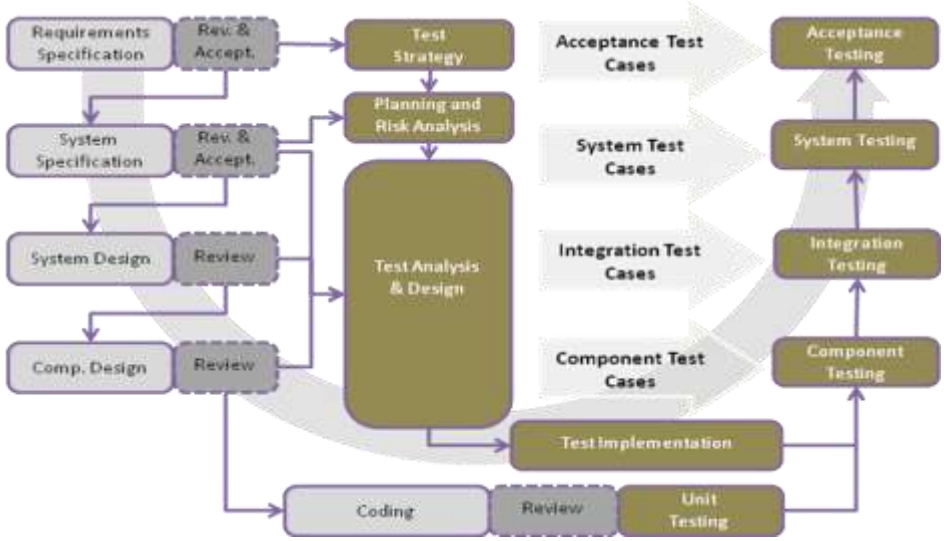

Figure 7. Test Preparation and Reviews in the W-Model

In the V-Model and traditional development testers have quite a well defined role with almost sharp separation between testing and other roles within the development team.

Table 1. Responsibilities in Traditional Teams

Activity/	**Traditional Development**		
Work Product	**Analyst**	**Developer**	**Tester**
Business Req.	Develop	-	Understand
Acceptance Criteria	Support	-	Develop UAT Test Cases
System Req.	Develop	Understand	Understand
Test Cases	-	-	Design Test Cases

Coding /Unit Testing	-	Code /Unit Test	-
Test Automation	-	-	Automate Test Cases
Test Execution	-	Debugging & Bug Fixing	Execute Test Cases
Completion	-	Deliver Builds	Evaluate Exit Criteria

4- UNDERSTANDING TESTING IN AGILE

Do testers really have to work differently in agile projects? What challenges meet them? And how can they adapt to agile testing?

One of the main difference between traditional and agile development is the idea of short iterations (sprints in Scrum). Testing is no longer an independent phase at the end of the project. An Iteration must result in a working (possibly shippable) product increment, which means well-tested components.

Testing iteratively poses challenges to testers which include (but not limited to): testing is no longer done in isolation and testers have to closely coordinate with developers, handling regression testing which leads to increasing testing load over iterations, and dealing with technical-oriented activities such as test automation.

4-1 TESTER'S ROLE IN AGILE

In Agile development the question of roles and responsibilities turns into a question of skill and experience. Agile sets no sharp distinction between roles and requires no specific team structure. It follows the whole-team approach where the team shares responsibility over project success. Thus the traditional separation of interests between quality and delivery is eliminated in agile. A single "done" definition is provided which embeds quality targets into the completion criteria for everyone. No team member can claim completion of tasks if the user story is not done according to the done criteria and is not yet "possibly shippable" to the customer.

The traditional dispute between quality and delivery interests owned by two distinct roles: delivery-oriented people (namely developers and PMs) and quality-oriented people (namely testers), is no longer valid. Accountability for both delivery and quality is equally shared. It is now a question of sufficing the necessary skills needed to achieve the "done" target than allocating different responsibilities to different roles.

The same thing applies to others. In agile teams (and supposed to be in traditional setups as well) quality is everybody's responsibility! As far as testing is concerned, everybody got to have a minimum level of testing and quality assurance skills. Such sharing of the quality concern is to be more integrated into the design aspects of the product and to a large extent built in the product design and coding from the beginning of the iteration.

Analysis (business analysis to some extent and largely system analysis) is

also another responsibility and skill set that is now passed on to “The Team” for more elaboration on the user story and for developing the story solution.

Therefore, the biggest challenge for testers when joining agile teams is to understand their participation in these areas and the amount of skills they need to acquire in the other skill sets. We can categorize the skills needed within agile teams into three main categories and propose a minimum set of shared core skills by all agile team members to cover the essentials of each area and allow for more collaborative efforts toward a well-engineered and defect-sanitized product:

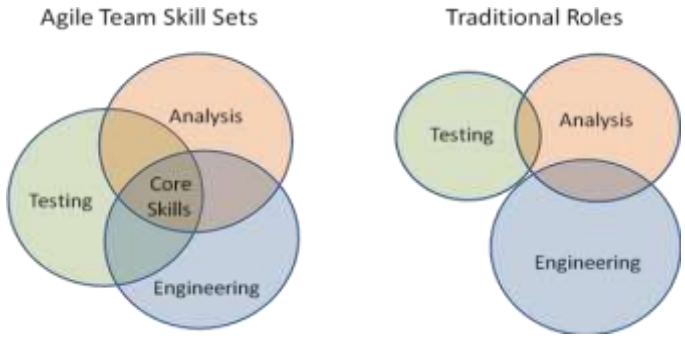

Figure 8. Testers Roles in Agile Vs. Traditional

Core skills include basic knowledge of the business domain, system analysis concepts, and general architecture and design understanding. For testers and developers: some basic coding (scripting) skill is needed for testers, and some test design techniques skill is required for developers. Testers might need to understand the impact of certain design patterns on system testability and other quality characteristics.

4-2 TEST-FIRST DEVELOPMENT AND TEST-DRIVEN DEVELOPMENT (TDD)

Test-Driven Development (TDD) means writing automated test drivers (scripts) which represent a unit test case ahead of development so that developers will start writing code against it with the objective of getting the script passed. Tests must start as failed with the absence of the corresponding code. As the unit code is developed tests start to pass. After passing their tests code units are refactored to improve their non-functional aspects keeping the tests passed. All unit tests must remain successful as any change occurs to any unit (see the continuous integration section below.)

TDD is largely considered a design and engineering practice rather than a quality control practice. However, in TDD quality is built in the product units from the beginning and testers can play an important role.

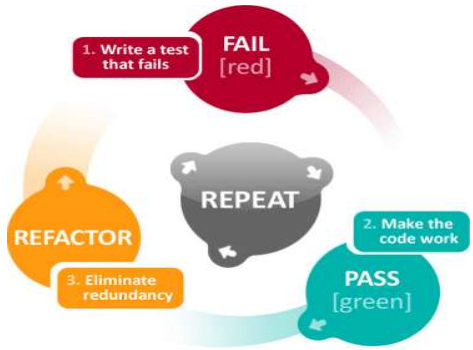

Figure 9. Test-Driven Development Cycle

In Agile development product modularity and maintainability is desirable to allow for more modifiable and change friendly code. Smaller parts of the product should independently function and provide value to end users. That means a code unit such as a class or a service will have clearer user value than was in traditional development which reduces the distance between system testing and unit testing. Units are no longer purely technical with no clear traceability to system test cases, and testers can very well help in specifying and designing unit tests. Testers will start their test case specification on the unit level taking into consideration the functional role of these units. Testers can proceed (based on their skills) in automating and scripting such tests and handing them to the developers. The practice helps developers understand the value and business context of their code units, as well as it helps testers to understand product architecture and code structure.

The TDD (if well collaborated between testers and developers) enforces code granularity with strong business alignment which largely enhances product maintainability and changeability than if done by developers alone.

Test-First Development

Test-first development is the manual version of TDD where test cases are not automated. It is a less aggressive approach and does not lead to the same level of granularity, but at least it ensures developing against very clear and detailed targets and criteria. In this approach test cases are written against user stories rather than individual classes. Test cases are considered as detailed examples for clarifying the user story requirements to developers.

Specifying test cases in this approach starts before but can overlap with development. Testers start by writing positive (happy scenario) test cases. As developers are busy developing against these positive tests, testers continue specifying the negative and exceptional test cases which can then get addressed as another round of developing the target user stories.

The test-first technique works very well with the test condition techniques discussed below.

5- RISK-BASED TESTING AND AGILE

Risk-based testing is one of the important modern analytical testing strategies. It introduces a new philosophy to the testing lifecycle. Traditional testing approaches depend on enumerating software requirements and designing test cases for the best requirement coverage. Requirements are treated equally and test execution is usually arranged based on First-In-First-Out basis with testing effort proportionate to development effort.

A product/quality risk means a potential failure (or failure category) that has direct impact to end user. Risk-based testing is defined by the ISTQB as an approach to testing to reduce the level of product risks and inform stakeholders of their status, starting in the initial stages of a project. It involves the identification of product risks and the use of risk levels to guide the test process [4]. This means the more harmful defects will receive more effort and more aggressive approach to detect and thus have a smaller chance to escape to the operational environment. Moreover, components with higher risk levels are supposed to be developed earlier in the project so they may be stabilized enough before the final deadline is reached.

The article proposes an important intersection between risk-based testing and agile development as discussed later.

5-1 RISK ASSESSMENT IN AGILE

There is an important intersection between risk-based testing and agile. User stories are prioritized in the backlog considering the product risk level as one of the prioritization parameters. Other parameters may include urgency to business or dependency on other user stories. Testers can contribute their risk-assessment skills in this stage.

Product risks need to be analyzed and assessed to assign an appropriate risk level. Risk levels are very often measured based on risk impact and probability. Impact means severity of the problem to the end user. The higher a failure impacts users' interests, the higher the effort testers must incur to prevent it. Measuring risk impact depends mainly on the type of the application and its operational environment. Analysis is needed to understand the role played by each function and its business sensitivity within the business domain. Such analysis requires domain experience (business analysis) and adds a lot of depth to understanding the business domain by the whole team. Such understanding is very important to testers as well as business and system analysts. It is an important intersection between testing and business analysis.

Analyzing non-functional risks such as performance and security risks require architectural knowledge and understanding of technology impact on these quality characteristics. Proposed Architectural solutions can be compared for their ability to reduce non functional product risks.

Risk probability means the likelihood of certain functional or non-functional failures. The higher the probability the higher the effort to be incurred to detect and fix this failure and prevent the risk. Failures with less probability will still

get adequate attention which will normally be less than the more likely failures. It is a common experience that failures' probability is proportionate to code complexity. Code and design complexity cause the probability to differ between two risks of equal impact. Probability may also be determined based on requirement complexity (e.g. calculations and mathematical equations are more likely to have mistakes made by analysts and developers.)

To understand the probability level of each risk, testers as well as analysts and developers must perform complexity analysis of the different system functions from both business and technical perspectives. The later will have to be done late in the development lifecycle when the technical architecture and design are available. Here comes more intersections with both business analysis and system architecture.

5-2 RISK MAPPING AND RISK PROFILE IN AGILE

Risks are then mapped to test objects to create a complete risk profile of the application. Such risk profile will then be the basis for most of the decisions taken by test leads starting from selecting the proper testing approach, to test estimation, to evaluating test results and approving test exit. Drawing risk profiles (after completing risk assessment) requires good understanding of the proposed software solution which intersects with system analysis. A new intersection between testing and development.

Adequate resolution of risks (often called minimization of residual risk) is to be part of the testing exit criteria and hence part of the done definition of any user story. Acceptance of user stories by Product Managers and Customers largely depends on the team success to reduce the probability of the identified risks below a safe threshold (probability can never get to zero). Assessment of the residual risk level and comparison to accepted thresholds is a skill contributed by testers to the rest of the team.

So we can conclude that risk-based testing supports the collaborative development encouraged by the agile methodologies and adds more rigor to a number of agile practices. We can list the following intersections between testing and the other disciplines within the agile teams as a result of the Risk-Based Approach:

Business Analysis

- Product risk identification and assessment workshops where both functional and non-functional risks are enumerated and studied for their impact.

System Analysis

- Requirement complexity is assessed for evaluating product risk probability.
- Product risk mapping and evaluation of their applicability to user stories.

System Architecture

- Product risk probability is revised based on architectural decision.

- Architecture is selected/optimized for risk level minimization.

Release Planning

- Stories are prioritized, sequenced and allocated to releases and sprints with risk taken into consideration.

Story Completion

- The done criteria of the user stories include reducing product risk probability and having residual risk below thresholds defined in the testing exit criteria.

The whole Risk-Based testing approach improves the whole team understanding of both the problem and solution domains and builds a shared vision.

6- TEST CONDITIONS AS AN ANALYSIS TOOL

Test Condition is defined by the ISTQB as "An item or event of a component or system that could be verified by one or more test cases, e.g. a function, transaction, feature, quality attribute, or structural element."

Test condition identification is an interim step between requirement understanding and test case derivation. Requirements are often represented in the unstructured form of textual description as in business requirements or user stories, or in a lightly-structured form as in use cases and operational scenarios. Unstructured requirements are often not concise enough and have room for ambiguity, omissions and under specifications as well as inconsistency, redundancy and over specifications. Requirement reviews may lead to considerable reduction in requirement anomalies. However, big percentage of these anomalies escapes review to reach code. The less structured the requirement representation the higher the chance for requirement anomalies. Structured representations may include lists, ranges, boundaries, equations, logical conditions, process diagrams…etc. Requirements that involve mathematical equations may not be expressed in textual format but have to be specified using concise operations (e.g. <, >, <=, >= …etc.).

Deriving test cases from unstructured requirements is a heuristic process and may lead to poor requirement coverage. Coverage cannot be guaranteed nor calculated for unstructured requirements. There is always a room for false assumptions or between-the-lines inferences. Moreover systematic test design techniques cannot work on unstructured inputs. For a technique to apply systematic steps to derive test cases, requirements must be provided in a predefined structure that can be interpreted by the techniques. For example input data ranges with clear boundaries can be used as input for the Boundary Value Analysis (BVA) technique.

Having this said, Test Condition is a powerful approach to bridge the gap between requirement and test case and allow for the application of systematic Black-Box Techniques.

A test conditions usually takes the form of an attribute that has one or more mutually exclusive alternatives. For example a user account can be either:

open, closed or suspended. An insured person could be either under 18 (Age <18) or from eighteen to sixty (18 =< Age <= 60), or older than sixty (Age > 60) where the "=" operation is clearly assigned to identify the exact boundary between age ranges. Without such level of specification there is always a chance for omitted account types (e.g. a closed account can be either expired or terminated), or ambiguous specifications of boundaries (18 is part of which range?, Should the system accept ages up to a maximum value such as 99?) Test conditions leaves very little room for ambiguous or unconcise requirements. A number of black-box techniques can apply on test conditions such as Equivalence Partitioning, Boundary Value Analysis, and Decision Tables to derive test cases with very well measured coverage of the underlying requirements.

Table 2. Test Condition Attributes

Test Condition Attribute	Description
Condition Name	Describe the general rule name such as Customer Age Bands or Account Type
Test Object	The system object (Screen, Service, Interface…etc.) on which the rule apply.
Requirement Reference	The user story, use case, or requirement item from which the condition is derived.
Condition Type	Conditions can be Boolean (True, False), Ranges, distinct values, or a constraint.
Alternatives	These are the different (mutually exclusive) cases for the conditions. Each condition can be an input range or a distinct value.
Expected Outcomes	The expected system behavior per alternative.
Related conditions	Other conditions that must be considered before determining the exact system behavior. E.g. the result depends on both customer age and account type.
Priority	Help determining the priorities of the derived test cases.

With the above examples of test conditions we can see that such level of structured specification although intended to serve test case design can also serve as an analysis technique on its own merits. Expressing requirements in such very detailed, concise, consistent, complete (…etc.) manner requires answering many questions and validating many assumptions. Should we consider age 18 part of the first or second age band? Should we have three or four account types? Should we reject age after certain limit? And so on. The one who can answer such questions is supposedly the customer or his representative such as the product manager. Such work can be very justifiably described as analysis work rather than testing work with considerable business and system analysis skills involved.

Test condition specification provides big intersection between analysis and testing, and also big collaboration point between testers and developers. It is a powerful approach in any project setup whether traditional or agile.

However, in traditional setups such practice is rarely recognized as analysis effort, and little coordination is done between testers and developers in both producing and consuming test conditions as they are often considered part of the test process that is invisible to developers. Moreover, they cannot be validated by the customer to whom the tester has no access!

In Agile, test conditions can be recognized as full-fledged analysis activity that elaborates on user stories and allows testers to act as complete analysts who interact with the product manager as well as the customer. Test conditions are validated by all stakeholders and are no longer inputs to test design only but rather consumed by the whole team to provide a solid shared ground for design and coding. Design aspects such as "should the age field allow for more than two digits" can be discussed and made between developers and testers during test condition specification. There are many design aspects that can impact how test conditions are specified. The same set of requirements can be designed for better testability and more optimized tests impacting how test conditions and test cases are specified. A workflow system that has a generic toolbar that appear in all screens will be tested in a different way and with different number of test cases than a customized toolbar that changes from one screen to another. Validations that are done on a field-by-field basis require different test cases than aggregated validations at form submission, and so on.

6-1 TEST CONDITIONS AND TEST-FIRST APPROACH

A more aggressive augmentation to the Test-First approach would be the use of Test Conditions. In this approach the analysis role of the agile tester mentioned abie gets extended to include test cases as well. Test cases serve as examples on test conditions for developers. The whole set of test preparation work products (test conditions and test cases) become the basis for the development work and provide the lowest possible level of analysis details that can ever be furnished to developers. Test execution becomes a matter of reassurance while most defects are avoided from the beginning.

7- REGRESSION TESTING AND TEST AUTOMATION

Regression testing is highly related to change. Code changes may introduce risks to already tested and stabilized components. Even with loosely coupled product architecture, both business and technical impacts are never eliminated. Business impact means inter-related requirements with change on one requirement entails revision to the other. Technical dependency results from shared code resources. Regression is a big burden on testers as it requires rerunning previously passed tests.

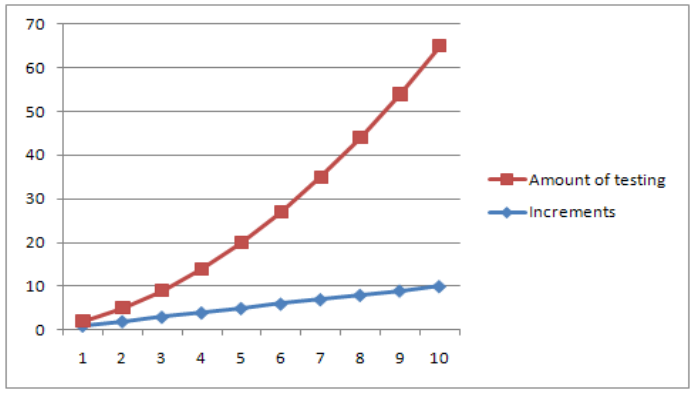

Figure 10. Testing Effort Growth per Iteration [5]

In Agile, regression risks are higher as change is introduced all the time. Even if no user stories are modified after being developed, the fact that some dependency exist between user stories, require retesting previously completed stories with each new iteration. That means the more we proceed with new iterations the more old user stories will possibly subject to regression. That makes the regression suite grow over time to an extent that could consume a whole iteration and the team velocity will get much slower by the time. The only solution to this situation is Test Automation (TA). Testers (and developers) are supposed to automate test cases (of all test levels) as they execute them for the first time (often manually) so little manual regression testing will be spent in future iterations.

As mentioned later in the Continuous Integration section, TA suites whether for unit, integration or system tests can be included in the continuous integration process. So regression results can be reported to developers instantly (or daily) after code submissions.

The TPIG product suite provides guidance for TA planning as well as designing TA suites and frameworks [6]:

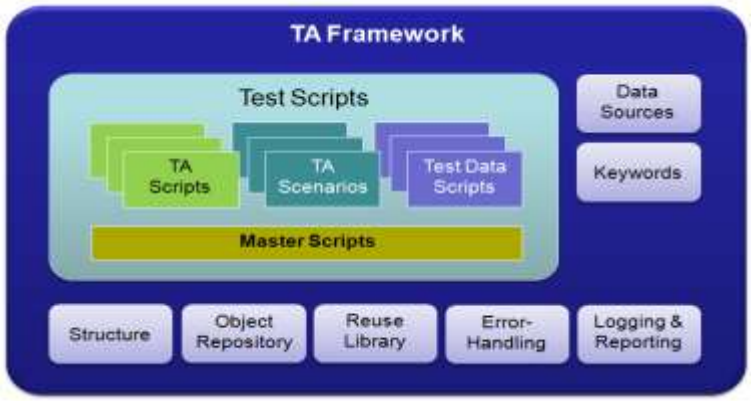

Figure 11. TPIG Test Automation Framework

Test Automation requires considerable planning and preparation effort and often faces technical challenges such as the use of custom controls and complex programming interfaces. This makes Test Automation an important area for collaboration between developers and testes. It is possible that developers focus on unit test automation while testers focus on UI/System Test automation. But the more collaborative approach would be to work jointly toward all types and levels of test automation, resulting in more efficient and well designed automation suites and also continuous integration suites.

8- THE ADAPTED V-MODEL FOR SCRUM

Many perceive the V-Model as a competing model to Agile development, very few proposals were made to align the V-Model to agile methodologies. Below are two examples.

8-1 PREVIOUS STUDIES

Among the few examples is a proposal by Mark Monteleone President of Monteleone Consulting in 2013 on modernanalyst.com [7] that maps the V-Model phases to the Scrum activities.

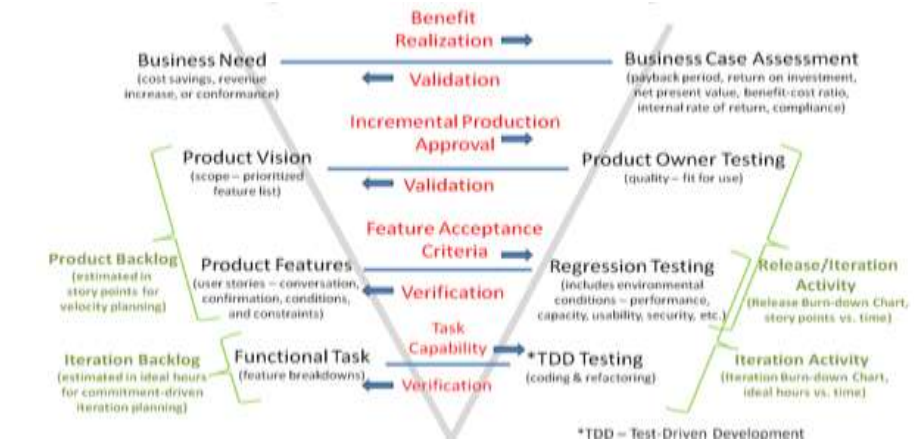

Figure 12. Mark Monteleone V-Model for Agile Testing

The given mapping is useful for showing the applicability of mapping the V-Model to the Scrum activities. As implied by the above figure, the V-Model phases do not have to apply to development projects in the same frequency. Phases can occure before or inside iterations. Which implies a kind of “spinning” V-Model that spins with iterations but at different rates at its different levels (the idea is clarified further below.) However this proposal is very high level and does not adopt the modern testing standards and techniques. Testers still need to see how testing processes such as the ISTQB fundamental test process or SECC TPIG [8] can get aligned with Scrum.

Another attempt was made by Dublin Institute of Technology in 2013 in the context of Medical Device software development projects [9].

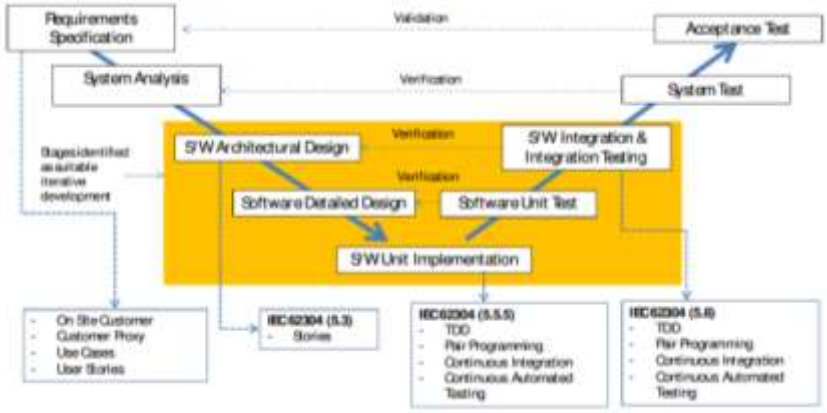

Figure 13. Dublin Institute of Technology AV-Model

This proposal covers part of the V-Model phases and proposes them as suitable for iterative development. The previous criticsm applies to this attempt too.

8-2 THE PROPOSED MODEL

This paper proposes an adapted version of the V-Model for Scrum that aligns test preparation and test execution activities (as defined by the ISTQB) with the activities proposed by the Scrum methodology. The adapted V-Model for Scrum highly integrates the test preparation activities into the product construction (Analysis/Design/Coding) activities.

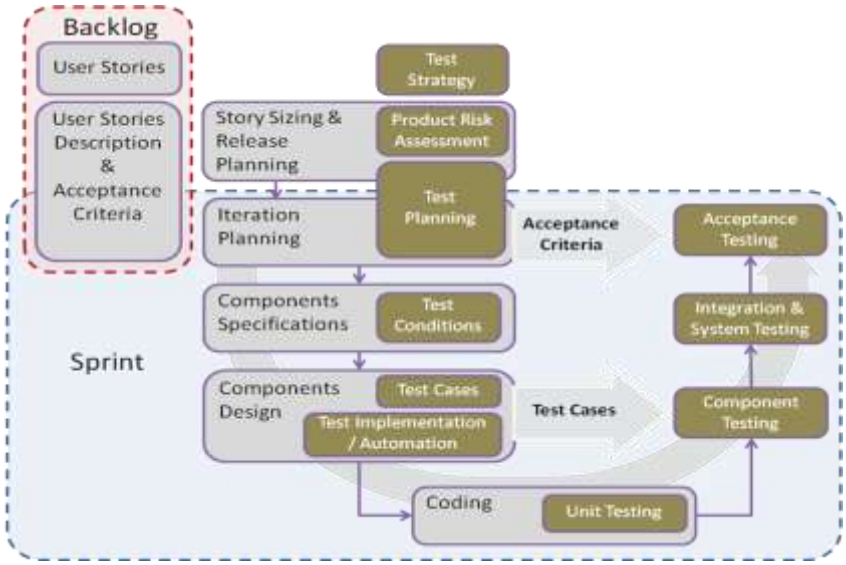

Figure 14. The Adapted V-Model for Scrum

In the adapted V-Model, product components are specified, designed and coded with the whole team working hand in hand. Testers participate in the daily stand-ups, get involved in the architectural and engineering discussions, and start acquiring more engineering knowledge. Testers' quality concerns get incorporated into product and components design from the beginning. Reviews (formal or informal) become a culture with the whole team providing useful inputs.

Dealing with defects becomes more proactive. More defect prevention takes place than defect detection and testers' effort shifts more toward preventive techniques. The agile techniques of Test-First and TDD can be easily applied by basing coding on test case design (Test-First) or on automated tests (TDD) where in both cases testers guide the development effort, not the other way around, and quality is built-in from the beginning.

8-3 THE MINI-WATERFALL TRAP!

A sprint is not a mini-waterfall, as taught by Agile Coaches! Waterfall implies sequential steps with little collaboration. A sprint follows a lean approach where task sequencing is based on inherent dependencies rather than predefined phases. Stories might get completed at different points of times during the sprint. Some activities might be shared between stories and some are not. Testers and developers work in parallel toward completing all activities. There are almost no activity within the sprint that does not require the involvement of both skill-sets of testing and development. One more practice that helps eliminate the mini-waterfall pattern is Continuous Integration.

8-4 CONTINUOUS INTEGRATION

Continuous Integration (CI) streamlines the transition between coding and testing and allows for parallel activities with almost zero lead time between coding/fixing and testing/retesting. In contrary to daily or sometimes weekly builds between development and testing as in the more traditional setups, builds are made available to testers as new code unit is submitted. In CI, configuration management, compilation, software building, unit testing, and deployment are wrapped into a single, automated, repeatable process [10]. Even integration and regression testing can get included through Test Automation.

Combining automated unit tests with CI makes code submissions and builds instantly verified so they can get corrected by the submitting developer. Build verification and entry criteria to testing become largely automated and build frequency can be very high with little risk to quality.

Following developers' coding, debugging, and check-in of code into a shared source code repository, a continuous integration process consists of the following automated activities [10]:

- Static code analysis.
- Compilation, linking, and generating executable files.
- Unit testing and checking code coverage.

- Deployment into test environment.
- Integration testing
- Regression testing.
- Results and status reporting to the team.

Continuous Integration frees testers from focusing on Build Verification Tests (BVT), smoke tests, and solving build problems to focusing on defining an effective CI strategy. They can add to the design of CI processes by adding automated integration and system test cases. Testers can also contribute by providing requirements for static code review for the non-functional aspects of the code such as performance and security.

8-5 THE ADAPTED V-MODEL FOR SCRUM – COLLABORATIVE VERSION

The following diagram shows a more improved version of the adapted V-Model for scrum which emphasis the collaborative approach:

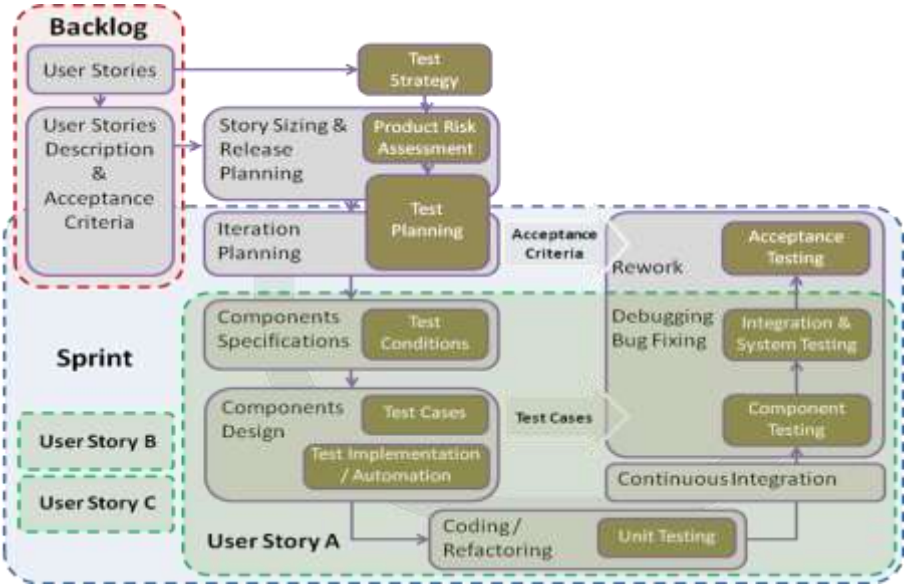

Figure 15. The Adapted V-Model – Collaborative Version

In this version of the adapted V-Model, working on user stories overlap and the mini-waterfall trap is avoided. Testing is highly integrated with development. Testers and developers are closely coordinating their micro-steps. Potential collaboration includes, but not limited to the activities' listed in the following table:

Table 3. Tester-Developer Collaboration

Activity/Work Product	Tester	Developer
Component Specification	Support/Develop Coordinate with Customer/ Product Manager	Develop

Test Conditions	Develop	Support and Review
Test Cases	Develop	Review
Test Automation Suite	Develop	Support/Develop
Component Design	Reassess Product Risks Improve Test Cases	Develop
Automated Unit Tests	Support/Develop	Develop
Code	Improve Unit Tests	Develop Improve Unit Tests Refactoring
Component Testing	Execute	Debug/Fix
System & Integration Testing	Execute Analyze Dependency and Re-gression	Debug/Fix Support Analysis
Acceptance Testing	Customer/Product Manager Coordination	Rework

9- PROPOSED TESTER RESPONSIBILITIES IN THE AGILE

Based on the above discussions we can summarize tester's responsibilities in Agile projects as proposed by this article as follows:

Table 4. Proposed Tester Responsibilities in Agile

Activity/ Work Product	Traditional	Proposed Approach
Business Req./ User Stories	Understand	Review User Stories Ensure Functional/Non-Functional specification.
Test Strategy	Develop	Develop Identify Product Risks Collaboratively
Acceptance Criteria	Develop UAT Test Cases	Review Acceptance Criteria
Project/Release Plan	Commit!	Assess/Map risks to user stories Risk Point Estimation Contribute to user stories priorities and sequence
Test Plan	Develop	Develop as part of Release/ Sprint Plan
System Req.	Understand	Develop Test Conditions
Test Cases	Design Test Cases	Design Test Cases
Coding / Unit Testing	-	Design Unit Tests Guide coding (TDD)
Test Automation	Automate Test Cases	Automate Test Cases
Test Execution	Execute Test Cases	Execute Test Cases Participate in designing Continuous Integration suites.
Completion	Evaluate Exit Criteria	Calculate Residual Risk Satisfy the "Done" definition

We can see that this approach assigns testers greater responsibilities than thought before which means the weight assigned to professional and skilled testers in agile is very high and can even be higher than traditional development.

10- BENEFITS OF THE PROPOSED APPROACH

In general, the early involvement of testers in the development cycle, their clear participation in Business/System analysis, and the close collaboration in design/coding, leads to quality being built in product from the beginning rather than assessed and controlled at the end. The essence of the V-Model is early testing and early defect detection/removal which are highly evident in the proposed approach. The "throw-over-the-wall" syndrome where one role passes its ill-completed output to the next role is eliminated (such behavior was often responsible of another syndrome known as the "squeeze-against-the-wall" where testers often get stuck with over squeezed testing time and a non-movable delivery wall!) The final result is assumed to be reduced Defect Density (DD), reduced Cost of Quality (COQ), and improved Defect Removal Rate (DRR). More measures include improved Project Schedule Variance (SV) and Project Cost Variance (CV).

Little empirical data is available on the DD and COQ measures. However, in the number of agile companies that implemented this TPIG approach considerable DRR improvement was measured with also high SV and CV improvements. The DRR is calculated as follows:

DRR = Pre-Release Defects / (Post Release Defects + Pre-Release Defects)

The following graph is based on real TPIG implementation at SECC customers during the first round of TPIG implementation in 2010/2011. It shows considerable improvement in three different measures (DRR, SV and CV).

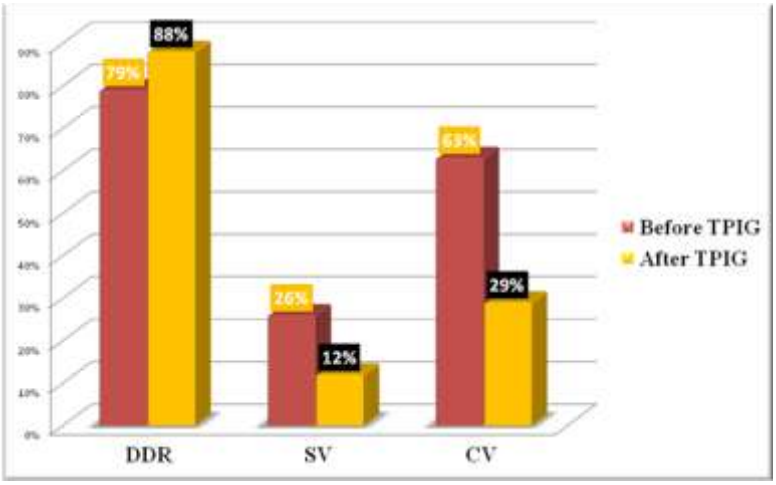

Figure 16. TPIG for Agile Improvements

The Figure shows around 9% DDR improvement before and after the adoption of the proposed approach which means 9% of the overall product defects (around 42% of the production defects) are now discovered by testers rather than users. A great quality gain based on a single process improvement initiative.

For the COQ measure, the ASQ (asq.org) defines it as follows:

> **COQ = Prevention Costs + Appraisal Costs + Failure Costs (Internal and External)**

Based on the empirical analysis performed by SQS Company in Europe over 5000 projects completed in the past fifteen years, the defect failure cost is reduced by at least 50% by moving defect detection from production to testing.

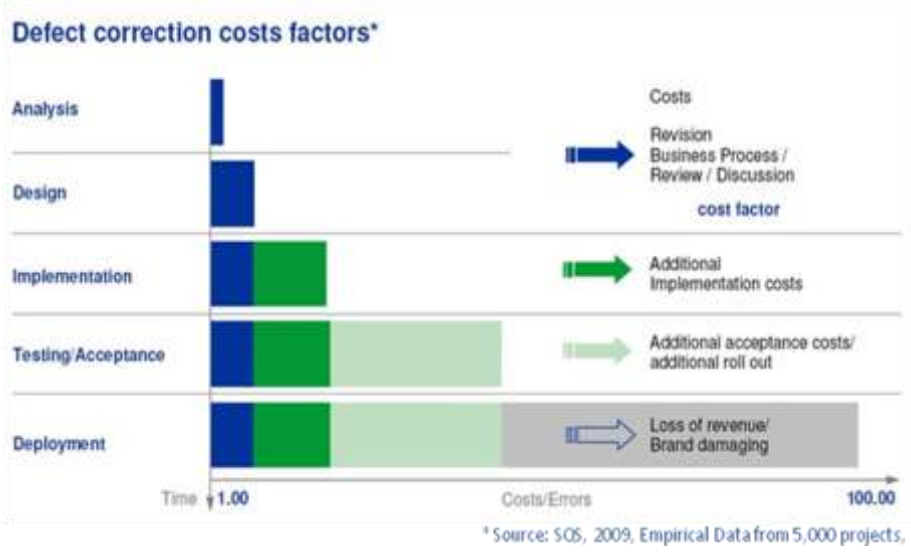

Figure 17. SQS Cost of Defect Curve

Combining this with the 9% DDR improvement shown in the above case study and assuming 100 EGP average cost per production defect, and 50% cost reduction in pre-production defects, the following results are gained for each 100 defects:

a) Before TPIG Implementation:

Failure Cost of Production Defects = 21 * 100 = 2,100

Failure Cost of Pre-Production Defects = 79 * 50 = 3,950

Total Failure Cost = 2,100 + 3,950 = 6,050

b) After TPIG Implementation:

Failure Cost of Production Defects = 12 * 100 = 1,200

Failure Cost of Pre-Production Defects = 88 * 50 = 4,400

Total Failure Cost = 1,200 + 4,400 = 5,600

c) Failure Cost Saving:

Failure Cost Saving per 100 defects = 6,050 – 5,600 = 450

Failure Cost Saving Percentage = (450/6,050)*100 = 7.4%

The 7.4% figure is the absolute minimum as it assumes the DDR improvement is based on defect detection during the test execution phase only. Considering improved defect prevention (early detection) the ratio can grow to anywhere between 7.4% and around 14% (detection cost during analysis is only 1% of the production cost)

Further reduction could be calculated should we have data related to Defect Density (DD). Defect correction (debugging and bug fixing) as part of failure costs is reduced through the reduction of defect density in the product as a direct result of early defect detection.

In addition to the above quantitative improvements the following qualitative improvements are assumed in different areas:

Management Benefits

1. Time-to-Delivery is improved by taking the test-rework-retest cycles to the minimum.
2. A unified understanding of scope and requirements improves collaboration and reduces conflicts among the team, and requirement assumptions are largely eliminated.
3. Testers' analytical skills are leveraged for the benefit of the whole team. There is little room for ambiguous, incomplete, inconsistent or non-testable requirements.
4. Test-First and TDD lead to better traceability of requirements to code units.
5. Risk assessment during story writing leads to better priorities. It allows the product manager to wisely set a proper release plan to maximize risk mitigation and minimize residual risk at interim or final product releases.

Analysis Benefits

1. Test conditions, positive/negative tests with clear expected result, test design techniques, requirement coverage measures, all augment business and system analysis by additional analysis levels that result in clean specifications with small chance for defects and false assumptions.
2. The practice of risk assessment and risk mapping is another addition to analysis. For example, identifying localization issues (multi language support, currency exchange…etc.) during risk assessment workshops will lead to additional rounds of analysis for these areas and hence the development of more complete and accurate requirements.

Engineering Benefits

1. In addition to providing complete and concise input to developers, agile testers develop better understanding of system architecture and design and their impact on the different quality aspects (functional or non-functional). This will lead to improving architecture and design over time and taking more enlightened engineering decisions.
2. Trade-offs between architectural and design alternatives better considers impact on quality attributes which can stem from the team accumulated experience in resolving functional and non-functional product risks.
3. Testers may start contributing to design reviews and static code analysis. They can also provide a lot of functional value to unit tests by making them more purposeful.

11- CONCLUSION

Modern testing concepts, approaches and techniques are not in conflict with agile methodologies. In contrary they can perfectly fit within agile development and foster the fulfillment of the agile values and principles. Approaches such as risk-based testing and test condition strongly support agile teams and allow for more collaborative effort toward analysis, engineering and quality control. It also supports developing shared vision and common understanding between the team members and other stakeholders. Tangible management benefits are also achieved in avoiding the mini-waterfall trap and reducing team conflicts.

The V-Model which is a dominating model behind many other development process models such as the CMMI for Dev. can be adapted to agile methodologies to carry on its very same benefits as in traditional development. The V-Model will serve as carrier of the modern testing techniques (risk-based, test conditions, Black-box techniques…etc.) to the world of agile development. Effort has already been made in implementing the adapted V-Model model through SECC Testing Process Improvement Guide (TPIG). Case studies that prove the benefit of the approach are already established and some data are gathered and analyzed to prove the case. More data are still to be collected for the different testing measures such as the DDR, DD and COQ.

The conclusion is that testing is not a downgraded role in agile. The misconception that agile teams do not need specialized testers is to be cleared. Testers play very powerful role in agile but with some adaptation and skill development toward business analysis and engineering. Testers need also to utilize their analytical skills for the whole team benefit, as well as providing support to product engineers which will require acquiring/developing some understanding of product architecture and design. Scripting skills are strongly needed for test automation in addition to testing the non-functional test types such as performance.

REFERENCES

[1] Agile Alliance Guide, Various contributors, guide.Agilealliance.org [Last accessed: June 2015]

[2] Agile Manifesto, Various contributors, www.agilemanifesto.org [Last accessed: June 2015]

[3] Jonathan Kohl, http://www.kohl.ca/2003/testers-on-agile-projects/ [Last accessed: June 2015]

[4] International Software Testing Qualifications Board (ISTQB): Certified Tester Foundation Level Syllabus, version 2010.

[5] Hans-Eric Grönlund: www.hans-eric.com [Last accessed: June 2015]

[6] Y. Ghanim, G. Aly, “A Practical Approach to Test Automation,” EuroMed SPI conference, 2011.

[7] Mark Monteleone, A Proposal for an Agile Development Testing V-Model, http://www.modernanalyst.com/Resources/Articles [Last accessed: June 2015]

[8] Y. Ghanim, H. Osman, G. Aly, “A Practical Approach to Functional Testing for SMEs,” EuroMed SPI conference, 2010.

[9] M. Mc Hugh, O. Cawley, F. McCaffery, I. Richardson, X. Wang, “An Agile V-Model for Medical Device Software Development to Overcome the Challenges with Plan Driven SDLCs,” 5th International Workshop on Software Engineering in Health Care (SEHC), pp. 12-19, 20-21 May 2013.

[10] International Software Testing Qualifications Board (ISTQB): Foundation Level Extension Syllabus Agile Tester, version 2014.