Barry McManus

Richie Siconolfi

AN INTRODUCTION TO AGILE METHODOLOGIES

AGILE TERMS THE QA PROFESSIONAL NEEDS TO KNOW

For the second article in our Agile software development methodology series, we look at the terminology used extensively by our technically competent colleagues within Information Technology. Each Agile methodology almost seems to have invented a new set of nomenclature and ideas that can be quite daunting and intimidating to a quality professional. We hope that the previous article in Quasar 139 demonstrated that there is nothing new in Agile and that definitions of Agile terminology are in fact, quite familiar to standard software development practices.

www.therqa.com/goodpractices/digit/agile

Our objective is to point you to our website where we aim to debunk some of the terminology and indicate what quality professionals need to bear in mind when these terms are being discussed.

Users beware! Agile organisations are quick to promote individuals' [technical] abilities over process. Be wary that this should never be at the expense of adequate documentation. Developers have always been averse to documentation (long before Agile became vogue), but the Agile mind-set has emboldened developers to lessen their focus further. Unfortunately, not only is this at odds with regulatory needs, this is also counter to ensuring and maintaining good software quality. The following is an extract of a new section of the DIGIT website dedicated to technical terms with some additional comments from the DIGIT Committee to bear in mind.

Please bear in mind this is not an exhaustive list, nor is it a checklist that must be followed, we would prefer it is used as guidance to help you remember the fundamentals of software development and apply some common sense to the ever-changing Agile world we live in.

ABOUT THE DIGIT AGILE WORKING GROUP

The purpose of the DIGIT Committee is to develop and maintain an active glossary of terms, which we will post on the RQA website and periodically update to keep it current. DIGIT will also from time to time, prepare articles on Agile from the QA professional's point of view. We will rely on and encourage you all as RQA members to review this glossary and submit updates or new terms as they become available.
Agile has been a fixture at software houses around the world for more than a decade and DIGIT is committed to keeping RQA members up to date.

Glossary:

A

Agile:

A term to encompass collaborative, incremental, iterative software development methodologies that favours:

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

Agile Manifesto, http://agilemanifesto.org/ & http://agiledictionary.com/

For QA to consider:

- If every supplier's methodology is agile, ask the question is what "flavour" of agile is theirs?
- What deliverables will it produce and are the integrity of the deliverables known?
- Is it wholly traceable? Are all decision points captured?
- Is quality being built in at all key junctures?
- Is the approach effective in its implementation?

B

Backlog:

A list of requirement items that need to be worked on by development (at some point in time) that regularly changes and is constantly re-prioritised.

Items can be stories, use cases, acceptance tests, (High level business) requirements.

http://agiledictionary.com/

Backlog grooming:

The process of:

1) Adding new items to the backlog
2) Re-prioritizing current backlog items
3) Removing "dead" items from the backlog

http://agiledictionary.com/

Baseline:

In configuration management, a "baseline" is an agreed description of the attributes of a product, at a point in time, which serves as a basis for defining change. A "change" is a movement from this baseline state a next state. The identification of significant changes from the baseline state is the central purpose of **baseline** identification.[2]

Typically, significant states are those that receive a formal approval status, either explicitly or implicitly. An approval status may be marked individually, when a prior definition for that status has been established by project leaders, or signified by association to a position above or below the established baseline. Nevertheless, this approval status is usually recognized publicly. Thus, a baseline may also mark an approved configuration item, e.g. a project plan that has been signed off for execution. In a similar manner, associating multiple configuration items with such a baseline indicates those items as being approved.

https://en.wikipedia.org/wiki/Baseline_(configuration_management)

The term BigDesignUpFront is commonly used to describe methods of software development where a "big" detailed design is created *before* coding and testing takes place. Several ExtremeProgramming (XP) advocates have said that such "big" designs are not necessary, and that most design should occur throughout the development process

http://wiki.c2.com/?BigDesignUpFront

Big Design Up Front (**BDUF**) is a software development approach in which the program's design is to be completed and perfected before that program's implementation is started. It is often associated with the waterfall model of software development.

Proponents of waterfall model argue that time spent in designing is a worthwhile investment, with the hope that less time and effort will be spent fixing a bug in the early stages of a software product's lifecycle than when that same bug is found and must be fixed later. That is, it is much easier to fix a requirements bug in the requirements phase than to fix that same bug in the implementation phase, as to fix a requirements bug in the implementation phase requires scrapping at least some of the implementation and design work which has already been completed.

Critics (notably those who practice agile software development) argue that BDUF is poorly adaptable to changing requirements and that BDUF assumes that designers are able to foresee problem areas without extensive prototyping and at least some investment into implementation.

https://en.wikipedia.org/wiki/Big_Design_Up_Front

For QA to consider:

- A lack of design consideration is a real problem as it is a regulatory expectation.
- Be wary of a lack of design documentation especially about data integrity considerations (data flow, data definitions, data control, performance, security, disaster recovery, etc).
- Check that the design is considered BEFORE the coding was performed – that is, the design is not a reflection of the implementation.
- Does the design provide a supporting framework for the future maintainability of the system?
- Does the design provide support for any increased future usage of the system?

Burn down chart (list):

A line chart that plots “units of items” (points) that are yet to be completed against units of time.

http://agiledictionary.com/

A **burn down chart** is a graphical representation of work left to do versus time. The outstanding work (or backlog) is often on the vertical axis, with time along the horizontal. That is, it is a run chart of outstanding work. It is useful for predicting when all of the work will be completed.

https://en.wikipedia.org/wiki/Burn_down_chart

- For QA to consider:
- How are incomplete items managed and tracked at the end of development iteration?
- What activities occur within each item?
- Are the activities documented? If so, when?

C

Continuous Integration:

is the practice of merging all developer working copies to a shared mainline (location of the core project code) several times a day.CI was proposed by Grady Booch CI in 1991 (Booch method for Object Oriented software development) his 1991 method,[2] although he did not advocate integrating several times a day. Extreme programming (XP) adopted the concept of CI and did advocate

integrating more than once per day - perhaps as many as tens of times per day. This is just GOOD PRACTICE.

Continuous delivery (**CD**):

This is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time. It aims at building, testing, and releasing software faster and more frequently.

https://en.wikipedia.org/wiki/Continuous_delivery

Configuration Item:

The term **configuration item** (**CI**) refers to the fundamental structural unit of a configuration management system.[1] Examples of CIs include individual requirements documents, software, models, and plans. The configuration-management system oversees the life of the CIs through a combination of processes and tools by implementing and enabling the fundamental elements of identification, change management, status accounting, and audits. This system aims to avoid the introduction of errors related to lack of testing as well as of incompatibilities with other CIs.

https://en.wikipedia.org/wiki/Configuration_item

For QA to consider:

- Discover if CI's are identified and managed during a project. If not, then why not?

Configuration Management:

In software engineering, **software configuration management** (**SCM** or **S/W CM**) is the task of tracking and controlling changes in the software, part of the larger cross-disciplinary field of configuration management. SCM practices include revision control and the establishment of baselines. If something goes wrong, SCM can determine what was changed and who changed it. If a configuration is working well, SCM can determine how to replicate it across many hosts.

The goals of SCM are generally:

- Configuration identification - Identifying configurations, configuration items and baselines.
- Configuration control - Implementing a controlled change process. This is usually achieved by setting up a change control board whose primary function is to approve or reject all change requests that are sent against any baseline.
- Configuration status accounting - Recording and reporting all the necessary information on the status of the development process.
- Configuration auditing - Ensuring that configurations contain all their intended parts and are sound with respect to their specifying documents, including requirements, architectural specifications and user manuals.
- Build management - Managing the process and tools used for builds.
- Process management - Ensuring adherence to the organization's development process.
- Environment management - Managing the software and hardware that host the system.
- Teamwork - Facilitate team interactions related to the process.
- Defect tracking - Making sure every defect has traceability back to the source.

https://en.wikipedia.org/wiki/Software_configuration_management

For QA to consider:

- This is the foundation process for computerised systems. If your supplier doesn't possess an SOP on configuration management or doesn't adhere to configuration management then switch off the lights and get a new supplier.

- A good supplier will define their configuration management approach within a configuration management plan, detailing, e.g. configuration items.

D

Definition of Done:

SCRUM: Software has been fully integrated, tested and documented and is potentially shippable.

https://en.wikipedia.org/wiki/Scrum_(software_development)

DSDM: The Timebox (iteration/sprint) close-out session records formal acceptance of all the products delivered by the Timebox.

www.agilebusiness.org/content/timeboxing-0

DIGIT Agile Working Group: The QA Professional must ensure that the definition of 'done' during the sprint includes adequate documentation.

http://www.therqa.com/assets/js/tiny_mce/plugins/filemanager/files/Publications/Quasar/Quasar_139.pdf

For QA to consider:

- How do know what [user stories/requirements] are done?
- During an iteration/spring/timebox, how they are dropped, incomplete or new stories/requirements tracked?
- How are these decisions manifested to the testing activities?

DSDM (Dynamic Systems Development Methodology):

Fixes costs, quality and time as system development constants and treats functionality as flexible. The incremental and iterative focus increases flexibility to meet ever changing business needs

DSDM Atern Handbook, www.agilebusiness.org/content/fundamentals-0

.....a personal favourite of the author(s).

E

EPIC:

A large user story that awaits decomposition into smaller stories prior to implementation. Epics are typically stories that are far off on the development horizon, usually lower priority items. When an epic story works its way up the backlog, it is usually decomposed into smaller stories.

NB: One can easily be tempted to associate the term 'epic' with importance; in Agile, epic relates only to size.

http://agiledictionary.com/letter/e/

For QA to consider:

- Find out how this is performed. How are all the necessary tasks identified and tracked to completion?
- AKA Requirements. One of the key thoughts here is what is selected to be implemented and was it completely implemented, partially implemented or dropped? What is the decision point for deciding what is planned for iteration? What is the decision point of capturing what was actually implemented in iteration? What is the decision point if anything additional got added to the iteration that was not planned (e.g. defect fix, 3rd party library, new functionality to support an intended functionality)? What is the

decision point and tracking for intended functionality that was not implemented? How is this reconciled with the iteration plan? What happens to this functionality and how is it tracked within the remainder of the process? How are all of these decisions documented, what is the approval process for deciding what the iteration team works on and what is the approval process for confirming what the iteration team delivered?

Estimation:

Software development effort estimation is the process of predicting the most realistic amount of effort (expressed in terms of person-hours or money) required to develop or maintain software based on incomplete, uncertain and noisy input. Effort estimates may be used as input to project plans, iteration plans, budgets, investment analyses, pricing processes and bidding rounds

There are many ways of categorizing estimation approaches, see for example. The top level categories are the following:

- Expert estimation: The quantification step, i.e., the step where the estimate is produced based on judgmental processes. (e.g. bottom up estimation Work based Structure, Planning Poker (see below))
- Formal estimation model: The quantification step is based on mechanical processes, e.g., the use of a formula derived from historical data. (E.g. COCOMO, Function Point Analysis)
- Combination-based estimation: The quantification step is based on a judgmental and mechanical combination of estimates from different sources. (e.g. Expert judgment based on estimates from a parametric model and group estimation)

https://en.wikipedia.org/wiki/Software_development_effort_estimation

F

Fail early & fail often:

Release the code to independent testing sooner (rather than at the very end of development) so that defects can be targeted and revealed sooner, when the cost to correct is cheaper and the risk of additional latent defect introduction is reduced

I

Integration hell:

Where one developer's work-in-progress breaking another developer's copy. This is a common symptom of poor configuration management and planning, which may lead to delays, quality shortcuts, latent defects and increased technical debt (see below). Continuous integration is a process designed to overcome this problem.

Iteration:

An iteration, in the context of an Agile project, is a timebox during which development takes place, the duration of which:

- may vary from project to project, usually between 1 and 4 weeks
- is in most cases fixed for the duration of a given project
- at the end of which the team delivers "potentially shippable" product.

This deliverable can be a new feature or feature set, or the improvement or expansion of an existing feature that was completed in an earlier iteration. In Agile, iterations typically begin with a planning meeting, and end with a retrospective.

Iterative Development :

A project life-cycle strategy used to reduce risk of project failure by dividing projects into smaller, more manageable pieces of "potentially shippable" product delivered over the course of a series of brief iterations, or sprints. Iterative development processes afford teams the ability to "inspect and adapt" their processes between iterations, leading to continuous improvement.

Iteration:

(Software engineering) is a process wherein a set of instructions or structures are repeated in a sequence a specified number of times or until a condition is met. When the first set of instructions is executed again, it is called an iteration. When a sequence of instructions is executed in a repeated manner, it is called a loop.

www.agiledictionary.com/

Impediment:

In Scrum, any obstacle preventing a developer or team from completing work. One of the three focusing questions each member of a Scrum team answers during the daily Stand Up Meeting is: What impediments stand in your way?

Impediments may include such things as:

- A meeting to attend
- A lack of technical expertise.
- A technical issue (e.g., a network is down).

www.agiledictionary.com/

L

Lean software development (**LSD**):

A translation of lean manufacturing and lean IT principles and practices to the software development domain. Adapted from the Toyota Production System,[1] a pro-lean subculture is emerging from within the Agile community.

Software systems nowadays are not simply the sum of their parts, but also the product of their interactions. Defects in software tend to accumulate during the development process – by decomposing the big tasks into smaller tasks, and by standardizing different stages of development, the root causes of defects should be found and eliminated. The larger the system, the more organizations that are involved in its development and the more parts are developed by different teams, the greater the importance of having well defined relationships between different vendors, in order to produce a system with smoothly interacting components. During a longer period of development, a stronger subcontractor network is far more beneficial than short-term profit optimizing, which does not enable win-win relationships.

https://en.wikipedia.org/wiki/Lean_software_development

M

MOCK:

A technique commonly used in the context of crafting automated tests. It consists of instantiating a test-specific version of a software component (typically a class), which (instead of the normal behaviour) provides precomputed commands and results, and often also checks that it's invoked as expected by the objects being tested.

For instance, the "mock" version of a database component will a) provide "canned" answers to database queries, instead of connecting to a real live database, and b) verify that the database is being accessed in the manner expected and stipulated in the test.

https://www.agilealliance.org/glossary/mocks/

Common variances include: Drivers, Stubs.

MoSCoW:

A prioritisation model where a requirement should be considered in light as to whether it is (in descending order or priority <e.g. High/Medium/Low>) :

- **Must** have -essential -project is not worth implementing without this requirement.
- **Should** have -important -in business terms, this requirement may not be required immediately, but must be introduced very soon thereafter
- **Could** have -project is not dependant in business terms on this requirement, but is worth recording or aspiring to in light of market predictions
- **Would** have (this time) -a 'gold plating' requirement, some additional functionality that may allow minor business advantage, but should only be included as an afterthought

https://www.agilebusiness.org/content/moscow-prioritisation-0 & www.empowermentqe.com

P

Parallel development:

Parallel development occurs whenever a software development project requires separate development efforts on related code bases. For example, when a software product is shipped to customers, a product development team may begin working on a new major feature release of the product, while a product maintenance team may work on defect corrections and customer patch releases of the shipped product. Both teams begin work from the same code base, but the code necessarily diverges. Frequently the code bases used in parallel development efforts must be merged at some future date, for example, to ensure that the defect corrections provided by the product maintenance team are integrated into the major release that the product development team is working on. This process of divergence and merging is managed by a version control tool such as GIT.

http://www.solutionsiq.com/agile-glossary/parallel-development/

For QA to consider:

- This cannot occur without adequate configuration management.

Planning Game:

Planning poker, also called **Scrum poker**, is a consensus-based, technique for estimating, mostly used to estimate effort or relative size of development goals in software development. In planning poker, members of the group make estimates by playing numbered cards face-down to the table, instead of speaking them aloud. The cards are revealed, and the estimates are then discussed. By hiding the figures in this way, the group can avoid the cognitive bias of anchoring, where the first number spoken aloud sets a precedent for subsequent estimates.

https://en.wikipedia.org/wiki/Planning_poker

Product owner (epics):

On an Agile development team, the real customer or end user, or a stand-in for the customer or end user; a non-coding team member who has a complete grasp of the requirements and business value of the product and is responsible for prioritizing work for the team.

http://agiledictionary.com/letter/p/

The product owner is a role on a product development team responsible for managing the product backlog in order to achieve the desired outcome that a product development team seeks to accomplish. Key activities to accomplish this include:

- Clearly identify and describe product backlog items in order to build a shared understanding of the problem and solution with the product development team
- Make decisions regarding the priority of product backlog items in order to deliver maximum outcome with minimum output
- Determine whether a product backlog item was satisfactorily delivered
- Ensure transparency into the upcoming work of the product development team.

https://www.agilealliance.org/glossary/product-owner/

Q

Quality:

Is agile development yielding a greater quality "software product" than "traditional" approaches?

Are the volume of defects raised I production reduced?

R

Rapid Application Development:

(RAD) incrementally (bit-by-bit) builds functionality to deliver important features sooner and performs parallel tasks to reduce delivery time

Rapid Application Development, Martin, McMillan, 1991

Refractor:

Refactoring is a controlled technique for improving the design of an existing code base. Its essence is applying a series of small behaviour-preserving transformations. The cumulative effect of each of these transformations is quite significant. By doing them in small steps you reduce the risk of introducing errors.

https://martinfowler.com/books/refactoring.html

Release planning (release):

Planning and estimating in the agile world depend on a single key metric: the development team's velocity *(see below)*, which describes how much work the team can get done per iteration. Given a team's known velocity for its last project (if it is known), a release plan represents how much scope that team intends to deliver by a given deadline.

https://www.versionone.com/agile-101/agile-management-practices/agile-development-release-planning/

Retrospective:

An agile retrospective, or sprint retrospective as Scrum calls it, is a practice used by teams to reflect on their way of working, and to solve problems and continuously improve.

https://www.benlinders.com/2013/whats-an-agile-retrospective-and-why-would-you-do-it/

For QA to consider:

- A great practice that should be performed for all quality activities, e.g. at the end of a Validation project.

S

Sprint:

The Scrum term for an iteration or timebox (see below).

http://www.solutionsiq.com/agile-glossary/sprint/

SPIRAL:

Iterative prototyping approach before settling into a traditional sequential approach.

Spiral Development, Experience, Principles and Refinements,
Boehm, SMU/SEI-2000

SCRUM:

Approach that seeks to reduce the reliance on prescribed processes and allow software teams to self-organise themselves to decide on the best way to work

SCRUM Values, www.scrumalliance.org/why-scrum/core-scrum-values-roles?gclid=CMLfxbWS-tECFYML0wodiJIBEw

Scrum Master:

The scrum master is the team role responsible for ensuring the team lives agile values and principles and follows the processes and practices that the team agreed they would use.

The responsibilities of this role include:

- Clearing obstacles
- Establishing an environment where the team can be effective
- Addressing team dynamics
- Ensuring a good relationship between the team and product owner as well as others outside the team
- Protecting the team from outside interruptions and distractions.

https://www.agilealliance.org/glossary/scrum-master/

Systems Architect:

Systems architects define the architect of a computerized system (i.e., a system composed of software and hardware) in order to fulfil certain requirements. Such definitions include: a breakdown of the system into components, the component interactions and interfaces (including with the environment, especially the user), and the technologies and resources to be used in the design.

Their work includes determining multiple design alternatives, assessing such alternatives based on all identified constraints (such as cost, schedule, space, power, safety, usability, reliability, maintainability, availability, and so on), and selecting the most suitable options for further design. The output of such work sets the core properties of the system, and those that are hardest to change later.

https://en.wikipedia.org/wiki/Systems_architect

For QA to consider:

- Ensure to check that this role has adequate training and experience to perform the role effectively.

Story points:

Story points are unit-less measures of relative size assigned to requirements for functionality. They are assigned by the entire team utilizing the planning poker exercise. Story points allow the team to focus on the pure size and complexity of delivering a specific piece of functionality rather than trying to perfectly estimate the duration of time required for its completion

http://www.solutionsiq.com/agile-glossary/story-points/

T

Technical debt:

A programming concept that reflects the extra development work that is required when code that is quick and easy to run in the short term is developed instead of applying a robust designed solution.

If technical debt is not repaid then it can accumulate interest that is harder to repay – that is, it becomes harder to implement the necessary changes later on.

Technical debt may arise as a result of a number of issues. E.g. business pressure, lack of process understanding, a lack of design, poorly engineered software, lack of documentation, lack of standards, limited test strategy, last minute changes and so forth.

https://en.wikipedia.org/wiki/Technical_debt

The risk of technical debt is manifested by missed deadlines, a high volume of issues captured during Validation, a high volume of Change Controls arising in production use, a high instance of negative effects (ripples) as a result of implemented Change Controls.

Timebox:

A timebox is a time period of fixed length allocated to achieve some objective. In agile development, iterations and sprints are examples of timeboxes that limit work in process and stage incremental progress.

http://www.solutionsiq.com/agile-glossary/timebox/

Test Driven Development (TDD):

is a software development process where:

- The developer writes a failing automated test case that defines a desired improvement or new function.
- They then produce code to pass that test.
- They refactor the new code to acceptable standards.

This is a good practice as it helps to reduce “gold platting”, a practice that adds un-necessary code or features to a system

Test Automation:

In software testing, test automation is the use of special software (separate from the software being tested) to control the execution of tests and the comparison of actual outcomes with predicted outcomes. Test automation can automate some repetitive but necessary tasks in a formalized testing process already in place, or perform additional testing that would be difficult to do manually, e.g. interface testing, messaging protocol testing, performance testing.

It is critical for verifying nightly software builds.

https://en.wikipedia.org/wiki/Test_automation

For QA to consider:

- Test automation has an overhead in that it requires the same techniques, planning and skills as for development. Poor quality code or code that is highly changeable will require rework. Associated automated tests will also require rework.
- It is important to ensure that automated tests have been reviewed to ensure that they are still effective and not providing false positives/negatives. (False positive: working functionality is marked as failed, False negatives: failed functionality is marked as passed).

Testing:

Will warrant its very own future QUASAR article.

Unit Testing, Integration Testing , System Testing, User Acceptance Testing, Disaster Recovery, etc.

For QA to consider:

- Agile organisations tend to discuss unit testing in fine detail, as unit testing can be coded. Agile is developer driven and developers often have a limited appreciation of the benefits of testing.
- Be careful to ensure that the key testing activities are not overlooked. Testing is a fundamental tool for quality assurance and control. Each testing activity has a specific objective and purposes. For example, unit testing seeks to challenge a unit of code (such as a control loop) whereas integration testing seeks to challenge the interface between 2 distinct modules of code.
- What is the scope of testing?
- Questions to ask include: What type of testing is performed by the organisation? Is there an overarching test strategy that covers specific scenarios or risks or phases in the methodology? If there is a focus on unit testing (focus on the logic of code) is there a corresponding focus on the functionality of requirements?
- Considering that unit testing is a big focus. What is the scope of unit testing? How effective is it? What documentation is available to demonstrate the effectiveness, adequacy, coverage of automated unit tests?
- The scope of automated unit testing may be difficult to assess if access to the code is denied. Testing at the iteration level is normally not documented
- Is there a formal test phase? How is this documented? Are formal tests designed against a formal specification? How are defects managed? Is there traceability of failed test cases, defects raised, fixed, release, retested and regressed? Is this documented? If a defect cannot be fixed, is there documented analysis of the impact of the defect to the business process and any subsequent workarounds? Is there documented evidence of the defective functionality being removed from the release?
- Is there a focus on negative testing at each test phases (unit, integration, system)? If there is this documented? If not, what are the risks?
- Is testing involved in the release process?
- For example, regression testing to verify that negative ripple effects have not been introduced after a change or after refactoring? Are smokes tests performed to ensure that the build and release approach is not compromised?
- Even though agile is a “different” approach (or is it?) it doesn’t detract from the objectives of robust and efficient testing – all methodologies still need to consider and plan for appropriate and adequate testing… and we haven’t even touched on performance or security!

U

User story (Scrum):

A requirement, feature and/or unit of business value that can be estimated and tested. Stories describe work that must be done to create and deliver a feature for a product. Stories are the basic unit of communication, planning, and negotiation between the Agile Team, Business Owners, and the Product Owner. Stories consist of the following elements:

- A description, usually in business terms
- A size, for rough estimation purposes, generally expressed in story points (such as 1, 2, 3, 5) (see velocity below)
- An acceptance test, giving a short description of how the story will be validated

http://www.solutionsiq.com/agile-glossary/story-user/

typical Format of a User Story:

"As a *<role>,* I want *<goal/desire>* so that *<benefit>*"

https://en.wikipedia.org/wiki/User_story

e.g.

As a user, I can indicate folders not to backup so that my backup drive isn't filled up with things I don't need saved.

Cohn, Mike. "User Stories". Mountain Goat Software. Retrieved 27 April 2016.

Unit Testing:

A unit is the smallest testable piece of software that can be compiled. A unit test is the testing performed to verify that the unit's construction is correct e.g. branch logic, iteration loops, data structures.

For QA to consider:

- This is an important task, not to verify the design, but to verify the algorithms that when amalgamated will fulfil a design. It is a vital tool to verify the data integrity capabilities of units of code.

V

VoC – voice of customer

"Voice of the Customer (VOC) is a term used in business and Information Technology (through ITIL) to describe the in-depth process of capturing a customer's expectations, preferences, and aversions. Specifically, the Voice of the Customer is a market research technique that produces a detailed set of customer wants and needs, organized into a hierarchical structure, and then prioritized in terms of relative importance and satisfaction with current alternatives.

https://en.wikipedia.org/wiki/Voice_of_the_customer

Velocity:

Velocity is a capacity planning tool. Velocity tracking is the act of measuring said velocity. The velocity is calculated by counting the number of units of work completed in a certain interval (e.g. story points, see above), the length of which is determined at the start of the project.

The main idea behind velocity is to help teams estimate how much work they can complete in a given time period based on how quickly similar work was previously completed.

https://en.wikipedia.org/wiki/Velocity_(software_development)

W

Wiki

A **wiki** is a website that provides collaborative modification of its content and structure directly from the web browser. In a typical wiki, text is written using a simplified mark-up language and often edited with the help of a rich text editor.

For QA to consider:

- Suppliers will deploy instructions for using tools, performing reviews, providing code standards and so forth in a wiki. Questions to ask include:
- How much of the quality processes is contained within a wiki?
- How is this content controlled and managed?

X

Extreme Programming (XP)

concentrates on taking good engineering practices to 'extremes' e.g. test driven development (writing unit tests to fail and then writing the code to pass those tests), test early and test often and increased feedback (team feedback, code feedback and customer feedback)

Extreme Programming, https://en.wikipedia.org/wiki/Extreme_Programming

Y

You're Not Gonna Need It = YAGNI

Always implement things when they are **actually** needed, never when it is guessed that it may be needed**.** The rationale for this is that it may turn out that it may not be need it after all or what is actually required is different from what was originally thought of earlier. This is the idea of simple and incremental design.

http://wiki.c2.com/?YouArentGonnaNeedIt

For QA to consider:

- This is a good practice for concentrating on the immediate needs of a system in order to get value add.
- However, it should never pertain to items that make a system easy to maintain, modify, secure, support data integrity, support performance needs and so forth.

End