Agile Software Quality Management

Ian E. Savage

IESavage0000@gmail.com

Abstract … for your consideration

This paper introduces a discipline -- Agile Software Quality Management (ASQM) – to reach consensus on what we will build next to provide the most value to the most people – internal and external.

This discipline, ASQM, is a framework that helps us resolve conflicting product priorities, and to enumerate risks, and to budget for and manage those risks.[1] ASQM uses a new structure for these communications: ***quality priorities tables***.

A cornerstone for ASQM is a usable operational definition of agile quality: Quality for a cycle is some function of the salient product attribute(s) in context for that cycle:

$$Q_{cycle} = f\ (q1,\ \ldots,\ qN)_{cycle}$$

This definition says that quality varies from place to place and time to time. What is important one day may not be important the next day – you may have solved all of yesterday's problems. Likewise, what is unimportant one day may be pivotal the next. The priority tables may change from day to day and certainly from cycle to cycle.

ASQM relies upon normalizing the Iron Triangle "**project** constraints" of cost, schedule, and scope as **product** qualities. Once these constraints are normalized, they can be prioritized and specified along with any other important quality attributes.

These **quality priorities tables** use existing technology: Vic Basili's Goal-Question-Metric, Robert Grady's (et. al.) FURPS, and Wirfs-Brock's Landing Zones with some small additions by the author.

This paper will be most useful for Agile teams who want more effective cycle/sprint planning and cleaner communications between major players in their eco-systems.

Biography

I've had the pleasure to be associated with some fine communities: Agile Open Northwest, PNSQC, AgilePDX, SAO/TAO, Tech Alliance of Central Oregon, SAO QASIG (TAO Quality Forum) and working in interesting Pacific Northwest firms including Tek, ASG, Bidtek, CFI, Tiger Systems, McAfee, and Intel,

[1] Testing after all is an exercise in risk management.

Introduction

Q: What should we test?
A: Whatever keeps you awake.

This paper presents a straightforward way to plan and document your testing objectives in a way that empowers everyone on an Agile team. When we applied part of this at McAfee with good success, the entire company started thinking in terms of quality goals, questions, and metrics.

Teams sometimes "lose their testers" when their company adopts agile methods. That is an antipattern. Another antipattern is to maintain a separate testing organization. The alternative, the one presented in this paper, incorporates solid testing methods from giants in the testing community: Juran, Basili, Grady, Wirfs-Brock, and others. I call it Agile Software Quality Management or ASQM.

The material is presented in five sections:

1. Normalizing the "Iron Triangle" – thereby integrating management people into the "whole team"
2. Quality priorities tables – setting cycle-specific goals and handling the resultant risks
3. An operational definition of quality – a function of "the vital few" qualities (ala Juran)
4. Scaling ASQM – from limited WIP/flow to very large systems (smaller is better)
5. Conversations – examples of priority/quality tradeoff discussions

My proposal: ***Quality priorities***[2] are the elementary drivers of acceptance tests in Agile shops. These quality priorities can be engineered before each implementation cycle - allowing your team to focus on, say, ***performance*** in one cycle, ***security*** in the next, and ***reliability*** in the next. This gives you a logistical framework to promote/demote work within your backlog. At scale, an organization can drive its entire engineering operation using ASQM.

With ASQM all team members are expected to ask questions, suggest improvements, and implement those improvements. It blends quality management techniques with agile software methods to guide and document your path through the management of a software solution.

The paper covers a lot of ground, hopefully in a way that builds usable stuff as it goes. Setting the stage, it begins with a review of the traditional "project constraints" of the Iron Triangle and then smites those constraints. It presents how ASQM varies as WIP limits vary, discusses the efficacy of small cycles, then finishes up with examples of short examples of quality trade-off discussions.

To implement ASQM, organizations adopt a flexible operational definition of quality wherein 1) any quality can become the top priority for a cycle; 2) any team member can promote and champion any quality attribute; and 3) your team extends and adapts ASQM to your projects and your contexts.

Major potential learnings:
- Aligning business goals with engineering goals
- Using metrics to articulate context-specific engineering goals
- Using risk management to select salient quality attributes
- Appreciation for the simplicity of short cycles
- Quality priorities tables that drive design decisions

[2] In this paper these terms are basically equivalent: qualities, attributes, goals, properties, priorities.

1 Normalizing the "Iron Triangle"

Q: You really expect senior management to attend iteration planning meetings?
A: If they care, yes. But it's your shop.

1.1 Integrating management into the team

In the current world, circa 2019: schedule, scope, and cost are typically viewed as sacrosanct project constraints that control the course of a development project.[3] And "quality" is "just supposed to emerge" as shown here. Seems like that isn't going per plan in some shops. Let's look at a managed approach:

ASQM asks managers and engineers to jointly decide cycle priorities, goals, and measurements (metrics). Unlike the traditional Iron Triangle and its constraints, the agile team is flat and all team members are expected to contribute. Here's my contribution: I believe Iron Triangle constraints can be recast as ***product qualities*** and managed as other qualities are managed. More on that later.

Perhaps you release every few minutes. Will you hold a meeting for each release? Yes. Members who cannot be present can delegate their vote(s). But it is your shop. Meet how you meet. Do meetings mean clanking chairs, sidebars, loss of focus? No. You can use technology.

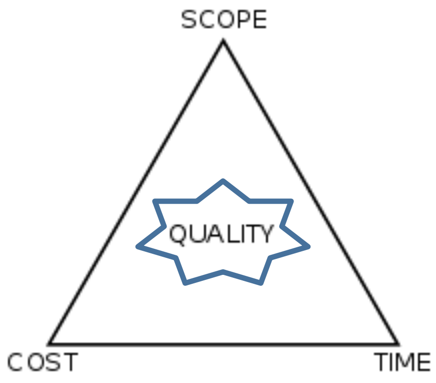


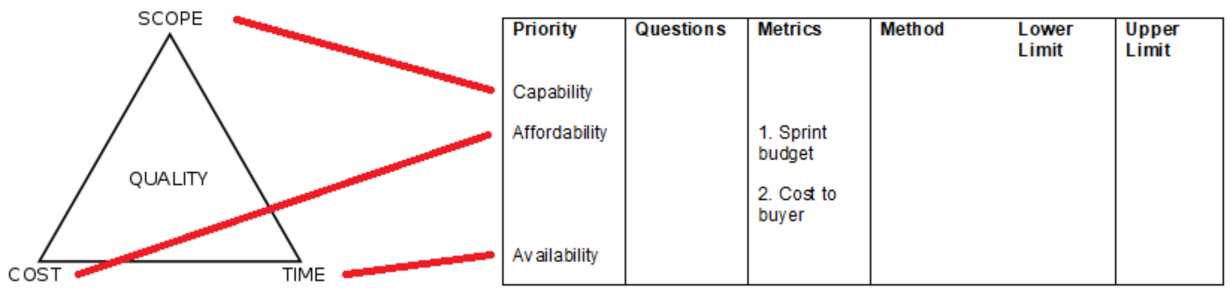
Figure 1: Ye Olde Iron Triangle... the ball-and-chain view of projects.

If you think of quality as some amorphous, squishy blob, then dive deeper. Quality is defined in Section 3. It says that the quality of any cycle is some function of the cycle's attainment of its salient attributes.

Data normalization is a stepwise process that creates data structures (tables) that need no special pre-processing to read, write, and operate upon seamlessly. This same normalization technique can be applied to the Iron Triangle – an arbitrary structure that most software companies seem to have adopted that has Schedule, Scope, and Cost as the ***project constraints*** – those things that project management typically handles. But these ***project constraints*** can be normalized such that they can be treated as ***product qualities***. Once normalized, they can be prioritized along with the rest of the product qualities. Specifically, once these transformations are understood, the Iron Triangle fades away because…

Schedule → ***Availability***	*Availability* has an internal and external component (see below)
Scope → ***Capability***	*Capability* is another name for feature / functionality
Cost → ***Affordability***	*Affordability* also has an internal and external component

[3] Indeed, this author previously held that prudent project managers should assign Fixed, Firm, Flexible, or Fluid to project constraints.



Priority	Questions	Metrics	Method	Lower Limit	Upper Limit
Capability					
Affordability		1. Sprint budget 2. Cost to buyer			
Availability					

Figure 2: Recasting the constraints as qualities

Why recast the project constraints as product qualities? Two reasons:

1. doing so makes them easy to define, measure and track.
2. it elevates project management and general management to the same level as the "Whole Team" and puts them at the same table, facing the same direction.

Once accepted as the norm, the prioritization process becomes more direct and effective, the team works better because all the members understand 1) the business and 2) the technology that will be used. See also Jeremy Lightsmith's Lean Coffee.

Once the project constraints are normalized, they become product qualities that can be prioritized along with all the familiar quality attributes - the "ilities" and others – as shown in Figure 2.

1.2 Implications

Once the "project constraints" are recognized as manageable product qualities, they can be evaluated, ranked, and prioritized along with all the other quality attributes. The next section shows how that might look in three different environments: a limited WIP shop, a more traditional software shop, and a big shop.

For now, let's look at the high-level implications:

1. Project management, program management, and people management will rethink their roles. They may all want to be involved in deciding the cycle goals, and some will want to help manage the backlog queue in a hands-on way. That is, some will be more involved than others in cycle planning and micro decisions making.

2. Software engineers will rethink their role as code adders and realize they need a clear understanding and appreciation for quality attributes in general and for the business-related stuff: product direction, market opportunities, value chain planning and mapping, etc. so they make better detailed design and coding decisions.

3. To facilitate those two things, both groups will need to actively listen to one another's issues and concerns. This non-trivial change changes their lives and the organization for the better. This is the big payday. The demise of the Iron Triangle removes barriers between organizational units: There is no longer a need for a separate PM/PO and for disconnected general management. Everyone gets on the bus and speaks in the same language. In any given context, our oral and written communications are less error-prone when we use the same words and definitions. Every team needs people who listen for ambiguity (topical, lexical, and deeper) and point it out.

4. In general, this expands the set of Agile practitioners to include general management (who are mostly concerned with ***profitability*** and ***affordability***) and product management (who are mostly concerned with ***availability*** and ***capability***). Expanding the set of Agile practitioners to include

these managers in quality planning brings the true whole team to the table and gives them a framework to discuss salient attributes in context. In ASQM, the whole team can decide which qualities are important, what questions the testing must answer, the things to be measured to answer those questions, and the lower-limit and upper-limit (the "landing zones") for each quality.

Summary: Normalizing the Iron Triangle's project constraints into measurable quality attributes:

- equalizes decision making
- opens dialogs
- allows direct comparison of qualities based on their merits and the business conditions.

The next section demonstrates how to create and manage a **Quality Priorities table** based on quality attributes, and discusses the differences between:

qualities-to-assure,
risks-to-manage, and
risks-to-accept.

2 Quality Priorities Tables

Q: Suppose we adopt this "everyone is equal" mentality. How does that help?
A: It enables simpler prioritizing.

2.1 (New) Quality Priorities Tables

Since schedule, scope, and cost become ***availability***, ***capability***, and ***affordability***, that "terrible triad of tyranny," the Iron Triangle, is no longer operant. We need it no longer. Indeed, the death of that Iron Triangle frees us to compare qualities directly – on a level playing field.

Focusing on one quality attribute affects design decisions made during that cycle/iteration. If ***security*** is your primary focus, your software will have fewer attack surfaces. If ***performance*** is your primary focus, you may have more attack surfaces – and you will make those design decisions mindfully.

Quality priorities tables allow us to responsibly answer the difficult question: "How will we know the product is ready for prime time?" **Answer: When all the landing zones are green, we ship it.**

But ASQM holds potential for more than rigorous quality assurance. It also can change your daily life. Normalizing the Iron Triangle constraints into simple product quality attributes is a sea change for organizations that have siloed functional groups like Product Management, Sales and Marketing, Development, Maintenance, Testing/QA. These departments will not instantly disappear, they will fade away once the organization coalesces around short release cycles with consensus on small, well-defined increments.

Note: ***Affordability*** and a***vailability*** may always be in your Quality Priorities tables. Don't be overly concerned… that's a reasonable business decision. With the constant involvement of managers on the team, engineers will gain more appreciation for those business realities.

Your team will eventually reach its equilibrium as your team goes through the forming, norming, storming team maturation stages. Norms develop and practices synchronize. As co-equal qualities, ***affordability*** and ***availability*** take shape through several cycles, and your team eventually finds its proper and fitting risk aversion/acceptance level. Juran, in his watershed book on quality control (*sic*), taught us that quality assurance is simply a matter of assisting management to make good decisions.

The other face of the test/risk coin is risk. We test because we want knowledge about our software. But because we cannot test everything, we must live with some risk. Quality priority tables give us the tool we need to juxtapose our testing against our risk analysis – to present the complete test plan for any cycle.

So, here is the new playing field. The Quality Priorities table with three sections represent descending magnitude of risk:

Qualities to Assure						
Priority	**Questions**	**Metric**	**Method**	**Lower Limit**	**Upper Limit**	**Result(s) [Red | Green]**
Most important property						
Next MIP (optional)						
…						
Last MIP (optional)						
Risks to Monitor						
Risk	**What is the risk?**	**Probability**	**Potential Impact $**	**Risk Budget $**	**Trigger**	**Mitigation**
Highest risk						
Next highest risk						
…						
Last high risk						
Risks to Accept						
Ignorable risks 1…	**What is the risk?**	**Probability**	**Potential Impact $**	**Avoidance measures**	**Reasons**	**References**

Figure 3: The Quality Priorities table layout

2.1.1 Qualities to Assure

These are the properties that are critical to the success of the current release. If any of the qualities in this section are not met (i.e. they do not fall between the Lower Limit and Upper Limit), the software is not ready for release. Fewer entries in this section means the less overall testing is needed. Ideally, N = [0|1] (lowest possible WIP).

Priority ~ which property is critical and must be measured/tracked? Example: ***Reliability***

Question ~ question(s) we have about this priority. Example: "What is the uptime?"

Metric ~ the specific thing we will measure. Example: (uptime) / (uptime + downtime)

Method ~ how we will gain that information? Example: automated monitoring

Lower Limit ~ four nines (99.99%)

Upper Limit ~ five nines (99.999%)

Result(s) [Red | Green] ~ [red | green | yellow | {}]

2.1.2 Risks to Monitor

These second-tier properties indicate some quantifiable risk to the organization. To qualify for this section, a property must be something that would cause an interruption in the flow of software to end users, operations/production, or whatever is appropriate for your shop. Actively managing this second section indicates a healthy team.

Risk ~ which property presents a risk? Example: ***Usability***

What is the risk? ~ A short description such as "New UX will be rejected by some customers."

Probability ~ A number between zero and one. (Risks with a probability of zero can be accepted.)

Potential Impact $ ~ What's the most money the organization could lose if the risk occurs?

Risk Budget $ ~ Probability * Potential Impact $. Budget this much to prepare for risk occurrences.

Trigger ~ The thing(s) that tell us that the risk has occurred.

Mitigation ~ The thing(s) we will do to control the effects of the risk using the Risk Budget.

2.1.3 Risks to Accept

These are the dicey areas that every company accepts every day without documenting it. Put your very best people in charge of these **risks to accept** because there's no money allocated to deal with these problems. Your best people can smell problems and they can navigate organizational chutes and ladders to prepare the organization for potential failures in these low-probability areas.

These people must have the organizational horsepower to elevate a risk into Section 2 or even Section 1. Mid-level managers can be effective risk analysts.

This third section lets you enumerate qualities you are expressly NOT testing. This is an important list. You will make mistakes of omission and of commission. Learn from each mistake. Don't take these low-running risks lightly. These risks are not negligible.

Just like Lean Coffee topics, attributes are reprioritized continually or after each delivery using something super simple like index cards or PostIt® notes. As Ward Cunningham and Kent Beck say: "What's the simplest thing that could possibly work?" If you choose to reprioritize continually, make sure all team members are advised quickly – the goals of the next cycle can affect their design decisions in the current cycle.

2.2 Example Quality Priorities/Risks

Below are a few examples quality priorities tables. Your issues are different so your mileage will vary.

2.2.1 Safety first

"***Safety*** is our highest priority" says the aircraft supplier, "but we need to get our planes back into operation ASAP." Okay, the first two lines of our Quality Priorities table – our top two priorities – would be ***Safety*** and ***Availability***:

Qualities to Assure

Priority	Questions	Metric	Method	Lower Limit	Upper Limit	Result(s)
Safety	Do our planes have fatal software flaws?	Unhandled faults	Static analysis	zero	zero[4]	Green[6]
	How many lesser flaws do we have?	Weighted MTBF	Evaluating failures found in “final test” (simulations, exploratory testing, etc.)	zero hours	10 hours[5]	Red
Availability	When can our planes fly again?	Calendar	Track remediation progress vs plan	January	June	Yellow

Risks to Monitor

Risk	What is the risk?	Probability	Potential Impact $	Risk Budget $	Trigger	Mitigation
Marketability – short term	Reduced customer acceptance	25%	$500m	$125m	Largest customers place orders with competitors	Deep discounts for early adopters
Certifiability	FAA reluctance	50%	$100m	$50m	Planes grounded beyond EOY	Constant contact with FAA

Risks to Accept

Risk	What is the risk?	Probability	Potential Impact $	Avoidance measures	Reasons	References
Marketability – long term	Substantial loss of market share	low	All of it	Constant market analysis	Senior mgmt approval	Internal restricted access docs.
Others		high	$100m	TBD	No data	N/A

Figure 4: Hypothetical Quality Priorities table – safety as primary driver

[4] Typically, the UL will differ from the LL. But releasing critical flight systems software with known fatal flaws is indefensible so the UL is zero – you cannot ship known-defective mission-critical software.
[5] See “defect density” calculations.”
[6] The Results column enables Quality Priorities tables to be used as BODs (big overhead displays).

2.2.2 Cash is king

"We are in business to make money. *Profitability* is our highest priority" says the CFO and the GMs agree.[7]

Qualities to Assure						
Priority	**Questions**	**Metric**	**Method**	**Lower Limit**	**Upper Limit**	**Result(s)**
Profitability	Are we competitive?	Net return from operations	GAAP	20%	N/A	Green[8]
	Are we improving?	Total CoQ and YoY financial results.	CoQ tracking	-10% / year	-20% / year	Red
Risks to Monitor						
Risk	**What is the risk?**	**Probability**	**Potential Impact $**	**Risk Budget $**	**Trigger**	**Mitigation**
Marketability – short term	Reduced customer acceptance	25%	$500m	$125m	Largest customers place orders with competitors	Deep discounts for early adopters
Certifiability	FAA reluctance	50%	$100m	$50m	Planes grounded beyond EOY	Constant contact with FAA
Risks to Accept						
Risk	**What is the risk?**	**Probability**	**Potential Impact $**	**Avoidance measures**	**Reasons**	**References**
Marketability – long term	Substantial loss of market share	low	All of it	Competitive analyses	Senior mgmt approval	Internal restricted access docs.
Others		high	$100m	Unknown	N/A	N/A

Figure 5: Hypothetical Quality Priorities table – profitability as primary driver

[7] This makes sense only for software that is NOT safety-critical, e.g. games.
[8] These Quality Priorities tables can be used as BODs (big overhead displays) via this Results column.

2.2.3 Maslow's favorite

"***Security*** is our highest priority" say the security providers and many companies.[9]

Qualities to Assure						
Priority	**Questions**	**Metric**	**Method**	**Lower Limit**	**Upper Limit**	**Result(s)**
Security	Is it penetrable?	Successful breaches	Penetration testing	Zero	Zero	two
	Does it fail safely?	Privilege elevations	Be-bugging	zero privilege escalations	zero privilege escalations	zero

Risks to Monitor						
Risk	**What is the risk?**	**Probability**	**Potential Impact $**	**Risk Budget $**	**Trigger**	**Mitigation**
Availability	Memory leakage	20%	$100k	$20k	Leakage observed in sys tools	Memory tracing
Performance		10%	$100k	$10k	Throughput degradation	Performance monitoring

Risks to Accept						
Risk	**What is the risk?**	**Probability**	**Potential Impact $**	**Avoidance measures**	**Reasons**	**References**
Testability	UI changes break our GUI tests	20%	$20k	UX team involvement	Status quo	See John Doe

Figure 6: Another hypothetical Quality Priorities table – security

2.3 Other continuing agile testing of course

Using these quality priority tables to test the salient qualities is insufficient by itself. Several other testing practices (such as those listed below) are employed by responsible agile teams. These are part of the team's rhythm and do not need to be repeated in each cycle's quality priorities table.

- TDD / BDD
- T2B (top two boxes on customer feedback)
- Automated, randomized capability testing (background processing) to simulate production
- Analytics and telemetry
- Exploratory testing

[9] Software security people say there are two categories of companies: those that have been hacked and those that don't know they have been hacked.

Repetitive manual regression testing (which has an impossibly low ROI) is expressly excluded from this list of high ROI testing methods. It is replaced by1) automated, randomized capability (feature/functionality) testing and 2) frequent unit test executions with no tolerance for broken tests.

3 Background: Operational definition of "quality"

Q: Where do we put QA?
A: She leads the meetings.

So finally, we get to the question "What is quality?" This question comes around every time we have a significant change in software paradigms. Obviously, we have experienced such a change with the advent of lightweight / agile methods around the turn of the century.[10]

Agile software engineering can be scary: What do we do without testers? How do we engineers test software? How do we know it's good enough? Relax, breathe, all will be fine. ASQM gives us a simple way to plan and execute software testing and to reach consensus on 1) what will be built and 2) when it is truly ready for use in production ("done done").

As seen above, ASQM provides a framework to communicate about important software goals. It helps us to resolve conflicting product objectives, set priorities, and to enumerate and manage risks. ASQM uses these new quality priorities tables work through these difficult and important communications. But these priorities tables require an operational definition of quality – one that promotes action and limits risk. Here is that definition:

For any cycle, quality is a function of the salient quality attributes for that cycle in context:

$$Q_{cycle} = f\ (q1,\ \ldots,\ qN)_{cycle}$$

That is… the quality of any deliverable is some function of its salient attributes.

This definition asserts that quality is context sensitive, it varies from cycle to cycle as the needs of the team change with each cycle. As you have seen above, each of the important qualities – q1 through qN – have their own metrics and targets / acceptable limits. The fewer, the better.

This context-sensitive, cycle-specific operational definition of quality lets us create better software by using quality priorities tables focused on important quality attributes.[11]

So why do we need this new tool (quality priority tables) and this new definition of quality (function of salient attributes)? Because software quality is definable, it is discussable, it is manageable. And it helps everyone in your development stack by identifying where you will focus your testing efforts, what you consider as success, what risks you are assuming and what risks you are tracking.

The following sections discuss how these quality priority tables help management, product owners, and engineering.

[10] While the agile movement is typically associated with the Agile Manifesto, several lightweight methods were gaining popularity in the late 1990s.
[11] Other than this footnote, you will not see "non-functional requirements" anywhere in this paper. The term non-functional already has a meaning in natural language: broken. It serves no purpose in the software industry. Quality attributes are not broken.

3.1 Helping management

An operational definition of quality suggests a strategy to resolve conflicting requirements and, significantly, to include general management and product management on the "whole team" and thusly eliminate appeals to higher authorities and opaque office politics. This reduces the "hidden factory"[12] that saps an organization's energy and profits. As you saw in Section 1, engaging these management types in quality goal setting is as simple as recasting schedule, scope, and cost as product attributes and inviting them to join your decision-making.

3.2 Helping project management / owners

Product owners are caught between two worlds: management and engineering. Successful POs have great organizational skills and technical skills. ASQM gives them a tablet they can use to defining "quality" for the next iteration. And it gives them a template that provides a limited lexicon to bound those quality discussions.

3.3 Helping engineering

Effective use of ASQM gives engineers specific quality targets and provides them and management with visibility and alignment. Quality priorities tables also make engineers wade into the management world of risk analysis and mitigation. Risk management is the other side of the testing coin. Product qualities that present a significant risk are tested. Qualities that fall below the must-be-tested threshold are then treated as risks – some that require monitoring (e.g. with triggers and mitigations) and some that do not require rigorous monitoring but need the attention of your best players. Agreeing on the cut-lines between these sets is non-trivial and it is the gist and grist of ASQM.

3.4 Silo-busting

To implement ASQM, an organization starts by understanding the normalization described above, articulating any serious blocking issues, and filling out a quality priorities table. Once your team has defined cycle quality a few times, then you can extend the model as appropriate for your context.[13] And the PO/PMs among you can talk with your fellow PMBOKs, the managers to other managers, and engineers to other engineers.

But don't stay in those silos. If you find ASQM useful, gather your comrades and share. Every product group deserves a couple evangelists.

The rest of this paper presents how ASQM varies as WIP limits – speaking to the efficacy of small cycles and finishes up with a few examples of quality trade-offs:

- ***affordability*** vs ***X***
- ***affordability*** *vs* ***availability*** vs ***reliability***
- ***affordability*** vs ***availability*** vs ***capability***
- ***affordability*** vs ***availability*** vs ***performance***
- ***affordability*** vs ***availability*** vs ***security***

[12] http://knowledgehills.com/six-sigma/cost-quality-defects-hidden-factory.htm

[13] The author would love to hear your experiences implementing ASQM. Use the email atop this paper.

4 Scaling ASQM

Q: Is ASQM for everyone?
A: Some orgs might be too big.

Will it scale… this Agile Software Quality Management?

Since we are talking about a new field, let's consider how those four words may relate to one another. Then we'll talk about scaling.

On its surface, the term ASQM has several possible meanings. Deciding as a team your definition of ASQM will set the course for your team. Agile can succeed at any level of the organization – it's basically adopting a pattern of inspection and adaptation. Results will vary depending on several factors such as the proportion of skilled Agile Level 2 team members (see Cockburn) and management commitment to active participation.

But are we talking about the agile management of software quality or the quality of agile software management? You decide. It is your organization. You are responsible for your responses to your organization's needs. As your team matures, you can make structural and organizational changes to deliver even better results on a team-by-team basis.

Will your successes be repeatable? Probably. ASQM grows by word of mouth. Success breeds success. If you will feel comfortable with goal-driven engineering, you will feel comfortable improving your capabilities and you will get good value from quality priorities tables. However, if you have scaled agile by simply relabeling your legacy processes, then you will get some value from using quality priorities tables but not the organizational transformation that ASQM can provide.

If you try to expand one super-huge instance of ASQM, your challenges will be many. Big projects will always have big project problems. Sorry. The idea of small goals and quick feedback is antithetical to big projects.

There is hope. Large organizations with extremely well-defined interfaces, low coupling, high cohesion, excellent customer involvement, a quality improvement ethos, learning organizations with compassionate management, etc. those organizations can make ASQM work. Let the author know how that goes. The main problem is that huge projects always have too much communication because there are too many communication channels. That is a subject for a different paper but see "Old and Big" in the table below.

There is no hypothetical size limit for ASQM: the quality priorities tables are resizable. You can test as many qualities as your context indicates, likewise with risks. Of course, your teammates may pull you back from the brink during planning – hey, pay attention – you want fewer items, not more. The more qualities in the mix, the more interdependencies you have between qualities, and the more task switching (lost time, technical debt, and frustration) you will experience.

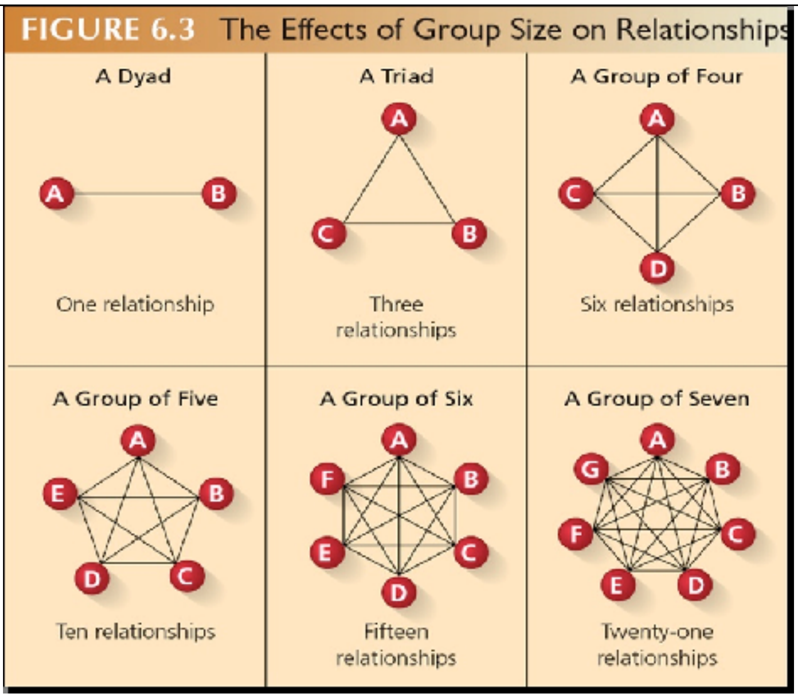
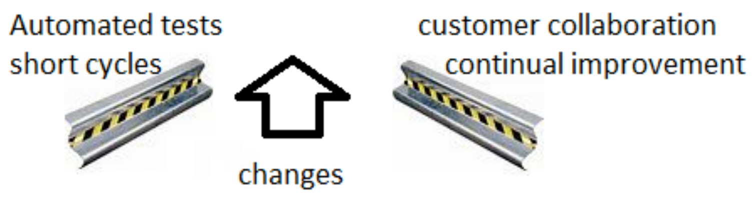
If you are looking to consolidate reporting, to aggregate data for corporate reasons, you may struggle. The leading "A" of ASQM[14] means Agile approaches work best in small groups. Some authors, notably Jorgen Hesselberg, have written persuasively about enterprise transformation. It is non-trivial. Communication becomes less reliable as team size increases. If you find yourselves in a "***thar be dragons***" mode of writing big one-size-fits-all programs and policies and procedures, stop and rethink your commitment to your customers. Agile doesn't fit some organizations. See also Larsen and Shore's Agile Fluency work.

But scaling goes both ways: bigger and smaller. What would happen if you put energy into making cycles as short as possible? What would happen if you assure just one quality per cycle? My contention is that as the cycle time goes to zero, the risk associated with secondary quality attributes goes to zero. If so,

[14] Resources abound about software quality management – Weinberg's Systems Thinking is an excellent starting point.

then driving down the cycle time to its absolute minimum would drive down the number of defects inserted into the software. This remains an open question and worthy of more study. See also #NoProjects and #NoEstimates on Twitter.

For now, let’s focus on the more typical projects: large, medium, and small. The table below divides software engineering shops into those three sizes – you can supply your own magnitudes.

Project size	Team size	Zeitgeist
Old and Big	Characteristics: Many people, perspectives, alliances, motives, styles, plans, desires. Many misunderstandings, corners cut, late projects, stress.	"the software problem" 1950s-present
Middling	 FIGURE 6.3 The Effects of Group Size on Relationships A Dyad One relationship A Triad Three relationships A Group of Four Six relationships A Group of Five Ten relationships A Group of Six Fifteen relationships A Group of Seven Twenty-one relationships Characteristics: Fewer side effects than with big projects. Team size limited by cognitive capacity - the rule of seven plus or minus two. Well-defined modules, data, and interfaces	Structured Systems RUP 1980-present
Small	 Automated tests short cycles customer collaboration continual improvement changes Characteristics: Tests as guardrails for more continuous testing. Limited WIP. many:1 sets of eyes (pairing, mobbing). Large automated test-base that keeps the rubber on the road. Extremely few bugs. Excellent customer relationships. Better predictability.	ZD for software DevOps 2000-present

4.1 Implications for large, medium, small WIP shops

Most organizations aim for growth at any cost… at their peril. As Larsen and Shore (cited above) have taught us with their Agile Fluency work, different organizations should be at different levels of agile fluency because their business doesn't require full dials-to-11 agile software engineering capabilities.

4.1.1 Implications for large WIP shops (aka "Big Projects")

Large organizations with large projects must track more qualities than smaller orgs with smaller projects. Large projects affect more things. So, your quality priorities tables may have many qualities in the "Test" section and many concerns listed in the "Risk" sections. It could have five, 10, 15 qualities and even more risks to manage.[15]

Some qualities are interdependent, so some effects are complex. My advice is to simplify as much as possible and when interdependencies remain, include all the related qualities, set targets for each, then simplify by aggregating into a placeholder quality attribute such as "***Maintainability****-related goals*."[16] Whatever works in your environment/context… inspect and adapt.

Management/Engineering cost is exponential with size, not linear. Many small, well-defined projects with well-defined interfaces, are easier and less expensive to manage.

Expected results of ASQM in large shops

Lots of meetings, lots of closed-door meetings, some alignment meetings

Levels of approval and synchronization

Big roll-out after several "dogfooding" trials

Early adopters, early majority, late majority, but not the stalwarts. Offer them a severance.

4.1.2 Implications for medium WIP shops

Somewhat smaller companies are better prepared than the behemoths to make quick moves. Smaller companies can adopt ASQM easier than large companies. They have demonstrated to themselves the efficacy and sufficient quality of their products. But now the world changes. Disruptive stuff happens. Adapt.

Expected results of ASQM in medium shops

Some buy-in trouble. Structured systems are very attractive. Listen acutely to the nay-sayers – they may be right – radical engineering isn't for everyone. Listening is the only way to find consensus.

4.1.3 Implications for small WIP shops

Already in the habit of working closely with customers and delivering value frequently, small shops can take more risks. Small companies can pivot. The most they stand to lose is the WIP – a week or two (or less) of work. A well-run small company is eager for finding and filling market spaces. You don't bet the company on any one project. You don't need to because over the months and years, you have reinvested all those well-earned Tugmans into evolving your organization's quality culture.

[15] A student asked whether all the IEEE quality attributes would appear in one of these Quality Priorities tables. I believe the answer is "Yes, and in the real world of practicalities, not very often."
[16] Maintainability-related qualities include serviceability, extensibility, portability, and others.

Small companies can grok the idea of having only one item in WIP [see also mob programming]. As a steppingstone to straight-through-processing [see Product Flow], the team can have multiple small feature teams: two to four people if those people can represent the various departments in your organization. Feature teams can work in parallel and must keep current with design changes from adjacent feature teams. No big problem. The team can work out their resolution norms with working agreements / terms of engagement.

Expected results of ASQM in small shops

Extremely nimble business moves

Ability to disrupt major markets

Very high ROI with very small losses

4.2 Cycle duration

Like organizational size, cycle size greatly influences your ability to implement Agile and ASQM.

4.2.1 Huge cycles

N/A. Huge cycles cannot exist in Agile environment unless we use artificial meanings for “huge” and “Agile.” Big, old projects had far too many people, creating far too many associations and queues to be effective. That’s why so many projects from the last century ended in failure.

In these agile times, using huge cycles require suppression of disbelief, or magical thinking.

4.2.2 Middling cycles

The middling-size cycle is most common in this second decade of the twenty-first century. This is the “chaordic edge” that Jim Highsmith recommends we agilists skate along. If we are too comfortable, we’re not engage enough, not pushing ourselves hard enough. If we are too uncomfortable, we might crash.

As a purist, I feel that these one-week or two-week iterations are a mistake. They mask inefficiencies and ineffectiveness. They allow engineers to move undone work to next cycles without analyzing their failures.

4.2.3 Tiny cycles – Limiting risk by limiting scope

Using tiny cycles requires diligence. In Agile we want frequent releases and quick feedback. Inspect and adapt is even more important than delivering working software because adaptations allow more effective delivery for all of time.

Also, the smaller the changes, the fewer side effects generally, so lower risk.

It may be possible to focus on one quality for some cycles because the cycle is tiny, the testing drives the change(s), and the change is well-understood (e.g. easily explainable to a teammate on a white board).

4.3 Quality planning with ASQM

Now here’s the big payoff: These quality priority tables can be used to engineer your software solutions.

4.3.1 Run/Drive Engineering

“The spec is the test.” Therefore, the test is the spec by the identity principle. When we practice test-driven development (TDD), we are intentionally equating the specification (typically an early/left activity) with test planning (typically a late/right activity).

Since quality priorities tables specify each goal – including the acceptance criteria – these quality planning tables can also be used to implement the desired change(s). They can specify ***capability*** changes (formerly feature/functionality) as easily as any other quality. Indeed, once implemented, they can be deployed and sent along with the modified software to the IV&V[17] group or to the customer.

4.3.2 Qualities normalize the Iron Triangle constraints

As we saw in Section 2, ***availability***, ***affordability**,* and ***capability*** are the Agile quality equivalents of schedule, cost, and scope. And they always need to be considered. A product team must have a very good reason to put them low in the priorities queue. These are very important qualities as evidenced by their traditional assignment to very important people in organizations: product managers, general managers, and such. Once these people are convinced that they can take a seat at the table, they become members of the "Whole Team" that Agilists speak about. They are no longer just welcome to attend planning meetings; they are expected to do so. That supercharges Agile teams as they get into their define-develop-deliver rhythm.

The presence of these manager types on your "whole team" gives your team the horsepower needed to refine and decompose product direction into epics and from epics into stories. The three "project constraints" over time will wither away as more teams in your organization adopt the quality priorities approach.

As your customers and management team become more involved, your discussions about the next MVP[18] become more meaningful, your communications improve, and you know that you deliver usable stuff on Friday (for instance). Your sales and marketing people will know it, too. You may hear the aircraft supplier say "***Safety*** is our highest priority, yes. But we also must have it by Friday. Yes, you can increase the sprint budget by up to 15%." The quality priorities tables support these multi-dimensional product needs.

5 Conversations Re: Competing qualities

Q: Which quality is more important?
A: Let's talk here and now.

Salient qualities drive design decisions. For instance, a focus on ***performance*** would have completely different effects than a focus on ***security***. One result of a focus on ***performance*** would be to have fewer page swaps whereas a focus on ***security*** would tend to have fewer attack surfaces. So, the practice of partitioning **capability** (nee ***functionality***) into modules may differ from cycle to cycle.

Because the focus will change from time to time, we must not paint ourselves into corners – unless we like spending time rewriting systems. And if we rewrite systems, we still must decide which are our most important aspects/properties/qualities from time to time as context varies.

Below are some imaginary conversations about which quality should be superordinate and which subordinate.

5.1 Affordability vs X

Affordability is the biggie, so the biggies get to call this dance. Your company's president may well want to see how this works. When she does, she will be an active participant in the detailed planning. She will always be interested in ***affordability***. That's how important ***affordability*** is – those people with P/L responsibility are ultimately responsible for significant costs, losses, and profits. And garden variety

[17] IV&V = Independent Validation and Verification. In the software world it is incorrectly called "QA."
[18] MVP = Minimum Viable Product

quality is important: A colleague of mine once used this example: “Volkswagens and Mercedes both have brakes. The latter are better.”

There are two parts to ***affordability***: 1) what does it cost the company to produce it 2) what are the customers costs (initial and ongoing).

Affordability 1: Producer’s cost

> Initial cost to develop (planning and execution)
>
> market (requirements, rollout)
>
> support costs

Affordability 2: Customer costs:

> Initial cost layout
>
> > Purchasing decision, hardware, software, internal support
>
> Installation costs
>
> Support costs

Will it take some money? Will it pay for itself? When? Like many other quality improvement efforts – and ASQM certainly will improve quality – there are initial startup costs. Yes. But the costs <u>avoided</u> by having common understandings about the cycles content will pay for themselves almost immediately. The guardrails provided by the automated tests will pay for themselves many times over.

5.2 Affordability, Availability, Reliability

Availability: For the purposes of this example ***availability*** means when the software will be ready for the customer or, for large shops, general ***availability***.

Let’s suppose ***affordability***, ***availability***, and ***reliability*** are selected as the three salient qualities. How would the discussion go?

> ***Affordability*** champion: “This must cost the customers less than $8k and it cannot cost us any more than 20 person-weeks.”
>
> ***Availability*** champion: “Person-weeks? Oh, okay I get it. Well I need the first three epics done by Thanksgiving for the big blast-off.”
>
> ***Reliability*** champion: “Do all three of those epics need everything we’ve specified? I fear that we are over-committed. Can any of those epics be beta quality?”
>
> ***Affordability*** champ: “What do you mean by ‘beta quality’?”
>
> ***Reliability*** champ: “Is 99% uptime good enough?”
>
> ***Affordability*** champ: “What’s that mean ‘99%’?”
>
> ***Reliability*** champ: “The software will be up 99% of the time users need it.”
>
> ***Affordability*** champ: “That’ll work for the Thanksgiving demo.”

5.3 Affordability, Availability, Capability

About ***capability***: Software shops using some form of test-driven development (TDD) will be able to appreciate this advice: Don't wait until some arbitrarily late point in your development cycle to verify that the expected functionality is present and performs as expected. In these times, feature functionality / ***capability*** really must be demonstrated before quality attributes testing begins. Goals for ***affordability*** and ***availability*** are more attainable when basic ***functionality*** is treated as an entry criterion to acceptance testing. [See EITVOX]

Therefore, an ***affordability***, ***availability***, ***capability*** quality priorities table may not be necessary. This radical advice will be adopted slowly. It is an outright rejection of the beat-it-to-fit/paint-it-to-match mentality.

5.4 Affordability, Availability, Performance

Performance is always a concern – ever since the manned moon missions in the late 1960s. The USA was challenged to send a man to the moon and return him safely to Earth by the end of that decade. We didn't have the abilities and certainly didn't have the technology – we had to dream it up – starting from scratch…

Performance champion: "We need to minimize the weight of everything. How much does the software weigh?"

Affordability champion: "I don't care about that. We will spend whatever it takes to win the race."

Performance champion: "I must know the weight of everything on the vehicle."

Availability champion: "Wait. Software has no weight."

Performance champion: "What?! Everything has a weight. Don't risk the mission!"

Availability champion: "Okay. You see this punch card?"

Performance champion: "Yes (thinking 'now we're getting somewhere.')"

Availability champion: "You see these holes?"

Performance champion: "Yes, yes of course (thinking "now we're getting somewhere)."

Availability champion: "Add up the weight of all these holes and that's the weight of the software."

[Decision: Software does not present a weight risk.]

5.5 Affordability, Availability, and Security

Security champ: "***Security*** is everything. We could lose the business with another vulnerability like that last one. Let's get it right or call it quits."

Affordability champ: "Well, we only have so many resources for that task."

Availability champ:" We can outsource most of that verification, right?"

Security champ: "If we want our enemies to know our vulnerabilities."

Affordability champ: "Oh – good point. Let's double the budget. I'll talk with the CFO."

6 Conclusion, Summary, Further Research

The idea that we can fully specify an iteration using quality priorities does not appear elsewhere in the literature. Some in the software world still believe that manually rerunning test cases – mind-dulling, soul-stealing ***functionality*** testing – is a good way to qualify products for distribution. That is way too primitive. Those days were never here – we never needed brute force functionality testing. It is wasteful and – NOW – it has been superseded by quality priorities tables and continually reworking the backlog.

Applications of ASQM outside of software seem easy to find. After all, what is a thing other than some function of its attributes? Perhaps there are some non-attributes that help describe things. The author would like to know.

It is entirely possible that an organization with multiple related teams can employ synchronized/parallel small cycles. That is outside the scope of the present paper. The author would also like to know your experiences with that: How are the teams related? How do they interact? Please share the near term and long-term results.

References

Traditional Iron Triangle defined: https://en.wikipedia.org/wiki/Project_management_triangle (accessed August 16, 2019).

Dr. Victor Basili's (University of Maryland): Goal, Question, Metric: https://en.wikipedia.org/wiki/GQM (accessed August 16, 2019).

Grady, Robert; Caswell, Deborah (1987). *Software Metrics: Establishing a Company-wide Program.* Prentice Hall. p. 159. ISBN 0-13-821844-7.

Juran, Joseph: https://www.juran.com/blog/a-guide-to-the-pareto-principle-80-20-rule-pareto-analysis/ (accessed August 16, 2019).