A PUBLICATION OF THE INTERNATIONAL FUNCTION POINT USERS GROUP

MetricViews

ifpug.org

September 2019

Vol. 13 • Issue 2

MEASURING SOFTWARE QUALITY

inside

IN THIS EDITION

Here is a snapshot of the exciting articles you will find in this edition of *MetricViews:*

Understanding and Defining Quality for Your Organization
(By Phil Lew)

Phil Lew presents a framework for defining quality in an organization and then using that definition as the basis for determining measurements and metrics. With those measures in place, you can better understand whether or not you are moving forward or backward.

A Semiclassical Approach to Agile Quality Metrics
(By Dr. Raymond Boehm, CFPS Fellow)

Dr. Raymond Boehm shares an overview of techniques to ensure a software product meets quality standards. Techniques discussed include defect analysis, failed deployments, net promoter score and recidivism.

Considerations for Establishing Agile Quality Metrics
(By Joe Schofield)

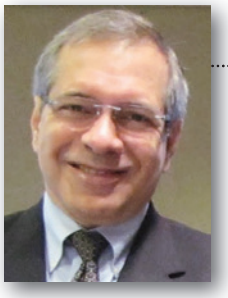
Agile includes a broad set of frameworks and techniques roughly bound together by a set of principles and a manifesto. Joe Schofield shares his thoughts on and experiences with the use of agile measurements and metrics for establishing agile quality metrics.

Agile Testing Methods: A Path to Improved Software Quality
(By Sheila P. Dennis, CFPS)

Sheila Dennis relates that Test Driven Development (TDD) executed within an agile framework facilitates higher quality software in less time. In addition, she shares her thoughts about supporting business needs with the positive aspects of TDD.

A Solution to Track and Move
(By Ankitha Pareek and Anupama Karal)

Ankitha Pareek and Anupama Karal introduce a DevOps Dashboard, which was designed as a solution to achieve cross-enterprise IT visibility. Learn how it provides visibility across all aspects of the IT lifecycle.



Mauricio Aguiar

Message from the President

Size Still Matters

As the years go by, I tend to think of the future of Functional Size Measurement as a way of quantifying software. Because software will probably be around for a long time, I can risk saying the size of software will also remain relevant for many years to come. Does this sound unlikely? Well, maybe not if you remember that "area"—measured in square meters, square feet, or any other unit you think of—has been around and relevant for thousands of years.

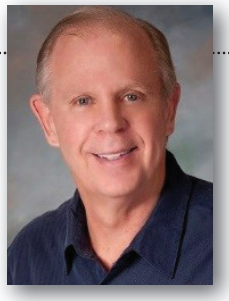
Software will certainly change and be applied to situations we may only dream of today, but it will still perform functions and keep data that those functions will use. Show me a function that uses data and I will show you a function point count!

I have seen wonderful presentations about function point counting new technologies such as IoT (think robots), games, and even weird things such as Second Life (remember that). Maybe enhanced humans will carry embedded software in the future and I can clearly imagine function points measuring that.

Without trying to preach to the converted, I'd like to think we measure software to estimate important variables such as the effort to develop a piece of software, the duration of a software development project, the defect density, and the productivity of software teams.

Needless to say, it should be no surprise that size remains relevant independently of the development method used—waterfall, agile, or whatever they come up with in the future. The amount of software functionality an organization is able to develop in a fixed period and how much that costs will remain relevant as long as money is relevant in a society. ■

Mauricio Aguiar
IFPUG President



David Herron

From the Editor's Desk

Phillip Crosby published his wildly successful book, *Quality is Free*, in 1979. Crosby's principle, *Doing It Right the First Time*, was his answer to the quality crisis. He defined quality as full and perfect conformance to the customers' requirements. His argument for quality being free was that an investment in improving quality pays itself back very quickly.

Where are we today with respect to improving software quality? I think Joe Schofield summed it up nicely with the opening statement in his article (in this edition), *Considerations for Establishing Agile Quality Metrics*. He states, "The quest for quality continues."

As the IT industry continues to embrace agile methods and frameworks, it is time we take a look at the issue of quality when using those frameworks. Do tried-and-true practices such as reviews and inspections still apply? Are one-time accepted measures such as defect density still in play?

We reached out to the IFPUG community to see if we could gain some insight into what IT shops are doing with respect to measuring and improving quality. We received a number of responses (more than we could publish in this edition). What we learned from these authors is that quality is still very much a part of the overall software development and deployment framework. Is there a perfect one-size-fits-all solution? Of course not, but the articles in this edition of *MetricViews* show the variety of possibilities for improving software quality. ■

David Herron
Communications and Marketing Committee

Understanding and Defining Quality for Your Organization

By Phil Lew

Abstract

Leadership often talks about software quality, but rarely investigates and determines what practices it will take to reach the destination let alone the destination itself. Do people in your organization discuss quality, but then when it comes time for concrete measures to improve quality, nothing happens? The reason this happens is that most organizations don't know what quality means to them. They say they want their customers or end users to be "satisfied," but what does that really mean? And if you can't define quality, then how can you even begin to track, measure and improve it? This paper provides a framework for first defining what quality is for your organization, and then using that definition as a basis for determining measurements. With measurements and metrics, you can then understand whether or not you're moving forward or backward as you race to attain velocity from sprint to sprint.

Why Measure Quality?

If you are going to trouble yourself with measuring quality, what benefits will you get? In agile, some don't want to think about it. They may say the only thing that counts is what customers or end users say. That is TRUE. We all care about the end result, just as when we go to the doctor and want a clean bill of health.

What is Quality?

The age-old question still remains and always will. The reason is that there is no standard definition of quality although many have tried to define it.

- William Edwards Deming (1900-1983): Known for his term "Total Quality Management" and PDCA (Plan, Do, Check, Act), Deming said, "Good quality is a predictable degree of uniformity and dependability with a quality standard suited to the customer. The underlying philosophy of all definitions is the same—consistency of conformance and performance, and keeping the customer in mind."
- Philip Crosby (1926-2001): Known for his books, *Quality is Free* and *Quality Without Tears*, Crosby said, "The definition of quality is conformance to requirements. The system of quality is prevention. The performance standard is zero defects. The measurement of quality is the price of non-conformance."
- ISO 25010 (2011): This standard defines quality as "the degree to which the system satisfies the stated and implied needs of its various stakeholders, and thus provides value."

> "Depending on your point of view and context, your idea or concept of quality will certainly be different from that of your CEO."

The major difference in ISO 25010 versus Crosby and Deming is that ISO 25010 sets out a more formal framework for evaluating quality depending on the "stakeholders." As we all know, there can be many stakeholders, each with different motives and goals. "Beauty is in the eye of the beholder." The same goes for quality. It's different for everyone. Depending on your point of view and context, your idea or concept of quality will certainly be different from that of your CEO, for example. Quality to the CEO most certainly will involve revenue and customer satisfaction. To the developer, it may include clean code. For a tester, it may involve finding (or not finding) defects. For the end user, it may mean something else. In accounting, accuracy to the penny for example. Once you've developed a general concept of what quality means to you, there are two steps to define it to the point where you can evaluate and improve it.

Decomposition

Complex concepts are usually best understood by grasping their smaller parts or components. We call this decomposition. For example, instead of cooking dinner, you break the meal down into a main dish with meat, fish, or poultry complemented by a vegetable side dish along with a starch such as rice, bread, potatoes or pasta. There are many decomposition paradigms or models. Examples are tree, mind map and sets. Each of the decomposition methods has its advantages and disadvantages depending on how you want to decompose quality.

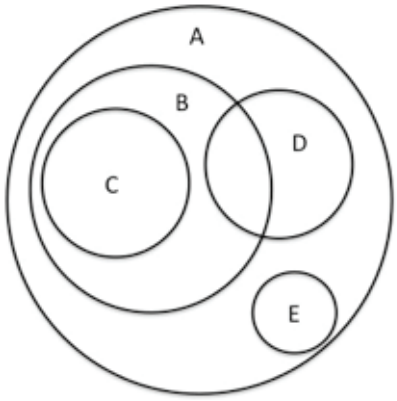


Figure 1. Set Decomposition

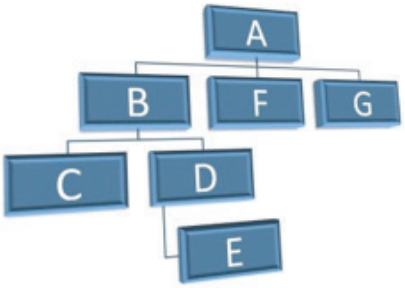


Figure 2. Tree Decomposition

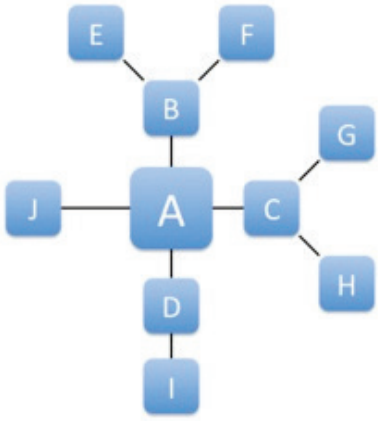


Figure 3. Mind Map Decomposition

Transition

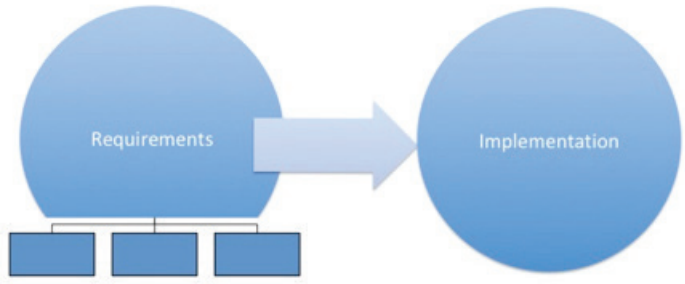
Once you've defined quality for you, your organization, and viewpoint, you can begin to grasp how to measure quality. The next part of your journey is to understand your process and

how one step in your process affects subsequent steps. For example, if you want to lose weight, naturally you measure your weight, but there are at least two things that you can and should measure before getting on the scale—your diet and exercise. Anything you do in the areas of diet and exercise will impact your measurement of weight at the end of the day, right? So why not measure your food intake and exercise rather than weight?

For software development, we all tend to count defects as a critical measurement of quality, but this measurement is similar to measuring your weight. What comes earlier in the process? Depending on your software development process, this could be requirements, design, grooming, user stories, etc. The question now becomes what measurements can you take before these other phases in your software development process that precede testing (finding defects). To answer this question, you need to examine your process.

Examining Your Process

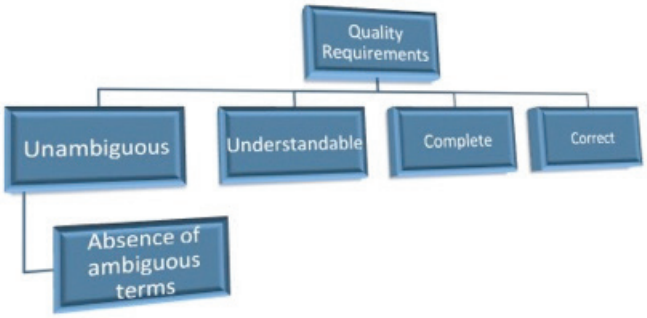
If you don't know where to start, start with defects. Of course, you've been tracking defects for years. Find your defects and examine their causes. This will lead you backward in your process. Go back and examine that cause (requirements, code logic). If the cause is code logic, was it because the code was incorrect or incomplete? If so, why? If the code is incorrect, this could be because the requirement or user story was written incorrectly, or it could be that the user story was written correctly but it was implemented incorrectly. If it was implemented incorrectly, was it because the user story was not accurate or incomplete? As you can see, it is possible to move backward or forward when examining your process and measure the quality of each work product each step of the way.



Once you've begun to map out your development process, you may begin to realize that requirements are a major cause of defects. Understanding that requirements have a major influence on code implementation (as do other influencing factors), you can then examine requirements at the next level.

Requirements

Many articles and research have pointed to requirements as the source of most defects, but not many will tell you specifically what a good requirement is, how to define it and how to identify it. In developing a definition for quality requirements, we decompose it into four elements—complete, unambiguous, correct, and understandable. Refer to Figure 2, the tree structure for decomposition.



Let's take a deeper look at ambiguity. We can further define ambiguity as the absence of certain words to increase clarity. For example:

- Efficient
- Fast
- Effective
- Compatible
- User-Friendly
- Straightforward
- Powerful
- Easy
- Reliable
- Normal
- Few
- Intuitive

Requirements are important, so give yourself plenty of quality checkpoints. For example, use grooming sessions as a quality checkpoint. While you are reviewing and updating user stories or requirements, note how many need revision and why. A defect in the user story at this point is a defect prevented later in the code.

> "Many articles and research have pointed to requirements as the source of most defects, but not many will tell you specifically what a good requirement is."

Defining Software Quality For You

As mentioned earlier, you can move backward and forward in your process as you examine different transitions and decompositions in your quality definition. Don't forget that much of the pressure on quality comes from customer or user expectations. So where does the user begin to form his/her expectations? If you are a software company, he/she may investigate your company through the internet to examine the features and functions of your product that are described online. If you have a demo, he/she may watch the video or perhaps get a trial version. Based on the demo and the information on your website, are your customers or end users getting a correct, complete and unambiguous understanding and, therefore, accurate expectation of your product? With the concepts introduced in this article, transition and decomposition, you can begin to work backward to map the end user's perception of quality to your internal development processes. By doing so, you can finally understand what quality is for you and your end users. ■

About the Author:

Philip Lew *is the CEO at XBOSoft, a firm specializing in software QA and software testing services. As a corporate executive, development manager, product manager and software engineer, Philip has managed teams to tackle broken processes, developed solutions to difficult problems, and coached others to be leaders, managers and experts. He leverages his academic background combined with hands-on work experience to work with clients and colleagues around the world. His hobbies include cycling and traveling the world to quench his thirst for exploration and learning. He can be reached via email at philip.lew@xbosoft.com.*

UPCOMING EVENTS

Round table on ISO 25000 [product/service quality] (Rome – Italy, Oct 30, 2019) - https://gufpiisma.wildapricot.org/event-3408818

BFPUG Metricas 2019 (Sao Paulo – Brazil, Nov 7, 2019) - https://bfpug.wordpress.com/conference-2019/

A SEMICLASSICAL APPROACH TO AGILE QUALITY METRICS

By Dr. Raymond Boehm, CFPS Fellow

Introduction

In the beginning of the 20th century, a group of physicists began experimenting with and publishing information about a new approach to physics called quantum mechanics. They thought it was a new way of looking at the natural world. It was. Some thought that it would replace the classical approaches that were already in use. It did not. Many problems had to be solved by using both elements of quantum mechanics as well as classical physics. This approach was called semiclassical physics. About 100 years later, a group of software developers began experimenting with and publishing information about a new approach to software development called agile development. They thought it was a new way of developing software. It was. Some thought that it would replace classical approaches that were already in use. It did not. A semiclassical approach must be taken to most software development activities, including the development of agile quality metrics.

At first, it may seem that software quality metrics should not be influenced by the techniques that were used. The user should not care how software is developed nor if it is fit for its intended use at the end. There is some truth to this. However, the agile approach to software development causes us to develop software in a series of relatively quick releases. There must be ways to monitor quality at every step to ensure that the product is of acceptable quality. This paper will discuss defect analysis, failed deployments, net promoter score (NPS) and recidivism. Some of these techniques are classic, others are classic but have been changed when used in agile projects and others are only used in agile development.

Defect Analysis

Tracking, fixing and analyzing defects have been software development activities since the very beginning. In classic waterfall implementations, defects were reported at every phase of the project. This continued after the software was deployed. After deployment, the number of defects found could be tracked over time. Usually, the number should decrease with time, but not always. Sometimes, more people are added to the user community and they detect more problems. In any case, the number of defects is a measure, not a metric. To be able to compare it to other projects, it must be normalized. It is often normalized by application size in function points.

Defect tracking is done in agile projects as well. In agile development, software is implemented in iterations. An iteration consists of all software development activities: analysis, design, coding and testing. It makes no sense to think about analysis defects, for example. The defects are just associated with the iteration. Iterations are normally all the same size. If an organization has committed to the idea of two-week sprints (an iteration in scrum), then all the iterations will be two weeks each. They may have different team sizes. Therefore, care must be taken in comparing the number of defects from one iteration to another. Even with these possible anomalies, defect count by sprint is a commonly used agile measure.

According to the 12 principles behind the agile manifesto, the team should "deliver working software frequently, from a couple of weeks to a couple of months." Many organizations violate this principle and take more than a couple of months on the first release. In any case, this means that not all iterations result in software being delivered to the user community. The iterations that do deliver software are referred to as releases. Defects that are found in these releases are called escaped defects. They need to be tracked and resolved just like they always have been.

Agile practitioners also have the need to normalize defect counts by size. One size can still be function points. Function points can be estimated based on user stories. However,

"Defect count by sprint is a commonly used agile measure."

they can only be estimated for releases. Function points are counted for delivered software. Iterations that do not deliver software to the user community cannot be measured with function points. There is another issue. Users have been developing user stories as part of the development process. It often makes sense to normalize the number of defects by the number of user stories that have been implemented in the iteration or release. There are caveats. User stories may vary in size. An iteration might implement a single epic story or several small user stories. Some iterations may only implement a portion of a story.

Story points is an estimating technique in which each user story is assigned an effort estimate. Some agile practitioners normalize their defect counts using story points. It is important to remember that story points are not measures of size, they are estimates of effort. Done properly, story points will be consistent across iterations and releases in the project. Because of the way they are derived, they cannot be expected to be consistent across multiple projects. They are certainly not consistent across different organizations. This means that the same story in different projects may have a different number of story points. Therefore, the number of defects per story point cannot be compared across projects or organizations.

Failed (or Unsuccessful) Deployments

Failed deployments have been an issue since the beginning of software development. Some organizations simply hope that they will not happen. Most organizations have contingency plans that include rolling back to a previous stable version of the software that failed. When deployments are done every few months, these incidents are anecdotal. Once they become frequent enough to track, the organization has a huge problem on its hands. Obviously, it then must be tracked and rectified. Some people call failed deployments unsuccessful deployments. This is like pharmaceutical companies that refer to death as a fatal event. Failed deployments are devastating to your team's reputation no matter what they are called.

In the agile world of frequent, or continuous, deployments, it is still necessary to keep this measure close to zero. However, now it makes sense to track them. It is easier for the team to lose track of the number of failed deployments. The users will still remember. The number of failed releases must decrease over time. Otherwise, rectifying this becomes the organization's top priority.

Net Promoter Score

Developers have long been familiar with the idea of customer satisfaction surveys. They are often part of outsourcing contracts or tools promoted by chief information officers who are attempting to prove the worth of their organizations. That is not what this is. The Net Promotor Score (NPS) gauges how likely users of an application would be to recommend it to others. The interesting twist is that it is measured before a release is delivered.

The point to remember about agile development is that an application is delivered in a series of releases. Each release must be usable to a certain extent. This is a theme that has appeared in many agile and lean contexts. In lean marketing, there is an emphasis on developing a minimal viable product (MVP). This is something that potential customers can touch and feel and use. From this, they will be able to understand the product that is being pitched to them and start to articulate the features for which they would be willing to pay. This whole concept has been taken from agile software development practices.

> "The Net Promotor Score (NPS) gauges how likely users of an application would be to recommend it to others."

Agile software releases must take less than one year, with many organizations pushing for six-month cycles or less. This means that the functionality of the first releases may be less than many stakeholders really want. However, it is a compromise. By pushing for the earlier software releases, the development team is getting experience with both the problem domain and the technical environment that they are working in. If there are problems, then they will surface sooner. This mitigates the risk of working on a software application for 18 months and then finding major misconceptions that make the software product useless.

As a result of the push for early delivery, some users may be unwilling to recommend that next release to a friend or colleague. This is understandable. Here, the key is to keep an eye on the trend. It is good when the NPS starts out low and gets higher with each successive release. If the NPS drops with successive releases, then the software product is diverging from what the users want.

The calculation of NPS is interesting. The only quantitative question is, "How likely is it that you would recommend this release to a friend or colleague?" The user utilizes a 10-point scale to answer this question. If he/she answers with a nine or 10, he/she is considered a promotor. If he/she answers one through six, he/she is considered a detractor. The NPR is calculated by subtracting the percentage of detractors from the percentage of promotors. Thus, the NPR is a value between -100 and 100. Positive values indicate that the promoters outnumber the detractors, which is good. A value more than 50 is considered excellent; over 70 is considered exceptional.

When users are asked to grade their willingness to recommend a product, there is usually one other question asked: "Why?" This is an open-ended question that does not lend itself to numerical analysis. However, reading through the answers will give the team a better idea of what users like and which direction their software development should take.

Some people have questioned the idea of measuring NPS before a release. NPS addresses the proposed feature set at a point in time. Other characteristics like reliability are captured by defect analysis and other metrics.

Recidivism

Recidivism counts the number of times that a user story re-enters a sprint. The concept of recidivism can be tricky. There is a woman who runs a bed and breakfast in New Hampshire. When the breakfast dishes are cleared, she analyzes what was eaten and what was left behind. She uses this information to constantly adjust both her menu and her recipes. She has decorated each room differently and beautifully. Her staff is trained to please the guests. People who stay there often return, which is good. In many ways, it is the opposite of the Middlesex County Jail. In the jail, guests are all given a one-inch thick mattress. Guests complain about the quality of the food and claim that the staff is mean. However, many guests return habitually, which is not good.

In classical waterfall projects, no real thought is given to recidivism. If a list of requirements is generated, each requirement is met and marked complete. If a requirement is not met, then it is considered a defect. In agile development, there are several reasons that a user story may make multiple trips through development. Good user stories are negotiable. For example, a story like "as a customer service representative, I can find customer payment history" can be implemented in different ways. In order to get an early release done more quickly, the user may agree that the representative can look up the payment history based on the customer's account number.

Once this software is released, the users may find it is unusable as most customers do not know their customer number. In the next release, the story must be reimplemented with the representative using a telephone number. Sometimes, a user story goes back into development because there has been a change in business requirements. Agile development welcomes this type of change. Unfortunately, some user stories go back into development because they were implemented incorrectly, and this went undetected before the release. Recidivism is always a cause for concern. For metrics in general, particularly recidivism, it is necessary to conduct a root-cause analysis to determine if there is a problem or not.

Conclusion:

In some ways, the conclusion is familiar to many software developers. Agile development is a lot more like classic development than most agile practitioners believe. Unfortunately, it is also much more different than many classically-trained managers had hoped. Like physicists who solved their problems 100 years ago, we must take a semiclassical approach to solving our software development difficulties. ■

About the Authors:

Dr. Raymond Boehm *is an agile estimating and measurement consultant. He is also a methodologist and researcher. He has been involved with software development since the 1970s and agile development since 2003. He is an IFPUG CFPS Fellow and a Quality Assurance Institute (QAI) Certified Software Quality Analyst. He has a Doctorate of Professional Studies from Pace University and an MBA from New York Institute of Technology. In addition to independent consulting, he has served as the Metrics Manager for the consulting division of Computer Sciences Corporation and held other positions involving system development.*

Considerations for Establishing Agile Quality Metrics

By Joe Schofield

The quest for quality continues. Manufacturers promote its importance. Consumers tend to benefit from it, and may pay more to ensure they get it. Speakers talk about it.[1] Authors write about it.[2] International standards are established for it.[3] Certifications are issued for it.[4] Organizations include it in their names.[5] A simple online search returns more than 7.2 billion hits on the word quality.[6]

Agile hardly stirs less interest, attention and scrutiny. Organizations claim their dominance in the community.[7, 8] Nearly everyone claims to be "doing agile."[9] Even "the fed" claims agile as its way of working.[10] Agile usage has spread well beyond IT as evidenced with 61% of marketing organizations using or planning to use it in their work in 2019.[11]

Despite all of the interest, no standard exists for agile today. Rather, we have a dozen or so approaches that claim a position in the agile market. Some of these frameworks capitalize on notions like iterative and incremental (concepts introduced in 1957 at IBM)[12], some on the popularity of products in the past.[13] A clear understanding of agile is further complicated by introducing quality measures across such a broad set of frameworks and techniques roughly bound together by a set of principles and a manifesto. Nonetheless, the following thoughts may advance the thinking of organizations exploring the use of agile measurements and metrics.

Most often quality is described as the product's "conformance to requirements." We could expand this definition to also include requirements for services, which are often codified in Service Level Agreements (SLAs). The quality measurement challenge begins here, since the first of 12 agile principles declares we "welcome changing requirements even late in

development." An agile mindset accepts the fact that traditional requirements churn is actually acceptable (embraced?) in agile frameworks and approaches (from here forward, agile will be used to include the set of frameworks like Scrum, Crystal Clear and DAD, and more targeted approaches like Test-Driven Development (TDD)). With frequently and constantly emerging and evolving requirements, determining which set of requirements to verify could be anything but straight forward. As iterative development and incremental delivery occurs, assessing conformance to requirements needs to recognize refinement and grooming as suitable change management activities. Comparing committed features to released features is a reasonable bound for assessing "conformance to requirements"—at least until the next release.

> "Establishing defect injection and detection meEsures should consider the iterative nature, the intentionally vague-to-better-understood nature of the agile work definition."

Using "conformance to requirements" as a potential definition for defects may require reconsideration as iterations and releases occur (while defects come in varying levels of types and severity, they are not the subject of this article; however, detailed analyses are available[14]). As a product owner shifts content priority in the product backlog and unceasingly grooms (changes, adds, merges, splits, deletes) specific requirements captured as stories, some which have been released earlier, defect clarity may become obscured. Acceptance criteria associated with new stories that introduce incremental change with the same feature may render previous non-conformances obsolete. As an example:

Release	Story ID	Acceptance Criteria	Comment
1	1	a – Attribute G can contain only red, or green	The chili[15] selection on a breakfast burrito
2	1a	a.a – Attributed G can contain red, green, or *none*	The restaurant owners never imagined a consumer not wanting red or green chili
3	1b	a.b – Attribute G can contain red, green, *both,* or none	The restaurant owner forgot about the New Mexico Christmas tradition of both red and green in the season though both are valid any day

While "both" and "none" are defects in Release 1, neither is in Release 3. The thinking that led to the acceptance criteria in Release 1 was incomplete (though close enough at the time), but acceptable with iterative development. What appeared to be a defect (entering "both") in Release 1 was not a defect in Release 3. Should we still count this as a defect or merely a benefit of iterative development? Do we expect (or demand) that regression tests are consistent with changes to the code? Does iterative develop tax traceability or merely justify its need?

This ongoing upheaval suggests a very different "requirements churn" in agile. The product owner has full ownership of the product backlog, changing it seemingly whimsically, arbitrarily, capriciously and ephemerally (WAC-E) (pronounced "whacky")). So then, establishing defect injection and detection measures should consider the iterative nature, the intentionally vague-to-better-understood nature of the agile work definition. And a final twist. What if in the example above, those three iterative cycles resulted in one release? Would we consider Story ID 1 and Story ID 1a to be defective? Purposely, the answer is left unanswered since the intent is to enhance our thinking about what quality means in agile before we begin to measure it.

While many organizations rely on testing, including full, daily regression testing for newly-integrated code, even world-class testing, will only remove as many as 50% of the defects in a product.[16] The other 50% or so were injected during requirements and design work; that is, story development and sprint task execution in an agile environment. Considering a sprint (or iteration) timebox for defect removal efficiency may hold promise[17] since defects can be tied directly to acceptance criteria and their associated story. This seems to be a simpler answer and is only useful once the considerations related to quality and defects during iterative development are better established.

Thus far, this article seems mostly to have offered cautions regarding quality measurements related to defects and requirements in agile efforts. Exactly right!

From Cautions to Suggestions

Improving quality, which should be the primary motive behind quality measurements, can still be driven with practices like peer reviews used in conjunction with Capture/Recapture Methods (C/RM). C/RM allow teams to predict, with statistical confidence, the remaining number of defects in a product in a peer review setting.[18] Refer to the IFPUG-cited source for step-by-step guidance of this quality enhancing technique. On agile teams, use peer reviews with C/RM on selected and critical, not all, product components. Consider swapping team members from other teams occasionally for peer reviews to cross-pollinate best-quality practices and to add an element of objectivity (account for this during sprint planning to avoid over-committing sprint commitments). Employing both design and code reviews might be worthy for inclusion in an

organization-level definition of "done." Requirement reviews are a natural part of product backlog grooming when performed by the product owner and the developers. Together, these reviews have a direct impact on the quality of work of products before testing is initiated, thereby helping to address the 50% of defects not subject to testing coverage.

A related suggestion is apropos. Resist any temptation to reward teams (certainly not individuals when working with self-organized teams in agile) for either defect detection or correction. Rewarding this behavior will inspire teams to create more defects in order that they may be discovered, and potential subsequent recognition for either total number of defects found or corrected. This caution is an example of a much greater warning: beware of unintended consequences associated with the introduction of any measurement system or resultant metrics. Instead, hold teams accountable for the product they produce. Since value delivery is a major thrust of agile in general, value lost or delayed as a result of defects might be a useful quality metric; that value delivery less value lost per release.

Beware of unintended consequences associated with the introduction of any measurement system or resultant metrics

Early in my career, I heard about an organization that was going to measure the number of calls received by its service center. Customers began complaining that their calls were seemingly dropped after a couple of seconds. The "manager" of the service center went to check on the team. He heard a phone ring, saw a staff member pick-up the phone and then return it to its base. The same staff person then tallied a mark on a sheet of paper. The manager said, "What is happening here. Callers aren't getting answers to their questions." The staff person looked at the manager and said, "We are being measured by the number of calls we receive, not the number resolved." The dumbfounded manager could only blame himself for the new dilemma and the metric "calls received." He quickly replaced it with "percentage of calls resolved on first contact." In turn, this change had a negative impact on the duration of calls (they took longer, not usually seen as positive) and the wait time for calls to be answered (as call center folks were taking the time to resolve issues). Over time, applying the queueing theory and optimization stabilized expected response times.

Rethinking the Importance of Quality Expectations in Agile Efforts

Quality is no stranger to agile development. Quality is minimally implied in the ninth agile principle, which states "continuous attention to technical excellence . . ."[19] Teams that use retrospectives are constantly addressing improvements and at least some of those will be related to quality. Grooming continually improves the quality of the product backlog content. I offer these as a few examples of quality inherent in agile work.

> "Teams that use retrospectives are constantly addressing improvements and at least some of those will be related to quality."

In an article released in April 2019, scruminc describes Schlumberger's use of Scrum resulting in defect reduction by about half, while also increasing productivity 25%, reducing headcount by 40% and reducing costs by 25%.[20] Reducing rework (defects) had a positive effect on productivity, which enabled Schlumberger to reduce headcount simultaneously. Historically, rework has been estimated to consume between 30% and 80% of software development cost.[21, 22] "Nailing" the requirements at the start of the sprint was a noted contributor to the Schlumberger turnaround.

In a study released in May 2019, speeding delivery, managing priorities, increasing productivity and aligning with the business all took precedence over enhancing software quality as motives for adopting agile. Improving quality was ranked even lower, ninth, as the perceived benefit of adoption. However, improved quality was listed second as a success objective with DevOps transformations. The same report found defect reduction eighth behind other agile success measures like C-Sat, value delivery, velocity, burndown and story completion.[23]

The State of Scrum survey reported a similar theme. Value delivery (71%) and responsiveness (56%) were selected over quality (44%) as most valued by executives. "Quality of life" received high scores (more than 80%) by Scrum practitioners. Seventy-seven percent of respondents expect to continue using Scrum in the future.[24]

Oddly, quality was not even mentioned in one recent project management annual survey.[25]

Don't Forget the Function Point Angle

IFPUG practitioners and researchers have often calculated defects per function point as a quality metric.[26] Function points are not a natural by-product of agile story-based requirements. Comparisons between well-defined function points and inconsistent (as intended) relative measurements using story points

aren't usually productive. However, in 2013, a case study proposed the use of elementary processes as a "common denominator" since they are found naturally in function point analysis, and are conceptually a worthwhile parallel in agile story decomposition.[27] To the extent that organizations find value in measuring defects per function point, that metric might bridge more traditional requirements definition and agile stories.

Steps Forward

One can reasonably expect the topic of quality, as it relates to our product deliveries, to continue for some time into the future. With most organizations using agile today and agile's expanding acceptance in numerous other industries, it may be time to reset the discussion on why software and other products are developed and released. Most of us willingly acknowledge a preference for quality over costlier, less useful, life-limited and defect-ridden products. Quality must therefore be an aspect of the value delivery, and is so highly-evidenced as crucial in this article. Simply stated, delivery without quality is of little value.

Delivery Without Quality is of Little Value, Agile or Not!

As you participate in assessing quality in your organizations, the closing list seems self-evident in agile organizations:

- Set expectations with all stakeholders as part of the product visioning that encompass quality attributes. Incorporate quality expectations in an organizational definition of done.
- Reduce variation with minimally-documented practices to promote consistency within and across teams. This approach doesn't keep teams from being agile; it does minimize re-learning and re-discovery. Of the 91% of organizations that offer training, 81% report improvement in practice.[28]

"Quality must therefore be an aspect of the value delivery."

- Employ techniques to detect defects early, well before testing. Less rework will lead to high productivity and lower total cost of ownership.
- Measure definitively, consistently and purposefully. Carefully consider unintended consequences and behaviors that may result from measurement activities.
- Shift the dialogue around agile measurements in general. Strengthen the focus on value delivery, priorities and releases versus cost, scope and schedule.
- Use existing reports and research to guide measurement efforts. You may learn some desirable practices or some you wish to avoid. Don't imitate other cultures that don't reflect your own. Best practices in the wrong context often make for bad practices.
- Continuous improvement doesn't happen by chance. Continuous learning feeds continuous improvement.
- Get the right people, trust them, keep them.
- Stop talking and thinking; start doing! ■

Special Thanks

I want to thank the following colleagues, active agile practitioners, for their review and insightful comments that enhanced this article and sharpened my thoughts:

- *Jennifer Turgeon, Systems Research and Engineering, Sandia National Laboratories*
- *Karen Grigg, Agile Advocate, Sr. Director of IT Development, Caesars Entertainment*
- *Brandon Hart, Technical Product Manager, A Place for Mom*

References:

[1] https://www.joejr.com/present.htm; most of these 50+ presentations deal with quality, defects and improvement frameworks

[2] https://www.joejr.com/pub.htm; most of these 35 books and articles deal with quality

[3] ISO 9000, Quality Management Systems; ASQ; 2015, 2011, 2009

[4] CSQ Certified Software Quality Analyst, as an example

[5] Quality Assurance Institute, as an example

[6] retrieved using Google on 5/27/2019

[7] 2017 State of Scrum Report identified more than 500,000 members

[8] SCRUMstudy reports more than 2,000 course registrations worldwide per week

[9] 85.9% of more than 101,000 internationally surveyed software developers use agile; Developer Survey Results; Stack Overflow; 2018

[10] 80% of federal IT projects describe themselves as agile or iterative; Agile by the Numbers, Deloitte Insights; 2017

[11] State of Agile Marketing 2018, Agile Sherpas; 2018

[12] Iterative and Incremental Development: A Brief History; IEEE; 2003; The recollections of Gerald M. Weinberg, who worked on the project, provide a window into some practices during this period. In a personal communication, he wrote: We were doing incremental development as early as 1957, in Los Angeles, under the direction of Bernie Dimsdale [at IBM's Service Bureau Corporation]

[13] Rational Unified Process with ancestral relationships with Agile Unified Process, Agile Modeling, DAD, Enterprise Unified Process, as examples; http://www.agilemodeling.com/essays/agileModelingRUP.htm

[14] The Economics of Software Quality; Jones & Bonsignour; 2011

[15] While there are a number of "correct" ways to spell the word chili (chile, chilli), I used mine—chili, green please; https://hotsaucefever.com/chile/chile-chili-chille-correct-spelling-of-the-word/

[16] https://www.ifpug.org/Documents/Jones-SoftwareDefectOriginsAndRemovalMethodsDraft5.pdf; page 15

[17] See also: http://static1.1.sqspcdn.com/static/f/702523/9242274/1288742153797/200806-Jones.pdf

[18] The IFPUG Guide to IT and Software Measurement; IFPUG; 2012; Chapter 36

[19] http://agilemanifesto.org/principles.html; retrieved 6/2/2019

[20] Doing What A Billion Dollars Couldn't: How Scrum Inc. Partnered With Schlumberger To Solve Their ERP Implementation; scruminc; April, 2019

[21] Dr. Dobb's Report; informationweek; July 12, 2010

[22] The Economic Impacts of Inadequate Infrastructure for Software Testing; National Institute of Standards & Technology; US Dept of Commerce; May, 2002

[23] 13th Annual State of Agile SurveyTM Report; COLLABNET VERSIONONE; May, 2019; pages 7, 8, 16, 11 (as referenced)

[24] The State of Scrum 2017 – 2018; Agile Alliance;

[25] The State of Project Management Annual Survey; Wellingtone; 2018; page 14

[26] The Mess of Software Metrics; Capers Jones; March, 2017

[27] Function Points, Use Case Points, Story Points: Observations from a Case Study; CrossTalk; May / June, 2013

[28] 2017 State of Scrum Report; Scrum Alliance; page 18

About the Author:

Joe Schofield *is a recent Past President of the International Function Point Users Group. Today, he is an enterprise agile transformation coach, an Authorized Training Partner and a Scrum-Certified Trainer with SCRUMstudy™. He has certified hundreds of Scrum Masters, Developers and Product Owners in the last few years. He retired from Sandia National Laboratories as a Distinguished Member of the technical staff after a 31-year career. Joe taught more than 100 college courses, 75 of those at graduate level. He has more than 80 published books, papers, conference presentations and keynotes—including contributions to the books: The IFPUG Guide to IT and Software Measurement (2012), IT Measurement, Certified Function Point Specialist Exam Guide, and The Economics of Software Quality. Joe has presented several worldwide webinars for the Software Best Practices Webinar Series sponsored by Computer Aid Inc.*

Joe holds six agile-related certifications: SA, SCT™, SMC™, SDC™, SPOC™ and SAMC™. He is also a Certified Software Quality Analyst and a Certified Software Measurement Specialist. Joe was a CMMI Institute-certified instructor for the Introduction to the CMMI®, a Certified Function Point Counting Specialist and a Lockheed Martin-certified Lean Six Sigma Black Belt. He completed his master's degree in MIS at the University of Arizona in 1980.

AGILE TESTING METHODS:
A PATH TO IMPROVED SOFTWARE QUALITY

By Sheila P. Dennis, CFPS

Since its creation, the agile movement has continued to evolve and grow in popularity. The reason for this is simply that when implemented correctly, agile works—and its benefits are undeniable. The Standish Group says, "Agile projects are successful three times more often than non-agile projects."

For example, most agile implementations foster increased communication among stakeholders and improved time-to-market. However, while testing is certainly just as important within the agile framework as in traditional development, it is not strictly defined, leaving individual teams to determine how to best approach testing-related tasks. This is problematic because testing is essential for mitigating risk, but it is often undervalued and not considered a priority. As a result, there may be a reduction in quality, ultimately risking the reputation of the team or the company.

Testing includes activities that execute the product (dynamic testing) and activities that review the product (static testing). Most people in software development would recognize dynamic testing, which includes executing test cases and comparing results. Static testing, however, includes reviews and inspections in which a person (or tool in some cases) looks at the code or deliverable and compares it to requirements or another standard.

Reviews and inspections can be applied to any piece of work at any time in the development lifecycle. Reviewing or testing work products as soon as they are available will help find defects earlier in the process and reduce the possibility of rework.

One of the most successful solutions is to combine agile with Test-Driven Development (TDD). The synergy between the two frameworks facilitates higher quality software in less time. More importantly, it allows IT to comprehensively support the needs of the business and the end users of the software, starting early in the lifecycle.

Agile Software Development

Agile software development runs counter to the traditional waterfall methodology that many organizations may still have in place. Agile methods empower a team's ability to deliver by involving the whole team in planning and meeting the business need, utilizing a structure that allows the team to control their process to meet the environment.

Agile is defined by continuous delivery, focusing on what can be defined, designed, coded, delivered and tested in stages. These stages (called "sprints" or "iterations") are short, time-boxed increments, sometimes as little as a week or two. Short cycles, constant feedback and close engagement with the product owner lead to increased communication and reduce the impact if a change in requirements is requested.

Development is a collage of multiple, interrelated processes. Whether the project uses extreme programming, test-driven development, black-box testing or exploratory testing in order to deliver functionality, the processes used to develop the code must be synchronized with the processes used to test the code. Agile techniques leveraging cross-function teams that include developers and testers put teams in the best position to ensure a synchronized process.

Effective testing scenarios require a tester to work with users, product owners, business analysts, developers and others to determine whether a deliverable is what it is supposed to be and whether it meets the definition of "done" and standards of quality. Testing is a collaborative enterprise which is facilitated by the agile framework.

Test-Driven Development

A traditional development cycle utilizes the "Test Last" method, meaning that most testing takes place after all of the other stages of development, often right before delivery. Test Last also involves a separation between developers and testers, whereby the developers complete their work and "throw it over the wall" to a tester, separating development from the user experience and ultimately hampering communication.

"Testing is a collaborative enterprise which is facilitated by the agile framework."

This frequently leads to bottlenecks near the end of a cycle, as issues are discovered that require rolling the product back several stages.

Even within the agile framework, teams can choose to leave testing as the last task in a sprint. Of course, the delays are considerably less than with waterfall (in which testing is completed at the very end of the development process), but bottlenecks are still an issue that impede quality and delivery.

"Test First," in which unit tests are written before the code, can mitigate bottlenecks. In this case, the test helps to define what the code is meant to do, providing guidance for the developer in terms of user functions. This concept is a natural fit with agile in two ways:

1. By developing the tests from the requirements, rather than the code, communication increases. The creator of the requirements, the developer and the tester must collaborate on the tests and the subsequent code, thereby increasing everyone's understanding of the work at hand.

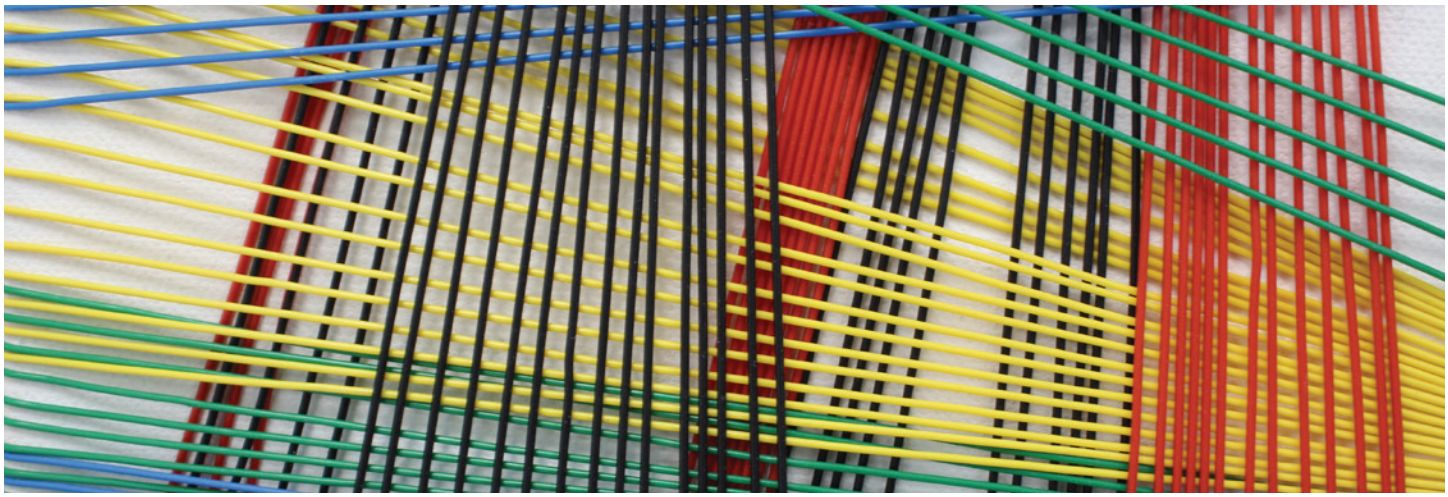
2. By having the test or test suite written first, there is no need to wait for the testing to be done. The code can be written and tested immediately, especially when automated testing is included in the process (considered a best practice). If the code fails, it can be pushed onto the backlog and if it succeeds, the next item can be started.

TDD is a specific type of Test First process. The primary difference is in adding refactoring (step five, below). Table 1 shows the steps in the TDD process:

The Steps in Test-Driven Development

1. Accept a unit of work and write a test case for it. This is often, but not always, a unit test.
2. Run the test, which is expected to fail since the code has not been written.
3. Write just enough code to pass the test.
4. Rerun the test. If it fails, return to step three; otherwise, proceed.
5. Refactor the code to simplify it and return to step four.
6. Repeat all steps until the end of the iteration.

Table 1. TDD Process



This process is sometimes referred to as "red-green-refactor," in which "red" represents writing the test and not passing and "green" is creating the code and passing the test.

Advantages of TDD

There are a number of inherent benefits in TDD, including:

- Fewer delivered defects. This is largely due to early testing, which prevents defects.
- Improved communication. This is a central theme in agile.
- Higher quality tests. This is due to development through the collaboration of all stakeholders.
- Improved code quality:
  - o The code is generally kept simple as a result of the TDD process.
  - o Less dead code primarily due to the refactoring step, which simplifies and cleans up the code. Since each code section is as simple and clean as possible, there is usually less dead code—even late in the application lifecycle.

Disadvantages of TDD

TDD does have its share of drawbacks, which should always be a consideration:

- It is a change, and change requires effort. As with any business transformation, there will be resistance, and the added effort needs to be shown to be worthwhile for internal buy-in.
- TDD often begins with no application to run the tests, so it is often necessary to develop stubs (a testing segment for the code to send results to), drivers (something to send results to the code under test) and other extra blocks of code. However, these items are often reusable as the product proceeds, so they may only need to be created near the beginning of the project and occasionally thereafter.
- The testing is not complete. There will always need to be security testing and acceptance testing on most products.
- It requires strong communication between team members, but this is generally true of all agile teams.
- It usually requires the developers to also do some testing. Then again, developers almost always do some degree of testing, at least a simple check to make sure the code they create works. In fact, it can be argued that testing should be done by developers even with a separate testing team, as it ties the two groups more closely together and can lead to better code.

Other Forms of TDD

In the same way that TDD is a refinement of "Test First" development, there are techniques that take TDD further still.

While TDD creates unit tests, Acceptance Test-Driven Development (ATDD) creates acceptance tests before coding begins, based on the team's understanding of the requirements. This requires even more discussion with the requirements' authors, creating deeper collaboration and furthering the understanding of the requirements by the developers and the authors.

Behavior-Driven Development (BDD) combines unit tests and acceptance tests within specific contexts. The test often follows this formula:

- Given a context
- When an event happens
- Then an outcome is generated

While TDD can be implemented on its own, without the use of ATDD or BDD, it's important to consider which framework most appropriately supports a given team or project.

Improved Software Quality and Value

The true value of IT is how well it can support the needs of the business. One of the core strengths of agile is its ability to increase communication between testing and development groups, as well as between technical and business teams. TDD supports increased communication as well. It furthers the collaborative environment encouraged in agile and enables

> "With a better understanding of the requirements from the business, and by collaborating earlier with testers, developers have a more accurate picture of expectations."

improved understanding of requirements by all parties.

With a better understanding of the requirements from the business, and by collaborating earlier with testers, developers have a more accurate picture of expectations; as a result, they can write cleaner code with fewer attempts. This method impacts and even multiplies its effect throughout the lifecycle, reducing defects and dead code down the line. Less time and money need to be spent in defect fixes and design revisions due to misunderstood or poorly-defined requirements.

There are numerous ways in which these methodologies combine for value-added outcomes. The greatest and truest gains come by way of facilitating a well-interpreted picture of the business objectives and what meeting them will require as early as possible in the lifecycle. ■

Resources:

SPaMCAST 31 – Ambler, Test Driven Development, Words and Change: http://www.spamcast.libsyn.com/s_pa_mcast_31_ambler_test_driven_development_words_and_change

SPaMCAST 295 – TDD, Software Sensei, Cognitive Load: https://tcagley.wordpress.com/2014/06/22/spamcast-295-tdd-software-sensei-cognitive-load/

SPaMCAST 401 – Listening, Quality, Testing and Contract Closure, Developers and Testing: https://tcagley.wordpress.com/2016/07/03/spamcast-401-listening-quality-testing-and-contractclosure-developers-and-testing/

Agile Java: Crafting Code with Test-Driven Development by Jeff Langr

The Cucumber Book: Behaviour-Driven Development for Testers and Developers by Matt Wyne and Aslak Hellesoy

Clean Code: A Handbook of Agile Software Craftsmanship by Robert C. Martin

Five Problems That Impact Testing Effectiveness and Efficiency, DCG Software Value Trusted Advisor Report: https://www.softwarevalue.com/insights/publications/5-problems-that-impact-testing-effectiveness-and-efficiency/

How Can You Make Integration and Acceptance Testing Truly Agile? DCG Software Value Trusted Advisor Report: https://www.softwarevalue.com/insights/publications/ta-archives/how-can-you-make-integrationac-ceptance-testing-truly-agile/

About the Author:

Sheila Dennis *is a familiar figure around IFPUG, being a CFPS member for more than 20 years and having served on at least four committees, including as past chair of the Certification Committee. She is an active contributing author to IFPUG publications, trainer and presenter at global conferences and for global clients. With more than 35 years of experience in the IT industry, Sheila has specialized in business process modeling, benchmarking, cost estimation services, function point analyses and process compliance. In addition to more than 20 years of service for the Department of Defense, she has more than 15 years of experience working at the senior and executive level for client-facing engagements for Gartner Group, CSC and the David Consulting Group/ DCG Software Value. She has a BA in mathematics from Columbia and is currently a senior cost analyst providing estimation modeling for Cobec Consulting Inc. and Logapps Inc.*

A Solution

TO TRACK AND MOVE

By Ankitha Pareek and Anupama Karal

Introduction

Today, Enterprise IT in large Organizations is a complex mix of Tools, Frameworks and Processes. While individual departments focus on delivering maximum throughput to increase ROI, they tend to invest in building their own processes & reports to bring about transparency in the IT Life Cycle. In projects involving cross functional flows and collaboration between departments, there will not be a uniform framework for Monitoring, Reporting and Escalations.

In addition, the absence of uniformity of Tools and Frameworks make the Program Managers and Project Managers spend inordinate amount of time generating and combining reports at Program, Department or Enterprise level.

STATS Dashboard can be designed as a one-stop solution for driving transparency to achieve cross-enterprise IT visibility. It would provide visibility across all aspects of IT Life Cycle from Planning to Operations.

While it would be built tailor-fit for each department, seamlessly fitting into their existing tools and functions, it would subtly push the leadership to self-evaluate and adopt best practices, tools and guidelines.

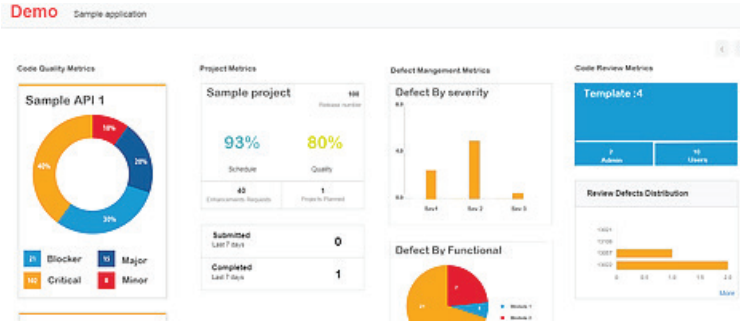


Figure 1 - Snapshot of the dashboard

Birth of STATS Dashboard:

Small to large Enterprises typically have several departments.

Each department can run multiple Programs in parallel. Each Program can have several projects running in parallel.

Every program strives for Continuous Improvement. Continuous Monitoring & Measurement is a pre-cursor to Continuous improvement. Every Program at some point, realizes missing dots in their reporting of Schedule, Adherence & Quality functions. Running around and capturing details from every tool manually drains the team. Setting rules and having a standardized process is what every team wishes to have.

> "STATS Dashboard can be designed as a one-stop solution for driving transparency to achieve cross-enterprise IT visibility."

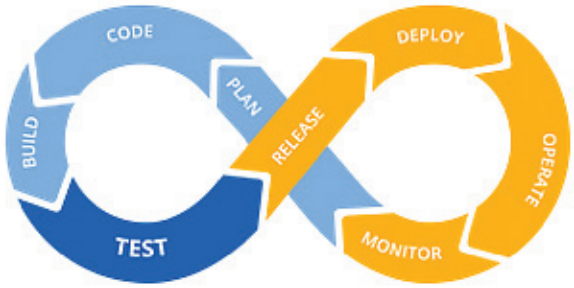
Identifying a solution to below problems can give birth to STATS Dashboard.

- Teams operating in silos and no easy way to measure the engineering maturity of application team.
- Lack of standardized processes across the applications.
- Engineering rigor varies across applications, resulting at different levels of maturity.
- Multiple tools in the ecosystem without any unified dashboard
- Lack of visibility into tool adoption by different application teams.
- Manual reporting of metrics

Aim of STATS Dashboard

STATS Dashboard would be an application which would provide end to end visibility across a Program. This would also include view of Planning to operations on individual projects in the program. Be it about getting visibility to tools which captures defects or a tool which certifies the quality of code, we would have everything on the tip of our fingers.

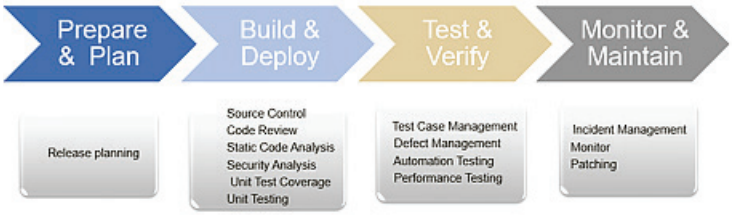
This Dashboard would aim to provide a view into tool adoption and engineering rigor across different applications. Phases from Planning till Delivery would be covered in the Dashboard.



Visibility across different phases

- Plan & Design
- Build & Unit Test
- Test & Verify
- Release & Deploy
- Monitor & Operate

Figure 2 Provides Statistics of each process in every phase



This would aim to provide a real-time data along with trends associated to different applications. This will in turn help in measuring Key Performance Indicator which would define the overall health of all the applications.

This would improve the Continuous Integration/Continuous Deployment maturity across all the applications, resulting in improved agility and reduce silos.

Here are the few Statistics which can be captured in the Dashboard:

- Release Statistic – Monitor the progress as per plan of each application / project
- Code Quality Statistic – Monitor Health of API's
- Code Coverage Statistic – Capture Unit Test Coverage for every API
- Security Statistic – Monitor Security level of Code
- Defect Management Statistic – Monitor Defects in every Release
- Test Automation Statistic – Monitor Health check and Regression suits of each Track
- Performance Statistics – Monitor performance of each Application
- Test Management Statistics – Monitor Test case for every Release.

Excepted Outcome:

This Dashboard would cater to different stake holder of an IT Program. While the executive leadership can get an eagle's eye view of the Schedule, Adherence and Quality of their department initiative, Program & Project Managers will be able to monitor and act on continuous improvement goals.

Delivery teams can use it for their day to day operations to report to IT Stakeholders on their progress, Quality of the product and any immediate risks. IT Operations can use the Dashboard to monitor the heartbeat of the applications.

This Dashboard will not intend to provide in-depth details of the project or replace any of the existing tools, it would merely bring a holistic view of IT Life cycle and act as an accelerant for decision making.

Following incorporation of this Dashboard in the day to day Project life, we will be able to experience below improvements -

- Rapid feedback & continuous improvements
- Usage of Real time trend data for future predictions
- Teams would be organized around KPIs
- Lean engineering teams & improved productivity gains
- Shared Ops and Dev responsibilities
- Continuous delivery
- Continuous testing
- DevOps scorecard
- Zero touch deployment
- Tools Optimization

> "This would improve the Continuous Integration/ Continuous Deployment maturity across all the applications, resulting in improved agility and reduce silos."

Conclusion

STATS Dashboard would bring a holistic view of the IT Life cycle and act as an accelerant for decision making, even though it would not provide in-depth details of the project or replace any of the existing tools.

Focusing only on the deliverables is always not enough. A Dashboard like this would bring critical value proposition when starting new Strategic Programs, as this Dashboard will be custom-fitted to the Engineering Processes and Tools used in any organization and save hours of effort on future monitoring and reporting. It would also ensure peace of mind for the Business Stakeholders to have instant Quality Reports at their fingertips. ■

I'm the sole owner of this article. I have not violated any Mindtree policies as well as not provided any confidential information.

About the Authors:

Ankitha Pareek *is a Business Analyst at Mindtree, Bangalore, India. She has 2 years of experience working in Agile Projects.*

Anupama Karal *is a Project Manager at Mindtree Bangalore, India. She has 13 years of experience and specializes in Agile Projects. She is a Certified SCRUM and PRINCE Practitioner.*

MetricViews

Published twice a year by the International Function Point Users Group (IFPUG), headquartered in Princeton Junction, New Jersey, United States

MetricViews September 2019

Editor
David Herron

Please submit all articles, news releases and advertising to:

IFPUG/*MetricViews*
191 Clarksville Road
Princeton Junction, NJ 08550
United States
(609) 799-4900
metricviews@ifpug.org

Certification Committee

By Gregory Allen, Committee Chair

Certified Function Point Specialist (CFPS) certification extensions remain popular in the 2019 fiscal year. There were 236 certification extensions approved worldwide from July 1, 2018 through June 30, 2019. Details for extending your CFPS certification can be found by clicking "Certification" and then "Certification Extension Program" on the IFPUG website.

Most CFPS and Certified SNAP Practitioner (CSP) exams are conducted using the iSQI FLEX method in which an individual schedules and takes the exam at a time and location of his/her choosing. There is also the option for a group of people to schedule a SMEX exam at a set time in one location. While this option is most often associated with an IFPUG or regional conference, it is not restricted to conferences. For example, a CFPS SMEX exam was recently held in Malaysia for 10 participants. Welcome, Malaysia!

The Certification Committee has submitted Spanish and Korean translations of the CFPS exam to iSQI for publication. The publication dates will be announced as soon as they are scheduled.

Communications and Marketing Committee

By Antonio Ferre Albero, Committee Chair

During the past few months, the Communication and Marketing Committee (CMC) has been focused on multiple technological topics. In April and May, ifpug.org suffered from different downtimes/unavailability as a result of causes outside of IFPUG's control. After discussions that did not lead to clear and concrete root causes, we made the decision to change the website hosting provider. The CMC created a new site from scratch in order to prevent further issues. We migrated, in a plain mode and in a controlled way, the concrete information from the internal tables of the previous host, instead choosing an easy migration between systems, to prevent moving technical components from one place to another. So, from a technical point of view, the site is totally new. We also implemented a system to detect and repair broken links and harmonize folders.

We moved from two providers (one for the hosting and a second one for the IFPUG mail system) to a new one, avoiding historical configuration of redirects between the domain register, the email system and the hosting, and without DNS MX (mail exchanger records). Now, the domain points directly to the hosting provider DNS, with a set of benefits. Under this email system renewal, a reconfiguration and reorganization of email accounts has been done.

Another, not less important, change done by the CMC has been to implement in the ifpug.org site a Secure Sockets Layer (SSL) certificate. So, now IFPUG is "https:" instead of "http:" with the interesting benefits that it provides, starting with more credibility and finishing with having improved positions in the search engines.

Perhaps those changes have been a little bit invisible but a lot of challenging topics, deadlines and milestones have been accomplished in the last months.

Conference and Education Committee

By Filippo De Carli, Committee Chair

IFPUG returned to Bangalore, India with ISMA17 in March 2019 and celebrated "40 Years of Function Points." Allan Albrecht's paper, which created our functional size measurement movement and community, was published in May 1979, just 40 years ago. We discussed plenty of interesting topics related to Function Points, as well as how to size new and different technologies such as the Internet of Things and much more. Visit https://www.ifpug.org/isma17/ for more information and to see photos from the event. Interested in accessing the presentations? IFPUG members can access conference proceedings, at no charge, in the Knowledge Base within the "Member Services Area" of the IFPUG website and partly from the external website by clicking here.

IFPUG supports Métricas 2019. Organized and hosted by the Brazilian Function Point User Group (BFPUG), the event will be held again in Sao Paulo, Brazil on Nov. 7. It has yet to be recognized as CEP-valid. The final program will be published during the next few weeks. Learn more at https://bfpug.wordpress.com/conference-2019.

As any IFPUG committee, the Conference and Education Committee (CEC) is delighted to work with anyone interested in helping us. Would you like to join the CEC? Send an email to ifpug@ifpug.org or complete the volunteer form available on the IFPUG website.

Last but not least, feel free to contact us at cec@ifpug.org!

Functional Sizing Standards Committee

By Dan French, Committee Chair

The first half of 2019 finds the Functional Sizing Standards Committee (FSSC) working hard on the "Mobile Applications" and "Elementary Process" white papers while simultaneously beginning work on the "Application Boundary" and "Single-Sign On" projects. Also, the "XML" white paper is nearly ready for publication.

The FSSC is working closely with the NFSSC to review the General System Characteristics (GSC) as part of an Non-Functional Sizing Standards Committee (NFSSC) research project by Charley Tichnor and Esteban Sanchez through Marymount University.

The committee continues to meet monthly but, due to budget cutbacks, did not hold an annual meeting in June. The FSSC is working on an virtual alternative and our plans are to meet before the end of the year.

Steve Keim, who has served IFPUG on both the FSSC and its predecessor, the Counting Practices Committee (CPC), has announced his retirement. The committee would like to thank Steve for his contributions to IFPUG over the past 20+ years. His efforts are greatly appreciated, and he will be missed.

With Steve's retirement, the FSSC now has an opening on the committee for anyone who is interested in working on a counting practices committee. Volunteers receive CEC credit for their participation. If you are interested, please fill out the volunteer form and submit it to Dan French, FSSC chairman, at (dfrench@cobec.com).

The committee appreciates the support of the IFPUG membership and is always looking for new projects to work on. We welcome suggestions from members on topics of interest. Please submit your ideas.

Industry Standards Committee

By Carol Dekkers, CFPS, Committee Chair

This has been an exciting six months for the IFPUG Industry Standards Committee. Thank you to Steve Woodward who stepped down as committee chair but will remain an active contributing member of our committee.

Industry Standards Work:

- ISO/IEC 25020 Software Quality Standard: Steve Woodward's involvement on various ISO/IEC committees bore fruit with the successful mention of Functional Size Measurement (IFPUG Functional Sizing ISO/IEC 20926) in the recently released ISO/ IEC Standard 25020: Systems & SW Quality Requirements & Evaluation (SQuaRE), Quality measurement framework.
- SNAP becomes IEEE P2430 (C/S2ESC) Standard for SW Nonfunctional Sizing Measurement! Led by IFPUG NFSSC chair, Talmon Ben-Cnaan, the SNAP project was recently

completed and is pivotal for progression as an ISO/IEC standard.

- International Cost Estimating and Analysis Association (ICEAA) Software Cost Estimation Body of Knowledge (sCEBOK): The new Software CEBOK work (involving IFPUG, NESMA, Galorath, COBEC Consulting, PRICE Systems and others) will culminate in a new certification including Function Points as a key component. IFPUG involvement includes myself, Dan French (FSSC Chair), Roopali Thapar (IFPUG Board), and Christine Green (IFPUG Vice President).
- International Software Benchmarking Standards Group (ISBSG): Our IFPUG representative, Pierre Almen, was recently elected as ISBSG President and will become the face of ISBSG worldwide.
- OMG automated Function Points becomes an ISO/IEC standard: Chair of the Consortium for Software Quality (CISQ) Bill Curtis recently announced ISO 19515:2019 Information technology—Object Management Group Automated Function Points (AFP), 1.0 became a standard.

Congratulations to Steve, Talmon and Pierre on your accomplishments. Best wishes to our board candidates, Dan French and Talmon Ben-Cnaan, in the IFPUG election!

International Membership Committee

By Saurabh Saxena, Committee Chair

The International Membership Committee (IMC) is committed to enhancing IFPUG members' experiences by providing quick resolutions to all sorts of queries. In addition to existing country representatives—Gianfranco Lanza (Italy), Lionel Perrot (France), Cao Ji (China) and Ivan Pinedo (Spain)—two new members recently joined the IMC. They are Rajesh Koduru (India) and Sergio Brigido (Brazil). IMC members are doing a wonderful job. In Italy, the IMC successfully closed 44 requests last year; we close 11 requests per month (an average of 2 hours/month of support) in Brazil.

Based on discussions with members worldwide, the IMC has recognized and plans to address the following three challenges:

- Bringing more value to IFPUG membership by identifying and providing additional benefits to members.
- Updating CFPS/CFPP details on the IFPUG website quickly.
- Simplifying the CEP process.

The IMC is working with other committees to ensure that IFPUG not only grows in new geographical regions but also retains its existing members.

Non-Functional Sizing Standards Committee

By Talmon Ben-Cnaan, Committee Chair

SNAP as an IEEE Standard

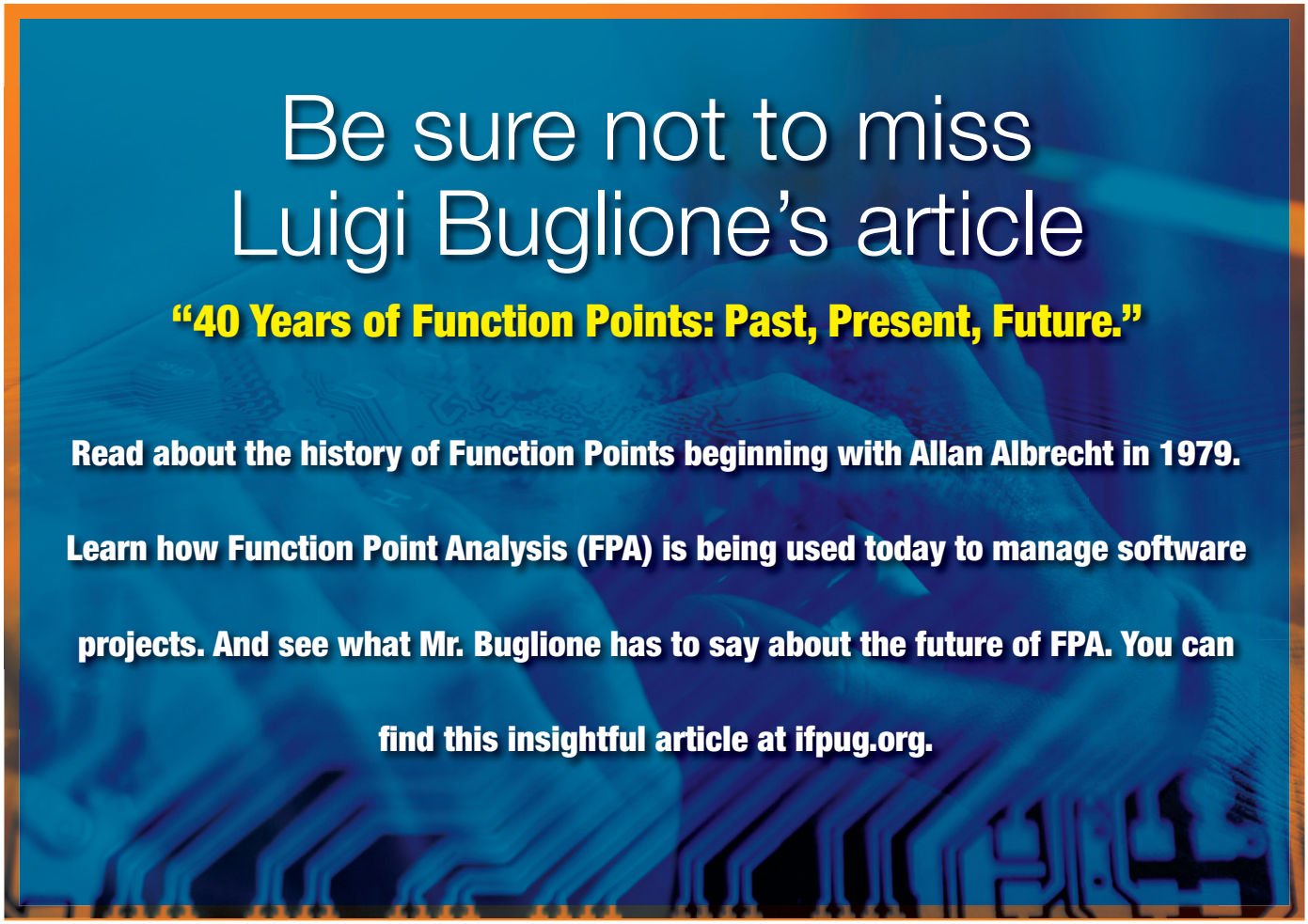
SNAP was approved as an IEEE standard; the new standard will be published in September.

SNAP and GSCs: A New Research on General System Characteristics (GSCs)

In Allan Albrecht's original 1977 paper on function point analysis (Measuring Application Development Productivity), he included 10 "complexity factors" which were weighted from zero to five depending on their degree of influence toward the application being developed. He updated these in his 1983 publication (Software Function, Source Lines of Code, and Development Effort Prediction: A Size Validation) into 14 complexity factors, which are the foundations of the GSCs published today in the Counting Practices Manual. These 14 GSCs have been relatively unchanged since 1983 although additional clarification has been published.

IFPUG Functional Sizing Standards Committee (FSSC) and the Non-Functional Sizing Standards Committee (NFSSC), together with Marymount University, are analyzing the current GSCs considering the new technologies and in light of the introduction of SNAP, and will come with recommendations to update the GSCs, especially for SNAP users.

The NFSSC is calling on users to send us counting data, so that we can analyze the data and provide insights regarding productivity, benchmarking and quality measurements.



IFPUG Board Election Schedule

Date	Action
July 9	Call for nominations.
July 22	Nominations due to IFPUG Office.
August 26	Ballots emailed to membership.
October 4	Ballots and selections due to IFPUG Office by close of business (6:00 pm Eastern Daylight Time). Ballots may only be cast by (then) current voting members.
October	Election results will be presented at the 2019 meeting.

STAY CONNECTED

As an IFPUG member, you are part of an international association dedicated to iÜroving the quality and future of the information technology industry. Are you taking full advantage of all that your membership offers?

Benefits include:

- Access to education and professional growth through semi-annual IFPUG Workshops and the annual IFPUG Conference at special member rates.
- Opportunity to join a local IFPUG Chapter, where you can exchange ideas, share experiences, and learn about new techniques on an ongoing basis, in your area.
- Participation in IFPUG communities to advance state-of-the-art software measurement and professional networking with colleagues from around the world.
- Professional certifications, which establish your credentials as a specialist in the growing field of software metrics.
- Access to state-of-the-art products and services at vendor showcases during the annual conference.
- Special member rates on IFPUG materials (e.g., the Function Point Counting Practices Manual and the International Software Benchmarking Standards Group publications).

IFPUG's social media channels allow you to stay connected to your fellow IFPUG colleagues and the HQ staff.

Be Informed! Stay Connected!