IEEE std 829-2008 and Agile Process– Can They Work Together?

Ning Chen

Department of Computer Science, California State University, Fullerton, California, USA

***Abstract* -** IEEE **Standard for software and system test documentation** (i.e., IEEE std 829-2008) is a comprehensive guide that specifies a common framework for planning the testing activities. The agile process is known for its promotion of frequent delivery of working software over comprehensive documentation and responding to change over following a plan. Although the IEEE std 829-2008 has strong association with the traditional waterfall development process, it does offer flexibility that allows user to combine or eliminate some of the test documentation content topics. Furthermore, it does not prohibit short-term and incremental planning. The underlining philosophies of the test standard and agile process are not at odd. This paper attempts to investigate whether they can be married and work together to great effect.

Keywords: IEEE std 829-2008, Agile

1 Introduction

One measurement of the importance of testing is the cost associated with it. Some industry survey reveals that between 30 and 50 percent of the cost of development is spent on testing [1]. Since any modification of the software, even a simple change, may inadvertently break the whole software, testing will not stop even after the end of the development. For this reason alone, having quality test documents during and after the development phase to support testing activities becomes essential. Instead of inventing quality test documents, one can easily find templates from IEEE std 829-2008 [2] that offers a general framework for needed test documents. Professionals coming from traditional waterfall development camp embrace IEEE std 829-2008 wholeheartedly due to the fact that the standard indeed has a deep root in the waterfall community. Time moves on and nowadays, agile process with a philosophy of working software over extensive documentation comes into the picture [3]. The arrival of agile stirs up two important questions. The first question is that do we still need to have standard test documents when using agile as the development and testing process? If the answer for the first question is affirmative, we have a follow-up question on hand– can IEEE std 829-2008 and agile development/testing process work together? This paper starts with a review of IEEE std 829-2008 and agile development and testing process. An analysis and comparison of IEEE std 829-2008 and Agile is followed. Our answer to the question we raised is affirmative. We, then, propose a way of integrating IEEE std 829-2008 to a variant of agile (Scrum) with some insights we contemplated. The paper ends with a conclusion section that summaries with our findings, insights and suggestions.

2 What is IEEE 829-2008?

We start our discussion on IEEE829-2008 with one of its main goals of “establish(ing) a common framework for test processes, activities, and tasks in support of all software life cycle processes, including acquisition, supply, development, operation, and maintenance Processes.” [2] As we noted in the introduction, the goal of establishing a common framework for test processes, activities, and tasks is the key that motives us to see whether this common framework can work with the agile development and testing process. The standard comes with 132 pages in length and is not that easy to comprehend. We feel that the entry point of unwrapping this not-so-small document is the understanding of the consequence-based integrity level scheme promoted by the standard. The standard says that there are four integrity levels:

Level 4–Catastrophic
Level 3–Critical
Level 2–Marginal
Level 1–Negligible

The descriptions of level are:

Level 4 (Catastrophic) -Software must execute correctly or grave consequences (loss of life, loss of system, environmental damage, economic or social loss) will occur. No mitigation is possible.

Level 3 (Critical) - Software must execute correctly or the intended use (mission) of system/software will not be realized causing serious consequences (permanent injury, major system degradation, environmental damage, economic or social impact). Partial-to-complete mitigation is possible.

Level 2 (Marginal) – Software must execute correctly or an intended function will not be realized causing minor consequences. Complete mitigation possible.

Level 1 (Negligible) - Software must execute correctly or intended function will not be realized causing negligible consequences. Mitigation not required.

Most readers will not have any difficulty on accepting this consequence-based integrity level scheme, after all, the descriptions are very easy to understand and they are quite reasonable and acceptable. In terms of what documents are required at each level, the standard says that:

Level 4: 10 test documents
Level 3: 10 test documents
Level 2: 8 test documents
Level 1: 7 test documents

It is a bit surprising to see that there is not too huge difference between levels. No difference (counting number of documents) between Level 4 and Level 3. The main difference between Level 3 and Level 2 is the adding of two so-called Master Test Plan and Master Test Report. The adding of the master plan and report probably is due to the desire to give stakeholders some long-term (in the context of time) and global (in the context of scope) view and awareness of what's going on. The difference between Level 2 and Level 1 is the adding of a so-called Level Interim Test Status Report. The adding of the interim report most likely is driven by the idea that the stakeholders may need to know the status of the project more frequently (shorter time period). Although the small difference as the level goes up is a bit unusual, the increased frequency of reporting and the more long-term planning and broader view as level goes up are quite expected.

What are those 10 documents (maximum number for Level 3 and 4)? The standard specifies the following:

Master Test Plan (MTP)
Level Test Plan (LTP)
Level Test Design (LTD)
Level Test Case (LTC)
Level Test Procedure (LTPr)
Level Test Log (LTL)
Anomaly Report (AR)
Level Interim Test Status Report (LITSR)
Level Test Report (LTR)
Master Test Report (MTR).

All users of the standard have no problem on forming an intuitive understanding of the term "plan, design, case, procedure, log, and report." The term "master" is also quite straightforward. The only curiosity one may have is on the definition of "level." What is the definition of the term "level"? Is it related to the term "integrity level" in some way? A careful reader of the standard may soon find the following:

(T)he word "Level" is replaced by the organization's name for the particular level being documented by the plan (e.g., Component Test Plan, Component Integration Test Plan, System Test Plan, and Acceptance Test Plan).

After further readings, a reader may encounter the following:

Other possible examples of levels include operations, installation, maintenance, regression, and nonfunctional levels such as security, usability, performance, stress, and recovery. Any one of the example levels may be more than one level for an organization; e.g., Acceptance testing may be two levels: Supplier's System and User's Acceptance test levels.

At this point, most of the readers of the standard can easily come to the following realizations:

1. We are not talking about 10 documents – it actually is 10 different *kinds* of documents. Depending on the actual project (and the replacement of the term Level by other terms such as Component, Integration, System, and Acceptance), the total number of documents may easily explodes.

2. For those who are familiar with the V model shown in Fig. 1 [4], they may immediately feel that IEEE829-2008 maps to the V model almost perfectly. For example, in the V-model, it talks about Unit (component) testing, Integration testing, System testing and Acceptance testing that mirror to the Level Test Plan/Design/Case/Procedure/Log/Report mentioned in the IEEE 829-2008 directly.

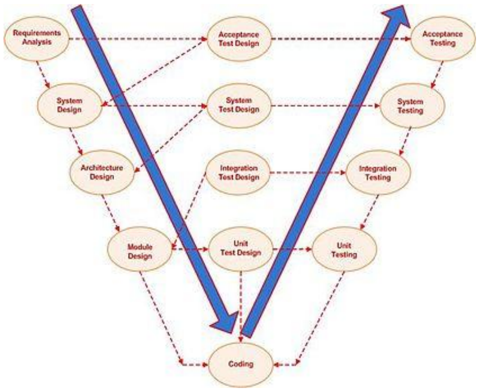

Figure 1. The V-Model [4]

To end our discussion on IEEE 829-2008 in this section (and to provide convenience to the readers of this paper), we decide to include a brief description of those 10 different kinds of documents as follows:

Master Test Plan (MTP) - There can be only one MTP for a project. The MTP identifies how many levels of test are required

Level Test Plan (LTP) - it covers scope, approach, resources and schedule of the testing activities and identifies the items being tested, the features to be tested, the testing tasks to be performed, the personnel responsible for each task, and the associated risks.

Level Test Design (LTD) - it specifies features to be tested, approach refinements, test identification, feature pass/fail criteria and test deliverables.

Level Tests Case (LTC) - it identifies inputs/outputs for each test.

Level Test Procedure (LTPr) - it covers the description of the steps to be taken to execute the test cases.

Level Test Log (LTL) - it provides a chronological record of relevant details about the execution of tests.

Anomaly Report (AR) - it documents any event that occurs during the testing process that requires investigation.

Level Interim Test Status (LITSR) - it summarizes the results of the designated testing activities and optionally to provide evaluations and recommendations based on these results.

Level Test Report (LTR) - it summaries the results of the designated testing activities and to provide evaluations and recommendations based on these results.

Master Test Report (MTR) - it summarizes the results of the levels of the designated testing activities and to provide evaluations based on these results.

3 What is Agile?

Like most researchers in software engineering, we start our discussion on Agile by quoting the Agile Manifesto [2]:

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 method of software development is built on a series of iterative development cycles where a set of features or user requirements are the basis for each iteration. The process is repeated until all requirements are delivered in the released software. The Agile framework is based upon the Value and Principles of the Agile Manifesto

We also would like to quote the Twelve Principles of Agile [5]:

1. *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
2. *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*
3. *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
4. *Business people and developers must work together daily throughout the project.*
5. *Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*
6. *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*
7. *Working software is the primary measure of progress.*
8. *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.*
9. *Continuous attention to technical excellence and good design enhances agility.*
10. *Simplicity--the art of maximizing the amount of work not done--is essential.*
11. *The best architectures, requirements, and designs emerge from self-organizing teams.*
12. *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.*

Next we would like to summarize some insights reported in literatures on the agile process from several aspects:

From the aspect of Test Documentation [6]:

- *Agile is not an excuse to not providing test documentation.*
- *Agile does say that huge volume of test documentation most likely is counter-productive.*
- *From the Manifesto – "Valuing working software over documentation" does not mean that test documentation is not valuable.*
- *Agile encourages test documenting early and often.*

From the aspect of testing [1]:

- *To get working software, it must be tested.*

- *To know if it was tested properly, there should be some test documentation.*

From the aspect of processes and plans [7]:

- *Agile means that individuals should make conscious decisions that react to changing situations. They should not just follow rigid plans.*

From the aspect of timing of the documentation [7]:

- *In agile we write test case for each iteration. We get feedback from stakeholders and then write test cases for the next iteration.*

4 Is it possible integrating IEEE 829-2008 to Agile Process?

At a first glance, we may conclude that IEEE 829-2008 is an alternative expression of the V-model and demands great number of documents. Since the V-model follows purely the waterfall process, integrating a waterfall model to an agile process is, of course, futile. This first glance, in our opinion, is a fallacy. A careful analysis reveals that there is a time-line expression embedded in the V-model. The left leg of the V implies a sequence of events that happened at a sequenced time line. The bottom of the V indicates the midpoint of the process and the right leg, again, shows a sequence of events in a time-line manner. Does the IEEE 829-2008 dictate any time-line fashion? The answer is no. The IEEE 829-2008 does tell us *what* documents to produce [8][9][10]. Nonetheless, it never tells us *when* to produce those documents, nor it tells us *how* to produce those documents. One may still argue that IEEE 829-2008 is so heavily documentation oriented. There is no hope of integrating it into the agile process in which we value simple or even no test documents. Again, we believe this argument is a fallacy too. Clearly, a careful reader can find the following description that shows the flexibility of the standard [2]:

Users of this standard may choose to add, combine, or eliminate whole documents and/or documentation content topics based on the needs (and integrity level) of their individual systems.

As for the argument that agile tends to end up with simple or even no test document, our counter argument goes as follows: Since any software project eventually ends up with spending 30 to 50% of its resource and budget on testing, a decision to produce (using any process) simple or even no test documents does not make business sense. Lastly we wish to argue that the IEEE 829-2008 focuses mainly on *what* to produce, not on *when* to produce, and not on *how (in the context of process)* to produce test documents. On the other hand, the Agile Process focuses mainly on *how* to produce, for sure, not on *what* to produce. We really don't see any inherent barriers in integrating *what* and *how* together to achieve a greater effect. Our answer to the question asked in the title of this paper – "IEEE std 829-2008 and Agile Process– can they work together?" therefore is affirmative.

5 Our attempt on integrating IEEE 829-2008 to Agile Process

Of course, the devil is in the details. As a reader of this paper, you may demand to see the details on integrating IEEE 829-2008 to an agile process. We present our attempt as follows. For simplicity, in this paper we focus our attempt on one variant of agile process (i.e., Scrum) only. Figure 2 [11][12][13] shows a typical Scrum process.

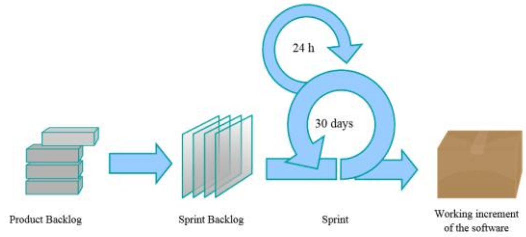

Figure 2 Scrum Process [13]

First we would like to briefly describe the Scrum Process. The main difference between Scrum and traditional waterfall or V model is that the Scrum development is done in time-boxed efforts called Scrum sprints. At the beginning of each Scrum sprint, the team conducts a sprint planning on the goal of the sprint driven by some user stories or requirements. The duration of the Scrum sprint typically varies from two weeks to a month. The important rule is that the team keeps a very close interaction at a 24-hour cycle called daily Scrum meeting and stand up. The goal of each Scrum sprint is to produce some working software. The desire of producing working software at the end of every Scrum sprint implies that each Scrum sprint needs to go through all phases of the software development life cycle. Since the testing is part of the software development life cycle, it becomes clear that testing must be one of the activities performed in each Scrum sprint. Agile promotes the iterative code development. Can test and test documentation also be iteratively done? We think the answer is affirmative. We argue that iterative test activities (in which planning and developing test documents are continuously refined and logging and reporting are continuously performed) can tag along with iterative code development seamlessly. Even after accepting the iterative test activities, a critic may still complain the excess number of documents required by IEEE 829-2008. How about the 10 different kinds of test documents (shown below again for convenience) specified in IEEE 829-2008?

Master Test Plan (MTP)
Level Test Plan (LTP)
Level Test Design (LTD)
Level Test Case (LTC)
Level Test Procedure (LTPr)
Level Test Log (LTL)
Anomaly Report (AR)
Level Interim Test Status Report (LITSR)
Level Test Report (LTR)
Master Test Report (MTR).

How do you weave those 10 kinds of test document development into Scrum sprints? Our attempt starts at the Level Test related documents first and address the Master Test Plan and Report later.

Level Test Plan (LTP)
Initially LTP can be roughly drafted at the first sprint planning meeting. In most sprints, level test plans may include ***unit test plans***, ***integration test plans***, ***system test plan*** and ***acceptance test plan.*** The main reason for having a complete set of level test plans (unit, integration, system, acceptance) in most sprints is that the goal of each sprint is to deliver a potentially shippable product by the end of each sprint. A shippable product indeed needs to go through, at least, unit test, integration test, system test and acceptance test [14]. Will a complete set of level test documents bogs down the sprint? We don't think so. In early sprints, although we need to work on a complete set of level test plans, every one of them, in fact, is very simple to begin with. Again, the rationale is that development plans are iterative and test plans will be developed iteratively as well. Those level plans are reviewed at every sprint retrospective meeting and revised as necessary.

Level Test Design (LTD)
Level test designs include unit test designs, integration test designs, system test design and acceptance test design.

Level Test Case (LTC) and Level Test Procedure (LTPr)
Level test designs include unit test cases and procedures, integration test cases and test procedures, system test cases and test procedures and acceptance test cases and test procedures.

Level Test Log (LTL) and Anomaly Report (AR)
Level test logs and anomaly reports may include unit test logs and anomaly reports, integration test logs and anomaly reports, system test logs and anomaly reports and acceptance test logs and anomaly reports**.** LTL and AR are continuously created, reviewed, and revised as needed during sprint.

Level Interim Test Status Report (LITSR)
Created and updated daily following daily scrum.

Level Test Report (LTR)
Level test reports may include unit test reports, integration test reports, system test reports and acceptance test reports**.** Most of those reports can be created and revised prior to sprint review meeting.

How about the **Master Test Plan (MTP)**? We propose that a Master Test Plan can be produced early in the project at sprint 0 to start the process. Later on, we could use the Master Test Plan to tie the Level Test Plans generated from each sprint together to create a final version of the Master Test Plan and Report. Sure enough, some of our readers may point out that what we have attempted is just to compress the whole testing life cycle into one individual Scrum sprint. Doing so will simply bog down each Scrum sprint and is totally against the sprite of Agile. There are two arguments to respond to such a criticism. First, if iterative code planning and development can be accepted/tolerated why not iterative test planning, design, and reporting? Second, if it becomes apparent that resources need to be reserved for other high priority tasks, we may also consider to combine some type of test documents which is certainly allowed by IEEE 829-2008. For example, in some small-size projects, one may combine Level Test Plan (LTP), Level Test Design (LTD) and Level Test Procedure (LTPr) into one document. Level Test Log (LTL) and Anomaly Report (AR) also can be merged.

6 Conclusions

In this paper our main goal is to convince our readers that integrating a testing standard such as IEEE 829-2008 to an agile process *should* be done and *can* be done. First, why it *should* be done? Our premise on “*should* be done” is purely based on business reasoning and is not related to what development process used (waterfall or agile). Any modern software product development requires, at the minimum, some testers’ participation. In some large organizations, having a separate department or team that works on software quality assurance is also not that uncommon. Furthermore, it is an industry consensus that testing eventually may consume 30 to 50% of all resources spent. Having spent and committed such a large portion of resources and personnel on testing but not demanding the ultimate fruit of testing (i.e., test documents) is simply beyond any business sense. If the premise on demanding quality test documents is valid, the desire to have standardized test documents (such as documents specified in

IEEE 829-2008) becomes not that to understand. In this globalization era insisting on one-of-kind, ad-hoc approach, in most business scenarios, proves fatal. The argument on "*can* be done" is a bit challenging due to some ill perceptions from both agile and waterfall communities. Our main defense is to point out that IEEE 829-2008 is *NOT* a mirror image of the V model. The standard does not have embedded time-line as in the V model and it mainly focuses on the notion of "what to produce." The agile process, on the other hand, mainly focuses on "how to produce." Integrating "what to produce" and "how to produce" is actually natural and logical. We further support our argument by providing an attempt in which we integrated IEEE 829-2008 documents to Scum agile process. The corner stone of this integration is hinged on the fact that at the end of each Scrum sprint a potentially shippable product is created. This fact implies that we should start a complete set of level test documents at the beginning of each sprint and incrementally improve them very similar to what we have done on the iterative development of source code.

7 Acknowledgement

The author would like to thank

1.the class of cpsc545 of the Master of science in Software Engineering (MSE) program at California State University, Fullerton for contributing some of insights mentioned in this paper.

2. the MSE research and development fund for partial supports.

8 References

[1] Burnstein, Ilene. "Practical Software Testing". NY: Springer-Verlag, 2003

[2] IEEE 829-2008 IEEE Standard for Software and System Test documentation

[3] Agile manifesto. Retrieved from http://agilemanifesto.org/

[4] The V-Model (Software Development). Retrieved from http://en.wikipedia.org/wiki/VModel_%28software_development%29

[5] 12 Principles of Agile software Development. Retrieved from http://www.agilityspeaks.com/capabilities/aboutus/agile-development/

[6] *Lisa Crispin, article titled "Agile Documentation",* 03/02/2011. Retrieved from http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=COL&ObjectId=16698&tth=DYN&tt=siteemail&iDyn=2

[7] Janet Gregory, Lisa Crispin, Book - Agile Testing: A Practical Guide for Testers and Agile Teams, Jan. 2009, Addison-Wesley Professional

[8] Glazer, H., J. Dalton, D. Anderson, M. Konrad, S. Shrum. "CMMI or Agile: Why Not Embrace Both!" Software Engineering Institute. November 2008.

[9] Curran, C. "Are Agile and CMMI Compatible?" CIO Dashboard. 25 June 2010. Retrieved from http://www.ciodashboard.com/it-processes-and-methodologies/agile-cmmi-compatible

[10] Shelton, C. "Agile and CMMI: Better Together". ScrumAlliance. 9 July 2008. Retrieved from http://www.scrumalliance.org/articles/100-agile-and-cmmi-better-together

[11] Agile Scrum. Retrieved from http://en.wikipedia.org/wiki/Scrum_(development)

[12] "Inside the Inbox". Retrieved from http://www.ecircle.com/blog/2011/08/04/enare-effectively-integrating-marketing-tools/

[13] Scrum (Development). Retrieved from http://en.wikipedia.org/wiki/Scrum_%28development%29

[14] Ship it! - Scrum's "Potentially Shippable" Product Increment. Retrieved from http://agilemakingprogress.blogspot.tw/2011/03/ship-it-scrums-potentially-shippable.html