Agile vs. Traditional Cost of Quality (CoQ) — Common Questions & Answers

1. What is "Cost of Quality or CoQ" and why is it important?

- "*Cost of Quality or CoQ*" first appeared in a Harvard Business Review article by Armand V. Feigenbaum in 1956.
- Dr. Feigenbaum of MIT is credited with the concept or notion of "*Total Quality Control or TQC*" (*which later became* "*TQM*").
- In his paper, Dr. Feigenbaum described CoQ as the total cost of quality related efforts.
- His four-part model still endures to this day, consisting of: (1) prevention, (2) appraisal, (3) internal failure, and (4) external failure costs.
- Barry Boehm discovered that the costs increase by an order of magnitude associated with each of Dr. Feigenbaum's four stages.
- That is, a defect that escaped all the way into the customer's hands was the most expensive to manage (*in terms of product recall or litigation for loss of life or property*).
- The American Society for Quality (ASQ) is quick to point out that CoQ is NOT the cost of creating a quality product or service (*but the cost of FAILING to create a high-quality product or service*).
- It's less expensive to create a high-quality product or service than it is to deliver a poor-quality one (*a widely misunderstood concept*).
- It is commonly held that creating a high-quality product or service is expensive (*while skipping quality control activities is inexpensive*).
- The notion that higher quality costs more and takes more time, was used to justify traditional systems and software engineering paradigms such as INCOSE's SE Handbook, CMMI, and a myriad of other process and document-intensive paradigms.
- The common belief is that slow, expensive paradigms such as traditional systems and software engineering increase quality.
- On the other hand, the notion is that faster, leaner approaches such as agile methods decrease quality (*lending credence to the notion of "quick-and-dirty" or speed decreases quality -- A misguided belief Harvard overturned in Dr. Kim B. Clark's 1991 landmark textbook*).
- Some pundits vested in traditional paradigms refer to agile methods as a "*deliver it now and fix it later methodology*" (*because they are faster, have fewer processes, are less expensive, and focus on a smaller, higher-priority value-adding footprint*).
- More recent pundits allege agile methods have too much emphasis on slower, more expensive, and less effective appraisal activities such as testing (*thus driving up CoQ when products have to be recalled and repaired, which is very expensive*).
- If your organization spends millions of dollars on an inordinately expensive quality management system or QMS such as one based upon Baldrige, CMMI, ISO 9001, Six Sigma, etc., then it is probably doing something wrong.

2. Do agile methods rely on late, expensive appraisal and testing (*thus leading to higher CoQ than traditional methods*)?

- These are baseless accusations posited by those vested in traditional methods (*who don't take the time to understand agile methods*).
- It really indicates a lack of understanding of systems and software development principles altogether (*be it traditional, agile, or lean*).
- Agile is based on numerous preventative techniques, which reduce the cost, risk, and probability that high-numbers of defects will be introduced in the first place.
- Agile designs have a smaller footprint than traditional ones, because they focus on only the highest priority requirements.
- Like the Pareto principle, agile methods posit that 80% of the business value lies in 20% of the requirements.
- Furthermore, agile methods focus on only the most essential processes necessary to build the highest quality product.
- Smaller teams work together to emphasize and institute preventative measures not only to build and deliver the necessary product or service, but constantly cross-check one another's work in real-time (*early in the process when it is least expensive*).
- Once a product or service component is constructed, millions of automated tests are executed in fractions of second to uncover any latent defects costing pennies on the dollar (*using free and open source tools vs. expensive commercial testing products*).
- These tool suites measure every aspect of the process and product (*and immediately identify, classify, and delineate both the successes and failures, which can be fed into strategic and tactical process and product improvements in near real-time*).
- Since agile teams work faster than traditional ones, daily, weekly, and bi-weekly process improvement cycles are executed to immediately implement measures to prevent latent defects from recurring (*again, costing pennies on the dollar*).
- Defect prevention activities happen in agile methods 98% faster than in traditional methods (*and order-of-magnitude gains can be seen in only a few hours, days, and weeks using basic agile techniques*).
- Newer agile techniques such as continuous integration, continuous delivery, and DevOps speed up the effects of process improvement (*over conventional agile techniques by yet another order of magnitude*).
- Therefore, agile CoQ is up to two orders of magnitude lower than that of traditional methods.

3. Why is it that traditional CoQ is one or two orders of magnitude higher than that of agile methods?

- Traditional paradigms are based on a number of misguided principles (*mainly scoping new products and services too broadly*).
- The traditional paradigm is that years and millions of dollars must be spent predicting, documenting, and realizing every conceivable customer requirement.
- Traditional pundits somehow believe that quality cannot be achieved without delivering 100% of the requirements (*vs. agile methods, which focus on only essential value adding requirements*).
- Documenting too many requirements results in a host of irreconcilable issues (*leading to poor effort, productivity, cost, schedule, quality, reliability, security, and even safety performance*).
- Most requirements are derived in a vacuum by systems and software engineers (*without speaking directly to customers or end users*).
- Second of all, most requirements exist as tacit knowledge (*and cannot be expressed in words, requirements, or other artifacts such as DoDAF, SysML, or UML models even if customers were present – 'Been there, done that, it doesn't work*).
- That is, true customer requirements exist as hidden, inexpressible needs embedded deep within the human psyche (*leading to the old adage, "customers won't know what they want until they see it*!").
- Again, somehow systems and software engineers believe that documenting tens of thousands of perceived customer requirements is the essence of systems quality (*and quality can't be achieved without "divining" customer requirements*).
- Furthermore, large process and document-intensive paradigms and guides such as the PMBoK, SE Handbook, CMMI, SWEBoK, and many others were devised to manage the complexity of simultaneously implementing thousands of requirements.
- Traditional systems and software engineering has become the science of building complex systems (*where complexity is defined as thousands of simultaneous requirements*).
- Agile methods also lay claim to the science of building complex systems, but rather by eating the elephant one bite at time starting where the customer wants to begin first (*and then stopping when the customer is satisfied before one dies from overeating*).

- Traditional paradigms are based on executing dozens of non-value added processes and creating non-value adding documents in the name of quality (*executing processes and creating documents with marginal value just for the sake of ensuring quality*).
- Spending large chunks of your career creating documentation for a large system such as a plane, train, or automobile is a great way to make a handsome, low-risk living (*but, not really do much else, like deliver a cost-effective, high-quality system*).
- Traditional systems and software engineers spend years and millions of dollars creating perfect architectures and designs to satisfy unneeded or errant requirements to last a century (*remember, Moore's Law, "Technology is obsolete in 18 months*").
- To add insult to injury, many traditional systems and software engineering paradigms rely heavily on manually-intensive appraisal activities (*such as code inspections and late, big-bang integration testing*).
- The lean community refers to all of these excess requirements, designs, processes, documents, appraisal activities, and late testing as waste (*and data shows that over 70% of traditional projects exceed their cost, schedule, and quality targets*).
- Of the few traditional all-encompassing products and services that do make it to customers at a cost of years, decades, millions, and sometimes billions of dollars, data shows that over 95% of these system functions are never used at all.

4. Why do people believe traditional methods result in higher quality (*and agile methods result in lower quality*)?

- Well a lot of it has to do with the current belief system (*which nations, industries, academia, management consultants, and professional trainers have spent greater than six decades trying to instill in systems and software engineers*).
- That is, project management, systems engineering, and software engineering were born in the 1950s and 1960s (*as a result of the advent of enormously complex systems based on the electronic computer following World War II*).
- Because of the enormous size of traditional paradigms, project management and systems/software engineering took more than 40 years to spread across the globe and become accepted as common dogmatic doctrine among systems and software engineers.
- By the 1990s, over 70% of projects used traditional project management and systems/software engineering techniques (*and millions of people were educated, trained, experienced, and certified in these paradigms*).
- On the other hand, agile methods are 10 times less complex than traditional paradigms and quickly swept the globe in about 4 years from 1999 to about 2003 (*and over 70% of IT projects used them by 2003, whereas over 90% of IT projects use them today*).
- Agile methods became the paradigm of information technology projects and left traditional project managers and systems engineers still clinging to traditional paradigms (*as well as stodgier IT management consultants and pundits from the 1970s*).
- Again, traditional systems and software engineers came to believe good systems must cost millions or billions of dollars, take years and decades, and consist of dozens of processes and documents (*voluminous documentation became the hallmark of quality*).
- Furthermore, traditionalists vehemently believed projects must contain thousands of requirements, utilize manual appraisal activities such as code inspections to be good, and design and perform thousands of late, manually-intensive big-bang integration tests.
- The earliest projects such as IBM mainframes had thousands of project participants, took decades to complete, cost billions of dollars, utilized dozens of processes, and produced loads of documentation (*which, again, came to symbolize quality*).
- Agile projects, on the other hand, focused on a handful of high-priority requirements, utilized small teams of 5 to 7 people, quickly coded their solutions, ran millions of automated tests, and rapidly delivered their code (*to elicit the true customer requirements*).
- The speed, agility, and ease of which 20 to 22 year old engineers could dance circles around their traditional counterparts who spent decades teaching, promoting, instituting, and practicing traditional paradigms was simply heresy to traditionalists.
- There you have it, the notion of agile vs. traditional CoQ boils down to a paradigmatic, religious, and doctrinal battle (*rather than one based on common sense, science, statistical data, or mathematical proofs*).
- Modern agilists and traditionalists use their prospective paradigms for no other reason than because they believe in them (*without understanding the mathematical underpinnings and global ramifications of complex systems theory*).
- The operations research community demonstrated that too much complexity in the form of people, requirements, processes, and time causes processing queues to deadlock (*and costs, schedules, and defects to increase, rather than decrease*).
- Mathematicians and other contemporary manufacturing scientists have been illustrating the positive effects of reducing "*batch size*" since the 1950s (i.e., *less waste, lower cost, higher quality, faster cycle time, satisfied customers, etc*.).
- Even Fred Brooks illustrated this in his 1975 seminal masterpiece, "*Mythical Man Month*," with his principle that "*adding people to a late project makes it later*" (*now immortalized in project management lore as the formula*, "*N* (*N - 1*) / *2*").

5. So, what's the bottom line concerning agile vs. traditional CoQ (*is agile better or worse than traditional CoQ*)?

- The use of agile methods reduces complexity, queue size, communication paths, and CoQ (*conversely traditional methods increase complexity, queue size, communication paths, and CoQ*).
- So you make the call, "*which is the true heir to the throne of complex systems, agile OR traditional methods*?"
- Agile methods are a pretty SIGNIFICANT paradigm shift for those greatly vested in traditional methods consisting of heavyweight processes and documentation (*and have been an easy target for pundits over the last 15 years*).
- Late, big-bang integration testing results in too many simultaneous people, activities, communication paths, and defects (*and will freeze, undermine, or destroy the average project, be it agile or traditional*).
- Early practitioners of agile methods utilized something called "*Sprint Waterfalling*" or "*Scrummerfalling*" (*where product and service evaluation was saved up until the end of the sprint, iteration, release, or even project using traditional late, big-bang integration testing*).
- Furthermore, Scrum teams did not utilize some of the advanced agile practices immortalized by Extreme Programming (*such as test-driven development, continuous integration, continuous delivery, etc*.).
- The combination of Sprint Waterfalling, Scrummerfalling, failure to utilize continuous integration early and often every few minutes throughout a project, and embracing traditional late, big-bang integration testing led to a host of problems.
- These included deadlocked queues, late over-budget agile projects, high technical debt, and more importantly, easy fodder for traditional systems and software engineers to attack and undermine agile theoretical tenets (*to defend their home ground*).
- Remember the major paradigm shift to standardize the width of train tracks in the 1850s (*when thousands rioted to protect their jobs of lifting trains and adjusting the wheel width for the next track -- That's where agile vs. traditional CoQ is today*)?
- Any good, sound, and valid idea will succeed purely based on its own merits and any bad idea will fail and fade away based on its own demerits (*that being said, lean and agile methods are here to stay and traditionalism will soon fade away*).
- It's almost NOT worth arguing any more (i.e., *I can't tell you how many times I've been invited to engage in "name calling" panel debates with disgruntled traditionalists around the globe -- It's just silly now, because the goal is to "argue" vs. "understand"*).