RSBA Technology Ltd

http://rsbatechnology.co.uk

# A Competitive Engineering Case Study: Price Sentinel

This paper describes a single case study of a project management approach known as Requirements Engineering or Competitive Engineering ("CE"), developed by Tom and Kai Gilb. It demonstrates the benefits of re-casting stated business requirements as quantified stakeholder value objectives, focusing all project effort on maximizing the improvement of those value objectives for minimal cost by delivering real measurable improvements to those stakeholders early and often.

Please note that all requirement examples and numeric requirement values in this paper have been re-created and altered from the original proprietary documentation, solely for the purpose of illustrating the engineering concepts described herein.



## Background

A multi-national market-making investment bank was considering adopting a new messaging -based Service-Orientated Architecture (SOA) to upgrade and eventually replace an existing foreign exchange e-commerce client/server internal trading platform. Richard Smith, the Director of consulting firm RSBA Technology Ltd, was engaged to provide expertise on analysis, design and project management for such complex global trading systems. The customer had undertaken an initial feasibility study prior to our involvement in the programme. They had identified a candidate business use case to serve as a real-world proof-of-concept of the new architecture. Our primary task was to start designing the strategic system architecture and core design concepts, using the proof of concept requirements as a test-bed of the first delivered software components. However, it was also important to deliver some real improvement for this business function in production.

The initial business requirement was stated similar to the following:

*"Create a new "Price Sentinel" component that can detect if the bank's published FX customer quotations go off-market, and then to immediately cancel all current quotations."*

It was known that another competitor bank had suffered financial loss as a result of unknowingly publishing and executing FX trades with its customers based on off-market prices. The mechanics by which off-market prices, where the bank's prices differ sufficiently from the current known price in the FX interbank market can cause financial, loss is related to the way market-maker banks have to cover each customer trade in the interbank market. For the purposes of this paper, it is sufficient to understand that relatively small numeric differences between a bank's quoted price and the interbank market FX rate lasting for even just a few seconds can incur financial loss from individual customer trades; if the discrepancy persists for minutes or hours, the losses can rapidly accumulate.

Here, we outline the steps we went through to gain a working understand of the requirements and the environment in which any solution would be operating within. The general procedure we followed is based on Gilb's work on Planguage, and is described in much greater depth in their book Competitive Engineering, as well as many papers and articles elsewhere on their website (for example, this and this).

## The Five Whys

We knew that the "Price Sentinel" had two primary goals:

1. To meet a business requirement (see initial statement of requirement above)
2. To demonstrate use of new SOA / messaging – based architecture design

Our first task was to really understand the deeper business requirements behind the high-level goals, and figure out who really cared about them (our "project stakeholders"). As stated, the high-level requirement had several problems with it, including:

- Mixing a statement of *need* ("Detect …" and "cancel…") with a *design solution* "Create a "Price Sentinel component…".
- Unclear, ambiguous key term "off – market". How "off" does a bank quote have to be from which "market (price)" before it needs to be "detected"?
- Use of the word "immediately" without any indication of what level of "immediate" is important.

The requirement was more focused on the likely solution, "Price Sentinel", and its role as a proof of concept use of a new architecture, than as a clear, unambiguous, design-free statement of what the customer really needed.

Finding the key stakeholder in this case was easy: the head of the FX e-trading desk who had contracted our services into the bank, and his management hierarchy up to the global head of FX trading.

Uncovering the real business needs required further investigation. There is a simple analysis tool known as the "Toyota 5 Whys?"[1] that helps to discover the functions and qualities that the stakeholders really meant when they said they needed a "Price Sentinel". The idea is to repeatedly find the next reason above the current level of understanding that justifies that requirement statement. *In practice, one would never just say "Why"? five times. The questions can be more user-friendly and less annoying to the stakeholder participant: "Why do you think <insert senior stakeholder name here> expressed the requirement in that way?", "What do you really mean by stating requirement Y", "Why do you think requirement Y is so important ?"*

This is what we came up with in discussion with the stakeholder (add "Why …" to each answer to get the next question), after starting with "Why do you need a "Price Sentinel" ?":

- Answer 1: To prevent publishing off-market tradable prices
- Answer 2: To prevent incurring trading loss (in simple terms, by having to buy traded currency amounts in the market at a "higher" price than the bank agreed to "sell" those assets to the customer)
- Answer 3: To demonstrate to senior management that e-trading business can safely (without incurring unexpected loss) manage current levels of customer trading activity
- Answer 4: To ensure that senior management will agree to expand e-trading business in the future, based on past / current business performance to other customer segments and business areas
- Answer 5: To meet business and individual medium / long-term financial targets.

We stopped when we had reached the first level that was beyond the effective influence boundary of responsibility of the stakeholder and the related internal business unit, "Answer 5". For the purposes of this project, we did not have to question the financial targets imposed from higher up the chain of command.

We had now moved from a requirement which was really a statement of design, to a deeper understanding of the business drivers and rationale behind the "off-market detection and correction" function. We had identified something about the key stakeholder values that were to be improved by this project.

## First Requirement Re-writes

For the purposes of our project, we decided to explore further the stakeholder value described in Answer #1 "To prevent publishing off-market tradable prices". Our task was to discover and separate the business requirements into four requirement types:

[1] http://en.wikipedia.org/wiki/5_Whys

1. Function ("what")
2. Performance Requirements ("how well")
3. Costs ("how much")
4. Constraints ("boundaries of allowed change")

This is where the formalism of Planguage comes in, providing a simple meta-language to describe interesting and useful things about the requirements themselves. Straightforward though Planguage is, we find the most efficient technique is to iterate towards a more formal definition, as demonstrated below.

The first re-write attempt (version "0.1") went something like this:

"A System to cancel existing customer quotes if those quotes deviate from the market by a significant amount over a long-enough period of time"

Here, we tried to capture the essence of the business function ("cancel if deviate"), and acknowledging there is some level ("significant") of deviation if maintained for a "long-enough" period of time to cause us concern. We also changed the "Price Sentinel" design idea (whatever that is) to a generic "System", here meaning *any* collection of valuable design solutions. A good start, but doesn't provide a lot else for us to work with.

Introducing some Planguage [2] constructs into the picture, here's version 0.2:

"A System to cancel existing customer quotes if those quotes deviate from the market by a <significant amount> over a <long-enough period of time>"

cancel: defined as: send message to external customer indicating last-published quote is no longer valid
customer quotes: defined as: tradable FX price quotes published electronically by the Bank to Customers
deviate from the market: defined as: the numeric differences between the bid and ask prices of a customer quote and a <current> <consensus market bid / ask price> for a given trade amount

We provided clarification for the three phrases of interbank FX trading jargon, by underlining them in the statement and providing a definition of each (using the Planguage "defined as" parameter). Note that these are intended as clarifications to the assumed readership of the requirements, assuming a working level of domain knowledge. We quickly added a Glossary section to the

[2] Refer to Section 1.3, Table 1.1 (Gilb 2005) for a summary of the main generic Planguage parameters and concepts

requirements document, moving common re-used defined phrases into a single table. This promotes shared unambiguous understanding of domain jargon across the project team and beyond[3] (it is surprising how often the business stakeholders disagree over the meaning of some terms, making a Glossary doubly important!).

We also explicitly owned-up to the fact that we don't yet understand what some of the phrases such as "significant" and "long-enough period of time" are, by surrounding those phrases with Planguage Fuzzy angle brackets ("< … >"). These simply indicate, clearly and unambiguously, a term that is known to be defective and in need of improvement[4]. Anyone reading this statement must not make any assumption of their meaning until we have been able to "de-fuzz" those terms. However, they also cannot ignore them. Fuzzy brackets act as placeholders for further research or discussions with subject matter experts (SMEs), whilst still ensuring the requirement, *as currently written*, is already clear, concise and unambiguous. One of the most common and dangerous practices in authoring requirements, is to know you don't understand something, but fail to pass on that doubt to the people that have to make some decisions based on the specification.

The main initial analysis effort can now proceed, decomposing the requirement into further detail, using Planguage requirement types: Function, Performance, Resource and Constraint, as described individually below. Requirements are written across all four major types *at the same time* as a result of the analysis work (meetings, reviews, brainstorms etc). The key is to determine and recognize which type of requirement each statement of need falls into, and then label and document them as such.

The most important separation of concerns is between Requirements and Solutions (Planguage calls solutions "Design Ideas"). During any analysis process with real Stakeholders, all Requirement types and Design Ideas naturally get identified and discussed at the same time. Always document any Design Ideas that come up in discussion alongside identifying the actual Requirement; the crucial point is recognizing they are *not the same thing*! See the section on Design Ideas below.

## Function Requirements

Planguage defines a Function Requirement as "specifies that the presence or absence of a defined function is required. A function requirement is binary…"[5]. They represent the domain tasks the stakeholders require the system to do, without reference to any particular design (unless perhaps referencing a known constraint or boundary to the requirement scope). Think of it as determining

[3] After only two Planguage projects in the same domain, it becomes valuable to maintain a central Glossary, with a list of re-usable terms that can be copied into project-specific documentation.
[4] Beautifully and funnily expressed by Douglas Adams as "We demand rigidly defined areas of doubt and uncertainty" in The Hitch-Hikers Guide To The Galaxy
[5] See Planguage Glossary Concept #074 Function Requirement (Gilb 2005)

what the stakeholders would have to do even if they were just using pen, paper and a new-fangled telephonic device. A Function has no variability; it either is provided by the system, or it is not.

Planguage introduces the use of Tags[6] to uniquely and consistently label every engineering artefact. When considering the set of Functions that stakeholders are interested in, we can use the ideas of Tags to iteratively build up a hierarchy of function requirements. This technique really helps to focus on the "breadth" of scope we are interested before exploring each successive branch in more detail. Such a mind map - like approach provides an ever-growing scaffolding to fit in new functions as they are discovered during the analysis process.

Our first attempt at a Function hierarchy focuses on identifying the key top-level functions that form the basis of the functional scope of our project. We create a Tag label for each, and provide an initial Description of each:

**Detection**: detection of a customer quote that has deviated from the market
**Notification**: notification of <interested parties> that an off-market customer quote has been detected
**Correction:** Any corrective action(s) determined appropriate by a notified responsible party to resolve
**Reporting**: Retention and reporting of information related to the Detection, Notification and Correction of an off-market customer quote

Happy with this overall Function scope, at least for now, we proceed to the second-level function of each top-level scope requirement, here using a dotted notation familiar from some programming languages to show the hierarchical relationship. For example, here's a second-level breakdown for Detection, describing three sub-Functions:

**Detection**:
**Detection.Market Data Sources:** Obtain Relevant Independent Market Data to perform off-market detection.
<- J. Bloggs Analysis Meeting Notes 23 March 2011
**Detection.Customer Quotes:** Obtain Relevant Pricing Information on Current Customer Quotes to perform Off-Market Detection.
**Detection.Off-Market**: Determine if deviation of current Customer Quotes(s) from relevant Market Data indicates an Off-Market Event has occurred.

[6] See Planguage Glossary Concept #146 Tag (Gilb 2005)

Note the "<- J. Bloggs Analysis Meeting Notes 23 March 2011" annotation above. Every real-world project has multiple Stakeholders, from the end-users, IT project team, operational support. It is therefore very important to attribute statements of requirement to identified sources, such that those statements can be challenged and reviewed as the project proceeds. Planguage provides a "Source:" parameter, or a shortcut "<-" icon to annotate any statement of fact with its attributable source or evidence (who or what, when). Ideally, every atomic statement of fact should be attributed; however in practice, I find this is most useful for statements which, judgment and past experience indicate, may be considered controversial or debatable.

Normally, the overall shape of a 2 or 3 – level Function map coalesces very quickly from the first couple of analysis meetings, and remains stable as further detailed Functions are added (this stability arises because it is a description of the business domain, irrespective of any method to implement or provide those functions). Once the Function map stabilises, it becomes a very useful communication tool, even before most of the details have been worked out:

- use in analysis meetings with new Stakeholders, acting as a checklist or agenda for discussion,
- review and demonstrate progress to senior Stakeholders and programme management.

Ideally, the function map should be updated after every analysis intervention, so it represents a snapshot of the project's understanding of the function scope. In the first few iterations, the map will just be textual, as the examples above show. However, it may also be useful to invest time in producing and maintaining a graphical representation of the function map (or find a tool that can generate them automatically). This can be an effective communication tool during discussions on scope coverage.

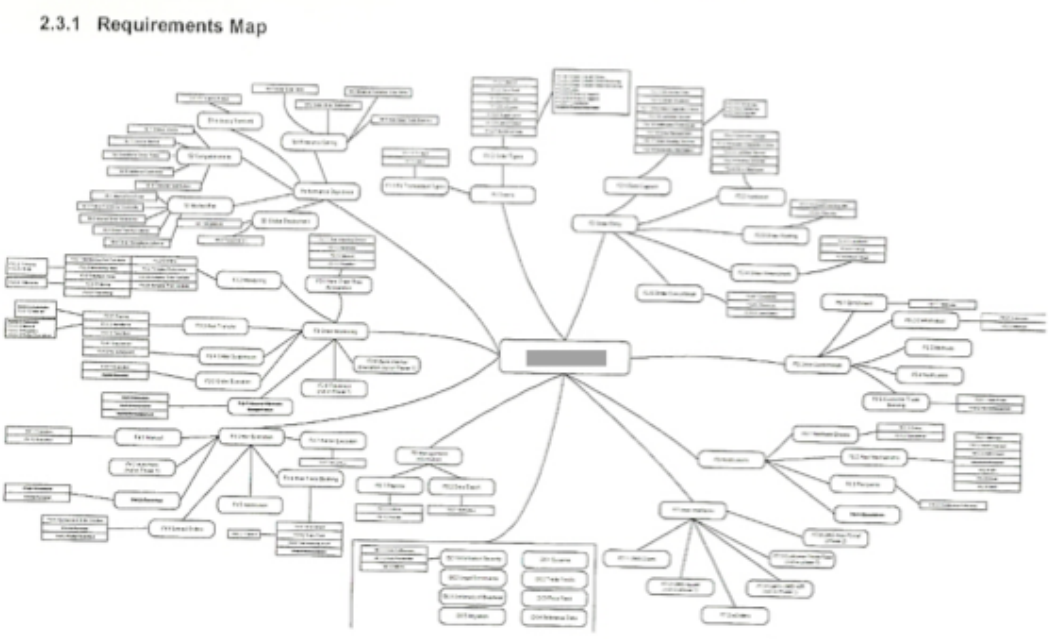


Figure 1 – Function Requirements Map

## Function Requirements Detail

During the analysis process, we continued to proceed to drill down further for each sub-Function, discussing and determining scope and definition of each sub-Function with the Subject Matter Experts and Stakeholders. Further domain-specific terms (e.g. "Off-Market Event") were added to the project Glossary, documented and clarified, de-composing into further detail if that appears useful. Remember that the focus here should be on identifying what the system has to do to meet the stakeholder requirements. Design ideas that arise during discussions should be noted for later investigation. Use any proposed design ideas together with the "5 Whys" technique as a means of reaching a deeper understanding of the stakeholder functions.

The final stage for documenting function requirements is to expand their individual definitions, enriching them with Planguage requirement attributes:

- attributes to help with managing each requirement's lifecycle (Version, Owner)
- attributes proving traceability back to stakeholders and other functions (Stakeholders, Supra-Functions, Sub-Functions, Supports)
- attributes to document known uncertainties (Assumptions, Risks, Issues)

Planguage defines a full template for defining Function Requirements (as for all other Planguage artifacts)[7]. However, organisations can and should adapt these for their own purposes. If not all attributes are explicitly stated, they should at least be used as checklists. For example, in the Requirements document used in the Price Sentinel project, attributes such as Assumptions and Issues were listed in dedicated tables at the front of the document, with references (using the requirement tags) to the associated requirement definition.

| # | Tag | Owner | Version | Status |
|---|---|---|---|---|
| F2.4.1 | **Corrective Action.Initiation.Manual** | A. Senior Trader | 0.6 | DRAFT |
| Description: | The System shall allow Authorised eFX Trading Users to trigger ad-hoc Corrective Action(s) at any time during Scheduled System Operation.<br>*eFX Trading Desk have oversight responsibility for intra-day trading platform behavior and so must be given means to jump in and take action even though any automated detection and correction mechanisms may not have noticed off-market event has occurred <- A. Senior Trader 23 March 2011*<br><br>Authorised eFX Trading Users: Named members of the {London, New York, Tokyo} eFX Trading Desks , responsible for business operational management of the eFX Trading Platform, who have been permissioned on <Bank security and access system> for this function. | | | |
| See Also: | #P1.2.0.1 Action.Time | | | |
| Stakeholders: | eFX Desk STM, Trading Compliance Department | | | |
| Related Design Ideas: | #DI2.2 Kill GUI, #DI2.4 All-Off Price Panel Control | | | |
| Assumptions: | #A3, #A4 | | | |
| Issues: | #I4 (*constraints on representing permission structure in <Bank security and access system>*) | | | |

Notes on the Function Requirement:

1. We used a concise numerical tag (#) in addition to a textual one, to make it easier to refer to this requirement in other project documentation. The "F" prefix indicates this is a Function -

[7] Page 106, Section 3.9 (Gilb 2005)

type requirement (this is explained in a usage and conventions subsection in the document Introduction). This is not a Planguage – suggested idiom, but a good example of how additional conventions and extensions can be utilized by different organisations.

2. More use of italicized supporting notes and the "<-" Source attribute enhances the traceability of this requirement back to the original key stakeholder
3. Embedded defined terms ("Authorised eFX Trading Users") are used to clarify which users can take manual corrective action. In this draft version, the term is left underneath the requirement itself. As the analysis proceeds, it may be useful to lift move this into a separate shared catalogue or Glossary, particularly if it turns out this becomes a central concept for the set of Solutions.
4. The "Related Design Ideas" attribute can be used to highlight previously identified Design Ideas that are particularly relevant to a Requirement. Perhaps a particular analysis meeting spent some time discussing this Function and brainstorming some initial ideas; it is useful to keep that connection between Requirement and Design Ideas together to enable either to change as the analysis progresses.
5. Assumptions and Issues have references to details defined elsewhere (e.g. in "Assumptions" table at start of document), making them easier to review and track together in one list
6. Use of the Planguage "set of" syntax { …. } to indicate a list or set of equivalent defined elements. By explicitly listing those Trading Desks in scope, we are similarly stating that, say, Singapore eFX Trading Desk is out of scope of this requirement (which may warrant an interesting phone call with the affected Stakeholders in Singapore when they get to review the document … far better to prompt those difficult conversations early as possible).

## Quantified Stakeholder Objectives

The core of the Planguage approach is to focus all project effort on delivering known, measurable value improvements to stakeholders early and often; any other use of project resources (time, people, money) is wasteful. The first step is to help the stakeholders determine their key improvement objectives for the project; the reasons for deciding to spend money on it.

As we saw in the "Five Why's" section, the objective was to provide new mechanisms to improve the ability to detect and prevent unexpected trading losses as a result of Off-Market Events from occurring in the Bank's eFX trading system. We needed to define "improve" in sufficient detail to help us make the smartest possible solution decisions, given all of the information available.

We had already started to breakdown the functions into a hierarchy of requirements. These are a great place to start looking for the most impactful qualities of critical functions we wanted to focus our improvement efforts on.

For the function of "Detection", we came up with an initial stakeholder wish-list list of statements (in Planguage, these are canonically called "Ambitions"[8]):

P1.1 Ambition: Minimise time taken to detect that an Off-Market Event has already occurred
P1.2 Ambitions Ensure mechanism(s) for Off-Market Event detection are at least as available as the trading system itself
P1.3 Ambition: Maximise independence of Off-market Event detection mechanisms from the trading system

We have found three qualities that the stakeholders agreed they really care about improving. For many software projects, this is as detailed as it gets. But, using Planguage, we can work on defining these in sufficient detail to base the entire project on.

The next step is to re-formulate each Ambition statement as a statement of something specific that we could, in theory, measure. How would the stakeholders perceive that their wishes had been granted? The key idea is try and express the wish list using numbers. Planguage calls these numeric statements "Scales"[9]. For now, we don't care whether we have any idea what the numeric values should be, or whether we have a clue how we could ago about measuring them in our system. Gilb's Competitive Engineering book dedicates Chapter 5 on how to formulate Scales (of Measure), and this chapter is also publically available on www.gilb.com[10]

Our first try at writing a Scale for #P1.1:

#P1.1 Scale: Once an Off-Market Event occurs in our trading system, how much time does it take for us to realize that it has happened?

A good start, but not yet much improved over the Ambition statement. Let's introduce another key Planguage concept: *qualifiers*[11]. Qualifiers let us explicitly acknowledge there are different scenarios or circumstances being affected. We want to allow for the ideas of specific things being affected in specific places over given periods of time. Also, remember we are trying to define something we could put an actual number to. We need to say something about that number (e.g. in what units will be expressed and measured in) to prevent misunderstanding of what we mean by an improvement of "3".

[8] See Planguage Glossary Concept #423 Ambition (Gilb 2005)
[9] See Planguage Glossary Concept #132 Scale (Gilb 2005)
[10] http://gilb.com/dl26
[11] See Planguage Glossary Concept #124 Qualifier (Gilb 2005)

#P1.1 Scale: Time, in seconds, from occurrence of an [Off-Market Event type] for [FX product] in [Location of Trading System] to Detection *(#F1)*, measured during a [Time Period]

Off-Market Event Type: {Published Customer Quote, Pre-Published Quote, Hedge Order}
FX Product: {G10 Spot, EM Spot, Any Forward Outright, Any Swap}
Part Of Trading System: {London, New York, Tokyo}

We added three qualifiers, and enumerated some of their values. We referenced the previously-defined function requirement of "Detection". The requirement has evolved from an imprecise "Minimise time…" to a clearer statement of:

- How will this improvement be measurable to stakeholders ("Time, in seconds")
- When do we start the stop clock ("occurrence of …") and when do we stop ("… to Detection")
- What types of Off-market Events, FX prices and Trading System locations we care about improving in this way.

Finally, we need to capture the stakeholders desired amount of improvement to be achieved by the project by putting numbers against the Scale. To achieve this, we utilize a set of Planguage parameters associated with such scalar requirements:

- Reference points from past or current systems ("Past", or "Benchmark")
- Required good-enough future levels ("Goal")
- Scale measurements below which the stakeholders would consider the function unusable, or worse than current ("Fail")
- Ambitious nice-to-have improvements beyond this project's goals ("Stretch") if sufficient resources were available, without committing to deliver this level.

| # | Tag | Owner | Version | Status |
|---|---|---|---|---|
| P1.1 | **Off Market Event.Detection Interval** | A. Senior Trader | 0.2 | DRAFT |
| Ambition: | Minimise time taken to detect that an Off-Market Event has already occurred | | | |
| Scale: | Time, in seconds, from occurrence of an [Off-Market Event type] for [FX product] in [Location of Trading System] to Detection, measured during [Time Period]. | | | |
| Benchmark: | [Published Customer Quote, EM Spot, Another London Bank, Autumn 2010] more than 14400 secs *<- Known market incident at A N Other Bank October 2010 (A Senior Trader)* | | | |
| Past: | [Published Customer Quote, EM Spot, London, February 2011] 600 ± 30 | | | |

| | |
|---|---|
| | secs *<- THB Incident 3 Feb 2011 (A Senior Trader)* |
| Goal: | [Published Customer Quote, {EM Spot, G10 Spot}, {London, New York}, end Sept 2011] 1-5 secs?? *<- A Senior Trader 23 March 2011* |
| Goal: | Goal[Published Customer Quote, {Any Forward, Any Swap}, London, end Dec 2011] 30 secs *<- A Senior Trader 23 March 2011 Forward and Swap pricing less time-sensitive to off-market deviations <- A N Other Trader 25 March 2011* |
| Stretch: | [Published Customer Quote, {Any Spot}, London, end Dec 2011] <0.1 secs *"Estimated state-of-art" <- Mr. Analyst 24 March 2011* |

Notes on the Performance Requirement:

1. Each target requirement has been specified with a set of qualifier values to provide the correct context for interpreting the scalar requirement value.
2. An additional "Benchmark" target has been used to declare a piece of relevant market knowledge known to the Stakeholders.
3. A "Past" target defined the sets the zero-point for improvements in this objective. Here we also recorded the magnitude of the likely error in the Past measurement (the relevant person could only remember it as "about 5 mins"). The measured value and the size of the likely error provides an opportunity to precisely challenge the measurement of how bad the current system was ("I recall it was 10 minutes")
4. Two "Goal" targets have been declared with different sets of qualified parameter values. This is a concise, unambiguous method of declaring what size of improvement in which critical set of circumstances (thing, place, time) we want to focus the current project effort on. There are other valid combinations that have not been defined; this requirement simply says that achieving improvements in just these two scenarios is sufficient for the project to succeed. Of course, this assumption is open to challenge by other Stakeholders; declaring these critical scenarios as clearly as possible enables such useful conversations to occur as early as possible in the project lifecycle.
5. The first goal declares the target as "1-5 secs??". The "??" is another Planguage term approximating the degree of uncertainty in the associated statement … you can use ?, ??, ??? for varying degrees of uncertainty (interpret these as you wish). Another valid way of writing this would be "3 ± 2 secs", which helps when building an Impact Estimation Table (see below).
6. A "Stretch" requirement has been recorded from information given by a member of the project team, based on knowledge of likely state of art of current technology. The key point is that it is less ("<0.1 secs") than the targeted Goal levels set by the Stakeholders. Planguage defines a Stretch target as nice-to-have, but without a commitment to deliver. This is generally because we don't yet know what the cost will be … typically the closer to state-of-

art we get, the cost rises exponentially. The project team can use this as a guide for the "strategic future" target, which may inform their short – to- medium term design decisions.

A similar analysis was undertaken of the remaining Detection – related performance requirements, and of the other key stakeholder objectives. In all cases, by some creative and lateral thinking, we found a way of expressing each objective as a scalar requirement with Past and Goal targets, albeit with varying levels of uncertainty. For instance, we decided to define the degree of independence of the new detection system(s) from the old system by reviewing the current trading platform and counting the number of discrete elements involved in processing customer quotations (let's say it was "8" <significant discrete system components>). The "independence" Goal could then be defined as "Maximum percentage of shared components", with appropriate thing, place and time – based attributes, as above.

The above examples demonstrate how a project can start to control critical value improvements using a few simple well-defined Planguage concepts, terms and constructs. Compare the half-page requirement description with the one-line stakeholder wish we started with. Planguage doesn't give you the answers, but it does provide a concise, clear language in which to express the critical questions.

## Constraints And Resource Budgets

The third requirements category defines the boundary beyond which the project is not empowered to propose change. We must define the wider context in which our improved System shall operate. Which other systems, teams, rules does our system depend on? But this category also extends to specifying the resources allocated to be spent on the project. An allocated project team of 20 competent people should achieve significantly more than a team of 2. A budget of $10m can buy a lot more "System" than a budget of $20k. We cannot begin to make any solution decisions unless we know (a) what we can / cannot change and (b) how much resource (people, time, money) we have to play with.

Planguage captures these concepts as additional Requirement types, including:

- Design Constraint[12]: an explicit and direct restriction regarding a design idea to be either compulsory or to be excluded. These commonly arise from previous system development and implementations. A Design Constraint is a binary requirement: it is either imposed on the project or not.

[12] See Planguage Glossary Concept #181 Design Constraint (Gilb 2005)

- Resource[13]: A scalar (numeric) requirement defining the amount of a given resource to be allocated to the project (people, time, money).
- Conditional Constraint[14]: requirements that impose a conscious restriction on a specified part of the project scope. Some kind of problem may be experienced if such a constraint is breached. Common sources of Conditional Constraints are from design, legal, market, geographic, safety and language. For example, a business unit may have had a deadline ("by July next year") imposed on it by an internal compliance unit to resolve a given issue that the project is intended to achieve. This would be a key requirement that may affect every design decision for the project's duration; it makes it very clear why compromises had to be made to meet such a deadline that otherwise would not have been made. If all stakeholders agree with the Constraint, no problem. However, by simply writing it down alongside the goals of the project, it provides opportunities for further challenge … "Hold on, I see the potential impact of this on the project, let's have a meeting with the Compliance department".

On the Price Sentinel project, we identified several key Design Constraints, based on the existing Systems with which it had to interact with, such as available sources of market data and the existing trading platform that had to be monitored. We could discuss and agree these with the relevant Stakeholders (both business and technology) as the project was initiated. By documenting them as project Requirements, this would also give us opportunities in the future to challenge and even change the nature of the Constraints, as circumstances dictate.

| # | Tag | Owner | Version | Status |
|---|---|---|---|---|
| DC1.1 | **Market Data.Sources** | A. Senior Trader | 0.2 | DRAFT |

Description: The following possible sources of Market Data may be available for use in Detection of Off-Market Events:

1. EBS AI
2. Reuters D3K
3. Currenex
4. Bloomberg

*"Although all of these sources are currently used by the trading platform, it may be possible to build new adapter software and / or separate network links to maximize independence of detection" <- Head of IT 22 March 2011*

[13] See Planguage Glossary Concept #199 Resource (Gilb 2005)
[14] See Planguage Glossary Concept #498 Condition Constraint (Gilb 2005)

The Resource requirements are probably the most familiar type of scalar requirements defined on software projects: how many people, how much time, how much money available for capital expenditure. We re-formatted these budget requirements as Planguage scalar requirements, to ensure the project's value improvements could be compared directly with its resource budget levels. Also note that Planguage emphasizes that it is important to capture the total cost over the lifetime of the system, rather than focusing only the upfront development expenditure. This can be quite difficult to estimate and even define targets; however I have found it possible to at least estimate maintenance costs over a single organization budget cycle (e.g. 6 or 12 months).

| # | Tag | Owner | Version | Status |
|---|---|---|---|---|
| R3.1.1 | **Budget.Work.Team** | Head of IT | 0.5 | DRAFT |
| Ambition: | Deliver initial scope by core London eFX team within next budget cycle | | | |
| Scale: | Number of work-days spent by [London eFX Core Team Role] on Price Sentinel improvements during [Budget Cycle]. | | | |
| Past: | [All team, all previous cycles] 0 work-days | | | |
| Goal: | [Developer, H1 2012] 25 ± 10 work-days *"approx. 1/4 to 1/3 available time over next 6 months" <- Head of IT 26 March 2011* | | | |
| Goal: | [Tester, H1 2012] 15 ± 5 work-days *"up to 3 or 4 weeks" <- Head of IT 26 March 2011* | | | |
| Goal: | [Project Manager / Other, H1 2012] 10 ± 5 work-days | | | |

## Capturing Design Ideas

So far, the emphasis of this paper has been to demonstrate how to use Planguage to capture the critical design-free requirements that will control the decisions made for the duration of the project. The separation of requirements from design is critical for many reasons, but rarely occurs. Arguable the biggest benefit of design-free requirements is that they provide both a clear, unambiguous statement of stakeholder needs, but also an opportunity to propose *any* set of solutions that meet those requirements.

Planguage refers all proposed solutions as "Design Ideas"[15], although these may not all involve writing software: a design idea maybe any interesting device, system, procedure or other change that may help resolve one or more requirement. They can and should be identified and discussed from day 1 of a project, but always clearly marked as Design Ideas, never as requirements (unless they are really a Design Constraint – see discussion above). Experience on this and other projects invariably shows that coming up with interesting ideas is never a problem; the challenge is to step

[15] See Planguage Glossary Concept #047 Design Idea (Gilb 2005)

back from the ideas to uncover the actual requirements (see the 'Five Why's" section for more on this).

Similarly to capturing requirements, we started out with a simple tagged list of Design Ideas that was added to during early meetings, for example:.

**"Kill" GUI**: separate desktop GUI to allow eFX Desk to remotely trigger shutdown of customer pricing components themselves

**Remote Task Server**: message bus application receiving "Kill" instruction messages from "Kill" GUI, execute pre-defined operating system script to stop/kill customer quoting application.

**Price Sentinel**: message bus application receiving market data and customer quote data, publishing off-market event notifications back onto message bus

**Independent Market Data Adapters:** message bus applications connecting to external FX market data sources, publishing normalised rates onto message bus for use by Price Sentinel

**Market Data Connections**: Set up separate bound network connections to market data sources for use by Price Sentinel

**Dedicated Servers**: Deploy Price Sentinel – related applications on separate dedicated servers

We then expanded each item into a more detailed description. At this stage, we are only looking for a paragraph or two on each, with maybe a diagram if that helps. The level of detail needs to support the estimated value improvement impacts and costs of the idea. Planguage suggests a set of useful additional attributes for Design Ideas that help focus on critical information for each idea:

- Design Constraints: if this design attempts to adhere to a known Design Constraint, note which Constraints are being met
- Impacts [Functions]: List which business Functions which this design is intended to affect or fulfil (use the Tags of the relevant Function requirements)
- Impacts [Performance]: List the main performance objectives which this Design Idea is intended to improve significantly (ie. the main rationale for proposing a particular idea)
- Impacts [Side Effects]: List of any performance objectives which may be partly impacted, but do not form the primary focus of this Idea
- Impacts [Cost]: List of Resource requirements this Idea will utilise in a significant way
- Related: List of any related Design Ideas
- Assumptions / Risks / Issues: same as for Requirements

Again, you may choose not to use each of the 43 suggested parameters in the Planguage Design Ideas template, but should at least be used a checklist for each Idea to help think about everything that is likely to be important in each case.

| # | Tag | Owner | Version | Status |
|---|---|---|---|---|
| DI2.2 | **Remote Task Server** | B. Analyst | 0.1 | DRAFT |

| | |
|---|---|
| Gist: | A Message bus application receiving instruction messages (e.g. from "Kill" GUI), execute pre-defined operating system script to stop/kill customer quoting application. |
| Description: | Utilise new message bus architecture to create a Remote Task Server application deployed on same server as the customer quoting application:<br>1. Subscribe to Remote Task messages (published by "Kill" GUIs)<br>2. Execute operating system script (.bat file on Windows, .sh on Linux)<br>3. Capture script output and publish Remote Task Response message |
| Design Constraints: | DC3 **Customer Quotation System** "use ant script to shutdown"<br>DC4 **Message Bus Infrastructure** "relatively simple use case for new infrastructure" |
| Related | DI2.1 **"Kill" GUI** |
| Impacts [Functions]: | F2.4.1 **Corrective Action.Initiation.Manual** |
| Impacts [Performance]: | P3.2 **Correction.Customer Quotation Cancellation Time**<br>*Cuts out Support desk from loop; FX desk users can immediately initiate remote kill without waiting for people to get back to their desks, log onto production server consoles, locate script. Guess should take no more than few hundred milliseconds once Remote Task message published on bus before RTS executing kill script.<- A. Analyst 24 March 2014* |
| Impacts [Costs]: | R3.1.1 **Budget.Work.Team** |
| Risks: | R1. May under-estimate work needed in new core message bus libraries, because of the new technology stack. |

Over the duration of the Price Sentinel project, the team created over 20 separate significant Design Ideas, which were all documented in a similar one-page style as the example above. As with all the

other requirement types discussed here, new Ideas were added to the list after every stakeholder or technical project meeting during the initial analysis phase and beyond. Each was successfully refined as new Function, Performance, Constraint and Resource requirements were identified.

## End of Initial Analysis Phase

The initial requirements gathering phase took about a week, primarily constrained by the availability of key Stakeholders to attend meetings. As the business analyst, I spent a few days writing up the information using Planguage – style templates, drawing up a function map diagram, and chasing up unresolved Issues and other questions that had been suggested. The Requirements were gathered into a Project Initiation Document for convenience:

1. Executive Summary (one page summary of project rationale, top three objectives and functions, outline of project approach)
2. Introduction (included tables of Assumptions & Issues)
3. Stakeholders
4. Stakeholder Objectives
5. Functions
6. Constraints
7. Resources
8. Initial Design Ideas
9. Glossary

At end of the second week, we held a short round of Review meetings with the Stakeholder, in which we presented the work so far. The key aim was to review and validate that the project team's understanding of their requirements covered the entire scope. Had we collectively understood the requirements in sufficient detail to be able to begin design selections and subsequent implementation work in earnest? Although we did not refer to the Planguage concept of Exit Conditions[16] overtly, these sessions were effectively reviewing whether we were ready to exit the initial analysis phase.

It was clear that the separation of requirements into different types, and separating requirements from design made these sessions highly productive. Business users could see functions and objectives they intuitively understood, IT people could review the initial list of ideas.

[16] Page 23 (Gilb 2005)

We also sorted all of the Design Ideas identified so far into one of the following categories[17]:

1. Trashcan (violates a Constraint, everybody agrees this is going to way exceed budgeted resources)
2. Needs More Work (agree do not yet sufficiently understand benefits and / or impacts to make reasoned decision)
3. Likely (general consensus that design idea is a strong candidate for implementation).

We decided that we had enough "Likely" design idea that we could profitably commence the first series of implementation steps. Items in the "Needs More Work" category would be followed up over the next few weeks, and placed in the Trashcan or Likely lists as appropriate. We experimented with this "triage" pre-process to help the team focus on a manageable list of five to ten Design Ideas to consider at the start of each proceeding Evo Step.

## Evo Planning and Value Decision Tables

We adopted the Competitive Engineering Evo methodology[18] to govern how we ran the development cycles on the Price Sentinel project. The project was divided into a series of short duration Steps, of approximately 2 weeks each. The target was to deliver real changes to stakeholders at the end of each step, into the live production environment. Stakeholder feedback from each delivery informed the design decisions of the next Step. Similar to the more well-known Agile methodologies (e.g. Scrum, Kanban), decisions on which designs to implement are deferred until the start of each Step, rather than planned out in great detail at the start (Waterfall). The Evo methodology controls this critical design selection process by comparing the estimated impacts of each proposed design idea on the project's identified functions and quantified stakeholder objectives. The designs selected for implementation in each Step are then the ones that provide the biggest value improvement for the least cost.

[17] Pre-sorting Designs in this manner is not part of Planguage, but closer to common Agile story prioritization techniques. In this case, we found we needed to make a best, non-quantified, guess of the most valuable designs to take into more detailed analysis using Impact Estimation Tables.
[18] See http://gilb.com/dl77 for an overview of Evolutionary Project Management

We created a Function Matrix spreadsheet to act as a checklist of which Function Requirements had been addressed by each Design implemented in each Step. The values in each cell were a simple "Yes" if the function had been provided, or blank, if not (recall that Planguage Function Requirements are binary: either the system provides a function, or it does not):

rice Sentinel Function Traceability Matrix

| inction | Total | Designs Step 1 PSS.2.4 | PSS.2.5 | PSS.2.3 | |
|---|---|---|---|---|---|
| :1 Off market Protection | | | | | |
| .1 Detection | | | | | |
| .1 Monitored Data | | | | | |
| .1 Tradable Quotes | | | | | |
| .1 Specified FX trade Types | | | | | |
| 1 FX Spot | | | | | |
| 2 FX Outright | | | | | |
| 3 FX Swap | | | | | |
| 4 NDF | | | | | |
| .2 Specified Trading Channels | | | | | |
| .3 Indicative Quotes | | | | | |
| .2 Price Streams | | | | | |
| .3 Deal Requests | | | | | |
| .4 Market Data | | | | | |
| .1 Spot | | | | | |
| 1 EBS | | | | | |
| 2 Currenex | | | | | |
| 3 Reuters D3K | | | | | |
| .2 Swap Points | | | | | |
| 1 Reuters D3K | | | | | |
| 2 RET | | | | | |
| .3 Scrubbing | | | | | |
| .5 Interbank Orders | | | | | |
| .2 Identification | | | | | |
| .3 Manual | | | | | |
| .4 Automatic | | | | | |
| .5 Auditing | | | | | |
| .1 Off-Market Event | | | | | |
| .2 Corrective Action | | | | | |
| .1 Quote Control | | | | | |
| .0 | | | | | |
| 1 Prevent further | Yes | Yes | | | |
| 2 Replace with Indicative | Yes | | Yes | | |
| 3 Inform Trading Channel | Yes | Yes | | | |
| .2 Deal Request Rejection | | | | | |
| .3 Hedge Order Cancellation | Yes | | Yes | | |
| .4 Initiation | | | | | |
| .1 Manual | Yes | | | Yes | |
| .2 Automatic | | | | | |
| .5 Security | | | | | |
| .6 Reporting | | | | | |

**Figure 2 – Function Traceability Matrix**

The most important project tool in the Competitive Engineering armoury is the Value Decision Table (or "Impact Estimation table")[19]. It is a simple, compact but very powerful means of communicating information about the likely value improvements inherent in each Design Idea compared with their respective costs. The information is structured and presented to enable side-by-side comparisons to be made between competing design ideas, irrespective of the potentially widely-varying natures of each design. This is where all of the previous analysis work on quantified key stakeholder objectives and costs are ultimately condensed down and used to help make smarter decisions on how to spend our precious allocated resources.

We had already decided to focus the first Steps on improving the objectives related to dealing with an Off-Market Event once it had been Detected. This was largely because we had realised whilst

[19] See Chapter 9 of (Gilb 2005), and many available presentations on gilb.com on use of Impact Estimation Tables.

sorting through the list of initial Design Ideas that the Correction – related Ideas were very likely to be cheaper than the necessarily more complex solutions for automated detection.

Here's a simulated re-creation of part of the Impact Estimation table from the first Project Step:

| **Requirements:** | | | | | **Designs:** | | | | **Totals:** |
|---|---|---|---|---|---|---|---|---|---|
| | | Past | | Goal | *Step:* | PSS.Step 1 | | | |
| | | | | | *Design Idea:* | **DI2.4** | **DI2.5** | **DI2.6** | |
| #P3.2 | **Correction.Cancellation Time** | | | | | | | | |
| | [ECM, London, eFX Desk] | 600 secs | -> | 1 sec | | 300s | 0s | 10s | 890s |
| | | | | | | ±150s | | ±5s | ±155s |
| | | | | | | 50%±25% | 0% | 98%±1% | 148%±26% |
| | [Quote Control, London, eFX Desk] | 600 secs | -> | 1 sec | | 0s | 10s | 0s | 590s |
| | | | | | | | ±5s | | ±5s |
| | | | | | | | 98%±1% | | 98%±1% |
| **Sum Of Performance:** | | | | | | **50%±25%** | **98%±1%** | **98%±1%** | |
| #R3.1.1 | **Budget.Work.Team** | | | | | | | | |
| | [Developer, H1 2012] | 0wd | -> | 25wd | | 0.5wd | 3wd | 10wd | 13.5wd |
| | | | | | | ±0.5wd | ±2wd | ±5wd | ±7.5wd |
| | | | | | | 2%±2% | 12%±12% | 40%±20% | 54%±34% |
| | [Tester, H1 2012] | 0wd | -> | 15wd | | 0.5wd | 1wd | 3wd | 4.5wd |
| | | | | | | ±0.5wd | ±1wd | ±2wd | ±3.5wd |
| | | | | | | 3%±3% | 7%±7% | 20%±14% | 30%±24% |
| | [Project Manager, H1 2012] | 0wd | -> | 10wd | | 0.5wd | 0.5wd | 2wd | 3wd |
| | | | | | | ±0.5wd | ±0.5wd | ±1wd | ±2wd |
| | | | | | | 5%±5% | 5%±5% | 5%±5% | 15%±15% |
| **Sum Of Costs:** | | | | | | **10%±10%** | **24%±24%** | **65%±39%** | |
| **Performance To Cost Ratio:** | | | | | | **5.0** | **4.1** | **1.5** | |
| | | | | | | | | | |

Notes on Step 1 Impact Estimation Table:

1. The example only shows a couple of Goals for a single Performance Requirement; the real table had entries for all of the key objectives for the project. Each Design Idea must be estimated against all relevant objectives: primary and side effects.
2. The real table had many cell comments providing information on the Sources of each estimate
3. The right-hand most **Totals** column gives some indication on whether we have thought of enough Design Ideas yet to have a high probability of hitting the required goals. Because there is uncertainty in the estimates, a good rule of thumb is aim for at least 200% (ie. have enough Design Ideas to have an estimated 2 x likelihood of meeting each goal). The numbers are not intended to be precise to the nearest 1%: but a Total of, say, only 50%, should be

taken as a warning that we probably need to do some brainstorming for further Ideas. Conversely, a total of 400% indicates we probably have sufficient ideas already.

4. The Sum Of Performance row adds up the percentage improvements of all Objectives for each Design Idea. The actual numbers are not as important as the comparison between them. If we have an Idea with a Sum Of Performance value of 10% vs. one with a value of 300%, that is a pretty clear indication of their relative value across all key objectives (and not just the ones they were primarily intended to address). In the example above, it looks like DI2.5 & DI 2.6 are approximately of equal value to each other, DI2.4 slightly less.
5. The Sum Of Cost row adds up the percentage of our Budgets and resources used up by each Design Idea. In the above example, the cost differences between the three Design Ideas are clearly shown; although the actual percentage numbers have large uncertainities, it is reasonable to conclude that, for example, DI2.6 is significantly (by factor of 5 or 6) more expensive than DI2.4
6. The Performance To Cost Ratio row incorporates all the currently known information about each Idea, key stakeholder improvement objective and resource budget. It is a calculated as the Sum Of Performance (%) divided by the Sum Of Cost (%). All of the analysis and design work described in this case study up to now has contributed to finding these ratio values.

## Project Delivery Steps

**Step 1**

The team then had to make the first design decision of the project, using the information in the Function Matrix and Impact Estimation Table. The decision was pretty clear by now. Although both DI2.4 & DI2.5 both had a high value to cost ratio (between 4 to 5 x “bangs for each buck”), DI2.4 needed less than a working day from each of the team members to implement. Design Idea DI2.4 was creating a simple operating system script file onto the desktop of the Windows-based production server, which force – quitted the system component responsible for publishing customer quotes.

Successful Evo projects need to be empowered by senior management to deliver changes to stakeholders as early as possible. In the case of Price Sentinel, we had identified something we could do within a day or two. In fact, the script was written and development-tested by the end of the first day, deployed onto the production server at end the end of the second day, which was about as fast as it was possible to make any non-emergency change in the production environment at that time. After a total of two weeks analysis plus two days implementation, we had delivered the first improvement to critical stakeholder objectives. During end-of-step testing, we measured the actual improvement in cancellation time due to the new script was about 6 minutes, mainly because the support staff did not have to spend time consulting with support documentation to find the

relevant operating system commands to shut down the application. We had improved one of the critical objectives by 60%.

**Step 2**

We initiated Step 2 on day three of the implementation phase, after the deployment of the desktop script. We updated the Impact Estimation Table from Step 1, by changing the [ECM, London, eFX Desk] goal "Past" level from 600 seconds to 240 seconds, and the "Past" work-days spent by each team member to reflect the effort we had spent on Step 1. Additionally, we revised some of the improvement & cost estimates incorporating recent better estimates. Re-computing the value improvements and performance to cost ratios for the remaining two designs gave updated performance to cost ratios (the ratios did not change because of the small effort spent on Step 1):

| **Requirements:** | | | | | **Designs:** | | | | **Totals:** |
|---|---|---|---|---|---|---|---|---|---|
| | | Past | | Goal | *Step:* | PSS.Step 2 | | | |
| | | | | | *Design Idea:* | | **DI2.5** | **DI2.6** | |
| #P3.2 | **Correction.Cancellation Time** | | | | | | | | |
| | [ECM, London, eFX Desk] | **240 secs** | -> | 1 sec | | | 0s | 5s | 235s |
| | | | | | | | | ±5s | ±5s |
| | | | | | | | 0% | 98%±2% | 98%±2% |
| | [Quote Control, London, eFX Desk] | 600 secs | -> | 1 sec | | | 10s | 0s | 590s |
| | | | | | | | ±5s | | ±5s |
| | | | | | | | 98%±1% | | 98%±1% |
| **Sum Of Performance:** | | | | | | | **98%±1%** | **98%±2%** | |
| #R3.1.1 | **Budget.Work.Team** | | | | | | | | |
| | [Developer, H1 2012] | **0.5wd** | -> | 25wd | | | 3wd | 20wd | 23wd |
| | | | | | | | ±2wd | ±8wd | ±10wd |
| | | | | | | | 12%±12% | 81%±32% | 95%±44% |
| | [Tester, H1 2012] | **0.5wd** | -> | 15wd | | | 1wd | 5wd | 4.5wd |
| | | | | | | | ±1wd | ±2wd | ±3.5wd |
| | | | | | | | 7%±7% | 34%±14% | 30%±24% |
| | [Project Manager, H1 2012] | **1.0wd** | -> | 10wd | | | 0.5wd | 2wd | 3wd |
| | | | | | | | ±0.5wd | ±1wd | ±2wd |
| | | | | | | | 5%±5% | 5%±5% | 15%±15% |
| **Sum Of Costs:** | | | | | | | **24%±24%** | **120%±39%** | |
| **Performance To Cost Ratio:** | | | | | | | **4.1** | **0.8** | |
| | | | | | | | | | |

Design Idea DI2.5 was to implement a similar shutdown script for another system component, taking estimated 3 days of development effort to complete. DI2.6 was to implement a more complex new one-button GUI and Remote Task server application to allow business users direct

access to "Kill Switch", using the new SOA architecture. We could see that it was possible to implement DI2.5 within a few days, but DI2.6 needed a lot more work. So, we started work on the DI2.5 shutdown script straight away (which ended up taking just over one day to complete development). At the same time, we tasked another developer to start work on the low-level technical development work on the new SOA architecture needed to support the Remote Task function. Any development on the new architecture was risky and uncertain, so it was also strategically important we started learning how to use it as early as possible.

Step 2 was decomposed (another core Planguage concept) into three sub-steps, each of which was about 2 weeks each:

1. Step 2A:
    a. (frontroom): Implement second shutdown script
    b. (backroom) : Implemented and tested (using automated testing) core messaging layer for new "Remote Task" service. 1 week.
2. Step 2B:
    a. (backroom) : Implement Remote Task Server application,
3. Step 2C:
    a. (frontroom) : Deployed Remote Task Server with simple Windows batch script client frontend on Windows Server desktop. Deployed into production. 2 weeks

Here, we invoked another Planguage idea of decomposing larger tasks into Backroom (necessary but invisible tasks) / Frontroom (visible and measurable to stakeholders)[20]. We knew we had unavoidable technical development to complete on the new SOA architecture before we could implement the Remote Task function. So, we decided to implement the required technical changes in the current Step in the "backroom", and deliver that work into production, albeit unused. However, we were able to deliver the shutdown script DI2.5 by the end of step 2A.

Steps 2A, 2B and 2C were delivered on schedule. We realised we could replace the GUI with a simpler script file that could still be launched on an end-user's desktop, but would still send a message to the Remote Task Server application, thus saving about a week of additional effort. Although the end-user had to double-click on an icon on their desktop instead of clicking a nice red button, it made very little difference to the objective of reducing the time to complete the shutdown action. This is a small demonstration of the value of clearly identifying and focusing everyone on achieving the performance objectives: any viable solution that provides an improvement can be considered, even if we only come up with a slightly better idea after the Step has already started.

It then turned out that a production incident (unrelated to Off-Market Events, but affecting the same system components) on the day after the release of the Step 2C delivery into production required

[20] See Planguage Glossary Concept #342 Backroom and #343 Frontroom (Gilb 2005)

the use of the Remote kill application for the first time. The new system worked as specified, allowing the head of the eFX trading desk to initiate and complete shutdown in about ten seconds (it would have been quicker, but he wanted to check with us first before clicking it for the first time!). The improvement was immediately recognised and earned early good will to the project team. This shows a nice side-effect of continuously looking for opportunities to deliver *some* change to *any* stakeholder at the earliest opportunity. If and when things go wrong during a project, it is incredibly useful to be able to point at a stream of improvements already made. We had already started building up the trust of the stakeholders in the capability of the new architecture platform, and in the team's ability to use it to deliver important improvements.

**Step 3.**

At the end of Step 2, we had demonstrably achieved almost 100% of the relevant Action – related objectives and functions, but we had spent almost 100% of the originally allocated budget. A decision had to made how to tackle the other main Objectives related to speed and accuracy of Detection of Off-Market Events. We drew up another Impact Estimation Table (not shown here) with another set of Design Ideas, performance impacts and resource costs.

Using the same decision techniques as the earlier Steps, we planned and initiated further Project Steps to start building new components to provide automated detection capabilities. We spent about six weeks further development effort on writing new market data adapter software and a Price Sentinel monitoring application, all using the ever-growing new SOA – based core architecture. Although the original budget for the Price Sentinel project had been expended, we were able to continue work by focusing on the benefits of building out the new core platform, which had been started in Step 2. The good will and trust earned by the success of the early designs made it easier for the senior stakeholders to continue to invest.

After about 6 weeks, it was decided to halt further work on the Price Sentinel – related new components, and re-focus on requirements on a different project. Business priorities had changed in the intervening period; improving the ability to monitor the trading system became relatively less important than building additional critical capability to the trading platform itself. Stakeholders decided that they were capable enough to spot possible Off-market Events using existing methods, knowing that they could then take action very fast. The improvements from Steps 1 and 2 were considered good enough to justify stop spending further valuable development resources. Happily, very little of the development work had been wasted, because most of the technical work had gone into the re-usable core of the new platform. The technical aim of using the Price Sentinel project as a proof-of-concept of the new architecture had also therefore been achieved.

## Conclusion

The Price Sentinel project started from a requirement stated as a design. We uncovered key stakeholder objectives using the 5 Whys technique. Design-free requirements were clarified, detailed and reviewed as Planguage quantified objectives, business functions, constraints and resource budgets. Separation of requirements from design gave everyone the freedom to brainstorm a wide range of design ideas, each with different product qualities and costs. Initial cheap categorization of ideas into a backlog then proceeded with first Evo step by building an Impact Estimation table for likely initial candidates. The first step was implemented in production within 48 hours. The second step delivered further improvements in production over 4 – 6 weeks. The project was halted after another 6 weeks due to business re-prioritisation based on the improvements already realised.

This case study presented a single example of using the ideas of Competitive Engineering in a real-world project in a major investment bank during 2011. It demonstrates the utility of following some simple rules, techniques and principles for requirements analysis known collectively as Planguage. Project delivery is run as a highly iterative process known as "Evo", using short Steps to deliver regular incremental improvements. Designs are continuously selected during the project, based on the best-possible quantified estimates and measurements of improvement amounts and resource usage. An Evo project can stop at any point once stakeholders agree sufficient improvements have been delivered against the desired objectives as assessed at that time.

## Works Cited

Gilb, Tom. *Competitive Engineering.* 1st Edition. London: Elsevier Butterworth-Heinemann, 2005.