Using TRIZ to help manage real-world requirements

Andrew Martin - Oxford Creativity
andrew.martin@triz.co.uk
www.triz.co.uk

Introduction

The Capture, Understanding and Management of Requirements is, very properly, considered to be a crucial element in the lifecycle of a successful project or system. “Getting the Requirements Right” has become a systems-engineering mantra – and with good reason: most project failures can be ascribed, in part at least, to a failure to properly understand and manage requirements. As recalcitrant children we were told to eat our greens (“they are good for you”). As responsible adult engineers, we are exhorted to digest all the requirements before we can move onto the sweet delights of the design phase. Along the way we are spurred on by a litany of cheery maxims: “Get it right first time”, “Do it once and do it right”, “Requirements should be clear, correct and complete”. In many ways, we are encouraged to look upon Requirements as wild beasts that must be captured, tamed and securely tied down before it is safe to proceed.

Much of the thinking behind this rigorous approach is based upon the premise that all the requirements can be known, and once known will remain stable (or, perhaps, predictable) through the project lifecycle. But is the real world like that? Let’s explore the implications of working with requirements that are harder to tie down – those that cannot be fully understood, or that emerge or change during the project lifecycle – and consider some strategies for dealing with them.

We will use some TRIZ tools to help tackle both the understanding, as well as some possible solutions to the problem.

Requirements Change: Where and why does it happen?

We start by considering the various different circumstances that lead to requirements change and the nature of changes that result. We will then use this as framework to consider ways of tackling the problem.

This exploration of the source of requirements change is facilitated through the use of the TRIZ ‘Time and Space’ or ‘Nine Boxes’ thinking tool. This tool helps us broaden our perspective by considering a situation in two dimensions: time and scale. The time dimension is typically represented in terms of past, present and future, while the scale dimension uses the concept of system scale or hierarchy: super-system, system and sub-system. We represent this generic scale as a 3x3 matrix, like this:
Each ‘box’ in the matrix relates to a specific partition of the overall situation. By dividing up the situation in this way we gain three important benefits:

1) it broadens our outlook by encouraging us to consider all aspects of the situation.

2) it allows us to focus on each box, confident that we will not miss out consideration of all of the other boxes.

3) it enables us to get both of the above benefits without them conflicting with each other. The nine boxes are both comprehensive in scope and limited to a manageable number. Thus they both encourage divergence, but limit it in a way that allows us to see detail without losing sight of the context and wider implications of the situation. Thus we simultaneously obtain the benefits of divergence (good for considering all the possibilities) and the benefits of convergence (good for focussing our attention and creativity) whilst also retaining an appreciation of the overall situation.

This ‘nine-box’ or ‘time and scale’ thinking can be applied to many different aspects of a situation, including:

- Situation understanding – mapping the context of the problem
- Problem definition (what do we want vs. what do we have in each box)
- Causes mapping (capturing all the possible causes of a problem)
- Resource capture (understanding available resources enables us to develop good (more ‘deal’ in TRIZ terminology) solutions to problems
- Solution capture (forces to look for solutions in all boxes)
- Situation communication (an excellent way of summarising a complex situation in one simple diagram)
Often it is used in a combination of all or some of the above. The Time and Scale map is a great aid to anyone trying to understand, communicate and then solve a problem.

To start with, we shall be using it to help situation understanding, or more specifically, understanding the scope of sources of requirement change. Before we can make full use of the nine-boxes we must first tailor the axes to reflect our situation of interest – in this case: requirements change.

For the time axis we consider three different periods in which requirement change can occur:

- Pre-design (including the traditional requirements capture phase)
- Design/implement phase (while the system is being designing and implemented)
- Post-delivery phase (while the system is deployed and in-use)

For the space axis we consider the source or driver behind requirement change:

- Changes in the environment outside the system, in many cases our control over these requirements is limited or non-existent.
- Changes in the system requirement e.g. an additional function is required. These changes are more or less directly under our control.
- Derived requirements that come about as a consequence of our implementation.
Our tailored nine boxes now look like this:

<table>
<thead>
<tr>
<th>Super System (e.g. environment)</th>
<th>Requirements Capture Phase</th>
<th>Design &amp; Implement Phase</th>
<th>Post-Delivery Support Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Requirements</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Derived or Sub-System Requirements</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We can now populate our nine boxes, considering each box in turn and using it to collect all causes of requirement change that fit the associated combination of ‘time’ and ‘space’. We include anything that has the potential to generate change directly, and/or indirectly, as a result of being subject to change itself.

Note that the term ‘Stakeholder’ used here includes all those persons or institutions that have an interest in the system – including Users, Purchasers, Designers, Maintainers and others.
### Super-System

**Stakeholders**
- Operating Environment
- Operating Policies
- Customer Funding
- Customer Preference
- Legislation
- 'The Market'
- Competitors
- Suppliers
- Partners
- Stakeholder Policy
- Stakeholder organisational changes
- New Opportunities
- Synergy with other projects or products
- Infrastructure
- Available Technology
- Badly expressed requirements
- What Stakeholders know
- What Stakeholders don’t know
- Bad assumptions

### System

**System requirements**
- Interpretation of system requirements
- Clarification of system requirements
- Missing Requirements
- Interfaces
- Available technology
- What we know
- What we don’t know
- Bad assumptions

### Sub-System

**Available technology**
- Available components
- Available suppliers
- Available partners
- Supply chain
- What we know
- What we don’t know
- Bad assumptions

### Requirements Capture Phase

- As for Requirements Capture Phase (left), plus the following:
  - 'Lessons learnt' by Stakeholders during project so far
  - Dependencies on other internal projects:
    - Technical
    - Schedule
    - Cost
  - Dependencies on other external projects (as above)
  - New Technology

### Design and Implement Phase

- As for Requirements Capture Phase (left), plus the following:
  - 'Lessons learnt' by Stakeholders through experience of using and maintaining the system.

### In-Service Support Phase

- As for Design and Requirements Capture and Design/Implement Phases (left), plus the following:
  - Need for changed or new functionality

---

### Requirements Interpretation

<table>
<thead>
<tr>
<th>Super-System</th>
<th>System</th>
<th>Sub-System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stakeholders</td>
<td>System requirements</td>
<td>Available technology</td>
</tr>
<tr>
<td>Operating Environment</td>
<td>Interpretation of system requirements</td>
<td>Available components</td>
</tr>
<tr>
<td>Operating Policies</td>
<td>Clarification of system requirements</td>
<td>Available suppliers</td>
</tr>
<tr>
<td>Customer Funding</td>
<td>Missing Requirements</td>
<td>Available partners</td>
</tr>
<tr>
<td>Customer Preference</td>
<td>Interfaces</td>
<td>Supply chain</td>
</tr>
<tr>
<td>Legislation</td>
<td>Available technology</td>
<td>What we know</td>
</tr>
<tr>
<td>'The Market'</td>
<td>What we don’t know</td>
<td>What we don’t know</td>
</tr>
<tr>
<td>Competitors</td>
<td>Bad assumptions</td>
<td>Bad assumptions</td>
</tr>
<tr>
<td>Suppliers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partners</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stakeholder Policy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stakeholder organisational changes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>New Opportunities</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synergy with other projects or products</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Infrastructure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Available Technology</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Badly expressed requirements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>What Stakeholders know</td>
<td></td>
<td></td>
</tr>
<tr>
<td>What Stakeholders don’t know</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bad assumptions</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### Lessons Learnt

- Project schedule
- Project resources
- New Technology

---

### Obsolescence

- Interaction between users/maintainers and system components
For example, we include ‘Stakeholders’ (first entry, top left-hand box), because they are a direct source of change – sometimes as a result of an initial lack of knowledge or understanding of their needs. We also include them because they are themselves subject to change – a not uncommon situation that also leads to requirements change.

Having completed our nine boxes, we can see that there are a variety of possible sources and causes that would lead towards requirement change, across the whole project lifecycle. Any of the factors identified in any of the boxes is subject to potential change, leading to a corresponding knock-on effect on the requirements.

No doubt the set of causes identified here is not exhaustive – but it serves to illustrate the broad nature of the problem. Not surprisingly, the Super-system boxes are particularly well filled – and, as is so often the case with such things, these are not only the most numerous, but also those over which we have the least influence and control. There is also a trend across time: as the lifecycle proceeds, the range of possible requirement sources widens (in general, each phase is susceptible to changes from all of the sources identified for preceding ones, plus more).

Note too that we have included factors such as ‘What we don’t know’ and ‘bad assumptions’. Like that of perfectly fixed requirements, any expectation of perfectly understood requirements is also naïve.

We have also included a lot of implementation issues as potential requirement change sources. At first sight this may appear questionable - after all, a fundamental tenant of systems engineering is that the requirements and the implementation should be kept separate – to muddle the two is (very properly) considered to be a ‘bad thing’. However, here we are concerned here with the possible sources of requirement change and the unhappy reality is that system implementation often does influence requirements. For example, many requirement specifications are written with an eye to the art of the possible. A less comfortable, but pragmatic example is the case of a project that, for whatever reason, arrives at a point where it is unable to meet the original requirements. In this case although the original requirement remains, the practical specification of the system may have to change – and it is that change that the project team will have to deal with.

When we look at this analysis we might begin to wonder how any project can hope to be successful in the face of such capricious aims.

**In a World of Shifting Requirements, What Can We Do?**

We have established that requirements will change. What should our response be? What should we do about it?

The first, and most crucial, step is adopting an appropriate state of mind – one that will inform everything we do: both in the design of our processes and in the implementation of our project.
This state of mind demands that we:

1. Acknowledge that requirements will change.
   (to expect anything less is immature and naïve - get real)
2. Anticipate requirement change
   (get ready)
3. Be adaptable
   (expect the unexpected)
4. Be positive
   (this is an opportunity as well as a problem)

The last of these is not immediately obvious. Variable requirements might be seen as a ‘bad thing’; however they can present opportunities as well as threats. If the requirements change and we are able to meet them, then the system/product we create will be all the better for it.

But how does this translate into practical measures? What do we do?

Clearly requirements change is something that we should expect as a certainty rather than a possibility, and that the management of these moving targets should form the basis of our engineering processes.

To be fair, most systems engineering development environments contain requirements management processes (in many cases supported by tools) that enable requirements changes to be tracked and the consequential effects on derived requirements and the system design to be traced, identified and evaluated. These methods and tools are important weapons in the battle with requirement change – but can we do more?

We will use some TRIZ tools to help identify a range of other possible solutions.

**Solving the Contradiction: Fixed Requirements vs. Variable Requirements**

TRIZ uses the concept of a ‘Contradiction’ to describe any situation where we want two things that appear to be in conflict with one another. Conventional thinking would lead us to seek some sort of compromise in such circumstances. TRIZ, on the other hand, encourages us to seek solutions without compromise – and then goes on to help us discover them.

TRIZ classifies contradictions into two types:

- **Technical Contradictions**, in which the conflicting needs are related to different system parameters (e.g. I require a piece of furniture to be strong AND light)
- **Physical Contradictions**, in which the conflicting needs relate to the same system parameter, but in an opposing or opposite way (e.g. I want my umbrella to be BIG to protect me from the rain AND I want it to be SMALL when I’m not using it).
TRIZ first encourages us to accept the existence of such an unreasonable-sounding state of affairs. By acknowledging that we may be able to solve this apparent contradiction, and have both then we take the first step towards to achieving it. TRIZ provides a number of systematic methods for tackling these apparent contradictions and resolving them.

In our case we are dealing with a Physical Contradiction: we want requirements to be fixed and variable (unfixed). We want fixed requirements because it makes the project easier and simpler. We want variable requirements because this reflects the real world, and will mean that the system we create will be more likely to be useful and acceptable.

One TRIZ method for dealing with Physical Contradictions is based upon the notion of separating the contraction through the use of Separation Principles.

**Separation Principles**

The Separation Principles we will consider here are:

- Separation in Time
- Separation in Space
- Separation on Condition
- Separation by Transition to Super-System or to Sub-System

By way of an introduction to separation principles, we shall illustrate each by a simple example.

**Separation in Time:** A jet airliner requires wings that are both suitable for slow flight (high lift, high drag) and suitable for high-speed flight (low lift, low drag). However, it does not require both at the same time. So, by using a variable form wing (e.g. though the use of dynamic flaps/slats) that has one set of characteristics at one time and a different set of characteristics at another time it is possible to achieve both sets of desirable characteristics – i.e. separation in time. Another example is a board pointer used during a presentation: we want it LONG when pointing and SHORT when we are finished and want to store it away.

**Separation in Space:** We wish to design a wedge to hold a door open. We require the wedge to be THIN, so that it can pass under the door, but we also require it to be THICK– so that it prevents the door from moving. The simple design of a wedge solves this contradiction by being thin in one part and thick in another – i.e. using separation in space. Similarly a coffee cup needs to be HOT next to the coffee but COOL next to the hand and lips.

**Separation upon Condition:** A domestic vacuum cleaner sucks in a mixture of air and dirt. We want the air to pass through the cleaner, but we want the dirt to be trapped and collected within the clearer – so the cleaner must allow things to pass, and not allow things to pass. We therefore introduce a mechanism (such as a filter of some sort) that takes advantage of the physical properties of air and dirt to enable them to be separated in some way – i.e. separation on condition.
Separation by Transition to Super-System or to Sub-System: A bicycle chain needs to be flexible, so that it can conform to the shape of circular sprocket wheels and it needs to be rigid – because we wish to manufacture it using materials such as metal.

Our solution is to make it flexible at the system level (the chain), but rigid at the sub-system level (the individual links).

We will now try each separation principle in turn, identifying if it is applicable to our problem, and then using it to identify possible solutions to the dilemma of wanting requirements to be both fixed and changeable.

Solutions through Separation in Time

We ask ourselves the question: “Do we require the requirements to be fixed and variable at the same time?”

If the answer to this question is “No”, then Separation in Time cannot be used. However, if we feel that the answer could be “Yes”, then we can consider using the principle by dividing time into periods when requirements are allowed to change, and times when they are not. There are three possibilities:

1) Requirements are initially variable, but at some point they are frozen. This suggests a ‘traditional’ requirement capture phase during which Stakeholders refine the requirement set (perhaps through some sort of requirements review process) before fixing them.

2) Requirements are initially fixed, but after some point they are allowed to change. This suggests that there may be some benefit in allowing the project to proceed rapidly against a fixed (but possibly incorrect or incomplete) set of requirements, and then use the experienced gained (e.g. through prototyping) to support the introduction of new or different requirements later on.

3) Periods of fixed and variable requirements are interleaved in some way. This suggests some sort of hybrid approach, with managed periods of implementation and requirements capture.

Clearly there are a lot of different possibilities here, but some techniques that could be derived from this are:

- Iterative requirements definition
- Iterative development periods
- Phased requirements reviews (possibly interleaved with design reviews)
- ‘Gates’ for both requirements definition and implementation
- Deferring the definition of some of the requirements until later in the project lifecycle
Solutions through Separation in Space

We ask ourselves:
“Do we require the requirements to be fixed and variable across the whole requirement set?”

This question suggests that it may be possible to partition the requirements into those that we will allow to change (or we consider more likely to change) and those that we will consider to be fixed (or less likely to change). This way we can focus our preparations and efforts towards managing those requirement changes that present the greatest threat and/or opportunity.

This could be exploited in a number of ways, for example:

- ‘Ring-fencing’ of certain requirements
- Associating some sort of ‘stability’ attribute to each requirement and use this as part of our analysis of the ‘change susceptibility’ of the various derived requirements. This, in turn, could inform the phasing of our development: deferring the development more susceptible parts of the system.
- Applying some of the separation principles discussed in this paper selectively according to an assessment of the stability of each requirement.

Solutions through Separation upon Condition

This principle suggests some sort of adaptability within either the process through which we deal with the requirements, the design of the system, or in the system itself. Let’s consider all three of these.

The concept of an adaptable process is both exciting and disconcerting. This is a further example of the tension between a fixed and rigorous way of doing things and a more pragmatic ‘real-world’ approach – yet another physical contradiction! But it makes sense: most practical processes require a degree of adaptability to be effective, and the more complex the context in which the process is required to operate, the more reasonable it becomes to accept that adaptability or tailoring of the process will be necessary. Applying this to the problem at hand, the separation upon condition principle suggests that we should consider using tailored or adaptable processes to manage requirements (and also, by inference, implementation). There is a wide range of possibilities here, but breaking away from the assumption that our processes must be fixed is the first step towards realising them.

Moving onto the second application of the principle, the concept of an adaptable design looks promising. It suggests that we somehow create a design that can be readily modified to meet the demands of changing requirements (note that, for the moment, we are discussing the design information, not the system that is being designed). This suggests such techniques as:
• Automated requirements management systems to trace dependencies between requirements and design.
• Use of automation in the design process, particularly systems that allow the system to be defined at some higher level of abstraction, leaving an automated system to fill in the detail.
• Creating an intermediate system that builds the final system for us, rather than directly building the final system ourselves.
• Incorporate flexible components into the system design, such as ‘programmable’ components or human beings.

Finally let us consider a system or product that is, in itself, adaptable. That is, we create a system that can adapt to meet new or changing requirements. The programmable computer is an obvious example. Containerised transport solutions are another – the container contains an inherently flexible space into which a variety of goods can be placed, without affecting the functionality of the container handling infrastructure.

Solutions through Transition to Super-System or Sub-System

Finally we consider solutions based upon transition to the Super-System or Sub-System.

This suggests the possibility of trying to design our system in such a way that the effects of requirements change on the system implementation are constrained and contained as far as is practical. Much of good system engineering and software engineering practice is based upon this principle: minimising the interdependencies between sub-systems and the dependencies between the requirements and the implementation. We have already noted that the majority of requirements change sources are beyond our control, however we do have much greater control over the way we design and implement the system itself. We can use this flexibility in the design to create sub-systems whose requirements enable them to adapt or stretch to meet possible changes in higher level requirements. This suggests such things as:

• ‘Object Oriented’ design techniques
• Loose coupling between subsystems
• Minimising dependencies between subsystems
• Using intermediaries between subsystems to mitigate against the effects of interface changes
• Over-specifying sub-systems (tolerance tiering or stretch targets)
• Specifying scalability and/or adaptability into sub-systems
• Flexible designs or architectures that can be reconfigured
• Increasing system granularity, to promote scalability or adaptability
• Specifying redundancy in sub-systems
• Careful use of off-the-shelf design elements or sub-systems
We can also use this principle of minimising interdependency at all levels of the system implementation, not just engineering. It should be applied to the project organisation, partnerships, sub-contracting, marketing - and so on.

**Summary**

Through the use of some TRIZ tools we have been able to (briefly) consider the scope of the sources of variable requirements. From there, we have gone on to identify a physical contradiction that lies at the heart of the problem:

*We want requirements to be fixed AND we want them to the variable (unfixed).*

We have then used some TRIZ separation principles to explore some possible solutions to resolving this apparent dilemma. We have seen how many of the systems engineering techniques necessary for survival in a landscape of shifting requirements can be ‘discovered’ through this method.

The TRIZ techniques of the ‘nine-boxes’ and Separation Principles for the resolution of Physical Contradictions are simple, yet powerful, tools that can be used to tackle a wide range of problems – both technical and non-technical. The requirement management example demonstrates how apparently dilemmas can be analysed and tackled in a systematic way to yield a range of possible solutions.

**References**