Requirements Engineering Specialist Group

Requirements Quarterly

 http://www.resg.org.uk RQ 53 (January 2010)

Contents

RE-soundings

From the Editor

A Happy New Year to all our members, and welcome to the first RQ of 2010! I hope you find this edition to be entertaining, informative and occasionally provoking. One of the features of our RESG community is the enthusiasm to share experiences, good and bad, and to test what appear to be accepted principles. The reflection on our recent *Working in RE* event illustrates this. "*Theories of RE…may be very neat but practice is rather different*" does not seem to be a good way to demonstrate the benefits of RE, but it does show that there needs to be a healthy interaction between the research and practitioner communities.

My family is renovating a pile of rubble that I hope will, one day, resemble the C17th farmhouse it once was – but with electricity and central heating! I have tried plastering, with varying success, and realised I had a couple of options – do it myself and risk divorce, learn how to do it properly, or employ an expert. Fortunately I chose the latter option, and like most experts Darren the Plasterer showed a complete disregard to all instructions and guidance but still produced flat, smooth walls and ceilings.

Of course, at some stage Darren had learnt how to do things properly 'by the book', but over time developed his own short-cuts and techniques. That is where practice and theory probably differ – the benefit of experience through being a requirements engineer. But there is still a need for research because the world changes – the techniques that put the lime and horse-hair plaster on my walls are different to those used to-day, even if the end result is similar.

We do need theory *and* practice to ensure that the benefits of requirements engineering continue to deliver value as the problem context becomes increasingly complicated and complex. The trick is to keep both aligned, and I would suggest that RESG has a role to play in this through the newsletter and our improved web-site.

So share your research and real-world experiences with us and perhaps we can all benefits. The alternative is to do it yourself from scratch, and if your plastering is anything like mine you may regret it!

Enjoy.

Simon Hutton, Headmark Analysis

Chairman's Message

One of the things that consultants and trainers in engineering can't help noticing is that stakeholders have extremely different points of view on pretty much everything.

Expectations for a requirements tool, for instance, depend on where you are sitting. The busy manager sees a requirements problem, orders a tool, and gets it installed.

It doesn't solve the problem.

Not too long afterwards, the manager orders some requirements training, to get people up to speed with the tool. Now, surely, they'll all knuckle down and start using the thing.

It doesn't solve the second problem, let alone the first one.

The people on the ground aren't maintaining the requirements because they seem irrelevant. The specifications are old; the system has been procured. Now all the attention is on Change Requests or Software Problem Reports or Engineering Change Notices or whatever (there is no end to the acronym soup on this one). Unfortunately, this means that to understand the system you have to

* read the dusty old spec dating back 10 years

* understand 10 years' worth of approved change requests

* build a mental model tracing the changes to the old requirements.

Ah, what do we think of traceability. To misquote Gandhi, it would be a good thing.

What the busy manager needed to do - obviously, you may think, if you are a certain kind of stakeholder - was to get people to agree a process that includes enough traceability to make it worth their while to keep the requirements freshly dusted. Then the requirements traceability tool, and its training course, would suddenly take centre stage.

But as long as requirements have a "fire and forget" feel to them - things to write once for a procurement - it is hardly likely they will be very popular inside organizations. The contractual list of "the system shalls" may be great for tying down a contractor, but it is not the right instrument for recording design rationale, or of maintaining it through organizational changes over many years. A goal model, a set of operational scenarios, a rich picture context model, an automated project dictionary, a carefully-argued design rationale linked directly to the requirements ... these things, that are close to the requirements but are not actually the contractual list, would do the job rather better.

Ian Alexander, Scenario Plus, January 2010

RE-treats

For further details of all RESG events, see www.resg.org.uk

RESG – Goals Day

Wednesday 24th March 2010, University of Westminster

A goal is something that a stakeholder wants to achieve, whether that is a function or a desired quality. Goals are basic descriptions of stakeholders' intentions, and imply their key requirements. But goals are permitted to be neither fully achievable nor measurable, unlike requirements which must be both. Goals may conflict, whereas requirements must not. Therefore, goals must be translated into realistic, measurable requirements.

Goals are thus fundamental to requirements engineering, but they are handled very unevenly in industry. This one-day event combines a tutorial on the essentials of goal modelling with a seminar of talks on

how to use goals from leading experts in industry and research. The day concludes with a panel discussion.

Programme

9:00 Goals Tutorial, Part 1 Ian Alexander (Scenario Plus) & Ljerka Beus-Dukic (University of Westminster)

Overview of approaches to goal modelling

Theories including i* and KAOS

Team exercise on goals

10:30 Coffee

11:00 Goals Tutorial, Part 2

Simple practical approaches

Integrating goal modelling with other requirements techniques

Team exercise on obstacles & threats

12:30 Lunch

13:30 i* at City University James Lockerbie (City University)

14:00 Relating Goals and Architecture During System Evolution Emmanuel Letier (UCL)

14:30 Goal Structured Notation (GSN) and Satisfaction Arguments Phil Wilkinson (Rolls-Royce)

15:00 Tea

15:30 Using the Design Rationale Editor (DRed) Gareth Armstrong (Rolls-Royce)

16:00 Goals for Adaptive Systems Pete Sawyer (Lancaster University)

16:30 Questions to the Panel

17:00 Close

A full description of the event, programme, speakers' bios and the venue can be found at:

http://www.resg.org.uk/event35.html

Registration
Attendance at this one-day event will cost £75 (inc. VAT), and numbers will be limited, so do book early. To register, visit the BCS event registration page: www.bcs.org/events/registration

Please contact Rachel Browning if you have any queries concerning registration:

Rachel Browning, MBCS Specialist Groups BCS First Floor, Block D, North Star House, North Star Avenue, Swindon SN2 1FA

Tel +44(0)1793417416 / fax +44(0)1793417444,

e-mail rachel.browning@hq.bcs.org.uk

SAC 2010

ACM Symposium on Applied Computing

22-26 March 2010, Sierre, Switzerland

For the past twenty-four years the ACM Symposium on Applied Computing (SAC) has been a primary gathering forum for applied computer scientists, computer engineers, software engineers, and application developers from around the world.

The Third Edition of the Requirements Engineering Track (RE-Track'10) is part of the SAC 2010, sponsored by the ACM Special Interest Group on Applied Computing (SIGAPP), and is hosted by the University of Applied Sciences, Western Switzerland (HES-SO) and Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland.

The objective of the Requirements Engineering track is to explore different advances in requirement engineering in a general way, its relation with different areas, reducing the gap between software engineering solutions and the way one specific domain of knowledge was seen up to given point.

Further details may be found at www.acm.org/conferences/sac/sac2010, or contact the track organizer, Maria Lencastre, at maria@dsc.upe.br.

RESG – Postgraduate Workshop

Spring 2010, Imperial College London

Further details will be available prior to the event through RQ and on our web site www.resg.ork.uk. For further information of to express an interest in supporting contact the organisers:

Dalal Alrajeh (dalal.alrajeh@imperial.ac.uk) or
Ben Jennings (b.jennings@cs.ucl.ac.uk).

RESG – Social Network Tools for RE

April 2010, Open University, Milton Keynes

Further details will be available prior to the event, and will be provided through RQ and on our web site www.resg.ork.uk.

RESG – Industry Event

2.00pm, 14th April 2010

Bournemouth University

Further details will be available prior to the event through RQ and on our web site www.resg.ork.uk. For further information of to express an interest in supporting contact the organiser Cornelius Ncube (cncube@bournemouth.ac.uk).

CAiSE 2010

22nd International Conference on Advanced Information Systems Engineering

June 7– 11, 2010 Hammamet, Tunisia

This year's special theme is "Evolving information systems".

Modern information systems are the result of the interconnection of systems of many organizations, are running in variable contexts, and require both a lightweight approach to interoperability and the capability to actively react to changing requirements and failures. In addition, users of information systems are becoming more and more mobile and ubiquitous, requiring the system to adapt to their varying usage contexts and goals.

The evolution of an information system should be a continuous process rather than a single step, and it should be inherently supported by the system itself and the design of the information system should consider evolution as an inherent property of the system.

BUSITAL'10

5th International Workshop on BUSiness/IT Alignment and Interoperability

June 7th 2010, in conjunction with the CAISE'10 conference, at Hammamet, Tunisia

Organizations are today becoming more and more dependent on their information systems and IT-based support systems to realize their business strategies, building value networks with partners, and managing their resources effectively. But ensuring that their IT investments are well aligned is not easy. Such alignment is a critical early stage activity to understand how information systems contribute to business strategy and to set directions for the development and maintenance processes that follow. Its requires a good understanding and solving of issues at all levels ranging from information technology issues, through organizational issues up to business and strategic issues, and the ability to rapidly, smoothly and consistently adapt all these.

A number of frameworks and methods have been designed to help managers in aligning business and IT. Recently, novel methods and techniques based on conceptual and enterprise modelling have been proposed to support mutual alignment between business needs and IT solutions.

The overall objective of the workshop is to bring together a larger community (both Information Systems and Information Management) contributing to exploring the benefits, challenges and solutions of business and IT alignment.

Important dates:

* Papers submission: March 1st, 2010
* Notification of acceptance: April 1st, 2010
* Camera-ready copies: April 9th, 2010
* Workshop: June 7th, 2010

More information can be found at: http://www.info.fundp.ac.be/BUSITAL2010/

R e f s Q ' 10

30 June - 2 July, 2010, Essen, Germany.

The 16th International Working Conference on Requirements Engineering: Foundation for Software Quality.

Since 1994, when the first RefsQ took place, Requirements Engineering (RE) continued to be a dominant factor influencing the quality of software, systems and services. The RefsQ working conference series has now established itself as one of the leading international forums to discuss RE in its (many) relations to quality. RefsQ'09 seeks reports of novel ideas and techniques that enhance the quality of RE's products and processes, as well as reflections on current research and industrial RE practices. In the past, REFSQ has been organised in conjunction with other conferences, mainly the International Conference for Advanced Information Systems Engineering (CAiSE). For the first time, REFSQ will in 2010 be a stand-alone conference.

Further details at www.refsq.org.

RE'10 - Requirements Engineering in a Multi-faceted World

27 Sep – 1 Oct 2010, Sydney, Australia

Software systems in today's multi-faceted world are as diverse as the people who use them. While some are built according to rigorous government regulations, others must be delivered quickly to meet time-to-market deadlines or must be responsive to changing business needs. From a requirements engineering perspective there is certainly no 'one-size-fits-all' solution.

Diversity is also prevalent across software development teams where end-users, developers, and other stakeholders often come from entirely different cultural, linguistic, religious, and educational backgrounds. Only by understanding and embracing this diversity can we communicate effectively across these boundaries and collaboratively build software systems that meet stakeholders' needs, wants, and desires. As the Requirements Engineering research and practice community, we therefore need to develop innovative and useful techniques for eliciting,

analysing, specifying, and managing requirements effectively across diverse project teams for a broad spectrum of projects.

RE'10 invites papers that address all facets of the requirements engineering process ranging from formal to informal, from large to small, and across people-centric, business centric and system-centric viewpoints.

We specifically encourage submissions which address multi-cultural requirements engineering practices that facilitate working with diverse stakeholders and project teams.

Further details at www.re10.org.

RE-Course

Mastering the Requirements Process

23-25 February 2010 and 13-15 September 2010, London. Presented by Suzanne Robertson, Atlantic Systems Guild

This 3 day seminar & workshop presents a process for eliciting requirements, testing them for correctness and recording them clearly, comprehensibly and unambiguously. A 10% discount is offered to current RESG Members.

Details can be found at www.irmuk.co.uk/1/ or e-mail customerservice@irmuk.co.uk.

Mastering Business Analysis

26-27 April 2010, London. Presented by James Robertson and James Archer

This two day seminar and workshop in business analysis gives you the skills and tools to discover your client's real business, and to determine and demonstrate the best ways of improving it. A 10% discount is offered to current RESG Members.

Details can be found at www.irmuk.co.uk/90/ or e-mail customerservice@irmuk.co.uk.

RE-member

RESG.org.uk Update

You may have noticed that the RESG Website has recently been overhauled by our new Web Master, Camilo Fitzgerald. The RESG website aims to provide interested practitioners, researchers and students with a useful source of RE related information. The main topics of interest are:

- The latest on **events** organised by the group.
- Many speakers' slides can be found in the **archive** of past events.
- Archive of our regular **newsletter** (Requirements Quarterly).
- Links to pertinent RE **resources** on the web.
- An archive of **articles** on the topic of Requirements engineering.

The web site also allows comments against articles, the Newsletter, and RESG events, giving you a greater opportunity to let the Committee and Event Organisers know what you like and don't like, and to contribute to our RE Body of Knowledge. Plans for the future include using the web site rather than RQ to publish articles and notices, improving the frequency and topicality of news and features. RQ will continue, but possibly as a quarterly round-up of recent web content.

Is this what you want? Are there any areas that need improvement? Do let us know through the website – www.resg.org.uk!

Simon Hutton, Editor

And the Winner is…

RQ52 included a review of Tim Zaal's book "*Integrated Design and Engineering*". I offered the review copy as a prize for the 'best' article to appear in the edition – all entirely subjective, editor decision final. I am sure you will agree that **Olly Gotel** is the worthy recipient for her entertaining and provoking article on building a kayak as an object lesson in Requirements Engineering.

In a blatant attempt to encourage your active participation in our newsletter I am repeating the exercise with the book reviewed in this edition. A copy of Tom Gilb's "*Competitive Engineering*" will be awarded to the author of the most informative and/or entertaining article published in RQ54. As with this edition there are no clear selection criteria – the decision is purely subjective and very un-democratic, but any constructive comments, complaints or alternatives can be posted on our web-site. Again, you are encouraged to participate!

Simon Hutton, Editor

Promoting RESG

Our new Logo was introduced in the last Newsletter, and now appears on the website and other promotional material. A brand new screen with the new logo had its first airing at our recent RESG Event on Working in RE. Details of the event can be found later in this Newsletter.

RESG Secretary James Lockerbie at the first airing of the new logo screen

Volere and LinkedIn

Discussions in the new Volere Requirements Linked In group have already covered a wide range of requirements-related subjects: grammar, entropy, creativity, testing and many others. You can join the group and talk to other Volere users at:

http://www.linkedin.com/e/vgh/2491512/

SQUARE

The following announcement about a tool to support SQUARE (Security Quality Requirements Engineering) recently appeared on an e-mail circulation, and is copied here to ensure our members are aware of its availability. Note that inclusion does not endorse or recommend the tool or process, but I would welcome any comments on the tool or about the underlying process for future editions of RQ – a good review would be of immense benefit to our community.

Carnegie Mellon University staff within CERT, part of the Software Engineering Institute, and CyLab are pleased to announce that a robust tool has been developed by a team of Carnegie Mellon Master of Software Engineering (MSE) students to support the nine-step Security Quality Requirements Engineering (SQUARE) process. The SQUARE project is identifying and assessing processes and techniques to improve requirements identification, analysis, specification, and management. The project is also focusing on management issues associated with the development of good security requirements.

The tool is available for free and can be used experimentally at:

https://squaretool.cylab.cmu.edu/Square

You will need a user ID and password to use the tool; contact Nancy Mead at nrm [at] sei.cmu.edu to request log-in information. In order to use the tool on one of your projects, it will need to be installed at your location. It is available for download from the CERT website.

The tool has been designed for use by stakeholders, lead requirements engineers, requirements engineers, and administrative roles. Educational instructions are embedded in the tool to give a brief overview of each step.

The tool supports transfer of security requirements to and from Requisite Pro.

SQUARE has been field tested in a series of client case studies. One client reported that "Our company operates in a lean, fast-paced, ever-changing environment and I had some reservations as to how much time we could spend in accommodating the project goals. I was impressed with how well we coordinated efforts in setting meeting dates, adhering to the schedule, and sharing information with minimal inconvenience to either side. Our company provided them [the requirements engineering team] with an opportunity to assess a many-faceted product and they responded graciously by sharing the different techniques they used to analyze the security aspects of our application. Their results gave us insight that has since influenced our application development and configuration. It was a pleasure working with the three separate groups and their sponsor over the two-year period."

The baseline SQUARE process is available on the CERT website. These are SQUARE's nine steps:

1. Agree on definitions.
2. Identify assets and security goals.
3. Develop artefacts to support security requirements definition.
4. Assess risks.
5. Select elicitation technique(s).
6. Elicit security requirements.
7. Categorize requirements.
8. Prioritize requirements.
9. Inspect requirements.

Because many operational systems problems are traceable to requirements problems, the SQUARE project team hopes to enable the development of systems that are more secure and survivable by successfully using requirements engineering methods. In addition, the SQUARE project team hopes that this focus on security requirements will result in more predictable development activities and processes, as well as systems whose costs and schedules are more predictable.

For more information on SQUARE, visit http://www.cert.org/sse/square.html or contact Nancy Mead at nrm [at] sei.cmu.edu.

Simon Hutton, Editor

RE-writings

Building Myself a Kayak: Some Lessons for Requirements and Software Engineering - Part I

Olly Gotel

During the summer of 2009, I decided to build myself a skin on frame kayak[1]. Being a complete kayaking fanatic, I wanted to learn more about the history of the kayak and to understand how design decisions impact a kayak's behaviour on the water. What better way to do this than to go through the process of building a traditional Inuit kayak for myself? Having absolutely no practical skills, it was clear that I was going to have to learn to do this in a workshop setting under the watchful eye of a master kayak builder.

With my master kayak builder located, along with a very appealing workshop location (a waterfront boathouse on the island of Vinalhaven in Maine), I received a list of required tools. Given that I was clueless about woodworking tools, this was obviously going to be an interesting shopping experience for me. What is a Japanese pull saw exactly? What is a spokeshave used for? Does it matter if I get a plane that is not low angle? I was probably the first person to request photos of all tools. After having borrowed everything possible from friends, and having raided almost every hardware store in New York City, I finally packed up my bag of tools and headed off to coastal Maine.

In the weeks leading up to the workshop, I had been thinking long and hard about the kind of kayak I wanted to build, only to realise that I did not really know. I knew that I valued fit, comfort and aesthetics, so much so that I took these things for granted. I also simply assumed that my kayak would float and be water tight, so I gave that no thought at all. However, I wanted my kayak to be fast and to track well (so it had to be long and narrow), to be an excellent roller (so it had to be low volume and sit low in the water), and to be able to handle rough water, waves and wind (so it had to be short enough to turn easily and have some rocker on its waterline for manoeuvring). These three things were, of course, all in conflict. I wanted to build myself the multi-purpose kayak that excelled at everything … and, as I jumped on an airplane to go north for two weeks, I thought I could actually do it!

Day one of the workshop was my wake-up call. Along with four fellow kayak builders, we discussed what we each wanted to build with our master kayak builder. Did I really want to build a compromise kayak, a kayak that would never be good at any one thing? No, but I could not decide what to prioritise. I was asked to think about what I actually wanted to do with my kayak above all else and most of the time. It was at this point that I realised that I was about to embark on a requirements engineering journey. I therefore decided that I should see if I could learn a few lessons for the day job as I sawed and drilled my way through the next couple of weeks.

To cut a long story short, I decided that I would build myself a rolling kayak. Rather than take you step-by-step through my internal requirements negotiation, and kayak design and building process, I have compiled a list of my top twenty observations from my efforts, and I draw some simple lessons for requirements and software engineering from them.

[1] 'Qajaq' (pronounced 'kayak') is the traditional Inuit term used to refer to a Greenlandic skin on frame kayak. For ease of reading, I shall use 'kayak' throughout this article.

1. *Rip a Story Stick to serve as a baseline*

On day one of the workshop, we each ripped ourselves a 'Story Stick'. A Story Stick is a long piece of off-cut wood that acts as a reference point throughout the entire kayak building process (as shown in Figure 1). It is used to record all the key measurements of an individual in one physical place, such as their wingspan (i.e., the tip of one middle finger to the other with outstretched arms) and their cubit (i.e., the tip of the elbow to the tip of the middle finger), measures that translate to important dimensions of the kayak so that it is custom built to fit. You start with the Story Stick measures that you design to and then, so long as you comply with these key dimensions, you can make ongoing design decisions every step of the way. In fact, it is somewhat possible to shape the characteristics and behaviour of the kayak as far as 90% into the woodworking portion of the build, the critical point of no return being the placement of the chines[2].

Figure 1. *The Story Stick for a kayak is always close to hand (it is the stick on the floor close to the hand!)*

We have been struggling with the process of requirements discovery ever since the dawn of software engineering. Our approaches range from attempting to specify all the requirements upfront (i.e., Waterfall) through to incrementally adding requirements stories as they arise (i.e., Agile). In the kayak-building world, we take a middle way. We work towards an overarching goal (e.g., a functional rolling kayak) and identify key measurements to then work within as we decompose and realise this goal. We neither make all the decisions upfront nor add features arbitrarily as and when they arise in our minds. We proceed along a truly incremental path that is governed by a few key constraints. In requirements and software engineering, we still need to find our middle way.

2. *Agree Terminology*

We used traditional Inuit and maritime terms to refer to all aspects of the kayak, so we knew exactly which part of a kayak we were referring to at any one time. When there are multiple deck beams and ribs in a wooden frame, it is important to have a scheme to refer to each beam and rib uniquely and precisely. For instance, the first deck beam in front of the cockpit is called the Masik and the first deck beam behind the cockpit is called the Isserfik. The traditional Inuit kayak terms are shown in Figure 2. The same rigour was applied to the tools and techniques we used. For example, if we were instructed to chamfer the wood, we knew we were to put a 45-degree angle on to the edge to take out a square corner. We were hence able to understand what parts of a kayak were being discussed and what techniques were required in any group instruction or communication. We were therefore also able to work on each other's kayaks interchangeably and as needed.

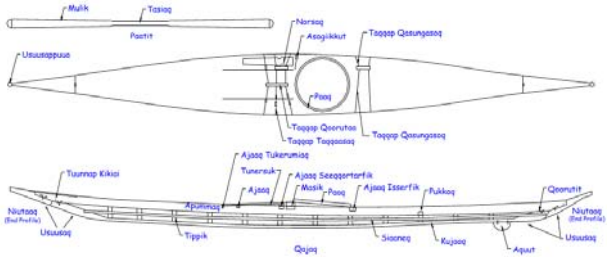

Figure 2. *The traditional language of the Inuit kayak*

(Please visit the Image Map of Shawn Baker and Craig Bumgarner to hear the terms spoken and to view their English translations: http://www.qajaqusa.org/Movies/audio_glossary_map.html)

In requirements and software engineering circles, we still debate the meaning of fundamental terms. Does requirements management refer simply to the management of requirements once specified or to all the activities of requirements engineering (i.e., elicitation, analysis, negotiation, specification, management, etc.)? When we ask someone to validate a requirement, do they verify instead? Are we able to use agreed terms to refer to the internal contents of a requirements specification or to the component parts of a software system? Use of agreed terminology facilitates the conduct of any complex task that involves more than one person. In requirements and software engineering, we need to go back to basics and agree on our terms.

[2] The reader is directed to a number of online glossaries at the end of Part II of this article. These define all the terms that are used in this article, more precisely. The chines are two long pieces of wood that define the edge of the kayak, from the side of the kayak (the gunwale) to the bottom (the keel), thereby providing its hull shape.

3. ***Make marks consistently***

The manner in which a pencil is held affects the angle and width at which a line is drawn. This can have quite dramatic consequences if a sawing task is soon to follow. I learned that there is a 'proper' way to sharpen a pencil to get a crisp line. I also learned that there is a 'right' way to hold a pencil against wood to draw this line in a repeatable way. There is a need to be consistent in how marks are made in woodworking for reliable results. The marks remaining from newly cut wood are shown in Figure 3, along with the conventions for deck beam and rib positioning on the gunwales. Where multiple lines or thick lines are visible on wood, and where these cannot be erased completely, standard conventions are then used for indicating where exactly to cut.

Figure 3*. Clear markings for fixing and cutting*

How do we indicate to others where we want task attention to be focused when we engineer requirements and software? It is somewhat tricky to mark-up the artifact of interest itself, in the hope that the required action will be noticed and undertaken as anticipated. Instead, we maintain associated project plans and create tickets for work, and we attempt to describe what needs to be done and where within these. The addition of ever more fragmented written artifacts, encodings of our intentions, leads to growing coordination and communication issues across teams and over time. This then sets up a challenge for subsequent traceability. In requirements and software engineering, we need a better way to juxtapose process information with our product for communication and recording purposes.

4. ***Proceed from an architectural backbone***

With a kayak, all the woodwork required to build the frame is completed prior to the skinning[3]. The first woodworking task was to join the gunwales, the two long pieces of wood that form the port and starboard sides of the kayak, to mark the midpoint of both, then to pull them apart to define the desired shape of the overall kayak. This shape was retained temporarily using a number of windlasses, as shown in Figure 4. There are three fundamental shapes for the kayak: neutral, where the maximum breadth of the kayak is at the midpoint, making the kayak symmetrical; fish-form, where the widest point is in front of the midpoint; and Swede-form, where the widest point is behind the midpoint. This decision determines the distribution of the kayak's volume. Although the eventual completed frame may resemble the bones of a fish, the backbone of the kayak is not the keel at the bottom of the kayak, but it is these very important gunwales. These provide the kayak with strength, and a solid basis to build upon and attach further wood to. Once these are in place, work can proceed.

Figure 4. *Shaping the gunwales using windlasses to hold their position*

In software engineering, requirements could be said to be the backbone of the system, since the end result is likely to be unsatisfactory if these are found wanting. However, we rarely elucidate and articulate the core requirements upon which all the others depend, let alone experiment with options for the shape they could take before proceeding with all those that follow. As per the gunwales, the core structure may be framed by non-functional properties, such as the overall shape and strength required of the system. It is all too easy to build on shifting sand, in our attempts to consider either every possible requirement upfront, or to accommodate any requirement that arises later. In requirements and software engineering, we need to differentiate those requirements upon which others depend. We need to do this far more concertedly, early on in the process, and create an architecture that allows other requirements to emerge later.

5. ***Envision prior to any commitment***

Progress appeared to be extremely rapid at the onset of the workshop, as pieces of wood were assembled into something that resembled a kayak in one day. There was then a visible sign of progress with every subsequent day, although less pronounced in the second week. This ability to see a kayak materialising

[3] Note that fabric is mostly used nowadays to skin a kayak, not seal skin! This is usually a heavy-grade nylon.

kept people motivated and positive. The use of clamps made it possible to attach pieces of wood to each other to experiment with positioning and shaping, prior to any permanent change or attachment. This enabled a three-dimensional view of the envisioned kayak at every step of the way, visibility that was indispensible for fixing deck beams and ribs to the gunwales, positioning the keel and chines, and for finalising the profile of the hull, bow and stern before commitment. The ability to experiment with the keel position is shown in Figure 5.

Figure 5. *Working out the positioning of the keel before drilling*

It is useful to be able to see the impact of decisions before they take hold. We attempt to do this in software and requirements engineering with rapid prototyping and user interface design mock-ups, but we rarely see beyond a two-dimensional surface or gain a view of the bigger picture. In particular areas, like change impact analysis, our ability to pre-visualise the impact of change prior to commitment, both locally and systemically, would be invaluable from a cost perspective. However, we perceive many of our actions to be reversible, so we sometimes forget the cost that rework incurs. In requirements and software engineering, we need ways to visualise the intentions and likely outcomes of our abstract intangible work.

6. *Measure twice, cut once… revisited*

The Carpenter's Maxim is well known and quoted in software engineering circles. As suggested by Figure 6, wood once cut stays cut. When it comes to kayak building, not only is this maxim imperative, it actually goes much further than this. We did measure more than once before cutting any wood. More important, we used different strategies and techniques to achieve any one measure. The trickiest task for me was placing the chines on the kayak because, when I took two different approaches to measurement, I kept coming up with two different results for their placement. It was the only way to catch that I was doing something wrong.

Figure 6. *Preparing to make that cut*

It is not an exaggeration to say that requirements and software engineers rarely think about measurement. When they do, they cling to one of a few well-known metrics and use those as a basis for decisions. In requirements and software engineering, we really ought to take a multi-pronged approach to how we obtain our measurements, particularly if we are to gain confidence in our measures and advance as a true engineering discipline.

7. *Stabilise to localise change and care for what remains*

Whether sawing, drilling or planing, we always ensured that the kayak was stable so that we could work on a specific area without causing a negative impact on the rest of the kayak. Throughout the entire building process, it was important to ensure that the wooden frame rested on two saw horses (or more) as we worked on specific areas, as shown in Figure 7. This was to maintain the frame's overall integrity and to prevent sagging as pressure was applied to areas. When sawing, we learned to hold the piece of wood that was to remain, rather than the piece of wood that was to be disposed of. When fixing, we would clamp the join for a number of hours to hold the wood in position until the glue had set.

Figure 7. *Stabilising the frame to work on a stem piece*

We do a reasonable job in requirements and software engineering when it comes to change management, particularly when working on versions of documents or code with a team of people. However, given that it can be quite tricky to understand all the hidden or emergent interrelations in an abstract system of this nature, assuring stabilisation of the whole as changes are made and take effect is not a trivial matter. The perceived ability to rollback any changes to a previous state, as mentioned earlier, is a belief that sometimes leads to a lack of initial care and attention with initial modularisation. In requirements and software engineering, we need to focus on this proactively when structuring requirements and the designs that flow from them.

8. *Work within acceptable tolerances*

Small mistakes propagate in kayak building and the impact accumulates. For example, if the ribs are positioned slightly off of the original marks when drilled and dowelled, you may find that you can no longer lash the ribs to the gunwales as securely as desired. To minimise the impact of errors, you need to decide what will be acceptable tolerances early on. Whenever we measured anything (see Figure 8), everything was acceptable to $1/16^{th}$ of an inch and we learned to work within this margin of error. The consequence of every small deviation becomes even more critical, however, as the kayak nears completion. When it was time to attach the final stem pieces to the bow and the stern of my kayak, I was too terrified to do the drilling, as a slight mishap in the drill angle would have meant redoing a lot of work. When undertaking critical tasks nearing completion of the build, the master kayak builder played a critical handholding role for the under-confident.

Figure 8. *Checking those tolerances again*

Do we make sufficient allowance for inevitable human error when we engineer software? We have approaches to help us detect, address and tolerate errors. We also have strategies to prevent errors in the first place, if we do a good job at requirements engineering. However, do we pay sufficient attention to all the small innocuous deviations that can eventually add up, or do we focus mostly on the larger and more obvious failure modes? Do we know what our acceptable tolerances are for everything we do and work within them? In requirements and software engineering, we should begin to understand the acceptable tolerances for different tasks and try to work within them.

9. *Avoid single points of failure and unnecessary rework*

The reliability of a kayak relies upon engineering multiple back-ups in the final product. For example, the integrity of the wooden frame is secured in three ways: using wooden dowels and glue for a primary attachment of wood to wood; lashing wood to wood for a secondary attachment; and tensioning the skin around the final frame for a tertiary. The first two levels of this system are shown in Figure 9. Kayak building also requires pre-planning if you are to avoid having to do work over. For instance, you need to plan for the attachment of float bags and put the deck lines in place before closing up the skin, else you have a nasty cutting and additional sewing job to do later.

Figure 9. *Dowels and lashings to secure each deck beam*

When engineering software, it is common to have to undertake much rework; refactoring is an acceptable term these days. With a physical structure such as a kayak, rework is incredibly difficult, frustrating and costly in time. As such, it demands anticipation and thinking ahead from the kayak builder, continuously reflecting on what is required in the eventual end product and understanding what needs to be put in place in the present to enable this to happen with ease. It does not mean doing all the requirements upfront, but it does mean understanding the overall concept early on and not flying blind into the build. Pre-emption is a precursor to gaining resiliency in most forms of engineering. In requirements and software engineering, we need to start to share more information on rework efforts and failures across projects,

especially if we are to avoid learning the same painful lessons over and over.

10. *Employ models and create informal sketches where useful*

At the start of the workshop, we made use of a reusable wooden rig to simulate sitting in a kayak, in order to get a feel for the required dimensions of the cockpit (e.g., length and width) and to assist with the positioning of critical deck beams. This would ensure that we would be able to slide in and out of our kayaks once completed without removing our kneecaps. When it came to forming the cockpit itself, we selected a prior cockpit coaming template to refine to our own dimensions. These would be used to bend wood to form our own individual cockpits, as shown in Figure 10. The role of a model or template is clear in kayak design and, once used, it is set aside. Throughout the entire kayak building process, impromptu sketches on scrap pieces of wood were also used for the communication of concepts and techniques, and to support decision-making tasks. For example, a two-dimensional matrix was sketched out on wood (resembling a 'Go' board) to help seat the ribs evenly, while the performance characteristics of differing hull shapes were sketched as needed to help with difficult chine positioning decisions.

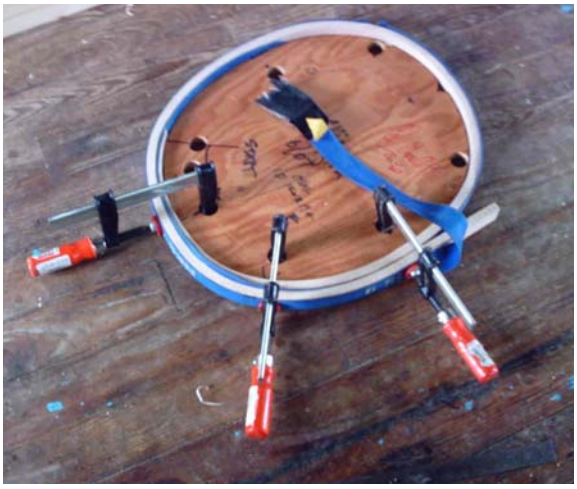

Figure 10. *Bending wood around a cockpit template to form a coaming*

Requirements and software engineering is all about modelling, and we reuse prior patterns in our design and coding activities with some regularity. However, we tend to call anything a model, so the line between what is the model and what is the final product can get quite blurred. As a consequence, the role of the model may mutate beyond its intended role and its planned useful life may be extended unwisely. Our use of informal sketching in requirements engineering is also still quite limited, even though sketching makes for a natural way to express tentative ideas and concepts early on amongst non-specialist stakeholders. In requirements and software engineering, we need to clean up our modelling practices and also look at how we can exploit much simpler means of informal communication.

11. *Understand the role and value of the expert*

The entire kayak building process would have been a complete disaster without the oversight of an expert eye. In our case, these were the four expert eyes of Figure 11. Our master kayak builder was able to see things that the rest of us overlooked or simply missed, moving seamlessly from the most intricate detail to the overall line of the kayak, across all five kayaks in parallel. The expert was also able to troubleshoot and improvise fixes to all our disasters, from poorly drilled holes, dowels popping out of wood, severed lashings, to split ribs. Continuous quality control and endless resourcefulness was achieved thanks to years of accumulated knowledge. The critical role of the expert even came into play long before the workshop began, in selecting the raw materials. Shamefully, I was unable to discern oak from cedar at the start of the workshop, let alone differentiate viable rib stock for bending from that with bad grain. I am proud to say that I am far savvier with wood now! With an expert on hand, we learned that there are very few points of irrevocable disaster. With an expert on hand, we had the confidence to try things out.

Figure 11. *Our experts, Turner Wilson and Cheri Perry*

(Photo from http://www.kayakways.net/, used with permission)

One of the best ways to learn any engineering discipline is to apprentice with a master, and this is indeed what we did with our kayaks. Having the opportunity to learn about failure modes, and to witness the tactics derived to mitigate these first hand, provides for a deeper level of understanding than simply repeating activities and getting everything right the first time. In requirements and software engineering, we rarely have the opportunity to work so

closely with such an individual. This is problematic to do, not only because the majority of the work that we do is solitary, abstract and intangible in nature, but also because there is not always a single individual with all the requisite skills and necessary patience. Instead, we tend to be thrown into the deep end. Our own mistakes are not always so immediately visible either. Many software-related errors are left for others to find and fix in the deployed future. Furthermore, when the inexperienced are left to procure core platforms and technologies to support a new software development venture, the shortcoming may never be overcome. In requirements and software engineering, we need to revisit the way in which we teach and train, and we need a way to identify those inspiring master builders.

12. Know the most precious resource and the most ubiquitous

The bandsaw was the most indispensible tool for all the kayak builders in the workshop; its reliability lay on the critical path for each of us as it was used to pre-cut the wood to approximate size. However, the bandsaw malfunctioned early on in the workshop and was subsequently unreliable. We further exhausted all the replacement parts for it. Thus, work on the critical path stopped for everyone as acquiring new parts became a lengthy off-island experience for one person. The most used item in all of our kayaks was dowels and we exhausted our supply twice. Running out of dowels therefore led to another hold up in operations. Clamps were both precious and needed in large quantities, and these were the one item that we had in abundance, since emphasised in the kayak building literature. When building a kayak, you can never have too many dowels, clamps or back-up bandsaw blades (see Figure 12 to appreciate why).

Figure 12. *You can never have too many clamps*

In requirements and software engineering, we tend to think of our key resources as people and money, and indeed they probably are. But, skilled people and money are prerequisites to do most things in life. A reliable inventory of traceable requirements, moreover, can be considered precious in software development and is something that we therefore need to maintain. Requirements drive the critical path of what we design and build. Traceability helps these requirements change as we learn more about them and facilitates confirmation of their eventual satisfaction. As to the most used resource in requirements and software engineering, it is possibly the myriad of channels that we rely on for communication, but generally take for granted. We often do not pay as much attention to planning for softer skills and team working needs, as perhaps we should. In requirements and software engineering, there are many critical resources that we neglect and need to pay far more attention to.

To be continued…

Did I avoid that irrevocable disaster? Did I return from Maine with a handcrafted skin on frame kayak? If I did, did it actually fit and float? Or, did this kayak-building project go the way of a typical software project? In Part II (RQ54), find out about the tooling, the testing and the teaming of kayak building. Find out what happens when you work in an unpredictable environment. Share my pains of balancing perfection with letting go.

Contact Details:

Olly Gotel, PhD (olly@gotel.net), New York City, December 2009.

RE-flections

Making Your Requirements Knowledge Count: Working in RE - RESG Event

University of Westminster, 2-4 pm, 11 November 2009

Ljerka Beus-Dukic welcomed everyone to this mainly-for-students event; it was good to see several non-student visitors too. The event was in two equal parts. First the four speakers each gave a short talk on how they used **requirements in industry or commerce**. Then the speakers formed a panel and answered questions from students about practice "in the real world".

The Panel: Keith Derham, Vesna Music, Phil Cantor, Ireri Ibarra

Phil Cantor (Smartstream Technologies) began by commenting that he had no idea how to get on to the Evening Standard's Rich List by working in software.

He found people in Amazon, Google, Facebook and Microsoft – but even these were not exactly software engineers.

He was not sure about the importance of book-learning in RE either; commercial practice was probably mostly "winging it" (it was evident that this expression was new to the students; it seems to be Victorian actors' slang for going on stage unprepared, relying on the prompter hidden in the wing of the stage for the lines) rather than following any academically-correct procedure.

But he was quite sure that choosing the best technical solution was wrong: in commercial "real life", time to market was key, since the first software product could take 80% or more of the business. Compared to this enormous competitive advantage, almost all technical requirements fade into insignificance.

Keith Derham (Barclays Capital) studied RE at Westminster after 6 years of life "in the real world", so he came to academic study of requirements with genuine curiosity about what one should do to solve the many problems of requirements work. The students looked at him with genuine curiosity, too.

It was not just a matter of building software "the right way". You could build a house "the right way" with walls that stood up and pipes that did not leak and a pretty white picket fence out the front, but if it did not deliver on its purpose, it was useless. A perfect 2-bed terrace house for a family of 10 would not be fit for purpose, however well-constructed.

Students would not find jobs advertised for "Requirement Engineers". In fact job titles meant next to nothing anyway. Derham was a "Systems Integration Analyst" – he got system A to talk to system B; but the work could change from day to day.

In RE, "you have to be a pacifist". You must not lean towards any solution; you have to break up a lot of fights. Everybody would like a big requirements document containing all the requirements, including all of their own: but that's no good. Instead, you have to "keep all communications channels open". That might mean taking minutes or arranging meetings. It was a team effort, working with people divided by a big financial organization into many small roles – requirements, development, rollout, service to clients. Everything is audited, traceable to the original decision-maker, controlled. It was very different from working in a small free-and-easy firm, with frequent requirements changes and short prototyping-style life cycles, whether or not those involve actual "Agile" practices.

RE, being unknown in Derham's corner of the world, is "a very helpful secret weapon"! Students should make the most of their time learning it.

Ireri Ibarra (RPS Group) worked in a very different area – system safety. Here, risk implied a very large requirement not to cause death, injury or loss of property.

Requirements had to be provable, traceable and feasible. You had to show through very careful work that risk was mitigated.

Sometimes, as in the aviation industry, there was heavy regulation to enforce a legal obligation to demonstrate safety. For instance, you might have to show – she apologized to the audience while displaying a grim clause from a safety standard – that the risk of death was no more than 1 in a million.

In other cases the regulatory hand was lighter, but it was still vital to ensure safety.

Safety had to be built in to systems from the start: it could not be "bolted on" afterwards. Half the requirements came out of safety analysis.

Vesna Music (Delphi Diesel Systems) – her name is pronounced "moo-shich" – was also a systems engineer, working in the automotive industry.

Her introduction to the importance of requirements came through a scarring experience on a job with a Far Eastern client. The task was to create a custom Electronic Control Unit (ECU), one of about 30 small computers scattered about a modern car.

An Electronic Control Unit (Wikimedia Commons)

The project had a very tight timescale and an absurdly small budget. Her company assumed the low figures implied that the job would be 90% off-the-shelf, with just a little customisation here and there.

It turned out to be 90% custom.

The client did not share Western assumptions about give-and-take in requirements trade-offs – we'll give you these features if you'll drop those requirements. The client wanted everything.

Eventually the job was finished; sign-off (and hence payment) was achieved only by going painfully through all the emails to prove to the client that he had in fact agreed to each requirement and schedule change. Ultimately it all came down to very careful traceability – there was nothing academic about it.

She used Simulink to show that the system produced the specified output under the specified conditions. In other words, it was an executable specification. That enabled her to go to the client and ask the classic prototyping question "Is that what you want?" – followed immediately [of course] by a torrent of missed requirements.

It was specially important not to miss requirements in a safety-critical system.

But not all requirements can be found by simulation. In one "horror story", she put a version of the software on the ECU processor "target" (i.e. a different processor from the simulator's), only to find that the shutdown checks when the driver turned off the ignition took a longer than expected time, about 150 milliseconds. You might think this not very serious, but the test drivers got up to quite a lot of tricks. They discovered that if they restarted within 60 to 90 milliseconds (!) the ECU got completely stuck – it had to be "flushed", which would mean a breakdown and either a workshop visit or a roadside fix with special equipment. The problem should really have been detected earlier but it wasn't covered by the specifications either.

Executable specifications are not just about prototyping. Execution depends on a model which serves as the single source of information about the design. Then you only have to change the specification in one place to change everything (if you remember to push all the right buttons, she remarked ruefully).

The panel took questions from the floor.

Were requirements really important? Phil Cantor thought not; meeting the requirements came a poor second to getting the product out of the door. He ranked the requirements strictly on a ladder – no equals allowed – and drew two [triage] lines across it to create 3 categories:

- the top few "you don't do the product if you can't do all of these";
- the "nice commercial features, you rewrite the sales brochure to sell whichever of these you manage to build";
- and the "won't ever be built" remainder.

Another way of looking at this was that in a Big Bank, the Project Manager worked to the Prince 2 methodology:

> *"on time, to budget"*.

But that focuses on the cost side of the equation. What about the benefits? A completely new approach is to appoint a Benefits Manager whose philosophy is

> *"what does this buy us?"*

Internal projects now got cancelled before expensively failing: a big improvement.

There was an extensive panel discussion about the importance of communication, both with people one-to-one, and by facilitating discussion (including through workshops). The panel agreed that this was easily the most important skill for getting the right requirements.

The overall impression that the panel gave was that industrial reality is simple and pragmatic. Academic theories of RE – goal modelling, how to use traceability tools, etc – may be very neat but practice is rather different. Tools of all kinds are in use: not surprisingly, the more formal kinds of tool are preferred in safety-related industries; less formal tools like Word and Excel are often the preferred media for communicating requirements elsewhere.

Industrial practice strongly emphasizes change, people skills, rules, and perhaps above all the unpredictability of markets and management.

RE-verberations

Executable Specifications in SequenceL

One of the nice things about getting older is that you recognise old ideas when, after they have had their time and gone out of fashion, they eventually come around again.

In the September 2009 issue of *IEEE Computer*, Daniel E Cooke and J Nelson Rushton revisit the set of exciting ideas around declarative programming languages in their article *Taking Parnas's Principles to the Next Level: Declarative Language Design* (pages 56-63).

Everyone knows that programming in a procedural (Alan Turing-style) language is error-prone: you spend more time on bugs and syntax than you do on actual algorithms. Back in 1981, Tony Hoare said "*the price of reliability is the pursuit of utmost simplicity*".

People have also realized, since the early 1980s (ahem), that Moore's Law would run out of steam on single-processor computers, so that progress would depend on using the enormous uplift in power of parallel processors.

Unfortunately, it is terribly difficult to adapt algorithms written in procedural languages to make efficient use of multiple processors – all those parallel threads make design and debugging even more of a chore.

At the same time, people like J W Backus tried to create short, clear executable specifications in functional languages like Hope and ML and Miranda. The idea was to apply nice clean functions to lists of single items (scalars), like this:

```
reduce(apply(square,myIntList),+)
```

meaning "the result of adding up the squares of all the numbers in my list". This was certainly short, but it wasn't terribly friendly, and it wasn't too easy to make it efficient, either.

But now, Cooke and Rushton have gone a step further. The problem with the functional languages was that since they worked on just one item, one scalar, at a time, you had to write funny things like `reduce` and `apply` to make them handle multiple items (lists, vectors, matrices), just as you had to write nested `for..do` loops in conventional languages like FORTRAN to do the same thing.

Either way (functional or procedural), you had to know a lot about how the language worked, inside the box, to make programs do anything useful. This is contrary to David Parnas' principle of information hiding: you should only see – and need to know about – the things you are actually specifying.

So the new language, SequenceL, works directly with both scalars and non-scalars like lists (of numbers, strings, etc) and matrices. And the programmer does not need to know anything about how it all works inside the box.

For instance, to take the square root of all numbers in a list:

```
sqrt([9,16,25]) → [3,4,5]
```

A function to multiply two matrices is simply

```
mm(i,j) (matrix m1, matrix m2) =
   sum(m1(i,all) * m2(all,j))
```

which reads pretty much how you'd expect.

Specification at last becomes clean and simple. For instance, set membership in SequenceL is

```
in(scalar X,any S) =
   or(X=S)
```

Using this, set intersection is simply

```
intersection(sets S1,S2) =
   (X when in(X,S1) and in(X,S2))
```

Now you may be thinking that this is all quite delightful for mathematicians. But what does it have to do with you?

Well, SequenceL can be interpreted or compiled down to C or C++, whereupon it runs efficiently: in the best case, actually faster than hand-coded algorithms; in the worst case, "programs approached C's execution time within a factor of 2". The programs were serious: guidance, navigation and control software for the NASA simulator for space shuttle and Orion, and the competing programmers were experienced NASA engineers.

And the most remarkable thing was, that when the programs were shown to a roomful of engineers, they all said they could understand them at once, and could see whether they were correct as specifications.

The benefit is that the specifications can at once be executed, so as soon as you have written your specification you have a working prototype. And sure enough, execution revealed several bugs that the NASA engineers had failed to notice in their own handwritten specifications.

What is more, any SequenceL operation on a non-scalar is automatically parallel. The more processors you have available, the faster the program will run, given a suitable compiler.

SequenceL is Turing-complete, meaning it can run any algorithm. It manages this with just 12 grammar rules, compared to over 150 for Java.

If Parnas, Backus and Hoare back in the 1980s had seen that, they would have been mightily impressed.

Chaos is Bunk (allegedly)

Sometimes those queasy feelings of something not being at all right are entirely justified. Only, you don't find out until years afterwards.

According to J. Laurenz Eveleens and Chris Verhoef writing in the Jan/Feb 2010 issue of *IEEE Software*, the much-quoted Chaos reports are seriously (for which read fatally) flawed.

Back in 1994, Standish created a sensation by claiming that only a "shocking" 16% of projects were successful, 53% were "challenged", and an astounding 31% failed outright.

Further, Standish presented a small table of figures which purported to demonstrate beyond doubt that the overwhelming reasons for project failure were down to poor project management, and in particular to lack of proper attention to requirements.

The figures arrived at just the right moment for would-be vendors of Requirements Management tools to set out their wares. They quoted Standish to "prove" that SOMETHING HAD TO BE DONE to improve requirements, and that something was their tool. It was far too convenient to question.

Requirements textbook authors and publishers, too, leapt onto the bandwagon. Page 3 of *Writing Better Requirements*, for example, quotes the Standish "Reasons for project failure", with "Incomplete Requirements" at 13.1%, "Didn't involve users" at 12.4% and so on. Mind you, the same book also mentioned a study in IEE Review which found that only 22% of project managers identified requirements as an important cause of project difficulties.

In any case, there was an odd mismatch with experience. Everybody had seen projects which were running a bit late, a bit over budget, or which had shifting requirements – but out-and-out disasters were not especially common, even allowing for the fact that people tended not to shout about them.

Doubts about Standish were raised by Robert Glass, writing in IEEE Software (2005) and elsewhere; and by M. Jørgensen, writing in *Information and Software Technology* (2006). They suggested, say Eveleens and Verhoef, that "the only way to assess the Chaos results' credibility is to use Standish's data and reiterate their analyses." But "there's another way: obtain your own data and reproduce Standish's research to assess its validity."

And that's what they did. They looked at no fewer than 5,457 forecasts applied on 1,211 industrial projects. Guess what they found?

"Our research shows that the Standish definitions of *successful* and *challenged* projects have four major problems: they're misleading, one-sided, pervert the estimation practice, and result in meaningless figures", they wrote. So much for feather-bedded academic pussyfooting: RQ hopes they've got good lawyers.

The trouble with Standish, explain Eveleens and Verhoef, is that success is defined as forecast/actual (f/a) cost and time are greater than or equal to 1, while f/a functionality is less than or equal to 1. Challenged is defined similarly, but using strict "greater than".

They quote Jørgensen to show that a project that is, for example, within budget and time does not fall into ANY of the Standish categories! They arbitrarily assigned such projects to *challenged*, which may be what Standish did also.

They point out that projects that are, say, 25% over budget or time are often perfectly acceptable. This seems obvious to RQ: work may have been novel and obviously risky; overruns may have been allowed for in contingency; and such projects may have led to software that brought in many times the money invested.

The Standish definitions also do not allow for underruns. Eveleens and Verhoef show by modelling their own data that estimation approaches such as Barry Boehm's or Tom DeMarco's rightly expect some estimates to be high and some low; but Standish does not. In one case, a large financial services provider made 667 careful estimates on 140 software projects. The forecasts were assessed independently by a metrics group at the time they were made, and shown to be "best-in-class".

Those projects were, in other words, exceptionally well managed.

But under the Standish definitions, working with the initial forecasts, the projects were (calculated Eveleens and Verhoef) only 59% successful. It wasn't the projects that were at fault, or the organization's management: it was the Standish methodology.

In another example, Eveleens and Verhoef compare Standish "success" to "real estimation accuracy". Organization X seems on the Standish approach to be far more successful than others; but that is because X "overestimates from tenfold to a hundredfold".

The authors communicated their findings to the Standish Group, "and Chairman [Jim] Johnson replied: 'All data and information in the Chaos reports and all Standish reports should be considered Standish opinion and the reader bears all risk in the use of this opinion.'"

Eveleens and Verhoef conclude "We fully support this disclaimer, which to our knowledge was never stated in the Chaos reports."

There are good reasons for doing requirements well. But dodgy statistics isn't one of them.

RE-readings

Competitive Engineering

A Handbook for Systems Engineering, Requirements Engineering and Software Engineering Management using Planguage

Tom Gilb

Elsevier Butterworth Heinemann (2005) ISBN 0750665076

Tom Gilb is a Systems Engineering consultant and a well known speaker. As the author of Competitive Engineering, he has set himself a challenge by aiming to provide a practical set of tools and techniques that enable readers to effectively design, manage and deliver results in any complex organization - in engineering, industry, systems engineering, software, IT, the service sector and beyond.

The central pillar to Competitive Engineering (CE) is *planguage*, an approach to communicating management objectives and requirements clearly and unambiguously. Planguage, a deliberate mix of Plan and Language, is essentially a common vocabulary based on a dictionary of clearly defined words. Gilb has been developing the idea since 1960 into more than a dictionary, and this book provides the framework within which planguage is applied. It also explains why the book appears to be so long – the glossary takes up over 100 pages!

The book is structured around four core processes or planguage methods:

- Requirement Specification to capture all requirement types, with an emphasis on quantitative attributes.
- Impact Estimation to evaluate designs against requirements and to track progress to satisfying the requirements.
- Specification Quality Control to check adherence to the plan, bid or technical specification.
- Evolutionary Project Management to plan and monitor implementation of the selected designs.

The emphasis on requirements is a consistent theme, and even without planguage the approach is clearly developed and illustrated with practical examples, no doubt derived from Gilb's own experiences. These practical examples give the book a solid feel, and help explain how the ideas and principles can be applied in the real world. In one example the rather woolly requirement for 'increased ease of service' for a mobile phone handset is given the planguage treatment to define measurable performance requirements.

I must confess now that whilst I can approve of the process I am still not sure about planguage. We have all been told a requirement must be unambiguous so it can be understood by all stakeholders, and that all words, phrases and terms used should be in common use or clearly defined. Planguage certainly defines the terms used in the process, but often to a level of detail that could prove to be a constraint, especially if being applied within a corporate engineering framework. For example, a requirement is "*a stakeholder-desired or needed target or constraint. Within planguage requirements specifically consist of vision, function requirements, performance requirements, resource requirements, condition constraints and design constraints...*".

I do, however, support the use of principles, additional ideas and templates to elaborate the glossary definitions. The requirement specification template illustrates the structure used to build up a requirement, with each of the terms defined by the glossary to explain exactly what is expected in each variable:

Requirement Specification Template (A Summary Template)

Tag: <Tag name for the system>.
Type: System.
========== **Basic Information** ==========
Version: <Date or other version number>.
Status: <{Draft, SQC Exited, Approved, Rejected}>.
Quality Level: <Maximum remaining major defects/page, sample size, date>.
Owner: <Role/e-mail/name of the person responsible for changes and updates>.
Stakeholders: <Name any stakeholders (other than the Owner) with an interest in the system>.
Gist: <A brief description of the system>.
Description: <A full description of the system>.
Vision: <The overall aims and direction for the system>.
========== **Relationships** ==========
Consists Of: Sub-System: <Tags for the immediate hierarchical sub-systems, if any, comprising this system>.
Linked To: <Other systems or programs that this system interfaces with>.
========== **Function Requirements** ==========
Mission: <Mission statement or tag of the mission statement>.
Function Requirement:
<**{Function Target, Function Constraint}**>: <State tags of the function requirements>.
Note: 1. See Function Specification Template. 2. By default, 'Function Requirement' means 'Function Target'.
========== **Performance Requirements** ==========
Performance Requirement:
<**{Quality, Resource Saving, Workload Capacity}**>: <State tags of the performance requirements>.
Note: See Scalar Requirement Template.
========== **Resource Requirements** ==========

Requirement Specification Template Extract

I really warm to Gilb when he introduces the chapter on Design Ideas and Design Engineering with the line:

> *The basic design process is finding 'means' for 'ends': it is finding designs that match the requirements.*

All too often projects fail because this basic principle is lost, and the design process diverges from the requirements. Iteration is necessary to improve the design *and* the related requirements – they cannot be determined in a single pass, and trade-offs might have to be made. Gilb introduces the concept of a *balanced level* of design and requirements, and until this is achieved it is not possible to finalise a competitive design for implementation. However, Gilb defines the process with so much detail and structure that I found it difficult to step back and visualise how the iteration and balancing happens.

Indeed, my only criticism is that whilst the level of detail and structure could be great for a new project wishing to apply an 'out of-the-box', requirements led process, that detail could overwhelm the project and prove to be too much of an overhead.

Competitive Engineering is clearly aimed at industry, and it is easy to see why Gilb and his approach have such an enthusiastic following. There is a lot in here that will take time to absorb but it is worthwhile persevering, helped by Gilb's style of presentation, humour and practical examples. Even if planguage seems to be a step too far, the approach and advice is first rate.

RE-partee

The Requirements Engineer and the Project Manager

A Project Manager and a Requirements Engineer are sitting next to each other on a long flight from London to New York.

The Project Manager leans over to the engineer and asks if he would like to play a fun game. The requirements engineer just wants to take a nap, so he politely declines and rolls over to the window to catch a few winks.

The Project Manager persists and explains that the game is real easy and is a lot of fun. He explains "I ask you a question, and if you don't know the answer, you pay me £5. Then you ask me a question, and if I don't know the answer, I'll pay you £100."

This catches the requirement engineer's attention, and as he sees no end to the torment unless he plays, he agrees to the game.

The Project Manager asks the first question. "What's the distance from the earth to the moon?" The engineer doesn't say a word, but reaches into his wallet, pulls out a five pound note and hands it to the Project Manager.

Now, it's the requirement engineer's turn. He asks the Project Manager "What goes up a hill with three legs, and comes down on four?"

The Project Manager looks up at him with a puzzled look. He takes out his laptop computer and searches all of his references. He taps into the Airphone with his modem and searches the net. Frustrated, he sends e-mail to his project support office--all to no avail.

After about an hour, he wakes the Requirements Engineer, hands him £100 and asks "Well, so what's the answer?" Without a word, the requirements engineer reaches into his wallet, hands the Project Manager £5, and turns away to get back to sleep.

Seen in a Hotel…

RE-sources

Books, Papers

RQ archive at the RESG website:
http://www.resg.org.uk

Al Davis' bibliography of requirements papers:
http://www.uccs.edu/~adavis/reqbib.htm

Ian Alexander's archive of requirements book reviews:
http://easyweb.easynet.co.uk/~iany/reviews/reviews.htm

Scenario Plus – free tools and templates:
http://www.scenarioplus.org.uk

CREWS web site:
http://sunsite.informatik.rwth-aachen.de/CREWS/

Requirements Engineering, Student Newsletter:
www.cc.gatech.edu/computing/SW_Eng/resnews.html

IFIP Working Group 2.9 (Software RE):
http://www.cis.gsu.edu/~wrobinso/ifip2_9/

Requirements Engineering Journal (REJ):
http://rej.co.umist.ac.uk/

RE resource centre at UTS (Australia):
http://research.it.uts.edu.au/re/

Volere template:
http://www.volere.co.uk

DACS Gold Practices:
http://www.goldpractices.com/practices/mr/index.php

Software Requirements Engineering Articles (India):
http://www.requirements.in

Media Electronica

RESG Mailing List
http://www.resg.org.uk/mailing_list.html

RE-online
http://discuss.it.uts.edu.au/mailman/listinfo/re-online

ReQuirements Networking Group
www.requirementsnetwork.com

RE Yahoo Group
http://groups.yahoo.com/group/Requirements-Engineering/

RE-actors

The committee of the RESG

Patron:
Prof Michael Jackson,
Independent Consultant,
jacksonma @ acm.org

Chair:
Ian Alexander,
Scenario plus,
iany @ scenarioplus.org .uk

Vice-Chair:
Dr Emanuel Letier
University College London,
e.letier @ cs.ucl.ac.uk

Immediate Past Chair:
Dr Peter Sawyer,
Lancaster University,
sawyer @ comp.lancs.ac.uk

Secretary:
James Lockerbie
City University London,
J.Lockerbie @ soi.city.ac.uk

Treasurer:
Dr Steve Armstrong,
The Open University,
S.Armstrong @ open.ac.uk

Publicity Officer:
Camilo Fitzgerald
University College London,
C.Fitzgerald @ cs.ucl.ac.uk

Membership Secretary:
Dr Yijun Yu
The Open University
Y.Yu @ open.ac.uk

Newsletter Editor:
Simon Hutton
Headmark Analysis Limited
simon.hutton @ headmark-analysis.co.uk

Student Officer:
Ben Jennings
University College London
b.jennings @ cs.ucl.ac.uk

Post Doctoral Officer:
Dalal Alrajeh
Imperial College
dalal.alrajeh @ imperial.ac.uk

Newsletter Reporter:
Dr Ljerka Beus-Dukic
University of Westminster
L.Beus-Dukic @ wmin.ac.uk

Regional Officer:
Dr Cornelius Ncube,
University of Bournemouth,
cncube @ bournemouth.ac.uk

Industrial Member:
Alistair Mavin
Rolls-Royce,
alistair.mavin @ rolls-royce.com

Academic Member:
Prof Bashar Nuseibeh
The Open University
B.Nuseibeh @ open.ac.uk

Regional Officer:
Shehan Gunewardene,
CAP Gemini/Aspire,
shehan.gunawardena @ hmrcaspire.com

Industrial Member:
Suzanne Robertson,
Atlantic Systems Guild Ltd,
suzanne @ systemsguild.com

Academic Member:
Dr William Heaven
Imperial College,
wjh00 @ doc.ic.ac.uk

Contributing to RQ

To contribute to RQ please send contributions to Simon Hutton (simon.hutton@headmark-analysis.co.uk). Submissions must be in electronic form, preferably as plain ASCII text or rtf. As an incentive, a copy of Tom Gilb's book 'Competitive Engineering' will be awarded for the best contribution to RQ 54!

The deadline for RQ 54 (April 2010) is Friday 26th March 2010

Joining the RESG

Visit **http://www.resg.org.uk/** for membership details, or email **membership-RESG@open.ac.uk**