Requirements Engineering Specialist Group

Requirements Quarterly

`

© 2010 RESG http://www.resg.org.uk RQ 54 (April 2010)

Contents

RE-soundings

From the Editor

I suspect the majority of RQ readers have bought into the concept that *Requirements Can Be Really Useful*. Indeed, to some of you this will be a statement of the obvious that does not need to take up space in our newsletter. But there are some who do need convincing, and I am often surprised that this group does include a lot of project managers.

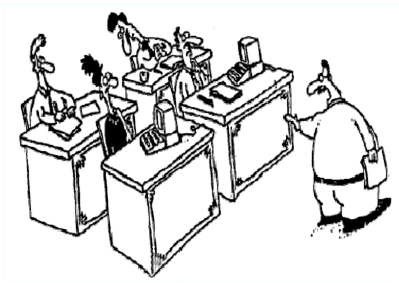
The contribution of requirements to successful projects is often identified in studies and articles. Two appear in the annual BCS Twenty:09 periodical (see www.bcs.org/server.php?show=nav.10360).

Bridging the Gap between Success and Failure by Dr John McManus examines the qualities a project manager needs for success, including identifying and refining needs, maintaining stakeholder involvement and managing compromise. George Sefri's article *The Seven Deadly Sins of Project Management* encourages us to ensure objectives are aligned with business strategy, managing expectations and stakeholder buy-in. All good requirements disciplines, so the Requirements Engineer should be a popular member of the project management team.

Project Management is not an easy life, aiming to balance the three related variables of Cost, Time and

Quality. Cost and time are usually well understood, manifested in the resourced project plan than can be analysed and reported using tools such as Earned Value Management. Quality tends to be less well understood as it moves into *Product* territory, but it is here that requirements can often support the Project Manager. A good Requirements Engineer will have an understanding of this relationship, and will reinforce the wider role of requirements by contributing to monitoring and control. Otherwise there is a danger that the requirements work becomes disconnected as the Project Manager pushes for *product* progress.

You may be familiar with the cartoon where the manager tells his team to do something (anything!) while he finds out what the Customer wants. All too often programme imperatives force us to do the wrong thing several times rather than the right thing once, a cultural disadvantage of any front-end loaded process that takes time to discover.



You guys start coding, I'll ask what they want

In addition to my membership of RESG, I must confess to membership of the BCS Project Management Specialist Group, to keep up with project management thinking so I can ensure my requirements are an integral part of the project. I also receive a monthly magazine called Project Manager Today, and last month there was a useful article by Steve Rifkin on *Managing Complexity in Large Development Projects,* extolling the use of Requirements. I am grateful to Steve and to Ken Lane, the editor of PMT, for allowing RQ to re-use the series of three articles. Steve presents a project based view on applying a requirement led approach that I am sure you will find of interest, especially to those working in industry and to those conducting research. Ian Alexander queries the effective range of academic arrows in his Chairman's Message - perhaps by understanding better the target and the industrial environment we can improve that effective range.

I am also grateful to Olly Gotel for the second and final instalment of her entertaining and inspirational description of building her own kayak, and how the lessons learnt are so relevant to requirements engineering. The first instalment appeared in RQ53 (January 2010), available on our web site at www.resg.org.uk/index.php/Newsletter for those who missed it first time around. Not only have I improved professionally from her article, my family is now the proud owner of a 16'Canadian Canoe, four buoyancy aids and a roof rack!

Finally, this edition of RQ includes a report of our *Goals Day* held in London last month. RESG events are often well supported and provide a wonderful opportunity to exchange ideas for minimal cost – something that would appeal to even the most sceptical and tight-fisted Project Managers!

Simon Hutton, April 2010

Chairman's Message

Having reviewed Parnas' opinions on Formal Methods (in RE-verberations, below), and participated in Goals Day, my thoughts returned to the question of applying research in practice.

At Goals Day, we heard about two rather academic methods, i* and KAOS. Both of these have long (15 or 16 years already) and distinguished research pedigrees. Both have led to many carefully-thought-out research papers and PhDs. Both are accompanied by detailed published evidence of having been put into practice in industry. Both are used and taught in a range of universities. Yet, neither is widely used in industry, and judging by the limited evidence of Goals Day, even well-informed engineers and analysts have generally not even heard of them.

This rather bleak dose of reality comes in spite of the fact that people in industry broadly agree that they aren't doing requirements very well. They know they need to be a bit more formal with their descriptions (and no, formal doesn't mean teaching everyone to write predicate calculus all over the place). Goal models of almost any kind would fit the bill very well.

Somehow, the message has not reached its target. Perhaps no arrow launched from the ivory towers of academia can hope to reach so far.

The only major change in requirements methods - since the NASA and DoD initiatives of the 1960s - has come from industry, in the shape of things like Use Cases and UML. These have been adopted enthusiastically all across industry.

UML does actually have a "goal" construct - it is, ironically enough, a drawing of an archery target. It seems to have been used very little, not least because it is not very expressive. I suspect the reason is actually linguistic: "goal" in the USA means something like "vision" or "mission". It doesn't have the vital connotation of possibly-small, possibly-decomposable requirement-sized thing, so it isn't susceptible of analysis.

Perhaps it's time for i* or KAOS protagonists to push for their adoption as UML models. When we hear analysts saying "This use case traces back to 2 softgoals and an obstacle" then we'll know things have improved.

Ian Alexander, Scenario Plus, April 2010

RE-treats

For further details of all RESG events, see www.resg.org.uk/index.php/Events

RESG Event – Postgraduate Workshop

23 April 2010, Room 218, Huxley Building, 180 Queen’s Gate, Imperial College London

Following the success of last year's student event, the Requirements Engineering Specialist Group is happy to announce the sequel:

"Postgraduate Workshop, the Reckoning"

The focus of this workshop is to gather like minded students together to share our work in a **low pressure** environment. This process hopes to inspire and provide a rich ground from which to push forward the state of the art in requirements.

This year, we are very lucky to have an opportunity to hear Anthony Finkelstein give our featured presentation. His work always proves to be thought provoking.

The final talk of the day will be a little different from the rest of the day but still closely related to requirements. **"Talking about Speaking"** will pose questions about the most effective way to present your material. This talk will lead into a group discussion on all material presented.

Up to date details are available on the RESG website at http://www.resg.org.uk/index.php/EventJan2010

If you have any questions, please contact: b.jennings@cs.ucl.ac.uk

RESG Event – Agile Requirements

26th May 2010, Shropshire

Julian Holmes of UP Mentors will give a talk on Agile Requirements. Further details will be available prior to the event, and will be provided on our web site www.resg.ork.uk as they are available.

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.

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

RESG Event – An Introduction to Requirements

19:00 onwards, 16th June 2010

CSA Lounge, Building 114, Cranfield University MK43 0AL

RESG has joined forces with the BCS Bedford Branch to host an evening event: "An Introduction to Requirements". In this mini-tutorial, Ian Alexander will answer the question *Why Requirements?*, and will show with short group exercises on Goals and Scenarios how to get started on discovering and structuring the requirements for systems and software.

Ian will be available after the tutorial to sign copies of his latest book Discovering Requirements, at a healthy Amazon discount.

Attendance is free, but registration is required. This can be done via the BCS Bedford Branch website at www.beds.bcs.org.uk.

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.

PLREQ'10 - International Workshop on Product Line Requirements Engineering and Quality

30 June 2010, in conjunction with REFSQ 2010, Essen

The workshop focuses on requirements engineering for software product lines. Traditional approaches for quality requirements and variability management do not fully address the problems associated with quality requirements during variability modelling, product instantiation and product line evolution.

The workshop provides a forum to discuss issues, novel approaches and tools within the area product line requirements engineering and quality engineering.

Quality in product line engineering is of major importance. In an organization with product lines, only good quality domain engineering artifacts are internally accepted by the engineers. Any quality flaws impact several products. However, this is also complicated by the fact that different products of a product line might be characterized by differences in quality requirements. Therefore, approaches should deal with variability in quality requirements as well.

RESC - 1st Workshop on RE in Small Companies

29th June 2010, in Conjunction with REFSQ'10, Essen

Small and Medium Sized Enterprises (SME) develop, customize and maintain a considerable part of software. Often, these companies are unable to apply RE methods and techniques without modifications. Besides, shortcomings in applying RE methods due to time constraints or limited resources may arise.

This workshop is intended to bring researchers together with the RE practices and experience of SMEs whose businesses are software. The workshop shall isolate current challenges regarding RE, occurring especially in smaller companies and outline an agenda heading towards solving these challenges.

RESG Event – Annual General Meeting

September 2010, Imperial College, London (TBC)

The RESG Annual General Meeting brings together our committee and members to review the year and to share plans for next year in a relaxed and sociable event. Further details will be available in the next edition of RQ and on our web site.

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.

Further details at www.re10.org

RESG Event – REET'10

Tuesday 28th September 2010, Sydney

The Fifth International Workshop on Requirements Engineering Education and Training (REET'10) held in conjunction with RE'10 will address issues related to RE education, both as part of a formal university degree and as ongoing skills training within the workplace. The workshop is intended to go much deeper than a surface discussion of curriculum issues and will examine specific ideas and techniques for teaching and assessing skills needed by an effective requirements engineer. The format of the workshop will include full papers, position papers, and pedagogical papers and activities that can be demonstrated during the workshop. The intent is to involve workshop attendees in performing the activities as well as interactive discussions about the topic.

Workshop topics may include curriculum development and creative contributions related to pedagogical techniques for teaching RE skills and could take the form of experience reports or demonstrations of specific teaching techniques and training materials.

The event is being organized by Joy Beatty (joy.beatty@seilevel.com) and Ljerka Beus-Dukic (l.beus-dukic@wmin.ac.uk). Further details are available at http://users.cscs.wmin.ac.uk/REET10/

RESG Event – Careers in RE Event

November 2010, University of Westminster, London

Further details will be available in the next edition of RQ and on our web site.

RE-Course

Mastering the Requirements Process

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, Atlantic Systems Guild

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

RE-sources Review

We have published a set of useful links at the end of RQ for some time now in the Re-sources section. I suspect some of these links may be out of date, and that you may be aware of a useful on-line resource that could be of use to our membership community. I am hoping to review this section over the next few months, and would welcome your contributions to simon.hutton@baesystems.com before 25 June 10.

Simon Hutton, Editor

RESG LinkedIn Group

We have recently started a LinkedIn group for the Requirements Engineering Specialist Group:

www.linkedin.com/groups?home=&gid=2662234

We hope to get our LinkedIn group going a place where those interested in Requirements can find and communicate with each other, so if you're on LinkedIn please do join!

Camilo Fitzgerald, Publicity Officer

Requirement Surveys

I am aware of two surveys being conducted at present, and would encourage you to participate - those conducting the surveys have kindly agreed to share their results with RQ.

Firstly, Yuri Chernak invites you to participate in his Requirements Reuse 2010 survey at:

www.surveygizmo.com/s/246670/requirements-reuse-2010

The goal of this survey is to gain visibility into the state of the practice with software requirements reuse in the IT industry in three areas:

- Practical experience with requirements reuse
- General project information
- Demographics of the survey participants

There are 22 questions in this survey, it should take you about 15 min to complete.

Secondly, Lihi Raichelson of Haifa University is conducting a survey as part of a research project based on the premise that organizational reality is a tight correlation between Information System (IS) requirements and business process design. In contrast, Requirements Engineering (RE) and Business Process Management (BPM) are separated and almost unrelated research disciplines.

The questionnaire aims is to explore the perception of RE and BPM by those who are involved in such activities. The response time is approximately 10 minutes and the questionnaire is at:

http://questionpro.com/t/CHS0xZDiIGu

Simon Hutton, Editor

RE-writings

Building Myself a Kayak: Some Lessons for Requirements and Software Engineering - Part 2

Olly Gotel

In Part I of this article (RQ53), I explained my determination to build a traditional Inuit skin on frame kayak last summer. With no prior knowledge of either kayak building or woodworking, I became an apprentice to an expert in a small kayak-building workshop on an island in Maine. As I went through the process, I recognised so many lessons for requirements and software engineering that I felt compelled to share my top twenty observations with others. In Part I, I discussed the role of communication, measurement, modelling, visualisation and architecture in kayak building. I also highlighted the issues of resourcing and the pivotal role of a keen eye. In Part II, I discuss eight more observations … and you get to find out the outcome.

Continued…

1. ***Use the right tool for the job***

While it was possible to use a small number of tools when building our kayaks, life was made far easier when the right tool for the particular job was used. The difference in size and angle of the plane that was used could make a task last either ten minutes or take two hours. A metal rasp could equally halve the time of a sandpapering task. A hot knife would seal the cut edges of the fabric at the same time as it was cut, thereby completing two tasks in one. Curved sewing needles would make a directional pull of thread around difficult wooden corners both feasible and easy, as evident in Figure 13. The smart kayak builder not only has a well-equipped boathouse and toolbox, but knows exactly what to use and when, and they can interchange between the use of these tools seamlessly[1].

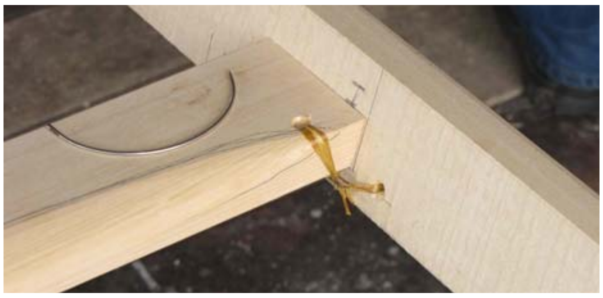


Figure 13. *Lashing made easy using a curved needle*

The large majority of tools in requirements and software engineering are general purpose and require considerable knowledge and tailoring to exploit fully. While there are also tools dedicated to niche techniques and tasks, they can often be difficult to use in concert, or it is prohibitively expensive to do so. More critically, there may be little that is transferrable from one tool to another by way of technique, so the value in taking the time to learn the tool could be

[1] On reading a draft of this article, one of the kayak builders (Eben) remarked on how he has always been fascinated by the power of tools to increase the equivalent skill level of a person: "*With a workshop full of tools we novices were able to produce a complex hunting craft equivalent to what an expert using just a hook knife and bow drill could make.*" Being a professional software engineer, he also drew the parallel: "*With the right IDEs (Integrated Development Environments) and function libraries an average developer can create and debug complex systems like database and web apps.*"

debatable. Given that the learning curve for tool use can be high, and the longevity of the acquired skill uncertain, we naturally resign ourselves to the use of the few tools that we own and know. Our relationship with tools is also quite different from that of a kayak builder's – do we keep our tools organised, neat, sharp and ready-to-hand? In requirements and software engineering, a collection of smaller and simpler tools may be the way to go, but only if we can get back to the essential underlying techniques that we need.

2. ***Master the underlying techniques to capitalise upon use of the tool***

It is not sufficient simply to have a collection of different sized chisels or planes in your toolbox. You need to learn the technique of chiselling or planing if use of any one of these tools is to be effective. As suggested in Figure 14, this is only achieved through practice on the job. Even the way you swing a simple hammer, either with or without gravity assistance, is going to impact how well and how easily it works in practice, irrespective of its size or your strength. Some activities we repeated over and over during the workshop, and we began to master the techniques with practice, such as sawing, dowelling and lashing. Other techniques would obviously take a few more kayak-building experiences, like mastering the transfer of angles with a sliding bevel and intricate spokeshave manoeuvres.

Figure 14. *Mastering that planing technique*

In software engineering, we sometimes do not distinguish the technique from the tool, assuming that we will acquire any necessary technique through use of the tool. This is a notable issue when it comes to requirements management tools. Do we really know what the fundamental techniques are that we need to learn and master to capitalise upon use of our tools? We can name the underlying techniques in the kayak-building world, but in the software world they tend to be somewhat vague and large in their scope: elicit requirements, specify requirements, code, test, etc. Also, where and how do we pick up these skills? This is on the job and under the watchful eye of a master kayak builder when learning to use the tool in the kayak world but, in the software world, we are often expected to make the transfer to a practical setting from the classroom on our own. Many of our tools are so overwhelming and so feature rich that we never quite get to become fully competent in the core underlying techniques that matter, the techniques that will serve us best in the longer term. In requirements and software engineering, we need to highlight the foundational and transferrable techniques of the discipline if we are to focus on them more overtly.

3. ***Embrace diversity and pull together***

Although each of us in the workshop was learning all the skills that we needed to build our own kayaks, some of us were naturally more talented at some tasks than others. I was considered a good lasher and seamstress, so I was able to trade the application of these skills for intricate carving or drilling tasks on my own kayak by others, two things that I certainly did not master. Eben carved the shark bow piece for my kayak that is shown in Figure 15[2]. While much can be done individually, some tasks do work better in pairs, such as having an extra eye to help maintain an angle whilst drilling, while some tasks require a whole team, such as four people to steam wood, then bend and affix the resulting ribs quickly. Given the nature of the workshop, as a joint learning experience, we all needed to stay somewhat in step in our kayak building schedules to make it manageable. So, when one person fell behind, we would all pull together at the end of the day to ensure alignment. We all wanted everyone to succeed in his or her own work.

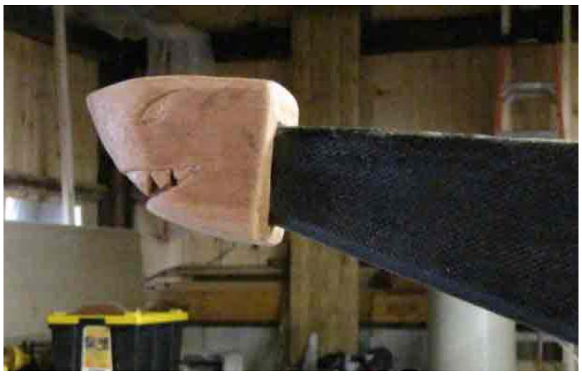


Figure 15. *We may not all be skilled at carving, but we can share skills*

It was once accepted in requirements and software engineering that individuals would specialise in particular areas and work together as an integrated product team to leverage complementary skills. More recently with the agile movement, the idea has become that individuals should be able to undertake any task in the software development process, any one member of the team being replaceable if the knowledge and skills are circulated continuously within the team. Competence and skill in all areas takes time to master though and we cannot always all be acknowledged experts at everything. What if all the team members are at the beginning of their skill learning curves? In

[2] For someone who spends so much time on and in the ocean, I still have a great fear of sharks (I blame it all on reading and watching 'Jaws' when I was way too young!) This bow piece is my talisman.

requirements and software engineering, we need to take care in resourcing our projects to blend the competencies and personalities of individuals, while nurturing their individual career aspirations, rather than simply expecting people to be autonomous cogs in jobs.

4. ***Provide for a healthy and safe working environment***

In a woodworking environment, there can be a lot of dust and noise when many people are sawing, drilling and hammering all day long. Protecting eyes and ears is critical, as is ensuring adequate ventilation and light. Equally important is ensuring that people are alert and aware around any machinery, and so they need to be well nourished and work at a sustainable pace. We were fortunate to have a team of fantastic cooks on hand who made sure that we had lots of refreshment breaks, often outside on the dock. We also listened to music to help some of us focus, while ongoing chatter about kayaks and paddling kept others fully immersed in the process. Where there was idle time, as some people had completed tasks before others, such time was used to sweep and clean the boathouse, collect and tidy tools, and to ensure the retention of a workable environment. A tidy moment is captured in Figure 16. We were not just workers doing a job; we were enthusiastic apprentices keen to learn and share, passionate about the product we were building and we wanted to enjoy the process in every way possible.

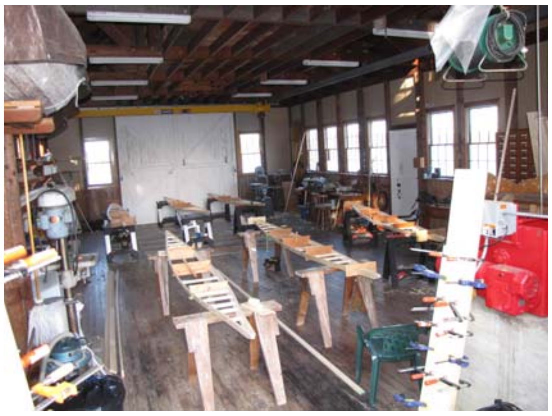


Figure 16. *Maintaining a tidy work environment*

Health and safety issues are not so easy to identify when it comes to the work of requirements and software engineers. Nevertheless, endless days of computer usage, poor posture and flood lighting can all take their toll. The idea of refreshments often only extends to pizza and caffeine in our world. It is consequently common for individuals and teams to reach burnout as deadlines arise. While some enlightened workplaces focus on ergonomics and comfort for their employees, imagine what a team of requirements and software engineers could accomplish with an in-house chef cooking up delights and forcing imposed downtime in natural surroundings! Also, imagine the commitment if the team had some future stake in the ongoing use and success of what they are each creating. In requirements and software engineering, we need to prepare and tend to our working environments much more carefully if we are to demand excellence from our engineers.

5. ***Recognise that the best is the enemy of the good***

For the first five days of the workshop, I wanted everything to be perfect. I even smoothed out rough edges on wood that would remain hidden inside my kayak and created a thumb nook to help me get into my kayak (which, to date, I have never used). With some non-critical measures, rough approximations using fingers were wholly sufficient as a heuristic, but I would insist on lining everything up and using my tape measure to be absolutely precise. As time passed, I realised that I was sweating the small stuff too much and needed to compromise on perfection if I was to finish on time. It was the 7:00am to 2:00am work day on day six[3], when steaming, bending and fixing the ribs for my kayak (as a team, at top speed and still falling behind), which led to this epiphany (Figure 17).

Figure 17. *There is no time to fuss about when working the steam box*

When we undertake requirements and software engineering activities, we rarely have any benchmarks to judge how efficiently and effectively we are working. Most of our estimates are arbitrary and individual productively, for one, can be a controversial thing to measure. Also, many projects do not start with a clear understanding of what it takes to be considered 'done', so we keep on with many activities long past the point of return on investment. It is essential to have a prioritised set of requirements and accompanying acceptance tests to ensure that we focus on what matters most and, in the interests of a wider schedule, know when it is time to move on. In requirements and software engineering, a perfectionist can be as hazardous as a negligent, so we need to learn how to recognise what is 'good enough'.

[3] Yes, I really did work for about seventeen hours that day, once all the enforced meal breaks had been factored in!

6. ***Watch that urge for closure***

Come the final days of the woodworking, I just wanted my kayak frame to be finished. All the care and attention that I had put into the previous days could easily have been wiped out with a careless slip of the drill. The trigger for the change in my behaviour was the realisation that I was not going to be completed in the two weeks I had expected, so I would not be taking my kayak home to New York City with me. None of us would be finished and we would all have to return to Maine later in the summer to skin our kayaks. The reasons are explained later and mostly come down to the environment (the Maine weather). At the time, I thought that I could speed things up and skin my kayak before the waterproofing was dry, just so the process would all be over in the one visit. Although waiting was painful, as the inactivity of Figure 18 shows, the alternative would have been detrimental. On my return visit to Maine, I was concerned that something else would go wrong and require a third visit, so I sewed up the skin at top speed on my arrival. I then desperately wanted to get the skin painted before the caulk sealant had dried, so that the paint would have a chance of drying before my scheduled departure. That would have been detrimental too, so I waited. The problem was that I could see my kayak emerging in front of my eyes. All I wanted to do was to take it home, get it on the water and paddle it before the summer was over. I did not want to chance nature again. As luck would have it, we had a glorious sunny painting and drying day, and we all took our finished kayaks home after the second trip north. Patience is so much harder to muster when the end is in sight, but yet so far from attainment. It is consequently the easiest time to introduce that irreversible disaster. This is the one time when the advice of the expert is the most invaluable, even though it can be very hard to digest at the time.

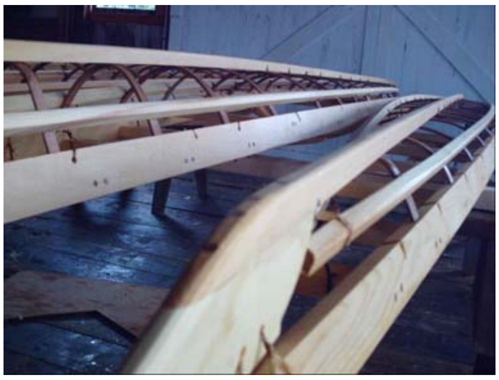


Figure 18. *Waiting for the waterproofing to dry (Note that the horizontal wood is the keel, chine and gunwale as you look down the upside-down kayak)*

In software engineering, the time when activity seems to peak on a project is when the reality of a deadline draws near. It is the time when a bug is fixed hastily, thereby introducing far more new bugs, and when other corners are cut to ship a product. This is exacerbated with an abstract intangible product like software – who can see that omission or shortcut? Irrespective of how or why this happens, it is the time in the process when more quality checks and balances than usual need to be imposed. In requirements and software engineering, if any part of a software project is to be scrutinised more than the initial requirements, it is the activities of those final hours.

7. ***Conduct early testing and a final fairing***

At many points during the workshop, we could actually sit in our wooden frames to check the dimensions and fit of our kayaks, as illustrated in Figure 19. One person even dropped his frame on the concrete floor of the boathouse to check that it would not fall apart. It didn't. We all undertook our own forms of idiosyncratic testing along the way, to convince ourselves that the kayaks we were building would actually fit, because they all looked so very small. However, we never once tested that they would float, so we took the primary requirement for any kayak completely on trust. We also never tested out their likely behaviour ahead of time to see whether they would perform as expected on the water. We relied on sketches and trusted the opinions of our experts. Prior to the skinning, we all conducted what is called a 'final fairing'. This is a careful look over the shape of the kayak, and a smoothing of all the rough wooden edges, to assure the eventual profile and silhouette once skinned. Any lashings needed to be seated into bevelled wood to permit for a non-lumpy skin to avoid drag in the water. We focused entirely on testing the form and fit; we relied entirely on expert judgement for all the rest.

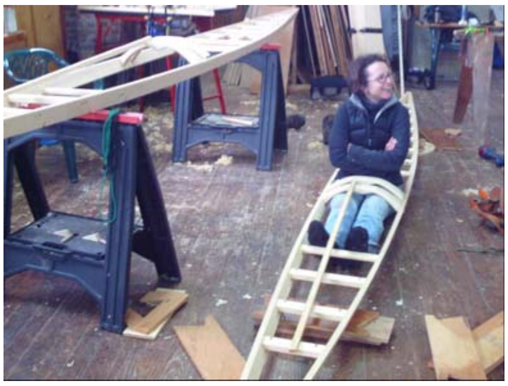


Figure 19. *Testing it fits before taking the next step*

In software engineering, we unit test fragments of code with some regularity, though we often tend to leave the overall functional testing of the assembly towards the end of the process. Early user testing is possible when we work with customers and end-users during the requirements activities, but we sometimes forget to continue to do this as the software evolves and as time becomes pressing. It is obviously easier to do all forms of testing when the product you are building is for yourself and when it is in your hands, literally. However, the idea of a final fairing applied to requirements and software engineering is an appealing one, ensuring a final check over all aspects prior to points of commitment. We attempt to do this with inspections, but we do this neither routinely nor rigorously. If only we could judge how a set of requirements would behave by standing back and simply looking at them from different angles. In requirements and software engineering, we are beginning to bring testing to the forefront, but we still have plenty of scope to make this practice integral to the forward engineering process every step of the way.

8. ***Determine the risks and have a back-up plan***
There were certain time windows that we needed to adhere to if we were to finish building our kayaks in two weeks. After the first few days, we were on schedule, and then things slipped. It became evident that our most valuable resource, the bandsaw, was compromised (as discussed in Part I). More critically, the damp Maine weather meant that everything took twice as long to dry than expected, quite the problem when the first level of integrity of the kayak frame is provided through glued wooden dowels. The Maine weather that we had counted upon is shown in Figure 20, but we only got that kind of weather maybe three days out of the two weeks. As a result, we experienced delays and only finished the wooden frames on the original visit, despite working twelve to seventeen hours every day (these long working days were only possible for the reasons discussed in 16). We returned to Maine to skin the frames six weeks later over a long weekend. On that trip, we had glorious sun.

Figure 20. *One of those rare good weather days in Maine*

Not finishing a project on time, with the functionality as initially anticipated, is accepted as the norm in software engineering. Risk identification and mitigation strategies can help us build in better contingency to project plans but, to some degree, we cannot anticipate everything and control the natural environment. If we did not accept some risk, we would never get to start on engineering anything, so we learn to proceed with acceptable risk. Time criticality is perhaps what really sets software engineering apart from kayak building. Deliver late and you may blow a contract or miss a market, and consequently do not get paid or lose a business. With my kayak building, there would always be another day for my indulgence. In requirements and software engineering, irrespective of the best project planning and risk assessment, perhaps we too can blame the weather sometimes?

In Closing…

My rolling kayak was worth waiting for and well worth building right. Yes, it floats, and the evidence of this is given in Figure 21! It also fits me like a pair of trousers and I can roll it like a whirling dervish when in the water. Not only did I plan, design and build something that required hard physical labour (hard for me that is), as well as acquire many new skills, I gained a wider appreciation for why it is necessary to apprentice with an expert when learning any engineering-related craft. The value of such an experience is further magnified when you can observe and learn from other apprentices who are going through the same process with you.

The technical world of requirements and software engineering is far removed from that of traditional Inuit kayak building. I would not even begin to claim that they are comparable in their complexity, as every now and again we see software systems that are engineered that stretch the bounds of the imagination. However, the kayak building process is engineering in the most classical sense, and it is a tradition that has survived and been passed on through the generations in Greenland. In going through the learning process myself, I decided that the requirements for a personal kayak are quite easy to determine once you get to the heart of what you actually want and why. Getting to pare down to those essentials however, in a world of endless possibilities and dreams, is the really hard part to do. I got what I really wanted ONLY by letting go of what I did not really need. That was a critical insight for me.

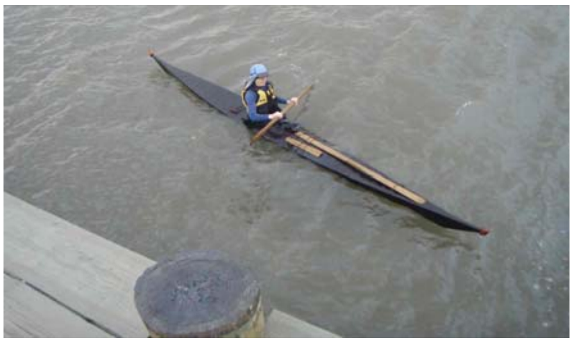


Figure 21. *My kayak and me, together at last!*

However, I probably could not have undertaken this exercise had I not been a pretty advanced kayaker. The trade-offs that I made and the precision that I required only made sense because I felt them, personally. We are rarely experts in the domains that we create our requirements and software for, but apprenticing to learn about the product domain is as essential to our success as apprenticing to master the engineering process. But even as domain experts, we certainly could not have built a single kayak to suit all five of us at the workshop. The problem of achieving an agreed set of requirements amongst a set of stakeholders was blatantly clear in this context. This issue remains one of the largest and underappreciated challenges of software requirements engineering; the only way in which this is sometimes possible is to build a software system that has compromised the requirements away.

Most importantly, we could not have evolved our requirements, and then satisfied them in our kayak designs and builds, without our kayak master builder's hard earned, tried-and-tested experience being passed on to us every step of the way. I therefore urge all requirements and software engineers to experience being an engineering apprentice in this same spirit. Build something that you have always dreamed of, but lacked the know-how and skills to do. You will gain a wider appreciation for why we, in requirements and software engineering, have such a long way left to go in understanding the nuances of our own engineering discipline and in passing on our craft to others. You never know, your reflections may help us all as we proceed on this important journey, and you might even have some fun! ☺

Acknowledgements

My special thanks go to Turner Wilson and Cheri Perry of Kayak Ways LLC (*http://www.kayakways.net/*) for making this kayak-building workshop such a memorable engineering experience for me! It was a real joy to see such a skilled and passionate pair of kayakers selflessly enabling others to pursue their own kayak building dreams. As a pair, Turner is the kayak master builder of this article, while Cheri is one of the best Greenland-style kayak rollers in the world.

The workshop would not have been such great fun if it had not been for my four fellow kayak builders: Dave, Eben, Bob and Geoff. I learned so much from each of them and their contributions are all now part of my aptly named 'Shark Attack' kayak. Thanks to Dave and Eben for taking the photos that were used in this article and, given they are both IT professionals, for suggesting connections between kayak building and software development that I had overlooked. I also thank Eben for making his wonderful waterfront boathouse in Vinalhaven our home for the workshop. I particularly thank Ali, Elise and Patsy for making sure that the workshop felt more like a luxurious gourmet eating vacation than the extremely hard work that it actually was!!!

Finally, my mum read and commented on early versions of this article, under some duress! While she is a keen kayaker, she is a reluctant user of technology and she has never quite understood what I do for a living. Not only does she now have a vague idea about requirements and software engineering, she also ensured that I did not veer off into a technical reverie and stuck to using Plain English! I also thank Ian Alexander and Simon Hutton of the RESG (Requirements Engineering Specialist Group of the British Computer Society) for their insightful comments and their enthusiasm to see my musings shared in RQ (Requirements Quarterly).

Glossary

Kayak and maritime terms used in this article:

Bow, Chine, Cockpit, Coaming, Deck Beam, Deck Lines, Final Fairing, Float Bags, Gunwale, Hull, Inuit, Isserfik, Keel, Lashing, Masik, Port, Qajaq, Rib, Rocker, Roll, Skin, Skin-on-frame, Starboard, Stem Piece, Stern, Track.

Please see:

http://www.qajaqusa.org/Movies/audio_glossary.html (for qajaq terms).

Please see:

http://andrews.com/kysc/terms.html (for maritime terms).

Woodworking terms used in this article:

Bandsaw, Chamfer, Chisel, Clamp, Dowel, Drill, Hammer, Hot Knife, Japanese Pull Saw, Plane, Rasp, Sandpaper, Saw, Saw Horse, Sliding Bevel, Spokeshave, Story Stick, Windlass.

Please see:

http://www.woodworkinghistory.com/glossary_access.htm.

For Further Reading

- Cunningham, Christopher. *Building the Greenland Kayak: A Manual for Its Construction and Use.* Camden, Maine: Ragged Mountain Press, 2003.
- Dyson, George B. The Aleutian Kayak. *Scientific American Magazine*, Volume 282, Issue 4, April 2000.
- Golden, Harvey. *Kayaks of Greenland: The History and Development of the Greenlandic Hunting Kayak, 1600–2000.* Portland, Oregon: White House Grocery Press, 2006.
- Morris, Robert. *Building Skin-on-Frame Boats.* Hartley and Marks, 2001.
- Peterson, H. C. *Instruction in Kayak Building.* Nuuk: The Greenland National Museum and Archives and Atuakkiorfik/Greenland Publishers, 2001.
- Qajaq USA. *The American Chapter of the Greenland Kayak Association.* (Peruse their extensive website of resources, *http://www.qajaqusa.org/*.)
- Zimmerly, David W. An Illustrated Glossary of Kayak Terminology. *Canadian Museums Association Gazette*, Volume 9, Issue 2, 1976. (Can be downloaded from his extensive Annotated Bibliography of Arctic Kayaks, *http://www.arctickayaks.com/biblio.htm.*)

Contact Details:

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

Managing Complexity in Large Development programmes

Article 1: Problems that are frequently encountered

Steve Rivkin

This is the first of three articles on Managing Complexity in Large Development Programmes. This first article describes a number of problems that occur frequently with large development programmes, and suggests how these can be avoided. A scenario is described, whereby contracts are awarded based on small sets of informally-specified Stakeholder Requirements, and designs are subsequently produced based on these requirements. The problems that may result as a consequence of this approach are described, and an alternative, 'Requirements-driven' approach is recommended.

One of the major strengths of UK industry is the vast amount of experience available that has been gained from working on numerous projects and involvement in large development programmes. Typically, senior staff and specialists from the both the major stakeholders, and the general pool of talent available in the marketplace, come together during the Concept and Definition[4] phases (Figure 1) to form the 'Project Team', which in this context, means the client-side team responsible for managing the development programme, and the contracted-out projects within the programme.

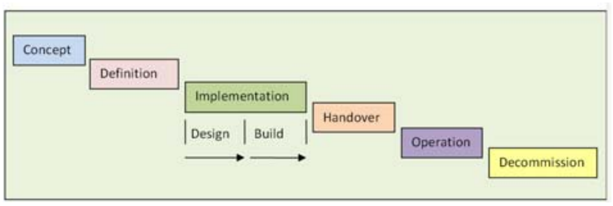


Figure 1 Key Programme Lifecycle

Despite this wealth of experience it is often the case, particularly on large development programmes, that problems are created by insufficient preparation during the Definition stage of the programme, and these problems only become visible in the Design and Implementation stages. Some typical areas where preparation can tend to be weak are:

- The major interfaces between key subsystems have not been identified;
- The key dependencies between projects within the programme have not been established;
- The programme infrastructure, and its supporting processes, has not been established sufficiently before the main contractors and project teams come to site;
- The Stakeholders' Requirements, and any resulting Programme and Project Requirements, have not been defined adequately.

Interfaces and Dependencies

During a typical procurement phase, it is common practice to award each of the major contracts for the Implementation phase on an individual basis. The bidders each submit a draft project plan as part of their proposal, and on appointment, the successful contractor usually refines their plan immediately after the start of the contract, adding more detail. It is after this point that some of the problems begin to surface. For example, Contractor A's plan may have a dependency on Contractor B for site data information, and needs that information to be provided by a specific date, and in a specific format (Figure 2). Contractor B's plan specifies that the site data will be provided in a different format, and at a much later time than Contractor A's plan date for receipt of the data. Each contractor has been awarded a contract that has accepted the basis of the draft plan in their proposal. The series of meetings that are needed to resolve the mismatch results in loss of valuable project time, and probable costly payments to each contractor for variation orders.

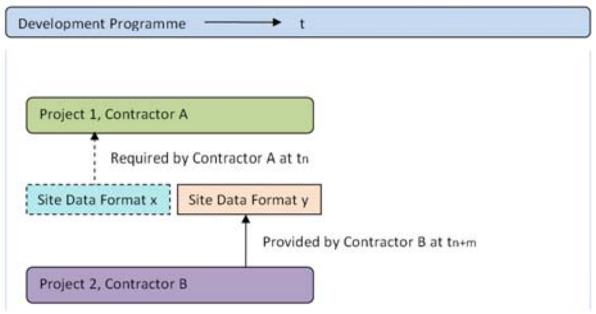


Figure 2 Misalignment of Dependencies and Mismatch of Interfaces

The causes of this type of problem lie in the roles and responsibilities of the various members of the Project Team. Whilst each individual should have a well-defined role and responsibilities, focusing solely on these may lead to a degree of 'tunnel vision'. The tendency is for the technical specialists to focus on the technical aspects required for each individual project, and the contract and procurement staff to concentrate on the commercial issues of the specific contracts associated with each project. Whilst this approach may well be appropriate for each project/contract on an individual basis, it misses the 'bigger picture', and the interactions, interfaces, and relative timescales between the projects are not taken into account. There needs to be an awareness of the roles of other members of the team, and how these roles interact. This is particularly important for the Project Team, which has been

[4] The Concept and Definition phases used here are as defined in the APM Project Lifecycle. A preferred solution is identified in the Concept phase, and this is further evaluated in the Definition phase, with options produced to meet the solution, and plans prepared for the implementation phase of the project.

populated by staff from a variety of cultures and backgrounds.

The problem could have been avoided by cross-checking the plans of the respective contractors prior to award of contract, and having a greater involvement of the senior engineering and planning staff in the procurement process. By spending more time on determining the inter-subsystem interfaces, and their nature, together with identifying, and agreeing the timing and specific requirements of the inter-project dependencies, the initial project plans could have been aligned prior to Award of Contract, and the requirement mutually to evolve and agree the details of interfaces and dependencies made a condition of the contracts. The contracts should also require that an Interface Manager from the client-side Project Team chairs the interface meetings with the contractors' project teams, and helps to broker the agreement of the interfaces and dependencies.

Infrastructure

Once the contracts have been awarded, and the start date for the contracts is reached, the contractors' staff start to arrive on site, and the spend rate on the various projects of the programme increases significantly. Frequently, the programme and project infrastructure have not been established. The contractors' project staff are keen to make progress, and start to produce project documents without the appropriate processes in place to guide them. As a consequence of this, much rework may be required to ensure that the documents conform to the project standards that are eventually established, that they adhere to the quality procedures, and that the appropriate reviews have been conducted.

A much more efficient use of time would be to get the senior staff to create the programme and project infrastructure. Even if the senior staff and contractors may have worked previously on similar types of development programme, each programme has a different geography and context, and the processes need to be tailored to the specific programme, and the processes integrated, so that they work together. Review procedures, document control and management procedures, change control, risk management, and all the other necessary project and programme management procedures, could be created in advance of the arrival of the majority of the contractors' staff. Also, agreement could be reached between the major stakeholders, regarding issues, such as: the makeup of the Project Review Board; the content of Gateway Review documentation; the format of the programme and project risk registers and the relationship between them; the structure of design documents. Also, for each type of report, a single style should be defined for the Project, thereby avoiding unnecessary duplication which occurs when major stakeholders insist on reports being produced to their own specific standard.

Requirements

Even on very large programmes, it is frequently the case that the Stakeholders' Requirements consist of a small set of ill-formed requirements that state, at a very high level, what is required from the development programme. These requirements are often loosely specified, and sometimes constitute little more than a 'wish list'.

The fact that the set of Stakeholders' Requirements is not rigorous, or comprehensive, is not necessarily viewed as a problem. The contractors' familiarity with the nature of the work required by the programme allows them to proceed with the design because, based on previous experience of similar programmes of work, they 'know' what is required. Usually, a more detailed set of System Requirements may be produced by the Project Team in order to expand on the set of Stakeholder Requirements, and provide a better source of guidance prior to the production of the design. However, the System Requirements are usually not linked formally to the Stakeholders' Requirements, and the designs produced by the contractors are not linked to the Stakeholders' Requirements, or to the System Requirements.

A risk-averse culture exists currently, and when the legal team for the Project Team becomes involved during the procurement stage, they invariably advise that the individual contracts for the Implementation phase must only be awarded against well specified sets of requirements, rather than against any preliminary designs (such as a Reference Design to gain Approval in Principle) that may have been produced by the Project Team. The reason for this is that, if the contract was based on the Reference Design, and this design caused subsequent problems, the contractors could claim that they were not liable, because the contract obliged them to base their design on the Reference Design, which proved to be flawed. Awarding the contracts against appropriate subsets of the System Requirements places the risk on the contractors, because they have to meet the requirements specified in the contract, and where a Reference Design is provided, it is provided for information only, and the contractor carries the risk if they use any part of the Reference Design.

Despite these precautions, there are still problems with the above approach:

> There is no linkage between the Stakeholders' Requirements and the System Requirements, so that it cannot be shown that System Requirements truly represent what is required by the Stakeholders, and therefore, there is no clear, detailed specification from which to produce the design;
>
> There is no auditable traceability from the Stakeholders' Requirements down through the System Requirements, to the design, so there is no way of showing that the design exactly fulfils what

is required by the Stakeholder Requirements, and the more detailed System Requirements;

When the completed designs are eventually studied by technical specialists on the client-side, aspects of the designs may be deemed to have failed to meet the System Requirements, and/or the Stakeholders' Requirements;

Unfortunately for the contractors, by the time the design problems have been identified, some, or all, of the teams who produced the designs will have been assigned to other work. In addition, because the designs did not completely meet the contractual requirements, much, or all, of the cost of the additional work may be charged to the contractors.

The above scenario can be avoided by using a rigorous 'Requirements-driven' approach to evolve the Requirements, starting design work only when a clear set of Requirements, at the right level, is available. The approach provides complete, auditable traceability from the Stakeholders' Requirements, down through more detailed levels of requirements that correspond to the more detailed levels of design, and throughout the whole of the development lifecycle. Verification evidence is also provided, justifying all the choice of all requirements derived from the Stakeholders' Requirements, and showing **how** all the design elements that result from these requirements implement **what** is required by each requirement. This approach is valuable for all development projects, but it is particularly important in sectors where Proof of Safety is required, such as rail, nuclear, and aviation.

The next article in the series will describe a Requirements-driven approach, and the benefits associated with the approach.

Dr Steve Rivkin, Ontrackprojects

Dr Steve Rivkin BTech MSc PhD CEng MBCS CITP *is a freelance consultant, specialising in systems integration, requirements management, and project management. Consultancy assignments include project management and systems integration work in the transport sector, on both Highways Agency and major rail projects. Steve has had responsibility for defining and implementing Requirements Management strategies in the rail industry using the DOORS Requirements Database*[5]*. He has also given workshops and presentations in the nuclear sector describing a Requirements-driven approach to system development. Steve can be contacted at steve.rivkin@ontrackprojects.com.*

RE-ports

Goals Day

24th March 2010, University of Westminster.

In the morning, Ian Alexander and Ljerka Beus-Dukic gave a tutorial on Goals.

Alexander kicked off by asking "What is a goal?", collecting suggestions from participants.

Is a football goal:

- The woodwork?
- The opening?
- A score?
- Or what?

A shout of "away goals can count double" came from the back of the audience, and without missing a beat Alexander commented that rather like in requirements the goal posts can and do move! So what is a goal? Ideas included personal aspiration and something desirable.

The audience were asked whether goal modelling was used in industry – the "hmm…" on the projected slide was matched with an "hmm…" from the participants. Not really, we should use it more was the general consensus. Alexander then presented his near legendary Onion model showing operational and non-operational roles within system levels and the wider environment.

Do interviewees state fully formed requirements? – No said the audience, they tend to go straight for the solution (unbeknown to them, this theory would be tested later). Alexander introduced his "Westminster Alarms Company" and the goals of householders, engineers, police and regulators. He pointed out that the personal goals of the burglar are beyond the scope of specifying the alarm system and that Westminster Alarm Company won't promise to meet the goal of being safe in the home – it doesn't want to be sued!

[5] IBM Rational® DOORS®, IBM Corporation Software Group, Route 100 Somers, NY 10589U.S.A.

After some interesting discussion the workshop moved onto goal modelling. Alexander introduced his simple goal modelling technique – just a blob and some arrows! – and also illustrated his DOORS-based tool support. A spacecraft example was presented, and he emphasised the key goals – don't model everything was the message here.

A role play exercise followed, with the participants forming groups of 2 to take the role of a requirements analyst and a stakeholder in the burglar alarm system to identify goals on sticky post-it notes. It was interesting to see that one interviewee tended toward the solution, despite this trap being flagged up a short while earlier! More than one goal on a note caused a problem, rather like AND in requirements statements. The example moved on to car thieves – a theme seemed to be emerging here!

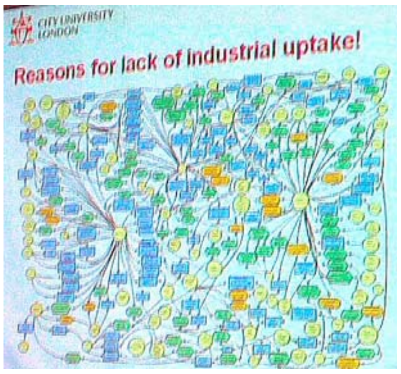
More goal modelling techniques were presented – Goal Structuring Notation (GSN), UML, i* and KAOS. Beus-Dukic pointed out that one of the biggest limitations of UML is the lack of support for quality (non-functional) goals. Alexander gave i* not so much a kick-in, but a friendly beating! He emphasised that its unique selling point is the identification of actor dependencies, but complexity of notation poses a barrier to its use. Beus-Dukic presented KAOS, a more formal approach with an even more academic feel to it. The use of obstacles in the notation was highlighted as a strength of the KAOS approach.

Generally there was a good exchange between the floor and the front and plenty of enthusiasm for the exercises. Ian also got in one or two plugs for his and Ljerka's book!

© James Lockerbie 2010

In the afternoon we had 5 talks from distinguished speakers in industry and academia.

James Lockerbie (City University) spoke about i* at City University. He made everybody laugh with his discussion of the choice of colours and symbols in i*. He did however make the point that i* can perfectly well be used with no more than 5 types of symbols.



James' Amazing Giant Model

He then showed a tremendously complex model with hundreds of boxes – but absolutely no softgoals (for NFRs). Clearly, when modelling spirals out of control or dives into "analysis paralysis", industry is not exactly attracted.

He showed an i* tool that he had implemented as two plug-ins (to create SD and SR diagrams) for Visio, called Redepend. This addresses a key weakness – lack of tool support – in the i* package, so is potentially an important step forward.

The tool (he was brave enough to demonstrate live, using a model of air traffic control) enables you to grey out irrelevant parts of a large model by automatically following the dependencies. That is a dramatic simplification, enabling i* work to scale up to realistic sizes without becoming unmanageable. Then you can filter the model by actor types and view attached functional and non-functional requirements.

i* models are vulnerable to "spaghetti" – everything seems to connect to everything else. The ability to grey out unconnected areas does address this: in the worked example, Lockerbie showed that you can quickly see if an actor who should be involved in a process is wrongly left out. The approach can also handle assumptions and satisfaction arguments.

Large i* models work well with teams that have grown up with them, but they remain quite forbidding to people new to a project.

In answer to questions from the floor, Lockerbie said that i* does not replace UML which remains useful for use cases and class modelling, but it is very helpful for defining and analysing softgoals which UML is weak on. People were keen on going further, and were pointed to the tutorials provided by the University of Toronto.

Emmanuel Letier (UCL) spoke about Requirements and Architecture during System Evolution. This is not limited to reasoning about goals in KAOS, but takes it further to link goals to design. Formality is justified by creating appropriate abstractions which permit better reasoning about development issues.

Emmanuel making a point

RE was not the "first stage of system development", he argued, but was "concerned with the real-world goals for, functions of, and constraints on (software) systems" [Pamela Zave]. Concomitantly, requirements and design grow together, influencing each other.

For instance, if we know (design) that bank customers are making payments by card, then the system needs to be able to block cards; and (looking at it the other way), card blocking mechanisms imply that we have decided to implement card payments.

So, there need to be connections of various kinds. Goal modelling needs to be linked to architectural (e.g. class) and behavioural (e.g. scenario, timing) models. Similarly, goals need to be linked to design decisions – for instance, reasoning about the relative level of security that would be provided by two alternative designs. This means working with partial models, a novelty in formal reasoning.

Many projects fail in a premature leap to software coding: can schedule be more important than quality? Similarly, requirements are generally incomplete: can we identify how large a risk is being taken? This leads to a use of KAOS for risk-based RE.

Issues in requirements and design need to be reflected back to stakeholders, for example through web-based participation and social networking. This is a reworking of an old idea, participatory design. There may still be formality in the system, but it must be presented very simply for people to understand it.

Phil Wilkinson (Rolls-Royce) spoke about the Goal Structuring Notation (GSN) and Satisfaction Arguments. These connect requirements on the one hand, and safety and suitability arguments on the other. Safety arguments can be made in languages such as GSN and CAE (Claims, Arguments, Evidence).

The Praxis consultancy proposed the REVEAL method to provide satisfaction arguments, including rich traceability, to demonstrate why a design actually meets its requirements.

GSN comes from the work of John McDermid and Tim Kelly at the University of York. It shows how goals are supported by arguments, which in turn are supported by evidence from things like tests, fault tree analysis, FMECA and process evidence.

GSN provides for recording assumptions explicitly – something that everybody knows they should do, but rarely get around to, said Wilkinson.

Standards provide for safety arguments on various bases – such as doing FMECA, but give little guidance on what kinds of modelling to do when.

Arguments can be deductive, inductive or abductive. But what about arguments by example, analogy, causes, and from authority? Can these be valid? They are certainly often used. And many arguments are simply fallacious: can these be detected? Finally, argument is not the same as explanation.

Tutorials on argumentation give simple deductive examples; but more elaborate arguments are necessary. Proofs of Concept can be helpful in showing acceptable ways of constructing arguments.

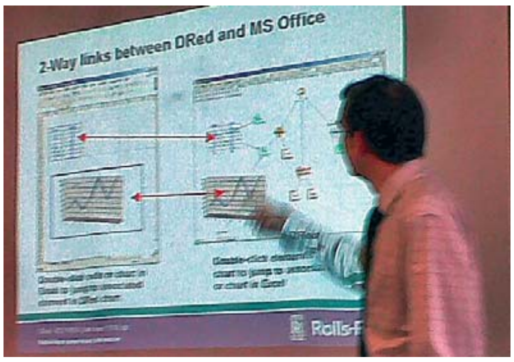
The REVEAL approach says simply that when the domain properties are combined with the specifications, the requirement (a system goal) is shown to follow.

Much of this works very well. The bits that don't fit are proof of concept; the disconnect between strategy and goal on the one hand, and requirement and satisfaction on the other; and the role of justifications and context, which are useful in support but are not exactly provided for in GSN (say).

Why bother with argumentation? It must be useful to understand why we think something is true. Arguments assist in co-operation between requirements people and safety engineers, argued Wilkinson. And safety engineers need to know that requirements are necessary and sufficient, as well as how critical they really are. Finally, writing or drawing a satisfaction argument seems to assist the thought process in constructing a safety argument.

Wilkinson felt it would be attractive to merge satisfaction (e.g. Reveal) and safety (e.g. GSN) argument notations as the structures are frankly very similar. But he wondered what everyone else's experience was: very probably, not as advanced as Rolls-Royce.

Gareth Armstrong (Rolls-Royce) spoke about DRed, the Decision Rationale Editor.



Gareth Armstrong and DRed

It is a software tool that captures rationale in graphical form – it was intended initially only for Design Rationale, but it clearly had wider applicability. It has saved much effort when preparing reports as the reasons for decisions are recorded far more accessibly than when in Word document form.

Somewhat like the free Compendium tool from the Open University, questions are recorded with a question-mark icon; ideas with a light-bulb. Icons are linked simply by arrows to show the flow of argument.

DRed has many features specifically designed to make rationale easy to describe and to understand. For instance, arguments are shown with boxes shadowed green if in favour, red if against.

Rejected ideas are marked with a large red X. If all your options are rejected, you can see at once that you need to find more options to explore.

As well as text boxes, one-way graphical hyperlinks can be made to existing documents and websites, and images such as photographs can be included where these convincingly show a step in an argument.

Two-way links can be created between DRed and tools such as Excel and PowerPoint to enable forward and backward navigation between rationale and other evidence. Argumentation elements can be created automatically when e.g. tasks are documented in Excel.

Argumentation can be spread across multiple diagrams by placing a box (more detailed arguments) on a diagram, giving access to a lower-level diagram when opened. The tool can in fact be used (with different semantics) for a nested set of function flow diagrams à la Yourdon.

DRed was developed from the basic IBIS (rationale modelling) concepts, adapted to the needs of industry.

The tool is in wide use within Rolls-Royce; more than 600 people have attended training courses, but since it is available company-wide and is easy to use, many more engineers have picked it up by watching their colleagues and trying it for themselves.

The audience were visibly impressed, and further inquiries can be directed to technology.licensing@rolls-royce.com

Pete Sawyer (Lancaster University) spoke on A Goal-Based Modelling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty. The work was done with Betty Cheng of Michigan State University.

Ljerka, Pete, Gareth

Projects fail for reasons of scale and complexity, and when problems are not fully understood, leading to unstable or incomplete requirements. Self-adaptation is a possible strategy: the idea is that the system adapts itself to changes in its environment, getting over the problem that we don't have a good track record of getting the requirements perfect before coding.

For instance (Sawyer illustrated with a photograph of the River Ribble) a riverbed sensor grid could operate in quiescent mode, doing its best to save on battery power while the river was low, or in flood mode, doing its best to be robust.

Requirements for Dynamically Adaptive Systems (DAS) could be derived starting with goals, then KAOS obstacle modelling to analyse threats, and the RELAX language to specify requirements with uncertainty. So, after the usual goal/obstacle cycle – identifying and mitigating obstacles/threats, the approach also identifies and mitigates uncertainty factors.

A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty
Betty H.C. Cheng, Michigan State University, US
Jon Whittle, Pete Sawyer, Nelly Bencomo, Lancaster University, UK
Pete Sawyer in full flow

Uncertainty factors were documented in KAOS simply by representing uncertainties as (possible) obstacles to system goals. These were applied both to "requirements" which are KAOS goals tied to system actors (which can achieve the goals), and to "expectations", which are KAOS goals tied to human actors (who you hope will do as you expect). So, humans can forget to do things, introducing unavoidable uncertainties into system operation.

The robustness of the requirements can be explored by "relaxing" one constraint after another, e.g. by asking if a requirement to achieve something within a certain time could in fact be met acceptably by allowing it to achieve the result eventually. If that is not acceptable, then a new goal, such as to issue an alarm, may be necessary: but that adds cost to the system design.

This seems to be a wholly new requirements process, permitting some degree of reasoning with uncertainty when discovering requirements. It corresponds to "reflection" (self-awareness) in knowledge-based systems, a concept that survives in work on formal methods such as KAOS. The method could clearly also be applied to softgoals (NFRs).

James, Phil, Emmanuel, Gareth, Pete, Ian

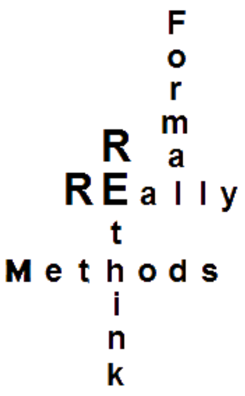
The meeting ended with a lively panel discussion, with questions from the floor on how "industry-ready" the various goal modelling techniques were. The answers were of course from immediately to far into the future, but it was clear that while people are concerned, there are many simple things that can be done now to improve industrial practice.

© Ian Alexander 2010

RE-verberations

Really Rethinking Formal Methods

In the January 2010 issue of *IEEE Computer*, David Lorge Parnas – one of the grandees of formal specification research – replied to the September issue of the journal, which was themed “Rethinking Formal Methods”.



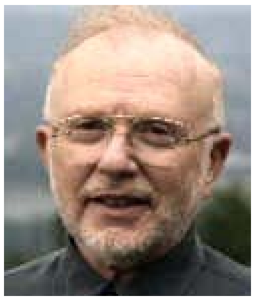
The problem was that the articles in the themed issue hadn’t rethought anything. They

“presented minor variations of ideas that their authors .. have advocated for years”.

Ouch.

When someone can write *“It has been at least 35 years since I first heard Jean-Raymond Abrial present the ideas that were the basis of Z”* it is probably best to sit up and take notice.

Obviously stung by Parnas’ challenge, the editors sensibly invited him to have another try at “Really Rethinking Formal Methods” on behalf of the community. In a 6-page article, Parnas came up with a lot of questions that 40 years of research have failed to answer.


David Lorge Parnas

Parnas is famous for saying uncomfortable truths such as

“Requirements in mathematical language are no use unless they are easier to read than the code.”

That is pretty uncomfortable stuff if you have just been working for three years on a new mathematico-logico-philosophical gadget involving the very latest in spatio-temporal reasoning that only takes six pages to conclusively prove that a ten-line computer program is in fact “correct”.

So, if you work in Formal Methods and are of a sensitive disposition, look away now.

Still reading? OK. Parnas dives in with *“Applications of formal methods in industrial practice remain such exceptions that they confirm that the use of formal methods is not common practice.”*

Ow.

We must question current assumptions so as to see what must be changed, Parnas continues, slipping on his surgical gown and gloves.

“Funding agencies often require that larger research-funded projects include some cooperation with industrial organizations [to] demonstrate the practicality of [their] approach on ‘real’ examples.”

Don’t you just love the inverted commas around ‘real’ there?

“When authors report such efforts, they state that they are successful. [But if they really were,] papers describing successful use would not be published.”

Youch.

You think the knife has really gone in now? Try this:

“Industry is so plagued by errors and high maintenance costs that it would use any method it thought would help; it chooses not to use methods such as Z or VDM.”

Yowza.

The problem on the academic side is that *“many research papers are written as if the mathematics were all that matters”*. The formal models and logical results fail to relate to *“the actual code on real machines”* and *“they offer no way to deal with the complexity of software systems.”*

Not too much wrong with current formal methods research, then.

What’s the answer? Parnas says that’s up to researchers to rethink, but he gives several hints that he has more time for engineers than for mathematicians.

Engineers use maths, but not to construct “proofs of correctness”.

Instead, engineers apply maths to evaluate and predict the qualities of SEVERAL alternative “correct” solutions.

How much would it cost to make the casing of steel? Of aluminium? Of carbon fibre-reinforced plastic? Would all of these be safe? Which would be most durable? What would each one weigh?

These questions relate to product qualities, and trade-offs are needed to choose the best solution.

Software, in a word, needs to become more like Engineering.

Parnas also takes a swipe at authors of articles on the use of maths in software development. There are two distinct messages: a general one

"that reminds us of the ubiquity of faults in software, and argues that the use of mathematical notation and reasoning can ameliorate the situation."

And a specific one *"that describes a detailed syntax and semantics for a language that can be used to describe a model, and rules that allow us to 'reason about' that model and thereby check certain properties of it."*

Note the inverted commas again. Parnas comments *"I always find the general message convincing."*

Damn'd with faint praise, perhaps.

What to rethink, then? Parnas comes up with no fewer than seventeen major problems in current formal methods.

For example, program variables are often sloppily equated with mathematical variables. But in a program, a variable is a finite state machine; its value is its state, and its identifier is just how we refer to it. In maths, variables like *x* don't have states, they're just placeholders.

Among the other problems are what to do about time: in real-time systems, something that runs too slowly is wrong.

And there is no particular reason why programs should terminate; do forever is a perfectly legitimate way to process a continual stream of events.

Same for non-determinism – why shouldn't there be many possible answers? It often happens. Parnas suggests that traditional determinism should perhaps be treated just as a special case.

Returning to specs being harder to read than the code, Parnas is quite clear:

"Unless the sequence of transformations is a requirement, programs should not be used as specifications."

At least here there is a definite alternative:

"view a specification as a predicate. With a predicate you cannot directly compute an answer, but you can easily check the correctness of a proposed answer."

Mind you, even here things aren't all sweetness and light. Parnas recalls a specification for a sort program. The precondition was *"only that there be some values in an array"*, and the postcondition *"that the array be sorted"*.

Sounds all right? Well, *"a program that assigned the value* j *to the* j*th element would satisfy this specification"*.

If you'd like an example of that, suppose the array holds 7, 4, and 8 at the start. The program would meet its specification if the array held 1, 2, and 3 at the end.

Oh dear. Perhaps separate pre- and post-conditions need to be abandoned.

RE-partee

Q. What is the difference between Mechanical Engineers and Civil Engineers?
A. Mechanical Engineers build weapons, Civil Engineers build targets!

Q. How many Requirements Engineers does it take to screw in a light bulb?
A. Just one. I hold on to the light bulb and the world revolves around me!

The graduate with a Science degree asks, "Why does it work?"
The graduate with an Engineering degree asks, "How does it work?"
The graduate with an Accounting degree asks, "How much will it cost?"
The graduate with an Arts degree asks, "Do you want fries with that?"

Q. : What is the difference between hardware and software?
A: Hardware gets faster, cheaper, smaller. Software gets slower, costlier and bigger!

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/

Tom Gilb's Downloads
http://www.gilb.com/downloads

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
BAESYSTEMS
simon.hutton @ baesystems.com

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@baesystems.com). Submissions must be in electronic form, preferably as plain ASCII text or rtf.

The deadline for RQ 55 (July 2010) is Friday 25th June 2010

Joining the RESG

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