Fundamental Principles of Effective Requirements Engineering.

Summary

Requirements Engineering is out of favour in the IT delivery world. Agile practices are king and these place limited emphasis on explicit requirements practices. However the current situation fails to recognise the importance of this domain. Furthermore there is a failure to recognise that where Agile is successful it is actually doing highly effective requirements engineering work without explicitly calling for it and that where Agile struggles one of the main factors can be that the requirements challenge is beyond the size that pure Agile can deal with.

There are fundamental principles of effective requirements handling that are universally applicable irrespective of the delivery model and practices in use. Twelve of these are identified and described in this article. There is a significant risk of serious requirements related issues unless each of these principles is effectively addressed during delivery. However there are no prescribed ways of implementing these principles; different situations require different approaches. What is the most appropriate in one situation may be a vast unnecessary overhead in another.

An assessment of a minimalist pure Agile approach against these principles indicates that it fits situations with certain characteristics but is unsuitable for others if good requirements discipline is to be maintained. There are two fundamental approaches to dealing with this whilst still practising Agile software delivery. The first is to embed suitable light weight requirements activities into an Agile structure. The second is to embed an Agile software delivery capability into a system engineering process that includes a Requirements Engineering discipline. The former is suited to close to Agile norm situations; the later to enterprise scale high intensity delivery situations.

Table of Contents

1. The Death of Requirements Engineering

Requirements Engineering is the discipline that focusses on understanding and expressing what something needs to do before you try and define what it will do and build it. Once seen as the foundation of software development its position is no longer as clear.

The Standing of Requirements Engineering

Within many sectors of the IT industry requirements engineering is most definitely out of favour. Tarnished with the same brush as the monolithic waterfall development approach that failed so many organisations it is considered old school, wasteful and ineffective. In a fast changing world with rapid delivery cycles requirements can't keep up.

Light Weight Process Dominance

Today light weight, Lean and Agile processes are king. On the surface these processes eliminate the need for requirements engineering. It certainly suits many software technology dominated teams to pick up and promote this view. It reduces the time and effort spent working with complex abstract ideas and concepts in a space where these is no standardised languages to express them. It allows an early move into hard engineering the defining, designing and coding of a system.

The User Story is the dominant requirement side partner in these approaches. It provides a thin requirements centric layer that requires limited isolated time and effort; any significant requirements effort being encompassed within the mainstream delivery effort. The scope of a release is planned using stories as the main building blocks. Stories provide common targets around which to align different activities. What is actually needed, the detail, is teased out during the delivery of the story with the cooperation of user representatives.

Is it Defunct?

So, as the title of this section implies, have we have reached the stage where Requirements Engineering is defunct, at least as far as mainstream IT development is concerned? Is it a discipline that Programme Management and Programme Assurance can ignore? Obviously, given the title of this article, the answers to these questions that will be given here are 'no' and 'no'.

Success for complex IT programmes, well for any type of programme, always depends on successfully tackling the requirements domain. If you don't you fail; full stop, period. Successful Agile delivery operations are doing highly effective requirements engineering; possibly without realising it. It may not be flagged as such, it may be a dispersed rather than a focussed effort, but it is happening. It works because the Agile approach or the derivative of Agile approach in use is well matched to delivering IT in these particular environments.

Does that mean that explicit, as opposed to implicit, requirements engineering is dead? Is it safe to assume that using an Agile process will deal with requirements? Again; no. There will be a core set of situations where 'standard' Agile practices will perform well in the requirements domain. As situations diverge further from this core initially refinements and eventually wholesale change in approach will be necessary to ensure requirements are adequately dealt with. There will be a continuum spanning from the very light weight approach inherent in Agile processes to more focussed and explicit requirements practices.

Success of any programme of work always depends on effectively addressing a number of fundamental Requirements Engineering elements. It does not matter how the elements are done or who does them as long as they are done well. The next section outlines these elements.

2. Fundamental Requirements Practices

This section outlines essential elements of an effective requirements practice.

Abstract Principles

The elements given in this section are viewed as fundamental abstract principles that enable successful handling of requirements and avoidance of failure through requirements issues. In no way is this section saying you must have a requirements phase nor that you must have explicit practices and processes for each or any of these elements. Similarly it says nothing about when in your cycle these things have to be addressed. Instead the argument is that you must look at and judge whether your practices deal adequately with each of these elements at the right time. If not then something has to be done about it.

Judging whether any particular practice is adequate is highly dependent on the environment within which you operate. Factors such as scale, the coherence of the organisation and the alignment of everyone's true objectives are key. Adequacy also depends on the consequences if things do go wrong and requirements are missed or misunderstood; expensive consequences mean practices need to be tighter.

Finally, as with all things, compliance with these principles is not binary; rather it is a matter of how well practices achieve the objectives stated for each element.

Fundamental Requirements Principles

At this time we identify twelve fundamental principles. Each is summarised in this section. A name and a description is provided for each. All of the names start with 'C'. At first this was a coincidence; as the principles were being written down it was noticed that the names chosen all started with 'C'. It then became a game; the objective of the game being to come up with a meaningful name for the subsequent ones that also started with a 'C'. So far so good on the naming front.

Conceived	Each requirement must be conceived; that is identified as something to potentially be concerned about. An effective requirement practice must have a low probability of completely omitting any significant requirement from the requirements inventory. It must encompass all affected parties and cover all significant dimensions.
Crystallised	At some point each requirements must be taken from its initial partial and potentially vague state to a crystallised state where it is internally coherent and defined with sufficient detail that compliance could be objectively assessed. Interestingly the principle does not define this as being an upfront activity; it does not preclude this being achieved as late as possible which in some cases may be after live trials and refinement. However accurately crystallising the requirement, even if this is a retrospective event, is important. A requirement that can be discarded that isn't important enough to warrant the effort of crystallising it probably was not real requirement in the first place but was most likely a characteristic of the solution. An effective requirements practice should ensure each requirement is crystallised or explicitly discarded as not being a true requirement.

Captured	Crystallised requirements must be expressed and captured in an appropriate form that will allow transmission of and prevent the loss of the knowledge they represent. The captured form should ensure that neither the passage of time nor changes in personnel will lead to the loss of the knowledge. Retention of knowledge under these conditions is heavily dependent on both clarity of expression and on explanation of context. The form must also be suitable for the effective transmission of the knowledge to the audiences that will need to use it to do their work. The suitability of a form for this purpose will depend on the nature of the audience, to what extent they already posses a common body of knowledge and on how closely they work together.
Confirmed	Each requirement must be Confirmed as being true to the thing the sponsor of that particular requirements believes they need. There should be some form of interaction that presents the requirement back to the sponsor in some form and then solicits from them confirmation that the understanding is correct.
Challenged	Requirements should always be challenged. This is to test whether the sponsor fully understands the implication of what they are or are not asking for. Over prescriptive requirements or ones that insist on too high a specification incur both cost penalties and also create complexity that can result in failure. Requirements that are too lax can result in systems that are not fit for purpose and the sponsor may not immediately recognise this or may assume that unstated items are inherent which they may not be. It is important to ensure that each requirement is rigorously challenged from both perspectives. The practices of the organisation should minimise the risk of unnecessary cost or complexity being incurred from over specification and act to identify further constraints before the cost of incorporating them starts to escalate.
Contained	Requirements should be contained to the requirements domain. They should not stray into defining the solution where an alternative solution would be adequate. Practices should ensure a clear distinction is maintained between true requirements and characteristics of the solution whether or not these characteristics subsequently become a commitments to the customer.
Conflict Managed	Perfectly valid requirements can conflict and in all but the simplest of problem spaces there will be conflicts between requirements. These can be direct conflicts where complying with one requirement will mean directly violating another. They can also be indirect conflicts where optimising the solution to meet one requirement may increase the risk of problems for another requirement. For example making order entry easy to increase the likelihood of an customer completing the order process may cause the accuracy of order information to be lower threatening a requirement on captured order accuracy. Identification, management and resolution of requirements conflicts must be present in an effective process. Unresolved conflicts lead to failure. Many IT deliveries have failed because they tried to meet conflicting requirements. This requires: • Identification of direct conflicts. • Resolution of direct conflicts through requirement change agreed with the sponsors. • Identification of indirect conflicts. • Agreeing the relative priorities of each requirement linked to an indirect conflict amongst the sponsors of the requirements.

Communicated	Requirements and changes to requirements have to be reliably communicated to all parties that need to know and react to the requirements. Requirements that only some people know about or changes that were agreed without others having the chance to explain the consequences of the change have caused havoc in many projects. Good systematic communication and good access to the current requirements knowledge is an essential feature of an effective approach.
Classified	All requirements are equal; yet when push comes to shove some will be let go more easily than others. Each requirement should have a value and hence an importance or priority. Effective working practices should ensure that each requirement is classified in some way that ensure it is the most important ones that get the most attention to ensure they are right when they are delivered.
Concurrence	At some point everyone will have to concur that there is an agreed set of requirements that a delivery is working towards. The customer and sponsors will have to agreed that if it does these things it will be useful and provide adequate benefit. The supply chain will have to agree that it is likely to be done in the timescales and with the resources available. Some form of explicit agreement on a common understanding of the scope of supply and subsequent transparency around any changes to this is a key feature of a disciplined process.
Concessed	Having started aiming to achieve some requirement it may become clear at some point that it is not going to be met. If the requirement is a real one then some form of explicit concession practice should apply to record the recognition by all parties that the requirement was valid, still has value if it can eventually be achieved but is not going to be achieved at this time. For clarity and to ensure a firm basis for future maintenance and evolution this should be explicit and transparent.
Consolidated	At some point the requirements should be consolidated. The knowledge should be brought together, organised and stored in a form that will permit access by anyone who wishes to understand and explore the requirements. This consolidation may be driven by the requirement for effective communication as it may be a prerequisite of communication during the work. Alternatively it may be found that whilst everyone is focussed on the work at hand the required communication can occur without consolidation. In the later case consolidation at the end is still important because future evolution of the system will need access to and understanding of all of the requirements in force at the end of the previous cycle.

Recap the Nature of the Principles

The principles are intended to be as generic and independent of any particular process as possible. Each is included because we believe it is a fundamental factor in avoiding problems arising in the requirements domain or in the realisation of recognised requirements. There are many diverse approaches that can comply with these principles but only a subset of these will be suitable in any particular situation. The factors that influence this are themselves diverse and complex.

Neither compliance with the individual principles nor compliance with the overall set of principles is a binary matter. In any real world situation compliance with each one is a matter of degree and there can be no generally applicable measure of the overall level of compliance as the importance of individual principles will vary from situation to situation. Assessing the adequacy of practices against the principles first involves weighting each principle given the situation within which work is being performed.

3. Agile Approaches and Requirements

This section contains a brief discussion of Agile development and the factors that influence when it will and how well it support the principles outlined in this article.

Agile User Story Centric Development

In Agile development brief User Stories and associated acceptance criteria are generally used as the main requirements medium. In true Agile development all parties form a virtual team working together and are collocated to facilitate direct effective communication.

In simplistic minimal interpretations of the Agile ethos the next step on from a User Stories is the implementation process for the system which may, in some cases, mean going directly to code. So often there are no explicitly identified requirements actives inside the delivery process. The mantra “User stories are placeholders for a conversation and are not requirements” has been heard on a number of occasions without the speaker realising that the “conversation” could perfectly legitimately involve an explicit requirements crystallisation exercise.

This approach uses short cycle development with rapid feedback from the embedded customer representative having access to the ‘finished’ product to crystallise requirements and to obtain confirmation of the requirement. Communication comes with the virtual team model and the collocation of the members. Concurrence comes at the sprint planning stage and is maintained through the overall awareness of change within the team. Under the right conditions and at the right scale effective crystallisation and communication of functional requirements can be seen.

Inherent Limitations

Turning to the limitations of the approach the observations on this are summarised below:

- Ensuring all important requirements are conceived is heavily dependent upon the customer representatives in the team, on their experience and on what they choose or have the time to focus on. The nature and form of User Stories leads to a focus on localised functional behaviour; it does not encourage attention to wider behaviour nor to non-functional aspects.
- For similar reasons crystallisation tends to be good for primary functional behaviour but less good for other aspects.
- Agile processes have little provision for the capture of requirements. The end solution complies with the requirement but it is hard to discern from that what is a requirement and what is a characteristic of the solution. There is generally no concept of in-flight capture of requirements.
- Explicitly challenging the requirements is outside the scope of Agile processes apart from in the top level scope setting for each sprint where a requirement may be rejected due to the low benefit it provides.
- Any containment comes by chance. It comes from the way people choose to define their User Stories. There is no explicit recognition of the difference between the need and solution spaces. Our experience is that there is an awful lot of solution details and preferences embedded in a lot of User Stories and their acceptance criteria.
- As with the idea of challenging requirements so explicit conflict management falls outside the scope of Agile processes.
- User Stories tend to be prioritised for delivery but not on the importance of getting them or some aspect of them right. So the concept of classification of need from this angle is not present.
- Concurrence whilst effective at small to medium scale can become an issue as the scale and complexity starts to exceed the capacity of the semi formal methods used to control scope change in Agile processes.

- There is no formal concept of concession.
- Consolidation does not exist; in-fact it could be positively discouraged as the simplistic interpretation fo the Agile ethos is that ‘code’ is the only real output of the process.

Situations where Agile is generally adequate

Given the analysis in this previous section where does a minimalist Agile implementation effectively guard against requirements issues? Given the non-uniform delivery across the twelve principles outlined above the first constraint is that it is only good where the principles it does not embody are less important. However that in itself is not enough. It is also constrained to situations where nothing compromises its ability to deliver on the principles it can deliver on.

Characteristics that we would generally view as prerequisites to meeting these constraints are:

- There are a small number of customer representatives that operate as a single team. This would imply an upper limit on the number of representatives of around six.
- The customer team is experienced, pragmatic and understands the limitations of technology in its application to their own domain.
- On the delivery side the number of first level team leads is low enough for them to form a single coordination and liaison team. This would imply an upper limit of around six team leads. With each of these managing a team of say six people this implies an overall upper delivery team size of around forty people.
- The customer teams and the delivery teams are collocated in a common working environment. Alternatively the customer teams and the coordination and liaison team regularly collocate.
- The domain is generally understood within the customer and delivery teams. There is a common body of knowledge and a shared set of concepts. There are a small set of primary business domains that are really important.
- Most of the novelty in the need for the new system or in the change to the system sits in the functional behaviour within the primary business domains that everyone understands.
- The development is analogous to some other system that people have experience of or represents incremental change to an existing system.
- Wider requirements outside of the core functional space are fairly stable and there is a shared understanding of what it good, what is acceptable and what represents failure in this space.
- The cycle time for concrete outputs from the development process is not too long. The two key levels of output here are hands on access for customer representatives in the lab and entry into live service or at least into a live trial state. Lab cycles should be less than a month; ideally fortnightly or quicker. Service or trials cycles should be less than a quarter; ideally monthly.
- There is continuity within the teams. Team membership is stable and is not subject to high turnover from cycle to cycle.

Where Agile processes start to need help

Where will such a process start to crumble? Characteristics that indicate that requirements problems are likely to materialise if additional practices are not adopted include:

- Situations where some groups work far more closely together than they do with others. Groups that work closely together develop implicit common knowledge that others lack, they agree things without involving others and they often fail to understand the need to communicate effectively outside their own inner circle.
- Where the customer actually consists of multiple customers with different, potentially competing objectives.
- Where the number of customer representatives increases to ten or more.

- Where the customer representatives do not have access to representative early development releases in a simulated lab environment or where releases to these environments are infrequent or lag a substantial time behind development.
- When there are important requirements that are not prima facia functional behaviour and that are not business as usual for this application or class of application. For example non-functional requirements or complex access control and audit trail functional requirements.
- If the delivery team lacks experience of the domain it is working in.
- Where the scale of the delivery organisation grows beyond size that the team leads can form a single coordination and liaison team.

Ways around this

Once these characteristics start to occur we either accept that requirements practices will not comply with the principles espoused here or modify the practices being used. There are two very distinct structures that can be adopted and whilst they may at first appear similar they are actually very different.

The first approach is to use a process that at the top level is structured and operates as an Agile software delivery process. Within that process are embed activities to deal with deficiencies in the way the minimalist Agile approach handles requirements. The second approach is to establish an appropriate systems engineering process with a suitable requirements engineering discipline built in and embed within the system engineering practice an Agile software delivery engine.

These two approaches are fundamentally very different and suit different situations. The first approach is suitable for situations that are close to the Agile norm outlined earlier. It is not suitable for enterprise scale high intensity delivery with team sizes in the hundreds; that is the territory of the later approach. Similarly the second approach is not suitable for close to Agile norm situations as by its nature it is less suited to these than an enhanced Agile model.

Once a general approach has been selected the detailed definition of the practices needs to be directed by the need achieve target levels of performance against each of the principles according. The targets are set by the level of risk that is associated with issues that could emerge around each principle. The things that have to be done to achieve an acceptable level of performance will be heavily influenced by the nature of the work that has to be done.

Background Information

This section provides background information on SQC and on the author of this article. Readers requiring more information can visit the company web-site at www.sqc.co.uk or send a request by email to enquiry@sqc.co.uk.

Author Biography

The author of this paper is the principal consultant at SQC. His career has focussed on delivery and assurance of complex IT solutions and high risk systems. His professional career began in 1985 and over the years has spanned development, testing, test management, test automation, delivery management and programme assurance. He began to focus on testing in 1991 and spent over a decade as a lead practitioner in this field. In the four year period to the end of 2009 he was the head of integration and test for the retail arm of the principal UK telecommunications provider. In this role he not only built and managed a function with a multimillion pound annual test budget but also served as one of the main programme leads shaping and managing complex IT programmes worth hundreds of millions of pounds.

His career has encompassed a great diversity of system types, development practices, project characteristics and organisation types. It has seen him leading both technical definition and effective delivery of a diverse portfolio of activities spanning system definition, design, implementation and test. The latest phase of his career has seen him shaping very large scale IT delivery programmes, providing independent programme reviews, driving recovery programmes and building and leading an enterprise wide integration and test function.

SQC

SQC originated as a supplier of software testing services; a provider of consultancy, test management, test delivery, load testing and test automation. SQC started serving this market in 1991. The organisation's expertise and field of operation has broadened over time to include the wider programme delivery domain. Today SQC's expertise spans from Programme Assurance through Test Management and Test Delivery via specialist test automation to technical testing. In these fields SQC can provide leadership, delivery management, service delivery, associated technical services, oversight, consultancy and training.